FIRST Robotics Competition

WPILib

avr. 20, 2024

De Zéro à Robot

1 Introduction 3

2 Étape 1 : Construire votre robot 5

3 Étape 2 : Installation des composants logiciels 39

4 Étape 3 : Préparation de votre robot 73

5 Etape 4 : Programmation de votre robot 93

6 Aperçu des composants matériels 117

7 Aperçu des composants logiciels 147

8 Qu'est-ce que WPILib ? 157

9 2024 Overview 159

10 Survol de VS Code 171

11 Dashboards 197

12 Télémétrie 335

13 Programmation en LabVIEW FRC 349

14 FRC Python Programming 387

15 API du Matériel 393

16 Dispositifs avec bus CAN 489

17 Programmation de base 505

18 Ressources d'assistance 581

19 Glossaire FRC 583

20 Driver Station 591

21 RobotBuilder 619
22 Simulation du Robot 693
23 OutlineViewer 731
24 roboRIO Team Number Setter 733
25 Traitement de la vision 735
26 Programmation orientée commande 801
27 Cinématique et Odométrie 905
28 NetworkTables 935
29 Planification de trajectoire 995
30 roboRIO 1035
31 GradleRIO avancé 1049
32 Commandes Avancées 1071
33 Fonctionnalités Pratiques 1231
34 Exemples fournis de Projets WPILib 1239
35 Exemples de projets tiers 1245
36 Composants matériels - Notions de base 1247
37 Didacticiels des composants matériels 1303
38 Capteurs 1305
39 Premiers pas avec Romi 1345
40 Getting Started with XRP 1369
41 Introduction à la réseautique 1381
42 Utilitaires réseau 1415
43 Contribuer au projet frc-docs 1419
44 Développer avec allwpilib 1439
Index 1441

Bienvenue dans la Documentation du système de contrôle de la *FIRST*® compétition robotique ! Ce site contient tout ce qu'il faut savoir pour programmer un robot de compétition !

Les traductions faites par la communauté peuvent être trouvées dans une variété de langues dans le menu en bas à gauche.

Returning Teams

If you are a returning team, please check out the overview of changes from 2023 to 2024, known issues, and quick start guide for updating.

Changelog

Known Issues

Quick Start

New Teams

Le tutoriel Zero-à-Robot vous guidera à travers la préparation, le câblage et la programmation d'un robot de base !

Go to Zero-to-Robot

Hardware Overview

Survol des composants matériels disponibles pour les équipes.

Go to Hardware Overview

Software Overview

Un survol des composants logiciels et des outils mis à la disposition des équipes.

Go to Software Overview

Programming Basics

Documentation utile tout au long du processus de programmation pour une équipe.

View articles

Advanced Programming

Documentation adaptée aux équipes vétéranes. Cela inclut du contenu tel que la planification des trajectoires et la cinématique.

View articles

Hardware

Didacticiels des outils matériels et contenu disponibles pour les équipes.

View articles

Romi and XRP Robots

The Romi and XRP robots are low-cost platforms for practicing WPILib programming.

View Romi articles

View XRP articles

API Documentation

Java, C++, and Python class documentation.

Java

C++

Python

Software Tools

Des outils essentiels tels que l'application FRC Driver Station, Dashboards, roboRIO Imaging Tool et plus encore.

View articles

Example Projects

Cette section présente les exemples de projets disponibles que les équipes peuvent utiliser sous VS Code.

View articles

Status Light Quick Reference

Guide de référence rapide pour les voyants lumineux indiquant l'état de fonctionnement pour une variété de matériel FRC.

View article

3rd Party libraries

Tutoriel relatif à l'ajout des librairies tierces à votre projet de robot telles que CTRE et REV.

View article

1 Introduction

Bienvenue à la documentation officielle du système de contrôle et les progiciels WPILib pour la Compétition de Robotique *FIRST*®. Cette page est la ressource primaire documentant l'utilisation du système de contrôle FRC (comprenant câblage, configuration et logiciels) ainsi que les bibliothèques et outils WPILib.

1.1 Nouveau en programmation ?

Ces pages couvrent les détails de la bibliothèque WPILib et le système de contrôle FRC et ne décrivent pas les bases de l'utilisation des langages de programmation supportés. Si vous voulez des ressources pour apprendre les langages de programmation, veuillez consulter les ressources ci-dessous :

Note : Vous pouvez continuer avec cette section « De Zéro à Robot » pour obtenir un robot de base fonctionnel sans connaissance préalable d'un langage de programmation. Cependant, pour aller au-delà de ce robot de base, vous devrez connaître un langage de programmation dans lequel vous choisissez de programmer votre robot.

1.1.1 Java

- Code Academy
- Head First Java 2nd Edition est une introduction pour programmer en Java axée pour les débutants (ISBN-10 : 0596009208).

1.1.2 C++

- LearnCPP
- Programming : Principles and Practice Using C++ 2nd Edition est une introduction à C++ par le créateur du langage lui-même (ISBN-10 : 0321992784).
- C++ Primer Plus 6th Edition (ISBN-10 : 0321776402).

1.1.3 LabVIEW

- NI apprendre LabVIEW

1.1.4 Python

- List of various guides to learn Python

1.2 De Zéro à Robot

The remaining pages in this tutorial are designed to be completed in order to go from zero to a working basic robot. The documents will walk you through wiring your robot, installation of all needed software, configuration of hardware, and loading a basic example program that should allow your robot to operate. When you complete a page, simply click **Next** to navigate to the next page and continue with the process. When you're done, you can click **Next** to continue to an overview of WPILib in C++/Java/Python or jump back to the home page using the logo at the top left to explore the rest of the content.

2

Étape 1 : Construire votre robot

Un aperçu du matériel du système de contrôle disponible peut être trouvé *ici*.

2.1 Introduction au câblage du robot FRC

Note : Ce document détaille le câblage d'un panneau d'électronique basique pour le kitbot ou afin de permettre un test basique de la base de pilotage.

Certaines images dans cette section démontrent l'installation pour un système de contrôle du robot utilisant les contrôleurs de moteur SPARK MAX. Les diagrammes de câblage et la disposition devraient être similaire pour les autres contrôleurs de moteur.. Lorsque approprié, deux ensembles d'images sont fournis afin de démontrer les connections en utilisant de contrôleurs avec et sans fils intégrés.

2.1.1 Aperçu

REV

CTR

2.1.2 Rassembler des Matériaux

Localisez les outils et composants de système de contrôle suivants

- Matériaux du Kit :
  - Hub de distribution d'énergie (*PDH*) / Panneau de distribution d'énergie (*PDP*)
  - roboRIO
  - Hub pneumatique (*PH*) / Module de commande pneumatique (*PCM*)
  - Module d'alimentation radio (*RPM*) / Module régulateur de tension (*VRM*)
  - Radio OpenMesh (avec câble d'alimentation et câble Ethernet)
  - Signal lumineux du robot (*RSL*)

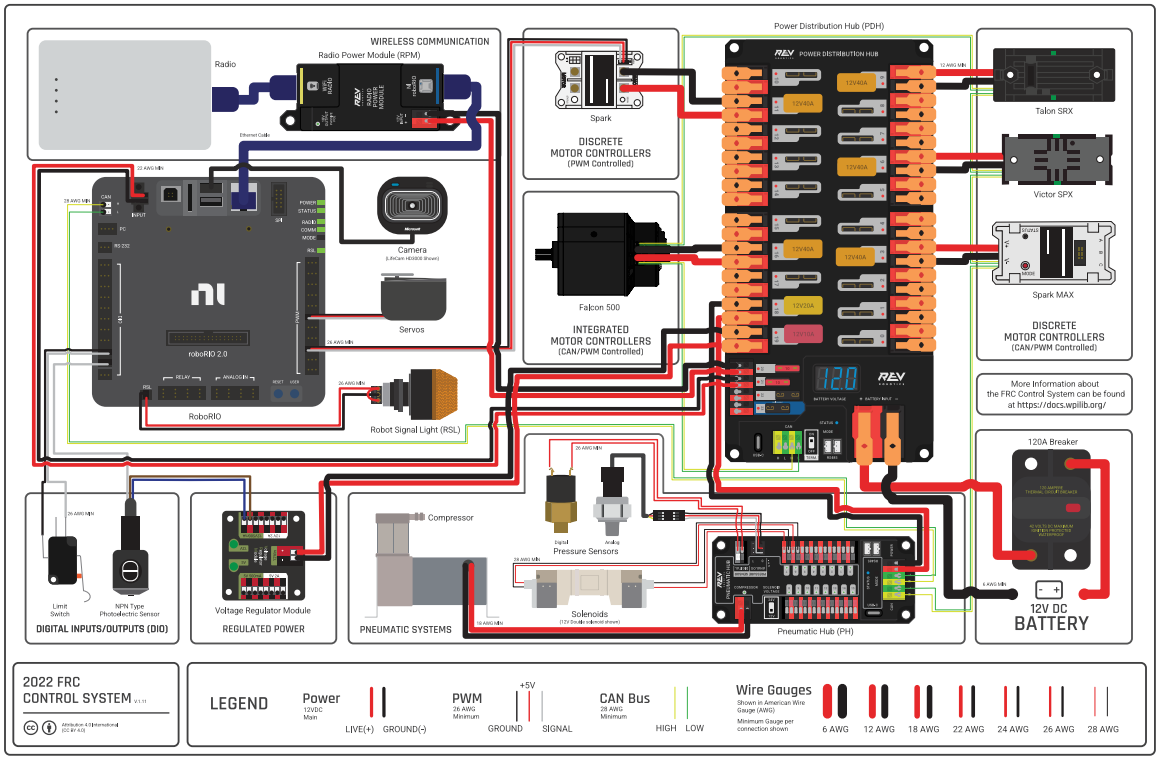


Fig. 1 – Diagramme gracieuseté de l'équipe FRC® 3161 et Stefen Acepcion.

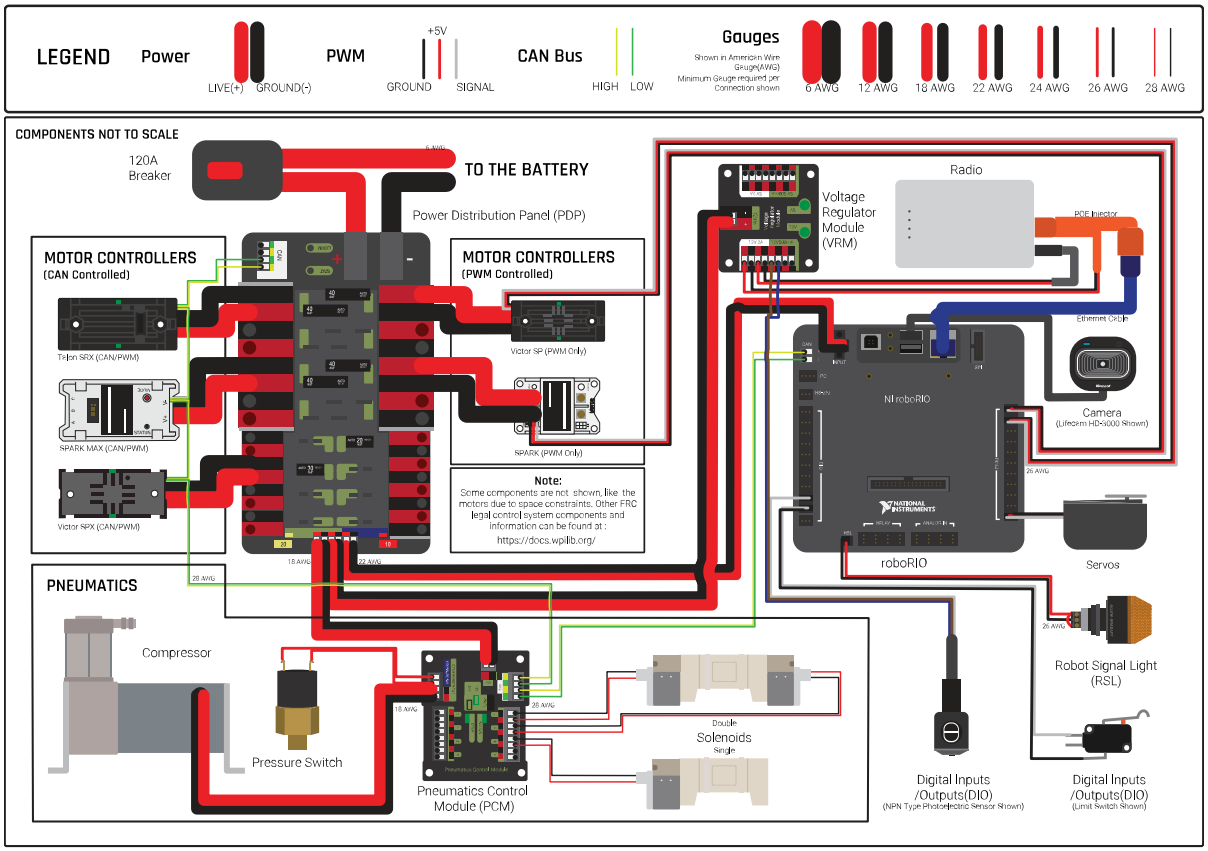


Fig. 2 – Diagramme gracieuseté de l'équipe FRC® 3161 et Stefen Acepcion.

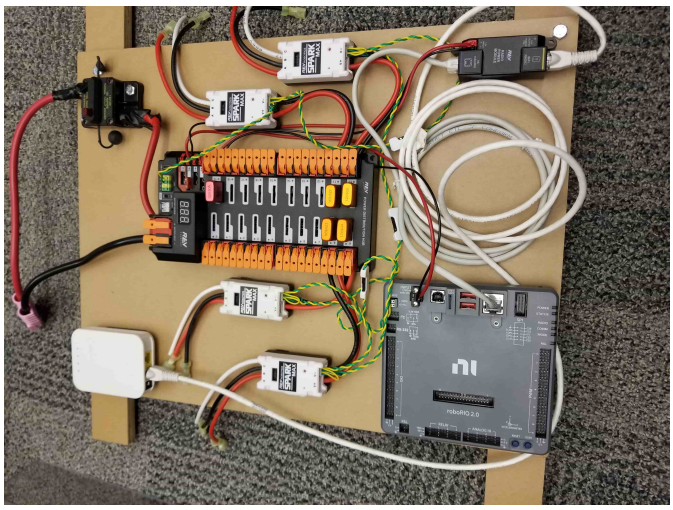
  - 4x SPARK MAX ou d'autres contrôleurs de moteur
  - 2x *PWM* cables en Y
  - Disjoncteur 120A
  - 4x Disjoncteur 40A
  - Fil rouge 6 AWG (16 mm^2)
  - Fil rouge/noir 10 AWG (6 mm^2)
  - Fil rouge/noir 18 AWG (1 mm^2)
  - 22 AWG (0.5 mm^2) filage jaune et vert *CAN*
  - 8x paires de terminaux à déconnexion rapide 10-12 AWG (4 - 6 mm^2) (Jaune) (16x terminaux en anneau si un contrôleur avec fils intégrés est utilisé).
  - 2x Connecteurs de batterie Anderson SB50
  - Cosses pour bornes de raccordement 6 AWG (16 mm^2)
  - Batterie 12V
  - Ruban électrique Rouge/Noir
  - Ruban Velcro avec double bande adhésive, ou vis
  - Attaches de style Tie-wrap ou Zip Ties
  - Contreplaqué 1/4″ ou 1/2″ (6-12 mm)
- Outils Nécessaires :
  - Outil spécialisé pour le branchement aux connecteurs Wago ou encore petit tournevis à tête plate.
  - Minuscule tournevis à tête plate (style réparation de lunettes)
  - Pinces coupantes, Pince à dénuder et pinces à sertir
  - Clé à 6 pans ou tourne-écrou de 7/16″ (11 mm peuvent fonctionner si le système impérial n'est pas disponible)
  - Clé/tournevis à boulon 7/16 » ou tournevis à tête Philips additionnel.
  - Pour le PDP CTR seulement : clé Hex 5mm (une clé 3/16 » peut fonctionner si la clé métrique n'est pas disponible).
  - Pour le PDP CTR seulement : clé Hex 1/16 »

2.1.3 Créer le panneau de support pour le Système de Contrôle

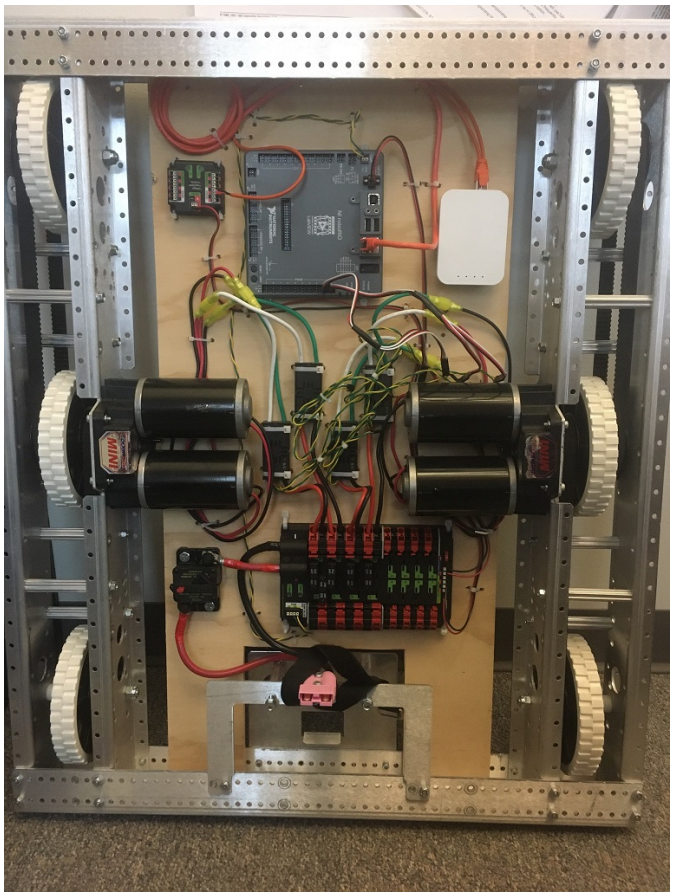
Pour le panneau-prototype, coupez un morceau de matériau de 1/4″ ou 1/2″ (6-12 mm) (bois ou plastique) d'environ 24″ x 16″ (60 x 40 cm). Pour un panneau de contrôle «Robot Quick Build» voir la documentation de support pour la carte de taille appropriée pour la configuration de châssis choisie.

2.1.4 Disposer les Composants du Système de Contrôle de Base

REV

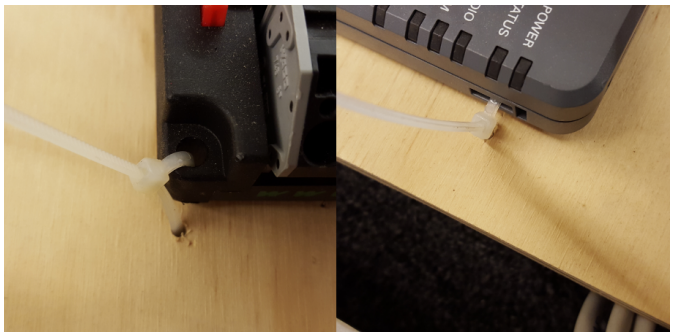


CTR



Disposez les composants sur le panneau. Un exemple de disposition est illustré dans l'image ci-dessus.

2.1.5 Fixer les composants



En utilisant le ruban Velcro ou la quincaillerie, attachez tout les composants au panneau. Notez que lors des tournois FRC, les contacts entre les robots peuvent être intenses, et le Dual Lock seul peut quelquefois se détacher. L'usage de vis et écrous (tel que démontré dans l'image ci-dessus) est une façon supérieure de fixer les composants électriques de grande taille. Ou encore, l'emploi de zip-ties peut être adéquat.

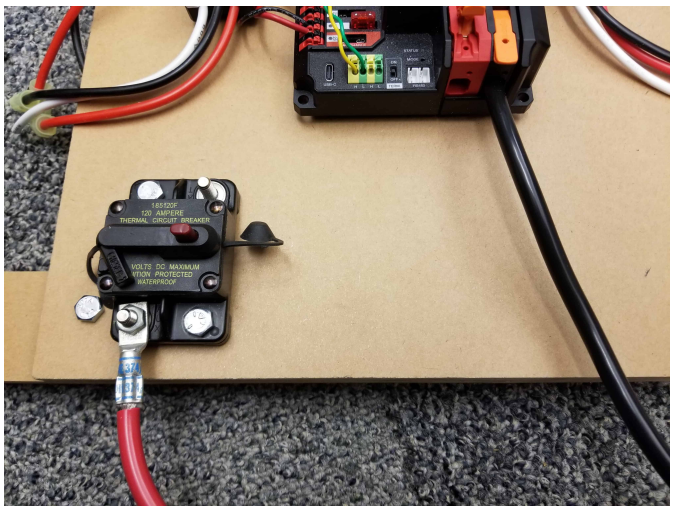
2.1.6 Attachez le connecteur latéral de la batterie du Robot.

REV

La prochaine étape impliquera l'utilisation de connecteurs Wago sur le PDH. Pour utiliser les connecteurs Wago, ouvrez le levier, insérez le fil et fermez le levier. Deux tailles de connecteurs Wago se trouvent sur le PDH.

- Connecteur principal d'alimentation : accepte de 4 à 18 AWG (.75 - 25 mm^2), dénudez 20 mm (~3/4 »)
- Connecteur des canaux haute puissance : accepte de 8 à 24 AWG (.25 - 10 mm^2), dénudez 12 mm (~1/2 »)

Pour maximiser la résistance mécanique à l'arrachement et minimiser la résistance électrique, les fils ne doivent pas être étamés (et idéalement non- torsadés) avant d'être inséré dans le connecteur Wago.



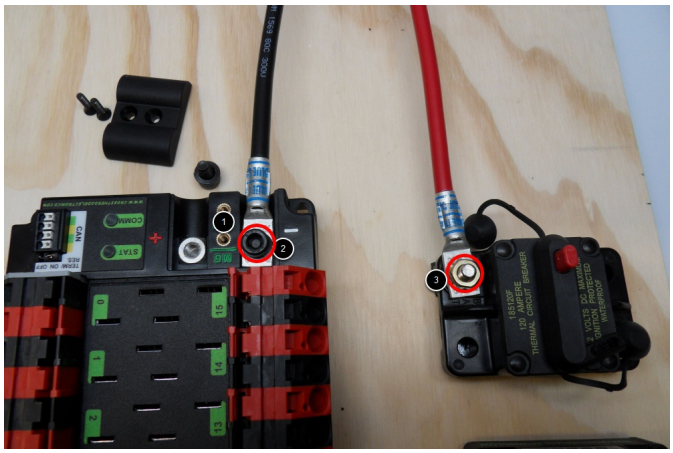
Nécessite : connecteur de batterie, cosses de borne 6 AWG (16 mm^2), clé 7/16 » (11 mm)

Fixez la cosse au fil positif (rouge) du connecteur de la batterie. Dénudez 0,75 » du fil noir.

Levez le levier au-dessus de l'entrée principale d'alimentation sur le PDH jusqu'il clique en place. Insérez le fil. Abaissez le levier pour sécuriser le fil.

À l'aide d'une clé Hex de 7/16″ (11 mm), retirez l'écrou du côté « Batt » du disjoncteur principal et fixez la borne positive du connecteur de batterie.

CTR



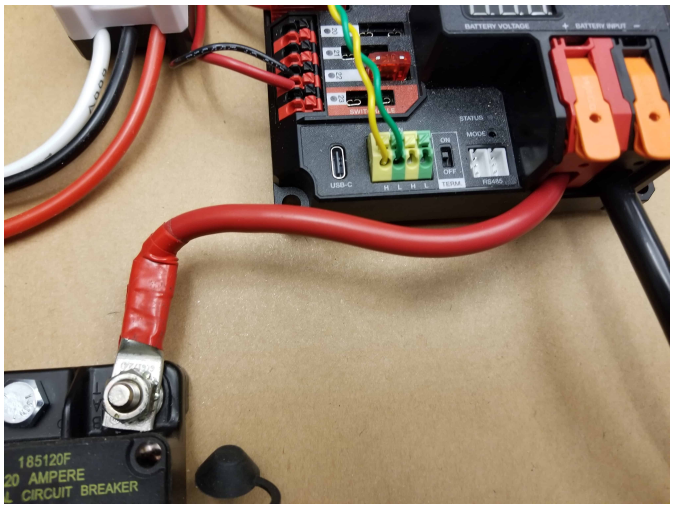
Requiert : Connecteur de batterie, Cosses de terminaison 6 AWG (16 mm^2), Clé Allen 1/16″, Clé Allen 3/16"" (5 mm) et Clé à 6 pans 7/16″ (11 mm)

Attachez les cosses à bornes au connecteur de batterie :

1. En utilisant un clé Allen 1/16″, retirez les deux vis fixant le couvercle des bornes du PDP.
2. À l'aide d'une clé Allen de 5 mm (3/16″), retirez le boulon négatif (-) ainsi que la rondelle du PDP et fixez la borne négative du connecteur de batterie.
3. À l'aide d'une clé Hex de 7/16″ (11 mm), retirez l'écrou du côté « Batt » du disjoncteur principal et fixez la borne positive du connecteur de batterie.

2.1.7 Connectez le disjoncteur au panneau de distribution.

REV

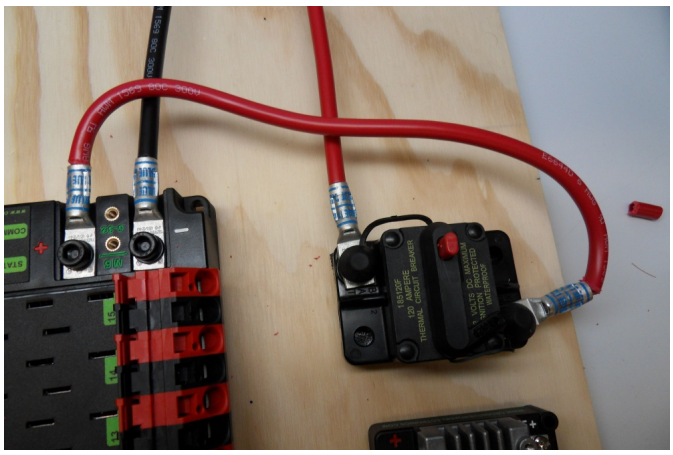


Nécessite : fil rouge 6 AWG (16 mm^2), 1x cosses de borne 6 AWG (16 mm^2), clé de 7/16 » (11 mm)

Fixez une cosse à l'extrémité du fil rouge 6 AWG (16 mm^2). À l'aide de la clé de 7/16 » (11 mm), retirez l'écrou du côté « AUX » du disjoncteur principal de 120 A et placez la borne sur le goujon. Fixez l'écrou sans serrer (vous souhaiterez peut-être le retirer sous peu pour couper et dénuder l'autre extrémité du fil). Mesurez la longueur de fil nécessaire pour atteindre la borne positive du PDH.

1. Coupez et dénudez l'autre extrémité du fil rouge.
2. À l'aide de la clé de 7/16 » (11 mm), fixez le fil du côté « AUX » du disjoncteur principal de 120 A.
3. Soulevez le levier de la borne d'entrée positive (rouge) du PDH, insérez le fil, puis fermez la borne.

CTR



Nécessite : fil rouge 6 AWG (16 mm^2), 2x cosses de borne 6 AWG (16 mm^2), Clé Allen 5mm, Clé à 6 pans 7/16″ (11 mm)

Fixez une cosse de borne à l'extrémité du fil rouge 6 AWG (16 mm^2). En utilisant la clé à 6 pans de 7/16″ (11 mm), retirez l'écrou du côté « AUX » du disjoncteur principal de 120 A et placez la borne sur le goujon. Fixez légèrement l'écrou (vous pouvez le retirer momentanément pour le couper, le dénuder et sertir l'autre extrémité du fil). Mesurez la longueur de fil nécessaire pour atteindre la borne positive du PDP.

1. Coupez, dénudez et sertissez la borne à la 2ème extrémité du fil rouge 6 AWG (16 mm^2).
2. À l'aide de la clé à 6 pans 7/16″ (11 mm), fixez le câble au côté « AUX » du disjoncteur principal de 120A.
3. À l'aide de la clé Allen de 5 mm, fixez l'autre extrémité à la borne positive du PDP.

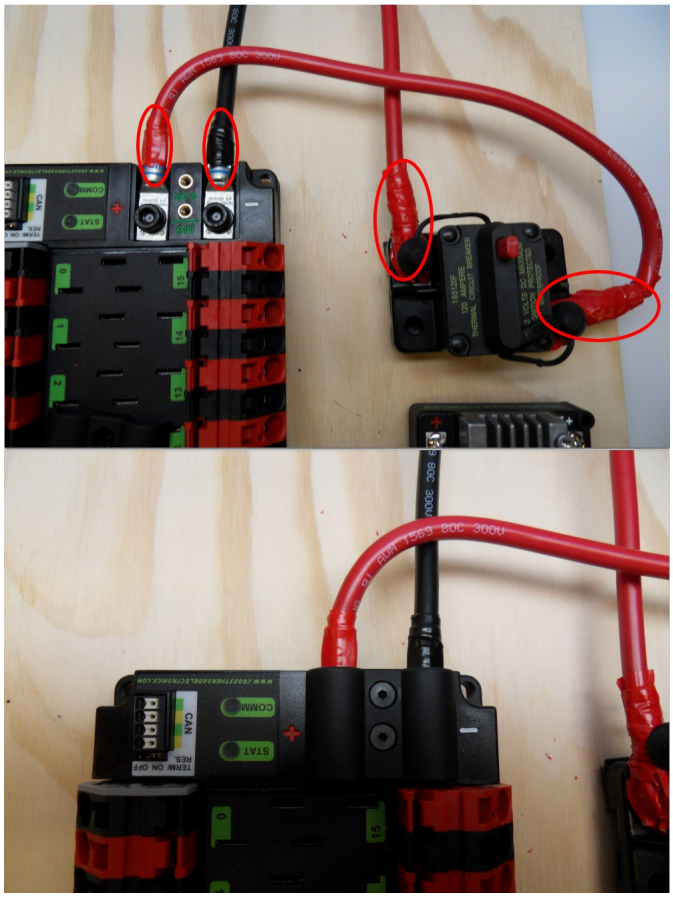
2.1.8 Isolation des connexions au PDP

REV

Requiert : ruban électrique

En utilisant un ruban électrique, isolez les deux connexions du disjoncteur 120A.

CTR

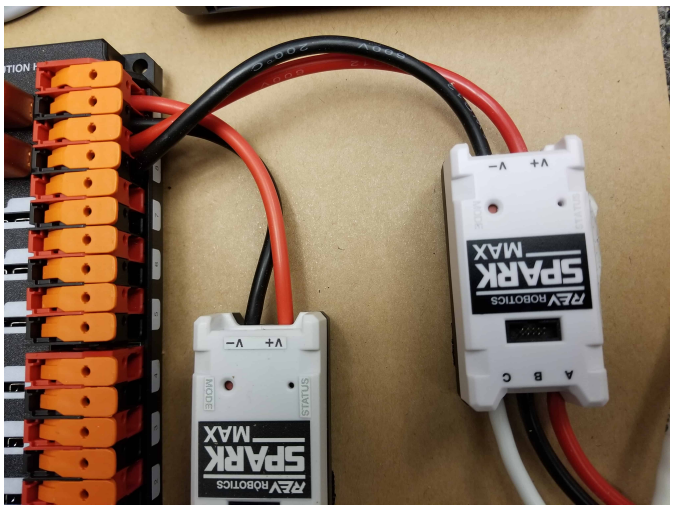


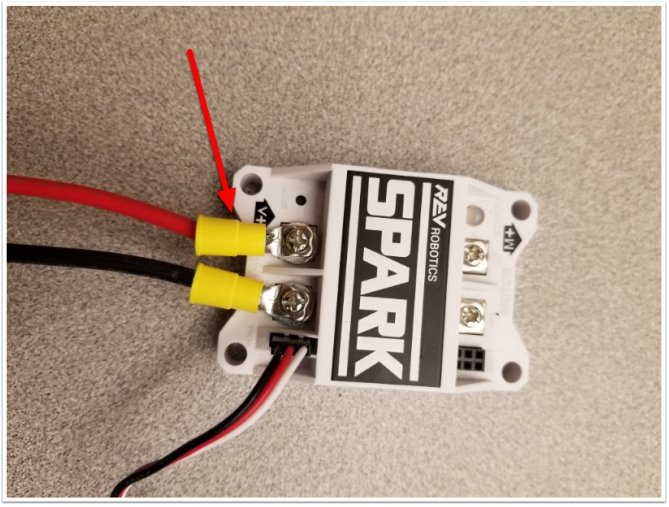
Requiert : Allen 1/16″, Ruban électrique

1. À l'aide de ruban isolant, isolez les deux connexions au disjoncteur 120A. Isolez également toute partie des bornes PDP qui sera exposée lorsque le couvercle sera remplacé.
2. En utilisant la clé Allen 1/16″, replacer le cache-borne PDP

2.1.9 L'alimentation du Contrôleur de moteur

REV





Nécessite : Contrôleurs de bornes à dénuder uniquement : fil 10 ou 12 AWG (4 - 6 mm^2), bornes à fourche/anneau 10 ou 12 AWG (4-6 mm^2), sertisseuse

Pour SPARK MAX ou d'autres contrôleurs de moteur avec fil intégré (image du haut) :

— Coupez et dénudez les fils d'entrée d'alimentation rouge et noir, puis insérez-les dans l'une des paires de bornes Wago.

Pour les contrôleurs de moteur avec bornes (image du bas) :

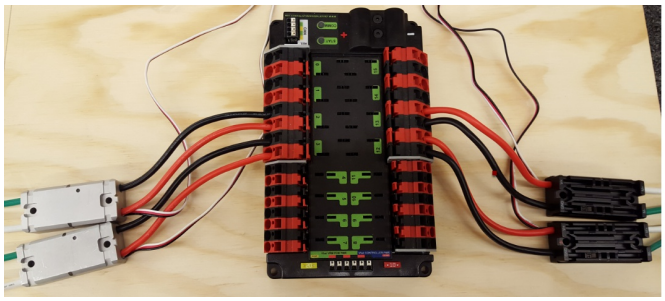
1. Coupez les fils rouge et noir à la longueur appropriée pour atteindre l'une des paires de bornes Wago jusqu'au côté entrée du contrôleur de moteur (avec un peu plus pour la longueur qui sera insérée dans les bornes à chaque extrémité).
2. Dénudez une extrémité de chaque fil, ensuite insérez dans les bornes Wago.
3. Dénudez l'autre extrémité de chaque fil, et sertissez une cosse en anneau ou à fourchette.
4. Reliez le terminal aux terminaux d'entrée du contrôleur de moteur (rouge à +, noir à -)

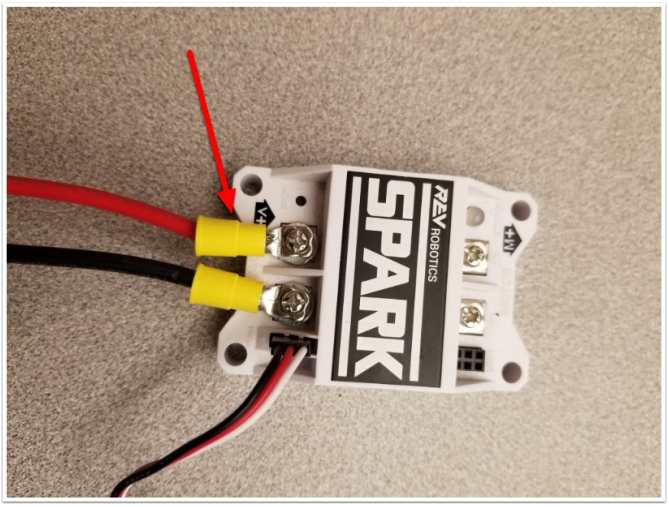
CTR

La prochaîne étape consiste à relier les fils aux connecteurs Wago sur le PDP. Pour utiliser les connecteurs Wago, insérez un petit tournevis à tête plate dans le trou rectangulaire à un angle peu prononcé (presque à l'horizontale) puis incliner le tournevis vers le haut pendant que vous continuez à appuyer pour actionner le levier, ce qui ouvre la borne. Le PDP a deux tailles de connecteurs Wago :

— Petit connecteur Wago : accepte 10 - 24 AWG (0,25 - 6 mm^2), bande 11-12 mm (~ 7/16″)
— Grand connecteur Wago : accepte 6 - 12 AWG (4 - 16 mm^2), bande 12-13 mm (~ 1/2″)

Pour maximiser la résistance mécanique à l'arrachement et minimiser la résistance électrique, les fils ne doivent pas être étamés (et idéalement non- torsadés) avant d'être inséré dans le connecteur Wago.





Nécessite : une pince à dénuder, un petit tournevis plat, des contrôleurs de bornes uniquement : fil de 10 ou 12 AWG (4 - 6 mm^2), fourchette de 10 ou 12 AWG (4-6 mm^2) / cosses à anneau, pince à sertir

Pour SPARK MAX ou d'autres contrôleurs de moteur avec fil intégré (image du haut) :

— Coupez et dénudez les fils d'entrée d'alimentation rouge et noir, puis insérez-les dans l'une des paires terminales Wago 40A (plus grands).

Pour les contrôleurs de moteur avec bornes (image du bas) :

1. Couper le fil rouge et noir à la longueur appropriée pour partir de l'une des paires terminales Wago 40A (plus grandes) et atteindre le côté correspondant à l'entrée du contrôleur de moteur (avec un petit bout de longueur additionnelle pour la longueur qui sera insérée dans les terminaux à chaque extrémité)
2. Dénudez une extrémité de chaque fil, ensuite insérez dans les bornes Wago.
3. Dénudez l'autre extrémité de chaque fil, et sertissez une cosse en anneau ou à fourchette.

4. Reliez le terminal aux terminaux d'entrée du contrôleur de moteur (rouge à +, noir à -)

2.1.10 Les connecteurs Weidmuller

Certains connecteurs pour le bus CAN et l'alimentation basse puissance utilisent une série de connecteur de type Weidmuller LSF. Voici quelques conseils à retenir lorsque vous utilisez ce type de connecteur pour obtenir les meilleurs résultats :

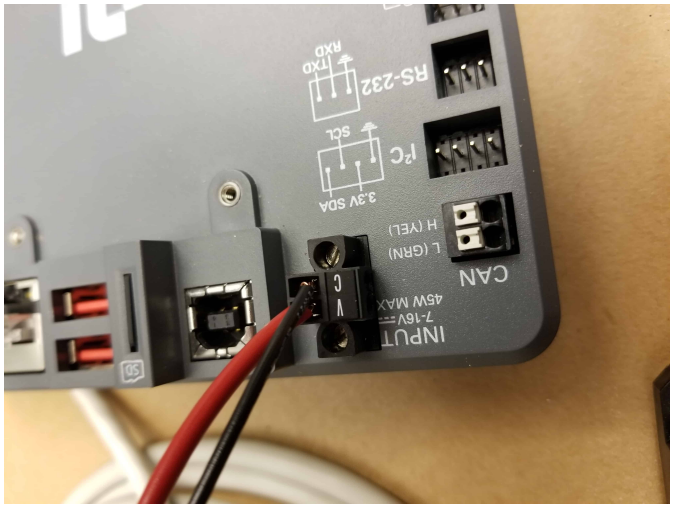
— Le fil doit être de 16 AWG (1,5 mm^2) à 24 AWG (0,25 mm^2) (consultez les règles pour vérifier le calibre requis pour le câblage d'alimentation)
— Les extrémités des fils doivent être dénudées d'environ 5/16″ (~ 8 mm)
— Pour insérer ou retirer le fil, appuyez sur le petit picot, ou « bouton » correspondant pour ouvrir la borne.

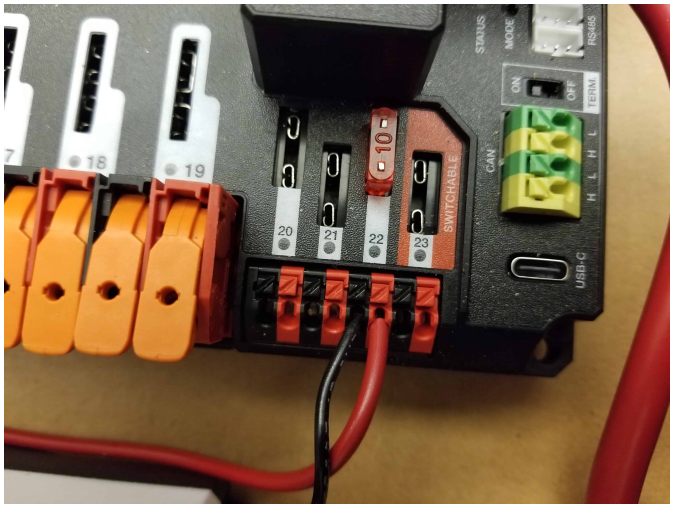
Après avoir fait la connexion, vérifiez qu'elle est propre et sécurisé :

— Vérifiez qu'il n'y a aucun brin de fil qui dépasse à l'extérieur du connecteur, ce qui pourrait causer un court-circuit
— Tirez sur le fil pour vérifier qu'il est bien en place. Si le fil sort et que son calibre est correct, il doit être inséré davantage et/ou dénudé davantage. Parfois, la borne peut rester ouverte avec le fil inséré et le bouton relâché même si le fil est dénudé et inséré correctement ; dans ces cas, remuer légèrement le fil vers l'intérieur et l'extérieur permettra souvent au connecteur de se verrouiller et de saisir le fil.

2.1.11 L'alimentation du roboRIO

REV

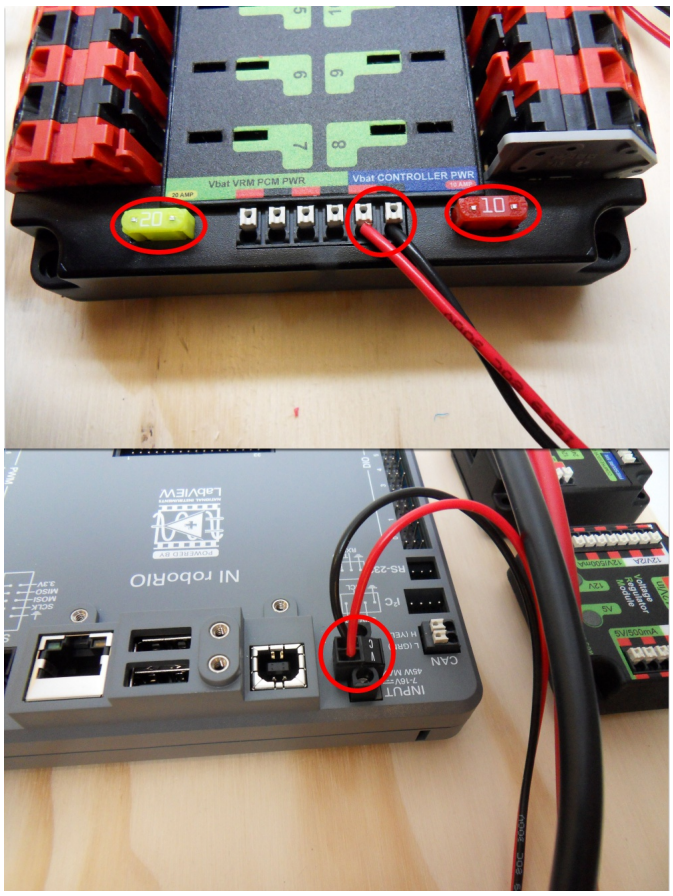




Nécessite : un mini fusible 10 A, une pince à dénuder, un très petit tournevis plat, 18 AWG (1 mm^2) rouge et noir

1. Insérez le fusible 10A dans le PDH dans l'un des canaux à fusibles non commutables (20-22).
2. Dénudez ~5/16 » (~8 mm) sur les fils rouge et noir 18 AWG (1 mm^2) et connectez-les aux bornes correspondantes sur le canal PDH où le fusible a été installé.
3. Mesurez la longueur nécessaire pour atteindre l'entrée d'alimentation sur le roboRIO. Prenez soin de laisser une longueur suffisante pour acheminer les fils autour de tout autre composant tel que la batterie et pour allouer assez de longueur pour gérer convenablement la disposition des câbles.
4. Coupez et dénudez le fil.
5. En utilisant le minuscule tournevis à tête plate, connectez les fils aux bornes d'alimentation du roboRIO (rouge à V, noir à C). Assurez aussi que le connecteur d'alimentation est vissé solidement au roboRIO.

CTR

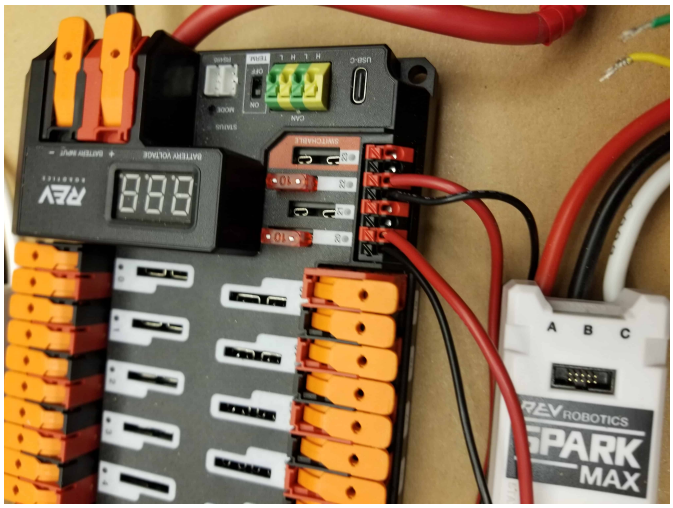


Nécessite : mini-fusibles 10A/20A, pince à dénuder, très petit tournevis plat, 18 AWG (1 mm^2) rouge et noir

1. Insérez les petits fusibles 10A et 20A dans le PDP aux locations illustrées sur la sérigraphie du PDP (et selon l'image ci-dessus)
2. Dénudez ~ 5/16″ (~ 8 mm) sur le fil rouge et noir 18 AWG (1 mm^2) et connectez aux bornes « Vbat Controller PWR » sur le PDB
3. Mesurez la longueur nécessaire pour atteindre l'entrée d'alimentation sur le roboRIO. Prenez soin de laisser une longueur suffisante pour acheminer les fils autour de tout autre composant tel que la batterie et pour allouer assez de longueur pour gérer convenablement la disposition des câbles.
4. Coupez et dénudez le fil.
5. En utilisant le minuscule tournevis à tête plate, connectez les fils aux bornes d'alimentation du roboRIO (rouge à V, noir à C). Assurez aussi que le connecteur d'alimentation est vissé solidement au roboRIO.

2.1.12 Alimentation de la radio

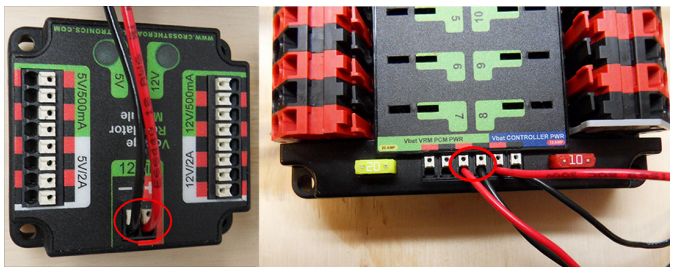
REV



Nécessite : une pince à dénuder, un petit tournevis plat (optionnel), un fil rouge et noir 18 AWG (1 mm^2) :

1. Insérez le fusible 10A dans le PDH dans l'un des canaux à fusibles non commutables (20-22).
2. Dénudez ~5/16 » (~8 mm) à l'extrémité du fil rouge et noir 18 AWG (1 mm^2) et connectez le fil aux bornes correspondantes du PDH.
3. Mesurez la longueur requise pour atteindre les bornes « Entrée 12 V » sur le module d'alimentation radio. Veillez à laisser suffisamment de longueur pour acheminer les fils autour de tout autre composant tel que la batterie et pour permettre tout soulagement de traction ou gestion des câbles.
4. Coupez et dénudez ~ 5/16″ (~ 8 mm) de l'extrémité du fil.
5. Connectez le fil aux terminaux d'entrée 12V du RPM.

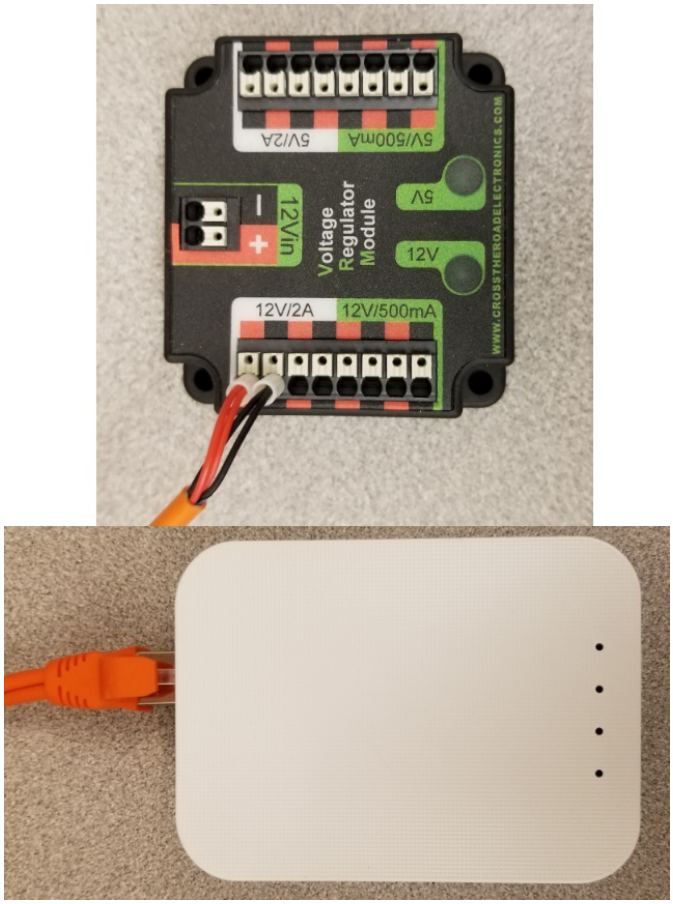
CTR



Nécessite : une pince à dénuder, un petit tournevis plat (optionnel), un fil rouge et noir 18 AWG (1 mm^2) :

1. Dénudez ~ 5/16″ (~ 8 mm) à l'extrémité du fil rouge et noir 18 AWG (1 mm^2).
2. Connectez les fils à l'une des deux paires de bornes étiquetée « Vbat VRM PCM PWR » sur le PDP.
3. Mesurez la longueur requise pour atteindre les terminaux « 12Vin » sur le VRM. Prenez soin de laisser une longueur suffisante pour acheminer les fils autours de tout autre composant tel que la batterie et allouez assez de longueur pour gérer convenablement la disposition des câbles.
4. Coupez et dénudez ~ 5/16″ (~ 8 mm) de l'extrémité du fil.
5. Connectez le fil aux terminaux 12Vin du VRM.

Avertissement : NE connectez PAS l'injecteur POE REV passif directement au roboRIO. Le roboRIO DOIT être connecté à la radio en utilisant un câble Ethernet tel que démontré dans la prochaine étape.

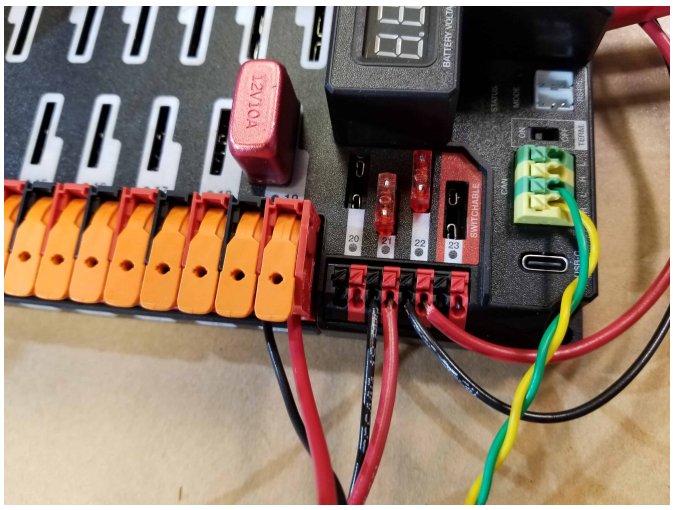


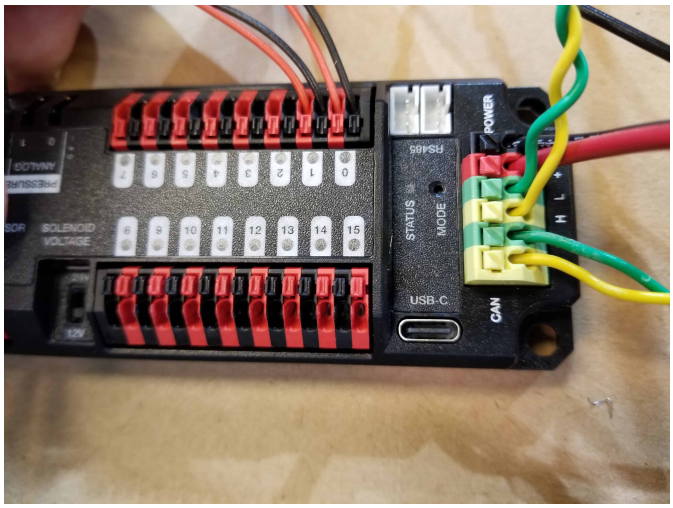
Nécessite : Petit Tournevis à tête plate (Optionnel), Câble POE de Radio REV

1. Insérez les fils du câble d'alimentation pour la radio dans le connecteur 12V/2A du VRM.
2. Connectez l'extrémité de la fiche RJ45 (Ethernet) du câble au port Ethernet de la radio le plus proche du connecteur cylindrique (étiqueté 18-24 V POE).

2.1.13 Alimentation des composantes pneumatiques (optionnel)

REV





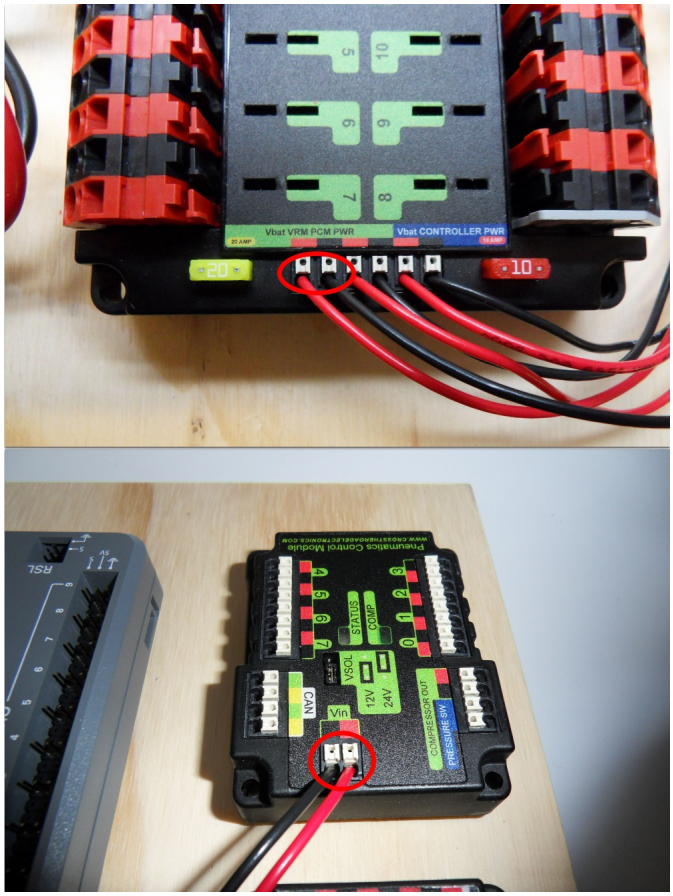
Nécessite : une pince à dénuder, un petit tournevis plat (en option), un fil rouge et noir 18 AWG (1 mm^2)

Note : Le hub pneumatique est un composant optionnel utilisé pour contrôler le système pneumatique du robot.

Le hub pneumatique peut être câblé soit à un port à fusible non commutable sur le PDH avec un fusible de 15 A ou moins, soit à un port protégé par disjoncteur avec un disjoncteur jusqu'à 20 A.

1. Dénudez ~ 5/16″ (~ 8 mm) à l'extrémité du fil rouge et noir 18 AWG (1 mm^2).
2. Connectez le fil au PDH de l'une des deux manières décrites ci-dessus
3. Mesurez la longueur nécessaire pour atteindre les bornes rouges sur l'extrémité courte du PH étiquetée +/-. Veillez à laisser suffisamment de longueur pour acheminer les fils autour de tout autre composant tel que la batterie et pour permettre tout soulagement de traction ou gestion des câbles.
4. Coupez et dénudez ~5/16 » (~8 mm) de l'autre extrémité du fil.
5. Connectez le fil aux bornes d'entrée PH.

CTR



Nécessite : une pince à dénuder, un petit tournevis plat (en option), un fil rouge et noir 18 AWG (1 mm^2)

Note : Le PCM est un composant optionnel utilisé pour contrôler le système pneumatique du robot.

---

1. Dénudez ~ 5/16″ (~ 8 mm) à l'extrémité du fil rouge et noir 18 AWG (1 mm^2).
2. Connectez les fils à l'une des deux paires de bornes étiquetée « Vbat VRM PCM PWR » sur le PDP.
3. Mesurez la longueur requise pour atteindre les terminaux « Vin » sur le PCM. Prenez soin de laisser une longueur suffisante pour acheminer les fils autours de tout autre composants tel que la batterie et allouez assez de longueur pour gérer convenablement la disposition des câbles.
4. Coupez et dénudez ~ 5/16″ (~ 8 mm) de l'extrémité du fil.
5. Connectez le fil aux terminaux 12Vin du PCM.

2.1.14 Câbles Ethernet

REV

Requiert : 2x Câbles Ethernet

1. Connectez un câble Ethernet de la prise RJ45 (Ethernet) du roboRIO au port du module d'alimentation radio étiqueté roboRIO.
2. Connectez un câble Ethernet de la prise RJ45 de la radio la plus proche de la prise du connecteur cylindrique (étiquetée 18-24v POE) à la prise étiquetée WiFi Radio sur le RPM.

CTR

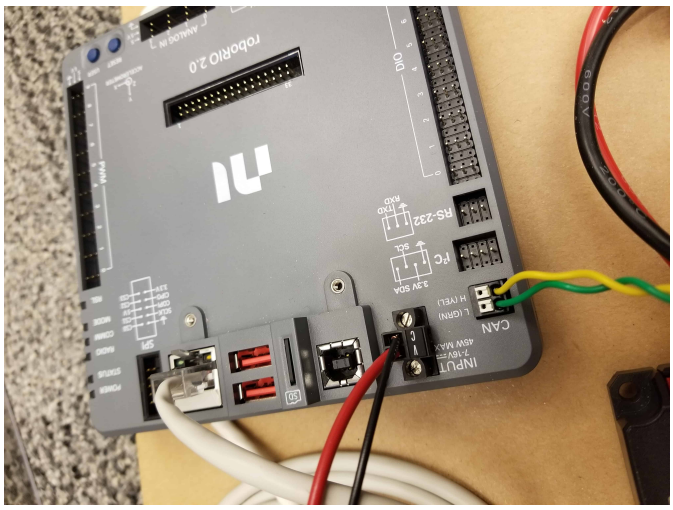
Nécessite : Câble Ethernet

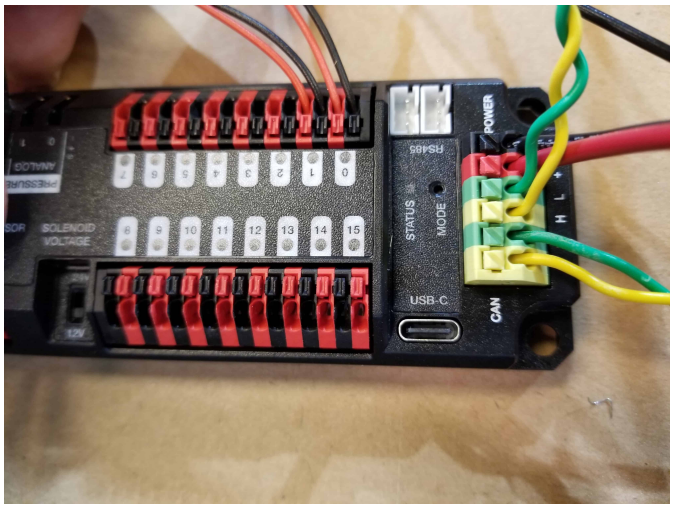
Connectez un câble Ethernet de la prise RJ45 (Ethernet) du câble Rev Passive POE au port RJ45 (Ethernet) du roboRIO.

2.1.15 Les dispositifs CAN

Bus CAN du roboRIO au contrôle pneumatique

REV





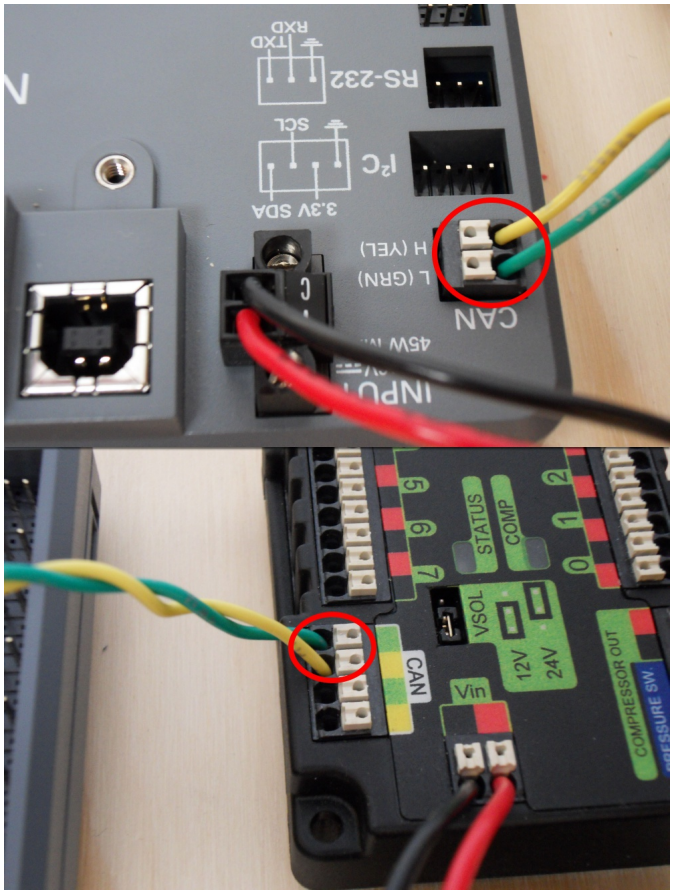
Necessite : Pince à dénuder, Petit tournevis à tête plate (optionnel), Câble jaune/vert CAN torsadé

Note : Le PH est un composant optionnel utilisé pour contrôler le système pneumatique du robot. Si vous n'utilisez pas le PH, câblez la connexion CAN directement du roboRIO (illustré dans cette étape) au PDH (illustré dans l'étape suivante).

---

1. Dénudez ~ 5/16″ (~ 8 mm) de chacun des fils CAN.
2. Insérez les fils dans les bornes approprié sur le roboRIO (Jaune->YEL, Vert->GRN).
3. Mesurez la longueur nécessaire pour atteindre les bornes CAN du PCM (l'une des deux paires disponibles). Coupez et dénudez ~ 5/16″ (~ 8 mm) cette extrémité des fils.
4. Insérez les fils dans les bornes CAN à code couleur appropriées sur le PH. Vous pouvez utiliser l'une ou l'autre des paires de bornes jaune/vert sur le PH, il n'y a pas d'entrée ou de sortie définie.

CTR



Necessite : Pince à dénuder, Petit tournevis à tête plate (optionnel), Câble jaune/vert CAN torsadé

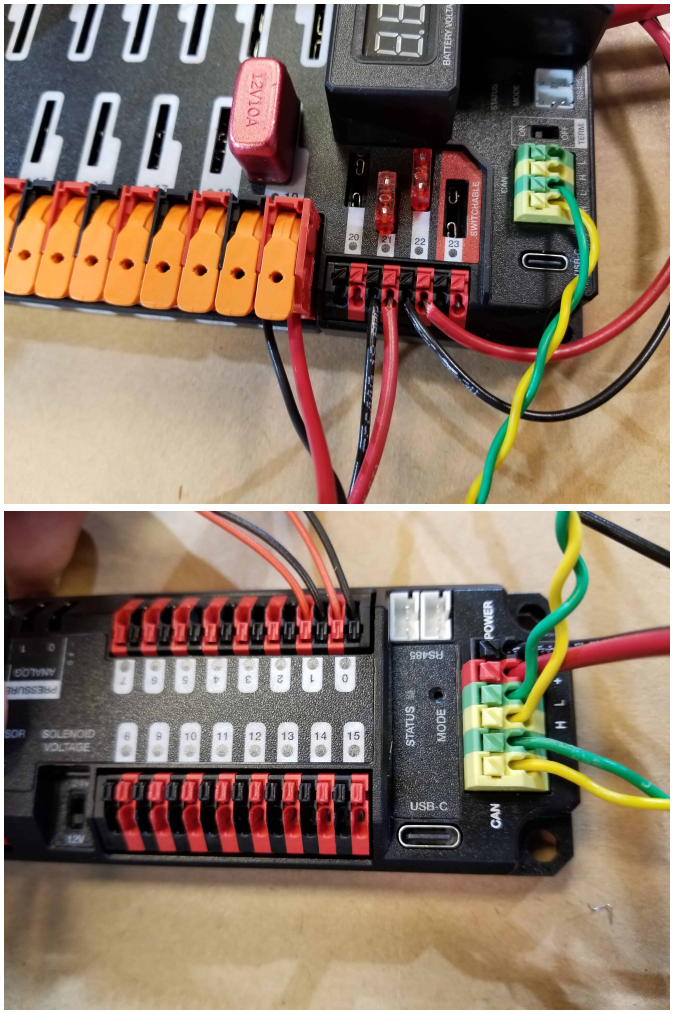
Note : Le PCM est un composant optionnel utilisé pour contrôler le système pneumatique du robot. Si vous n'utilisez pas le PCM, câblez la connexion CAN directement du roboRIO (illustré dans cette étape) au PDP (illustré à l'étape suivante).

---

1. Dénudez ~ 5/16″ (~ 8 mm) de chacun des fils CAN.
2. Insérez les fils dans les bornes approprié sur le roboRIO (Jaune->YEL, Vert->GRN).
3. Mesurez la longueur nécessaire pour atteindre les bornes CAN du PCM (l'une des deux paires disponibles). Coupez et dénudez ~ 5/16″ (~ 8 mm) cette extrémité des fils.
4. Insérez les fils dans les bornes CAN selon le code de couleur approprié sur le PCM. Vous pouvez utiliser n'importe quelle paire de bornes jaune/vert sur le PCM.

Pneumatique vers PD CAN

REV



Necessite : Pince à dénuder, Petit tournevis à tête plate (optionnel), Câble jaune/vert CAN torsadé

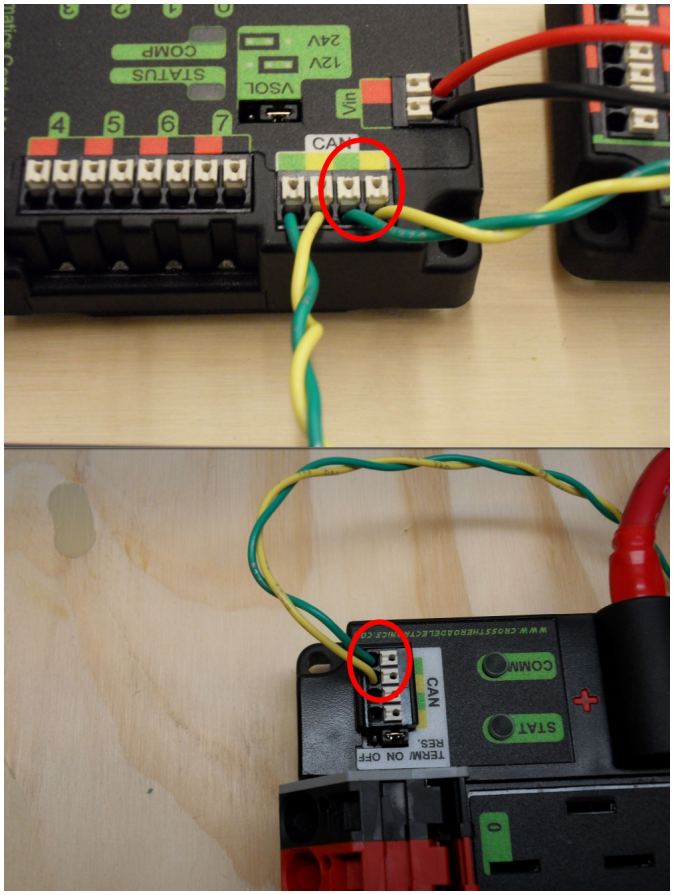
Note : Le PH est un composant optionnel utilisé pour contrôler le système pneumatique du robot. Si vous n'utilisez pas le PH, câblez la connexion CAN directement du roboRIO (illustré dans l'étape ci-dessus) au PDH (illustré dans cette étape).

---

1. Dénudez ~ 5/16″ (~ 8 mm) de chacun des fils CAN.
2. Insérez les fils dans les bornes CAN appropriées du PH.
3. Mesurez la longueur nécessaire pour atteindre les bornes CAN du PDH (l'une ou l'autre des deux paires disponibles). Coupez et dénudez ~5/16 » (~8 mm) de cette extrémité des fils.
4. Insérez les fils dans les bornes CAN à code couleur appropriées sur le PDH. Vous pouvez utiliser l'une ou l'autre des paires de bornes jaune/vert sur le PDH, il n'y a pas d'entrée ou de sortie définie.

Note : Consultez les *CAN Wiring Basics* si vous devez terminer le bus CAN ailleurs que sur le PDP.

CTR



Necessite : Pince à dénuder, Petit tournevis à tête plate (optionnel), Câble jaune/vert CAN torsadé

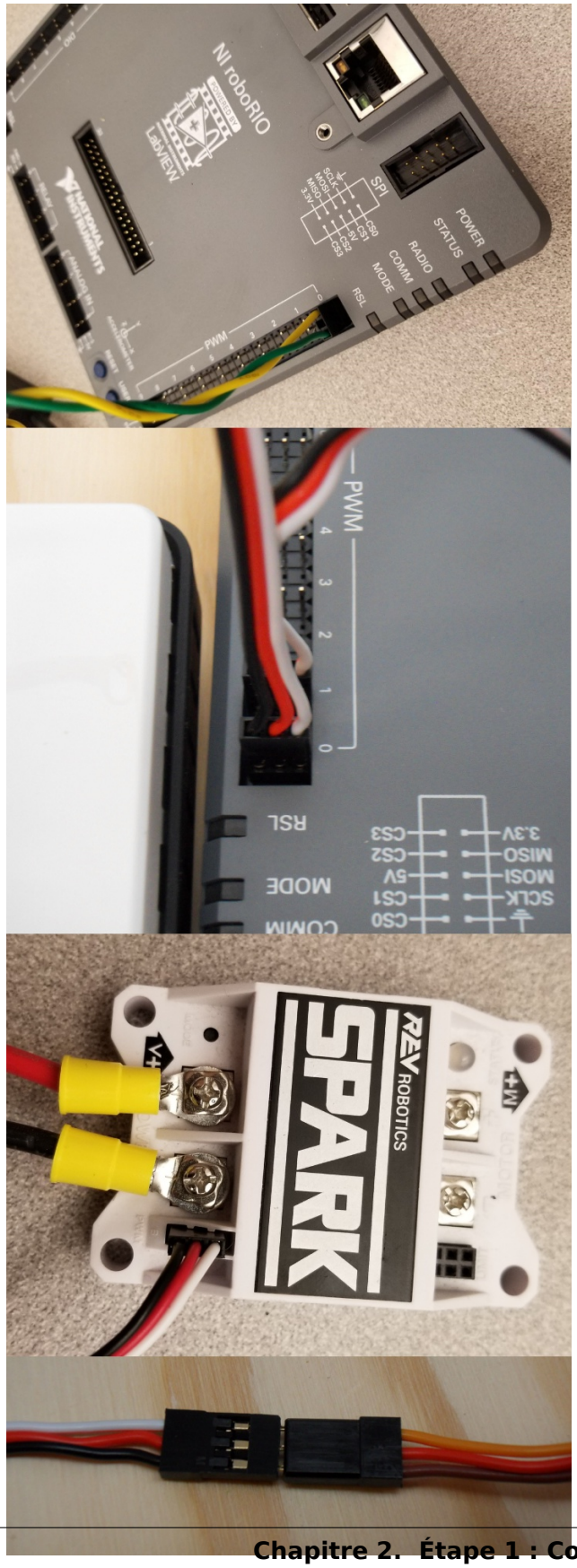
Note : Le PCM est un composant optionnel utilisé pour contrôler le système pneumatique du robot. Si vous n'utilisez pas le PCM, câblez la connexion CAN directement du roboRIO (illustré dans l'étape ci-dessus) au PDP (illustré dans cette étape).

1. Dénudez ~ 5/16″ (~ 8 mm) de chacun des fils CAN.
2. Insérez les fils dans les bornes CAN appropriés sur le PCM.
3. Mesurez la longueur nécessaire pour atteindre les bornes CAN du PDP (l'une des deux paires disponibles). Coupez et dénudez ~ 5/16″ (~ 8 mm) cette extrémité des fils.
4. Insérez les fils dans les bornes CAN de code couleur approprié sur le PDP. Vous pouvez utilisé n'importe quelle paire de bornes jaune/vert sur le PDP.

Note : Consultez les *CAN Wiring Basics* si vous devez terminer le bus CAN ailleurs que sur le PDP.

2.1.16 Fils de signal du contrôleur de moteur

PWM


NI roboRIO
LabVIEW
SPI
POWER
STATUS
RADIO
COMM
MODE
RSL
PWM
3.3V
MISO
MOSI
5V
SCLK
CS0
CS1
CS2
CS3
REV ROBOTICS
SPARK

Cette section explique comment brancher les contrôleurs SPARK MAX à l'aide des signaux de commande PWM. Il s'agit d'un point de départ recommandé car il est moins complexe et plus facile à dépanner que le fonctionnement du réseau CAN. Les SPARK MAX (et de nombreux autres contrôleurs de moteurs FRC) peuvent également être câblés à l'aide de *CAN* qui permet une configuration plus facile, les fonctionnalités avancées, de meilleures données de diagnostique et réduit la quantité de fils nécessaires.

Nécessite : 4x adaptateurs SPARK MAX PWM (si vous utilisez SPARK MAX), 4x câbles PWM (si les contrôleurs sont sans fils ou adaptateurs intégrés, autrement facultatif), 2x câble en Y PWM (facultatif)

Option 1 (connexion directe) :

1. Si vous utilisez SPARK MAX, fixez l'adaptateur PWM au SPARK MAX (petit adaptateur avec un connecteur à 3 broches avec fils noirs/blancs).
2. Au besoin, attachez des câbles d'extension PWM au contrôleur ou à l'adaptateur. Du côté du contrôleur, faire correspondre les couleurs ou les marques (certains contrôleurs peuvent avoir un câblage vert/jaune, le vert doit se connecter au noir).
3. Fixez l'autre extrémité du câble au roboRIO avec le fil noir vers l'extérieur du roboRIO. Il est recommandé de connecter le côté gauche à PWM 0 et 1 et le côté droit à PWM 2 et 3 pour concevoir une manipulation de programmation la plus simple, mais n'importe quel canal fonctionnera aussi longtemps que vous notez quel côté va à quel canal et ajuster le code en conséquence.

Option 2 (câble en Y) :

1. Si vous utilisez SPARK MAX, fixez l'adaptateur PWM au SPARK MAX (petit adaptateur avec un connecteur à 3 broches avec fils noirs/blancs).
2. Au besoin, fixez des câbles d'extension PWM entre le contrôleur ou l'adaptateur et le câble en Y de PWM. Du côté du contrôleur, faire correspondre les couleurs ou les marques (certains contrôleurs peuvent avoir un câblage vert/jaune, le vert doit se connecter au noir).
3. Connectez 1 câble en Y PWM aux 2 câbles PWM pour les contrôleurs contrôlant chaque côté du robot. Le fil brun sur le câble en Y doit correspondre au fil noir du câble PWM.
4. Connectez les câbles en Y PWM aux ports PWM sur le roboRIO. Le fil brun devrait être vers l'extérieur du roboRIO. Il est recommandé de connectez le côté gauche à PWM 0 et le côté droit à PWM 1 pour pour faciliter la programmation. Cependant, n'importe quel branchement fonctionnera convenablement si vous prenez soin de noter la disposition des PWMs et de changer le code en conséquence.

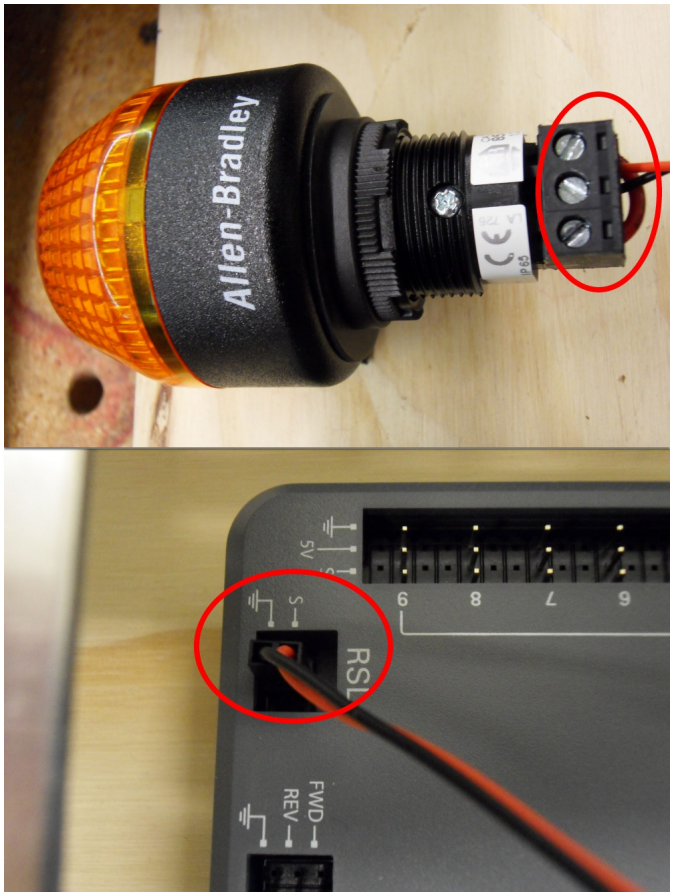
CAN

Les contrôleurs Spark MAX peuvent également être câblés via CAN. Lors du câblage CAN, l'objectif est de créer un seul bus complet partant du roboRIO à une extrémité et passant par tous les appareils CAN du robot. Il est recommandé d'avoir l'un ou l'autre des dispositifs de distribution d'alimentation à l'autre extrémité du bus car ils ont une terminaison intégrée. Si vous ne souhaitez pas localiser l'un de ces appareils à l'extrémité du bus, consultez *CAN Wiring Basics* pour plus d'informations sur comment effectuer la terminaison par vous-même.

Les contrôleurs Spark MAX sont livrés avec des câbles CAN pré-terminés par des connecteurs. Vous pouvez brancher ces câbles directement, ou acheter ou construire des rallonges pour combler des espaces plus importants. Pour vous connecter à d'autres appareils CAN tels que des contrôleurs pneumatiques, des cartes de distribution d'énergie ou le roboRIO, vous devrez soit couper l'un de ces connecteurs pré-terminés sur le contrôleur, soit couper un connecteur sur une extension, soit construire votre propre extension avec juste un seul connecteur.

Lorsque vous reliez les contrôleurs ensemble à l'aide des connecteurs fournis, assurez-vous d'utiliser le clip de retenue fourni. En cas d'indisponibilité, sécurisez la connexion avec une petite attache zip-tie, du ruban isolant ou toute autre méthode similaire.

2.1.17 Lumière de Signal du Robot



Nécessite : Pince à dénuder, Câble à 2 broches, Lumière du signal lumineux du robot, Fil rouge 18 AWG (1 mm^2) et un très petit tournevis plat

1. Coupez une extrémité du câble à 2 broches et dénudez les deux fils
2. Insérez le fil noir dans la borne centrale « N » et serrez la borne.
3. Dénudez le fil rouge 18 AWG (1 mm^2) et insérez-le dans la borne « La » et serrez la borne.
4. Coupez et dénudez l'autre extrémité du fil 18 AWG (1 mm^2) à insérer dans la borne « Lb »
5. Insérez le fil rouge du câble à deux broches dans la borne « Lb » avec le fil rouge 18 AWG (1 mm^2) et serrez la borne.
6. Connectez le connecteur à deux broche au port RSL sur le roboRIO. Le fil noir devrait être plus proche à l'extérieur du roboRIO.

Astuce : Il serait préférable de fixer temporairement le RSL au panneau de contrôle en utilisant des Zip-ties (Le RSL doit être très visible sur le robot).

2.1.18 Les disjoncteurs

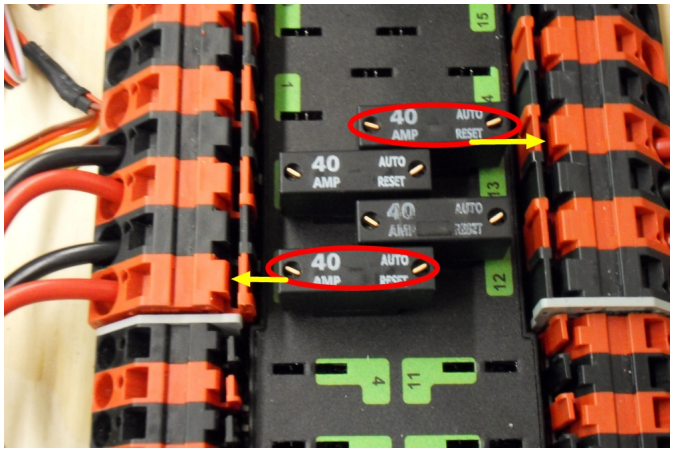
REV

Requiert : 4x disjoncteurs 40A

Insérez les disjoncteurs de 40 A dans les positions du PDH correspondant aux connecteurs Wago auxquels les contrôleurs de moteur sont connectés. Notez que le graphique blanc indique quels disjoncteurs sont associés à quelles paires de bornes.

Si vous optez de faire la Construction rapide du robot, arrêtez ici et insérez le panneau réalisé ci-haut dans le châssis de votre robot.

CTR

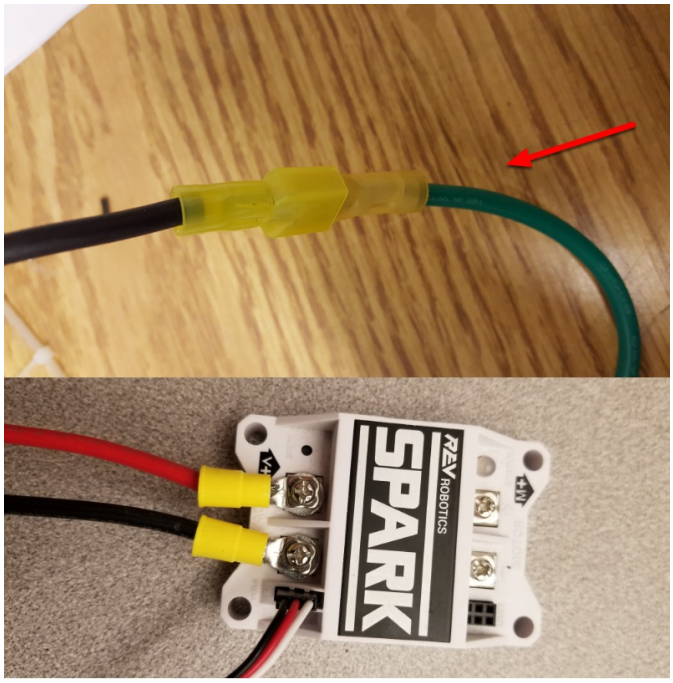


Requiert : 4x disjoncteurs 40A

Insérez les disjoncteurs de 40 ampères dans les positions du PDP correspondant aux connecteurs Wago auxquels les contrôleurs de moteur sont connectés. Notez que, pour tous les disjoncteurs, le disjoncteur correspond à la borne positive (rouge) la plus proche (voir graphique ci-dessus). Toutes les bornes négatives de la carte sont directement connectées en interne.

Si vous optez de faire la Construction rapide du robot, arrêtez ici et insérez le panneau réalisé ci-haut dans le châssis de votre robot.

2.1.19 Alimentation des moteurs



Nécessite : Pince à dénuder, Sertisseur de fil, Tournevis cruciforme, Matériel de connexion de fil

Pour chaque moteur *CIM* :
— Dénudez les extrémités des fils rouge et noir du moteur CIM
Pour les contrôleurs à fil intégré, y compris SPARK MAX (image du haut) :

1. Dénudez les fils rouges et noirs (ou fils blancs et verts) du contrôleur (le fil blanc SPARK MAX n'est pas utilisé pour les moteurs à balais tels que le CIM, il doit être fixé et l'extrémité doit être isolée par exemple avec du ruban électrique ou d'autres méthodes d'isolation).
2. Connectez les fils du moteur aux fils de sortie du contrôleur correspondant (pour les contrôleurs avec du blanc/vert, connectez-vous rouge à blanc et vert à noir). Les images ci-dessus montrent un exemple d'utilisation de bornes de déconnexion rapide qui sont fournis dans le KOP Rookie.

Pour le SPARK ou d'autres contrôleurs à fil non intégré (image du bas) :

1. Sertissez une cosse à anneau ou à fourchette sur chacun des fils moteur
2. Attachez les fils sur la sortie du contrôleur de moteur (rouge à +, noir à -)

2.1.20 STOP

Danger : Avant de brancher la batterie, assurez-vous que toutes les connexions ont été faites avec la bonne polarité. Idéalement, demander à quelqu'un qui n'a pas câblé le robot de valider que toutes les connexions soient correctes.

- Commencez avec la batterie et vérifiez que le fil rouge est connecté à la borne positive
- Vérifiez que le fil rouge passe à travers le disjoncteur principal et aboutit à la borne positive + du PDP et que le fil noir se rend directement à la borne négative -.
- Pour chaque contrôleur de moteur, vérifiez que le fil rouge va de la borne rouge PDP à la borne V+ du contrôleur de moteur (pas M+ !!!!)
- Pour chaque dispositif autre qu'un contrôleur de moteur, vérifiez que le fil rouge part d'une borne rouge sur le PD et se connecte à une borne rouge sur le composant.
- Assurez-vous que le câble PoE est branché directement sur la radio PAS SUR LE roboRIO !

Astuce : Il est aussi recommandé de mettre le robot sur des blocs afin que les roues ne touchent pas le sol ou la table de travail avant de procéder à la mise sous tension. Cette précaution évitera que le robot ne se déplace de façon dangereuse.

2.1.21 Gestion des câbles

Nécessite : attaches Zip Ties

Astuce : A ce moment, il serait préférable d'ajouter quelques Zip Ties pour placer convenablement les câbles avant de procéder plus loin. Cela aidera à garder le câblage du robot ordonné.

2.1.22 Connecter la Batterie

Connectez la batterie au côté robot du connecteur Anderson. Mettez le robot sous tension en déplaçant le levier sur le dessus du disjoncteur principal 120A.

Si des temoins LED clignotent, vous l'avez probablement bien fait. Si vous entendez un clic ou voyez de la fumée, éteignez immédiatement le système. Un clic est probablement le bruit du déclenchement des disjoncteurs.

Avant de continuer, si vous utilisez des contrôleurs SPARK MAX, il reste une étape de configuration supplémentaire à effectuer. Les contrôleurs de moteur SPARK MAX sont configurés pour contrôler un moteur sans balais par défaut. Vous pouvez le vérifier en vérifiant que le voyant du contrôleur clignote en cyan ou en magenta (indiquant respectivement le frein sans balais ou la roue libre sans balais). Pour passer en mode « avec balais », maintenir enfoncé le bouton de mode pendant 3 à 4 secondes jusqu'à ce que le voyant d'état change de couleur. La LED doit passer au bleu ou au jaune, indiquant que le contrôleur est en mode « avec balais » (respectivement freinage ou roue libre). Pour modifier le mode freinage ou roue libre, qui contrôle la rapidité avec laquelle le moteur ralentit lorsqu'un signal neutre est appliqué, appuyez brièvement sur le bouton mode.

Astuce : Pour plus d'informations sur les contrôleurs de moteur SPARK MAX, y compris la façon de tester vos moteurs/contrôleurs sans avoir à écrire de code en utilisant le client REV Hardware, veuillez consulter le document SPARK MAX Quickstart guide.

De là, vous devez vous connecter au roboRIO et essayer de télécharger votre code !

3 Étape 2 : Installation des composants logiciels

Un aperçu des composants logiciels disponibles du système de contrôle est présenté *ici*.

3.1 Préparation de l'installation hors ligne

Cet article contient des instructions/liens vers des composants qu'il serait souhaitable de réunir si vous devez effectuer l'installation hors ligne du logiciel du système de contrôle FRC® .

Astuce : Ce document compile tous les liens de téléchargement à partir des documents suivants pour faciliter l'installation sur des ordinateurs hors ligne ou sur plusieurs ordinateurs. Si vous installez sur un seul ordinateur connecté à Internet, vous pouvez ignorer cette page.

Note : L'ordre dans lequel ces outils sont installés n'a pas d'importance pour les équipes Java et C++. LabVIEW doit être installé avant les outils de jeu FRC ou les librairies tierces.

3.1.1 Documentation

Cette documentation peut être téléchargée pour une lecture hors ligne. Le lien pour télécharger la version PDF est disponible ici.

3.1.2 Logiciels d'installation

Toutes les équipes

— 2024 FRC Game Tools (Note : Click on link for « Individual Offline Installers »)
— 2024 FRC Radio Configuration Utility or 2024 FRC Radio Configuration Utility Israel Version

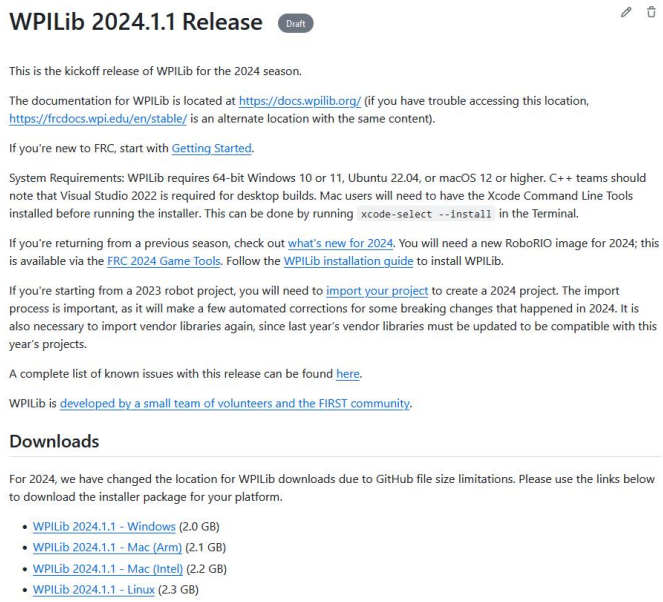
Équipes LabVIEW

— LabVIEW Base Installer (Note : Click on link for « Individual Offline Installers »)

Équipes Java/C++

— Java/C++ WPILib Installer

Once on the GitHub releases page, scroll to the download section in the middle of the page.



WPILib 2024.1.1 Release Draft

This is the kickoff release of WPILib for the 2024 season.

The documentation for WPILib is located at https://docs.wpilib.org/ (if you have trouble accessing this location, https://frcdocs.wpi.edu/en/stable/ is an alternate location with the same content).

If you're new to FRC, start with Getting Started.

System Requirements: WPILib requires 64-bit Windows 10 or 11, Ubuntu 22.04, or macOS 12 or higher. C++ teams should note that Visual Studio 2022 is required for desktop builds. Mac users will need to have the Xcode Command Line Tools installed before running the installer. This can be done by running `xcode-select --install` in the Terminal.

If you're returning from a previous season, check out what's new for 2024. You will need a new RoboRIO image for 2024; this is available via the FRC 2024 Game Tools. Follow the WPILib installation guide to install WPILib.

If you're starting from a 2023 robot project, you will need to import your project to create a 2024 project. The import process is important, as it will make a few automated corrections for some breaking changes that happened in 2024. It is also necessary to import vendor libraries again, since last year's vendor libraries must be updated to be compatible with this year's projects.

A complete list of known issues with this release can be found here.

WPILib is developed by a small team of volunteers and the FIRST community.

Downloads

For 2024, we have changed the location for WPILib downloads due to GitHub file size limitations. Please use the links below to download the installer package for your platform.

- WPILib 2024.1.1 - Windows (2.0 GB)
- WPILib 2024.1.1 - Mac (Arm) (2.1 GB)
- WPILib 2024.1.1 - Mac (Intel) (2.2 GB)
- WPILib 2024.1.1 - Linux (2.3 GB)

Then click on the correct binary for your OS and architecture to begin the download.

Note : Après avoir téléchargé le logiciel d'installation de la librairie Java/C++ WPILib, exécutez-le une fois tout en étant connecté à Internet et sélectionnez *Install for this User* ensuite *Create VS Code zip to share with other computers/OSes for offline install* et enregistrez le fichier zip VS Code téléchargé pour de futures installations hors ligne.

3.1.3 Bibliothèques/logiciels tiers

Un répertoire des logiciels de tierce-partie disponibles qui se connectent à WPILib peut être trouvé à la section *Librairies tierces*.

3.2 Installation de LabVIEW pour FRC (LabVIEW uniquement)

Note : This installation is for teams programming in LabVIEW or using NI Vision Assistant only. C++, Java, and Python teams not using these features do not need to install LabVIEW and should proceed to *Installing the FRC Game Tools*.

Les temps de téléchargement et d'installation varient considérablement selon en fonction des spécifications de l'ordinateur et de la connexion Internet, prenez note cependant que ce processus implique un téléchargement et une installation de fichiers volumineux et prendra probablement au moins une heure.

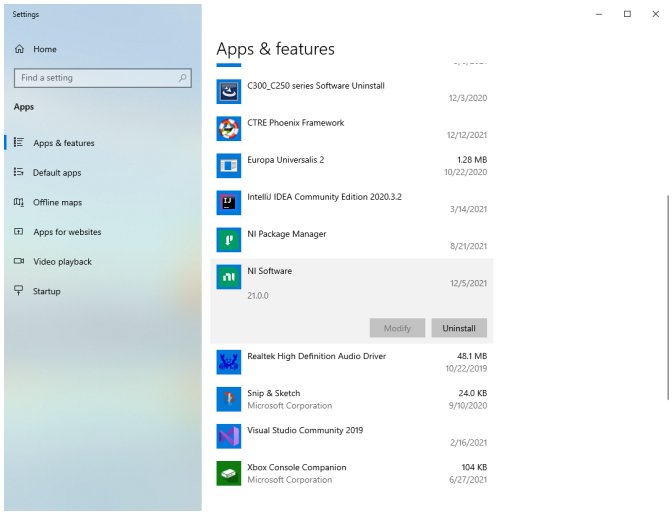
3.2.1 Exigences

— Windows 10 or higher (Windows 10, 11)

3.2.2 Désinstaller les anciennes versions (recommandé)

Note : Si vous souhaitez continuer avec la programmation de vos cRIOs, vous devrez maintenir une installation de LabVIEW pour FRC® 2014. La licence LabVIEW pour FRC 2014 a été prolongée. Bien que ces versions devraient être en mesure de coexister sur un seul ordinateur, ce n'est pas une configuration qui a été largement testée.

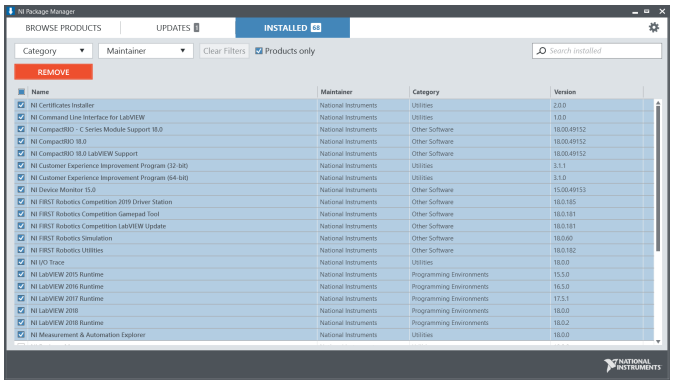
Before installing the new version of LabVIEW it is recommended to remove any old versions. The new version will likely co-exist with the old version, but all testing has been done with FRC 2024 only. Make sure to back up any team code located in the « User\LabVIEW Data » directory before un-installing. Then click Start >> Add or Remove Programs. Locate the entry labeled « NI Software », and select Uninstall.



Sélectionner les composants à désinstaller

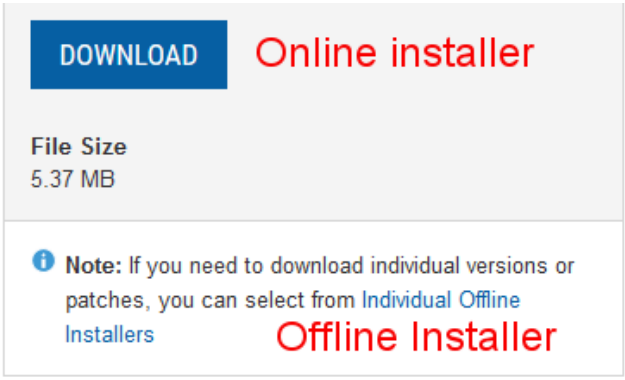
Dans la boîte de dialogue qui s'affiche, sélectionnez toutes les entrées. La façon la plus simple de le faire est de dé-sélectionner la case à cocher « Products Only » et de sélectionner la case à cocher à gauche de « Name ». Cliquez sur Supprimer. Attendez que le désinstallateur complète son travail et redémarrez si vous y êtes invité.

> **Avertissement :** Ces instructions supposent qu'aucun autre logiciel NI n'est installé. Si vous avez d'autres logiciels NI installés, il est nécessaire de décocher le logiciel qui ne doit pas être désinstallé.



3.2.3 Obtenir l'installateur LabVIEW

Download the LabVIEW for FRC 2024 installer from NI. Be sure to select the correct version from the drop-down.



Si vous souhaitez installer sur d'autres machines hors connexion, ne cliquez pas sur le bouton Télécharger, cliquez sur **Individual Offline Installers** puis cliquez sur **Download**, pour télécharger l'installateur complet.

Note : This is a large download (~10GB). It is recommended to use a fast internet connection and to use the NI Downloader to allow the download to resume if interrupted.

3.2.4 Installation de LabVIEW

NI LabVIEW nécessite une licence. La licence de chaque saison est active jusqu'au 31 janvier de l'année suivante (p. ex. la licence pour la saison 2020 expire le 31 janvier 2021)

Les équipes sont autorisées à installer le logiciel sur autant d'ordinateurs d'équipe que nécessaire, sous réserve des restrictions et des conditions de licence qui accompagnent le logiciel applicable, et à condition que seuls les membres de l'équipe ou les mentors utilisent le logiciel, et uniquement pour FRC. Les droits d'utilisation de LabVIEW sont régis uniquement par les termes des contrats de licence qui sont affichés lors de l'installation du logiciel applicable.

Démarrage de l'installation

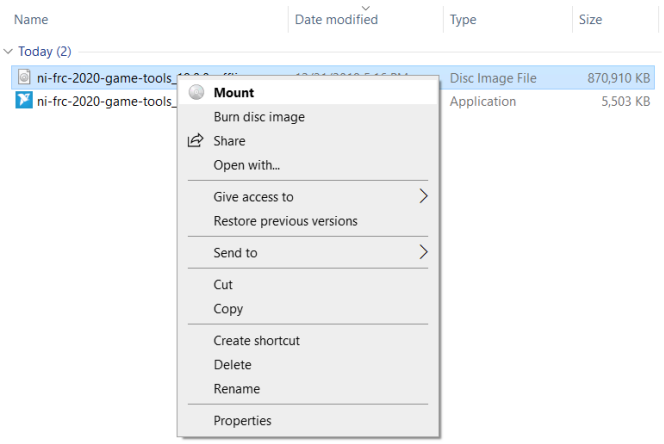
Installateur en ligne

Exécutez le fichier exe téléchargé pour démarrer le processus d'installation. Cliquez sur *Yes* si une invite de sécurité Windows apparaît.

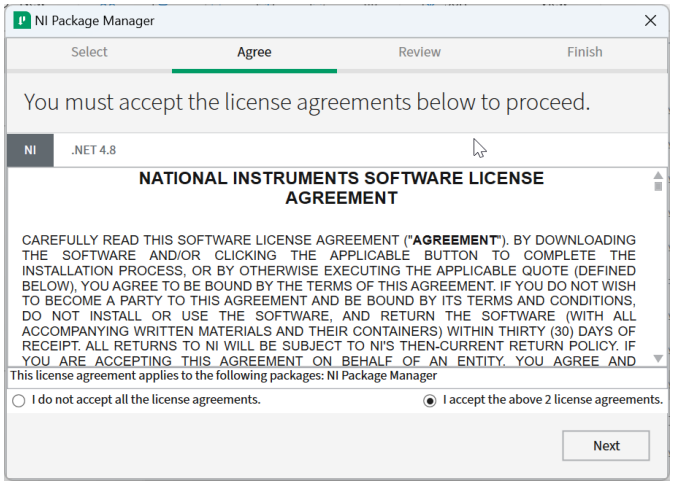
Installer hors ligne (Windows 10+)

Cliquez avec le bouton droit sur le fichier iso téléchargé et sélectionnez montage. Exécutez install.exe à partir de l'iso monté. Cliquez sur « Yes » si une invite de sécurité Windows apparait.

Note : d'autres programmes installés peuvent s'associer aux fichiers iso et l'option de montage peut ne pas apparaître. Si ce logiciel ne donne pas la possibilité de monter ou d'extraire le fichier iso, installez 7-Zip et utilisez-le pour extraire l'iso.

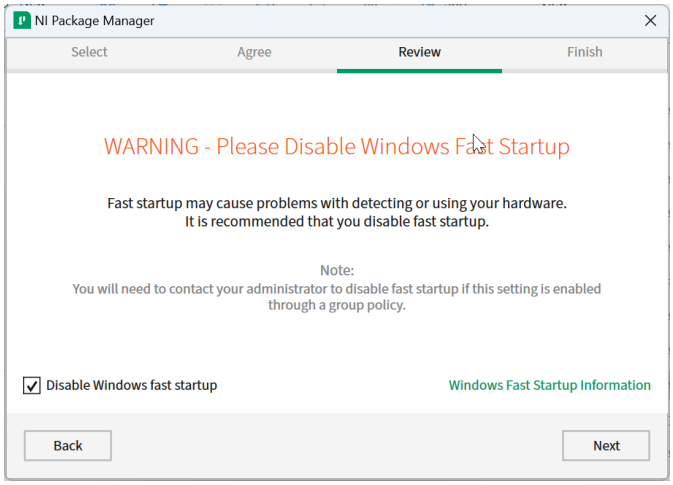


Gestionnaire de packages de licences NI



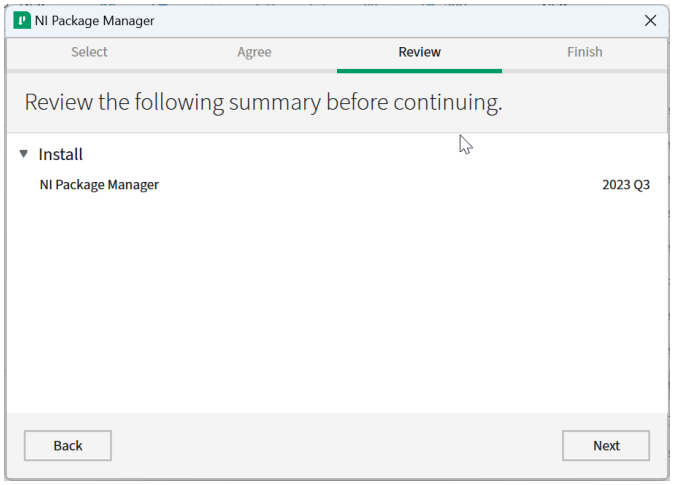
Si vous voyez cet écran, cliquez sur *Next*

Désactiver Windows Fast Startup



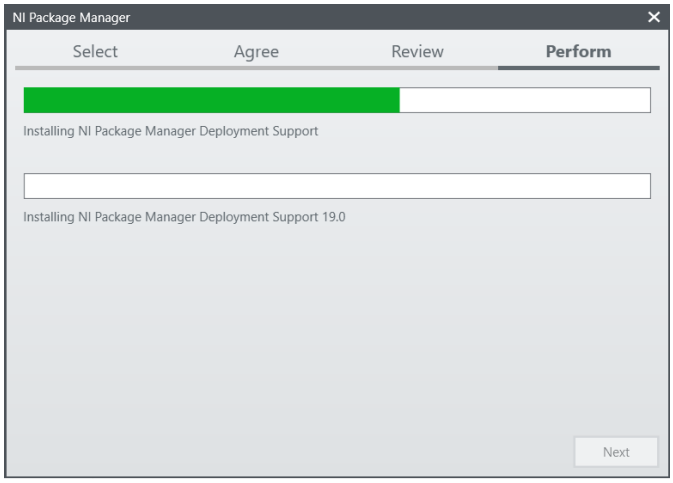
Si vous voyez cet écran, cliquez sur *Next*

Revue du gestionnaire de packages NI



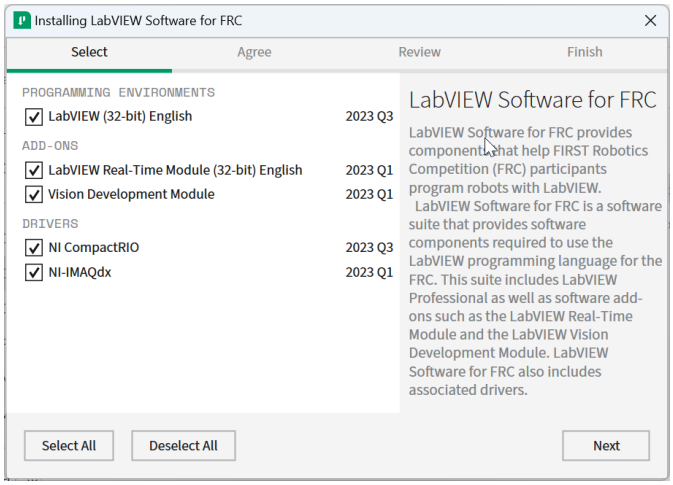
Si vous voyez cet écran, cliquez sur *Next*

Installation du gestionnaire de packages NI



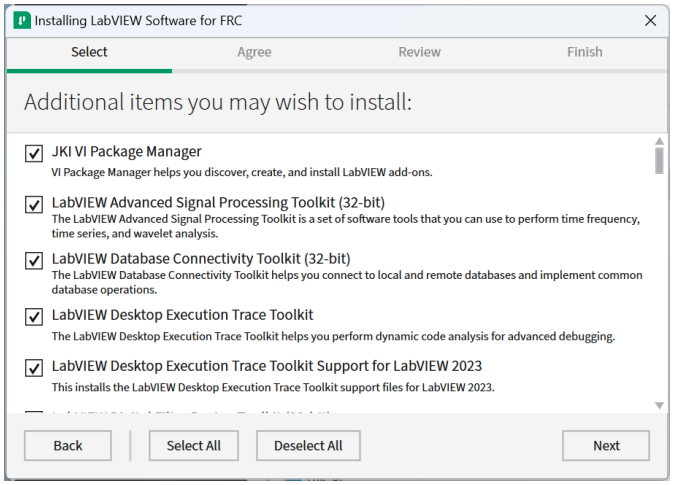
La progression de l'installation du Gestionnaire de packages NI sera suivie dans cette fenêtre

Liste des produits



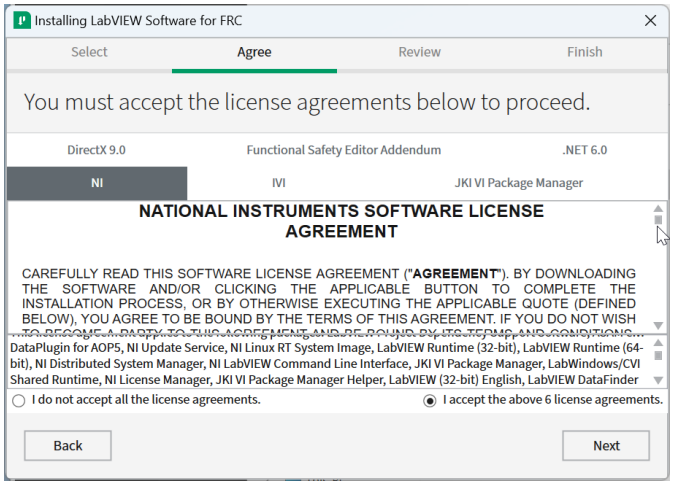
Cliquez sur *Next*

Packages supplémentaires



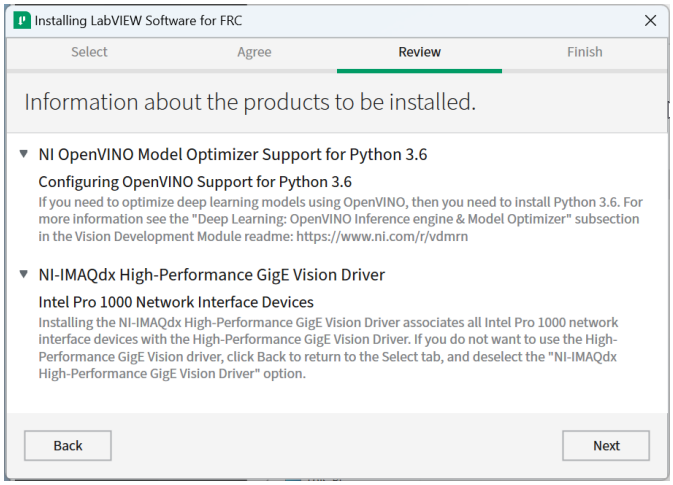
Cliquez sur *Next*

Accords de Licence



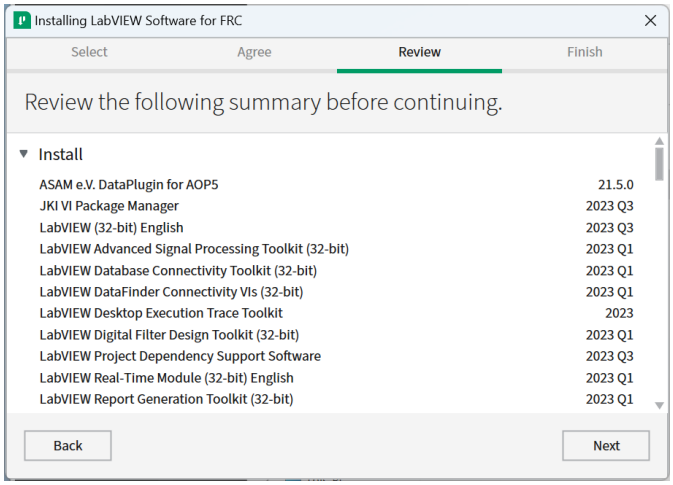
Cochez *I accept…* cliquez ensuite sur *Next*

Informations sur les produits



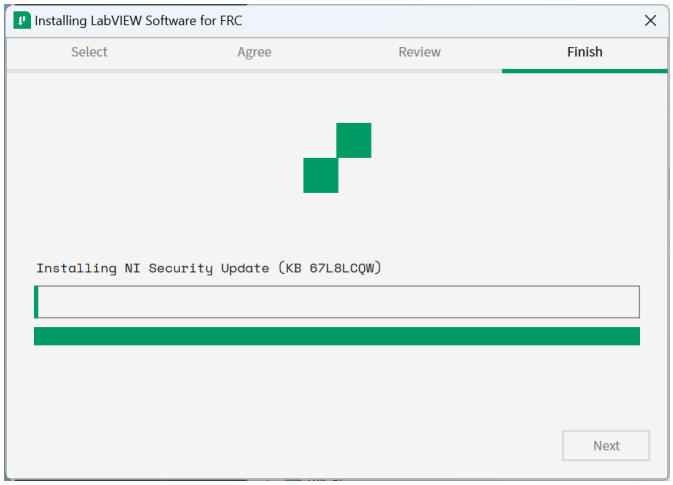
Cliquez sur *Next*

Démarrer l'installation



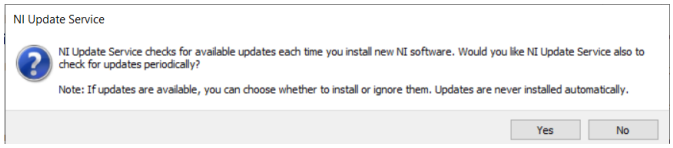
Cliquez sur *Next*

Progression globale



La progression de l'installation globale sera suivie dans cette fenêtre

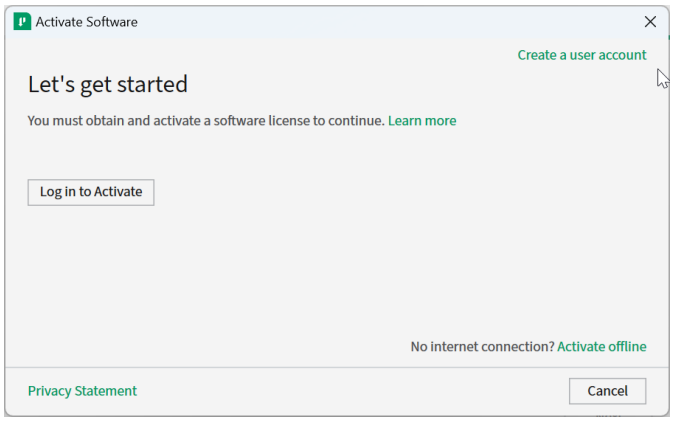
3.2.5 Service de mises à jour NI



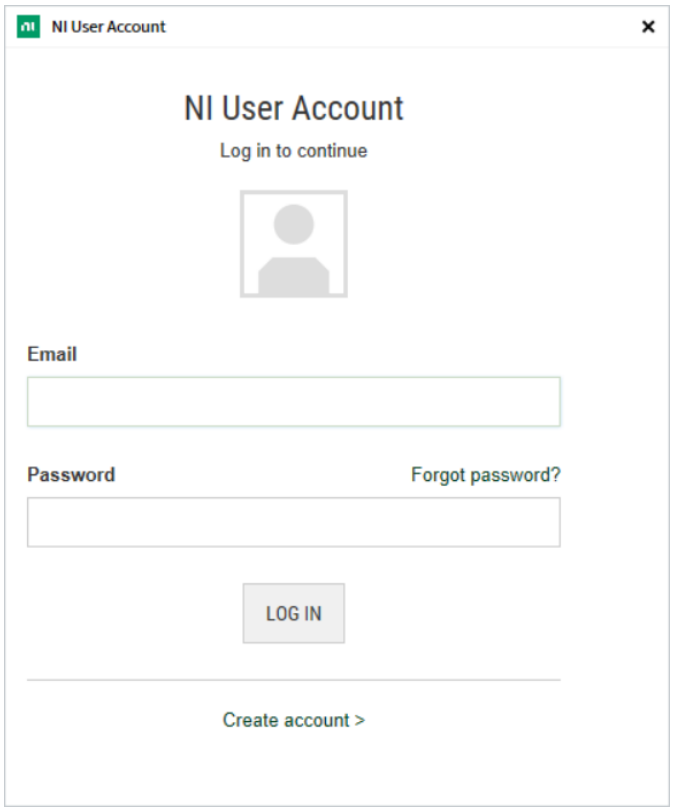
Vous serez invité à activer ou non le service de mises à jour NI. Vous pouvez choisir de ne pas activer le service de mises à jour.

Avertissement : Il n'est pas recommandé d'installer ces mises à jour sauf sur recommandation de FRC à travers nos canaux de communication habituels (FRC Blog, Team Updates ou E-mail Blasts).

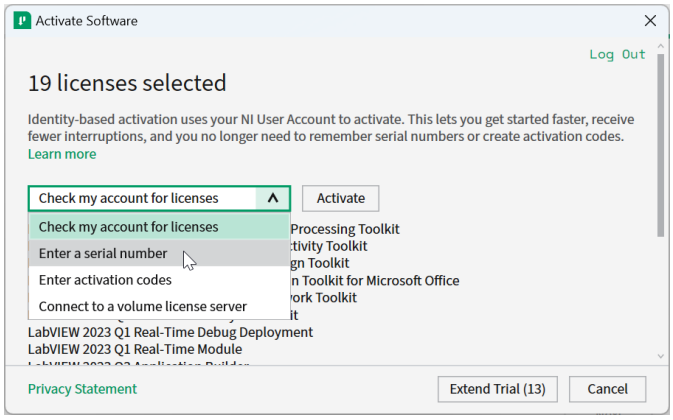
Assistant Activation NI



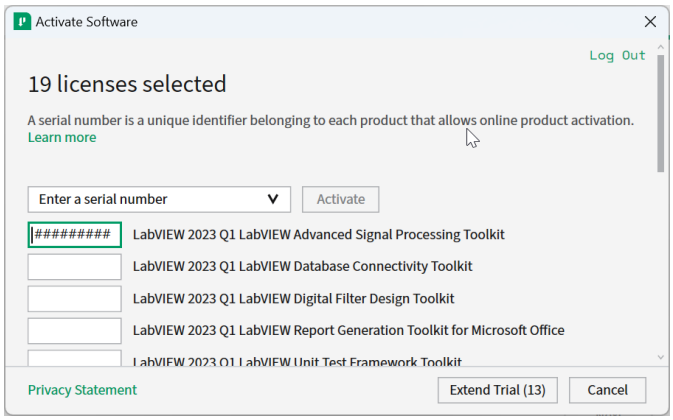
Cliquez sur le bouton *Log in to Activate*.



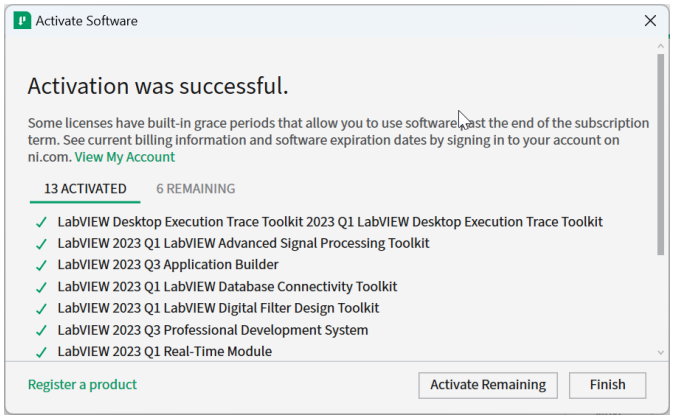
Connectez-vous au compte ni.com. Si vous n'avez pas de compte, sélectionnez *Create account* pour créer un compte gratuit.



Dans la liste déroulante, sélectionnez entrer un numéro de série

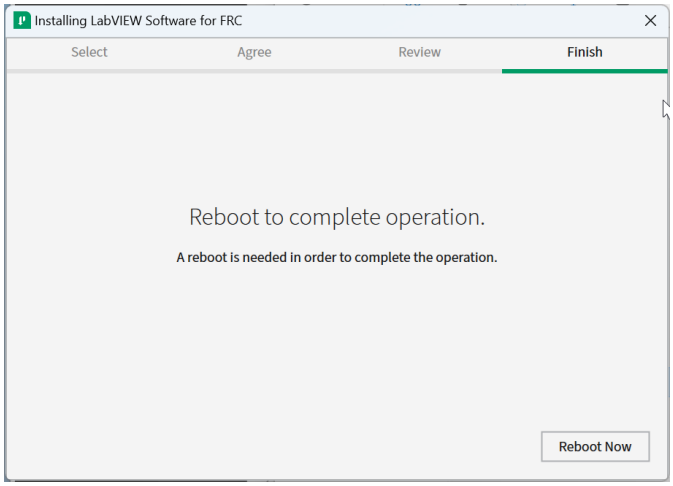


Entrez le numéro de série dans toutes les cases. Cliquez sur *Activate*.



Si vos produits s'activent avec succès, un message « Activation Successful » apparaîtra. Si le numéro de série était incorrect, une zone de texte s'affichera et vous pourrez ressaisir le numéro et sélectionner *Try Again*. Les éléments affichés ci-dessus ne devraient pas s'activer. Si tout est activé avec succès, cliquez sur *Finish*.

Redémarrer



Sélectionnez *Reboot Now* après la fermeture de tous les programmes ouverts.

3.3 Installer les Outils de Jeu FRC

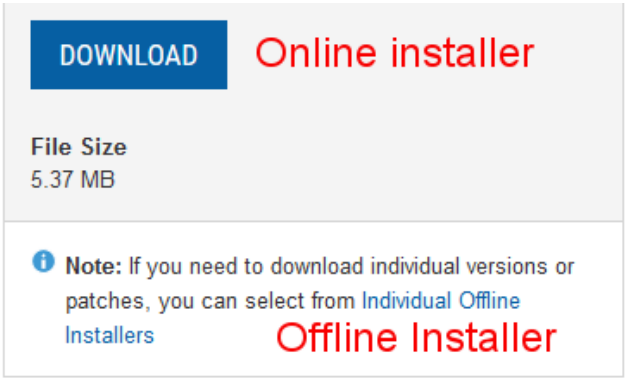
Les outils de jeu FRC® Game Tools contiennent les composants logiciels suivants :
- Mise à jour LabVIEW
- Driver Station FRC
- Outil d'imagerie et images FRC roboRIO

Les composants d'exécution LabVIEW requis pour la station de pilotage et l'outil d'imagerie sont inclus dans ce package.

Note : Aucun composant du package LabVIEW Software for FRC n'est requis pour exécuter la Driver Station ou l'outil d'imagerie.

3.3.1 Exigences

- Windows 10 or higher (Windows 10, 11).
- Download the FRC Game Tools from NI.



Si vous souhaitez installer sur d'autres machines hors ligne, cliquez d'abord sur *Individual Offline Installers* avant de cliquer ensuite sur *Download* pour télécharger l'installateur complet.

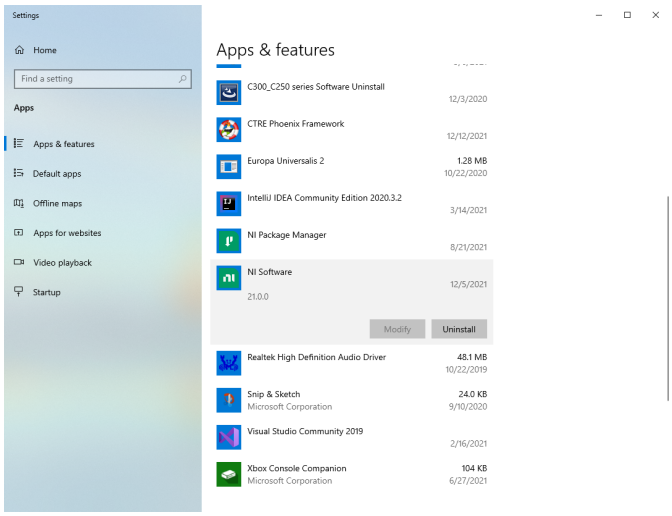
3.3.2 Désinstaller les anciennes versions (Conseillé)

Important : Les équipes programmant en LabVIEW ont déjà complété cette étape, ne la répétez pas. Elles doivent donc passer à la section *Installation*.

Before installing the new version of the FRC Game Tools it is recommended to remove any old versions. The new version will likely co-exist with the old version (note that the DS will overwrite old versions), but all testing has been done with FRC 2024 only. Then click Start >> Add or Remove Programs. Locate the entry labeled « NI Software », and select *Uninstall*.

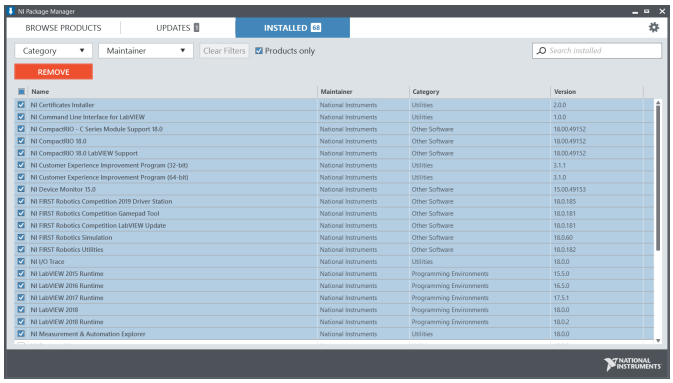
Note : It is only necessary to uninstall previous versions when installing a new year's tools (or when a beta is installed). For example, uninstall the 2021 tools before installing the 2022

tools. It is not necessary to uninstall before upgrading to a new update of the 2022 game tools.



Sélection des Composants à Désinstaller

Dans la boîte de dialogue qui s'affiche, sélectionnez toutes les entrées. La façon la plus simple de le faire est de dé-sélectionner d'abord la case à cocher *Products Only* et de sélectionner ensuite la case à cocher à gauche de « Name ». Cliquez finalement sur *Remove*. Attendez que le désinstallateur se termine et redémarrez si vous y êtes invité.



3.3.3 Installation

Important : Le programme d'installation des Outils de Jeu peut indiquer que le .NET Framework 4.6.2 dois être mis à jour ou installé. Suivez les instructions à l'écran pour finaliser l'installation, y compris un redémarrage si demandé. Après, continuer l'installation des Outils de Jeu FRC, redémarrez l'installateur si nécessaire.

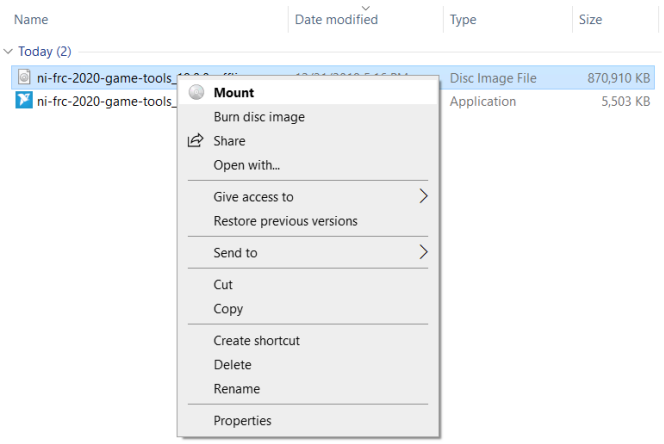
Extraction

En ligne

Exécutez le fichier exécutable téléchargé pour démarrer le processus d'installation. Cliquez sur *Yes* si une invite de sécurité Windows s'affiche.

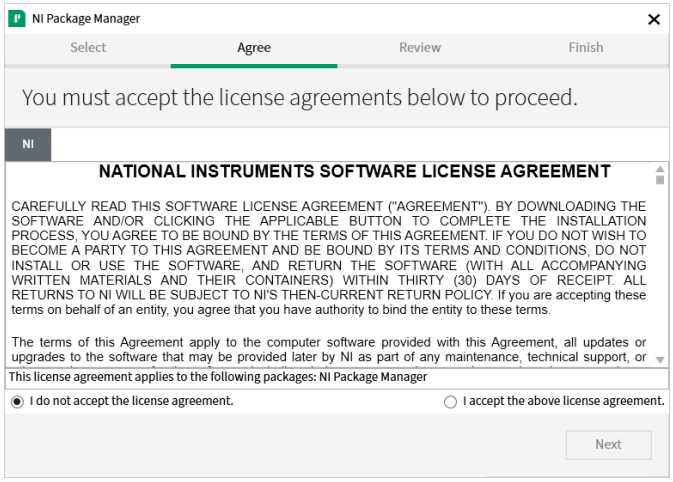
Hors ligne (Windows 10+)

Cliquez avec le bouton droit sur le fichier iso téléchargé et sélectionnez *mount*. Exécutez le fichier `install.exe` à partir de l'iso monté. Cliquez sur *Yes* si une invite de sécurité Windows s'affiche.



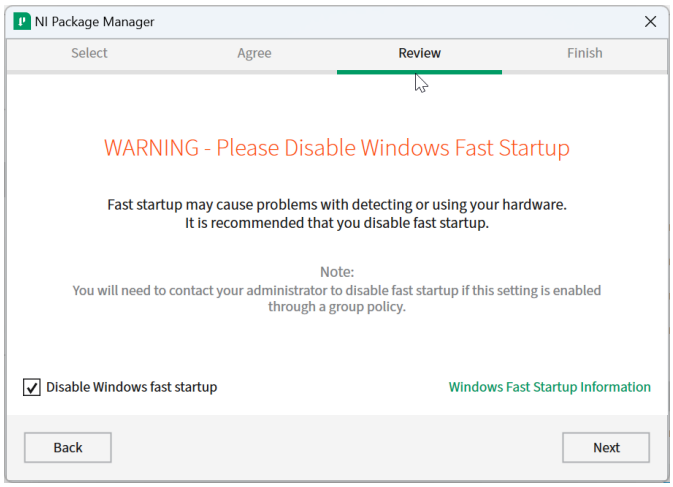
Note : D'autres programmes installés peuvent s'associer aux fichiers iso et l'option *mount* peut ne pas apparaître. Si ce logiciel ne donne pas la possibilité de monter ou d'extraire le fichier iso, installez 7-Zip et utilisez-le pour extraire l'iso.

License de Gestionnaire de Packages NI (NI Package Manager License)



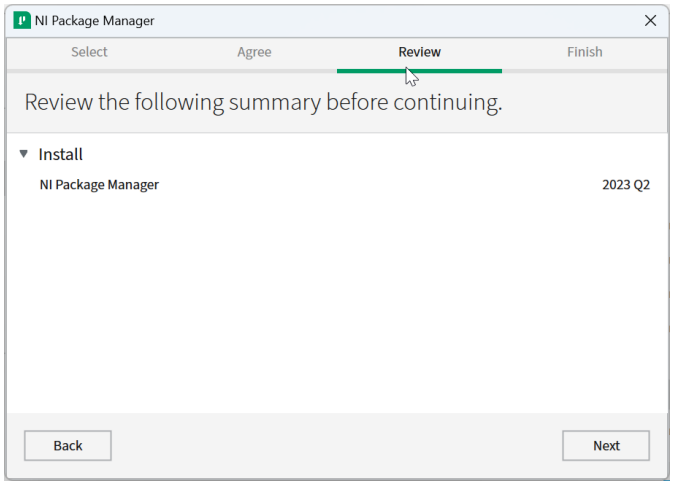
Si vous voyez cet écran, cliquez sur *Next*. Cet écran confirme que vous acceptez le contrat de licence NI Package Manager.

Désactiver le démarrage rapide de Windows



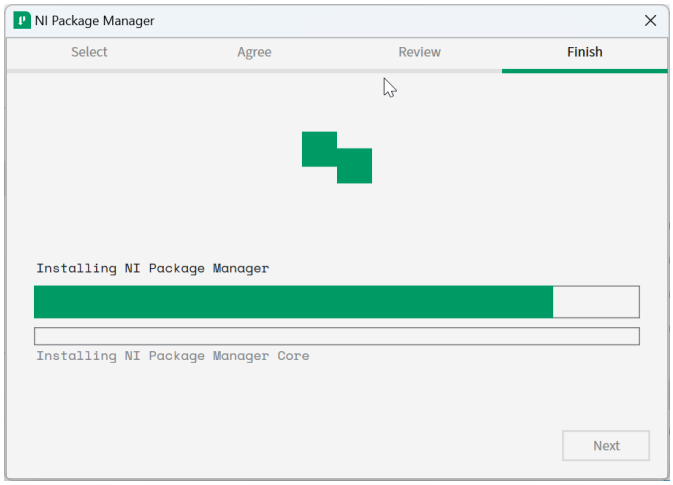
Il est recommandé de laisser cet écran tel qu'il est, car Windows Fast Startup peut causer des problèmes avec les pilotes NI requis pour l'image du roboRIO. Procédez et cliquez sur *Next*.

Revue du gestionnaire de packages NI



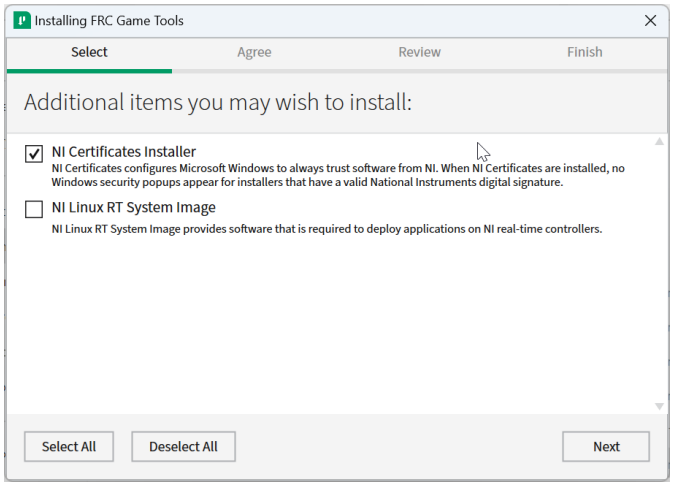
Si vous voyez cet écran, cliquez sur *Next*.

L’installation du gestionnaire de packages NI



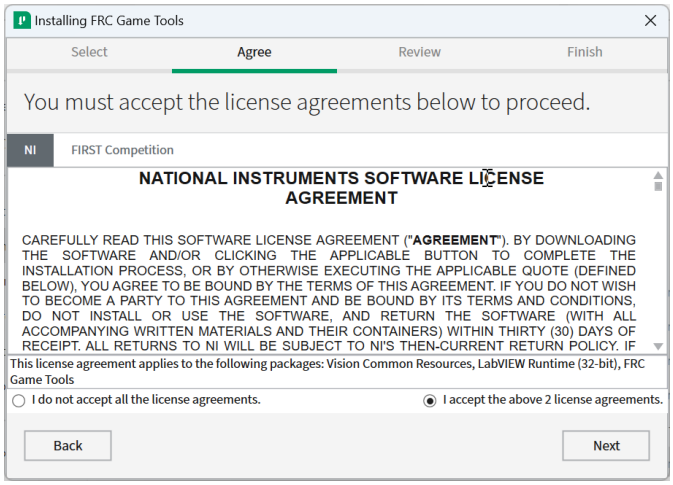
La progression de l’installation du gestionnaire NI Package Manager sera suivie de cette fenêtre.

Logiciel Supplémentaire



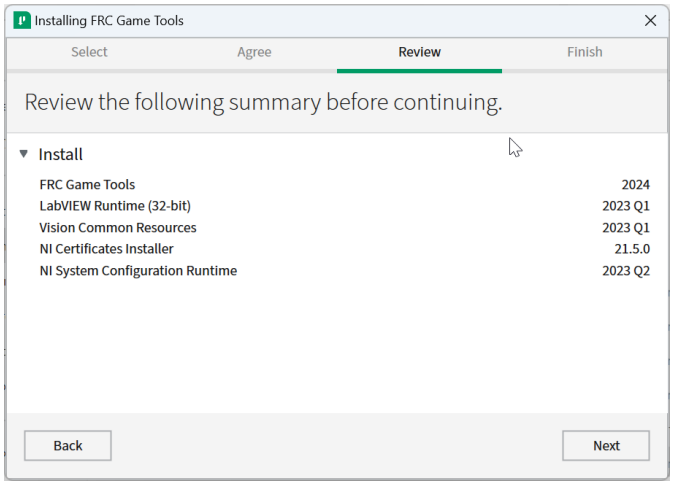
Si vous voyez cet écran, cliquez sur *Next*.

Accords de License



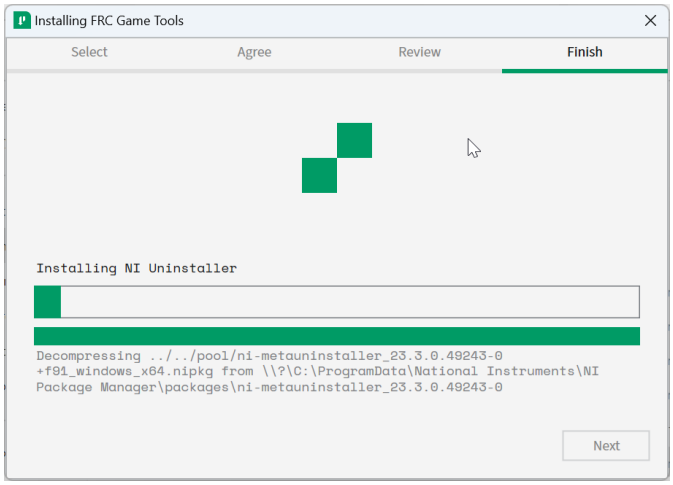
Sélectionnez *I accept...* puis cliquez sur *Next*

Résumé des commentaires



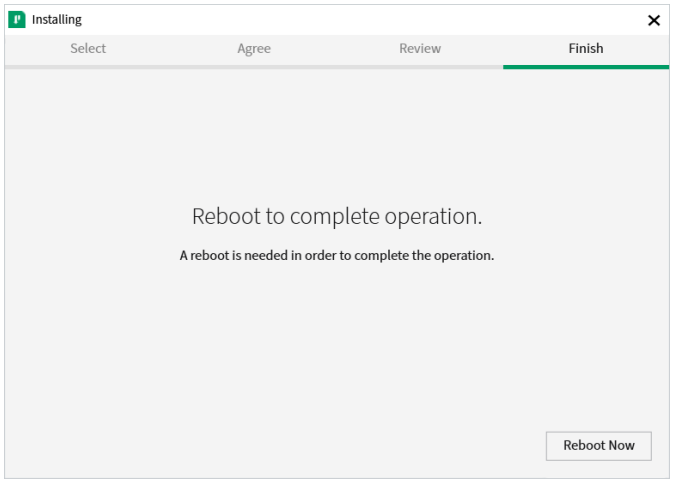
Cliquez sur *Next*.

Progrès de Détail



Cet écran présente le processus d'installation, procédez et appuyez sur *Next* quand il est terminé.

3.3.4 Faire un redémarrage, pour compléter l'Installation.



Si vous y êtes invité, sélectionnez *Reboot Now*, après avoir fermé tous les programmes ouverts.

3.4 Guide d'installation WPILib

This guide is intended for Java and C++ teams. LabVIEW teams can skip to *Installation de LabVIEW pour FRC (LabVIEW uniquement)*. Python teams can skip to *Guide d'installation de Python*. Additionally, the below tutorial shows Windows 10, but the steps are identical for all operating systems. Notes differentiating operating systems will be shown.

3.4.1 Prérequis

Supported Operating Systems and Architectures :

- Windows 10 & 11, 64 bit only. 32 bit and Arm are not supported
- Ubuntu 22.04, 64 bit. Other Linux distributions with glibc >= 2.34 may work, but are unsupported
- macOS 11 or higher, both Intel and Arm for Java. C++ requires macOS 12 or higher with Xcode 14.

Avertissement : The following OSes are no longer supported : macOS 10.15, Ubuntu 18.04 & 20.04, Windows 7, Windows 8.1, and any 32-bit Windows.

WPILib est conçu pour s'installer dans différents dossiers correspondant à différentes années, de sorte qu'il n'est pas nécessaire de désinstaller une version précédente avant d'installer la version WPILib de l'année en cours.

3.4.2 Downloading

WPILib Installer

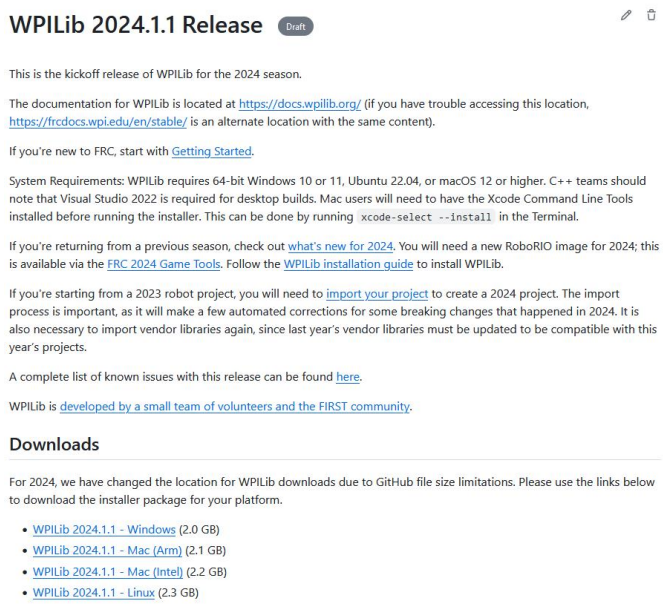
WPILib 2024.3.2 Release - March 14, 2024 Downloads

Downloads for other platforms

Release Notes

You can download the latest release of the installer from GitHub.

Once on the GitHub releases page, scroll to the Downloads section.



WPILib 2024.1.1 Release Draft

This is the kickoff release of WPILib for the 2024 season.

The documentation for WPILib is located at https://docs.wpilib.org/ (if you have trouble accessing this location, https://frcdocs.wpi.edu/en/stable/ is an alternate location with the same content).

If you're new to FRC, start with Getting Started.

System Requirements: WPILib requires 64-bit Windows 10 or 11, Ubuntu 22.04, or macOS 12 or higher. C++ teams should note that Visual Studio 2022 is required for desktop builds. Mac users will need to have the Xcode Command Line Tools installed before running the installer. This can be done by running `xcode-select --install` in the Terminal.

If you're returning from a previous season, check out what's new for 2024. You will need a new RoboRIO image for 2024; this is available via the FRC 2024 Game Tools. Follow the WPILib installation guide to install WPILib.

If you're starting from a 2023 robot project, you will need to import your project to create a 2024 project. The import process is important, as it will make a few automated corrections for some breaking changes that happened in 2024. It is also necessary to import vendor libraries again, since last year's vendor libraries must be updated to be compatible with this year's projects.

A complete list of known issues with this release can be found here.

WPILib is developed by a small team of volunteers and the FIRST community.

Downloads

For 2024, we have changed the location for WPILib downloads due to GitHub file size limitations. Please use the links below to download the installer package for your platform.

- WPILib 2024.1.1 - Windows (2.0 GB)
- WPILib 2024.1.1 - Mac (Arm) (2.1 GB)
- WPILib 2024.1.1 - Mac (Intel) (2.2 GB)
- WPILib 2024.1.1 - Linux (2.3 GB)

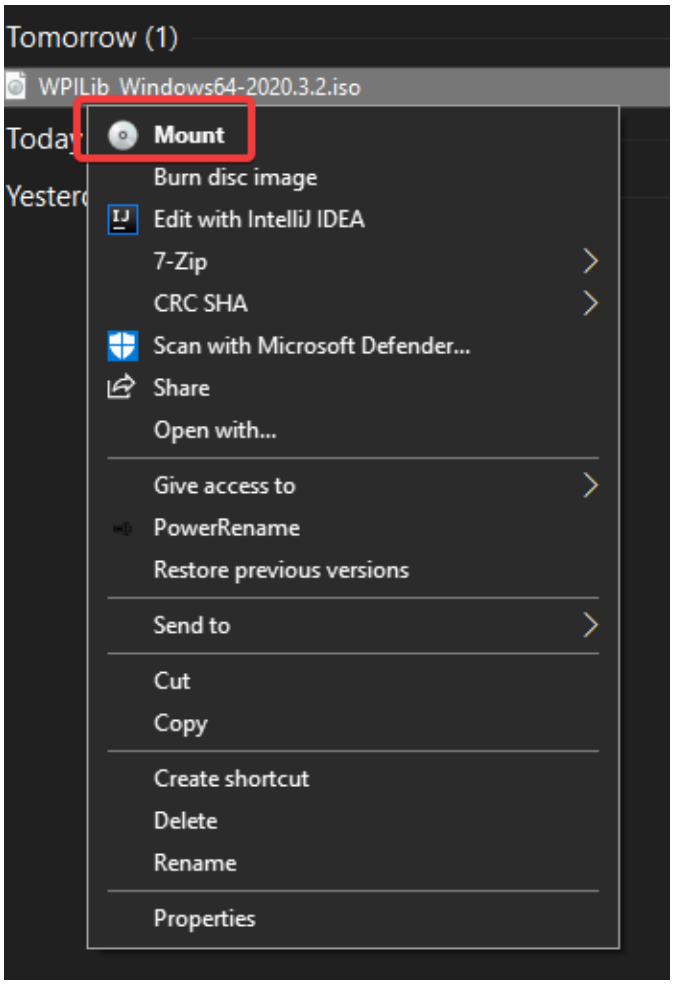
Then click on the correct binary for your OS and architecture to begin the download.

3.4.3 Extraction de l'installateur

Lorsque vous téléchargez l'installateur WPILib, celui-ci est distribué sous la forme d'un fichier d'image de disque `.iso` pour Windows, `.tar.gz` pour Linux, et distribué sous la forme de `DMG` pour MacOS.

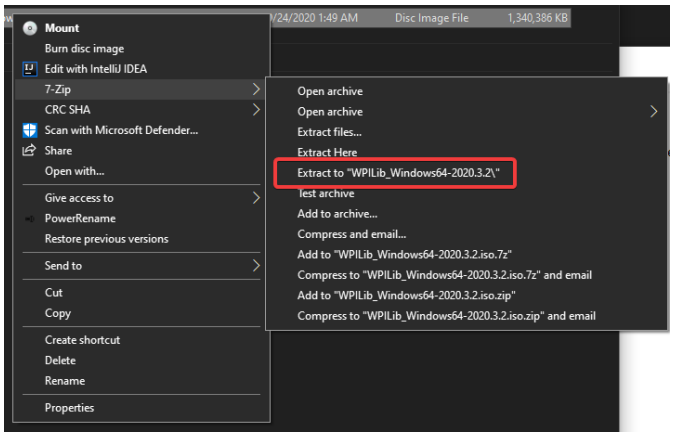
Windows 10+

Windows 10+ users can right click on the downloaded disk image and select *Mount* to open it. Then launch `WPILibInstaller.exe`.



Note : Other installed programs may associate with iso files and the *mount* option may not appear. If that software does not give the option to mount or extract the iso file, then follow the directions below.

You can use 7-zip to extract the disk image by right-clicking, selecting *7-Zip* and selecting *Extract to…*. Windows 11 users may need to select *Show more options* at the bottom of the context menu.



After opening the `.iso` file, launch the installer by opening `WPILibInstaller.exe`.

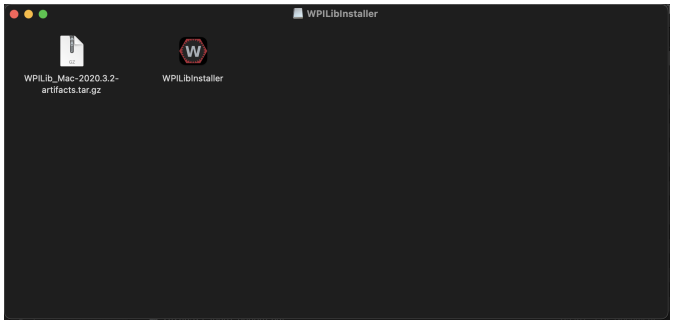
Note : After launching the installer, Windows may display a window titled « Windows protected your PC ». Click *More info*, then select *Run anyway* to run the installer.

Pour macOS

For this release, macOS users will need to have the Xcode Command Line Tools installed before running the installer ; we are working on removing this requirement in a future release. This can be done by running `xcode-select --install` in the Terminal.

Important : When upgrading from a 2024 beta release or 2024.1.1, it's necessary to manually delete AdvantageScope before running the installer. Navigate to `~/wpilib/2024/tools` and delete AdvantageScope.

Les utilisateurs du système macOS peuvent cliquer deux fois sur le `DMG` téléchargé, puis sélectionnez `WPILibInstaller` pour lancer l'application.



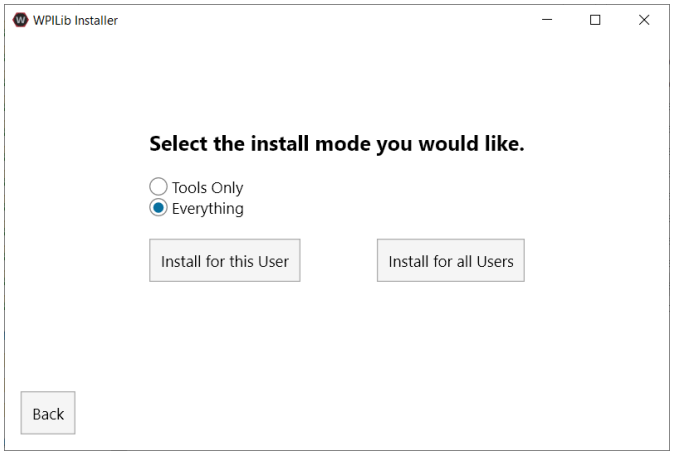
Pour Linux

Les utilisateurs de Linux devraient extraire le `.tar.gz` téléchargé, puis lancer `WPILibInstaller`. Ubuntu traite les fichiers exécutables dans l'explorateur de fichiers comme des libairies partagées, de sorte qu'en les double-cliquant, on ne les exécutera pas. À la place, exécutez les commandes suivantes dans un terminal avec `<version>` remplacé par la version que vous êtes en train d'installer.

```
$ tar -xf WPILib_Linux-<version>.tar.gz
$ cd WPILib_Linux-<version>/
$ ./WPILibInstaller
```

3.4.4 Exécution de l'installateur

Lors de l'ouverture de l'installateur, l'écran ci-dessous vous sera présenté. Procédez et appuyez sur *Start*.



Cela présente une liste d'options incluses avec l'installation de WPILib.

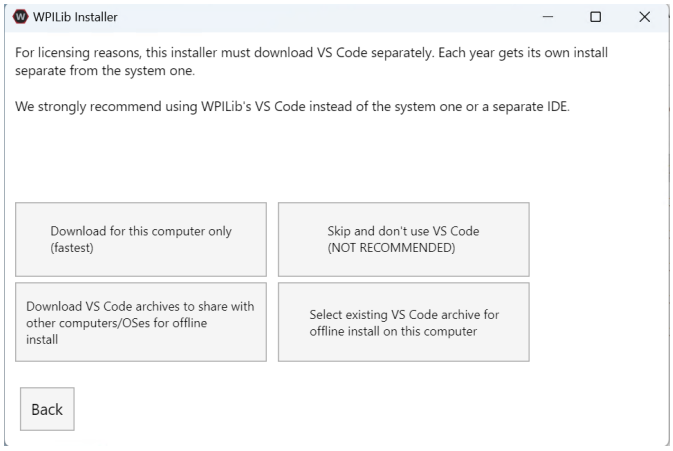
— *Tools Only* installs just the WPILib tools (Pathweaver, Shuffleboard, RobotBuilder, SysId, Glass, and OutlineViewer) and JDK.
— *Everything* installe l'environnement de développement complet (VS Code, extensions, toutes les dépendances), les outils WPILib et le JDK.

Vous remarquerez deux boutons, *Install for this User* et *Install for all Users*. *Install for this User* ne l'installe que sur le compte utilisateur actuel et ne nécessite pas de privilèges d'administrateur. Cependant, *Install for all Users* installe les outils pour tous les comptes système et *nécessitera* un accès administrateur. *Install for all Users* n'est pas une option pour macOS et Linux.

Note : Si vous sélectionnez *Install for all Users*, Windows vous demandera un accès administrateur via le contrôle de compte d'utilisateur pendant l'installation.

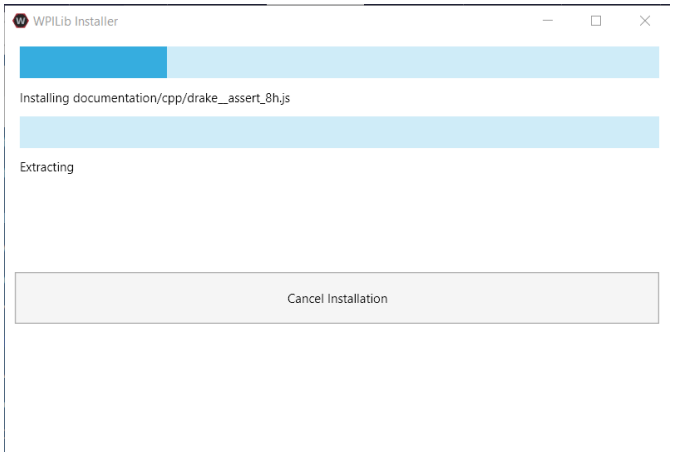
Sélectionnez l'option qui vous convient et l'écran d'installation suivant vous sera présenté.

Cet écran suivant concerne le téléchargement de VS Code. Malheureusement, pour des raisons de licence, VS Code ne peut pas être intégré dans l'installateur.

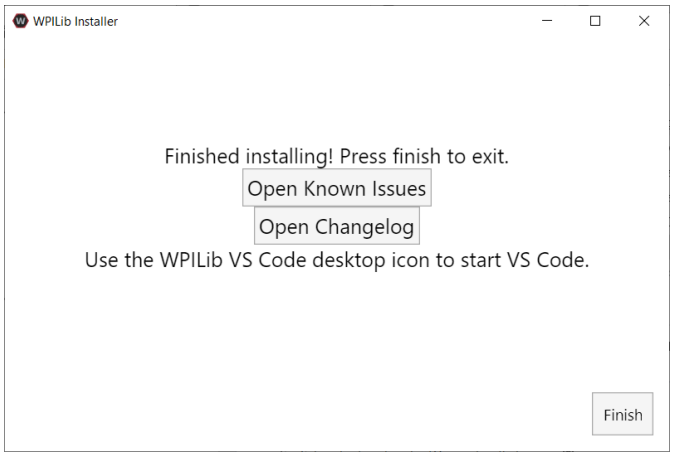


- Télécharger pour cet ordinateur uniquement
  - Cette option télécharge VS Code uniquement pour la plate-forme actuelle, ce qui correspond également au plus petit téléchargement.
- Ignorer et ne pas utiliser VS Code
  - Ignore l'installation de VS Code. Cette option est utile pour les installations ou les configurations avancées. Généralement pas recommandé.
- Select existing VS Code archive for offline install on this computer
  - La sélection de cette option fera apparaître une invitation vous permettant de sélectionner un fichier zip préexistant de VS Code qui a été téléchargé précédemment par le programme d'installation. Cette option ne vous permet **pas** de sélectionner une copie déjà installée de VS Code sur votre machine.
- Create VS Code archives to share with other computers/OSes for offline install
  - Cette option télécharge et enregistre une copie de VS Code pour toutes les plates-formes, ce qui est utile pour partager la copie du programme d'installation.

Go ahead and select *Download for this computer only*. This will begin the download process and can take a bit depending on internet connectivity (it's ~100MB). Once the download is done, select *Next*. You should be presented with a screen that looks similar to the one below.



Une fois l'installation terminée, l'écran de fin d'installation vous sera présenté.



Important : WPILib installe une version distincte de VS Code. Il n'utilise pas une installation déjà existante. Chaque année a sa propre copie des outils ajoutés avec l'année. IE : `WPILib VS Code 2022`. Veuillez lancer le VS Code WPILib non une copie installée du système !

Félicitations, l'environnement de développement et les outils WPILib sont maintenant installés sur votre ordinateur ! Appuyez sur Terminer pour quitter le programme d'installation.

3.4.5 Après l'installation

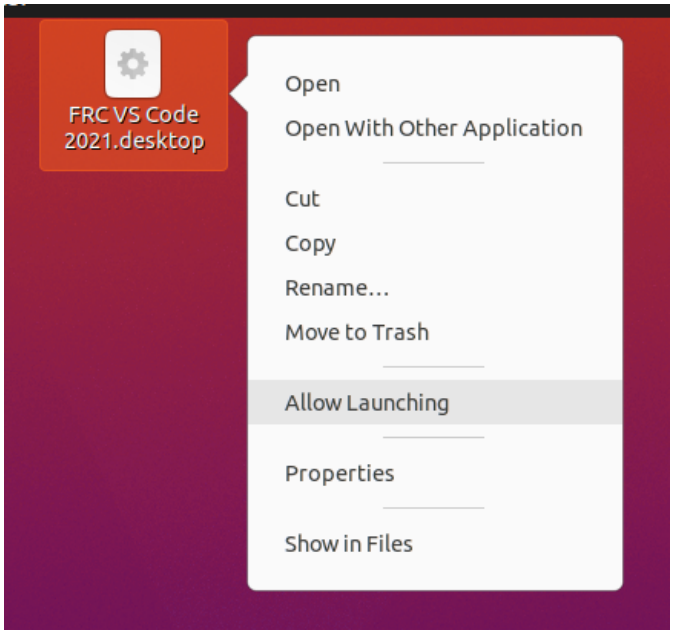
Certains systèmes d'exploitation nécessitent une dernière action pour terminer l'installation.

Pour macOS

Après l'installation, l'installateur ouvre le dossier WPILib VS Code. Faites glisser l'application VS Code sur le dock. Éjectez l'image WPILibInstaller du bureau.

Pour Linux

Certaines versions de Linux (par exemple Ubuntu 20.04) vous obligent à donner au raccourci de bureau la possibilité de démarrer. Cliquez à droite sur l'icône de bureau et sélectionnez Allow Launching.

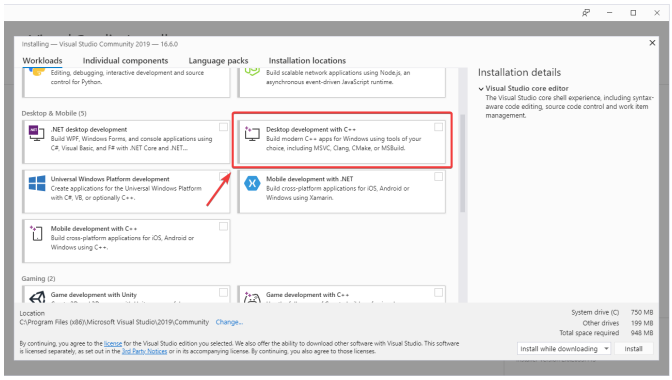


Note : Installing desktop tools and rebooting will create a folder on the desktop called `YYYY WPILib Tools`, where `YYYY` is the current year. Desktop tool shortcuts are not available on Linux and macOS.

3.4.6 Installation C++ supplémentaire pour la simulation

C++ robot simulation requires that a native compiler to be installed. For Windows, this would be Visual Studio 2022 version 17.9 or later (**not** VS Code), macOS requires Xcode 14 or later, and Linux (Ubuntu) requires the `build-essential` package.

Assurez-vous que l'option *Desktop Development with C++* est cochée dans le programme d'installation de Visual Studio pour la prise en charge de la simulation.



3.4.7 Qu'est-ce qui a été installé ?

Le programme d'installation hors ligne installe les composants suivants :

- **Visual Studio Code** - L'IDE pris en charge à partir de 2019 et en montant pour le développement du code robot. L'installateur hors ligne définit une copie distincte de VS Code pour le développement WPILib, même si vous avez déjà VS Code sur votre ordinateur. Cela est possible car certains des paramètres pour la configuration et le bon fonctionnement de la WPILib peuvent briser les flux de travail existants si vous utilisez VS Code pour d'autres projets.
- **C++ Compiler** - Les chaînes de compilation pour la création du code C++ pour le roboRIO
- **Gradle** - La version spécifique de Gradle utilisée pour la création/déploiement du code robot C++ ou Java
- **Java JDK/JRE** - Une version spécifique de Java JDK/JRE qui est utilisée pour construire le code robot Java et pour exécuter l'un des outils basés sur Java (Dashboard, etc.). Celle-ci cohabite avec les installations JDK déjà existantes et n'écrase pas la variable JAVA_HOME
- **WPILib Tools** - SmartDashboard, Shuffleboard, RobotBuilder, OutlineViewer, PathWeaver, Glass, SysId, Data Log Tool, roboRIO Team Number Setter, AdvantageScope
- **WPILib Dependencies** - OpenCV, etc.
- **VS Code Extensions** - WPILib and Java/C++/Python extensions for robot code development in VS Code
- **Documentation** - Offline copies of this frc-docs documentation and Java/C++/Python APIs

3.4.8 Désinstallation

WPILib est conçu pour être installé dans différents dossiers pendant différentes années, de sorte qu'il n'est pas nécessaire de désinstaller une version précédente avant d'installer le WPILib de cette année courante. Cependant, les instructions suivantes peuvent être utilisées pour désinstaller WPILib si vous le souhaitez.

Pour Windows

1. Delete the appropriate wpilib folder (`c:\Users\Public\wpilib\YYYY` where YYYY is the year to uninstall)
2. Supprimez les icônes du bureau dans `C:\Users\Public\Public Desktop`

Pour macOS

1. Delete the appropriate wpilib folder (`~/wpilib/YYYY` where YYYY is the year to uninstall)

Pour Linux

1. Delete the appropriate wpilib folder (`~/wpilib/YYYY` where YYYY is the year to uninstall). eg `rm -rf ~/wpilib/YYYY`

3.4.9 Dépannage

En cas de défaillance de l'installateur, signalez le problème dans le dépôt de l'installateur. Un lien est disponible ici. L'installateur doit signaler un message relatif à la cause de l'erreur, s'il vous plaît l'inclure dans la description de votre problème.

3.5 Guide d'installation de Python

Ce guide est pour les équipes qui utilisent Python. Pour Java et C++ , allez vers *Guide d'installation WPILib*. Pour LabVIEW allez vers *Installation de LabVIEW pour FRC (LabVIEW uniquement)*.

3.5.1 Pré-requis

Vous devez installer une version de Python supportée par votre système d'exploitation. Présentement, Python 3.8/3.9/3.10/3.11/3.12 sont supportés, mais seul Python 3.12 est disponible pour le roboRIO.

Système d'exploitation et Architectures supportés :

— Windows 10 & 11, 64 bits seulement. Les versions 32 bits et Arm ne sont pas prises en charge
— macOS 12 ou version ultérieure
— Ubuntu 22.04, 64 bits. D'autres distributions Linux avec glibc >= 2.35 peuvent fonctionner, mais ne sont pas supportées

Sur Windows et macOS, nous vous recommandons d'utiliser les installateurs officiels de Python distribués par python.org.

— Python pour Windows
— Python pour macOS

3.5.2 Installation de RobotPy

Une fois que vous avez installé Python, vous pouvez utiliser pip pour installer RobotPy sur votre ordinateur de développement.

Pour Windows

Note : Si vous avez précédemment installé une version de RobotPy antérieure à 2024 ou une version bêta 2024, vous devez d'abord désinstaller RobotPy via `py -m pip uninstall robotpy` avant la mise à niveau.

Avertissement : Sur Windows, le package Visual Studio 2019 redistributable doit être installé.

Exécutez la commande suivante depuis cmd ou Powershell pour installer les packages de base de RobotPy :

```
py -3 -m pip install robotpy
```

Pour effectuer une mise à niveau, vous pouvez exécuter ceci :

```
py -3 -m pip install --upgrade robotpy
```

Si vous n'avez pas de droits administratifs sur votre ordinateur, utilisez soit virtualenv/virtualenvwrapper-win, ou vous pouvez réaliser l'installation dans le répertoire des packages utilisateur :

```
py -3 -m pip install --user robotpy
```

Pour macOS

Note : Si vous avez précédemment installé une version de RobotPy antérieure à 2024 ou une version bêta 2024, vous devez d'abord désinstaller robotpy via `python3 -m pip uninstall robotpy` avant la mise à niveau.

Sur un système macOS avec pip installé, exécutez simplement la commande suivante depuis l'application Terminal (peut nécessiter des droits d'administrateur) :

```
python3 -m pip install robotpy
```

Pour effectuer une mise à niveau, vous pouvez exécuter ceci :

```
python3 -m pip install --upgrade robotpy
```

Si vous n'avez pas de droits administratifs sur votre ordinateur, utilisez soit virtualenv/virtualenvwrapper, ou vous pouvez réaliser l'installation dans le répertoire des packages utilisateur :

```
python3 -m pip install --user robotpy
```

Pour Linux

Note : Si vous avez précédemment installé une version de RobotPy antérieure à 2024 ou une version bêta 2024, vous devez d'abord désinstaller robotpy via `python3 -m pip uninstall robotpy` avant la mise à niveau.

RobotPy distribue des *wheels* binaires *manylinux* sur PyPI. Cependant, l'installation de celles-ci nécessite une distribution avec glibc 2.35 ou plus récente, et un installateur qui implémente **PEP 600**, tel que pip 20.3 ou plus récent. Vous pouvez vérifier votre version de pip avec la commande suivante :

```
python3 -m pip --version
```

Si vous avez besoin de mettre à niveau votre version de pip, il est fortement recommandé d'utiliser un environnement virtuel.

Si vous avez une version compatible de pip, vous pouvez simplement exécuter :

```
python3 -m pip install robotpy
```

Pour effectuer une mise à niveau, vous pouvez exécuter ceci :

```
python3 -m pip install --upgrade robotpy
```

Si vous parvenez à installer les packages et obtenez l'erreur suivante ou quelque chose de similaire, votre système n'est probablement pas compatible avec RobotPy :

```
OSError: /usr/lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3.4.22' not found␣
↪(required by /usr/local/lib/python3.7/dist-packages/wpiutil/lib/libwpiutil.so)
```

Linux sur coprocesseur ARM

Nous publions des *wheels* préconstruites sur artifactory, qui peuvent être téléchargées en donnant l'option `--extra-index-url` à pip :

```
python3 -m pip install --extra-index-url=https://wpilib.jfrog.io/artifactory/api/pypi/
↪wpilib-python-release-2024/simple robotpy
```

Installation à partir des sources

Alternatively, if you have a C++20 compiler installed, you may be able to use pip to install RobotPy from source.

Avertissement : Cela peut prendre beaucoup de temps à installer !

Avertissement : Mélanger nos *wheels* préconstruits avec des installations à partir de la source peut provoquer des erreurs d'exécution. Cela est dû à une incompatibilité interne de l'ABI entre les versions de compilateurs.

Nos *wheels* ARM sont construites pour Debian 11 avec GCC 10.

Si vous avez besoin de compiler avec une version spécifique du compilateur, vous pouvez le spécifier en utilisant les variables d'environnement `CC` et `CXX` :

```
export CC=gcc-12 CXX=g++-12
```

3.5.3 Télécharger RobotPy pour le roboRIO

Après avoir installé le projet `robotpy` sur votre ordinateur, diverses commandes sont disponibles et peuvent être exécutées par invite de commande via le module `robotpy`.

Voir aussi :

Documentation pour les sous-commandes de robotpy

Si vous avez déjà un projet de robot utilisant RobotPy, vous pouvez utiliser celui-ci pour télécharger les éléments nécessaires à l'exécution de ce dernier sur le roboRIO. Si vous n'avez pas de projet, l'exécution de cette commande dans un répertoire vide initialisera un nouveau projet de robot :

Pour Windows

```
py -3 -m robotpy init
```

Pour macOS

```
python3 -m robotpy init
```

Pour Linux

```
python3 -m robotpy init
```

Cela créera un fichier `robot.py` et `pyproject.toml`. Le fichier `pyproject.toml` doit être personnalisé et détaille les exigences nécessaires à l'exécution de votre code de robot, entre autres choses.

Voir aussi :

Le fichier `pyproject.toml` créé par défaut ne contient que la version de RobotPy installée sur votre ordinateur. Si vous souhaitez activer les packages vendeurs ou installer d'autres packages Python depuis PyPI, consultez notre *documentation pyproject.toml*

Ensuite, exécutez la sous-commande `robotpy sync`, qui :

— Télécharge Python compilé pour le roboRIO

— Télécharge les packages Python compatibles au roboRIO comme spécifié dans votre fichier `pyproject.toml`
— Installe les packages spécifiés dans votre `pyproject.toml` dans votre environnement local

---

Note : Si vous n'utilisez pas un environnement virtuel et n'avez pas les privilèges d'administrateur, la commande `robotpy sync` accepte l'argument `--user` pour installer dans le répertoire des packages utilisateur.

Pour Windows

```
py -3 -m robotpy sync
```

Pour macOS

```
python3 -m robotpy sync
```

Pour Linux

```
python3 -m robotpy sync
```

Lorsque vous déployez votre code sur le roboRIO, la sous-commande *deploy* installera automatiquement Python (si nécessaire) et les exigences de votre projet de robot sur le roboRIO dans le cadre du processus de déploiement.

3.6 Étapes suivantes

Toutes nos félicitations ! Vous avez terminé l'étape 2 et vous devriez maintenant disposer d'un environnement de développement logiciel fonctionnel ! L'étape 3 de ce didacticiel couvre la mise à jour du matériel afin que vous puissiez le programmer, tandis que l'étape 4 présente la programmation d'un robot dans l'environnement de développement intégré (IDE) VS Code. Pour plus d'informations, vous pouvez consulter *la section VS Code* pour vous familiariser avec l'EDI.

Les articles spécifiques que nous conseillons de lire sont :
— *Les bases de Visual Studio Code*
— *Commandes WPILib sous Visual Studio Code*
— *Créer le programme du robot*
— *Compiler et télécharger le code du robot*
— *Librairies tierces*

De plus, vous devrez peut-être effectuer une configuration supplémentaire applicable au robot de votre équipe. Veuillez utiliser la fonction de recherche pour trouver la documentation nécessaire.

Note : Il est important que les équipes utilisant les contrôleurs de moteur CAN fournis par des fournisseurs indépendants consultent l'article *Installation des librairies des fournisseurs* car des étapes supplémentaires sont nécessaires pour programmer ces composants.

4 Étape 3 : Préparation de votre robot

4.1 Imaging your roboRIO 2

Note : The imaging instructions for the NI roboRIO 1.0 are *here*.

The NI roboRIO 2.0 boots from a microSD card configured with an appropriate boot image containing the NI Linux Real-Time OS, drivers, and libraries specific to FRC. The microSD card must be imaged with a laptop and an SD burner application per the instructions on this page.

Important : Imaging the roboRIO 2 directly with the roboRIO Imaging Tool is not supported.

4.1.1 microSD Requirements

The NI roboRIO 2.0 supports all microSD cards. It is recommended to use a card with 2GB or more of capacity.

4.1.2 Operation Tips

The NI roboRIO 2.0 requires a fully inserted microSD card containing a valid image in order to boot and operate as intended.

If the microSD card is removed while powered, the roboRIO will hang. Once the microSD card is replaced, the roboRIO will need to be restarted using the reset button, or be power cycled.

No damage will result from microSD card removal or insertion while powered, but best practice is to perform these operations while unpowered.

Avertissement : Before imaging your roboRIO, you must have completed installation of the *FRC Game Tools*. You also must have the roboRIO power properly wired to the CTRE Power Distribution Panel or REV Power Distribution Hub. Make sure the power wires to

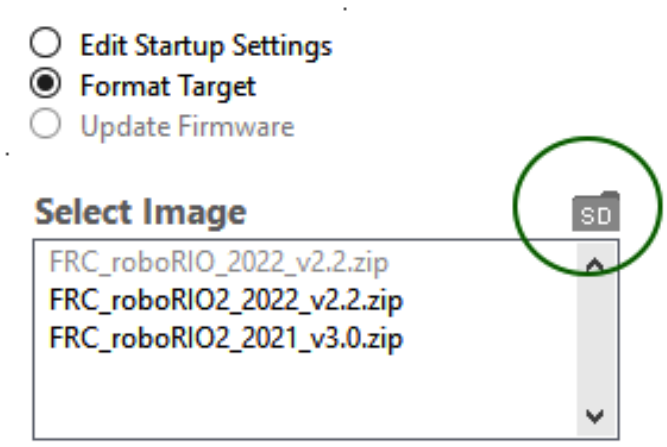
the roboRIO are secure and that the connector is secure firmly to the roboRIO (4 total screws to check).

4.1.3 Imaging Directly to the microSD Card

The image will be transferred to the microSD card using a specialized writing utility, sometimes called a burner. Several utilities are listed below, but most tools that can write arbitrary images for booting a Raspberry Pi or similar dev boards will also produce a bootable SD card for roboRIO 2.0.

Supported image files are named `FRC_roboRIO2_YEAR_VERSION.img.zip`. You can locate them by clicking the SD button in the roboRIO Imaging tool and then navigating to the `SD Images` folder. It is generally best to use the latest version of the image.

If using a non Windows OS you will need to copy this image file to that computer.



A microSD to USB dongle works well for writing to microSD cards.

Note : Raspberry Pi images will not boot on a roboRIO because the OS and drivers are incompatible. Similarly, a roboRIO image is not compatible with Raspberry Pi controller boards.

Writing the image with balenaEtcher

— Download and install balenaEtcher.
— Launch
— *Flash from file* -> locate the image file you want to copy to the microSD card
— *Select target* -> select the destination microSD device
— Press *Flash*

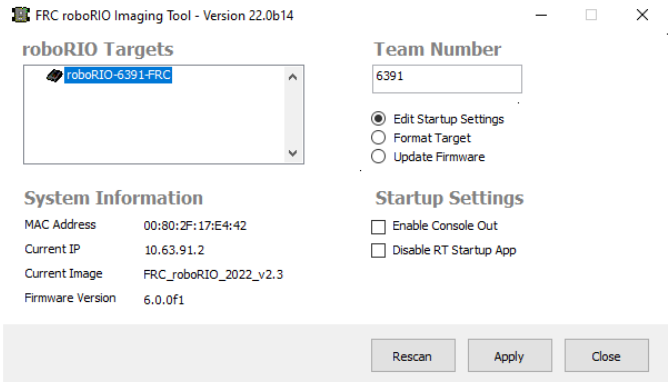
Writing the image with Raspberry Pi Imager

— Download and install from Raspberry Pi Imager.
— Launch
— *Choose OS* -> *Use Custom* -> select the image file you want to copy to the microSD card
— *Choose Storage* -> select the destination microSD device
— Press *Write*

Avertissement : After writing the image, Windows may prompt to format the drive. Do not reformat, or else you will need to write the image again.

Setting the roboRIO Team Number

The image writing process above does not set a team number. To fix this teams will need to insert the microSD card in the robRIO and connect to the robot. With the roboRIO Imaging Tool go to *Edit Startup Settings*. Next, fill out the *Team Number* box and hit *Apply*.



4.2 Imaging your roboRIO 1

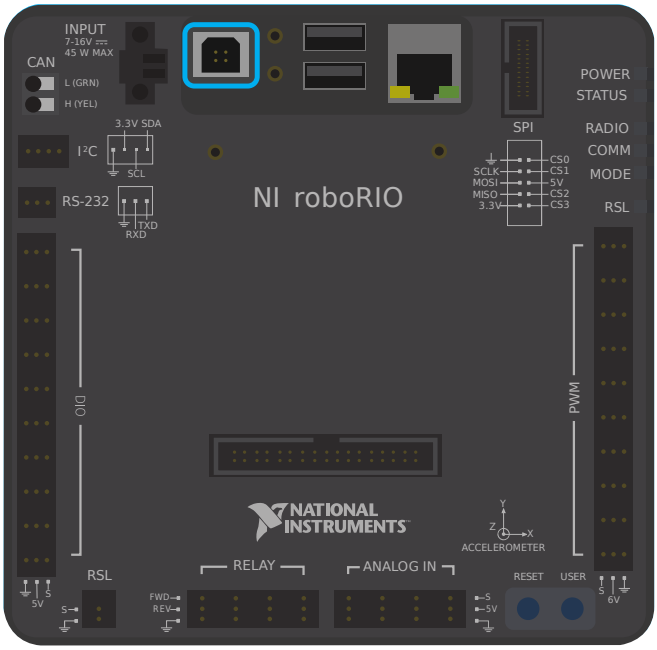
Avertissement : Avant d'installer l'image de votre roboRIO, vous devez avoir terminé l'installation des outils de jeu *FRC Game Tools*. Vous devez également avoir l'alimentation du roboRIO solidement câblé au Panneau de distribution de puissance. Assurez-vous que les fils électriques du roboRIO sont sécurisés et que le connecteur est solidement attaché au roboRIO (4 vis au total à vérifier).

Note : The roboRIO 2 uses different imaging instructions. The imaging instructions for the NI roboRIO 2.0 are *here*.

4.2.1 Configuration du roboRIO

L'outil d'installation d'image du roboRIO sera utilisé pour installer une image dans votre roboRIO à l'aide de la version la plus à jour du logiciel.

Connexion USB



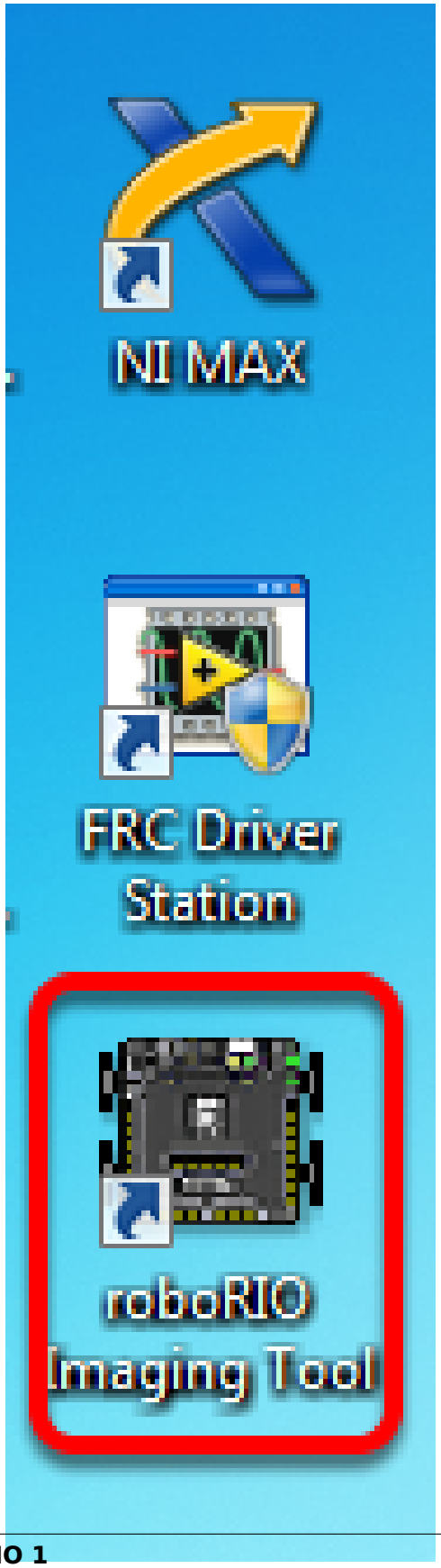
Connectez un câble USB du roboRIO au PC. Cela nécessite un câble USB type A mâle (extrémité de PC standard) à Type B male (carré avec 2 coins coupés), le plus souvent trouvé comme câble USB d'imprimante.

Note : L'installation de l'image du roboRIO devrait seulement être faite par l'intermédiaire de la connexion USB. Il n'est pas recommandé d'utiliser la connexion Ethernet pour créer l'image.

Installation des pilotes (Drivers)

Les pilotes devraient s'installer automatiquement. Si vous voyez un pop-up « New Device » en bas à droite de l'écran, attendez que l'installation du pilote soit terminée avant de continuer.

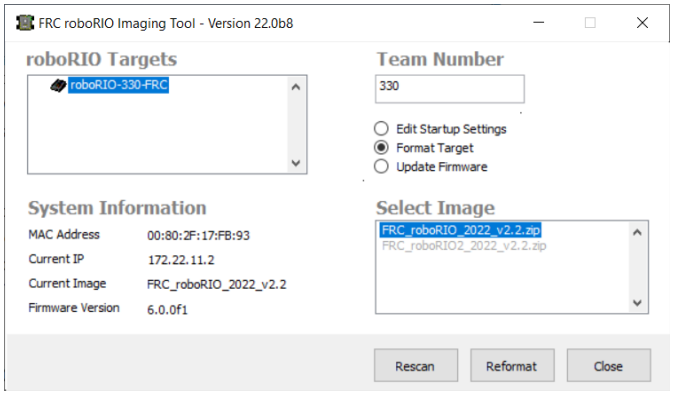
4.2.2 Démarrage de l'outil d'installation d'Image



L'outil d'installation d'image roboRIO imaging tool et la dernière image sont installés avec l'outil de jeu NI FRC® Game Tools. Lancez l'outil d'installation d'image en cliquant deux fois sur le raccourci sur le bureau. Si vous avez des difficultés à installer l'image de votre roboRIO, vous devrez peut-être essayer de cliquer à droite sur l'icône et de sélectionner Exécuter en tant qu'administrateur à la place.

Note : The roboRIO imaging tool is also located at `C:\Program Files (x86)\National Instruments\LabVIEW 2023\project\roboRIO Tool`

4.2.3 Outil d'installation d'image du roboRIO

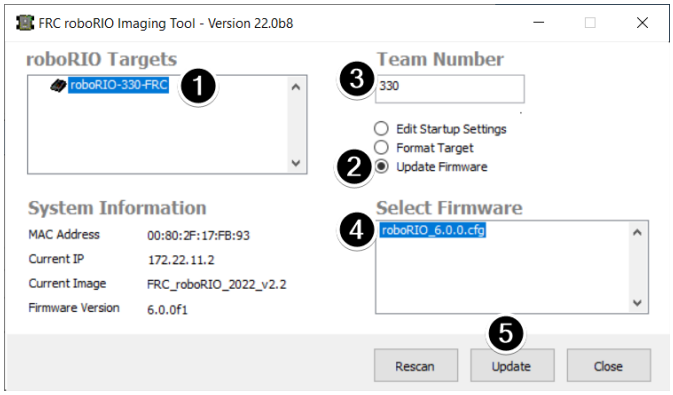


Après le lancement, l'outil d'installation d'image du roboRIO va scanner pour trouver des roboRIO disponibles et indiquer ceux qui sont trouvés en haut à gauche de l'écran. En bas et à gauche de l'écran, on peut visualiser l'information et les paramètres pour le roboRIO actuellement choisi. Le volet de droite contient les contrôles pour modifier les paramètres du roboRIO :

- **Edit Startup Settings** - Cette option est utilisée pour quand vous voulez configurer les paramètres de démarrage du roboRIO (les paramètres dans le volet droit), sans installer l'image au roboRIO.
- **Format Target** - Cette option est utilisée quand vous voulez charger une nouvelle image sur le roboRIO (où de reflasher l'image existante). C'est l'option la plus courante.
- **Update Firmware** - Cette option est utilisé pour mettre à jour le micrologiciel du roboRIO. Pour cette saison, il faut le micrologiciel de roboRIO version 5.0 ou supérieure.

Mettre à jour du micrologiciel

Avertissement : It is only necessary to update the firmware on a brand new roboRIO. It is not recommended to update the firmware unless it doesn't meet the conditions below.



Le firmware du roboRIO doit être au moins v5.0 pour travailler avec l'image 2019 ou plus. Si votre roboRIO est pourvu la version 5.0 au moins (comme indiqué en bas à gauche de l'outil d'installation d'image), vous n'avez pas besoin de faire de mise à jour.

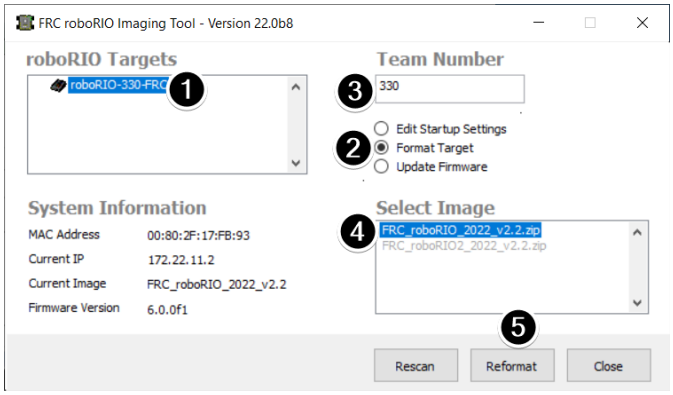
Note : roboRIO firmware has had different version numbering schemes over the years. It isn't necessary to update the firmware if it has version 5, 6, 8, 22.5, 23.5 or variations of those version numbers (e.g. 8.8.0f0 is a variation of 8). The firmware is only utilized in *safe mode*, it is not used in normal operations.

Pour mettre à jour le micrologiciel du roboRIO :

1. Assurez-vous que votre roboRIO est selectionné dans volet supérieur gauche.
2. Select *Update Firmware* in the top right pane
3. Enter a team number in the *Team Number* box
4. Selectionnez le dernier fichier de micrologiciel en bas à droite
5. Click the *Update* button

4.2.4 Installation de l'image dans le roboRIO

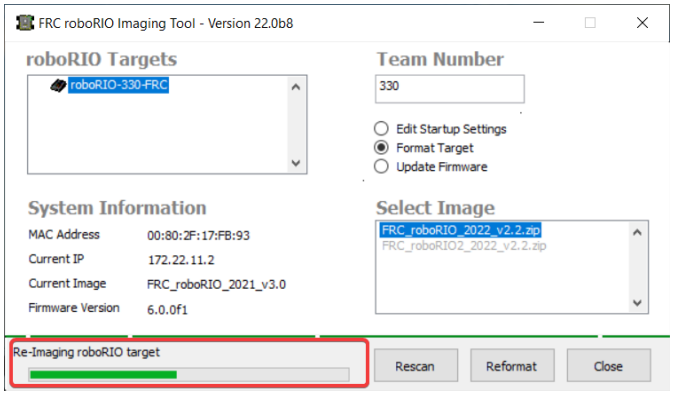
Avertissement : The roboRIO image is different then the firmware, and must be updated yearly.



Note : The available image versions will not show until you select *Format Target* per step 2 below.

1. Assurez-vous que le roboRIO est selectionné dans le volet supérieur gauche
2. Select *Format Target* in the right pane
3. Enter a team number in the *Team Number* box
4. Selectionnez la dernière version d'image dans la boîte
5. Click *Reformat* to begin the imaging process.

4.2.5 Processus d'installation de l'image



Le processus d'installation de l'image prendra environ 3-10 minutes. Une barre de progression en bas à gauche de la fenêtre indique le progrès.

4.2.6 Installation de l'image terminée

When the imaging completes you should see the dialog above. Click *Ok*, then click the :guilabel`Close` button at the bottom right to close the imaging tool. Reboot the roboRIO using the Reset button to have the new team number take effect.

4.2.7 Dépannage

Si vous êtes pas capable d'installer l'image de votre roboRIO, les étapes de dépannage sont :

- Sous le compte administrateur, essayez de lancer « roboRIO Imaging Tool » en cliquant avec le bouton droit de la souris sur l'icône du bureau.
- Essayez d'accéder à la page Web du roboRIO avec un navigateur web à `http://172.22.11.2/` et/ou vérifiez que l'adapteur réseau NI apparaît dans votre liste « Network Adapter » (adapteur réseau) dans le panneau de configuration. Sinon, essayez de réinstaller les Outils de Jeu FRC NI ou essayez un PC différent.
- *Désactivation des adapteurs réseau*
- Assurez-vous que votre pare-feu (Firewall) est désactivé.
- Some teams have experienced an issue where imaging fails if the device name of the computer you're using has a special character (e.g. dash -), or number in it, or the name is too long. Try renaming the computer (or using a different PC). On Windows 11, to rename the PC, go to Settings > System > About and click *Rename this PC*
- Essayez de démarrer le roboRIO en « Safe Mode » en appuyant et en maintenant le bouton reset pendant au moins 5 secondes.
- Essayer un câble USB différent
- Essayez un PC différent
- If the status LED is constantly flashing, and imaging in safe mode failed, follow the roboRIO recovery instructions

If the correct roboRIO image version isn't available :

- Ensure you've selected *Format Target*
- If an older version is shown, ensure you've installed the latest *FRC Game Tools*
- If the wrong version still shown after installing Game Tools, *Uninstall Game Tools* and then re-install.

4.3 Programmation de votre Radio

Ce guide vous montrera comment utiliser le logiciel FRC® Radio Configuration Utility pour configurer le pont sans fil (Wireless bridge) de votre robot pour une utilisation en dehors des événements officiels de la FRC.

4.3.1 Pré-requis

L'utilitaire FRC Radio Configuration Utility nécessite des privilèges d'administrateur pour configurer les paramètres réseau de votre radio. Le programme doit demander automatiquement les privilèges nécessaires (ceci peut nécessiter un mot de passe s'il est exécuté à partir d'un compte non administrateur), mais si vous rencontrez des problèmes, essayez de l'exécuter à partir d'un compte administrateur.

Téléchargez la toute dernière version du logiciel d'installation de l'utilitaire FRC Radio Configuration Utility à partir des liens suivants :

FRC Radio Configuration 24.0.1

FRC Radio Configuration 24.0.1 Israel Version

Note : La version _IL est destinée aux équipes israéliennes et contient une version du firmware OM5PAC avec des canaux restreints pour une utilisation en Israël.

Avant de procéder à l'installation du logiciel :

1. *Désactivez toutes les autres cartes réseau*
2. Branchez directement à partir de votre ordinateur dans le port Ethernet du pont sans fil le plus proche de la prise d'alimentation. Assurez-vous qu'aucun autre appareil n'est connecté à votre ordinateur via le port Ethernet. Si vous alimentez la radio via le port PoE, branchez un câble Ethernet du PC sur le côté de la prise de l'adaptateur PoE (où le roboRIO se brancherait). Si vous rencontrez des problèmes de configuration en vous connectant via l'adaptateur PoE, vous pouvez essayer de connecter le PC à l'autre port de la radio.

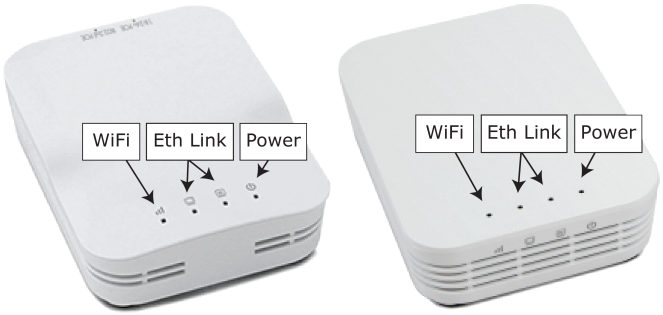
Avertissement : Le OM5P-AN et AC utilisent la même prise d'alimentation que le D-Link DAP1522, cependant ce sont des radios de 12V. Branchez la radio aux bornes 12V 2A sur le module VRM (la broche centrale étant de signe positif).

4.3.2 Notes d'application

Par défaut, l'utilitaire FRC Radio Configuration Utility programmera la radio pour appliquer la limite de bande passante de 4 Mbps sur le trafic sortant de la radio via l'interface sans fil. Dans le configuration maison (mode AP), ce nombre de 4Mbps signifie l'utilisation de la totalité de la bande passante, et non une limite par client. Cela veut dire que la vidéo en streaming vers plusieurs clients n'est pas recommandée.

L'utilitaire a été testé sur Windows 7, 8 et 10. Il peut fonctionner sur d'autres systèmes d'exploitation, mais n'y a pas été testé.

Configuration programmée



L'utilitaire FRC Radio Configuration Utility programme un certain nombre de paramètres de configuration dans la radio lors de son exécution. Ces paramètres s'appliquent à la radio dans tous les modes (y compris lors d'événements). Il s'agit notamment de :

— Définir une IP statique de `10.TE.AM.1`
— Définir une adresse IP alternative du côté relié par fil de `192.168.1.1` pour la programmation future
— Relier des ports câblés afin qu'ils puissent être utilisés de façon interchangeable
— La DEL de configuration notée dans le graphique ci-dessus.
— Limiter la bande passante à 4 Mo/s du côté sortie de l'interface sans fil (ceci peut être désactivé pour une utilisation à la maison)
— La qualité de service ou QS régularise la hiérarchisation interne des paquets (affecte la mémoire tampon interne et les paquets à éliminer si la limite de bande passante est atteinte). Les règles appliquées sont les suivantes :

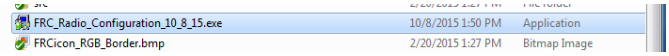
  - Contrôle et état du robot (UDP `1110`, `1115`, `1150`)
  - Robot TCP & *NetworkTables* (TCP `1735`, `1740`)
  - En vrac (Tout autre trafic). (désactivé si la limite de largeur de bande (BW) est désactivée)
- *DHCP* server enabled. Serves out :
  - `10.TE.AM.11` - `10.TE.AM.111` du côté relié par fil
  - `10.TE.AM.138` - `10.TE.AM.237` du côté sans-fil
  - Masque Subnet de `255.255.255.0`
  - Adresse de diffusion `10.TE.AM.255`
- Serveur DNS activé. L'adresse IP du serveur DNS et le suffixe de domaine (`.lan`) sont servis dans le cadre du DHCP.

À la maison seulement :

- SSID peut avoir un « Nom de robot » joint au numéro d'équipe pour distinguer plusieurs réseaux.
- L'option Pare-feu peut être activée pour imiter les règles du champ pare-feu (pour les ports ouverts veuillez consulter le Manuel de jeu)

Avertissement : Il n'est pas possible de modifier la configuration manuellement.

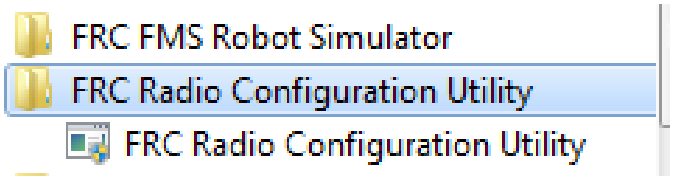
4.3.3 Installez le logiciel



Double-cliquez sur `FRC_Radio_Configuration_VERSION.exe` pour lancer le programme d'installation. Suivez les instructions pour terminer l'installation.

Durant l'installation, on proposera d'installer Npcap si il n'est pas déjà présent. L'installateur pour Npcap contient un certain nombre de cases à cocher pour configurer l'installation. Vous devez laisser les valeurs par défaut pour chacune des options.

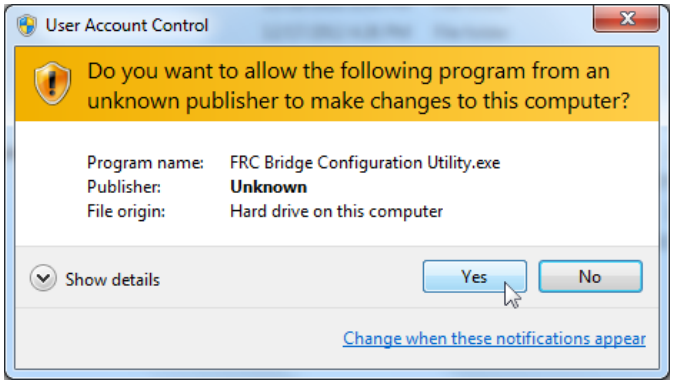
4.3.4 Lancer le logiciel



Utilisez le menu Démarrer ou le raccourci de bureau pour lancer le programme.

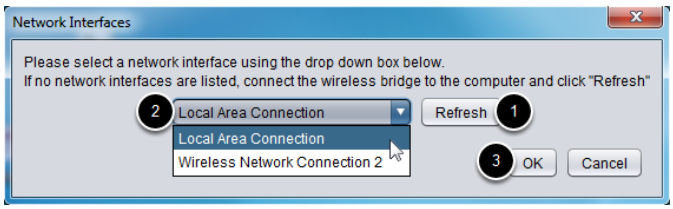
Note : Si vous avez besoin de localiser le programme, il est installé dans `C:\Program Files (x86)\FRC Radio Configuration Utility`. Pour les machines 32 bits, le chemin est `C:\Program Files\FRC Radio Configuration Utility`

4.3.5 Autoriser le programme à apporter des modifications, si vous y êtes invité



Une invite peut s'afficher pour autoriser l'utilitaire de configuration à apporter des modifications à l'ordinateur. Cliquez sur *Yes* si l'invite s'affiche.

4.3.6 Sélectionner l'interface réseau



Utilisez la fenêtre contextuelle pour sélectionner l'interface ethernet que l'utilitaire de configuration utilisera pour communiquer avec la passerelle sans fil. Sur les ordinateurs Windows, les interfaces ethernet sont généralement nommées « Local Area Connection ». L'utilitaire de configuration ne peut pas programmer une passerelle via une connexion sans fil.

1. Si aucune interface Ethernet n'est répertoriée, cliquez sur *Refresh* pour effectuer une nouvelle recherche des interfaces disponibles.
2. Sélectionnez l'interface que vous souhaitez utiliser dans la liste déroulante.
3. Cliquez sur *OK*.

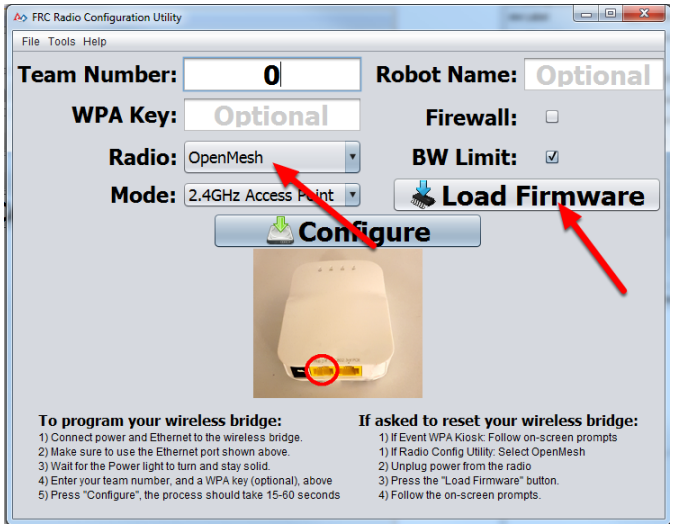
4.3.7 Note concernant le Firmware Open Mesh

Pour que l'utilitaire de configuration radio FRC puisse programmer la radio OM5P-AN et OM5P-AC, la radio doit fonctionner sous une version FRC spécifique du micrologiciel OpenWRT.

Si vous n'avez pas besoin de mettre à jour ou de réinstaller le firmware, sautez l'étape suivante.

Avertissement : Radios used in 2019-2023 **do not** need to be updated before configuring, the 2024 tool uses the same 2019 firmware.

4.3.8 Chargement du micrologiciel FRC pour ouvrir la radio Mesh



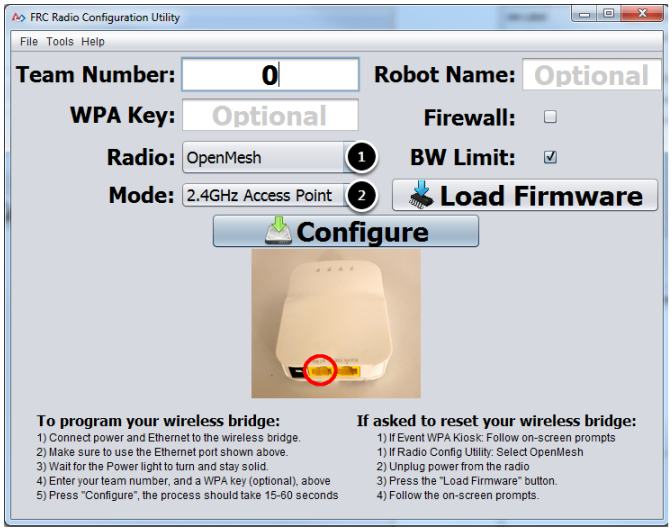
Si vous devez charger le firmware FRC (ou réinitialiser la radio), vous pouvez le faire à l'aide de l'utilitaire de configuration radio FRC.

1. Suivez les instructions ci-dessus pour installer le logiciel, lancer le programme et sélectionner l'interface Ethernet.
2. Assurez-vous que la radio Open Mesh est sélectionnée dans la liste déroulante Radio.
3. Assurez-vous que la radio est connectée au PC via Ethernet.
4. Débranchez l'alimentation de la radio. (Si vous utilisez un câble PoE, il faudra également débrancher le Ethernet sur le PC, ce qui est correct)
5. Appuyez sur le bouton Load Firmware
6. Lorsque vous y êtes invité, branchez l'alimentation de la radio. Le logiciel doit détecter la radio, charger le firmware et vous informer, une fois terminé.

Avertissement : Si vous voyez une erreur concernant le nom NPF, essayez de désactiver tous les adaptateurs autres que celui utilisé pour programmer la radio. Si un seul adaptateur est trouvé, l'outil doit essayer de l'utiliser. Consultez les étapes dans *Désactivation des cartes réseau* pour plus d'informations.

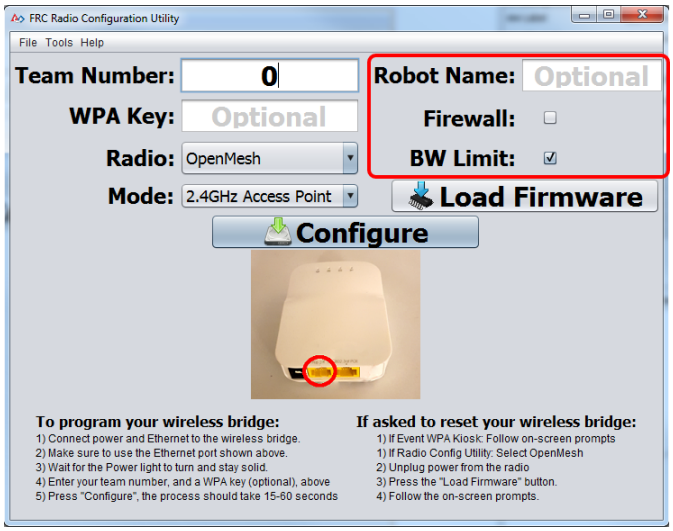
Teams may also see this error with Operating Systems configured for languages other than US English. If you experience issues loading firmware or programming on a foreign language OS, try using an English OS, such as on the KOP provided PC or setting the Locale setting to « en_us » as described on this page.

4.3.9 Sélectionnez la radio et le mode de fonctionnement



1. Sélectionnez la radio que vous configurez à l'aide de la liste déroulante.
2. Sélectionnez le mode d'opération à configurer. Dans la plupart des cas, la sélection par défaut du point d'accès 2.4 GHz sera suffisante. Si vos ordinateurs le supporte, le mode AP 5GHz est recommandé, car 5GHz est moins encombré dans de nombreux environnements.

4.3.10 Sélectionnez les options



Les valeurs par défaut des options ont été sélectionnées pour correspondre au cas d'utilisation de la plupart des équipes, cependant, vous pouvez personnaliser ces options à votre scénario spécifique :

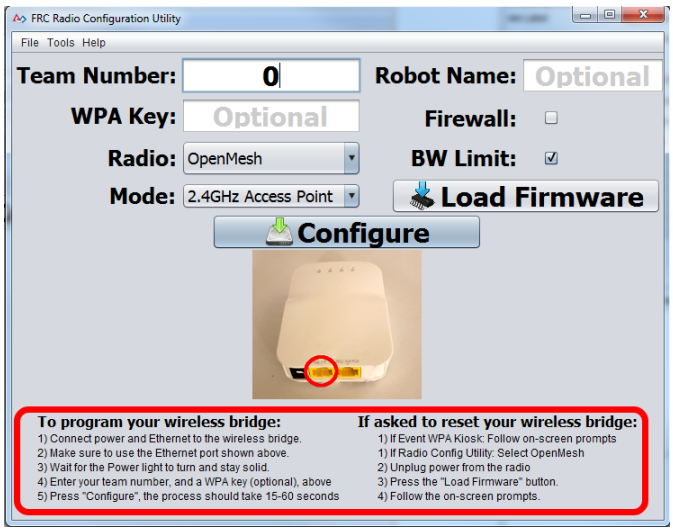
1. **Robot Name :** Il s'agit d'une chaîne de caractères qui est ajoutée au SSID utilisé par la radio. Cela vous permet d'avoir plusieurs réseaux avec le même numéro d'équipe et de toujours pouvoir les distinguer.

2. **Firewall :** Si cette case est cochée, le pare-feu radio sera configuré pour tenter d'imiter le comportement de blocage de port du pare-feu présent sur le champ FRC. Pour une liste des ports ouverts, veuillez consulter le manuel du jeu FRC.
3. **BW Limit :** Si cette case est cochée, la radio applique une limite de bande passante de 4 Mbps comme elle le fait lorsqu'elle est programmée lors d'événements. Notez qu'il s'agit d'une limite totale, pas par client, donc la diffusion de vidéo vers plusieurs clients simultanément peut provoquer un comportement indésirable.

Note : Le pare-feu et la limite BW s'appliquent uniquement aux radios Open Mesh. Ces options n'ont aucun effet sur les radios D-Link.

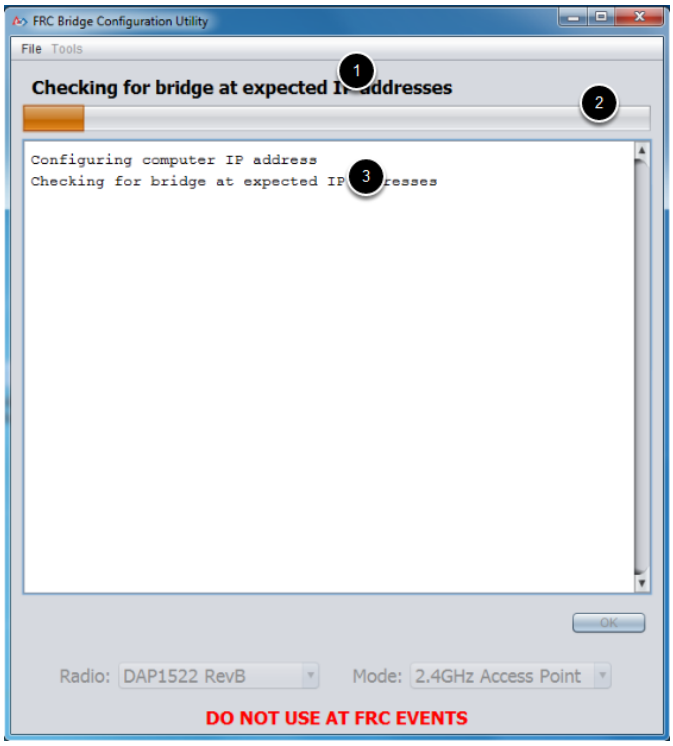
Avertissement : L'option « Firewall » configure la radio pour émuler le pare-feu de terrain. Cela signifie que vous ne pourrez pas déployer du code sans fil avec cette option activée. Ceci est utile pour simuler les ports bloqués qui peuvent exister lors de compétitions.

4.3.11 Démarrage du processus de configuration



Suivez les instructions à l'écran pour préparer votre passerelle sans fil, entrer les paramètres avec lesquels la passerelle sera configurée et démarrer le processus de configuration. Ces instructions à l'écran sont mises à jour pour correspondre au modèle de pont et au mode de fonctionnement choisi.

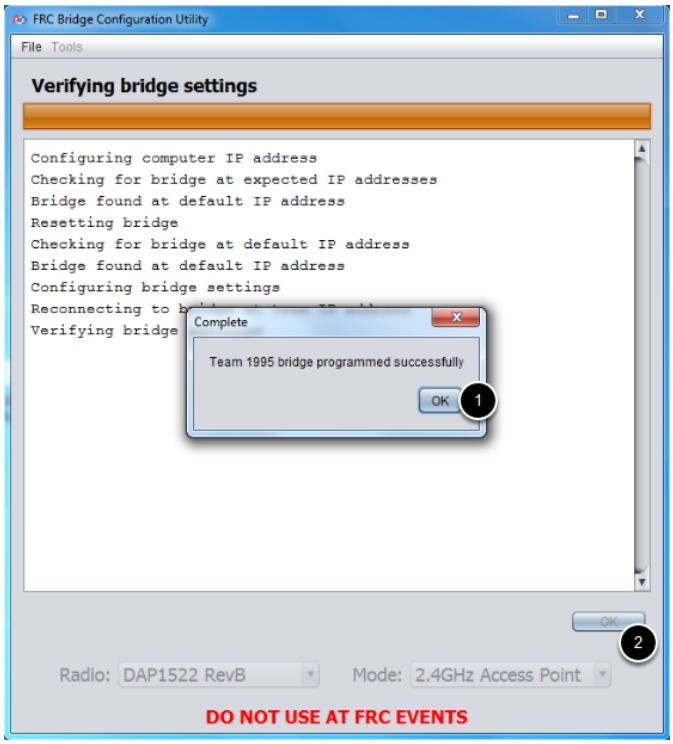
4.3.12 Progression de la configuration



Tout au long du processus de configuration, la fenêtre indique :

1. L'étape en cours d'exécution.
2. La progression globale du processus de configuration.
3. Toutes les étapes exécutées jusqu'à présent.

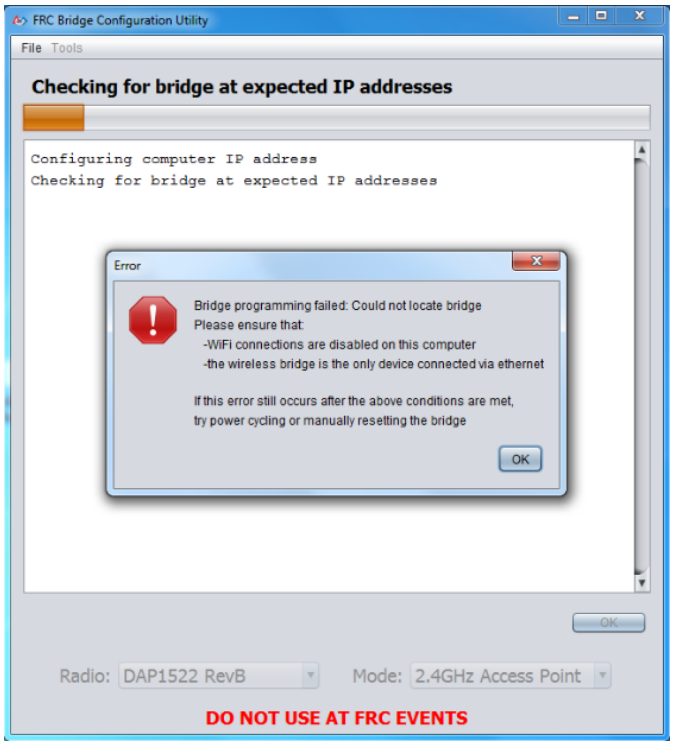
4.3.13 Configuration terminée



Une fois la configuration terminée :

1. Appuyez sur *OK* dans la fenêtre de dialogue.
2. Appuyez sur *OK* dans la fenêtre principale pour revenir à l'écran des paramètres.

4.3.14 Erreurs de configuration



Si une erreur se produit pendant le processus de configuration, suivez les instructions du message d'erreur pour corriger le problème.

4.3.15 Dépannage

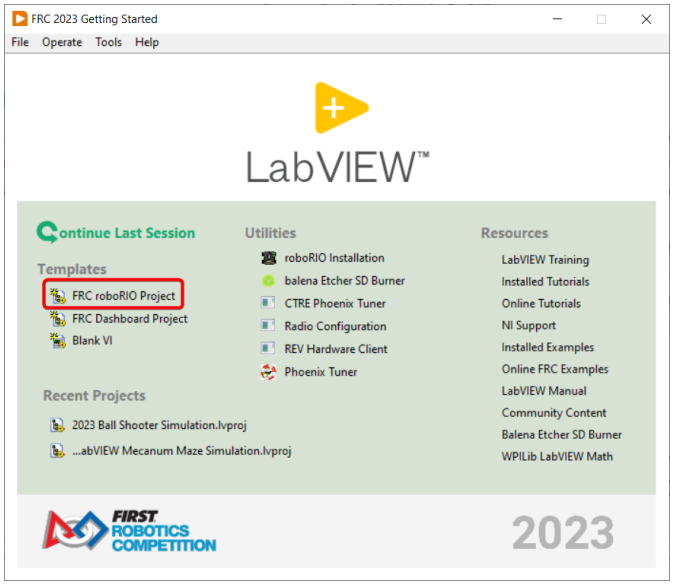
— *Désactiver tous les autres adapteurs réseaux*.
— Assurez-vous d'attendre suffisamment longtemps pour que le voyant d'alimentation reste fixe pendant 10 secondes.
— Assurez-vous que vous disposez de la bonne interface réseau et qu'une seule interface est répertoriée dans la liste déroulante.
— Assurez-vous que votre pare-feu (Firewall) est désactivé.
— Branchez directement depuis votre ordinateur dans le pont sans fil (Wireless bridge) et assurez-vous qu'aucun autre appareil n'est connecté à votre ordinateur via Ethernet.
— Assurez-vous que l'Ethernet est branché sur le port le plus proche de la prise d'alimentation sur le pont sans fil.
— If using an Operating System configured for languages other than US English, try using an English OS, such as on the KOP provided PC or setting the Locale setting to « en_us » as described on this page.
— Due to Unicode incompatibles, non-US Teams may face a configuration failure because of incorrect network interface reading. In that case, change the network adapter name to another name in English and retry.
— Certains utilisateurs ont signalé avoir réussi après avoir installé npcap 1.60. Si cela ne résout pas le problème, il est recommandé de désinstaller npcap et l'outil radio, puis de réinstaller l'outil radio afin de revenir à une configuration connue.
— Si tout le reste échoue, essayez un autre ordinateur.

5 Etape 4 : Programmation de votre robot

5.1 Creating your Test Drivetrain Program (LabVIEW)

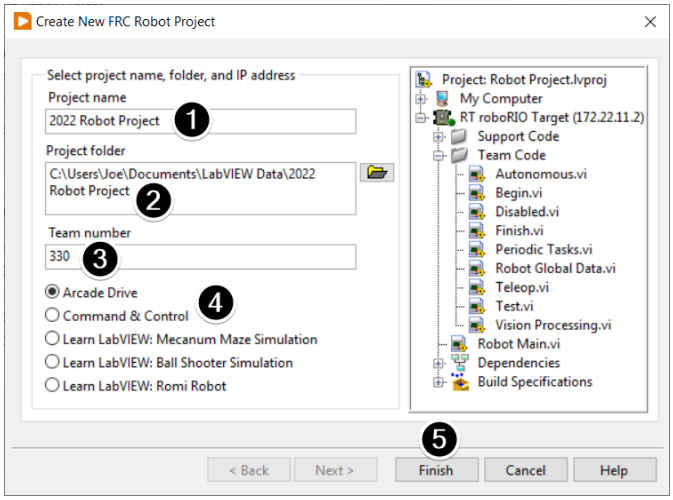
Note : This document covers how to create, build and load a basic FRC® LabVIEW program for a drivetrain onto a roboRIO. Before beginning, make sure that you have installed LabVIEW for FRC and the FRC Game Tools and that you have configured and imaged your roboRIO as described in the *Zero-to-Robot tutorial*.

5.1.1 Création d'un projet



Lancez LabVIEW et cliquez sur le lien Projet de robot FRC roboRIO pour afficher la boîte de dialogue Créer un nouveau projet de robot FRC.

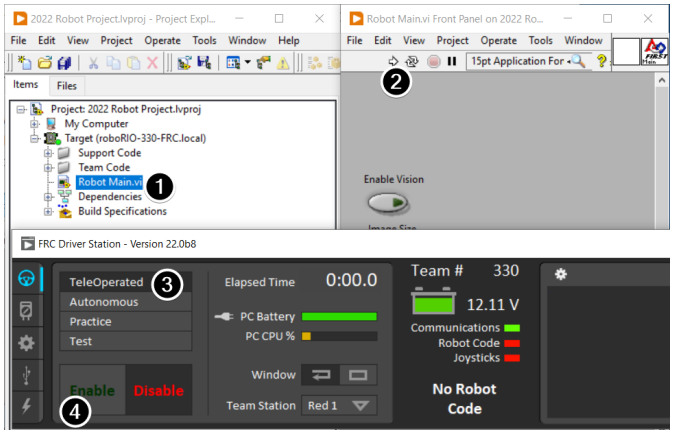
5.1.2 Configuration du projet



Remplissez la boîte de dialogue Create New FRC Project :

1. Choisissez un nom pour votre projet
2. Sélectionnez un dossier dans lequel placer le projet.
3. Entrez votre numéro d'équipe
4. Sélectionnez un type de projet. En cas de doute, sélectionnez *Arcade Drive*.
5. Cliquez sur *Finish*

5.1.3 Exécution du programme

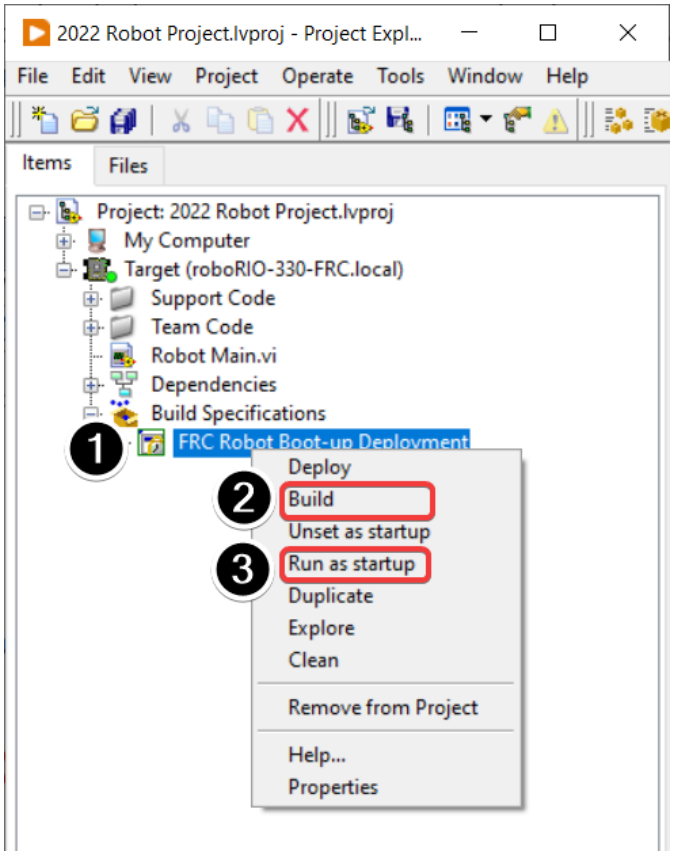


Note : Notez qu'un programme déployé de cette manière ne restera pas dans le roboRIO une fois que celui-ci aura été mis hors tension. Pour déployer un programme qui s'exécute à chaque fois que le roboRIO démarre, suivre l'étape suivante, Déploiement du programme.

1. Dans la fenêtre Explorateur de projets, double-cliquez sur l'instrument virtuel Robot Main.vi pour l'ouvrir.

2. Cliquez sur le bouton Run (Flèche blanche sur le ruban supérieur) du Robot Main VI pour déployer le VI sur le roboRIO. LabVIEW déploie le VI, tous les éléments requis par le VI et les paramètres de la cible dans la mémoire du roboRIO. Si vous êtes invité.e à enregistrer des VIs, cliquez sur Save à toutes les invites.
3. À l'aide de l'application Driver Station, placez le robot en mode Teleop. Pour plus d'informations sur la configuration et l'utilisation de la Driver Station, voir l'article FRC Driver Station Software.
4. Cliquez sur Enable.
5. Activez les joysticks et observez comment le robot réagit.
6. Cliquez sur le bouton Abort du Robot Main VI. Notez que le VI s'arrête. Lorsque vous déployez un programme avec le bouton Run, le programme s'exécute sur le roboRIO, mais vous pouvez manipuler les objets du panneau avant du programme à partir de l'ordinateur hôte dans ce cas-ci votre ordinateur de pilotage.

5.1.4 Déploiement du programme



Pour participer à la compétition, vous devrez déployer un programme dans la mémoire de votre roboRIO. Cela permet au programme d'être présent au roboRIO et de s'exécuter à tous les redémarrages du contrôleur, mais n'offre pas les mêmes fonctionnalités de débogage (panneau frontal, sondes, mise en surbrillance de l'exécution) que l'exécution à partir du panneau frontal. Pour déployer votre programme :

1. Dans l'Explorateur de projets, cliquez sur le + à côté de Build Specifications pour l'étendre.

2. Cliquez avec le bouton droit sur FRC Robot Boot-up Deployment et sélectionnez Build. Attendez que la compilation se termine.
3. Cliquez à nouveau avec le bouton droit sur FRC Robot Boot-Up Deployment et sélectionnez Run as Startup. Si vous recevez une boîte de dialogue de conflit, cliquez sur OK. Cette boîte de dialogue indique simplement qu'il existe actuellement un programme dans le roboRIO qui sera terminé/remplacé.
4. Après un déploiement réussi vous pouvez soit cocher la case près de la fenêtre de déploiement, soit cliquer sur le bouton close lorsque le déploiement se termine.
5. Le roboRIO démarre automatiquement l'exécution du code déployé dans les quelques secondes suivant la fermeture de la boîte de dialogue.

5.2 Création de votre programme test Drivetrain (Java/C++/Python)

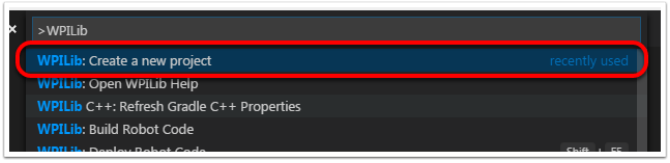
Une fois que tout est installé, nous sommes prêts à créer un programme de robot. WPILib est livré avec plusieurs modèles pour les programmes de robots. L'utilisation de ces modèles est fortement recommandée pour les nouveaux utilisateurs ; cependant, les utilisateurs avancés sont libres d'écrire leur propre code robot à partir de zéro. Cet article explore la création d'un projet à partir de l'un des exemples fournis qui a un certain code déjà écrit pour piloter un robot de base.

— *Création d'un nouveau projet WPILib (Java/C++)*
— *Création d'un nouveau projet WPILib (Python)*

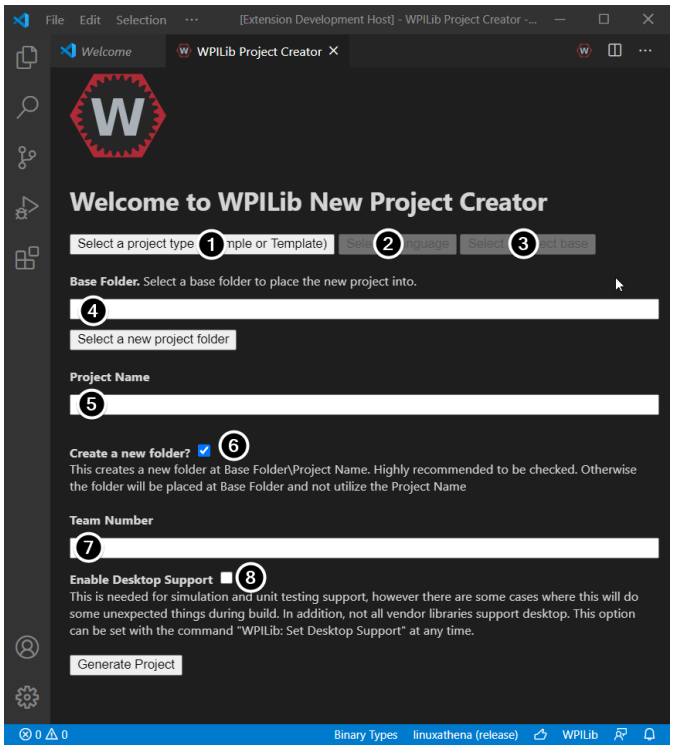
Important : Ce guide comprend des exemples de code qui impliquent le matériel du fournisseur pour la commodité de l'utilisateur. Dans ce document, *PWM* fait référence au contrôleur de moteur inclus dans le KOP. L'onglet CTRE fait référence au contrôleur de moteur Talon FX (moteur Falcon 500), mais l'utilisation est similaire pour TalonSRX et VictorSPX. L'onglet REV fait référence au CAN SPARK MAX contrôlant un moteur sans balais, mais il est similaire pour un moteur à balais. On suppose que l'utilisateur a déjà installé le **:doc :`vendordeps ` requis et configuré le(s) périphérique(s) (mettre à jour le micrologiciel, attribuer des ID CAN, etc.) conformément à la documentation du fabricant (`CTRE <https ://docs. ctr-electronics.com/>`__** REV).

5.2.1 Création d'un nouveau projet WPILib (Java/C++)

Affichez la palette de commandes de Visual Studio Code avec `Ctrl+Shift+P`. Ensuite, tapez « WPILib » dans l'invite. Étant donné que toutes les commandes WPILib commencent par « WPILib », cela affichera la liste des commandes VS Code spécifiques à WPILib. Maintenant, sélectionnez la commande « Créer un nouveau projet » :



Cette opération fera apparaître la fenêtre « New Project Creator Window » :



Les éléments de la fenêtre New Project Creator sont expliqués ci-dessous :

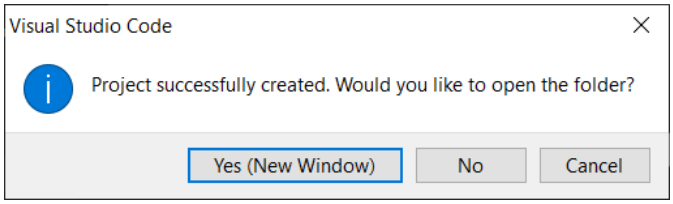
1. **Project Type** : Le genre de projet que nous voulons créer. Pour cet exemple, sélectionnez **Example**
2. **Language** : C'est le langage de programmation (C++ ou Java) qui sera utilisé pour ce projet.
3. **Project Base** : Cette zone est utilisée pour sélectionner la classe de base ou l'exemple à utiliser comme source pour générer le projet. Pour cet exemple, sélectionnez **Getting Started**
4. **Base Folder** : Ce paramètre détermine le dossier dans lequel le projet de robot sera situé.
5. **Project Name** : Le nom du projet de robot. Ce paramètre spécifie également le nom du dossier de projet si la zone Create New Folder est cochée.
6. **Create a New Folder** : Si cette case est cochée, un nouveau dossier sera créé pour contenir le projet dans le dossier précédemment spécifié. Si elle n'est *pas* cochée, le projet sera situé directement dans le dossier précédemment spécifié. Une erreur sera générée si le dossier n'est pas vide et que cette case n'est pas cochée. Un dossier de projet sera créé si la case Create New Folder est cochée.
7. **Team Number** : Numéro d'équipe du projet, qui sera utilisé pour les noms de packages dans le projet et pour localiser le robot au moment du déploiement du code.
8. **Enable Desktop Support** : Permet de faire le test unitaire et la simulation. Bien que WPILib prenne en charge cette fonctionnalité, pour les bibliothèques de logiciels tiers cette prise en charge n'est pas garantie. Si les bibliothèques logicielles ne prennent pas en charge les applications de bureau, votre code peut ne pas compiler ou planter. Cette case doit être laissée non cochée à moins que des tests unitaires ou une simulation ne soient nécessaires et que toutes les bibliothèques de logiciels le prennent en charge. Pour cet exemple, ne cochez pas cette case.

Une fois que tout ce qui précède a été configuré, cliquez sur « Generate Project » et le projet robot sera créé.

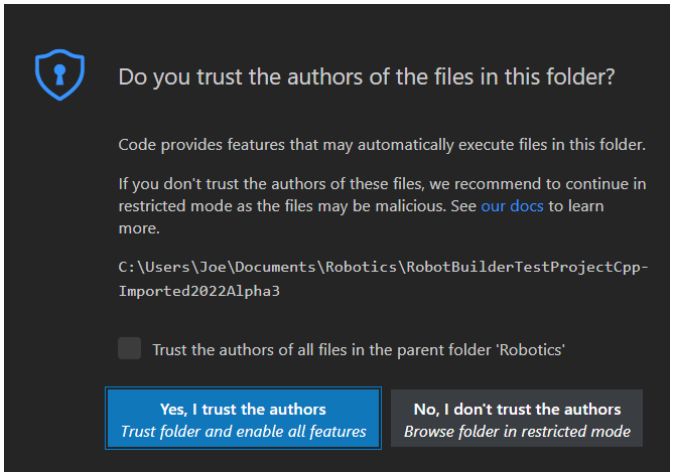
Note : Toutes les erreurs lors de la génération de projet apparaîtront dans le coin inférieur droit de l'écran.

Avertissement : La création de projets sur OneDrive n'est pas prise en charge car la mise en cache de OneDrive interfère avec le système de build. Certaines installations Windows placent par défaut les dossiers Documents et Bureau sur OneDrive.

5.2.2 Ouverture du nouveau projet

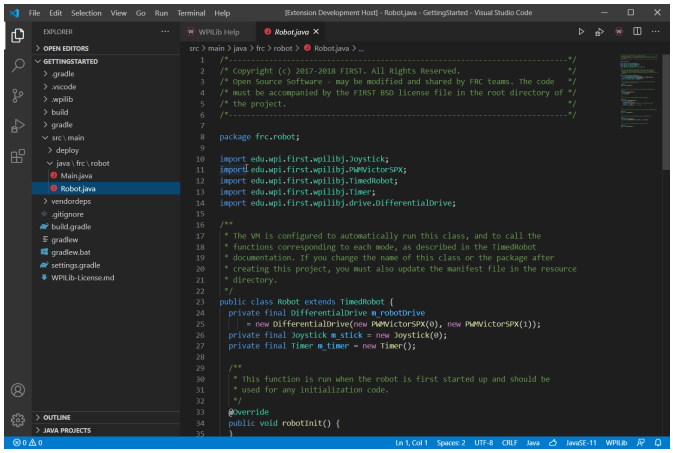


Après avoir réussi à créer votre projet, VS Code vous donnera la possibilité d'ouvrir le projet comme indiqué ci-dessus. Nous pouvons choisir de le faire maintenant ou plus tard en tapant `Ctrl+K` puis `Ctrl+O` (ou simplement `Command+O` sur macOS) et sélectionner le dossier où nous avons enregistré notre projet.



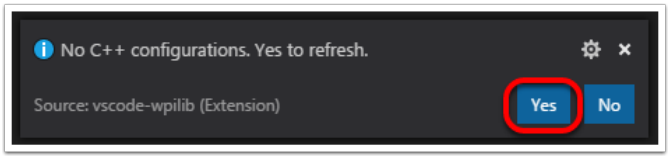
Cliquez sur *Yes I trust the authors*. (Oui, je fais confiance aux auteurs.)

Une fois ouvert, nous pouvons voir la hiérarchie du projet à gauche. En double-cliquant sur un fichier, on ouvre ce fichier dans l'éditeur.



5.2.3 Configurations C++ (C++ uniquement)

Pour les projets C++, il y a une étape de plus pour configurer IntelliSense. Chaque fois que nous ouvrons un projet, nous devrions obtenir un pop-up dans le coin inférieur droit demandant de rafraîchir les configurations C++. Cliquez sur « Yes » pour configurer IntelliSense.



5.2.4 Création d'un nouveau projet WPILib (Python)

Exécuter la commande `robotpy init` initialisera un nouveau projet de robot.

Pour Windows

```
py -3 -m robotpy init
```

Pour macOS

```
python3 -m robotpy init
```

Pour Linux

```
python3 -m robotpy init
```

Cela créera un fichier `robot.py` et `pyproject.toml`, mais n'écrasera pas un fichier existant.

- Le fichier `pyproject.toml` contient les exigences pour votre projet, qui sont téléchargées et installées via la commande `robotpy sync`.
- Le fichier `robot.py` est l'endroit où vous mettrez votre classe Robot.

Voir aussi :

docs/zero-to-robot/step-2/python-setup :Téléchargement de RobotPy pour roboRIO

5.2.5 Exemple de transmission de base

Tout d'abord, voici à quoi peut ressembler un code simple pour une transmission avec des moteurs contrôlés par PWM (tels que SparkMax).

Note : l'exemple Python ci-dessous provient de https://github.com/robotpy/examples/tree/main/GettingStarted

---

JAVA

```
// Copyright (c) FIRST and other WPILib contributors.
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

package edu.wpi.first.wpilibj.examples.gettingstarted;

import edu.wpi.first.util.sendable.SendableRegistry;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.XboxController;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;

/**
 * The VM is configured to automatically run this class, and to call the functions corresponding to
 * each mode, as described in the TimedRobot documentation. If you change the name of this class or
 * the package after creating this project, you must also update the manifest file in the resource
 * directory.
 */
public class Robot extends TimedRobot {
  private final PWMSparkMax m_leftDrive = new PWMSparkMax(0);
  private final PWMSparkMax m_rightDrive = new PWMSparkMax(1);
  private final DifferentialDrive m_robotDrive =
      new DifferentialDrive(m_leftDrive::set, m_rightDrive::set);
  private final XboxController m_controller = new XboxController(0);
  private final Timer m_timer = new Timer();

```

(suite sur la page suivante)

(suite de la page précédente)

```
  public Robot() {
    SendableRegistry.addChild(m_robotDrive, m_leftDrive);
    SendableRegistry.addChild(m_robotDrive, m_rightDrive);
  }

  /**
   * This function is run when the robot is first started up and should be used for any
   * initialization code.
   */
  @Override
  public void robotInit() {
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side instead.
    m_rightDrive.setInverted(true);
  }

  /** This function is run once each time the robot enters autonomous mode. */
  @Override
  public void autonomousInit() {
    m_timer.restart();
  }

  /** This function is called periodically during autonomous. */
  @Override
  public void autonomousPeriodic() {
    // Drive for 2 seconds
    if (m_timer.get() < 2.0) {
      // Drive forwards half speed, make sure to turn input squaring off
      m_robotDrive.arcadeDrive(0.5, 0.0, false);
    } else {
      m_robotDrive.stopMotor(); // stop robot
    }
  }

  /** This function is called once each time the robot enters teleoperated mode. */
  @Override
  public void teleopInit() {}

  /** This function is called periodically during teleoperated mode. */
  @Override
  public void teleopPeriodic() {
    m_robotDrive.arcadeDrive(-m_controller.getLeftY(), -m_controller.getRightX());
  }

  /** This function is called once each time the robot enters test mode. */
  @Override
  public void testInit() {}

  /** This function is called periodically during test mode. */
  @Override
  public void testPeriodic() {}
}
```

C++

```
// Copyright (c) FIRST and other WPILib contributors.
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

#include <frc/TimedRobot.h>
#include <frc/Timer.h>
#include <frc/XboxController.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/motorcontrol/PWMSparkMax.h>

class Robot : public frc::TimedRobot {
 public:
  Robot() {
    wpi::SendableRegistry::AddChild(&m_robotDrive, &m_left);
    wpi::SendableRegistry::AddChild(&m_robotDrive, &m_right);

    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side instead.
    m_right.SetInverted(true);
    m_robotDrive.SetExpiration(100_ms);
    m_timer.Start();
  }

  void AutonomousInit() override { m_timer.Restart(); }

  void AutonomousPeriodic() override {
    // Drive for 2 seconds
    if (m_timer.Get() < 2_s) {
      // Drive forwards half speed, make sure to turn input squaring off
      m_robotDrive.ArcadeDrive(0.5, 0.0, false);
    } else {
      // Stop robot
      m_robotDrive.ArcadeDrive(0.0, 0.0, false);
    }
  }

  void TeleopInit() override {}

  void TeleopPeriodic() override {
    // Drive with arcade style (use right stick to steer)
    m_robotDrive.ArcadeDrive(-m_controller.GetLeftY(),
                             m_controller.GetRightX());
  }

  void TestInit() override {}

  void TestPeriodic() override {}

 private:
  // Robot drive system
  frc::PWMSparkMax m_left{0};
  frc::PWMSparkMax m_right{1};
  frc::DifferentialDrive m_robotDrive{
      [&](double output) { m_left.Set(output); },
```

(suite sur la page suivante)

(suite de la page précédente)

```
      [&](double output) { m_right.Set(output); }};

  frc::XboxController m_controller{0};
  frc::Timer m_timer;
};

#ifndef RUNNING_FRC_TESTS
int main() {
  return frc::StartRobot<Robot>();
}
#endif
```

PYTHON

```
#!/usr/bin/env python3
#
# Copyright (c) FIRST and other WPILib contributors.
# Open Source Software; you can modify and/or share it under the terms of
# the WPILib BSD license file in the root directory of this project.
#

import wpilib
import wpilib.drive


class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        """
        This function is called upon program startup and
        should be used for any initialization code.
        """
        self.leftDrive = wpilib.PWMSparkMax(0)
        self.rightDrive = wpilib.PWMSparkMax(1)
        self.robotDrive = wpilib.drive.DifferentialDrive(
            self.leftDrive, self.rightDrive
        )
        self.controller = wpilib.XboxController(0)
        self.timer = wpilib.Timer()

        # We need to invert one side of the drivetrain so that positive voltages
        # result in both sides moving forward. Depending on how your robot's
        # gearbox is constructed, you might have to invert the left side instead.
        self.rightDrive.setInverted(True)

    def autonomousInit(self):
        """This function is run once each time the robot enters autonomous mode."""
        self.timer.restart()

    def autonomousPeriodic(self):
        """This function is called periodically during autonomous."""

        # Drive for two seconds
        if self.timer.get() < 2.0:
            # Drive forwards half speed, make sure to turn input squaring off
```

(suite sur la page suivante)

(suite de la page précédente)

```
            self.robotDrive.arcadeDrive(0.5, 0, squareInputs=False)
        else:
            self.robotDrive.stopMotor()  # Stop robot

    def teleopInit(self):
        """This function is called once each time the robot enters teleoperated mode."
↪""

    def teleopPeriodic(self):
        """This function is called periodically during teleoperated mode."""
        self.robotDrive.arcadeDrive(
            -self.controller.getLeftY(), -self.controller.getRightX()
        )

    def testInit(self):
        """This function is called once each time the robot enters test mode."""

    def testPeriodic(self):
        """This function is called periodically during test mode."""


if __name__ == "__main__":
    wpilib.run(MyRobot)
```

Examinons maintenant différentes parties du code.

5.2.6 Imports/Includes

PWM

Java

```
import edu.wpi.first.util.sendable.SendableRegistry;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.XboxController;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
```

C++

```
#include <frc/TimedRobot.h>
#include <frc/Timer.h>
#include <frc/XboxController.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/motorcontrol/PWMSparkMax.h>
```

Python

```
import wpilib
import wpilib.drive
```

CTRE

JAVA

```
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import com.ctre.phoenix.motorcontrol.can.WPI_TalonFX;
```

C++

```
#include <frc/Joystick.h>
#include <frc/TimedRobot.h>
#include <frc/Timer.h>
#include <frc/drive/DifferentialDrive.h>
#include <ctre/phoenix/motorcontrol/can/WPI_TalonFX.h>
```

PYTHON

```
import wpilib           # Used to get the joysticks
import wpilib.drive     # Used for the DifferentialDrive class
import ctre             # CTRE library
```

REV

JAVA

```
import com.revrobotics.CANSparkMax;
import com.revrobotics.CANSparkMaxLowLevel.MotorType;

import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.XboxController;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
```

C++

```
#include <frc/TimedRobot.h>
#include <frc/Timer.h>
#include <frc/XboxController.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/motorcontrol/PWMSparkMax.h>

#include <rev/CANSparkMax.h>
```

PYTHON

```
import wpilib           # Used to get the joysticks
import wpilib.drive     # Used for the DifferentialDrive class
import rev              # REV library
```

Notre code doit référencer les composants de WPILib qui sont utilisés. Cela se fait en utilisant les instructions `#include` (en C++) ou `import` (en Java). Le programme fait référence aux classes pour `Joystick` (pour le pilotage), `PWMSparkMax` / `WPI_TalonFX` / `CANSparkMax` (pour la commande des moteurs), ``TimedRobot` (la classe de base utilisée pour l'exemple) , `Timer` (utilisé pour le mode autonome) et `DifferentialDrive` (pour connecter la commande par joystick aux moteurs).

5.2.7 Définition des variables de notre prototype de robot

PWM

Java

```
20 public class Robot extends TimedRobot {
21   private final PWMSparkMax m_leftDrive = new PWMSparkMax(0);
22   private final PWMSparkMax m_rightDrive = new PWMSparkMax(1);
23   private final DifferentialDrive m_robotDrive =
24       new DifferentialDrive(m_leftDrive::set, m_rightDrive::set);
25   private final XboxController m_controller = new XboxController(0);
26   private final Timer m_timer = new Timer();
```

C++

```
12  public:
13   Robot() {
```

```
17     // We need to invert one side of the drivetrain so that positive voltages
18     // result in both sides moving forward. Depending on how your robot's
19     // gearbox is constructed, you might have to invert the left side instead.
20     m_right.SetInverted(true);
21     m_robotDrive.SetExpiration(100_ms);
22     m_timer.Start();
23   }
```

```
 private:
  // Robot drive system
  frc::PWMSparkMax m_left{0};
  frc::PWMSparkMax m_right{1};
  frc::DifferentialDrive m_robotDrive{
      [&](double output) { m_left.Set(output); },
      [&](double output) { m_right.Set(output); }};

  frc::XboxController m_controller{0};
  frc::Timer m_timer;
};
```

Python

```
class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        """
        This function is called upon program startup and
        should be used for any initialization code.
        """
        self.leftDrive = wpilib.PWMSparkMax(0)
        self.rightDrive = wpilib.PWMSparkMax(1)
        self.robotDrive = wpilib.drive.DifferentialDrive(
            self.leftDrive, self.rightDrive
        )
        self.controller = wpilib.XboxController(0)
        self.timer = wpilib.Timer()

        # We need to invert one side of the drivetrain so that positive voltages
        # result in both sides moving forward. Depending on how your robot's
        # gearbox is constructed, you might have to invert the left side instead.
        self.rightDrive.setInverted(True)
```

CTRE

Java

```
public class Robot extends TimedRobot {
   private final WPI_TalonFX m_leftDrive = new WPI_TalonFX(1);
   private final WPI_TalonFX m_rightDrive = new WPI_TalonFX(2);
   private final DifferentialDrive m_robotDrive = new DifferentialDrive(m_leftDrive,␣
↪m_rightDrive);
   private final Joystick m_stick = new Joystick(0);
   private final Timer m_timer = new Timer();
```

C++

```
 public:
  Robot() {
```

```
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side instead.
    m_right.SetInverted(true);
    m_robotDrive.SetExpiration(100_ms);
    m_timer.Start();
  }
```

```
private:
 // Robot drive system
 ctre::phoenix::motorcontrol::can::WPI_TalonFX m_left{1};
 ctre::phoenix::motorcontrol::can::WPI_TalonFX m_right{2};
 frc::DifferentialDrive m_robotDrive{m_left, m_right};

 frc::Joystick m_stick{0};
 frc::Timer m_timer;
```

Python

```
class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        """
        This function is called upon program startup and
        should be used for any initialization code.
        """
        self.leftDrive = ctre.WPI_TalonFX(1)
        self.rightDrive = ctre.WPI_TalonFX(2)
        self.robotDrive = wpilib.drive.DifferentialDrive(
            self.leftDrive, self.rightDrive
        )
        self.controller = wpilib.XboxController(0)
        self.timer = wpilib.Timer()

        # We need to invert one side of the drivetrain so that positive voltages
        # result in both sides moving forward. Depending on how your robot's
        # gearbox is constructed, you might have to invert the left side instead.
        self.rightDrive.setInverted(True)
```

REV

Java

```
public class Robot extends TimedRobot {
  private final CANSparkMax m_leftDrive = new CANSparkMax(1, MotorType.kBrushless);
  private final CANSparkMax m_rightDrive = new CANSparkMax(2, MotorType.kBrushless);
  private final DifferentialDrive m_robotDrive = new DifferentialDrive(m_leftDrive, m_
↪rightDrive);
```

(suite sur la page suivante)

(suite de la page précédente)

```
  private final XboxController m_controller = new XboxController(0);
  private final Timer m_timer = new Timer();
```

C++

```
 public:
  Robot() {
```

```
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side instead.
    m_right.SetInverted(true);
    m_robotDrive.SetExpiration(100_ms);
    m_timer.Start();
  }
```

```
private:
 // Robot drive system
 rev::CANSparkMax m_left{1, rev::CANSparkMax::MotorType::kBrushless};
 rev::CANSparkMax m_right{2, rev::CANSparkMax::MotorType::kBrushless};
 frc::DifferentialDrive m_robotDrive{m_left, m_right};

 frc::XboxController m_controller{0};
 frc::Timer m_timer;
```

Python

```
class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        """
        This function is called upon program startup and
        should be used for any initialization code.
        """
        self.leftDrive = rev.CANSparkMax(1, rev.CANSparkMax.MotorType.kBrushless)
        self.rightDrive = rev.CANSparkMax(2, rev.CANSparkMax.MotorType.kBrushless)
        self.robotDrive = wpilib.drive.DifferentialDrive(
            self.leftDrive, self.rightDrive
        )
        self.controller = wpilib.XboxController(0)
        self.timer = wpilib.Timer()

        # We need to invert one side of the drivetrain so that positive voltages
        # result in both sides moving forward. Depending on how your robot's
        # gearbox is constructed, you might have to invert the left side instead.
        self.rightDrive.setInverted(True)
```

Le robot de nos exemples a un joystick branché au port USB 0 pour le mode Arcade drive et deux moteurs branchés aux ports PWM 0 et 1 (les exemples des tierces parties utilisent CAN avec les ID 1 et 2). Ici nous créons des objets de type DifferentialDrive(m_robotDrive), Joystick(m_stick) et Timer(m_timer). Cette section du code accomplit trois choses :

1. Définit les variables en tant que membres de notre classe Robot.

2. Initialise les variables.

Note : L'initialisation des variables en C++ se trouve dans la section `private` en bas du programme. Cela signifie qu'elles sont privées à la classe `Robot`. Le code C++ configure également le paramètre `Motor Safety expiration` à 0,1 seconde (le déplacement s'arrêtera si nous ne lui donnons pas une commande à toutes les 0,1 secondes) et démarre le `Timer` utilisé pour le mode autonome.

5.2.8 Initialisation du robot

Java

```
37   @Override
38   public void robotInit() {
39     // We need to invert one side of the drivetrain so that positive voltages
40     // result in both sides moving forward. Depending on how your robot's
41     // gearbox is constructed, you might have to invert the left side instead.
42     m_rightDrive.setInverted(true);
43   }
```

C++

```
void RobotInit() {}
```

Python

```
def robotInit(self):
```

La méthode « RobotInit » est exécutée lorsque le programme du robot démarre, mais après le constructeur. Le « RobotInit » de notre programme d'exemple inverse le côté droit du groupe motopropulseur. Selon votre configuration de conduite, vous devrez peut-être inverser le côté gauche à la place.

Note : En C++, l'inversion du moteur est gérée dans le constructeur `Robot()` ci-dessus.

5.2.9 Exemple de mode autonome simple

JAVA

```
45   /** This function is run once each time the robot enters autonomous mode. */
46   @Override
47   public void autonomousInit() {
48     m_timer.restart();
49   }
```

(suite sur la page suivante)

(suite de la page précédente)

```java

  /** This function is called periodically during autonomous. */
  @Override
  public void autonomousPeriodic() {
    // Drive for 2 seconds
    if (m_timer.get() < 2.0) {
      // Drive forwards half speed, make sure to turn input squaring off
      m_robotDrive.arcadeDrive(0.5, 0.0, false);
    } else {
      m_robotDrive.stopMotor(); // stop robot
    }
  }
```

C++

```cpp
  void AutonomousInit() override { m_timer.Restart(); }

  void AutonomousPeriodic() override {
    // Drive for 2 seconds
    if (m_timer.Get() < 2_s) {
      // Drive forwards half speed, make sure to turn input squaring off
      m_robotDrive.ArcadeDrive(0.5, 0.0, false);
    } else {
      // Stop robot
      m_robotDrive.ArcadeDrive(0.0, 0.0, false);
    }
  }
```

PYTHON

```python
    def autonomousInit(self):
        """This function is run once each time the robot enters autonomous mode."""
        self.timer.restart()

    def autonomousPeriodic(self):
        """This function is called periodically during autonomous."""

        # Drive for two seconds
        if self.timer.get() < 2.0:
            # Drive forwards half speed, make sure to turn input squaring off
            self.robotDrive.arcadeDrive(0.5, 0, squareInputs=False)
        else:
            self.robotDrive.stopMotor()  # Stop robot
```

La méthode `AutonomousInit` est exécutée une seule fois au moment où le robot passe au mode autonomie à partir d'un autre mode. Dans le programme, nous redémarrons le `Timer` dans cette méthode.

`AutonomousPeriodic` est exécuté une fois par période lorsque que le robot est en mode autonome. Dans la classe `TimedRobot` la période est de durée fixe et par défaut est égale à 20ms. Dans cet exemple, le code périodique vérifie si la minuterie est inférieure à 2 secondes et, dans ce cas, fait avancer le robot à mi-vitesse à l'aide de la méthode `ArcadeDrive` de la classe

`DifferentialDrive`. S'il s'est écoulé plus de 2 secondes, le code met fin au déplacement du robot.

5.2.10 Contrôle par Joystick pendant le mode téléopéré

JAVA

```java
  /** This function is called once each time the robot enters teleoperated mode. */
  @Override
  public void teleopInit() {}

  /** This function is called periodically during teleoperated mode. */
  @Override
  public void teleopPeriodic() {
    m_robotDrive.arcadeDrive(-m_controller.getLeftY(), -m_controller.getRightX());
  }
```

C++

```cpp
  void TeleopInit() override {}

  void TeleopPeriodic() override {
    // Drive with arcade style (use right stick to steer)
    m_robotDrive.ArcadeDrive(-m_controller.GetLeftY(),
                             m_controller.GetRightX());
  }
```

PYTHON

```python
    def teleopInit(self):
        """This function is called once each time the robot enters teleoperated mode."
↪""

    def teleopPeriodic(self):
        """This function is called periodically during teleoperated mode."""
        self.robotDrive.arcadeDrive(
            -self.controller.getLeftY(), -self.controller.getRightX()
        )
```

Comme dans le mode Autonomous, le mode Teleop possède les fonctions `TeleopInit` et `TeleopPeriodic`. Dans cet exemple, nous n'avons rien à faire dans `TeleopInit`, qui est fourni uniquement à des fins d'illustration. Dans `TeleopPeriodic`, le code utilise la méthode `ArcadeDrive` pour faire correspondre l'axe Y du `Joystick` au mouvement rectiligne avancer/reculer des moteurs d'entraînement et l'axe X pour le mouvement de rotation.

5.2.11 Mode test

JAVA

```java
/** This function is called once each time the robot enters test mode. */
@Override
public void testInit() {}

/** This function is called periodically during test mode. */
@Override
public void testPeriodic() {}
```

C++

```cpp
void TestInit() override {}

void TestPeriodic() override {}
```

PYTHON

```python
    def testInit(self):
        """This function is called once each time the robot enters test mode."""

    def testPeriodic(self):
        """This function is called periodically during test mode."""
```

Le mode test est utilisé afin de tester la fonctionnalité du robot. À l'instar de `TeleopInit`, les méthodes `TestInit` et `TestPeriodic` sont fournies ici uniquement à des fins d'illustration.

5.2.12 Déploiement du code projet sur un robot

— *Compilation et déploiement du code Java/C++ du robot*
— *Déploiement du code Python*

5.3 Running your Test Program

5.3.1 Aperçu

You should create and download a Test Program as described for your programming language :

C++/Java/Python

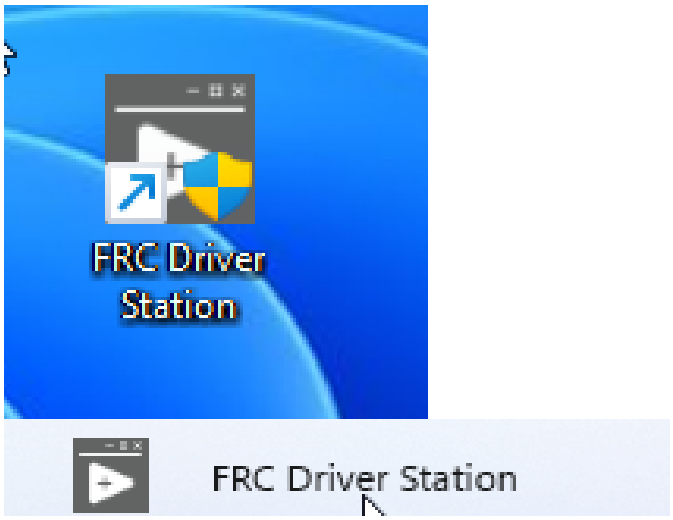
LabVIEW

5.3.2 Opération par l'intermédiaire d'un câble

Running your test program while tethered to the Driver Station via ethernet or USB cable will confirm the program was successfully deployed and that the driver station and roboRIO are properly configured.

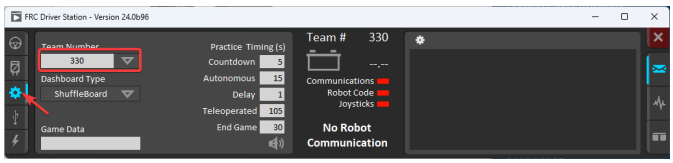
Le roboRIO doit être mis sous tension et connecté au PC via Ethernet ou USB.

5.3.3 Starting the FRC Driver Station



L'application FRC® Driver Station peut être lancée en double-cliquant sur l'icône localisée sur le bureau ou en sélectionnant Démarrer-> Tous les programmes-> FRC Driver Station.

5.3.4 Setting Up the Driver Station



L'application DS doit être configurée à votre numéro d'équipe afin de se connecter à votre robot. Pour ce faire, cliquez sur l'onglet Setup, puis entrez votre numéro d'équipe dans la zone Team number. Appuyez sur Return ou cliquez en dehors de la zone pour que le paramètre prenne effet.

PCs will typically have the correct network settings for the DS to connect to the robot already, but if not, make sure your Network adapter is set to *DHCP*.

5.3.5 Confirm Connectivity

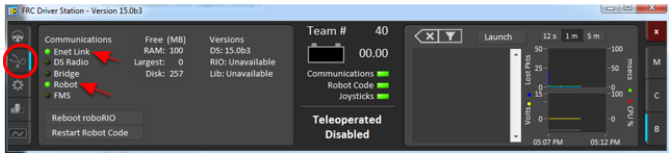


Fig. 1 – Tethered

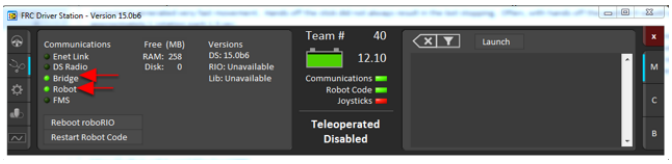
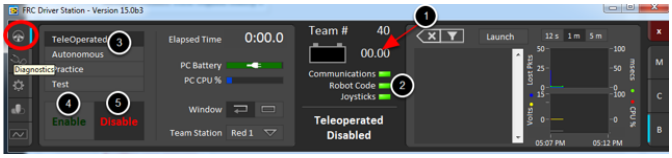


Fig. 2 – Sans fil

Using the Driver Station software, click Diagnostics and confirm that the Enet Link (or Robot Radio led, if operating wirelessly) and Robot leds are green.

5.3.6 Faire fonctionner le robot



Cliquer sur l'onglet Operation

1. Confirmer que la tension de la batterie s'affiche
2. Les indicateurs Communications, Code Robot et Joysticks sont verts.
3. Put the robot in Teleop Mode
4. Cliquez sur Enable. Actionnez les joysticks et observez comment le robot réagit.
5. Cliquez sur Disable

5.3.7 Fonctionnement sans fil

Before attempting wireless operation, tethered operation should have been confirmed as described in *Tethered Operation*. Running your test program while connected to the Driver Station via WiFi will confirm that the access point is properly configured.

Configuration du point d'accès

Consultez l'article *Programmer votre radio* pour plus de détails sur la configuration de la radio robot en vue d'une utilisation comme point d'accès.

After configuring the access point, connect the driver station wirelessly to the robot. The SSID will be your team number (as entered in the Bridge Configuration Utility). If you set a key when using the Bridge Configuration Utility you will need to enter it to connect to the network. Make sure the computer network adapter is set to DHCP (« Obtain an IP address automatically »).

You can now confirm wireless operation using the same steps in **Confirm Connectivity** and **Operate the Robot** above.

6 Aperçu des composants matériels

Le but de ce document est de fournir un bref aperçu des composants matériels qui composent le Système de contrôle FRC®. Chaque composant contient une brève description de la fonction du composant et un lien vers plus de documentation.

Note : Pour les instructions/schémas de câblage, veuillez consulter le document Wiring the FRC Control System.

6.1 Survol du système de contrôle

REV

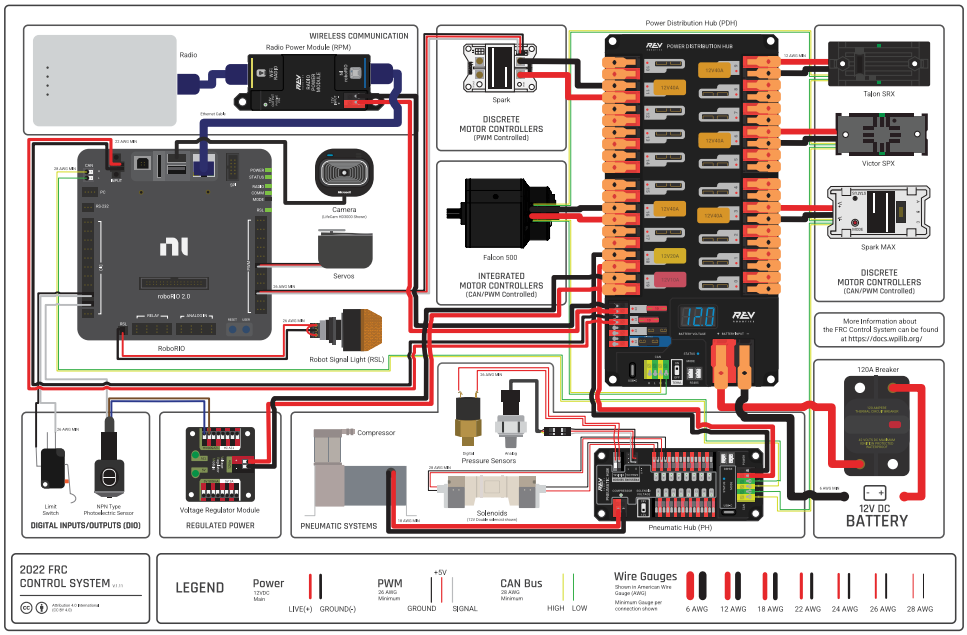


Diagramme gracieuseté de l’équipe FRC® 3161 et Stefen Acepcion.

CTRE

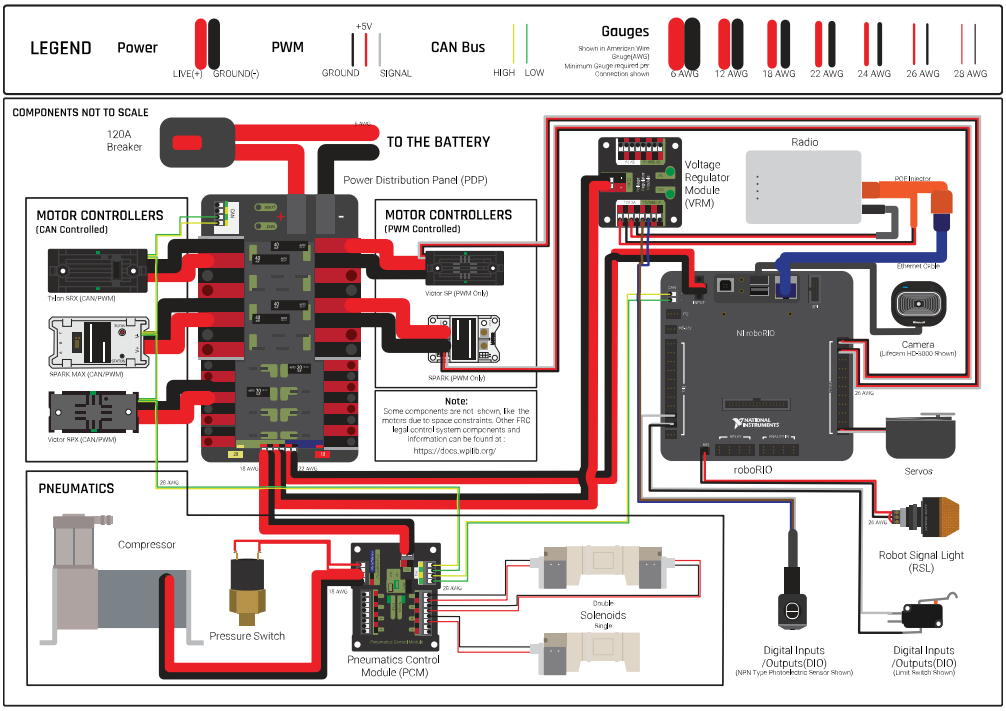
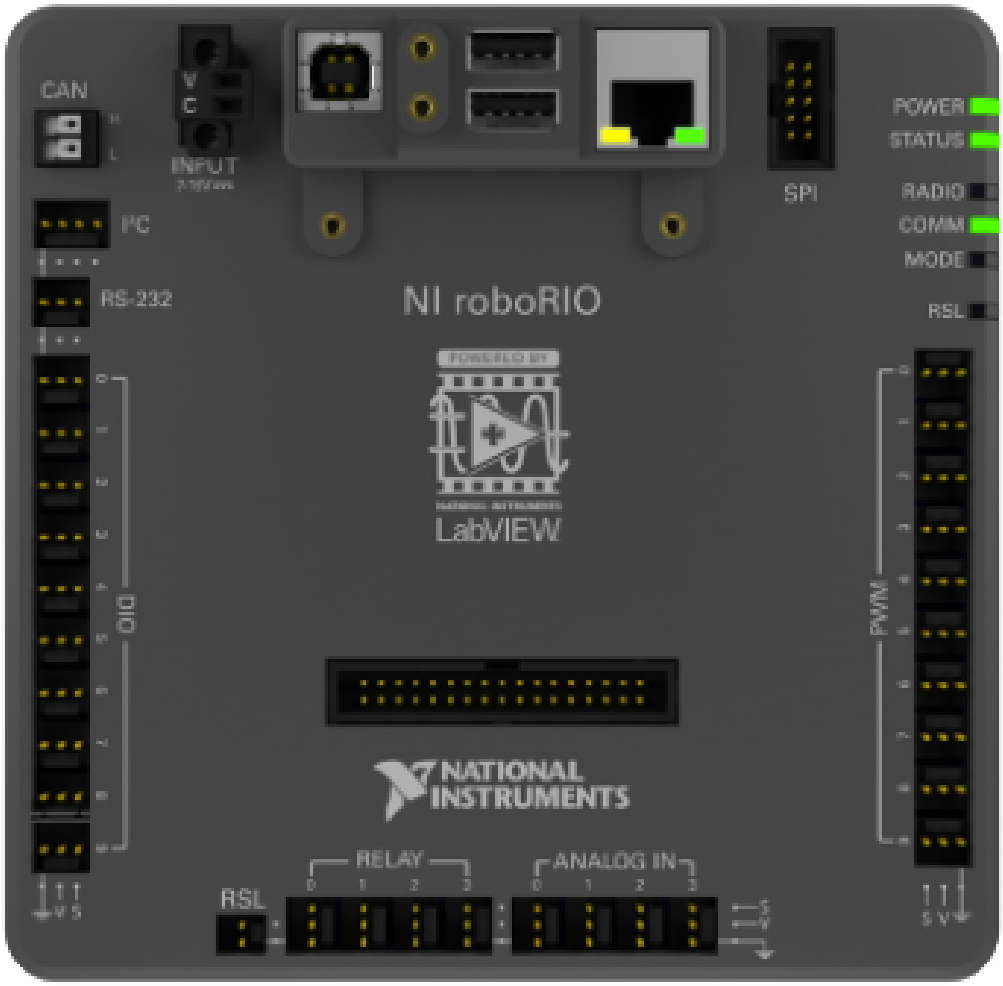


Diagramme gracieuseté de l’équipe FRC® 3161 et Stefen Acepcion.

6.2 Contrôleur principal NI RoboRIO de National Instrument

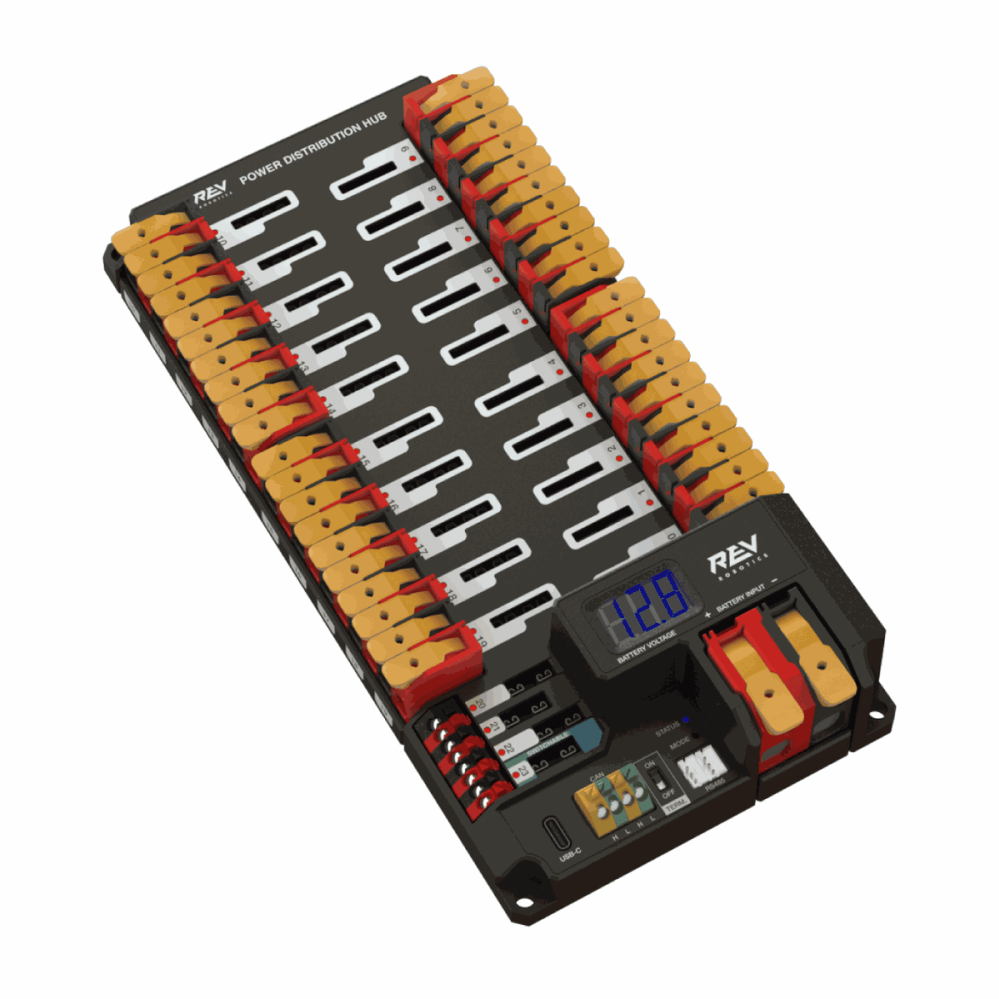


Le NI-roboRIO est le contrôleur principal de robot utilisé pour FRC. Le roboRIO sert de «cerveau» pour le robot en exécutant le code généré par l'équipe et qui commande tous les dispositifs électroniques et mécaniques.

6.3 Panneau de distribution de puissance CTRE

Le *Panneau de distribution de puissance CTRE* (PDP) est conçu pour distribuer le courant d'une batterie 12V CC aux divers composants du robot, par le biais d'un fusible à auto-réinitialisation et plusieurs connexions à fusibles. le PDP fournit 8 paires de sorties à puissance continue de 40 ampères et 8 paires de sorties à puissance continue de 30 ampères. Le PDP fournit des connecteurs dédiés à tension de 12V pour le roboRIO, ainsi que des connecteurs pour le Module de régulation de voltage et Module de contrôle pneumatique. Il inclut aussi une interface CAN pour la journalisation du courant, température interne et voltage de la batterie. Pour des informations plus détaillées, consultez le manuel de l'utilisateur du PDP.

6.4 Concentrateur de distribution de puissance REV



Le Concentrateur de Distribution de Puissance REV (PDH) est conçu pour distribuer l'énergie d'une batterie 12VDC à divers composants du robot. Le PDH dispose de 20 canaux à courant élevé (40 A max), 3 canaux à faible courant (15 A max) et 1 canal commutable à faible courant. Le concentrateur de distribution de puissance dispose de bornes WAGO à verrouillage sans outil, d'un affichage de tension LED et de la possibilité de se connecter via CAN ou USB-C au client matériel REV pour une télémétrie en temps réel.

6.5 Module régulateur de tension CTRE

Le module régulateur de tension CTRE (VRM) est un module indépendant alimenté par 12 volts. L'appareil est câblé à un connecteur dédié sur le PDP. Le module dispose de plusieurs sorties régulées 12V et 5V. Le but du VRM est de fournir une alimentation régulée pour la radio du robot, les circuits personnalisés et les caméras de vision IP. Pour plus d'informations, consultez le manuel VRM User Manual.

6.6 Module d'alimentation de radio REV

Le Module d'alimentation de radio REV est conçu pour maintenir alimenté l'un des composants les plus critiques du système, la radio WiFi OpenMesh, qui doit être alimenté en continu même dans les moments les plus difficiles de la compétition. Le module d'alimentation radio élimine le besoin d'alimenter la radio via une prise d'alimentation à barillet traditionnelle. Utilisant un POE passif 18V avec deux connecteurs RJ45 à prise, le module d'alimentation radio transmet le signal entre la radio et roboRIO tout en fournissant de l'énergie directement à la radio. Après avoir connecté la radio et le roboRIO, il envoie facilement de l'énergie au module d'alimentation radio en le câblant sur les canaux à faible courant du concentrateur de distribution de puissance à l'aide des bornes WAGO à bouton-poussoir à code couleur.

6.7 Radio OpenMesh OM5P-AN ou OM5P-AC

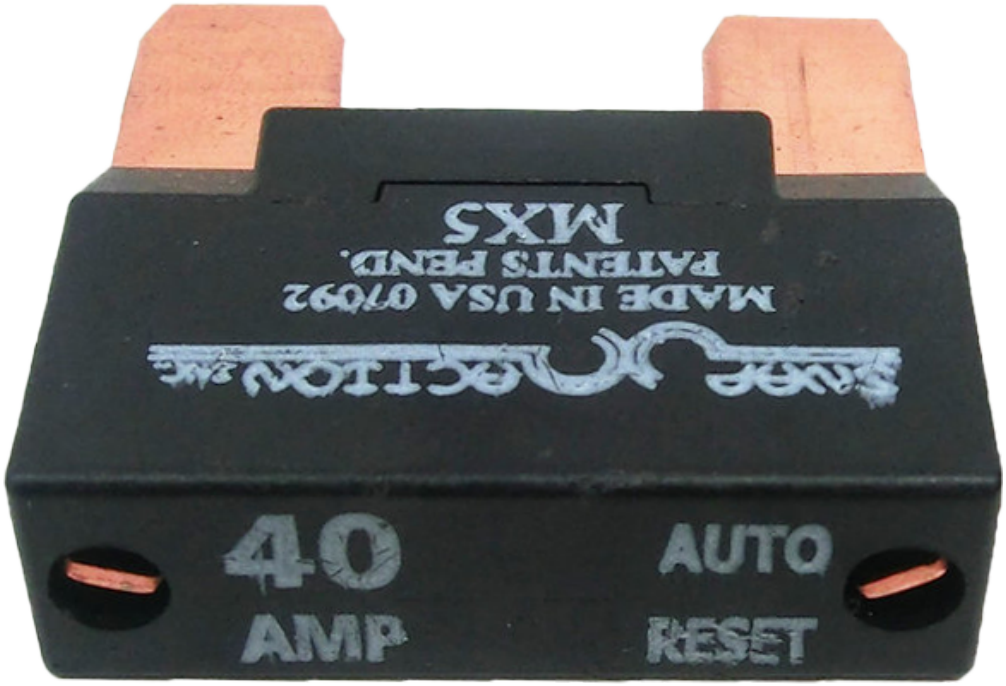
Soit l'OpenMesh OM5P-AN ou OpenMesh OM5P-AC la radio sans fil est utilisée comme radio du robot pour fournir une fonctionnalité de communication sans fil au robot. L'appareil peut être configuré comme point d'accès pour la connexion directe d'un ordinateur portable à utiliser à la maison. Il peut également être configuré comme un pont pour une utilisation sur le terrain. La radio du robot doit être alimentée par l'une des sorties 12 V/2 A du VRM et connectée au contrôleur roboRIO via Ethernet. Pour plus d'informations, voir *Programmer votre radio*.

Le OM5P-AN n'est plus disponible pour achat. Le OM5P-AC est légèrement plus lourd, possède plus de grilles de refroidissement et a une texture de surface rugueuse par rapport au OM5P-AN.

6.8 Disjoncteur 120A

Le disjoncteur principal de 120 A joue deux rôles sur le robot, soit l'interrupteur d'alimentation principal du robot et un dispositif de protection pour le câblage et les composants du robot en aval. Le disjoncteur de 120 A est câblé aux bornes positives de la batterie du robot et des tableaux de distribution d'alimentation. Pour plus d'informations, veuillez consulter la Fiche technique de la série Cooper Bussmann 18X (PN : 185120F)

6.9 Disjoncteurs à Action rapide



Les disjoncteurs Snap Action, série MX5 et série VB3, sont utilisés avec le panneau de distribution d'alimentation pour limiter le courant vers les circuits de dérivation. Les valeurs nominales de ces disjoncteurs concernent un courant continu, les valeurs de crête temporaires peuvent être considérablement plus élevées.

6.10 Batterie du Robot

L'alimentation électrique d'un robot FRC est composé d'une seule batterie plomb-acide scellée (SLA) 12V 18Ah, capable de répondre aux demandes de courant élevées d'un robot FRC. Pour plus d'informations, consultez la page *Batterie du Robot.*

Note : Plusieurs numéros de modèles de batterie peuvent être légalement utiisé. Consultez le Manuel FRC pour une liste complète.

6.11 Signal lumineux du robot

Allen-Bradley

Fig. 1 – Allen-Bradley 855PB-B12ME522

AndyMark

Le voyant de signalisation du robot (RSL) doit être le Allen-Bradley 855PB-B12ME522 ou bien le AndyMark am-3583. Il est directement contrôlé par le roboRIO et clignote lorsqu'il est activé et reste fixe lorsqu'il est désactivé.

Fig. 2 – AndyMark am-3583

6.12 Module de commande pneumatique CTRE

Le *module de contrôle pneumatique* (PCM) contient toutes les entrées et sorties nécessaires au fonctionnement des solénoïdes pneumatiques 12V ou 24V et du compresseur embarqué. Le PCM contient une entrée pour le capteur de pression et contrôle automatiquement le compresseur lorsque le robot est activé et qu'un solénoïde a été créé dans le code. Pour plus d'informations, consultez le document PCM User Manual..

6.13 Concentrateur pneumatique REV

Le Concentrateur pneumatique REV est un module autonome capable de commuter des électrovannes pneumatiques 12V et 24V. Le concentrateur pneumatique dispose de 16 canaux solénoïdes qui permettent prendre en charge jusqu'à 16 solénoïdes à simple effet, 8 solénoïdes à double effet ou une combinaison des deux types. La tension de sortie sélectionnable par l'utilisateur est entièrement régulée, ce qui permet même aux solénoïdes de 12 V de rester actifs lorsque la batterie du robot tombe aussi bas que 4.75 V.

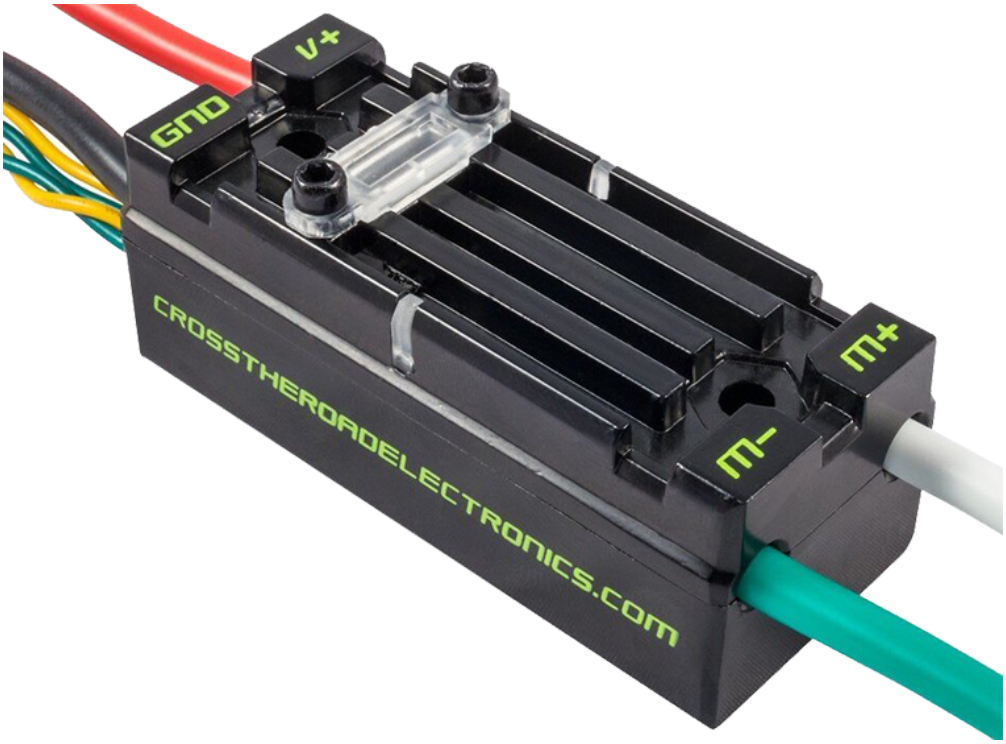
Des ports relatifs aux capteur de pression numériques et analogiques sont intégrés à l'appareil, ce qui augmente la flexibilité et la fonctionnalité de rétroaction du système pneumatique. La connexion USB-C sur le Concentrateur fonctionne avec le client matériel REV, permettant aux utilisateurs de tester des systèmes pneumatiques sans avoir besoin d'un contrôleur de robot supplémentaire.

6.14 Contrôleurs de Moteurs

Il existe une variété de différents *contrôleurs de moteur* qui fonctionnent avec le système de contrôle FRC et sont approuvés pour utilisation. Ces dispositifs sont utilisés pour fournir un contrôle de tension variable des moteurs à courant continu à balais et sans balais utilisés dans le FRC. Ils sont répertoriés ici par ordre d'utilisation.

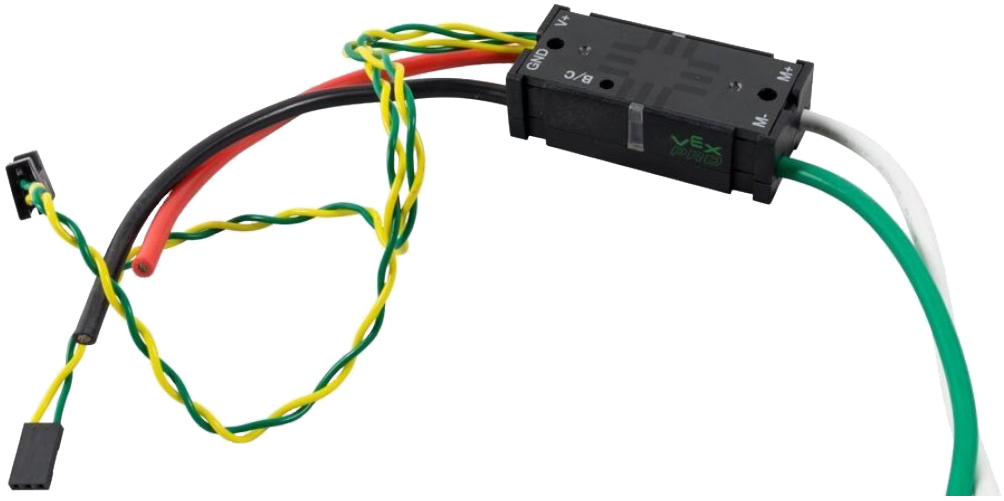
Note : Le contrôle CAN des tierce-parties n'est pas pris en charge par WPILib. Consultez cette section sur *Périphériques CAN provenant de tierce-partie* pour plus d'informations.

6.14.1 Talon SRX



Le contrôleur de moteur Talon SRX est un « contrôleur de moteur intelligent » de Cross The Road Electronics/VEX Robotics. Le Talon SRX peut être contrôlé via le bus CAN ou l'interface *PWM*. Lors de l'utilisation du contrôle par bus CAN, cet appareil peut prendre les entrées des interrupteurs de fin de course et des potentiomètres, des encodeurs ou des capteurs similaires afin d'effectuer un contrôle avancé. Pour plus d'informations, consultez le Guide de l'utilisateur Talon SRX.

6.14.2 Victor SPX



Le contrôleur de moteur Victor SPX est un contrôleur de moteur contrôlé CAN ou PWM de Cross The Road Electronics/VEX Robotique. Le dispositif est connecté pour permettre une connexion facile aux connecteurs roboRIO PWM ou à un bus CAN. Le boîtier est scellé pour empêcher les débris de pénétrer dans le contrôleur. Pour plus d'informations, consultez le Victor SPX User Guide.

6.14.3 Contrôleur de Moteur SPARK MAX

Le contrôleur de moteurs SPARK MAX est un contrôleur de moteur DC avancé avec balais et sans balais de REV Robotics. Lors de l'utilisation du bus CAN ou par contrôle USB, le SPARK MAX utilise des signaux d'entrée provenant des interrupteurs de fin de course, des encodeurs et d'autres capteurs, y compris l'encodeur intégré du moteur sans balais REV NEO, pour effectuer des modes de commande avancés. Le SPARK MAX peut être contrôlé sur PWM, CAN ou USB (pour configuration/test uniquement). Pour plus d'informations, consultez le document SPARK MAX User's Manual.

6.14.4 Contrôleur de moteur TalonFX

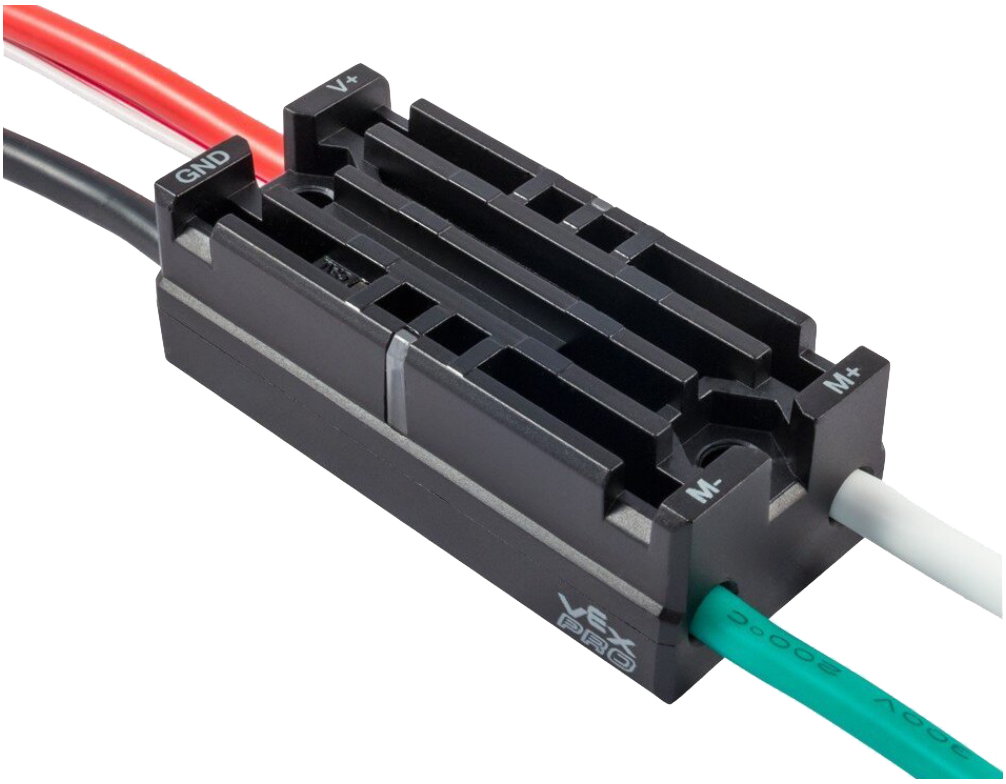
Le contrôleur de moteur TalonFX est intégré au moteur sans balais du Falcon 500. Il dispose d'un encodeur intégré et de toutes les fonctionnalités intelligentes du Talon SRX et plus encore ! Pour plus d'informations, consultez le document Falcon 500 User Guide.

6.14.5 Contrôleur de Moteur Spark

Avertissement : Bien que ce contrôleur moteur soit toujours légal pour une utilisation en FRC, le manufacturier a mis fin à la production de ce composant.

Le contrôleur de moteurs SPARK de REV Robotics est un contrôleur moteur DC avec balais bon marché. Le SPARK est contrôlé à l'aide de l'interface PWM. Les interrupteurs de fin de course peuvent être directement connectés au SPARK pour limiter les rotations du moteur dans une ou les deux directions. Pour plus d'informations, consultez le document SPARK User's Manual.

6.14.6 Victor SP



Avertissement : Bien que ce contrôleur moteur soit toujours légal pour une utilisation en FRC, le manufacturier a mis fin à la production de ce composant.

Le contrôleur de moteur Victor SP est un contrôleur de moteur PWM de Cross The Road Electronics/VEX Robotics . Le Victor SP a un boîtier métallique isolé électriquement pour la dissipation thermique, ce qui rend l'utilisation du ventilateur facultative. Le boîtier est scellé pour empêcher les débris de pénétrer dans le contrôleur. Le contrôleur fait environ la moitié de la taille des modèles précédents.

6.14.7 Contrôleur de Moteur Talon

Avertissement : Bien que ce contrôleur moteur soit toujours légal pour une utilisation en FRC, le manufacturier a mis fin à la production de ce composant.

Le contrôleur de moteurs de Cross the Road Electronics est un contrôleur de moteur DC avec balais à refroidissement passif contrôlé via les connexions PWM .

6.14.8 Contrôleur de Moteur Victor 888 / Contrôleur de Moteur Victor 884

Avertissement : Bien que ce contrôleur moteur soit toujours légal pour une utilisation en FRC, le manufacturier a mis fin à la production de ce composant.

Les contrôleurs de moteurs Victor 884 et Victor 888 de VEX Robotics sont des contrôleurs de moteurs de type PWM à vitesse variable à utiliser dans FRC. Le Victor 888 remplace le Victor 884, qui est également utilisable en FRC.

6.14.9 Contrôleur de Moteur Jaguar

Avertissement : Bien que ce contrôleur moteur soit toujours légal pour une utilisation en FRC, le manufacturier a mis fin à la production de ce composant.

Le contrôleur de moteur Jaguar de VEX Robotics (anciennement fabriqué par Luminary Micro et Texas Instruments) est un contrôleur de moteur à vitesse variable pour utilisation en FRC. En FRC, le Jaguar ne peut être contrôlé qu'à l'aide de l'interface PWM.

6.14.10 Contrôleurs de Moteurs DMC-60 et DMC-60C

Avertissement : Bien que ce contrôleur moteur soit toujours légal pour une utilisation en FRC, le manufacturier a mis fin à la production de ce composant.

Le DMC-60 est un contrôleur de motor PWM de la compagnie Digilent. Le DMC-60 dispose d'une détection thermique intégrée et d'une protection comprenant un repli-courant pour éviter la surchauffe et les dommages, de quatre LED multicolore pour indiquer la vitesse, la direction, et le statut pour un débogage plus facile. Pour d'autres informations, veuillez consulter le Manuel de référence DMC-60

Le DMC-60C ajoute des capacités de contrôleur intelligent CAN au contrôleur DMC-60. En raison de l'arrêt de production de ce produit par le fabricant, le DMC-60C n'est utilisable qu'avec des connexions PWM. Pour plus d'informations, consultez la documentation DMC-60C Product Page

6.14.11 Contrôleur de moteur Venom

Le Venom Motor Controller de Playing With Fusion est intégré dans un moteur basé sur le *CIM* original. La vitesse, le courant, la température et la position sont tous mesurés à bord, permettant des modes de contrôle avancés sans schémas de détection et de câblage compliqués.

6.14.12 Moteur Nidec Dynamo BLDC avec contrôleur

Le moteur Nidec Dynamo BLDC avec contrôleur est le premier moteur sans balais avec contrôleur intégré légal en FRC. Le contrôleur est intégré à l'arrière du moteur. La fiche de données du moteur fournit plus de détails sur ce périphérique.

6.14.13 Contrôleurs de Moteur SD540B et SD540C

Les contrôleurs moteurs SD540B et SD540C de Mindsensors sont contrôlés à l'aide des ports PWM. Le contrôle par réseau CAN n'est plus disponible pour le SD540C en raison du manque de soutien du fabricant. Les détecteurs de fin de course peuvent être câblés directement au SD540 pour limiter les déplacements des moteurs dans une ou les deux directions. Pour plus d'informations, voir la page Mindsensors FRC page

6.15 Relais H-Bridge Spike

Avertissement : Bien que ce relais soit toujours légal pour l'utilisation eb FRC, le manufacturier a cessé de fabriquer ce produit.

Le Relais H-Bridge Spike de VEX Robotics est un dispositif utilisé pour contrôler l'alimentation des moteurs ou d'autre circuits électroniques personnalisés de robot. Quand il est connecté à un moteur, le Spike procure un contrôle de Marche/Arrêt dans les deux sens de rotation. Les sorties du Spike sont contrôlées indépendamment donc ils peuvent également être utilisés pour alimenter jusqu'à 2 circuits électroniques personnalisés. Le Relais H-Bridge Spike devrait être connecté à une sortie relais du roboRIO et alimenté du Panneau de Distribution de puissance. Pour d'autres informations, veuillez consulter le Manuel de l'Utilisateur Spike.

6.16 Module d'Alimentation Servo

Le Module d'Alimentation Servo de REV Robotics est capable d'étendre la puissance disponible aux servos au-délà de ce que l'alimentation intégrée du roboRIO peut fournir. Le Module d'Alimentation Servo fournit jusqu'à 90W de puissance 6Volts sur 6 canaux. Tous les signaux de commande passent directement du roboRIO. Pour d'autres informations, veuillez consulter la page web du Module d'Alimentation Servo.

6.17 Microsoft Lifecam HD3000

La caméra Microsoft Lifecam HD3000 est une webcam USB qui peut être branchée directement dans le roboRIO. Cette caméra peut capturer des vidéos à une résolution maximale de 1280x720 à 30 IPS. Pour plus d'information sur la caméra, consultez la page de produit Microsoft. Pour plus d'information à propos de l'utilisation de la caméra avec le roboRIO, consultez la section *Vision Processing* de cette documentation.

6.18 Crédits d'Images

Image du roboRIO avec l'aimable autorisation de National Instruments. Image du DMC-60, une gracieuseté de Digilent. Image du SD540, une gracieuseté de Mindsensors. Images du contrôleur de moteur Jaguar, du Talon SRX, du Talon FX, du Victor 888, du Victor SP, du Victor SPX et du relais Spike H-Bridge, une gracieuseté de VEX Robotics, Inc. Images du SPARK MAX, du concentrateur de distribution de puissance, du module d'alimentation de la radio, et pneumatique, une gracieuseté de REV Robotics. Les photos des composants Lifecam, PDP, PCM, SPARK et VRM sont fournies par *FIRST*®. Toutes les autres photos sont une gracieuseté d'AndyMark Inc.

7 Aperçu des composants logiciels

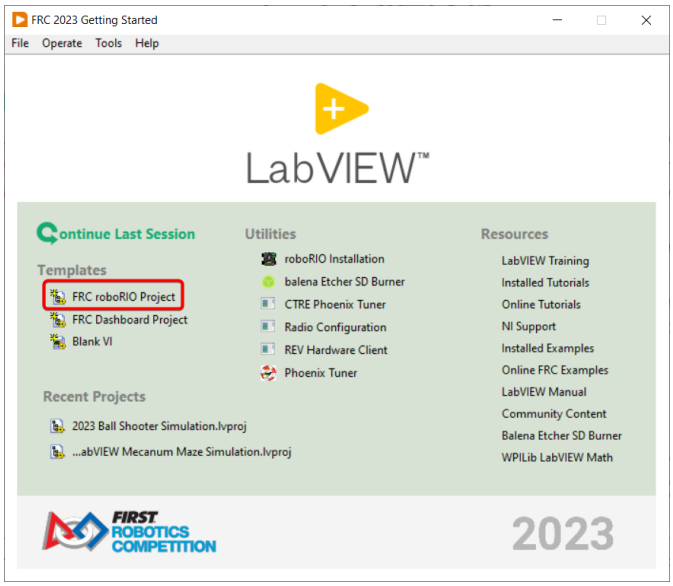
La partie logicielle FRC® se compose d'une grande variété de composants obligatoires et optionnels. Ces éléments sont conçus pour vous aider dans la conception, le développement et le débogage de votre code robot ainsi que pour aider au contrôle fonctionnement du robot et pour fournir des informations de retour lors du dépannage. Pour chaque composant logiciel, ce document fournira un bref aperçu de son objet, un lien vers le téléchargement du package, le cas échéant, et un lien vers d'autres documents, le cas échéant.

7.1 Compatibilité du système d'exploitation

Le principal système d'exploitation pris en charge pour les composants FRC est Windows. Tous les composants logiciels de FRC ont été testés sur Windows 10 et 11.

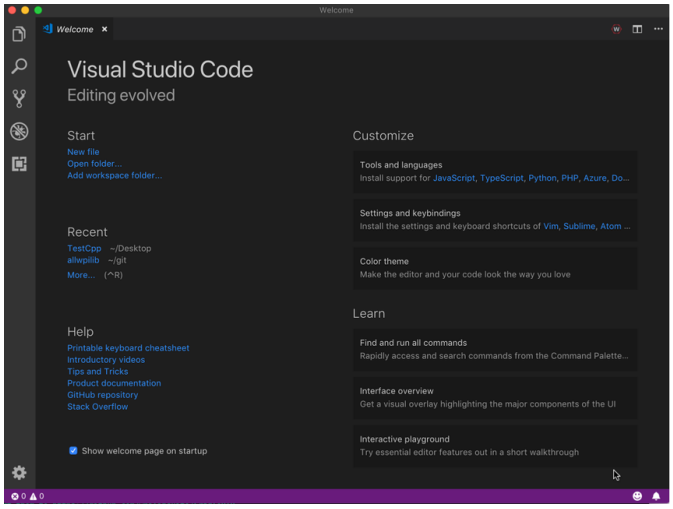
La plupart des outils de programmation C++/Java/Python sont également pris en charge et testés sous macOS et Linux. Les équipes qui programment en C++/Java/Python devraient être en mesure de développer à l'aide de ces systèmes, en utilisant un système Windows pour les opérations windows uniquement telles que l'application Driver Station, l'utilitaire Radio Configuration Utility et l'outil roboRIO Imaging Tool

7.2 LabVIEW FRC (Windows seulement)



LabVIEW FRC, basé sur une version récente de LabVIEW Professional, est l'un des trois langages officiellement pris en charge pour la programmation d'un robot FRC. LabVIEW est un langage graphique axé sur le flux de données. Les programmes LabVIEW consistent en une collection d'icônes, appelées Instruments Virtuels ou VIs, connectées avec des fils qui transmettent des données entre les VIs. L'installateur LabVIEW FRC est distribué sur un DVD livré au lancement dans le kit de pièces et est également disponible en ligne par téléchargement. Un guide pour commencer avec le logiciel LabVIEW FRC, comprenant les instructions d'installation peut être trouvé *ici*.

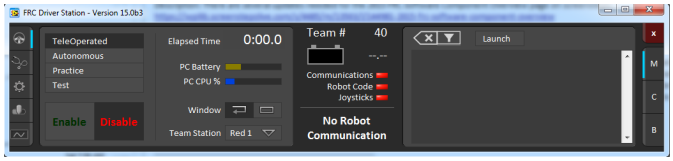
7.3 Visual Studio Code



Visual Studio Code est l'environnement de développement pris en charge pour C++ et Java. Un guide pour commencer avec Java et C++ pour FRC, comprenant l'installation et la confi-

guration de Visual Studio Code est disponible *ici*.

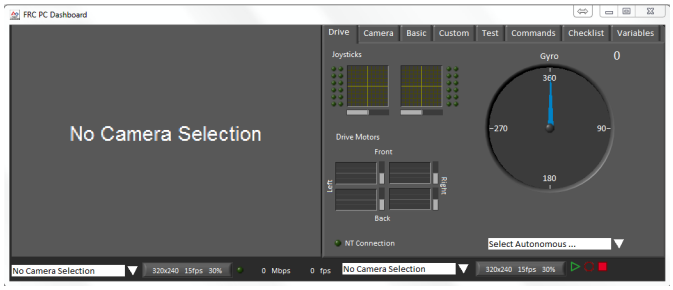
7.4 FRC Driver Station par NI LabVIEW (Windows seulement)



C'est le seul logiciel autorisé à être utilisé dans le but de contrôler l'état du robot pendant la compétition. Ce logiciel envoie des données à votre robot à partir d'une variété de périphériques d'entrée. Il contient également un certain nombre d'outils utilisés pour aider à résoudre les défaillances du robot. Plus d'informations sur FRC Driver Station Optimisé par NI LabVIEW peuvent être trouvées *ici*.

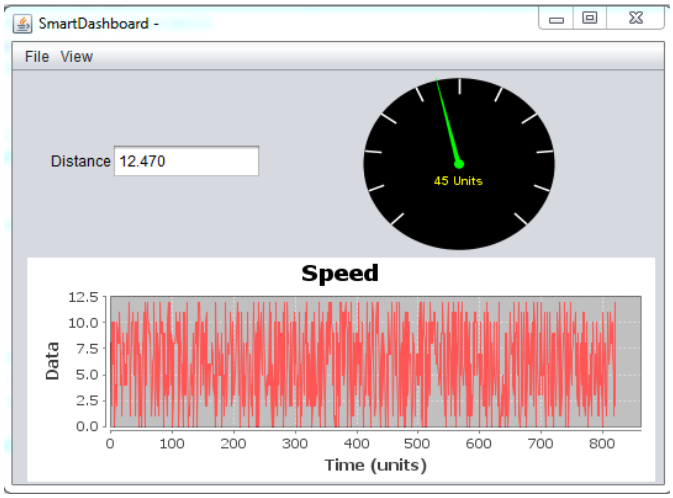
7.5 Options du Dashboard

7.5.1 LabVIEW Dashboard (Windows seulement)



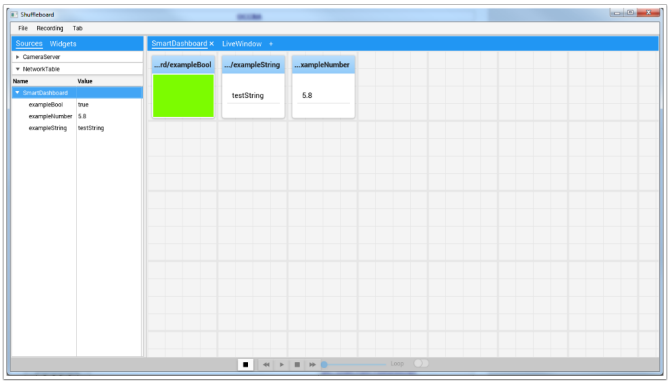
Le tableau de bord LabVIEW est lancé automatiquement par la FRC Driver Station par défaut. L'objectif du tableau de bord est de fournir des informations sur le fonctionnement du robot à l'aide d'un affichage à onglets avec une variété de fonctionnalités intégrées. Vous trouverez plus d'informations sur le logiciel FRC Default Dashboard *ici*.

7.5.2 SmartDashboard



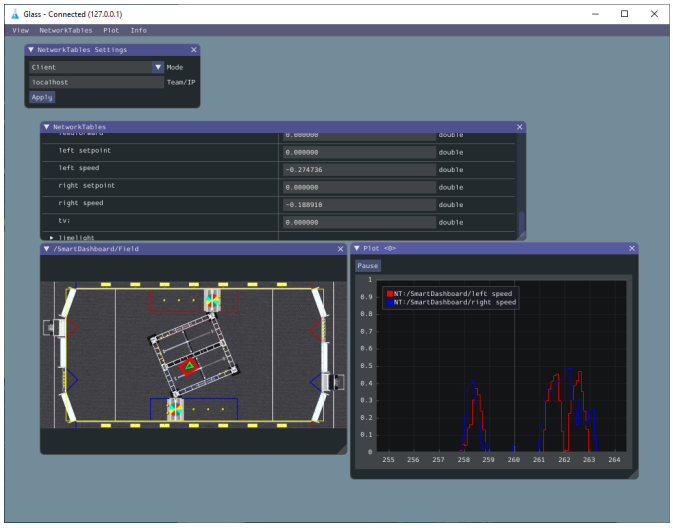
SmartDashboard vous permet de visualiser les données de votre robot en créant automatiquement des indicateurs personnalisables spécifiquement pour chaque donnée envoyée par votre robot. Une documentation supplémentaire sur SmartDashboard peut être trouvée *ici*.

7.5.3 Shuffleboard



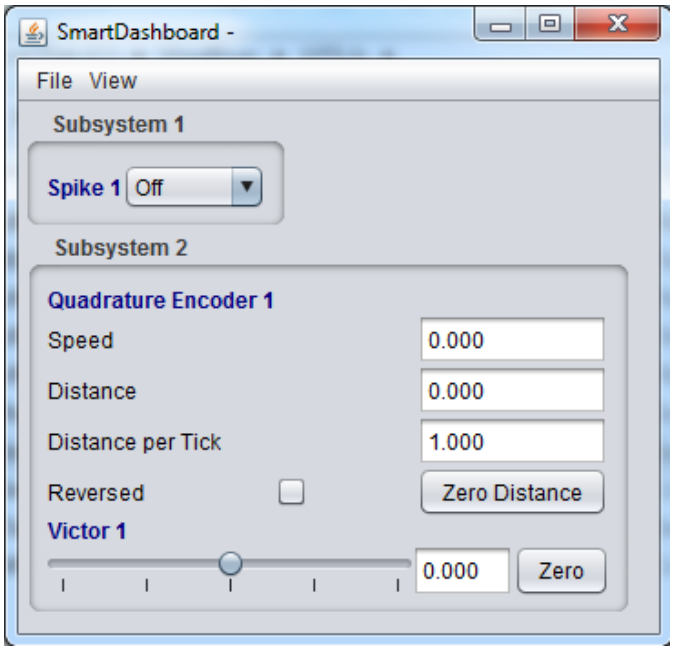
Shuffleboard a les mêmes fonctionnalités que SmartDashboard. Il améliore également la configuration et la visualisation de vos données avec de nouvelles fonctionnalités et une conception moderne au prix d'une utilisation moins efficace des ressources. Une documentation supplémentaire sur Shuffleboard peut être trouvée *ici*.

7.5.4 Glass



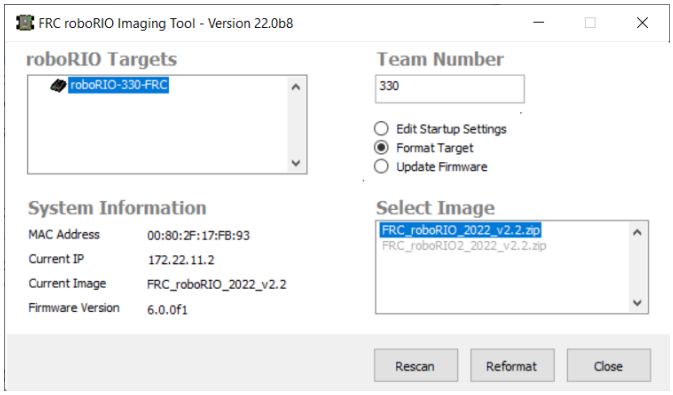
Glass est un tableau de bord destiné à être un outil de programmation pour le débogage. Les principaux avantages sont la vue sur le terrain, la visualisation de la pose et les outils avancés de traçage du signal.

7.6 LiveWindow



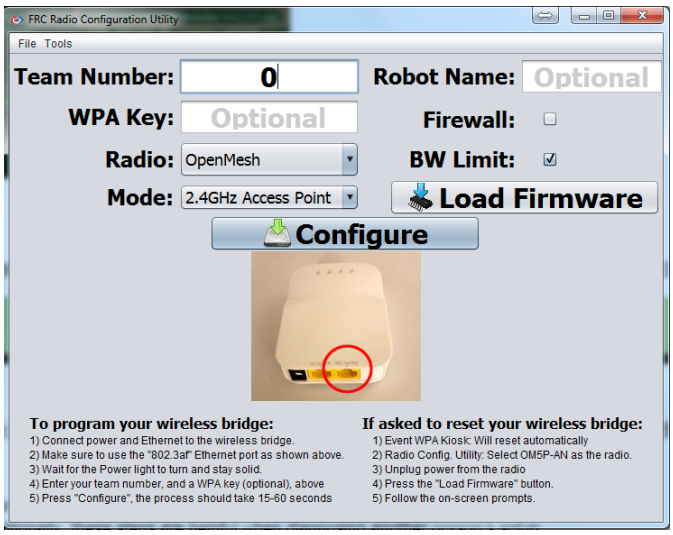
LiveWindow est une fonctionnalité de SmartDashboard et Shuffleboard, conçue pour être utilisée avec le mode test de la Driver Station. LiveWindow permet à l'utilisateur de voir les retours des capteurs sur le robot et les actionneurs de contrôle indépendamment du code utilisateur écrit. Plus d'informations sur LiveWindow peuvent être trouvées *ici*.

7.7 Outil d'installation d'image roboRIO FRC (Windows Seulement)



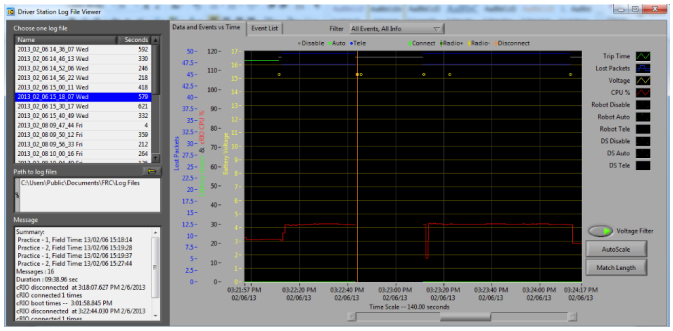
Cet outil est utilisé pour formater et configurer un roboRIO pour une utilisation en FRC. Les instructions d'installation peuvent être trouvées *ici*. Des instructions supplémentaires sur l'installation de l'image de votre roboRIO à l'aide de cet outil peuvent être trouvées *ici*.

7.8 Utilitaire de Configuration de Radio FRC (Windows Seulement)



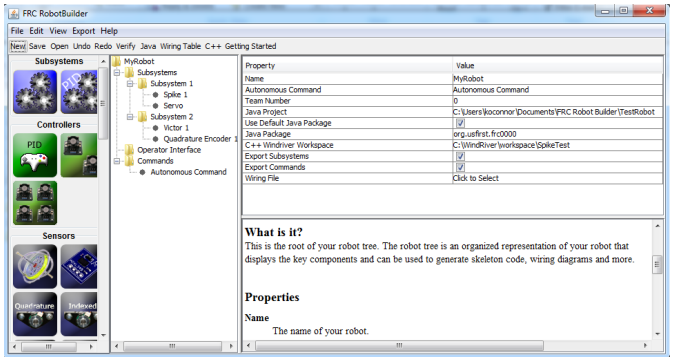
L'outil FRC Radio Configuration Utility est utilisé pour configurer la radio standard pour une utilisation pratique à domicile. Cet outil définit les paramètres réseau appropriés pour imiter l'expérience du terrain de jeu FRC. L'utilitaire FRC Radio Configuration Utility est installé à l'aide d'un installateur autonome qui peut être trouvé *ici*.

7.9 Visionneur de Journaux de la Driver Station FRC (Windows Seulement)



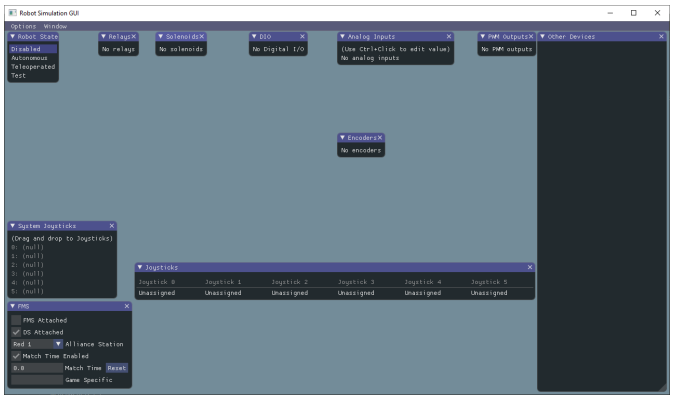
La visionneur FRC Driver Station Log Viewer est utilisé pour afficher les journaux créés par FRC Driver Station. Ces journaux contiennent une variété d'informations importantes pour comprendre ce qui s'est passé lors d'une séance de pratique ou d'un match FRC. Vous pouvez trouver plus d'informations sur le visionneur FRC Driver Station Log Viewer et l'interprétation des informations des journaux *ici*

7.10 RobotBuilder



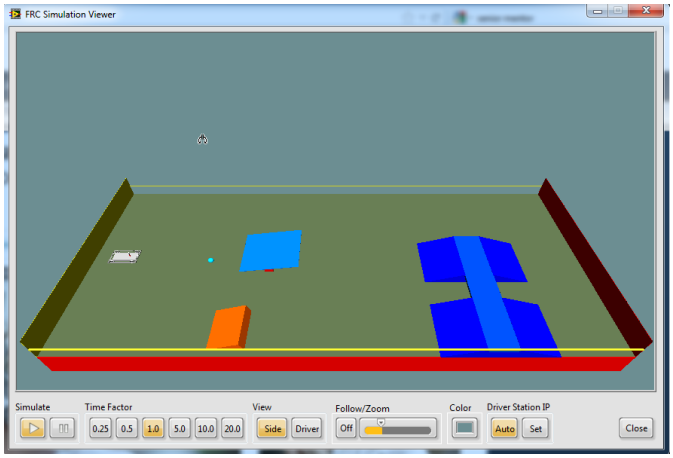
RobotBuilder est un outil conçu pour faciliter la configuration et la structuration d'un projet de robot orienté commande pour C++ ou Java (Python n'est pas encore supporté). RobotBuilder vous permet d'entrer dans les différents composants de vos sous-systèmes robot et interface opérateur et de définir ce que vos commandes sont dans une structure graphique arborescente. RobotBuilder générera alors un modèle structurel du code pour vous aider à démarrer. Plus d'informations sur RobotBuilder peuvent être trouvées *ici*. Pour plus d'informations sur l'architecture de programmation commande, cliquez sur *ici*.

7.11 Simulateur de robot



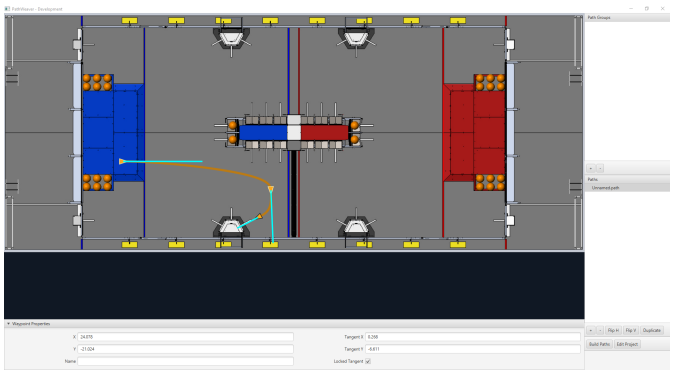
Robot Simulation est un outil qui offre aux équipes Java, C++ et Python un moyen de vérifier que leur code robot réel fonctionne dans un environnement simulé. Cette simulation peut être lancée directement à partir de VS Code et comprend un terrain 2D dont les utilisateurs peuvent servir pour visualiser le mouvement de leur robot. Pour plus d'informations, consultez la section *Simulation du robot*.

7.12 FRC LabVIEW Simulateur de robot (Windows seulement)



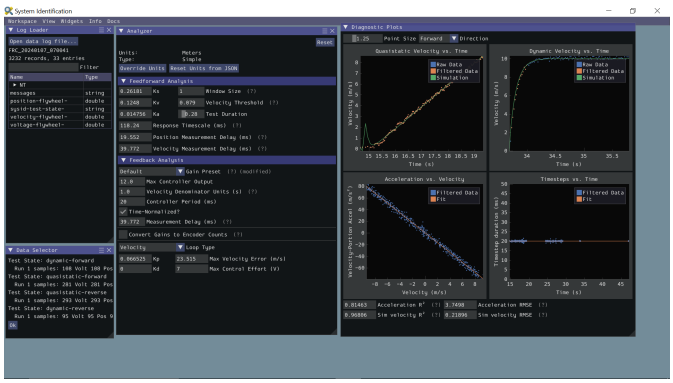
Le simulateur FRC Robot Simulatordriver station est un composant de l'environnement de programmation LabVIEW qui vous permet d'utiliser un robot prédéfini dans un environnement simulé pour tester le code et/ou les fonctions de la Driver Station. Vous pouvez trouver des informations sur l'utilisation du simulateur FRC Robot Simulator ici ou en ouvrant le fichier Robot Simulation Readme.html dans l'explorateur LabVIEW Project Explorer.

7.13 PathWeaver



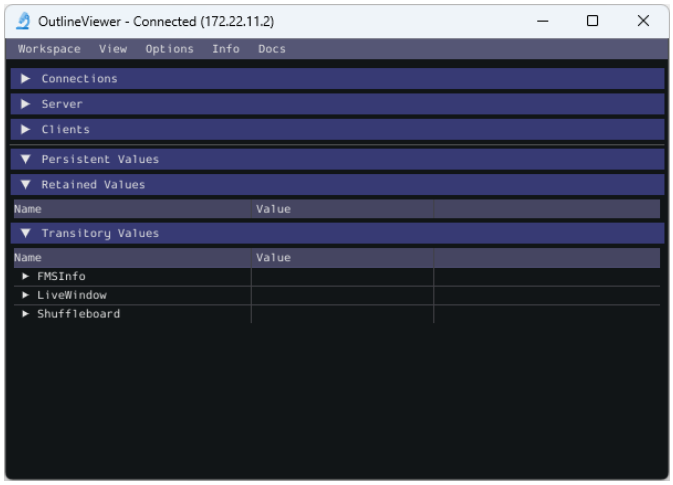
PathWeaver permet aux équipes de générer et de configurer rapidement des chemins pour des routines autonomes avancées. Ces chemins ont des courbes douces permettant à l'équipe de diriger rapidement leur robot entre les points sur le terrain. Pour plus d'informations, consultez la section *PathWeaver*.

7.14 Identification du système



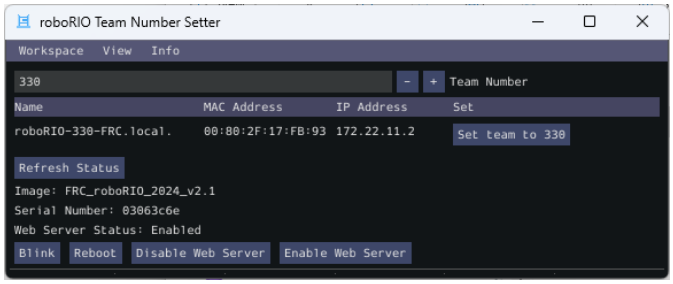
This tool helps teams automatically calculate constants that can be used to describe the physical properties of your robot for use in features like robot simulation, trajectory following, and PID control. For more information see the *System Identification section*.

7.15 OutlineViewer



OutlineViewer est un utilitaire utilisé pour afficher, modifier et ajouter à tout le contenu des NetworkTables à des fins de débogage. Les équipes LabVIEW peuvent utiliser l'onglet Variables du tableau de bord LabVIEW pour accomplir cette fonctionnalité. Pour plus d'informations, consultez la section *Outline Viewer*.

7.16 roboRIO Team Number Setter



The roboRIO Team Number Setter is a cross-platform utility that can be used to set the team number on the roboRIO. It is an alternative to the roboRIO imaging tool for setting the team number. For more information see the *roboRIO Team Number Setter section*.

8

Qu’est-ce que WPILib ?

The WPI Robotics Library (WPILib) is the standard *software library* provided for teams to write code for their FRC® robots. WPILib contains a set of useful classes and subroutines for interfacing with various parts of the FRC control system (such as sensors, motor controllers, and the driver station), as well as an assortment of other utility functions.

8.1 Languages supportés

There are three versions of WPILib, one for each of the three officially-supported text-based languages : WPILibJ for Java, and WPILibC for C++, and RobotPy for Python. A considerable effort is made to maintain feature-parity between these languages - library features are not added unless they can be reasonably supported for both Java and C++ (with the C++ able to be wrapped by pybind for Python), and when possible the class and method names are kept identical or highly-similar. Java, C++, and Python were chosen for the officially-supported languages due to their appropriate level-of-abstraction and ubiquity in both industry and high-school computer science classes.

In general, C++ offers better high-end performance, at the cost of increased user effort (memory must be handled manually, and the C++ compiler does not do much to ensure user code will not crash at runtime). Java and Python offer lesser performance, but much greater convenience. Python users should take care to test their program to ensure that typos and other issues don’t cause robot crashes, as Python is interpreted. New/inexperienced users are encouraged to use Java.

8.2 Code source et documentation

WPILib is an open-source library - the C++ and Java source code is in the allwpilib mono-repo and python source code is in the mostrobotpy mono-repo. The Java and C++ source code can be found in the WPILibJ and WPILibC source directories :

Le code source Java et C++ se trouve dans les répertoires source WPILibJ et WPILibC :

- Code source en Java
- Code source en C++
- Python source code

While users are strongly encouraged to read the source code to resolve detailed questions about library functionality, more-concise documentation can be found on the official documentation pages for WPILibJ and WPILibC and RobotPy :

— Java documentation
— C++ documentation
— Python documentation

9 2024 Overview

9.1 Problèmes connus

Cet article détaille les problèmes connus (et les solutions de contournement) relatifs à la partie logicielle du système de contrôle FRC®.

9.1.1 Questions ouvertes

AdvantageScope isn't updated by WPILib Installer on macOS

Issue : When running the WPILib Installer, a pop-up saying `"WPILibInstaller" was prevented from modifying apps on your Mac.` and AdvantageScope remains version 3.0.1. This issue occurs when upgrading WPILib, when a beta version of WPIlib or WPILib 2024.1.1 was installed on macOS.

Workaround : Delete AdvantageScope from `~/wpilib/tools` and re-run the WPILib Installer.

Driver Station randomly disabled

Issue : The Driver Station contains tighter safety mechanisms in 2024 to protect against control issues. Some teams have seen this cause the robot to disable.

Workaround : There are multiple potential causes for tripping the safety mechanisms.

Note : The new safety mechanisms will *not* disable the robot when connected to the *FMS*.

Driver Station 24.0.1 from Game Tools 2024 Patch 1 contains an update to the safety controls that may resolve the issue in certain circumstances. If the issue is still seen with this version installed, please continue with the troubleshooting steps below.

The Driver Station software has new tools for control packet delays that could cause this. The control system team requests that teams that experience this issue post screenshots of the *Driver Station Timing window* to https://github.com/wpilibsuite/allwpilib/issues/6174

Some teams have seen this happen only when the robot is operated wirelessly, but not when operated via USB or ethernet tether. Some potential mitigations :

1. Try relocating the robot radio to a better location (high in the robot and away from motors or large amounts of metal).
2. *Measure your robot's bandwidth* and ensure you have margin to the 4 Mbps bandwidth limit
3. See if the Wi-Fi environment is congested using a tool like WiFi Analyzer. As the 5 ghz WiFi spectrum has more channels and is less crowded, switching the robot radio to operate at 5 ghz will likely improve WiFi communication.
4. Update the Wi-Fi drivers for the computer.
5. If you operate multiple robots in close proximity in access point mode, setting up a router and operating the radios in bridge mode will reduce the number of wireless access points and may improve communications

Some teams have seen this happen due to software that is running on the driver station (such as Autodesk updater or Discord). Some potential mitigations :

1. Reboot the driver station computer
2. Close software that is running in the background
3. Follow the *Driver Station Best Practices*

While rare, this can be caused by robot code that oversaturates the roboRIO processor or network connection. If all other troubleshooting steps fail, you can try running with one of the WPILib example programs to see if the problem still occurs.

If you identify software that interferes with driver station, please post it to https://github.com/wpilibsuite/allwpilib/issues/6174

Driver Station Reports Less Free RAM then is Available

Issue : The Driver Station diagnostic screen reports free RAM that is misleadingly low. This is due to Linux's use of memory caches. Linux will cache data in memory, but then relinquish when the robot programs requests more memory. The Driver Station only reports memory that isn't used by caches.

Workaround : The true memory available to the robot program is available in the file `/proc/meminfo`. *Use ssh to connect to the robot*, and run `cat /proc/meminfo`.

```
MemTotal:         250152 kB
MemFree:           46484 kB
MemAvailable:     126956 kB
```

The proper value to look is as MemAvailable, rather then MemFree (which is what the driver station is reporting).

Driver Station Reporting No Code

Issue : There is a rare occurrence in the roboRIO 2.0 that causes the roboRIO to not properly start the robot program. This causes the Driver Station to report a successful connection but no code, even though code is deployed on the roboRIO.

Workaround : We are currently investigating the root cause, but FIRST volunteers have been made aware and the recommendation is to reboot the roboRIO when this occurs.

Note : Pressing the physical *User* button on the roboRIO for 5 seconds can also cause the robot code to not start, but a reboot will not start the robot code. If the robot code does not start after rebooting, press the *User* button. Ensure that nothing on the robot is in contact with the *User* button.

Radio Second Port Sometimes Fails to Communicate

Issue : There is a rare occurrence in the OM5P Radios that causes the second Ethernet port (the one farthest from the power plug) to not communicate.

Workaround : Generally, power cycling the radio will restablish communication with the second port. Alternately, utilize a network switch such as the tp-link switch formerly available from FIRST Choice or the brainboxes SW-005 and plug all ethernet devices into the network switch and then plug the switch into the radio's first Ethernet port. This also allows easier tethering while at competition.

Port I2C intégré provoquant des blocages du système

Issue : Use of the onboard I2C port on the roboRIO 1 or 2, in any language, can result in system lockups. The frequency of these lockups appears to be dependent on the specific hardware (i.e. different roboRIOs will behave differently) as well as how the bus is being used.

Workaround : The only surefire mitigation is to use the MXP I2C port or another device to read the I2C data. Accessing the device less frequently and/or using a different roboRIO may significantly reduce the likelihood/frequency of lockups, it will be up to each team to assess their tolerance of the risk of lockup. This lockup can not be definitively identified on the field and a field fault will not be called for a match where this behavior is believed to occur. This lockup is a CPU/kernel hang, the roboRIO will completely stop responding and will not be accessible via the DS, webpage or SSH. If you can access your roboRIO via any of these methods, you are experiencing a different issue.

Several alternatives exist for accessing the REV color sensor without using the roboRIO I2C port. A similar approach could be used for other I2C sensors.

— Use a Raspberry Pi Pico. Supports up to 2 REV color sensors, sends data to the roboRIO via serial. The Pi Pico is low cost (less than $10) and readily available.
— Use a Raspberry Pi. Supports 1-4 color sensors, sends data to the roboRIO via NetworkTables. Primarily useful for teams already using a Raspberry Pi as a coprocessor.

La mise à jour des propriétés du roboRIO 2.0 peut être lente ou bloquée

Problème : La mise à jour des propriétés d'un roboRIO 2.0 sans avoir à le reformatter à l'aide de l'outilitaire Imaging Tool (par exemple, la configuration du numéro d'équipe) peut être lente ou bloquée.

Solution : Après quelques minutes d'attente pendant l'exécution de l'outilitaire Imaging Tool, le roboRIO devrait pouvoir être redémarré et les nouvelles propriétés devraient être configurées.

Simulation crashes on Mac after updating WPILib

Issue : On macOS, after updating the project to use a newer version of WPILib, running simulation immediately crashes without the GUI appearing.

Workaround : In VS Code, run WPILib | Run a command in Gradle, `clean`. Alternatively, run `./gradlew clean` in the terminal or delete the build directory.

Compilation invalide en raison de l'absence de GradleRIO

Problème : Très rare, le cache Gradle d'un utilisateur sera brisé et ils recevront des erreurs similaires à ce qui suit :

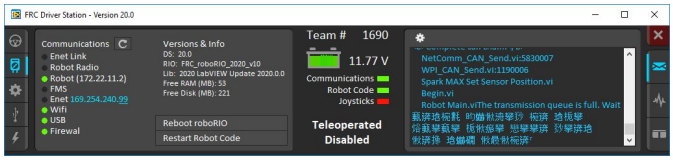
```
Could not apply requested plugin [id: 'edu.wpi.first.GradleRIO', version: '2020.3.2']␣
↪as it does not provide a plugin with id 'edu.wpi.first.GradleRIO'
```

Solution de contournement :

Supprimez votre cache Gradle situé sous `~$USER_HOME/.gradle`. Les machines Windows peuvent avoir besoin d'activer la possibilité de voir les fichiers cachés. Jusqu'à présent, ce problème n'est observé que sous Windows. S'il vous plaît signaler ce problème si vous l'observer sous un système d'exploitation alternatif.

Caractères chinois dans le journal du poste de conduite

Problème : Rarement, le journal du poste de conduite affichera des caractères chinois au lieu du texte anglais. Cela semble se produire uniquement lorsque Windows est défini sur une langue autre que l'anglais.



Solution de contournement : Il existe deux solutions de contournement connues :

1. Copiez et collez les caractères chinois dans le bloc-notes, et le texte anglais sera affiché.
2. Changez temporairement la langue de Windows en anglais.

C++ Intellisense - Les fichiers ouverts au lancement ne fonctionnent pas correctement

Problème : En C++, les fichiers ouverts au lancement de VS Code auront des problèmes avec Intellisense affichant les suggestions de toutes les options d'une unité de compilation et pas seulement celles appropriées ou ne trouvant pas de fichiers d'en-tête. Il s'agit d'un bogue dans VS Code.

Solution de contournement :

1. Fermez tous les fichiers dans VS Code, mais laissez VS Code ouvert
2. Supprimer le fichier c_cpp_properties.json dans le dossier .vscode, s'il existe
3. Run the « Refresh C++ Intellisense » command in VS Code.
4. En bas à droite, vous devriez voir quelque chose qui ressemble à une plate-forme (linuxathena ou windowsx86-64, etc.). S'il ne s'agit pas de linuxathena, cliquez dessus et définissez-le sur linuxathena (version)
5. Attendez ~ 1 min
6. Ouvrez le fichier cpp principal (pas un fichier d'en-tête avec extension .h). Intellisense devrait maintenant fonctionner

Problèmes avec les Dashboards WPILib et de simulation sur les éditions Windows N

Problème : Le code WPILib utilisant CSCore (dashboards et code robot simulé) aura des problèmes sur les versions Education N de Windows.
— Le shuffleboard fonctionnera, mais ne chargera pas les caméras
— Smartdashbard va planter au démarrage
— La simulation de robot va planter au démarrage
Solution : Installez le Media Feature Pack

9.1.2 Fixed in WPILib 2024.2.1

Visual Studio Code Reports Unresolved Dependency

Issue : Java programs will report `Unresolved dependency: org.junit.platform junit-platform-launcherJava(0)` on build.gradle. Programs that use unit tests will fail to build. This causes build.gradle to be highlighted red in the Visual Studio Code explorer, and the `plugins` line in build.gradle to have a red squiggle.

Workaround : This can be safetly ignored if you aren't running unit tests. To fix it, do the following :

On Windows execute the following in powershell :

```
Invoke-WebRequest -Uri https://repo.maven.apache.org/maven2/org/junit/jupiter/junit-
↪jupiter/5.10.1/junit-jupiter-5.10.1.module -OutFile C:\Users\Public\wpilib\2024\
↪maven\org\junit\jupiter\junit-jupiter\5.10.1\junit-jupiter-5.10.1.module
Invoke-WebRequest -Uri https://repo.maven.apache.org/maven2/org/junit/junit-bom/5.10.
↪1/junit-bom-5.10.1.module -OutFile C:\Users\Public\wpilib\2024\maven\org\junit\
↪junit-bom\5.10.1\junit-bom-5.10.1.module
```

On Linux/macOS execute the following :

```
curl https://repo.maven.apache.org/maven2/org/junit/jupiter/junit-jupiter/5.10.1/
↪junit-jupiter-5.10.1.module -o ~/wpilib/2024/maven/org/junit/jupiter/junit-jupiter/
↪5.10.1/junit-jupiter-5.10.1.module
curl https://repo.maven.apache.org/maven2/org/junit/junit-bom/5.10.1/junit-bom-5.10.1.
↪module -o ~/wpilib/2024/maven/org/junit/junit-bom/5.10.1/junit-bom-5.10.1.module
```

After running those, you'll need to refresh Java intellisense in VS Code for it to pick up the new files. You can do so by running the `Clean Java Language Server Workspace` command in VS Code.

9.1.3 Fixed in Game Tools 2024 Patch 1

Driver Station internal issue with print error and tags

Issue : The Driver Station will occasionally print `internal issue with print error and tags`. The message is caused when the DS reports a message on its side intermixed with messages from the robot ; it is not caused by and does not affect robot code.

Workaround : This will be fixed in the next Game Tools release. There is no known workaround.

9.2 New for 2024

A number of improvements have been made to FRC® Control System software for 2024. This article will describe and provide a brief overview of the new changes and features as well as a more complete changelog for Java/C++ WPILib changes. This document only includes the most relevant changes for end users, the full list of changes can be viewed on the various WPILib GitHub repositories.

Il est recommandé de consulter également la liste des *problèmes connues*.

9.2.1 Importing Projects from Previous Years

Due to internal GradleRIO changes, it is necessary to update projects from previous years. After *Installing WPILib for 2024*, any 2023 projects must be *imported* to be compatible.

9.2.2 Changements majeurs (Java/C++)

Ces changements contiennent *certains* des changements majeurs apportés à la bibliothèque qu'il est important que l'utilisateur reconnaisse. Cela n'inclut pas toutes les modifications de rupture, consultez les autres sections de ce document pour plus de modifications.

- Added support for *XRP robots*
- Projects now default to supporting Java 17 features
- Multiple NetworkTables networking improvements for improved reliability and robustness and structured data support using protobuf
- Java now uses the Serial GC by default on the roboRIO ; this should improve performance and reduce memory usage for most robot programs
- Performance improvements and reduced worst-case memory usage throughout libraries

— Added a typesafe unit system for Java (not used by the main part of WPILib yet)
— Disabled LiveWindow in Test Mode by default. See See *Enabling LiveWindow in Test Mode* to enable it.
— SysId has been rewritten to remove project generation ; Replaced with data logging within team robot program

Supported Operating Systems and Architectures :

— Windows 10 & 11, 64 bit. 32 bit and Arm are not supported
— Ubuntu 22.04, 64 bit. Other Linux distributions with glibc >= 2.32 may work, but are unsupported
— macOS 12 or later, Intel and Arm.

Avertissement : The following OSes are no longer supported : macOS 11, Ubuntu 18.04 & 20.04, Windows 7, Windows 8.1, and any 32-bit Windows.

9.2.3 WPILib

Librairies générales

— Commands :
  — Added proxy factory to `Commands`
  — Added `IdleCommand`
  — Fixed `RepeatCommand` calling `end()` twice
  — Added `onlyWhile()` and `onlyIf()` decorators
  — Implemented `ConditionalCommand.getInterruptBehavior()`
  — Added interruptor parameter to `onCommandInterrupt` callbacks
  — Added `DeferredCommand`, `Commands.defer()`, and `Subsystem.defer()`
  — Add requirements parameter to `Commands.idle()`
  — Fix Java `CommandXboxController.leftTrigger()` parameter order
  — Make Java `SelectCommand` generic
  — Add finallyDo with zero-arg lambda
— NetworkTables :
  — Networking improvements for improved reliability and robustness
  — Added subprotocol to improve web-based dashboard connection aliveness checking
  — Bugfixes and stability improvements (reduced worst case memory usage)
  — Improved update behavior for values continuously updated from robot code (improves command button behavior)
— Data Logging :
  — Improved handling of low free space conditions (now stops logging if less than 5 MB free)
  — Added warning about logging to built-in storage on RoboRIO 1
  — Reduced worst case memory usage
  — Improved file rename functionality to only use system time after it is updated by DS
  — NT publishers created before the log is started are now captured
  — Add delete without download functionality to DataLogTool
  — Changed default log location to logs subdirectory for better organization
— Hardware interfaces :
  — Getting timestamps is now ~10x faster
  — Exposed power rail disable and CPU temperature functionality
  — Exposed CAN timestamp base clock
  — Fixed and documented addressable LED timings

  - Fixed `DutyCycleEncoder` reset behavior
  - Added function to read the *RSL* state
  - Raw *PWM* now uses microseconds units
  - Fixed REVPH faults bitfield
  - C++ : Fix `Counter` default distance per pulse to match Java
- Math :
  - Refactored kinematics, odometry, and pose estimator internals to have less code duplication ; you can implement custom drivetrains via the `Kinematics` and `Odometry` interfaces and the `PoseEstimator` class.
  - LTV controllers use a faster DARE solver for faster construction (from 2.33 ms per solve on a roboRIO to 0.432 ms in Java and 0.188 ms in C++ on a roboRIO)
  - (Java) `Rotation3d.rotateBy()` got a 100x speed improvement by using doubles in Quaternion instead of EJML vectors
  - (Java) `Pose3d.exp()` and `Pose3d.log()` got a speed improvement by calling the C++ version through JNI instead of using EJML matrices
  - Improved accuracy of Rotation3d Euler angle calculations (`getX()`, `getY()`, `getZ()`, aka roll-pitch-yaw) near gimbal lock
  - Fixed `CoordinateSystem.convert()` Transform3d overload
  - Modified `TrapezoidProfile` API to not require creating new instances for `ProfiledPIDController`-like use cases
  - Added Exponential motion profile support
  - Add constructor overloads for easier Transform2d and Transform3d creation from X, Y, Z coordinates
  - Add `ChassisSpeeds fromRobotRelativeSpeeds` to convert from robot relative to field relative
  - Add method to create a LinearSystem from kA and kV, for example from a characterized mechanism
  - Add `SimulatedAnnealing` class
  - Fixed MecanumDriveWheelSpeeds desaturate()
- Added `RobotController` function to get the assigned team number
- Updated `GetMatchTime` docs and units
- Added function to wait for DS connection
- Added reflection based cleanup helper
- Added Java class preloader (no preloading is actually performed yet)
- Deprecated `Accelerometer` and `Gyro` interfaces (no replacement is planned)
- Updated to OpenCV 4.8.0 and EJML 0.43.1 and C++ JSON to 3.11.2
- Add `PS5Controller` class
- Add accessors for `AprilTagFieldLayout` origin and field dimensions
- ArcadeDrive : Fix max output handling
- Add `PWMSparkFlex` Motor Controller
- ADIS16470 : allow accessing all three axes
- Deprecated `MotorControllerGroup`. Use `PWMMotorController` `addFollower()` method or if using CAN motor controllers use their method of following.
- Added functional inteface to `DifferentialDrive` and `MecanumDrive`. The `MotorController` interface may be removed in the future to reduce coupling with vendor libraries. Instead of passing `MotorController` objects, the following method references or lambda expressions can be used :
  - Java : `DifferentialDrive drive = new DifferentialDrive(m_leftMotor::set, m_rightMotor::set);`
  - C++ : `frc::DifferentialDrive m_drive{[&](double output) { m_leftMotor.Set(output); }, [&](double output) { m_rightMotor.Set(output); }};`

Changements majeurs

- Changed `DriverStation.getAllianceStation()` to return optional value. See *example usage*
- Merged CommandBase into Command (Command is now a base class instead of an interface)
- Potentially breaking : made command scheduling order consistent
- Removed various deprecated command classes and functions :
  - `PerpetualCommand` and `Command.perpetually()` (use `RepeatCommand/repeatedly()` instead)
  - `CommandGroupBase`, `Command.IsGrouped()` (C++ only), and `Command.SetGrouped()` (C++ only) ; the static factories have been moved to `Commands`
  - `Command.withInterrupt()`
  - `ProxyScheduleCommand`
  - `Button` (use `Trigger` instead)
  - **Old-style Trigger functions : `whenActive()`, `whileActiveOnce()`, `whileActiveContinuous()`, `whenInactive()`, `toggleWhenActive()`, `cancelWhenActive()`. Each binding type has both `True` and `False` variants ; for brevity, only the `True` variants are listed here :**
    - `onTrue()` (replaces `whenActive()` and `whenPressed()`) : schedule on rising edge.
    - `whileTrue()` (replaces `whileActiveOnce()`) : schedule on rising edge, cancel on falling edge.
    - `toggleOnTrue()` (replaces `toggleWhenActive()`) : on rising edge, schedule if unscheduled and cancel if scheduled.
    - `cancelWhenActive()` : this is a fairly niche use case which is better described as having the trigger's rising edge (`Trigger.rising()`) as an end condition for the command (using `Command.until()`).
    - `whileActiveContinuously()` : however common, this relied on the *no-op* behavior of scheduling an already-scheduled command. The more correct way to repeat the command if it ends before the falling edge is using `Command.repeatedly()`/`RepeatCommand` or a `RunCommand` – the only difference is if the command is interrupted, but that is more likely to result in two commands perpetually canceling each other than achieve the desired behavior. Manually implementing a blindly-scheduling binding like `whileActiveContinuously()` is still possible, though might not be intuitive.
  - `CommandScheduler.clearButtons()`
  - `CommandScheduler.addButtons()` (Java only)
  - Command supplier constructor of `SelectCommand` (use `ProxyCommand` instead)
- Removed `Compressor.enabled()` function (use `isEnabled()` instead)
- Removed `CameraServer.setSize()` function (use `setResolution()` on the camera object instead)
- Removed deprecated and broken SPI methods
- Removed 2-argument constructor to `SlewRateLimiter`
- Removed `frc2::PIDController` alias (`frc::PIDController` already existed)
- For ease of use, `loadAprilTagFieldLayout()` now throws an unchecked exception instead of a checked exception
- Add new parameter for `ElevatorSim` constructor for starting height
- Report error on negative PID gains

9.2.4 Simulation

— Unified PWM simulation Speed, Position, and Raw values to be consistent with robot behavior
— Expanded DutyCycleEncoderSim API
— Added ability to set starting state of mechanism sims
— Added mechanism-specific SetState overloads to physics sims

9.2.5 SmartDashboard

Important : SmartDashboard is not supported on Apple Silicon (Arm64) Macs.

— Connection to the robot now always occurs after processing the save file. Fixes the problem that Choosers don't show up if connection to the robot happens before a chooser in the save file is processed
— Added LiveWindow widgets to containing subsystem widget when creating them from the save file
— Now properly handles putting the Scheduler on SmartDashboard with `SmartDashboard.putData()`

9.2.6 Glass / OutlineViewer / Simulation GUI

— Include standard field images for Field2D background
— Enhanced array support in NetworkTables views
— Added background color selector to glass plots
— Added tooltips for NT settings
— Improved title bar message
— Fixed loading a maximized window on second monitor
— Fixed crash when clearing existing workspace
— Fixed file dialogs not closing after window closes
— add ProfiledPIDController support

9.2.7 GradleRIO

— Use Java Serial GC by default
— Remove AlwaysPreTouch from Java arguments (reduces startup memory usage)
— Added support for XRP
— Enforces that vendor dependencies set correct frcYear (prevents using prior year vendor dependencies)
— Upgraded to Gradle 8.4
— Check that project isn't in OneDrive, as that causes issues

9.2.8 WPILib au complet dans un seul installateur

— Update to VS Code 1.85.1
— VS Code extension updates : cpptools 1.19.1, javaext 1.26.0
— Use separate zip files for VS Code download/install
— Update to use .NET 8
— AdvantageScope is now bundlled by the installer

9.2.9 Extension de code Visual Studio

— Java source code is now bundled into the deployed jar file. This makes it possible to recover source code from a deployed robot program.
— Added XRP support
— Check that project isn't created in OneDrive, as that causes issues

9.2.10 RobotBuilder

— Add POVButton
— Fixed constants aliasing
— Updated PCM references and wiring export for addition of REV PH

9.2.11 SysId

— Removed project generation ; Replaced with data logging within team robot program

9.3 Quick Start for Returning Teams

This section serves as a launching point for veteran teams that need to update to the current year's software.

It is advised that **all** teams read through the *changelog* and *known issues* for the season.

1. *Install LabVIEW* (LabVIEW teams only)
2. *Install Game Tools*
3. *Install WPILib* (Java / C++ teams only)
4. *Update third party libraries*
5. Reimage *roboRIO 1* or *roboRIO 2*
6. *Import robot project* (Java / C++ teams only)
7. Update software based on the changes described in the *changelog*

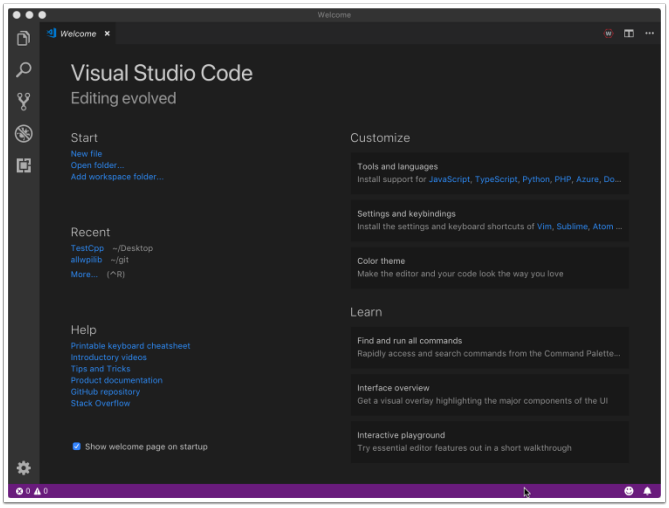
10

Survol de VS Code

10.1 Les bases de Visual Studio Code et l'extension WPILib

Visual Studio Code de Microsoft est l'IDE pris en charge pour le développement C++ et Java dans FRC. Cet article présente certaines des bases de l'utilisation de Visual Studio Code et de l'extension WPILib.

10.1.1 Page d'accueil



Lorsque Visual Studio Code s'ouvre pour la première fois, une page d'accueil vous est présentée. Sur cette page, vous trouverez quelques liens rapides qui vous permettent de personnaliser Visual Studio Code ainsi qu'un certain nombre de liens vers des documents et vidéos d'aide qui peuvent vous être utiles pour en apprendre d'avantage sur les bases de l'IDE ainsi que quelques conseils et astuces.

Vous pouvez également remarquer un petit logo WPILib dans le coin supérieur droit. Il s'agit d'une façon d'accéder aux fonctionnalités fournies par l'extension WPILib (discuté ci-dessous).

10.1.2 Interface utilisateur

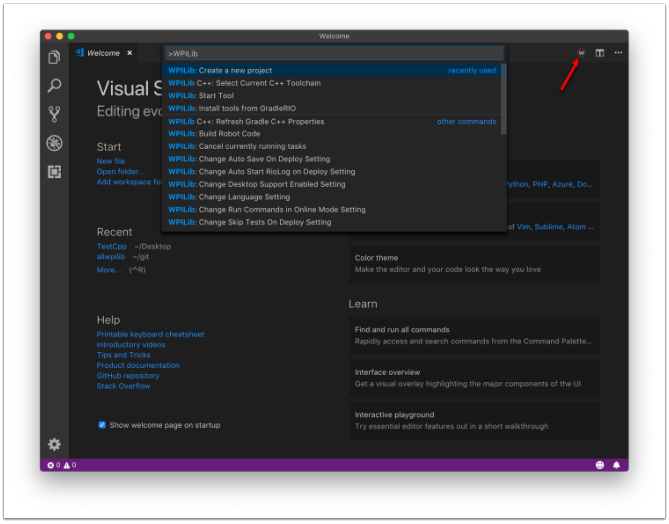
Le lien le plus important à examiner est probablement le document de base de l'interface utilisateur. Ce document décrit un grand nombre de fondements de l'utilisation de l'interface utilisateur et fournit la majorité des informations dont vous devez avoir besoin pour commencer à utiliser Visual Studio Code pour FRC.

10.1.3 Palette de commandes

La palette de commandes peut être utilisée pour accéder ou exécuter presque toutes les fonctions ou fonctionnalités de Visual Studio Code (y compris celles de l'extension WPILib). La palette de commandes est accessible depuis le menu Affichage ou en appuyant sur `Ctrl+Shift+P` (`Cmd+Shift+P` sous macOS). Taper du texte dans la fenêtre restreindra dynamiquement la recherche aux commandes pertinentes et les affichera dans la liste déroulante.

Dans l'exemple suivant « wpilib » est saisi dans la zone de recherche après l'activation de la palette de commandes, et cette opération réduit la liste aux fonctions contenant WPILib.

10.1.4 Extension WPILib



L'extension WPILib fournit la fonctionnalité FRC® spécifique liée à la création de projets et de composants de projet, à la création et au téléchargement de code sur le roboRIO et plus encore. Vous pouvez accéder aux commandes WPILib de deux manières :

- En tapant « WPILib » dans la palette de commandes
- En cliquant sur l'icône WPILib en haut à droite de la plupart des fenêtres. Cette opération ouvre la palette de commandes avec « WPILib » déjà saisi

---

Note : Il n'est **pas** recommandé d'installer le plugin Visual Studio IntelliCode avec l'installation FRC de VS Code car il est connu pour arrêter IntelliSense de manière inattendue.

Pour plus d'informations sur les commandes spécifiques de l'extension WPILib, veuillez consulter les autres articles de ce chapitre.

10.2 Commandes WPILib sous Visual Studio Code

Ce document contient une liste complète des commandes fournies par l'extension *WPILib VS Code Extension* et ce qu'elles font.

Pour accéder à ces commandes, appuyez sur Ctrl+Maj+P pour ouvrir la palette *Command Palette*, puis commencez à taper le nom de la commande comme illustré ici pour filtrer la liste des commandes. Cliquez sur le nom de la commande pour l'exécuter.

- **WPILib : Build Robot Code** - Construit un projet ouvert à l'aide de GradleRIO
- **WPILib : Create a new project** - Créer un nouveau projet de robot
- **WPILib C++ : Refresh C++ Intellisense** - Force une mise à jour de la configuration C++ Intellisense.
- **WPILib C++ : Select Current C++ Toolchain** - Select the toolchain to use for Intellisense (i.e. desktop vs. roboRIO vs...). This is the same as clicking the current mode in the bottom right status bar.
- **WPILib C++ : Select Enabled C++ Intellisense Binary Types** - Switch Intellisense between static, shared, and executable
- **WPILib : Cancel currently running tasks** - Annule toutes les tâches que l'extension WPILib exécute actuellement
- **WPILib : Change Auto Save On Deploy Setting** - Modifie si les fichiers sont enregistrés automatiquement ou non lors d'un déploiement. La valeur par défaut est Enabled..
- **WPILib : Change Auto Start RioLog on Deploy Setting** - Modifie si RioLog démarre automatiquement ou non lors du déploiement. La valeur par défaut est Enabled.
- **WPILib : Change Desktop Support Enabled Setting** - Modifie si la création de code robot sur le Bureau est activée. Activez ce paramètre à des fins de test et de simulation. Par défaut la prise en charge du bureau n'est pas activée.
- **WPILib : Change Language Setting** - Cette commande permet de modifier le langage du projet actuellement ouvert s'il est C++ ou Java.
- **WPILib : Change Run Commands Except Deploy/Debug in Offline Mode Setting** - Modifier si GradleRIO s'exécute en mode en ligne pour les commandes autres que déployer/déboguer (tentera d'extraire automatiquement les dépendances en ligne). Par défaut, activé (mode en ligne).
- **WPILib : Change Run Deploy/Debug Command in Offline Mode Setting** - Change si GradleRIO s'exécute en mode en ligne pour le déploiement/débogage (tente d'extraire automatiquement les dépendances en ligne). Par défaut, désactivé (mode hors ligne).
- **WPILib : Change Select Default Simulate Extension Setting** - Change si les extensions de simulation sont activées par défaut (toutes les extensions de simulation définies dans `build.gradle` seront activées)
- **WPILib : Change Skip Tests On Deploy Setting** - Cette commande permet d'ignorer ou non les tests lors du déploiement. Par défaut, ce paramètre est configuré à *Disabled* (les tests sont exécutés sur le déploiement)
- **WPILib : Change Stop Simulation on Entry Setting** - Configure l'arrêt ou non du code robot à l'entrée lors de l'exécution de la simulation. Par défaut, le paramètre est configuré sur Disabled (ne vous arrêtez pas à l'entrée).
- **WPILib : Change Use WinDbg Preview (From Store) as Windows Debugger Setting** - Change whether to use the VS Code debugger or WinDbg Preview (from Windows Store).
- **WPILib : Check for WPILib Updates** - Check for an update to the WPILib GradleRIO version for the project. This does not update the Visual Studio Code extension, tools, or offline dependencies. Users are strongly recommended to use the *offline wpilib installer*
- **WPILib : Debug Robot Code** - Permet de construire et déployer du code robot sur le roboRIO en mode débogage et commencer le débogage
- **WPILib : Deploy Robot Code** - Permet de construire et déployer du code robot pour

roboRIO

- **WPILib : Hardware Sim Robot Code** - This builds the current robot code project on your PC and starts it running in simulation using hardware attached to the comupter rather then pure software simulation. Requires vendor support.
- **WPILib : Import a WPILib 2020-2023 Gradle Project** - Open a wizard to help you create a new project from a existing VS Code Gradle project from 2020-2022. Further documentation is at *importing gradle project*
- **WPILib : Install tools from GradleRIO** - Installe les outils Java WPILib (p. ex. SmartDashboard, Shuffleboard, etc.). Notez que cela se fait par défaut par l'installateur hors connexion
- **WPILib : Manage Vendor Libraries** - Installer/mettre à jour des bibliothèques tierces
- **WPILib : Open API Documentation** - Ouvre la documentation WPILib Javadocs ou C++ Doxygen
- **WPILib : Open Project Information** - Ouvre un widget avec des informations de projet (version project, version d'extension, etc.)
- **WPILib : Open WPILib Command Palette** - Cette commande est utilisée pour ouvrir une WPILib Command Palette (équivalent de taper `Ctrl+Shift+P` et de taper `WPILib`)
- **WPILib : Open WPILib Help** - Cette commande ouvre une page simple qui se connecte à la documentation WPILib (ce site)
- **WPILib : Reset Ask for WPILib Updates Flag** - Cette commande effacera l'indicateur du projet en cours, ce qui vous permettra de rélancer l'invite afin de mettre à jour un projet à la dernière version WPILib si vous avez précédemment choisi de ne pas mettre à jour.
- **WPILib : Run a command in Gradle** - Cette commande vous permet d'exécuter une commande arbitraire dans l'environnement de commande GradleRIO
- **WPILib : Set Team Number** - Cette commande est utilisée pour modifier le numéro d'équipe associé à un projet. Cette opération n'est nécessaire que si vous devez modifier le numéro d'équipe de celui initialement spécifié lors de la création du projet.
- **WPILib : Set VS Code Java Home to FRC Home** - Cette commande configure la variable VS Code Java Home pour pointer vers l'emplacement de Java découverte par l'extension FRC. Cette opération est nécessaire si vous n'utilisez pas l'installateur hors connexion pour vous assurer que les paramètres intellisense sont synchronisés avec les paramètres de construction de WPILib.
- **WPILib : Show Log Folder** - Affiche le dossier où l'extension WPILib stocke les journaux internes. Cela peut être utile lors du débogage/signalement d'un problème d'extension aux développeurs WPILib
- **WPILib : Simulate Robot Code** - This builds the current robot code project on your PC and starts it running in simulation. This requires Desktop Support to be set to Enabled.
- **WPILib : Start RioLog** - Cette commande démarre l'affichage RioLog utilisé pour afficher la sortie de la console à partir d'un programme de robot
- **WPILib : Start Tool** - Cette commande vous permet de lancer des outils WPILib (par exemple SmartDashboard, Shuffleboard, etc.) à partir de VS Code
- **WPILib : Test Robot Code** - Cette commande construit le projet actuel de code robot et exécute tous les tests créés. Cette opération nécessite que Desktop Support soit configuré sur Enabled.

10.3 Créer le programme du robot

Une fois que tout est installé, nous sommes prêts à créer un programme pour le robot. WPILib fournit plusieurs modèles de programmes de robot. L'utilisation de ces modèles est fortement recommandée pour les nouveaux utilisateurs ; cependant, les utilisateurs avancés sont libres d'écrire leur propre code de robot à partir de zéro.

10.3.1 Choix d'une classe de base

Pour démarrer un projet à l'aide d'un des modèles WPILib de programme de robot, les utilisateurs doivent d'abord choisir une classe de base pour leur robot. Les utilisateurs sous-classent ces classes de base pour créer leur classe primaire `Robot`, qui contrôle le flux principal du programme du robot. Trois choix sont disponibles pour la classe de base :

TimedRobot

Documentation : Java - C++

Source : Java - C++

The `TimedRobot` class is the base class recommended for most users. It provides control of the robot program through a collection of `init()`, `periodic()`, and `exit()` methods, which are called by WPILib during specific robot states (e.g. autonomous or teleoperated). During these calls, your code typically polls each input device and acts according to the data it receives. For instance, you would typically determine the position of the joystick and state of the joystick buttons on each call and act accordingly. The `TimedRobot` class also provides an example of retrieving autonomous routines through SendableChooser (Java/ C++

Note : Un modèle *TimedRobot Skeleton* est disponible ; celui-ci supprime certains commentaires informatifs et l'exemple autonome. Vous pouvez l'utiliser si vous êtes déjà familier avec *TimedRobot*. L'exemple ci-dessous est de *TimedRobot Skeleton*.

JAVA

```
7  import edu.wpi.first.wpilibj.TimedRobot;
8
9  /**
10  * The VM is configured to automatically run this class, and to call the functions␣
   ↪corresponding to
11  * each mode, as described in the TimedRobot documentation. If you change the name of␣
   ↪this class or
12  * the package after creating this project, you must also update the build.gradle␣
   ↪file in the
13  * project.
14  */
15 public class Robot extends TimedRobot {
16   /**
17    * This function is run when the robot is first started up and should be used for␣
   ↪any
```

(suite sur la page suivante)

(suite de la page précédente)

```
   * initialization code.
   */
  @Override
  public void robotInit() {}

  @Override
  public void robotPeriodic() {}

  @Override
  public void autonomousInit() {}

  @Override
  public void autonomousPeriodic() {}

  @Override
  public void teleopInit() {}

  @Override
  public void teleopPeriodic() {}

  @Override
  public void disabledInit() {}

  @Override
  public void disabledPeriodic() {}

  @Override
  public void testInit() {}

  @Override
  public void testPeriodic() {}

  @Override
  public void simulationInit() {}

  @Override
  public void simulationPeriodic() {}
}
```

C++

```
#include "Robot.h"

void Robot::RobotInit() {}
void Robot::RobotPeriodic() {}

void Robot::AutonomousInit() {}
void Robot::AutonomousPeriodic() {}

void Robot::TeleopInit() {}
void Robot::TeleopPeriodic() {}

void Robot::DisabledInit() {}
void Robot::DisabledPeriodic() {}
```

(suite sur la page suivante)

(suite de la page précédente)

```

void Robot::TestInit() {}
void Robot::TestPeriodic() {}

void Robot::SimulationInit() {}
void Robot::SimulationPeriodic() {}

#ifndef RUNNING_FRC_TESTS
int main() {
  return frc::StartRobot<Robot>();
}
#endif
```

Les méthodes périodiques sont appelées toutes les 20 ms par défaut. Cela peut être changé en sollicitant le constructeur de superclasse avec le taux de mise à jour souhaité.

Danger : Changing your robot rate can cause some unintended behavior (loop overruns). Teams can also use Notifiers to schedule methods at a custom rate.

JAVA

```
public Robot() {
  super(0.03); // Periodic methods will now be called every 30 ms.
}
```

C++

```
Robot() : frc::TimedRobot(30_ms) {}
```

RobotBase

Documentation : Java - C++

Source : Java - C++

La classe `RobotBase` est la classe de base la plus simpliste offerte et n'est généralement pas recommandée pour une utilisation directe. Aucun flux de contrôle du robot n'est géré pour l'utilisateur ; tout doit être écrit à partir de zéro dans la méthode `startCompetition()`. Le modèle par défaut montre comment traiter les différents modes de fonctionnement (teleop, auto, etc.).

Note : Un modèle `RobotBase Skeleton` est disponible qui offre une méthode `startCompetition()` vide de code

---

Commande Robot

Le framework `Command Robot` ajoute à la fonctionnalité de base d'un `Timed Robot` en interrogeant automatiquement les entrées et en convertissant les données d'entrée brutes en événements. Ces événements sont liés au code utilisateur, qui est exécuté lorsque l'événement est déclenché. Par exemple, lorsqu'un bouton est enfoncé, le code lié à l'appui sur ce bouton est automatiquement appelé et il n'est pas nécessaire d'interroger ou de suivre directement l'état de ce bouton. Le framework `Command Robot` facilite l'écriture de code compact facile à lire avec un comportement complexe, mais nécessite un investissement initial supplémentaire en temps de la part d'un programmeur afin de comprendre le fonctionnement du framework Command Robot.

Les équipes utilisant `Commande Robot` devraient se référer à *Dictaticiel sur la programmation orientée commande*.

Romi

Les équipes utilisant un *Romi* devraient utiliser le modèle `Romi - Timed` ou `Romi - Command Bot`

Romi - Timed

Le modèle `Romi - Timed` fournit une classe `RomiDrivetrain` qui expose une méthode `arcadeDrive(double xaxisSpeed, double zaxisRotate)`. C'est à l'utilisateur d'alimenter cette fonction arcadeDrive.

Cette classe fournit également des fonctions pour récupérer et réinitialiser les encodeurs embarqués du Romi.

Romi - Command Bot

Le modèle `Romi - Command Bot` fournit un sous-système `RomiDrivetrain` qui expose une méthode `arcadeDrive(double xaxisSpeed, double zaxisRotate)`. C'est à l'utilisateur d'alimenter cette fonction arcadeDrive.

Ce sous-système fournit également des fonctions pour récupérer et réinitialiser les codeurs embarqués du Romi.

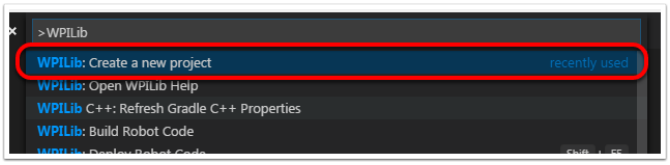
Choisir de ne pas utiliser une classe de base

Les utilisateurs peuvent omettre complètement la classe de base et simplement écrire leur programme dans une méthode `main()`, comme ils le feraient pour tout autre programme. Ceci est *fortement* déconseillé - les utilisateurs n'ont pas à « réinventer la roue » lors de l'écriture du code de leur robot - mais ceci est néanmoins pris en charge pour ceux qui souhaitent avoir un contrôle absolu sur le flux de leur programme.

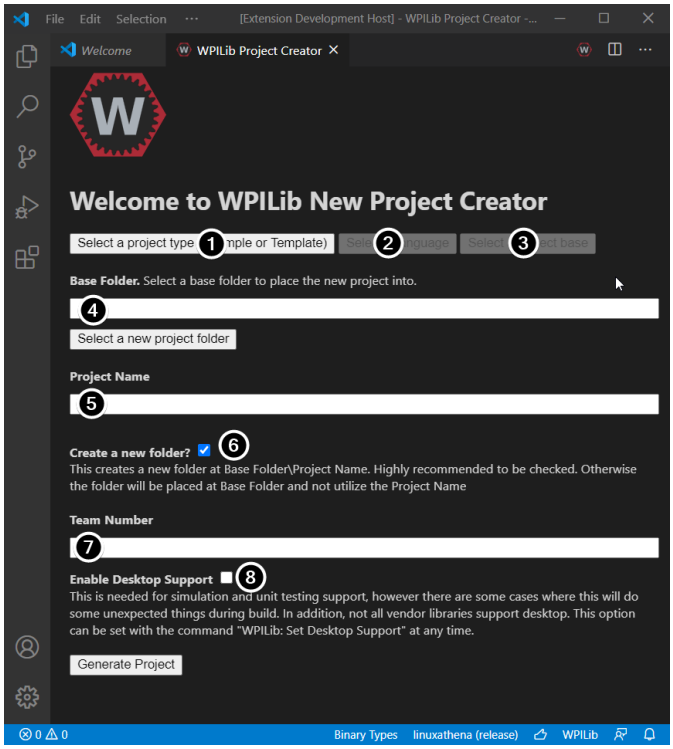
Avertissement : Les utilisateurs ne doivent *pas* modifier la méthode `main()` du programme du robot à moins qu'ils ne soient absolument sûrs de ce qu'ils font.

10.3.2 Créer un nouveau projet WPILib

Une fois que nous avons décidé d'une classe de base, nous pouvons créer notre nouveau projet de robot. Affichez la palette de commandes de Visual Studio Code avec `Ctrl+Shift+P`. Ensuite, tapez « WPILib » dans l'invite. Étant donné que toutes les commandes WPILib commencent par « WPILib », cela affichera la liste des commandes VS Code spécifiques à WPILib. Maintenant, sélectionnez la commande *Create a new project* :



La fenêtre « New Project Creator Window » apparaîtra.



Le contenu de la fenêtre « New Project Creator Window » est expliqué ci-dessous :

1. **Project Type** : Le genre de projet à créer. Il peut s'agir d'un exemple de projet ou de l'un des modèles de projet fournis par WPILib. Des modèles existent pour chacune des classes de base de robot. De plus, un modèle existe pour des projets *orientés commandes* qui sont rédigés à partir de la classe de base `TimedRobot` mais qui comprennent un certain nombre de fonctionnalités supplémentaires - ce type de programme de robot est fortement recommandé pour les nouvelles équipes.
2. **Language** : Le langage (C++ ou Java) qui sera utilisé pour ce projet.
3. **Base Folder** : S'il s'agit d'un modèle de projet, ce paramètre spécifie le type de modèle qui sera utilisé.
4. **Project Location** : Le dossier dans lequel le projet du robot sera enregistré.
5. **Project Name** : Le nom du projet du robot. Cela spécifie également le nom du dossier du projet si la case Create New Folder est cochée.

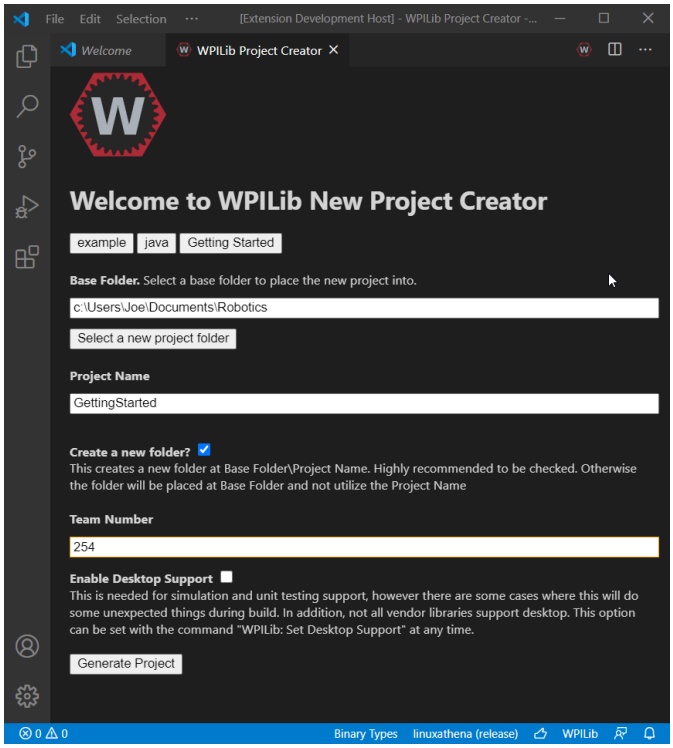
6. **Create a New Folder** : Si cette case est cochée, un nouveau dossier sera créé pour contenir le projet dans le dossier précédemment spécifié. Si elle n'est *pas* cochée, le projet sera situé directement dans le dossier précédemment spécifié. Une erreur sera générée si le dossier n'est pas vide et que cela n'est pas vérifié.
7. **Team Number** : Numéro d'équipe du projet, qui sera utilisé pour les noms de packages dans le projet et pour localiser le robot lors du déploiement du code.
8. **Enable Desktop Support** : Permet les tests unitaires et la simulation. Bien que WPILib les prennent en charge, il est possible que les librairies tierces ne puissent pas le faire. Si les librairies ne supportent pas le l'application bureau, votre code peut ne pas compiler ou peut se planter. Ce paramètre ne doit pas être coché à moins que des tests unitaires ou des simulations ne soient nécessaires et que toutes les librairies le prennent en charge.

Une fois que tout ce qui précède a été configuré, cliquez sur « Generate Project » et le projet du robot sera créé.

Note : Les erreurs dans la génération du projet apparaîtront dans le coin inférieur droit de l'écran.

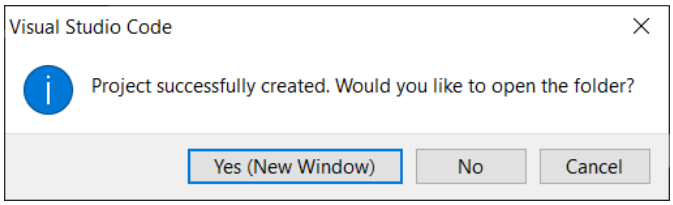
Avertissement : Creating projects on OneDrive is not supported as OneDrive's caching interferes with the build system. Some Windows installations put the Documents and Desktop folders on OneDrive by default.

Un exemple complet est illustré ci-dessous.

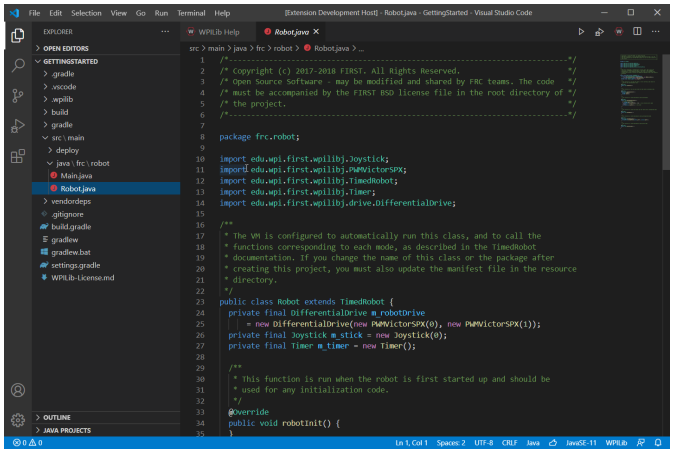


10.3.3 Ouvrir le nouveau projet

Après avoir créé votre projet avec succès, VS Code vous donnera la possibilité d'ouvrir le projet comme indiqué ci-dessous. Nous pouvons choisir de le faire maintenant ou plus tard en tapant `Ctrl+K` puis `Ctrl+O` (ou simplement `Command+O` sous macOS) et sélectionnez le dossier dans lequel nous avons enregistré notre projet.

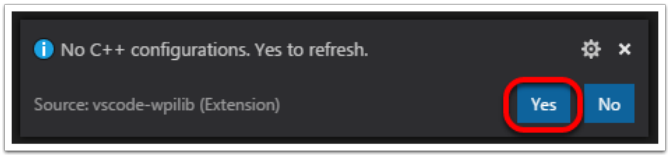


Une fois ouvert, la hiérarchie du projet s'affiche sur la gauche. Double-cliquez sur le fichier pour l'ouvrir dans l'éditeur.



10.3.4 Configurations C++ (C++ seulement)

Pour les projets C++, il y a une étape de plus pour configurer IntelliSense. Chaque fois que nous ouvrons un projet, nous devrions obtenir un pop-up dans le coin inférieur droit demandant de rafraîchir les configurations C++. Cliquez sur « Yes » pour configurer IntelliSense.



10.4 Librairies tierces

Teams that are using non-*PWM* motor controllers or advanced sensors will most likely need to install external vendor dependencies.

10.4.1 What Are Vendor Dependencies ?

A vendor dependency is a way for vendors such as CTRE, REV, and others to add their *software library* to robot projects. This library can interface with motor controllers and other devices. This way, teams can interact with their devices via CAN and have access to more complex and in-depth features than traditional PWM control.

10.4.2 Managing Vendor Dependencies

Vendor dependencies are installed on a per-project basis (so each robot project can have its own set of vendor dependencies). Vendor dependencies can be installed « online » or « offline ». The « online » functionality is done by downloading the dependencies over the internet, while offline is typically provided by a vendor-specific installer.

Avertissement : If installing a vendor dependency via the « online » mode, make sure to reconnect the computer to the internet and rebuild about every 30 days otherwise the cache will clear, completely deleting the downloaded library install.

Note : Vendors recommend using their offline installers when available, because the offline installer is typically bundled with additional programs that are extremely useful when working with their devices.

How Does It Work ?

How Does It Work ? - Java/C++

For Java and C++, a *JSON* file describing the vendor library is installed on your system to `~/wpilib/YYYY/vendordeps` (where YYYY is the year and `~` is `C:\Users\Public` on Windows). This can either be done by an offline installer or the file can be fetched from an online location using the menu item in Visual Studio Code. This file is then used from VS Code to add to the library to each individual project. Vendor library information is managed on a per-project basis to make sure that a project is always pointing to a consistent version of a given vendor library. The libraries themselves are placed in the Maven cache at `C:\Users\Public\wpilib\YYYY\maven`. Vendors can place a local copy here with an offline installer (recommended) or require users to be connected to the internet for an initial build to fetch the library from a remote Maven location.

This JSON file allows specification of complex libraries with multiple components (Java, C++, JNI, etc.) and also helps handle some complexities related to simulation. Vendors that choose to provide a remote URL in the JSON also enable users to check for updates from within VS Code.

How Does It Work ? - LabVIEW

For LabVIEW teams, there might be a few new *Third Party* items on various palettes (specifically, one in *Actuators*, one in *Actuators -> Motor Control* labeled *CAN Motor*, and one in *Sensors*). These correspond to folders in `C:\Program Files\National Instruments\LabVIEW 2023\vi.lib\Rock Robotics\WPI\Third Party`

In order to install third party libraries for LabVIEW, download the VIs from the vendor (typically via some sort of installer). Then drag and drop the third party VIs into the respective folder mentioned above just like any other VI.

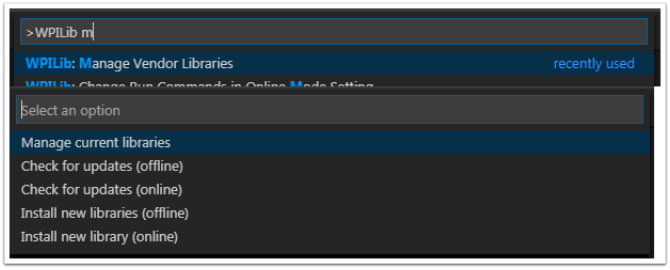
How Does It Work ? - Python

Third party libraries are packaged into Python wheels and uploaded to PyPI (if pure python) and/or WPILib's artifactory. Users can enable them as dependencies either by adding the component name to `robotpy_extras` (recommended) or by adding an explicit dependency for the PyPI package in `requires`. The dependencies are downloaded when `robotpy sync` is executed, and installed on the roboRIO when `robotpy deploy` is executed.

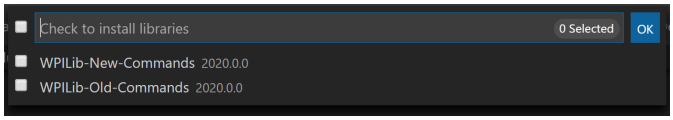
Installing Libraries

Java/C++

VS Code



Pour ajouter une librairie d'un fournisseur qui a été installée par un installateur hors ligne, appuyez sur `Ctrl+Shift+P` et tapez WPILib ou cliquez sur l'icône WPILib en haut à droite pour ouvrir la palette de commande WPILib et commencer à taper *Manage Vendor Libraries*, puis sélectionnez-la dans le menu. Sélectionnez l'option *Install new libraries (offline)*.



Sélectionnez les librairies que l'on souhaite ajouter au projet en cochant la case à côté de chacune, puis cliquez sur *OK*. Le fichier JSON sera copié dans le dossier `vendordeps` dans le projet, ajoutant la librairie comme une dépendance au projet.

In order to install a vendor library in online mode, press `Ctrl+Shift+P` and type WPILib or click on the WPILib icon in the top right to open the WPILib Command Palette and begin typing *Manage Vendor Libraries* and select it in the menu, and then click on *Install new libraries (online)* instead and copy + paste the vendor JSON URL.

Checking for Updates (Offline)

Since dependencies are version managed on a per-project basis, even when installed offline, you will need to *Manage Vendor Libraries* and select *Check for updates (offline)* for each project you wish to update.

Checking for Updates (Online)

Une partie du fichier JSON que les fournisseurs peuvent optionnellement remplir est un emplacement de mise à jour en ligne. Si une librairie a un emplacement approprié spécifié, l'exécution de *Check for updates (online)* vérifiera si une nouvelle version de la librairie est disponible à partir de l'emplacement distant.

Removing a Library Dependency

Pour supprimer une dépendance d'un projet à une librairie, sélectionnez *Manage Current Libraries* du menu *Manage Vendor Libraries*, cochez les cases correspondant à toutes les librairies à désinstaller et cliquer sur *OK*. Ces librairies seront supprimées en tant que dépendances du projet.

Command-Line

L'ajout d'une dépendance d'une librairie tierce à partir de l'URL du fournisseur peut également se faire à travers la ligne de commande via une tâche gradle. Ouvrez une instance de ligne de commande à la racine du projet, puis entrez `gradlew vendordep --url=<url>` où `<url>` est l'URL JSON du fournisseur. Cette ajoutera le fichier JSON de dépendance de la librairie des fournisseurs au dossier `vendordeps` du projet. Les librairies des fournisseurs peuvent être mises à jour de la même façon.

The `vendordep` gradle task can also fetch vendordep JSONs from the user `wpilib` folder. To do so, pass `FRCLOCAL/Filename.json` as the file URL. For example, `gradlew vendordep --url=FRCLOCAL/WPILibNewCommands.json` will fetch the JSON for the command-based framework.

Python

All RobotPy project dependencies are specified in `pyproject.toml`. You can add additional vendor-specific dependencies either by :

— Adding the component name to `robotpy_extras`
— Adding the PyPI package name to `requires`

Voir aussi :

pyproject.toml usage

10.4.3 Librairies

WPILib Libraries

Command Library

The WPILib *command library* has been split into a vendor library. It is installed by the WPILib installer for offline installation.

Java/C++

Librairie de la nouvelle architecture Command

Python

— PyPI package : `robotpy[commands2]` or `robotpy-commands-v2`
— In `pyproject.toml` : `robotpy_extras = ["commands2"]`

Librairy Romi

A Romi Library has been created to contain several helper classes that are used in the `Romi-Reference` example.

Java/C++

Romi Vendordep.

Python

— PyPI package : `robotpy[romi]` or `robotpy-romi`
— In `pyproject.toml` : `robotpy_extras = ["romi"]`

XRP Library

An XRP Library has been created to contain several helper classes that are used in the `XRPReference` example.

Java/C++

XRP Vendordep.

Python

— PyPI package : `robotpy[xrp]` or `robotpy-xrp`
— In `pyproject.toml` : `robotpy_extras = ["xrp"]`

Vendor Libraries

Click these links to visit the vendor site to see whether they offer online installers, offline installers, or both. URLs below are to plug in to the *VS Code -> Install New Libraries (online)* feature.

CTRE Phoenix Framework - Contains CANcoder, CANifier, CANdle, Pigeon IMU, Pigeon 2.0, Talon FX, Talon SRX, and Victor SPX Libraries and Phoenix Tuner program for configuring CTRE CAN devices

Java/C++

Phoenix (v6) : `https://maven.ctr-electronics.com/release/com/ctre/phoenix6/latest/Phoenix6-frc2024-latest.json`

Phoenix (v5) : `https://maven.ctr-electronics.com/release/com/ctre/phoenix/Phoenix5-frc2024-latest.json`

Note : All users should use the Phoenix (v6) library. If you also need Phoenix v5 support, additionally install the v5 vendor library.

Python

Vendor's package :
- PyPI package : `robotpy[phoenix6]` or `phoenix6`
- In `pyproject.toml` : `robotpy_extras = ["phoenix6"]`

Community packages :
- PyPI package : `robotpy[phoenix5]` or `robotpy-ctre`
- In `pyproject.toml` : `robotpy_extras = ["phoenix5"]`

Redux Robotics ReduxLib - Library for all Redux devices including the Canandcoder and Canandcolor

Java/C++

`https://frcsdk.reduxrobotics.com/ReduxLib_2024.json`

Python

Not yet available

Playing With Fusion Driver - Librairie pour tous les équipements PWF, dont le moteur/contrôleur Venom

Java/C++

```
https://www.playingwithfusion.com/frc/playingwithfusion2024.json
```

Python

Community-supported packages :
- PyPI package : `robotpy[playingwithfusion]` or `robotpy-playingwithfusion`
- In `pyproject.toml` : `robotpy_extras = ["playingwithfusion"]`

Kauai Labs - Librairies pour NavX-MXP, NavX-Micro, et Sensor Fusion

Java/C++

```
https://dev.studica.com/releases/2024/NavX.json
```

Python

Community-supported packages :
- PyPI package : `robotpy[navx]` or `robotpy-navx`
- In `pyproject.toml` : `robotpy_extras = ["navx"]`

REV Robotics REVLib - Library for all REV devices including SPARK Flex, SPARK MAX, and Color Sensor V3

Java/C++

```
https://software-metadata.revrobotics.com/REVLib-2024.json
```

Python

Community-supported packages :
- PyPI package : `robotpy[rev]` or `robotpy-rev`
- In `pyproject.toml` : `robotpy_extras = ["rev"]`

Librairies communautaires

Librairie du logiciel PhotonVision CV PhotonVision

Java/C++

```
https://maven.photonvision.org/repository/internal/org/photonvision/
photonlib-json/1.0/photonlib-json-1.0.json
```

Python

— PyPI package : `photonlibpy`
— In `pyproject.toml` : `requires = ["photonlibpy"]`

PathPlanner - Library for PathPlanner

Java/C++

```
https://3015rangerrobotics.github.io/pathplannerlib/PathplannerLib.json
```

Python

— PyPI package : `pathplannerlib`
— In `pyproject.toml` : `requires = ["pathplannerlib"]`

ChoreoLib - Library for reading and following trajectories generated by Choreo

Java/C++

```
https://sleipnirgroup.github.io/ChoreoLib/dep/ChoreoLib.json
```

Python

Not available

YAGSL - Library for Swerve Drives of any configuration

Java

```
https://broncbotz3481.github.io/YAGSL-Lib/yagsl/yagsl.json
```

Python

Not available

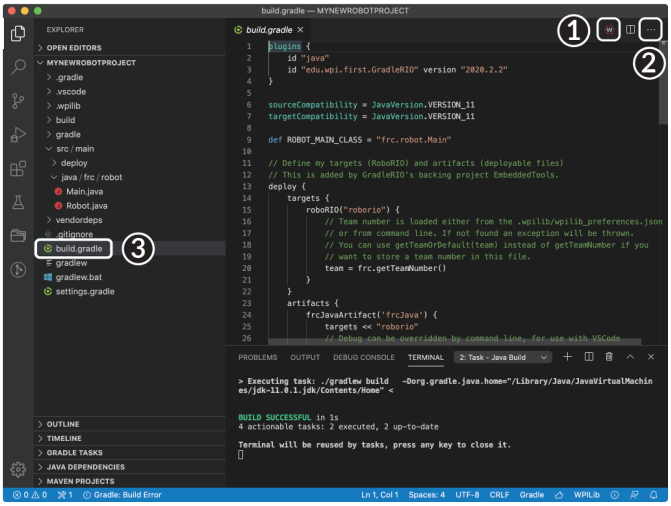
10.5 Compiler et Télécharger le Code du Robot

Les projets de robot doivent être compilés (« construits ») et téléchargés pour être exécutés sur le roboRIO. Puisque le code n'est pas compilé nativement sur le contrôleur du robot, c'est ce qu'on appelle la « compilation croisée ».

Pour compiler et télécharger un projet de robot, effectuez l'une des opérations suivantes :

1. Ouvrez la pallette *Command Pallette* et entrez/selectionnez « Build Robot Code »
2. Ouvrez le menu des raccourcis indiqué par les ellipses en haut à droite du fenêtre VS Code et selectionnez « Build Robot Code »

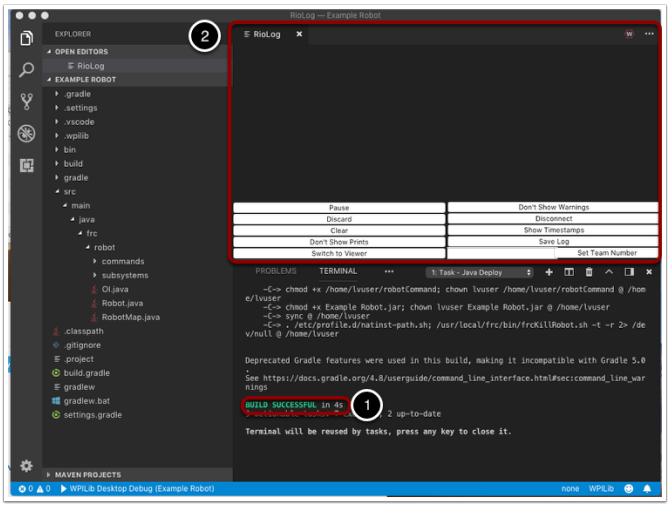
3. Cliquez avec le bouton droit sur le fichier build.gradle dans la hiérarchie du projet et selectionnez « Build Robot Code »



Deploy robot code by selecting « Deploy Robot Code » from any of the three locations from the previous instructions. That will build (if necessary) and deploy the robot program to the roboRIO.

Avertissement : Avoid powering off the robot while deploying robot code. Interrupting the deployment process can corrupt the roboRIO filesystem and prevent your code from working until the roboRIO is *re-imaged*.

If successful, we will see a « Build Successful » message (1) and the RioLog will open with the console output from the robot program as it runs (2).



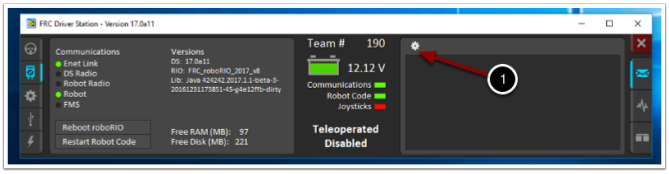
10.6 Affichage de la sortie de la console

Pour l'affichage de la sortie console des programmes basés sur du texte, le roboRIO implémente une NetConsole. Il existe deux façons principales d'afficher la sortie NetConsole du roboRIO : le Console Viewer dans le logiciel Driver Station FRC et l'extension Riolog dans VS Code.

Note : Sur le roboRIO, la NetConsole est uniquement pour les sorties du programme. Si vous souhaitez interagir avec la console système, vous devrez utiliser SSH ou la console série.

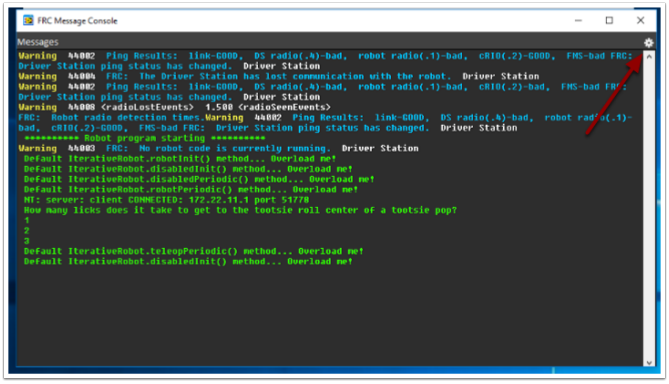
10.6.1 Console Viewer

Utiliser Console Viewer



Pour ouvrir la visionneuse de console, ouvrez d'abord le FRC® Driver Station. Ensuite, cliquez sur l'engrenage en haut de la fenêtre de l'afficheur de messages (1) et sélectionnez « View Console ».

La fenêtre Console Viewer

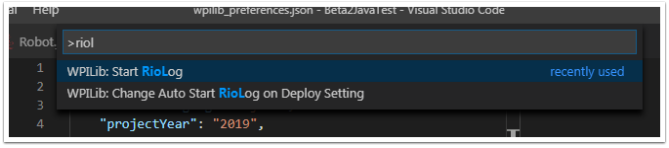


La fenêtre Console Viewer affiche les sorties du programme du robot en vert. L'engrenage en haut à droite permet d'effacer la fenêtre et de définir le niveau des messages affichés.

10.6.2 RioLog versus Code Plugin

Le plugin RioLog est une vue VS Code qui peut être utilisée pour visualiser la sortie de NetConsole dans VS Code (crédit pour la version originale Eclipse : Manuel Stoeckl, FRC1511).

Ouvrir la vue RioLog



Par défaut, la vue RioLog s'ouvre automatiquement à la fin de chaque déploiement de roboRIO. Pour lancer la vue RioLog manuellement, appuyez sur `Ctrl+Shift+P` pour ouvrir la palette de commandes et commencez à taper « RioLog », puis sélectionnez l'option WPILib : Start RioLog.

La fenêtre RioLog

La vue RioLog doit apparaître dans le volet supérieur. Riolog contient un certain nombre de commandes pour manipuler la console :

- **Pause/Resume Display** - Ceci met en pause/reprend l'affichage. En arrière-plan, les nouveaux paquets seront toujours reçus et seront affichés lorsque le bouton de reprise sera cliqué.
- **Discard/Accept Incoming** - Ceci permet d'accepter ou non de nouveaux paquets. Lorsque les paquets sont rejetés, l'affichage est mis en pause et tous les paquets reçus sont rejetés. Cliquez à nouveau sur le bouton pour reprendre la réception des paquets.
- **Clear** - Ceci efface le contenu actuel de l'affichage.
- **Don't Show/Show Prints** - Ceci affiche ou masque les messages classés pour impression.
- **Switch to Viewer** - Ceci passe à la visionneuse pour les fichiers journaux enregistrés
- **Don't Show/Show Warnings** - Ceci affiche ou masque les messages classés comme des avertissements
- **Disconnect/Reconnect** - Ceci déconnecte ou se reconnecte au flux de la console
- **Show/Don't Show Timestamps** - Ceci affiche ou masque l'horodatage des messages dans la fenêtre

— **Save Log** - Ceci copie le contenu du journal dans un fichier que vous pouvez enregistrer, afficher ou ouvrir plus tard avec la visionneuse RioLog (voir Switch to Viewer ci-dessus)
— **Set Team Number** - Ceci définit le numéro d'équipe du roboRIO pour se connecter au flux de la console. Défini automatiquement si RioLog est lancé par le processus de déploiement

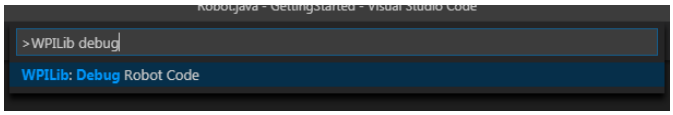
10.7 Débogage d'un programme de robot

Inévitablement, un programme ne se comportera pas de la façon dont nous nous attendons à ce qu'il se comporte. Lorsque cela se produit, il devient nécessaire de comprendre pourquoi le programme fait ce qu'il fait, afin que nous puissions lui faire faire ce que nous voulons qu'il fasse, à la place. Un tel comportement indésirable du programme est appelé « bogue », et ce processus est appelé « débogage ».

Un débogueur est un outil utilisé pour contrôler le flux du programme et surveiller les variables afin d'aider au débogage d'un programme. Cette section décrit comment configurer une session de débogage pour un programme de robot FRC®.

Note : Pour les utilisateurs débutants qui ont besoin de déboguer leurs programmes, mais ne savent pas/n'ont pas le temps d'apprendre à utiliser un débogueur, il est souvent possible de déboguer un programme simplement en imprimant l'état du programme pertinent à la console. Cependant, il est fortement recommandé que les élèves finissent par apprendre à utiliser un débogueur.

10.7.1 Exécution du Débogueur



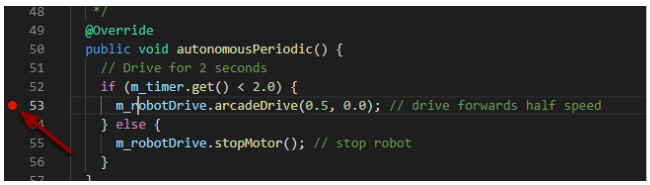
Appuyez sur `Ctrl+Shift+P` et tapez `WPILib` ou cliquez sur *WPILib Menu Item* pour ouvrir la palette de commandes avec WPILib pré-rempli. Tapez Debug et sélectionnez l'élément de menu Debug Robot Code pour démarrer le débogage. Le code sera téléchargé sur le roboRIO et commencera le débogage.

10.7.2 Points d'interruption

Un point d'interrogation ou « point d'arrêt » est une ligne de code à laquelle le débogueur suspendra l'exécution du programme afin que l'utilisateur puisse examiner l'état du programme. Ceci est extrêmement utile lors du débogage, car cela permet à l'utilisateur de suspendre le programme à des points spécifiques du code problématique pour déterminer où exactement le programme s'écarte du comportement attendu.

Le débogueur s'arrêtera automatiquement au premier point d'arrêt qu'il rencontre.

Configuration d'un point d'interruption



Cliquez dans la marge gauche de la fenêtre du code source (à gauche du numéro de ligne) pour définir un point d'arrêt dans votre programme utilisateur : un petit cercle rouge indique que le point d'arrêt a été défini sur la ligne correspondante.

10.7.3 Débogage avec des instructions d'impression

Une autre façon de déboguer votre programme consiste à utiliser des instructions d'impression dans votre code et à les afficher à l'aide du RioLog dans Visual Studio Code ou de la Driver Station. Les instructions imprimées doivent être ajoutées avec précaution car elles ne sont pas très efficaces, surtout lorsqu'elles sont utilisées en grande quantité. Ils doivent être supprimés pour la compétition car ils peuvent provoquer des dépassements de boucle.

JAVA

```
System.out.print("example");
```

C++

```
wpi::outs() << "example\n";
```

10.7.4 Débogage avec NetworkTables

Les *NetworkTables* peuvent être utilisées pour partager les informations du robot avec votre ordinateur de débogage. Les *NetworkTables* peuvent être visualisées avec votre tableau de bord préféré ou avec *OutlineViewer*. Un avantage de NetworkTables est que des outils comme *Shuffleboard* peuvent être utilisés pour analyser graphiquement les données. Ces mêmes outils peuvent ensuite être utilisés avec les mêmes données pour fournir ultérieurement une interface opérateur pour vos pilotes.

10.7.5 Apprendre encore plus

— Pour en savoir plus sur le débogage avec VS Code, consultez ce lien.
— Certaines des fonctionnalités mentionnées dans cet article VS Code vous aideront à comprendre et à diagnostiquer les problèmes liés à votre code. La fonction Quick Fix (ampoule jaune) peut être très utile avec une variété de problèmes, y compris ce qu'il faut importer.
— L'un des meilleurs moyens d'éviter d'avoir à déboguer autant de problèmes est de faire des tests unitaires.
— Vérifier que votre robot fonctionne dans Simulation est également un excellent moyen d'éviter d'avoir à effectuer un débogage complexe sur le robot réel.

10.8 Importing Last Year's Robot Code

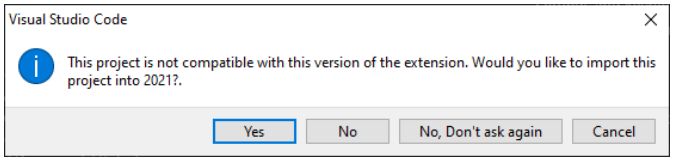
Due to changes in the project, it is necessary to update the build files for a previous years Gradle project. It is also necessary to import vendor libraries again, since last year's vendor libraries must be updated to be compatible with this year's projects.

10.8.1 Automatic Import

To make it easy for teams to import previous years gradle projects into the current year's framework, WPILib includes a wizard for importing previous years projects into VS Code. This will generate the necessary gradle components and load the project into VS Code. In place upgrades are not supported.

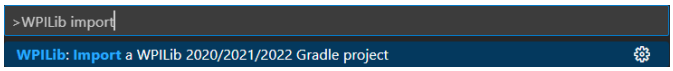
Important : The import process copies your project source files from the current directory to a new directory and completely regenerates the gradle files. Additionally, it updates the code for the package changes made in 2023. If you made non-standard updates to the `build.gradle`, you will need to make those changes again. For this reason, in place upgrades are not supported. It is also necessary to import vendor libraries again, since last year's vendor libraries must be updated to be compatible with this year's projects.

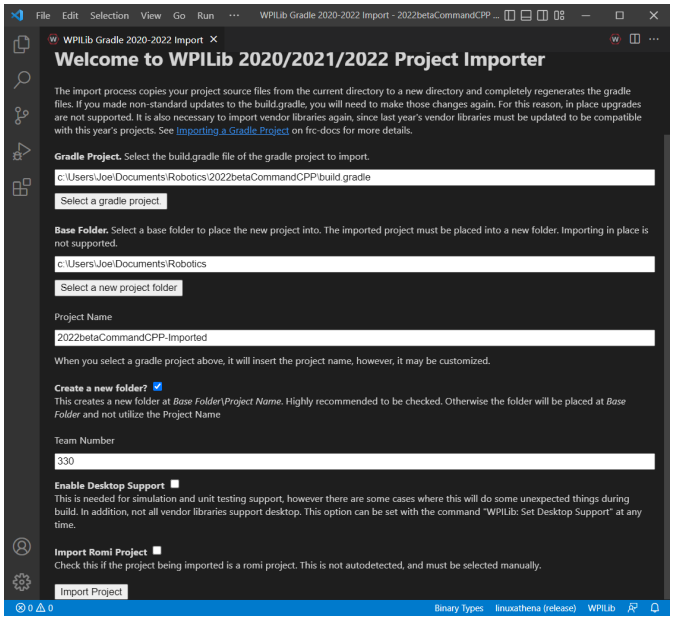
Launching the Import Wizard



When you open a previous year's project, you will be prompted to import that project. Click *yes*.

Alternately, you can chose to import it from the menu. Press `Ctrl+Shift+P` and type « WPILib » or click the WPILib icon to locate the WPILib commands. Begin typing « Import a WPILib 2020-2023 Gradle project » and select it from the dropdown as shown below.





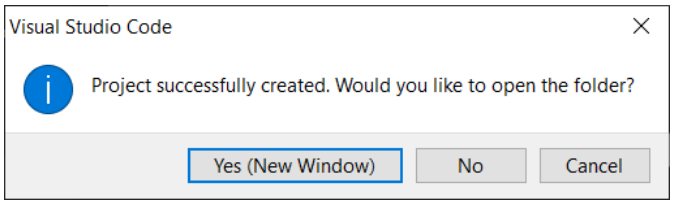
You'll be presented with the WPILib Project Importer window. This is similar to the process of creating a new project and the window and the steps are shown below. This window contains the following elements :

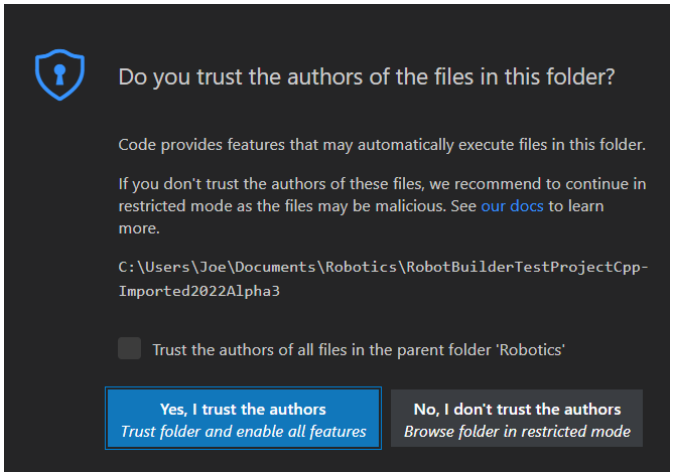
1. **Gradle Project** : Selects the project to be imported. Users should select the build.gradle file in the root directory of the gradle project.
2. **Project Location** : This determines the folder in which the robot project will be located.
3. **Project Name** : The name of the robot project. This also specifies the name that the project folder will be given if the Create New Folder box is checked. This must be a different directory from the original location.
4. **Create a New Folder** : If this is checked, a new folder will be created to hold the project within the previously-specified folder. If it is *not* checked, the project will be located directly in the previously-specified folder. An error will be thrown if the folder is not empty and this is not checked.
5. **Team Number** : The team number for the project, which will be used for package names within the project and to locate the robot when deploying code.
6. **Enable Desktop Support** : If this is checked, simulation and unit test support is enabled. However, there are some cases where this will do some unexpected things. In addition, all vendor libraries need desktop support which not all libraries do.
7. **Import Romi Project** : If this is checked, the project is imported using the Romi gradle template. This should only be checked for Romi projects.

Avertissement : Creating projects on OneDrive is not supported as OneDrive's caching interferes with the build system. Some Windows installations put the Documents and Desktop folders on OneDrive by default.

Click *Import Project* to begin the upgrade.

The gradle project will be upgraded and copied into the new project directory. You can then either open the new project immediately using the pop-up below or open it later using the `Ctrl+O` (or `Command+O` for macOS) shortcut.

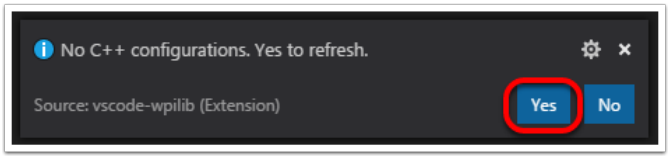




Click *Yes I trust the authors*.

C++ Configurations (C++ Only)

For C++ projects, there is one more step to set up IntelliSense. Whenever you open a project, you should get a pop-up in the bottom right corner asking to refresh C++ configurations. Click *Yes* to set up IntelliSense.



3rd Party Libraries

It is necessary to update and re-import 3rd party libraries. See *3rd Party Libraries* for details.

11 Dashboards

11.1 Choosing a Dashboard

A dashboard is a program used to retrieve and display information about the operation of your robot. There are two main types of dashboards that teams may need : driver and programmer dashboards. Some dashboards will try to accommodate both purposes.

11.1.1 Driver Dashboard

During competition the drive team will use this dashboard to get information from the robot. It should focus on conveying key information instantly. This is often best accomplished by using large, colorful, and easy to understand visual elements. Most teams will also use this dashboard to select their autonomous routine.

Take caution to carefully consider what *needs* to be on this dashboard and if there is another better way of communicating that information. Any members of the drive team (especially the driver) looking at the dashboard takes their focus away from the match. Using *LEDs* to indicate the state of your robot is a good example of a way to communicate useful information to the driver without having to take their eyes off the robot.

11.1.2 Programming Dashboard

This dashboard is designed for debugging code and analyzing data from the robot. It supports the monitoring of a wide variety of information simultaneously, prioritizing function and utility over simplicity or ease of use. This functionality often includes complex data visualization and graphing across extended periods. In scenarios where there is an overwhelming amount of data to review, real-time analysis becomes challenging. The capability to examine past data and replay it proves to be extremely beneficial. While some dashboards may log data transmitted to them, *on-robot telemetry* using the `DataLog` class simplifies the process.

11.1.3 Specific Dashboards (oldest to newest)

Note : SmartDashboard and Shuffleboard have a long history of aiding FRC teams. However, they do not have a person to maintain them so are not receiving bug fixes or improvements. Notably, Shuffleboard may experience performance issues on some machines under certain scenarios. PRs from external contributors will be reviewed.

LabVIEW Dashboard (Driver / Programming) - easy to use and provides a lot of features straight out of the box like : camera streams, autonomous selection, and joystick feedback. It can be customized using LabVIEW by creating a new Dashboard project. While it *can be used* by Java or C++ teams, they generally prefer SmartDashboard or Shuffleboard which can be customized in their respective language.

SmartDashboard (Driver) - simple and efficient dashboard that uses relatively few computer resources. It does not have the fancy look or some of the features Shuffleboard has, but it displays network tables data with a variety of widgets without bogging down the driver station computer.

Shuffleboard (Driver) - straightforward and easily customizable dashboard. It displays network tables data using a variety of widgets that can be positioned and controlled with robot code. It includes many extra features like : tabs, recording / playback, and advanced custom widgets.

Glass (Programming) - robot data visualization tool. Its GUI is extremely similar to that of the *Simulation GUI*. In its current state, it is meant to be used as a programmer's tool rather than a proper dashboard in a competition environment, with a focus on high performance real time plotting.

AdvantageScope (Programming) - robot diagnostics, log review/analysis, and data visualization application. It reads the WPILib Data Log (`.wpilog`) and Driver Station Log (`.dslog` / `.dsevents`) file formats, plus live robot data viewing.

11.1.4 Third Party Dashboards

FRC Web Components (Driver) - A web-based dashboard that can be installed as a standalone application, or as a JavaScript package for custom dashboard solutions.

Elastic (Driver) - simple and modern Shuffleboard alternative made by Team 353. It is meant to serve as a dashboard for competition but can also be used for testing. It features draggable and resizable card widgets.

QFRCDashboard (Driver) - described as reliable, high-performance, low-footprint dashboard. QFRCDashboard has been specifically designed to use as few resources as possible.

11.2 Shuffleboard

Shuffleboard is a straightforward and easily customizable driveteam focused dashboard. It displays network tables data using a variety of widgets that can be positioned and controlled with robot code. It includes many extra features like : tabs, recording / playback, and advanced custom widgets.

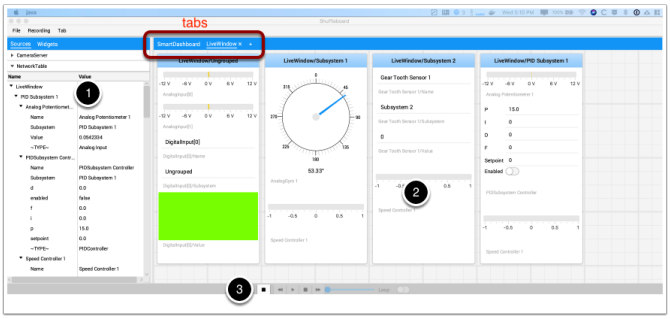
11.2.1 Shuffleboard-Démarrage

Aperçu de Shuffleboard

Shuffleboard is a dashboard for FRC® based on newer technologies such as JavaFX that are available to Java programs. It is designed to be used for creating dashboards for C++, Java, and Python programs. If you've used SmartDashboard in the past then you are already familiar with many of the features of Shuffleboard since they fundamentally work the same way. But Shuffleboard has many features that aren't in SmartDashboard. Here are some of the highlights :

- Les graphiques sont basés sur **JavaFX**, la norme des graphiques en Java. Chacune des composantes a une feuille de style associée de telle sorte qu'on peut avoir une variété de « skins » ou « thèmes » pour Shuffleboard. Les thèmes fournis par défaut sont le clair et le foncé.
- Shuffleboard prend en charge **plusieurs feuilles pour la visualisation de vos données**. En fait, vous pouvez créer une nouvelle feuille (se présentant sous la forme d'un onglet dans la fenêtre de Shuffleboard) et préciser si et quelles données doivent s'y insérer automatiquement. Par défaut, il y a un onglet Test et un onglet SmartDashboard qui se remplissent automatiquement à mesure que les données sont disponibles. D'autres onglets peuvent être ajoutés pour le débogage du robot et son pilotage.
- Les **éléments graphiques d'affichage (widgets) sont disposés sur une grille** pour maintenir l'interface soignée et facile à lire. Vous pouvez modifier la taille de la grille pour obtenir plus ou moins de précision dans vos dispositions et des repères visuels sont fournis pour vous aider à modifier votre disposition à l'aide de glisser-déposer. Vous pouvez également choisir de retirer les lignes de grille et, ce faisant, la disposition de la grille est quand même préservée.
- Les dispositions sont enregistrées et la version la plus récente est celle qui est activée par défaut lorsque vous ouvrez à nouveau Shuffleboard.
- Il existe une fonction **record and playback** qui vous permet d'examiner à postériori les données envoyées par le programme de votre robot. De cette façon, si quelque chose ne n'a pas fonctionné normalement, vous pouvez examiner minutieusement les actions du robot.
- **Des widgets graphiques sont disponibles pour les données numériques** et vous pouvez faire glisser ces données sur un graphique pour afficher simultanément plusieurs points et à la même échelle.
- You can extend Shuffleboard by writing your own widgets that are specific to your team's requirements. Documentation on extending it can be found in *Custom Widgets*.

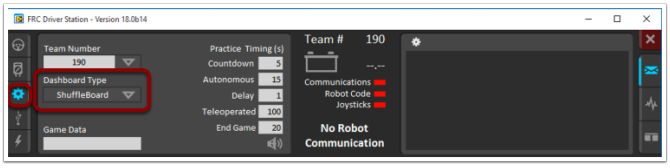
1. **Zone sources :** Dans cette zone se trouvent des sources à l'origine des données dont vous pouvez choisir les valeurs à partir de NetworkTables ou d'autres sources à afficher en faisant glisser une valeur dans l'un des onglets
2. **Volets d'onglets :** C'est ici que sont affichées les données en provenance du robot ou d'autres sources. Dans cet exemple, ce sont les sous-systèmes en mode Test qui s'affichent sous l'onglet LiveWindow. Dans cette zone, on peut afficher autant d'onglets



qu'on le souhaite, et chaque fenêtre possède son propre ensemble de propriétés comme la taille de la grille et la capacité de s'auto-remplir.

3. **Contrôles d'enregistrement/lecture :** Ensemble de contrôles de type média qui vous permet de rejouer la session sélectionnée pour voir les données enregistrées

Démarrage de Shuffleboard



Vous pouvez démarrer Shuffleboard de quatre façons :

1. Vous pouvez le lancer automatiquement lorsque Driver Station démarre en sélectionnant « Shuffleboard » dans « Dashboard Type » de l'onglet Configuration comme indiqué dans l'image ci-dessus.
2. You can run it by double-clicking the Shuffleboard icon in the *YEAR WPILib tools* folder on the Windows Desktop.
3. You can start from with Visual Studio Code by pressing `Ctrl+Shift+P` and type « WPILib » or click the WPILib logo in the top right to launch the WPILib Command Palette. Select *Start Tool*, then select *Shuffleboard*.
4. Vous pouvez l'exécuter en double-cliquant sur le fichier shuffleboard.XXX (où XXX est `.vbs` sous Windows et `.py` sous Linux ou macOS) dans `~/WPILib/YYYY/tools/` (où YYYY est l'année et `~` est `C:\Users\Public` sous Windows). Ceci est utile sous un système de développement où l'application Driver Station n'est pas installée tel que le système macOS ou Linux.
5. You can start it from the command line by typing the command : `shuffleboard` on Windows or `python shuffleboard.py` on macOS or Linux from `~/WPILib/YYYY/tools` directory (where YYYY is the year and `~` is `C:\Users\Public` on Windows). This is often easiest on a development system that doesn't have the Driver Station installed.

Note : Les scripts `.vbs` (Windows) et `.py` (macOS/Linux) aident à lancer les outilitaires à l'aide de la version JDK appropriée.

Obtenir des données du robot sur le tableau de bord

The easiest way to get data displayed on the dashboard is simply to use methods in the `SmartDashboard` class. For example to write a number to Shuffleboard write :

JAVA

```
SmartDashboard.putNumber("Joystick X value", joystick1.getX());
```

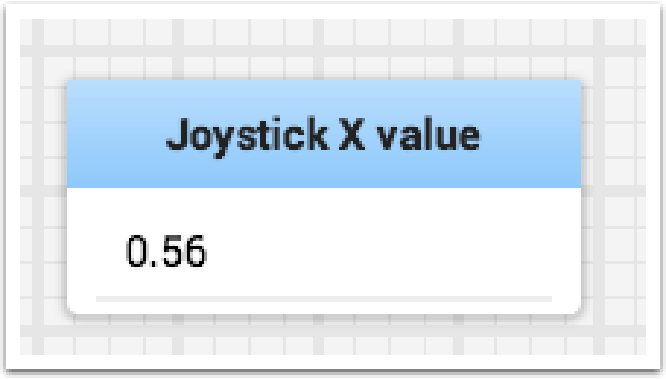
C++

```
frc::SmartDashboard::PutNumber("Joystick X value", joystick1.getX());
```

PYTHON

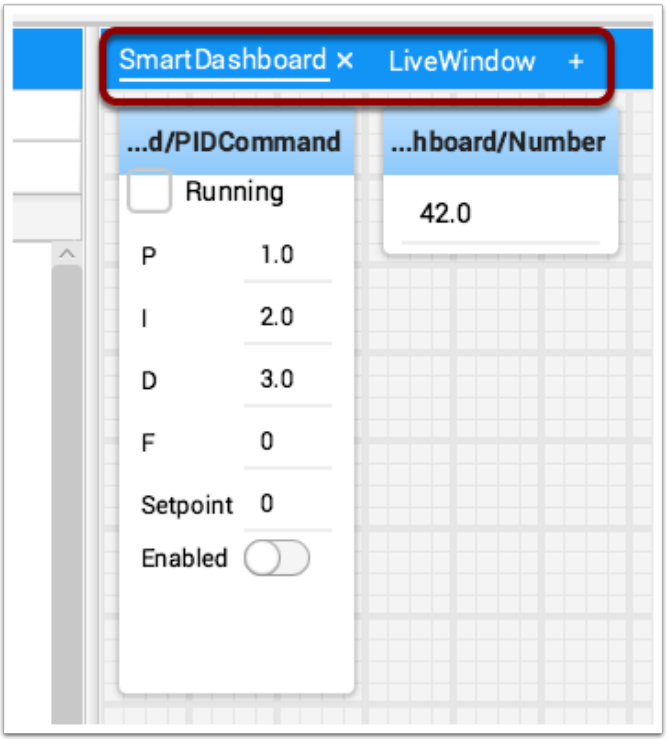
```
from wpilib import SmartDashboard

SmartDashboard.putNumber("Joystick X value", joystick1.getX())
```

pour afficher un champ dont l'étiquette est « Joystick X value » et la valeur actuelle de l'axe X de la manette. À chaque exécution de cette instruction, une nouvelle valeur de l'axe X de la manette est envoyée à Shuffleboard. N'oubliez pas : vous devez écrire la valeur en provenance de la manette à chaque fois que vous souhaitez voir une valeur mise à jour. L'exécution de l'instruction au début du programme n'affiche la valeur qu'une seule fois soit au moment de l'exécution de la ligne de code.



Affichage des données de votre robot

Vous pouvez afficher les données de votre robot dans les modes de fonctionnement réguliers Teleop et Autonome, mais vous pouvez également afficher l'état et activer tous les sous-systèmes du robot lorsque celui-ci est en mode Test. Par défaut, vous verrez deux onglets lorsque vous démarrez Shuffleboard, l'un pour Teleop/Autonomous et l'autre pour le mode Test. L'onglet actif est souligné, comme on peut le voir dans l'image ci-dessous.



Souvent, le débogage ou l'inspection de l'état d'un robot demande d'écrire un certain nombre de valeurs à la console et de les regarder par streaming. À l'aide de Shuffleboard, vous pouvez ajouter des valeurs par une interface générée automatiquement selon votre programme. Au fur et à mesure que les valeurs sont mises à jour, les éléments du GUI associés à ces valeurs s'adaptent - il n'est pas nécessaire d'essayer d'envoyer ces chiffres par streaming à l'écran.

Affichage des valeurs en mode de fonctionnement normal (Auto ou Teleop)

JAVA

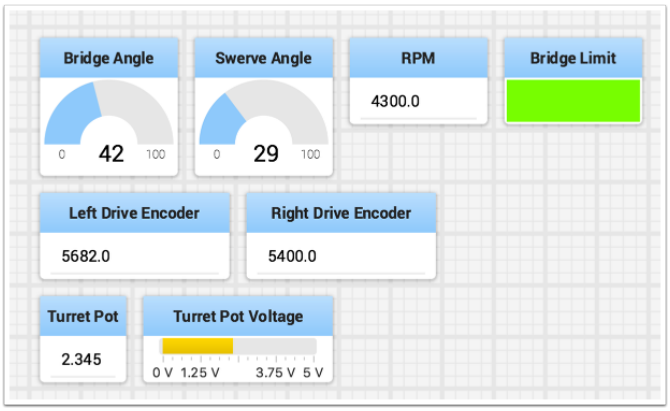
```
protected void execute() {
   SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge());
   SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition());
   SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle());
   SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder());
   SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder());
   SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle());
   SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage());
   SmartDashboard.putNumber("RPM", shooter.getRPM());
}
```

C++

```
frc::SmartDashboard::PutBoolean("Bridge Limit", bridgeTipper.AtBridge());
frc::SmartDashboard::PutNumber("Bridge Angle", bridgeTipper.GetPosition());
frc::SmartDashboard::PutNumber("Swerve Angle", drivetrain.GetSwerveAngle());
frc::SmartDashboard::PutNumber("Left Drive Encoder", drivetrain.GetLeftEncoder());
frc::SmartDashboard::PutNumber("Right Drive Encoder", drivetrain.GetRightEncoder());
frc::SmartDashboard::PutNumber("Turret Pot", turret.GetCurrentAngle());
frc::SmartDashboard::PutNumber("Turret Pot Voltage", turret.GetAverageVoltage());
frc::SmartDashboard::PutNumber("RPM", shooter.GetRPM());
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge())
SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition())
SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle())
SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder())
SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder())
SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle())
SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage())
SmartDashboard.putNumber("RPM", shooter.getRPM())
```

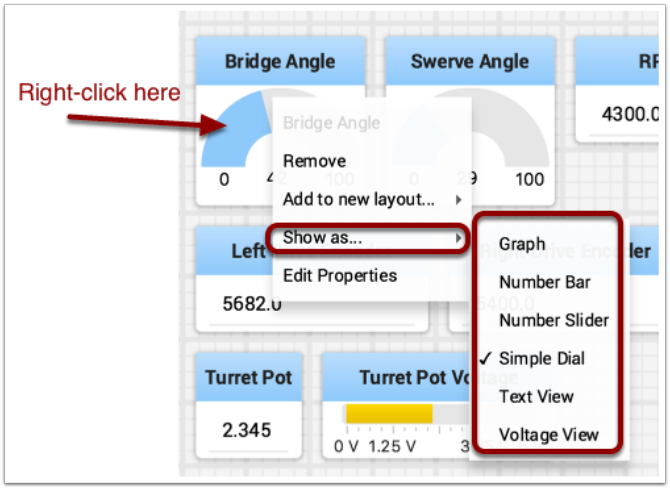


Vous pouvez écrire des valeurs booléennes, numériques ou des chaînes de caractères dans Shuffleboard en invoquant simplement la méthode appropriée au type et en précisant le nom et la valeur des données, aucun code supplémentaire n'est requis.

— Les types numériques tels que char, int, long, float ou double invoquent la méthode SmartDashboard.putNumber(« dashboard-name », value).
— Les types String invoquent la méthode SmartDashboard.putString(« dashboard-name », value)
— Les types booléens invoquent la méthode SmartDashboard.putBoolean(« dashboard-name », value)

Modification du type d'affichage des données

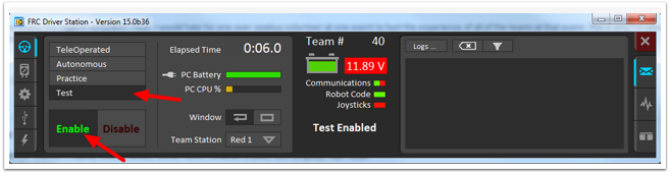
Selon le type de données des valeurs envoyées à Shuffleboard, vous pouvez parfois modifier le format d'affichage. Dans l'exemple précédent, vous voyez que les valeurs numériques ont été affichées sous forme de nombres décimaux, de cadrans pour représenter des angles et une représentation de la tension pour le potentiomètre. Pour définir le type d'affichage, cliquez avec le bouton droit sur la vignette et sélectionnez « Show as… ». Vous pouvez choisir les types d'affichage dans la liste du menu contextuel.



Affichage des données en mode Test

Vous pouvez ajouter du code à votre programme pour afficher les valeurs de vos capteurs et actionneurs pendant que le robot est en mode Test. Cela peut être fait à partir de Driver Station chaque fois que le robot n'est pas sur le terrain. Le code pour afficher ces valeurs est généré automatiquement par RobotBuilder ou ajouté manuellement à votre programme, tel que décrit dans l'article suivant. Le mode Test est conçu pour vérifier le bon fonctionnement des capteurs et des actionneurs sur un robot. En outre, il peut être utilisé pour obtenir des repères des capteurs tels que les potentiomètres et pour régler les boucles PID dans votre code.

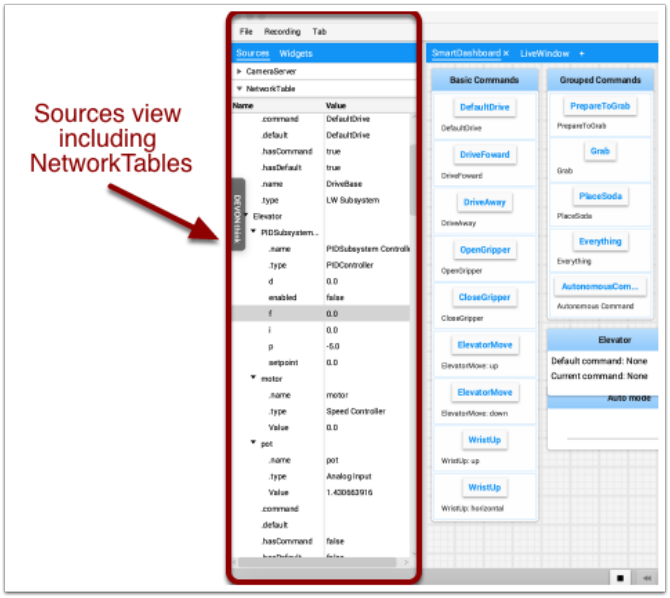
Configuration du mode Test



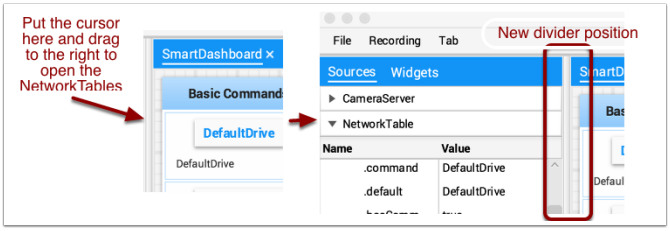
Activez le mode Test dans Driver Station en cliquant sur le bouton « Test » et choisir « Enable » pour le robot. Ce faisant, Shuffleboard affiche, par sous-système, l'état des actionneurs et capteurs utilisés par votre programme.

Obtention de données à partir de la vue Sources

Normalement, les données *NetworkTables* apparaissent automatiquement sur l'un des onglets et il vous suffit de réorganiser et d'utiliser ces données. Parfois, vous souhaiterez peut-être récupérer une valeur supprimée accidentellement de l'onglet ou afficher une valeur qui ne fait pas partie de la clé SmartDashboard / NetworkTables. Dans ces cas, les valeurs peuvent être glissées sur un volet à partir de la vue NetworkTables sous Sources sur le côté gauche de la fenêtre. Choisissez la valeur que vous souhaitez afficher et faites-la simplement glisser dans le volet et elle sera automatiquement créée avec le type de widget par défaut pour le type de données.



Note : Quelques fois, la vue Sources n'est pas visible à gauche - il est effectivement possible de faire glisser le diviseur entre les volets à onglets et les sources afin que les Sources ne sont pas visibles. Si cela se produit, déplacez le curseur sur le bord gauche et recherchez un curseur de séparation et redimensionnement, puis cliquez à gauche et faites glisser la vue. Dans les deux images ci-dessous, vous pouvez voir où cliquer et glisser. Une fois terminé le diviseur est comme indiqué dans la deuxième image.

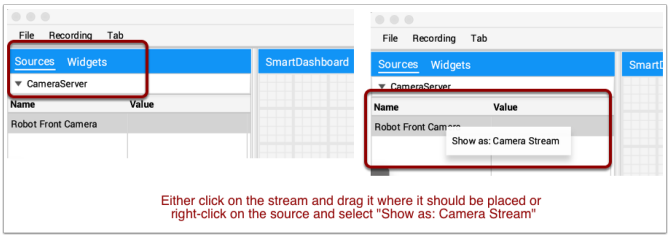


Affichage des flux de caméra

Les flux de caméra du robot peuvent être visualisés sur un onglet dans Shuffleboard. Ceci est utile pour visualiser ce que le robot voit pour donner une vue moins obstruée aux opérateurs ou visualiser la sortie d'un algorithme de vision fonctionnant sur l'ordinateur de la station de pilotage ou un coprocesseur sur le robot. Tout flux qui s'exécute à l'aide de l'API CameraServer peut être visualisé dans un widget de flux de caméra.

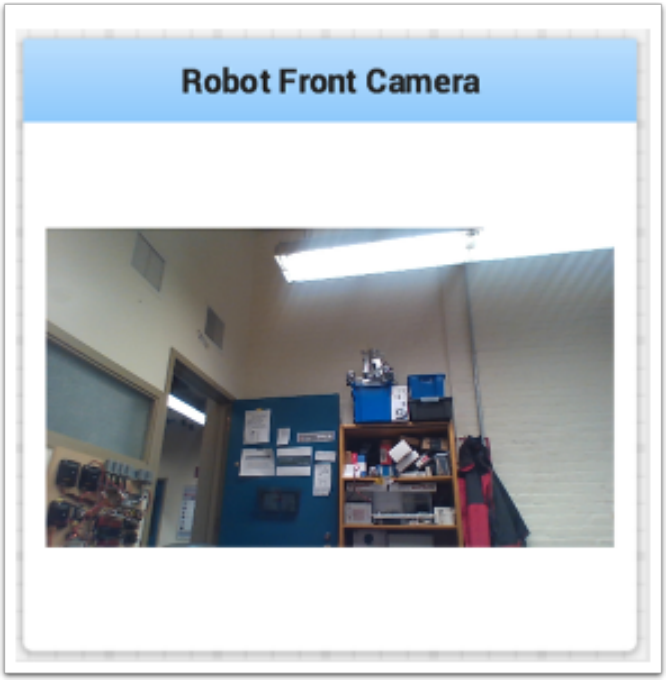
Ajout d'un flux de caméra

Pour ajouter une caméra à votre tableau de bord, sélectionnez « Sources » et affichez la source « CameraServer » dans le panneau gauche de la fenêtre de Shuffleboard comme illustré dans l'exemple ci-dessous. Une liste de flux de caméra sera affichée, dans ce cas-ci, il n'y a qu'une seule caméra appelée « Robot Front Camera ». Faites-la glisser dans l'onglet où elle doit être affichée. Alternativement, le flux peut également être placé sur le tableau de bord en cliquant avec le bouton droit sur le flux dans la liste Sources et en sélectionnant « Show as : Camera Stream ».



Une fois le flux de caméra ajouté, il s'affiche dans la fenêtre. Il peut être redimensionné et déplacé là où vous le désirez.

Note : Sachez que le fait d'envoyer trop de données à partir d'une résolution trop élevée ou une fréquence d'image trop élevée entraînera une utilisation élevée du processeur à la fois sur le roboRIO et l'ordinateur portable de pilotage.

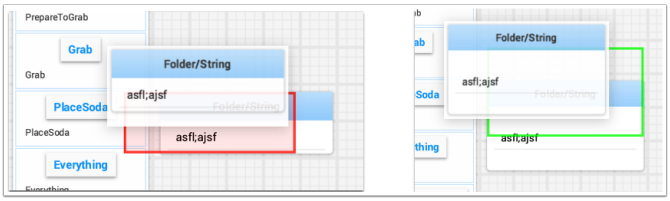


Utilisation de widgets

Les dispositifs d'affichage que vous manipulez à l'écran dans Shuffleboard sont appelés widgets. Les widgets sont généralement automatiquement affichés à partir de valeurs que le programme du robot publie dans NetworkTables.

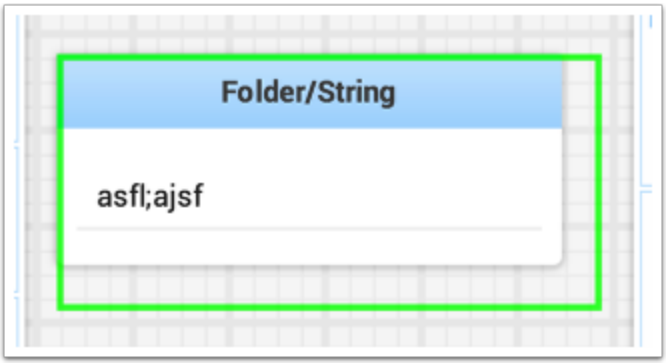
Déplacement des widgets

Les widgets peuvent être déplacés simplement avec glisser et déposer. Il suffit de déplacer le curseur sur le widget, cliquez à gauche et faites-le glisser vers la nouvelle position. Lorsque vous faites glisser, vous ne pouvez placer les widgets que sur les carrés de grille et la taille de la grille aura un effet sur la résolution de votre affichage. Lors d'un déplacement, un contour rouge ou vert s'affiche. Vert signifie généralement qu'il y a assez de place à l'emplacement visé pour y déposer le widget et rouge signifie généralement qu'il y aura chevauchement ou sera trop grand pour l'y déposer. Dans l'exemple ci-dessous, un widget est déplacé vers un endroit qui ne lui convient pas.



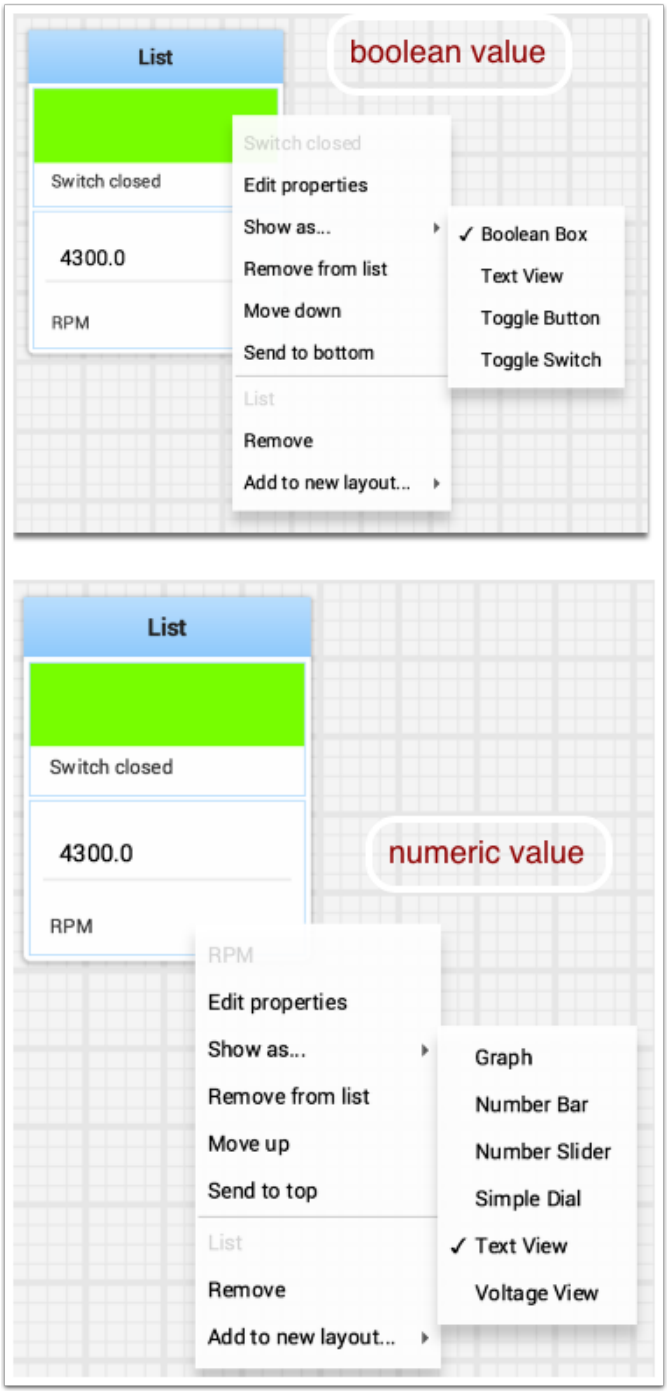
Redimensionnement des widgets

Les widgets peuvent être redimensionnés en cliquant et en faisant glisser le bord ou le coin de l'image du widget. Le curseur se transforme en curseur de redimensionnement lorsqu'il est dans la bonne position pour redimensionner un widget. Comme pour le déplacement de widgets, un contour vert ou rouge sera dessiné indiquant que le widget peut être redimensionné ou non. L'exemple ci-dessous montre un widget en cours de redimensionnement à une zone plus grande avec le contour vert indiquant qu'il n'y a pas de chevauchement avec les widgets environnants.



Modification du type d'affichage des widgets

Shuffleboard est très riche en types d'affichage pour les données publiées par le robot. Par défaut, un type d'affichage est choisi automatiquement mais vous pouvez par le modifier par la suite selon l'application. Pour voir les options d'affichage disponibles pour n'importe quel widget, cliquez avec le bouton droit sur le widget et sélectionnez « Show as… » et, dans le menu contextuel, choisissez le type souhaité. Dans l'exemple ci-dessous on trouve deux valeurs de données, l'une est numérique et l'autre est booléenne. Vous pouvez voir les différents types d'options d'affichage qui sont disponibles pour chacune. La donnée booléenne n'a que deux valeurs possibles (true/false) qui peut être affichée sous la forme d'une zone booléenne (de couleur rouge/verte), d'un texte ou d'un bouton à bascule ou d'un commutateur à bascule. La donnée numérique peut être affichée sous la forme d'un graphique, d'une échelle, d'un curseur, d'un cadran, d'un texte ou d'une vue de tension selon le contexte de la valeur.

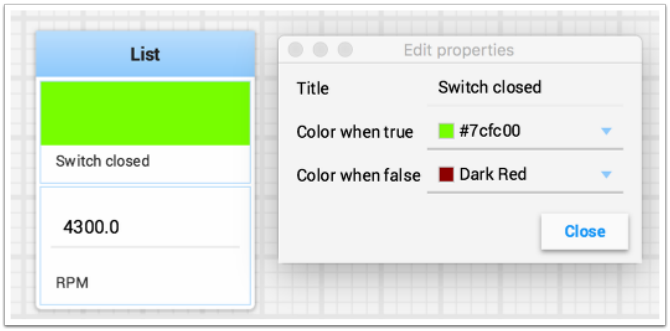

List
boolean value
Switch closed
4300.0
RPM
Switch closed
Edit properties
Show as...
Remove from list
Move down
Send to bottom
List
Remove
Add to new layout...
Boolean Box
Text View
Toggle Button
Toggle Switch
List
Switch closed
4300.0
RPM
numeric value
RPM
Edit properties
Show as...
Remove from list
Move up
Send to top
List
Remove
Add to new layout...
Graph
Number Bar
Number Slider
Simple Dial
Text View
Voltage View

Modification du titre des widgets

Vous pouvez modifier le titre des widgets en double-cliquant dans leur barre de titre et en modifiant le titre à la nouvelle valeur. Si un widget est contenu dans une mise en page, cliquez avec le bouton droit sur le widget et sélectionnez les propriétés. De là, vous pouvez modifier le titre du widget qui s'affiche.

Modification des propriétés d'un widget

Vous pouvez modifier l'apparence d'un widget telle que la plage des valeurs représentées, les couleurs ou tout autre élément visuel. Dans les situations où cela est possible, cliquez avec le bouton droit sur le widget et sélectionnez « Edit properties » dans le menu contextuel. Dans le widget de valeur booléenne ci-dessous, le titre du widget, les couleurs correspondant aux états vrai et faux peuvent tous être modifiés.



Utilisation des listes

Lists in Shuffleboard are sets of tiles grouped together in a vertical layout, making it visually obvious that those tiles are related. In addition, tiles in lists take up less screen space than individual tiles :

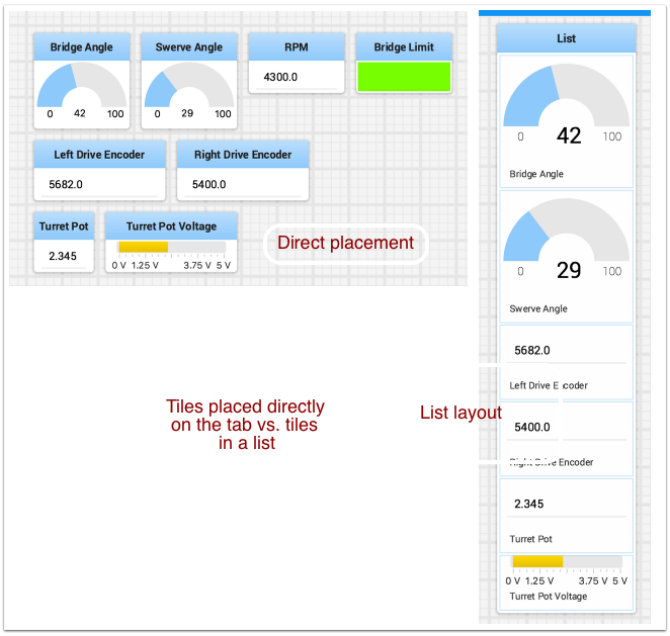
— Tiles in lists don't have individual header labels ; they instead have smaller labels within their list entries.
— Individual tiles placed together create gaps between one another ; lists have smaller gaps between tiles.

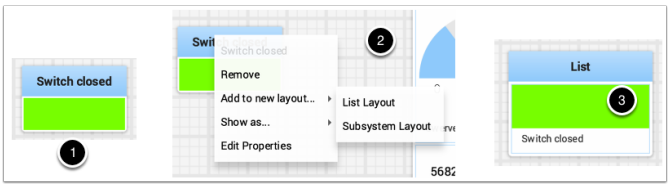
Création d'une liste

A list can be created as follows :

1. Right-click on the tile that should be first in the list.
2. Select « Add to new layout… », then « List Layout » from the popup menu.
3. A new list will be created labeled « List », and the tile will be at the top of it.

Note that tiles in lists do not have header labels ; their label is at the bottom of their list entry.





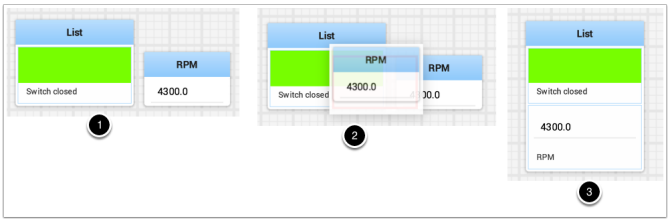
Adding tiles to/removing tiles from a list

A tile can be **added** to an existing list as follows :

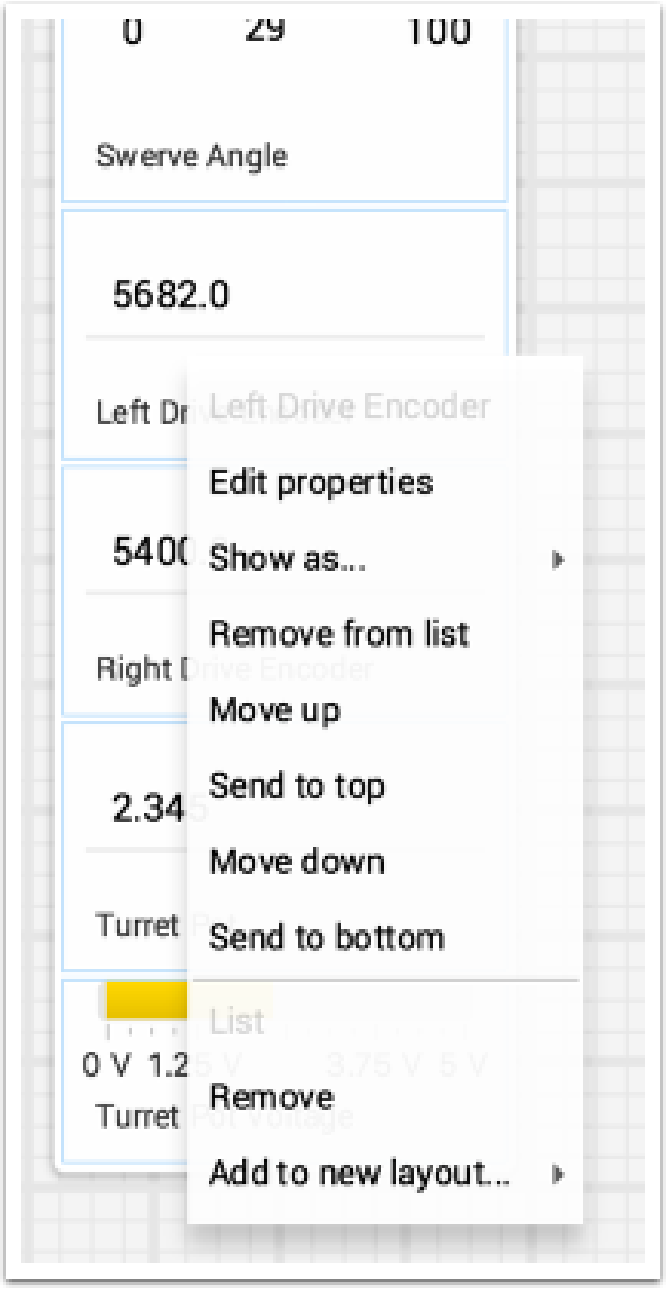
1. Identify the list and the tile to be added.
2. Drag the new tile onto the list.
3. The tile will be added to the list. If the current list size is too small to show it, the tile will be added to the list off-screen and a vertical scrollbar will be added if not already present.

A tile can be **removed** from a list by following the process in reverse :

1. Identify the list and the tile within it to be removed.
2. Drag the tile out of the list and place it anywhere with free space.
3. The tile will be removed from the list and placed at that location.



Réorganisation des vignettes dans une liste



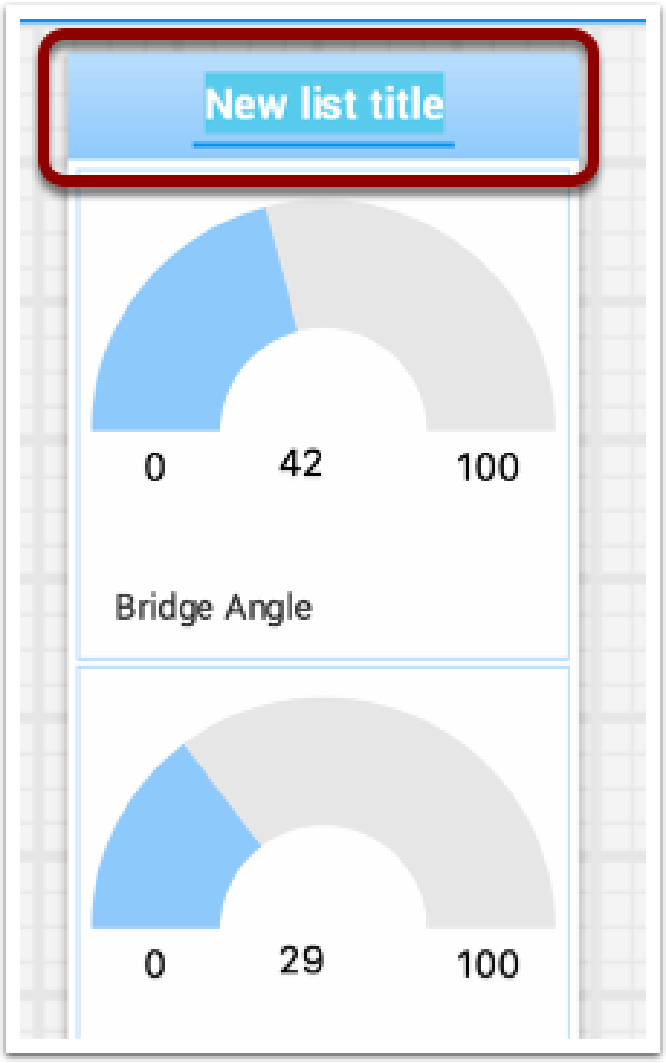
Tiles in a list can be rearranged by right-clicking on the tile and selecting :

- *Move up* moves the tile **towards** the *top* of the list.
- *Move down* moves the tile **towards** the *bottom* of the list.
- *Send to top* moves the tile **to** the *top* of a list.
- *Send to bottom* moves the tile **to** the *bottom* of a list.
- There are two buttons labeled *Remove*, and each button does :
  - The **top** *Remove* button (above the pinline ; section of dropdown with grayed-out *tile* label) **deletes** the *tile* from the Shuffleboard layout.
  - The **bottom** *Remove* button (below the pinline ; section of dropdown with grayed-out *list* label) **deletes** the *list and all tiles inside it* from the Shuffleboard layout.

— If you want to take an entry out of a list without deleting it, see *Adding tiles to/removing tiles from a list*.

Renommer une liste

You can rename a list by double-clicking on the list label and changing the name. Click outside the label to save changes.

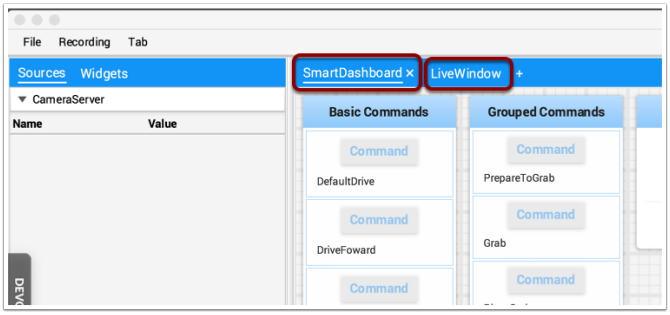


Création et manipulation des onglets

La présentation par onglets qu'utilise Shuffleboard aide à séparer différentes « vues » de données de votre robot et à rendre l'affichage plus pratique. Par exemple, vous pourriez avoir un onglet qui affiche des données pour le débogage de votre robot et un autre onglet pour l'utilisation en tournoi. Il existe plusieurs options qui rendent les onglets très puissants. Vous pouvez contrôler quelles données de NetworkTables ou d'autres sources s'affichent dans chacun de vos onglets en utilisant les options d'ajout automatique qui sont décrites plus loin dans cet article.

Onglets par défaut

Lorsque vous ouvrez Shuffleboard pour la première fois, deux onglets intitulés SmartDashboard et LiveWindow sont présents. Ils représentent les deux vues SmartDashboard dépendamment si votre robot est en mode autonome/téléopéré ou en mode test. Dans Shuffleboard, les deux vues sont disponibles en tout temps.



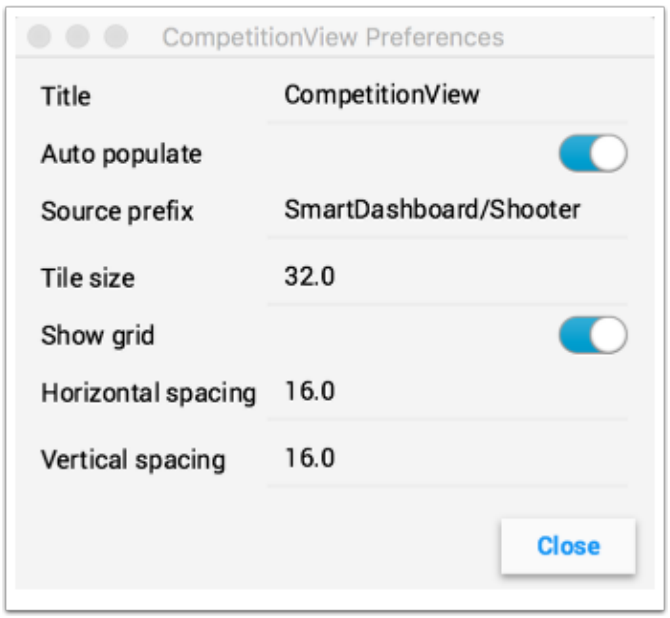
Les valeurs écrites en invoquant les différentes variantes de la méthode SmartDashboard.putType() sont affichées dans l'onglet SmartDashboard. Les valeurs de débogage générées automatiquement sont affichées dans l'onglet LiveWindow.

Navigation entre les onglets

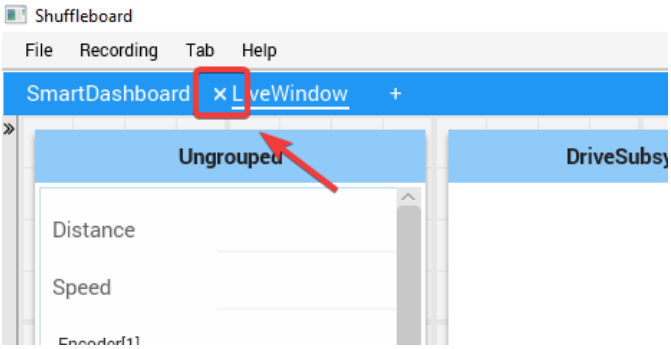
Vous pouvez naviguer entre les onglets en cliquant sur leur étiquette en haut de la fenêtre. Dans le cas ci-dessus, cliquez simplement sur SmartDashboard ou LiveWindow pour voir les valeurs associées à chaque onglet.

Ajouter ou masquer des onglets

Vous pouvez ajouter des onglets supplémentaires en cliquant sur le symbole plus(+) juste à droite du dernier onglet. Une fois que vous avez créé un nouvel onglet, vous pouvez définir son étiquette en double-cliquant sur celle-ci et en modifiant le texte. Vous pouvez également cliquer avec le bouton droit sur l'onglet ou utiliser le menu Tab pour ouvrir les préférences d'onglets. À partir de cette fenêtre, vous pouvez changer le nom en modifiant le champ Title.



Vous pouvez masquer les onglets en cliquant sur le symbole moins (-) à gauche du nom de l'onglet sélectionné. Étant donné que les onglets sont générés en fonction du *NetworkTable* pertinent, il n'est pas possible de les supprimer définitivement sans supprimer la table.



Configuration des onglets pour remplissage automatique

L'une des fonctionnalités les plus puissantes des onglets est de les configurer pour se remplir automatiquement de nouvelles valeurs en fonction d'un préfixe source fourni dans la fenêtre de préférences de l'onglet. Dans l'exemple ci-dessus, la fenêtre de préférences a un préfixe source de « SmartDashboard/Shooter » et l'option de remplissage automatique des valeurs, Auto populate, est activée. Toutes les valeurs écrites à l'aide de la classe SmartDashboard qui spécifie une sous-clé Shooter apparaîtront automatiquement sur cet onglet. Remarque : les clés qui correspondent à plusieurs préfixes Sources apparaîtront dans chacun de ces onglets. Étant donné que ces clés commencent également par SmartDashboard/ et qu'il s'agit du préfixe Source de l'onglet SmartDashboard par défaut, ces widgets apparaîtront dans les deux onglets. Pour que les valeurs n'apparaissent que dans un seul onglet, vous pouvez utiliser NetworkTables pour écrire des étiquettes et des valeurs et utiliser un chemin différent qui ne commence pas par SmartDashboard/. Alternativement, vous pouvez décider de tout laisser apparaître dans l'onglet SmartDashboard tout en ayant d'autres onglets spécifiques à vos besoins qui seront filtrés. L'onglet SmartDashboard sera très encombré, mais les autres onglets seront comme vous le voulez.

Utilisation de la grille et de l'espacement de l'onglet

Chaque onglet peut avoir sa propre taille de vignette (nombre de pixels par grand carré). Ainsi, certains onglets peuvent avoir une résolution plus grossière pour une mise en page plus facile et d'autres peuvent avoir une grille fine. Le paramètre de la taille de vignette, Tile size, dans la fenêtre de préférences de l'onglet remplace les paramètres globaux dans les préférences globales Shuffleboard. De plus, vous pouvez spécifier l'espacement entre le contenu d'un widget et le bord du widget. Si vous programmez des interfaces utilisateur, ces paramètres sont généralement appelés écart horizontal et écart vertical (hgap, vgap).

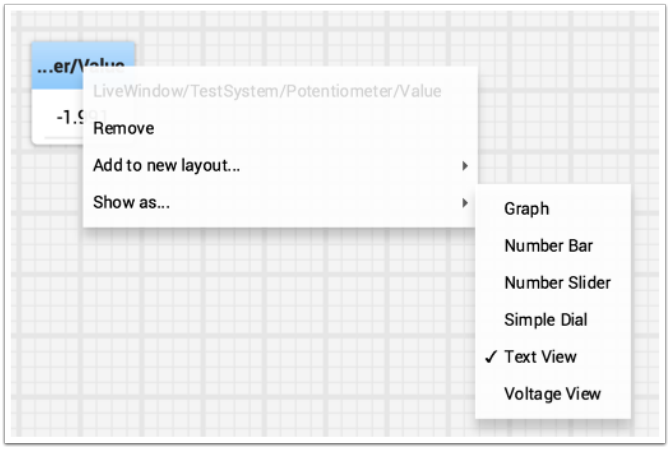
Déplacement des widgets entre les onglets

Actuellement, il n'existe aucun moyen de déplacer facilement les widgets entre onglets sans les supprimer d'un onglet et en faisant glisser le champ de la hiérarchie Sources sur la gauche vers le nouvel onglet. Nous espérons bientôt avoir cette fonctionnalité dans une prochaine mise à jour.

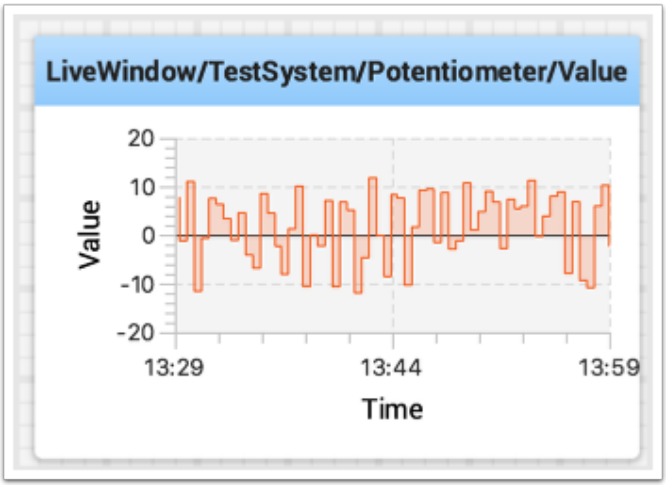
Utilisation de graphiques

With Shuffleboard you can graph numeric values over time. Graphs are very useful to see how sensor or motor values are changing as your robot is operating. For example the sensor value can be graphed in a PID loop to see how it is responding during tuning.

Pour créer un graphique, choisissez une valeur numérique, cliquez avec le bouton droit sur le titre et sélectionnez « Show as… », puis choisissez Graph.

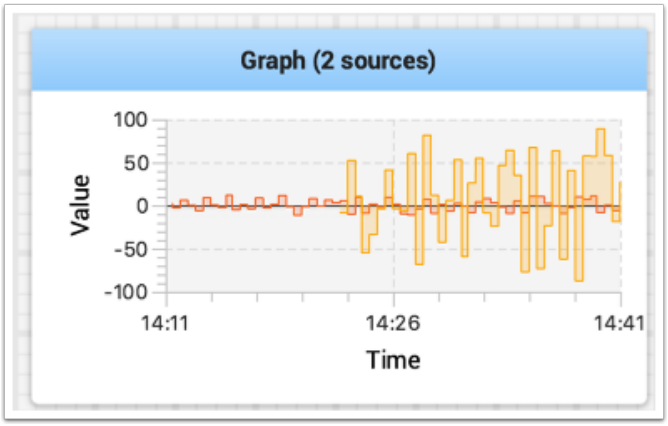


Le widget Graph affiche un graphique par segments de la valeur du paramètre que vous avez sélectionné. Il définira automatiquement l'échelle. L'intervalle de temps par défaut sera de 30 secondes. Vous pouvez modifier cela dans les paramètres du graphique (voir ci-dessous).

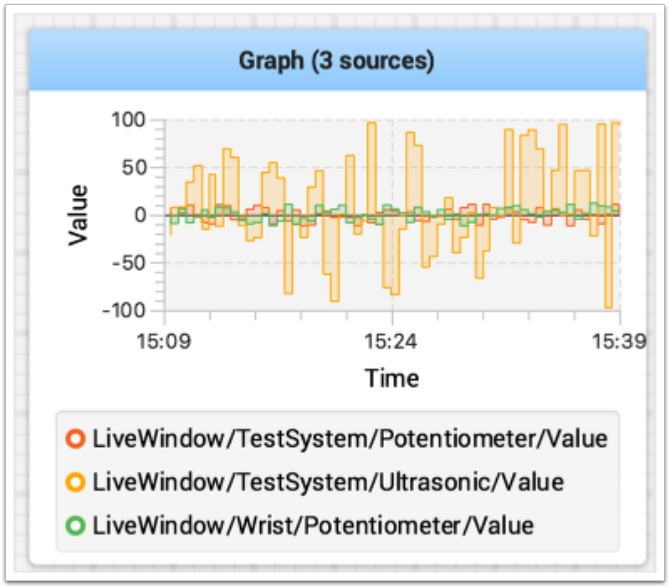


Ajout de valeurs de données supplémentaires

Pour les valeurs liées, il est souvent souhaitable d'afficher plusieurs valeurs sur le même graphique. Pour ce faire, faites simplement glisser des valeurs supplémentaires depuis la vue source NetworkTables (côté gauche de la fenêtre Shuffleboard) et déposez-les sur le graphique et cette valeur sera ajoutée comme indiqué ci-dessous. Vous pouvez continuer à faire glisser des valeurs supplémentaires sur le graphique.



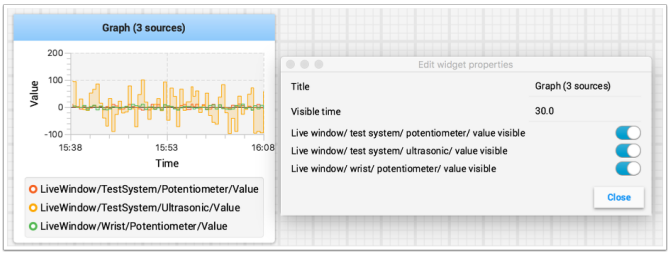
Vous pouvez redimensionner le graphique verticalement pour afficher la légende si elle ne s'affiche pas, comme indiqué dans l'image ci-dessous. La légende montre toutes les sources qui sont utilisées dans le graphique.



Propriétés du graphique

Vous pouvez définir le nombre de secondes qui s'affichent dans le graphique en modifiant la valeur « Visible time » dans les propriétés du widget graphique. Pour accéder aux propriétés, cliquez avec le bouton droit sur le graphique et sélectionnez « Edit properties ».

En plus de définir le nombre de secondes qui s'affichent dans le graphique, le graphique peut activer ou désactiver des sources en activant ou en désactivant l'interrupteur pour chacune des sources indiquées dans la fenêtre des propriétés (voir ci-dessous).

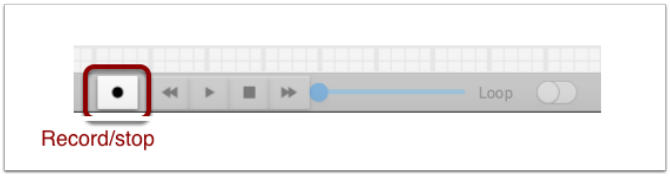


Enregistrement et lecture

Shuffleboard peut enregistrer toutes les mises à jour de widget au cours d'une session. Plus tard, le fichier journal peut être « rejoué » pour voir ce qui s'est passé pendant un match ou une pratique. Ceci est particulièrement utile si quelque chose ne fonctionne pas comme prévu pendant un match et que vous voulez voir ce qui s'est passé. Chaque enregistrement est consigné dans un fichier d'enregistrement.

Création d'un enregistrement

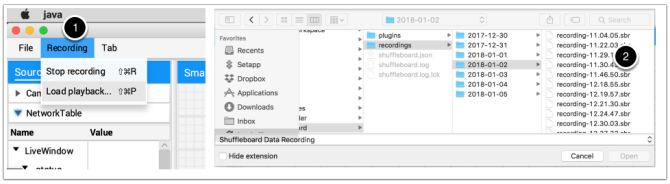
Lorsque le shuffleboard démarre, il commence à enregistrer toutes les valeurs NetworkTables et continue jusqu'à ce que le bouton d'enregistrement / d'arrêt dans les commandes de l'enregistreur soit appuyé, comme indiqué ci-dessous. Si un nouvel enregistrement est souhaité, par exemple lorsqu'un nouveau code ou un nouveau système mécanique est testé, arrêtez l'enregistrement en cours s'il est en marche et cliquez sur le bouton d'enregistrement. Cliquez à nouveau sur le bouton pour arrêter l'enregistrement et fermer le fichier d'enregistrement. Si le bouton est un cercle (comme illustré), cliquez dessus pour démarrer un enregistrement. Si le bouton est un carré, alors un recodage est en cours, alors cliquez dessus pour arrêter l'enregistrement.



Lecture d'un enregistrement

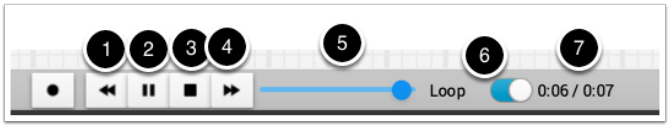
Les enregistrements antérieurs peuvent être rejoués ainsi :

1. Sélectionnez le menu « Recording » puis cliquez sur « Load playback ».
2. Choisissez un enregistrement dans le répertoire affiché. Les enregistrements sont regroupés par date et les noms de fichier correspondent à l'heure à laquelle l'enregistrement a été effectué pour ainsi aider à en identifier le bon. Dans la liste, sélectionnez l'enregistrement.



Commandes de lecture

La sélection du fichier d'enregistrement lancera la lecture de ce fichier. Pendant que la lecture est en cours, un compteur horaire permet d'en suivre la progression. Il existe également une option de lecture en boucle. L'interface du lecteur comprend également des commandes qui offrent un contrôle plus précis de la lecture de l'enregistrement.

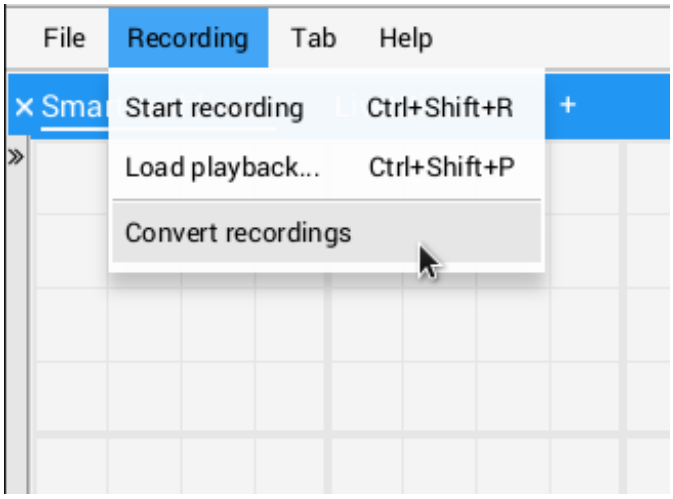


Les commandes fonctionnent de la manière suivante :

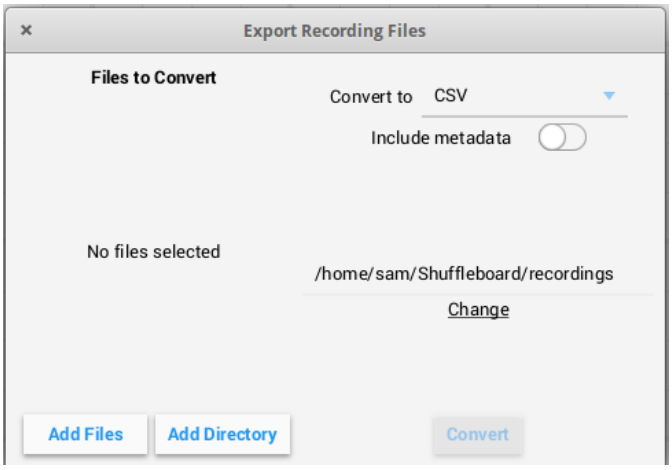
1. Le bouton double flèche gauche ramène la lecture à la dernière donnée modifiée.
2. Les commandes play/pause démarrent et arrêtent la lecture
3. De forme carrée, le bouton d'arrêt met fin à la lecture et Shuffleboard bascule en mode affichage des valeurs actuelles du robot

4. La double flèche droite avance la lecture vers la modification de donnée suivante
5. La barre de défilement permet de se positionner à n'importe quel moment afin d'afficher différentes parties de l'enregistrement
6. Ce bouton glisseur permet de passer à la lecture en boucle jusqu'à interruption.
7. L'indicateur horaire montre le temps écoulé depuis le commencement de la lecture de l'enregistrement et la durée totale de l'enregistrement

Conversion en différents formats de fichier



Les enregistrements de Shuffleboard sont dans un format binaire par souci d'efficacité. Pour analyser les données enregistrées sans les lire à travers l'application, Shuffleboard prend en charge des convertisseurs de données afin de convertir les enregistrements en un format de votre choix. Seul un convertisseur CSV élémentaire est fourni avec l'application, mais les équipes peuvent écrire des convertisseurs personnalisés et les inclure dans les plugins de Shuffleboard.



Plusieurs enregistrements peuvent être convertis à la fois. Les fichiers individuels peuvent être sélectionnés à l'aide du bouton « Add Files », ou tous les fichiers d'un répertoire peuvent être conjointement sélectionnés à l'aide du bouton « Add Directory ».

Les enregistrements convertis seront générés dans le répertoire `~/Shuffleboard/recordings`, mais peuvent être sélectionnés manuellement à l'aide du bouton « Change ».

Différents convertisseurs peuvent être sélectionnés dans la liste déroulante en haut à droite. Par défaut, seul le convertisseur CSV est disponible. Les convertisseurs personnalisés des plugins apparaissent comme options dans la liste déroulante.

Notes supplémentaires

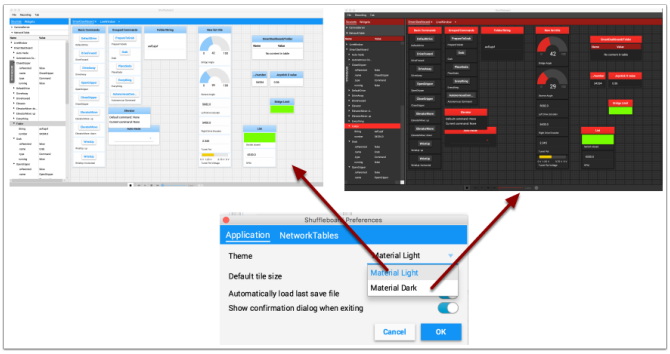
Les graphiques ne s'affichent pas correctement lorsque le curseur temporel est manuellement déplacé, toutefois si ce déplacement passe par des points précis présents dans l'historique du graphique alors le graphique résultant correspondra à celui qu'on obtiendrait de la lecture originale.

Réglage global des préférences de Shuffleboard

Il existe un certain nombre de réglages qui contrôlent l'apparence et le comportement de Shuffleboard. On peut y accéder dans le menu « File », sous le panneau « Preferences ».

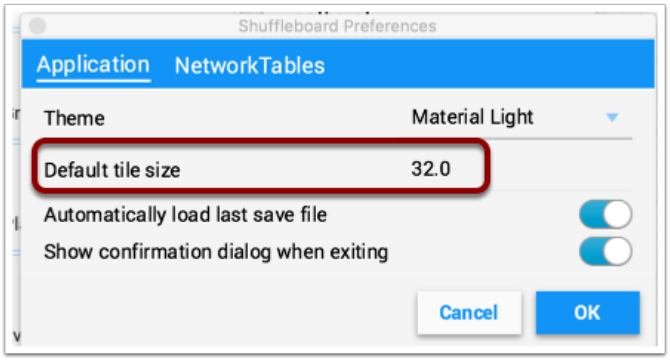
Réglage du thème

Shuffleboard supporte deux thèmes selon votre préférence : « Material Dark » et « Material Light ». Les feuilles de style en cascade, css ou Cascading Style Sheets, sont appliquées sur l'application entière et peuvent être modifiées à tout moment.



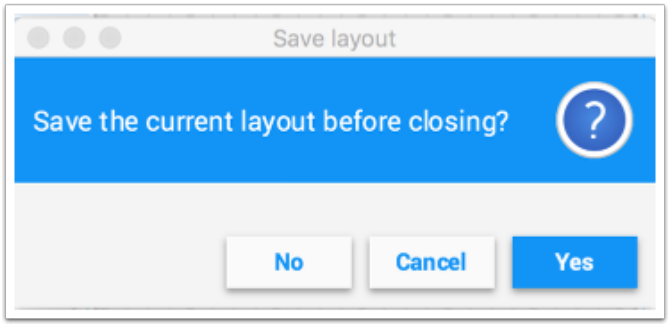
Réglage de la taille par défaut des vignettes

Shuffleboard positionne les vignettes sur une grille lorsqu'on les ajoute, les déplace, ou lorsqu'elles sont automatiquement générées. Vous pouvez choisir une taille par défaut pour chaque onglet, ou l'appliquer à tous les onglets nouvellement créés. Une résolution plus fine permet un placement plus précis des vignettes. Ceci est configurable dans la fenêtre des préférences ci-dessous.

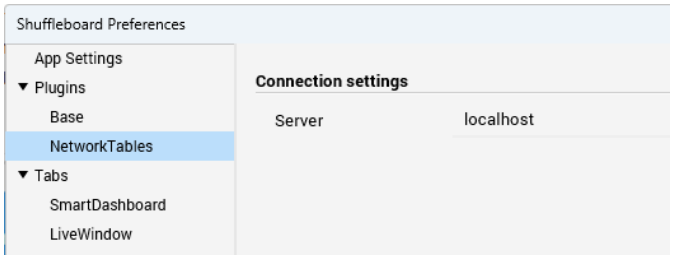


Travailler avec les sauvegardes de la disposition des vignettes

Vous pouvez enregistrer votre mise en page en utilisant les options File / Save et File / Save as... du menu. La fenêtre des préférences comporte des options permettant d'appliquer automatiquement la mise en page précédente au démarrage de Shuffleboard. De plus, le Shuffleboard affichera une fenêtre « Save layout » pour vous rappeler d'enregistrer la mise en page à sa fermeture, si la mise en page a changé. Vous pouvez choisir de désactiver l'invite automatique à la fermeture, mais, dans ce cas, assurez-vous d'enregistrer la mise en page manuellement, afin de ne pas perdre vos modifications.



Configuration du numéro d'équipe



Pour permettre à Shuffleboard de se connecter au serveur des NetworkTables de votre robot, il est nécessaire de lui indiquer le bon numéro d'équipe à l'aide de l'onglet « NetworkTables » dans le panneau de préférences. Si votre Shuffleboard a été lancé par Driver Station, ce dernier remplira automatiquement cette information via le champ « Server ». Si vous roulez Shuffleboard sans Driver Station, vous pouvez entrer soit votre numéro d'équipe, soit l'adresse réseau du roboRIO.

FAQ, problèmes et bogues concernant Shuffleboard

Avertissement : La plupart des composants du système de contrôle tel que Shuffleboard ont été développés avec Java 11. Ils ne sont donc pas compatibles avec Java 8. Avant de signaler un problème assurez-vous d'avoir Java 11 installé en tant qu'environnement Java par défaut.

Foire aux questions

Comment puis-je signaler des problèmes, des bogues et faire des demandes de fonctionnalités concernant Shuffleboard ?

There is no active maintainer of Shuffleboard, but we are accepting pull requests. Bugs, issues, and feature requests can be added on the Shuffleboard GitHub page by creating an issue. Please try to look at existing issues before creating new ones to make sure you aren't duplicating something that has already been reported or work that is planned. You can find the issues on the Shuffleboard GitHub page.

Comment ajouter mes propres widgets et extensions à Shuffleboard ?

Custom Widgets has a large amount of documentation on extending the program with custom plugins. Sample plugin projects that can be used for additional custom widgets and themes can be found on the Shuffleboard GitHub page.

Comment compiler et construire Shuffleboard à partir du code source ?

Vous pouvez obtenir le code source en téléchargeant, clonant ou fourchant la bibliothèque sur Github. Afin de construire et rouler Shuffleboard à partir du code source, placez le répertoire courant à la racine du code source, puis utilisez l'une des commandes suivantes :

Application	Commande (sur Windows lancer le fichier gradlew.bat)
Lancer Shuffleboard	./gradlew :app :run
Construire les APIs et les classes utilitaires pour la création de modules d'extension	./gradlew :api :shadowJar
Construire le fichier .jar de l'application	./gradlew :app :shadowJar

11.2.2 Shuffleboard - Dispositions avec code

Utilisation des onglets

Shuffleboard est une interface à onglets. Chaque onglet organise les widgets par regroupement logique. Par défaut, Shuffleboard a des onglets pour SmartDashboard et LiveWindow hérités - mais de nouveaux onglets peuvent maintenant être créés dans Shuffleboard directement à partir d'un programme de robot pour une meilleure organisation.

Création d'un nouvel onglet

JAVA

```
ShuffleboardTab tab = Shuffleboard.getTab("Tab Title");
```

C++

```
ShuffleboardTab& tab = Shuffleboard::GetTab("Tab Title");
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

tab = Shuffleboard.getTab("Tab Title")
```

La création d'un nouvel onglet est aussi simple que d'appeler une seule méthode sur la classe Shuffleboard, ce qui créera un nouvel onglet sur Shuffleboard et retournera une poignée- référence abstraite à une ressource utilisée- pour ajouter vos données à l'onglet. Invoquer plusieurs fois getTab avec le même titre d'onglet retournera la même poignée à chaque fois.

Sélection d'un onglet

JAVA

```
Shuffleboard.selectTab("Tab Title");
```

C++

```
Shuffleboard::SelectTab("Tab Title");
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

Shuffleboard.selectTab("Tab Title")
```

Cette méthode permet de sélectionner un onglet par titre. Ceci est sensible à la casse (donc « Tab Title » et « Tab title » sont deux onglets distincts), et ne fonctionne que si un onglet avec ce titre existe au moment où la méthode est appelée ; donc invoquer `selectTab("Exemple")` n'aura d'effet que si un onglet nommé « Exemple » a déjà été défini.

Cette méthode peut être utilisée pour sélectionner n'importe quel onglet dans Shuffleboard, pas seulement ceux créés par le programme du robot.

Avertissements

Les onglets créés à partir d'un programme de robot diffèrent de plusieurs manières importantes des onglets normaux créés à partir du tableau de bord :

- Non enregistrés dans le fichier de sauvegarde Shuffleboard
- Pas de support pour l'autopopulation
- Les utilisateurs doivent spécifier le contenu de l'onglet dans leur programme de robot
- Ils sont d'une couleur spéciale pour les différencier des onglets normaux

Envoi de données

Contrairement à SmartDashboard, les données ne peuvent pas être envoyées directement à Shuffleboard sans spécifier au préalable dans quel onglet les données doivent être placées.

Envoi de données simples

L'envoi de données simples (nombres, chaînes de caractères, booléens et tableaux de ces types) se fait en invoquant `add` sur un onglet. Cette méthode définira la valeur si elle n'est pas déjà présente, mais n'écrasera pas une valeur existante.

JAVA

```
Shuffleboard.getTab("Numbers")
     .add("Pi", 3.14);
```

C++

```
frc::Shuffleboard::GetTab("Numbers")
     .Add("Pi", 3.14);
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

Shuffleboard.getTab("Tab Title").add("Pi", 3.14)
```

Si les données doivent être mises à jour (par exemple, la sortie de certains calculs effectués sur le robot), appelez `getEntry()` après avoir défini la valeur, puis mettez-la à jour lorsque nécessaire ou dans une fonction `periodic`

JAVA

```
class VisionCalculator {
   private ShuffleboardTab tab = Shuffleboard.getTab("Vision");
   private GenericEntry distanceEntry =
      tab.add("Distance to target", 0)
         .getEntry();

   public void calculate() {
   double distance = ...;
   distanceEntry.setDouble(distance);
   }
}
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

def robotInit(self):
   tab = Shuffleboard.getTab("Vision")
   self.distanceEntry = tab.add("Distance to target", 0).getEntry()

def teleopPeriodic(self):
   distance = self.encoder.getDistance()
   self.distanceEntry.setDouble(distance)
```

Rétention des choix entre les redémarrages

Lors de la configuration d'un robot à partir du tableau de bord, on peut avoir besoin de retenir certain paramètres entre les redémarrages du robot ou de la station de pilotage au lieu de s'en remettre aux pilotes pour se souvenir (ou oublier) de configurer les paramètres avant chaque match.

La simple utilisation de *addPersistent* au lieu de *add* enregistrera la valeur sur le roboRIO et elle sera chargée au démarrage du programme du robot.

Note : Cela ne s'applique pas aux données transmissibles telles que les sélecteurs ou les contrôleurs de moteur.

JAVA

```
Shuffleboard.getTab("Drive")
     .addPersistent("Max Speed", 1.0);
```

C++

```
frc::Shuffleboard::GetTab("Drive")
     .AddPersistent("Max Speed", 1.0);
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

(Shuffleboard.getTab("Drive")
      .addPersistent("Max Speed", 1.0))
```

Envoi de capteurs, moteurs, etc.

Analogue à `SmartDashboard.putData`, n'importe quel objet `Sendable` (la plupart des capteurs, contrôleurs de moteurs et SendableChoosers) peut être ajouté à n'importe quel onglet

JAVA

```
Shuffleboard.getTab("Tab Title")
     .add("Sendable Title", mySendable);
```

C++

```
frc::Shuffleboard::GetTab("Tab Title")
     .Add("Sendable Title", mySendable);
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

(Shuffleboard.getTab("Tab Title")
      .add("Sendable Title", mySendable))
```

Récupération des données

Contrairement à `SmartDashboard.getNumber` et autres méthodes similaires, la récupération de données à partir de Shuffleboard se fait également via NetworkTableEntries, que nous avons couvert dans un article précédent.

JAVA

```
class DriveBase extends Subsystem {
   private ShuffleboardTab tab = Shuffleboard.getTab("Drive");
   private GenericEntry maxSpeed =
      tab.add("Max Speed", 1)
         .getEntry();

   private DifferentialDrive robotDrive = ...;

   public void drive(double left, double right) {
   // Retrieve the maximum speed from the dashboard
   double max = maxSpeed.getDouble(1.0);
   robotDrive.tankDrive(left * max, right * max);
   }
}
```

PYTHON

```
import commands2
import wpilib.drive
from wpilib.shuffleboard import Shuffleboard

class DriveSubsystem(commands2.SubsystemBase):
   def __init__(self) -> None:
      super().__init__()

      tab = Shuffleboard.getTab("Drive")
      self.maxSpeed = tab.add("Max Speed", 1).getEntry()

      this.robotDrive = ...
```

(suite sur la page suivante)

(suite de la page précédente)

```
   def drive(self, left: float, right: float):
      # Retrieve the maximum speed from the dashboard
      max = self.maxSpeed.getDouble(1.0)
      self.robotDrive.tankDrive(left * max, right * max)
```

Cet exemple simple a un défaut majeur : la vitesse maximale peut être réglée sur le tableau de bord à une valeur en dehors de de la plage [0, 1] - ce qui pourrait entraîner la saturation des entrées (toujours à la vitesse maximale), voire les inverser ! Heureusement, il existe un moyen d'éviter ce problème - ce sera abordé dans le prochain article.

Configuration des widgets

Les programmes de robot peuvent spécifier exactement quel widget utiliser pour afficher un point de données, ainsi que la façon dont ce widget doit être configuré. Comme il y a trop de widgets pour être listés ici, consultez la documentation pour plus de détails.

Spécification d'un widget

Call `withWidget` after `add` in the call chain :

JAVA

```
Shuffleboard.getTab("Drive")
   .add("Max Speed", 1)
   .withWidget(BuiltInWidgets.kNumberSlider) // specify the widget here
   .getEntry();
```

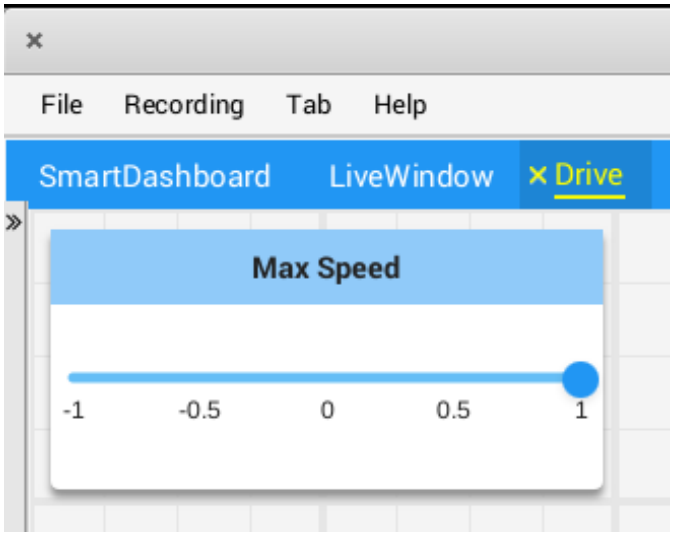
C++

```
frc::Shuffleboard::GetTab("Drive")
   .Add("Max Speed", 1)
   .WithWidget(frc::BuiltInWidgets::kNumberSlider) // specify the widget here
   .GetEntry();
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard
from wpilib.shuffleboard import BuiltInWidgets

(Shuffleboard.getTab("Drive")
   .add("Max Speed", 1)
   .withWidget(BuiltInWidgets.kNumberSlider) # specify the widget here
   .getEntry())
```

Dans cet exemple, nous configurons le widget « Max Speed » pour utiliser un curseur pour modifier les valeurs au lieu d'un champ de texte de base.



Définition des propriétés du widget

Étant donné que la vitesse maximale est significative seulement entre les valeurs 0 à 1 (arrêt complet à pleine vitesse), un curseur de -1 à 1 peut provoquer des problèmes si la valeur tombe en dessous de zéro. Heureusement, nous pouvons modifier cela en utilisant la méthode `withProperties`

JAVA

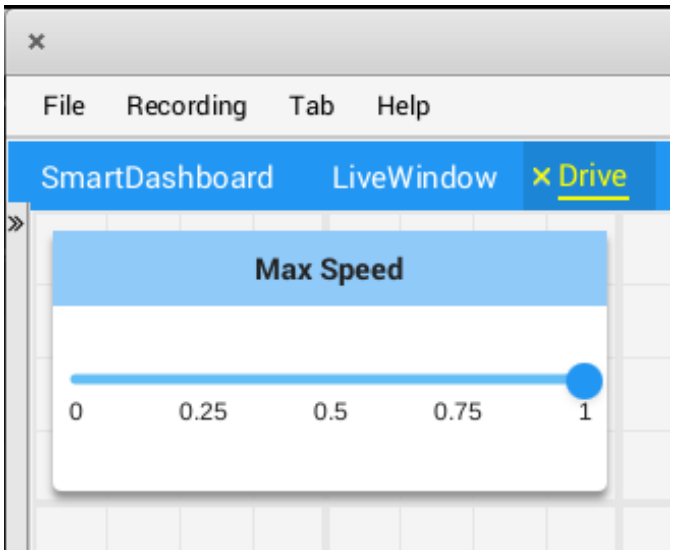
```
Shuffleboard.getTab("Drive")
   .add("Max Speed", 1)
   .withWidget(BuiltInWidgets.kNumberSlider)
   .withProperties(Map.of("min", 0, "max", 1)) // specify widget properties here
   .getEntry();
```

C++

```
frc::Shuffleboard::GetTab("Drive")
   .Add("Max Speed", 1)
   .WithWidget(frc::BuiltInWidgets::kNumberSlider)
   .WithProperties({ // specify widget properties here
      {"min", nt::Value::MakeDouble(0)},
      {"max", nt::Value::MakeDouble(1)}
   })
   .GetEntry();
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard
from wpilib.shuffleboard import BuiltInWidgets

(Shuffleboard.getTab("Drive")
   .add("Max Speed", 1)
   .withWidget(BuiltInWidgets.kNumberSlider)
   .withProperties(map("min", 0, "max", 1)) # specify widget properties here
   .getEntry())
```



Remarques

Les widgets peuvent être spécifiés par leur nom ; cependant, les noms sont sensibles à la casse et aux espaces (« Number Slider » est différent de « Number slider » et « NumberSlider »). Pour cette raison, il est recommandé d'utiliser la classe de widgets intégrée pour spécifier le widget plutôt que par nom brut. Cependant, un widget personnalisé ne peut être spécifié que par son nom ou en créant un `WidgetType` personnalisé pour ce widget.

Les noms de propriété de widget ne sont ni sensibles à la casse ni sensibles aux espaces (« Max » et « max » sont identiques). Consultez la documentation sur le widget dans la classe BuiltInWidgets pour plus de détails sur les propriétés de ce widget.

Organisation des widgets

Définition de la taille et de la position d'un widget

Appelez `withSize` et `withPosition` pour définir la taille et la position du widget dans l'onglet.

`withSize` définit la largeur en colonnes et la hauteur en lignes du widget. Par exemple, appeler `withSize(1, 1)` fait en sorte que le widget occupe une seule cellule dans la grille. Notez que certains widgets ont une taille minimale qui peut être supérieure à la taille spécifiée, dans ce cas, le widget utilisera la taille minimale requise et ignorera la taille spécifiée par l'utilisateur.

`withPosition` définit la ligne et la colonne du coin supérieur gauche du widget. Les lignes et les colonnes sont toutes deux indexées à compter de 0, donc la ligne la plus en haut est le numéro 0 et la colonne la plus à gauche est également le numéro 0. Si la position d'un widget dans un onglet est spécifiée, chaque widget doit également avoir sa position définie pour éviter le chevauchement des widgets.

JAVA

```
Shuffleboard.getTab("Pre-round")
  .add("Auto Mode", autoModeChooser)
  .withSize(2, 1) // make the widget 2x1
  .withPosition(0, 0); // place it in the top-left corner
```

C++

```
frc::Shuffleboard::GetTab("Pre-round")
  .Add("Auto Mode", autoModeChooser)
  .WithSize(2, 1)
  .WithPosition(0,0);
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

(Shuffleboard.getTab("Pre-round")
  .add("Auto Mode", autoModeChooser)
  .withSize(2, 1) # make the widget 2x1
  .withPosition(0, 0)) # place it in the top-left corner
```

Ajout de widgets aux mises en page

S'il existe de nombreux widgets dans un onglet avec des données associées, il peut être utile de les placer dans des sous-groupes plus petits plutôt que de les placer de façon lâche dans l'onglet. Tout comme la façon dont la référence d'un onglet est récupérée avec `Shuffleboard.getTab`, une disposition à l'intérieur d'un onglet (ou même dans une autre disposition) peut être récupérée avec `ShuffleboardTab.getLayout`.

JAVA

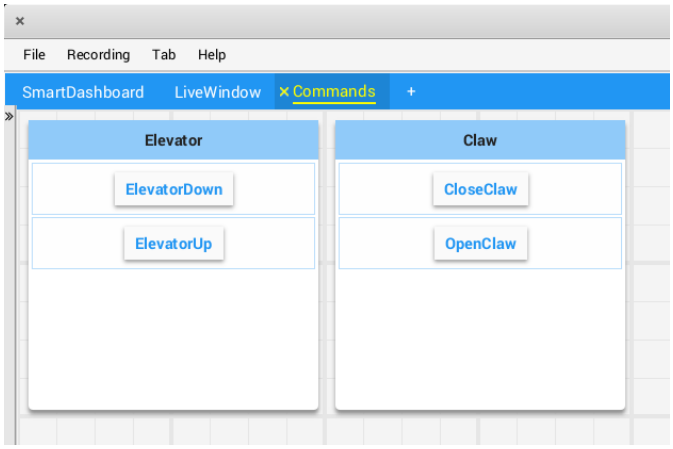
```
ShuffleboardLayout elevatorCommands = Shuffleboard.getTab("Commands")
  .getLayout("Elevator", BuiltInLayouts.kList)
  .withSize(2, 2)
  .withProperties(Map.of("Label position", "HIDDEN")); // hide labels for commands

elevatorCommands.add(new ElevatorDownCommand());
elevatorCommands.add(new ElevatorUpCommand());
```

C++

```
wpi::StringMap<std::shared_ptr<nt::Value>> properties{
  std::make_pair("Label position", nt::Value::MakeString("HIDDEN"))
};

frc::ShuffleboardLayout& elevatorCommands = frc::Shuffleboard::GetTab("Commands")
  .GetLayout("Elevator", frc::BuiltInLayouts::kList)
  .WithSize(2, 2)
  .WithProperties(properties);

ElevatorDownCommand* elevatorDown = new ElevatorDownCommand();
ElevatorUpCommand* elevatorUp = new ElevatorUpCommand();

elevatorCommands.Add("Elevator Down", elevatorDown);
elevatorCommands.Add("Elevator Up", elevatorUp);
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard
from wpilib.shuffleboard import BuiltInLayouts

(elevatorCommands = Shuffleboard.getTab("Commands")
  .getLayout("Elevator", BuiltInLayouts.kList)
  .withSize(2, 2)
  .withProperties(map("Label position", "HIDDEN"))) # hide labels for commands

elevatorCommands.add(ElevatorDownCommand())
elevatorCommands.add(ElevatorUpCommand())
```



11.2.3 Shuffleboard - Utilisation avancée

Commandes et sous-systèmes

Lorsque vous utilisez le framework basé sur des commandes, Shuffleboard facilite la compréhension de ce que fait le robot en affichant l'état des différentes commandes et sous-systèmes en temps réel.

Affichage des sous-systèmes

Pour voir l'état d'un sous-système pendant que le robot fonctionne en mode autonome ou téléopéré, c'est-à-dire quelle est sa commande par défaut et quelle commande utilise actuellement ce sous-système, envoyez une instance de sous-système à Shuffleboard :

JAVA

```
SmartDashboard.putData(subsystem-reference);
```

C++

```
SmartDashboard::PutData(subsystem-pointer);
```

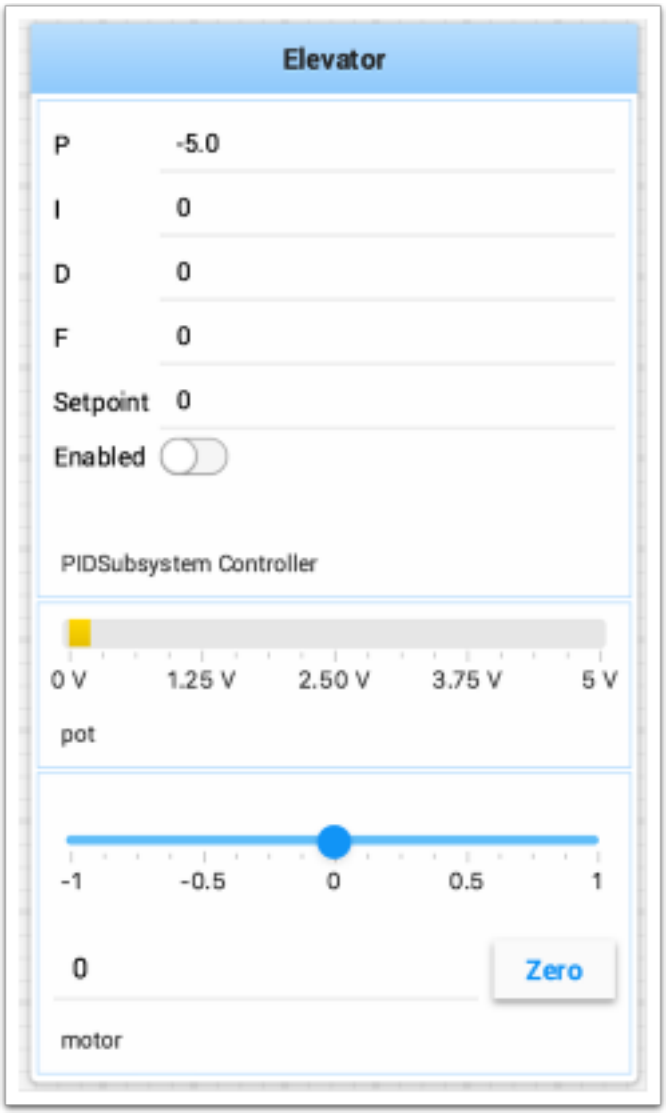
PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData(subsystem-reference)
```

Shuffleboard affichera le nom du sous-système, la commande par défaut associée à ce sous-système et la commande en cours d'exécution. Dans cet exemple, la commande par défaut pour le sous-système Elevator est appelée `AutonomousCommand` et c'est également la commande actuelle qui utilise le sous-système Elevator.

Sous-systèmes en mode test

En mode Test (Test/Enable sur la station de pilotage), des sous-systèmes peuvent être affichés dans l'onglet LiveWindow avec les capteurs et actionneurs du sous-système. C'est idéal pour vérifier que les capteurs fonctionnent en voyant les valeurs qu'ils renvoient. De plus, les actionneurs peuvent être commandés. Par exemple, les moteurs peuvent être actionnés à l'aide de curseurs pour définir leur vitesse et leur direction commandées. Pour les sous-systèmes PID, les constantes `P`, `I`, `D` et `F` sont affichées avec la valeur cible et un contrôle d'activation. Ceci est utile pour régler les sous-systèmes PID en ajustant les constantes, en insérant une valeur cible et en activant le contrôleur PID intégré. Ensuite, la réponse du mécanisme peut être observée. Ce cycle (modifier les paramètres, activer et observer) peut être répété jusqu'à ce qu'un ensemble raisonnable de paramètres soit trouvé.



Plus d'informations sur le réglage des sous-systèmes PID peuvent être trouvées *ici*. L'utilisation de RobotBuilder générera automatiquement le code pour afficher le sous-système en mode Test. Le code nécessaire pour afficher les sous-systèmes est indiqué ci-dessous, où « subsystem-name » est une chaîne de caractères contenant le nom du sous-système :

```
setName(subsystem-name);
```

Affichage des commandes

Using commands and subsystems makes very modular robot programs that can easily be tested and modified. Part of this is because commands can be written completely independently of other commands and can therefore be easily run from Shuffleboard. To write a command to Shuffleboard use the `SmartDashboard.putData` method as shown here :

JAVA

```
SmartDashboard.putData("ElevatorMove: up", new ElevatorMove(2.7));
```

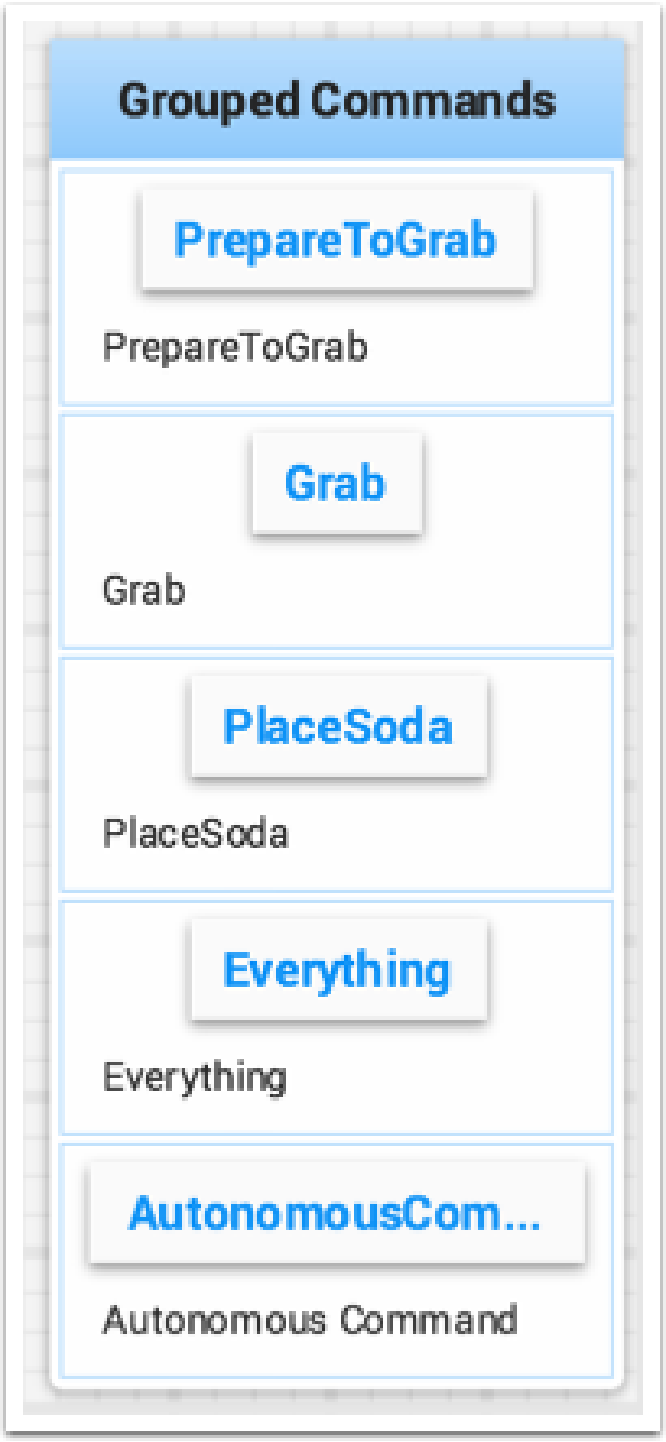
C++

```
SmartDashboard::PutData("ElevatorMove: up", new ElevatorMove(2.7));
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData("ElevatorMove: up", ElevatorMove(2.7))
```

Shuffleboard affichera le nom de la commande et un bouton pour exécuter la commande. De cette façon, les commandes individuelles et les groupes de commandes peuvent facilement être testés sans avoir besoin de code de test spécial dans un programme de robot. Dans l'image ci-dessous, il y a un certain nombre de commandes contenues dans une liste Shuffleboard. Appuyez une fois sur le bouton pour exécuter la commande et appuyez à nouveau pour l'arrêter. Pour utiliser cette fonction, le robot doit être activé en mode téléop.


Grouped Commands
PrepareToGrab
PrepareToGrab
Grab
Grab
PlaceSoda
PlaceSoda
Everything
Everything
AutonomousCom...
Autonomous Command

Test et réglage des boucles PID

One challenge in using sensors to control mechanisms is to have a good algorithm to drive the motors to the proper position or speed. The most commonly used control algorithm is called PID control. There is a good set of videos (look for the robot controls playlist) that explain the control algorithms described here. The PID algorithm converts sensor values into motor speeds by :

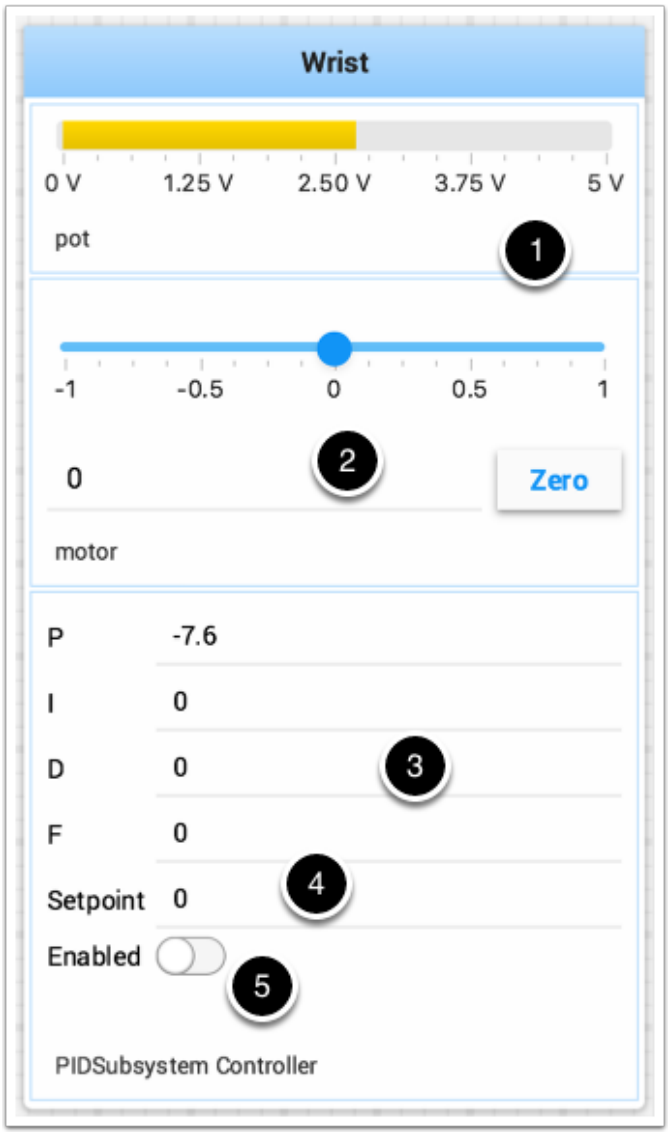
1. Lecture des valeurs du capteur pour déterminer la distance du robot ou du mécanisme par rapport à la valeur cible souhaitée. La cible est la valeur du capteur qui correspond à l'objectif attendu. Par exemple, un bras de robot avec un joint articulé similaire à un poignet, défini ci-dessous comme « poignet » pour alléger le texte, devrait pouvoir se déplacer très rapidement selon un angle spécifié et s'arrêter à cet angle, comme indiqué par un capteur. Un potentiomètre est un capteur qui peut mesurer l'angle de rotation. En le connectant à une entrée analogique, le programme peut obtenir une mesure de tension directement proportionnelle à l'angle.
2. Calcul de l'écart (la différence entre la valeur du capteur et la valeur souhaitée). Le signe de la valeur de l'écart indique de quel côté de la valeur cible le poignet se trouve. Par exemple, des valeurs négatives peuvent indiquer que l'angle de poignet mesuré est supérieur à l'angle de poignet souhaité. L'ampleur de l'écart est la différence entre l'angle du poignet mesuré et l'angle réel du poignet. Si l'écart est nul, l'angle mesuré correspond exactement à l'angle souhaité. L'écart peut être utilisé comme entrée de l'algorithme PID pour calculer la vitesse d'un moteur.
3. The resultant motor speed is then used to drive the motor in the correct direction and a speed that hopefully will reach the setpoint as quickly as possible without overshooting (moving past the setpoint).

WPILib a une classe PIDController qui supporte l'algorithme PID et accepte des constantes correspondant aux valeurs `Kp`, `Ki` et `Kd`. L'algorithme PID a trois composants qui contribuent au calcul de la vitesse du moteur à partir de l'écart.

1. `P` (proportionnel) - c'est un terme qui, multiplié par une constante (`Kp`), générera une vitesse du moteur qui aidera à déplacer le moteur dans la bonne direction et à la bonne vitesse.
2. `I` (intégrale) - ce terme est la somme des écarts successifs. Plus l'écart persiste, plus la contribution intégrale sera importante. Il s'agit simplement de la somme de tous les écarts dans le temps. Si le poignet n'atteint pas tout à fait la valeur cible en raison d'une grande charge qu'il essaie de déplacer, le terme intégral continuera d'augmenter (somme des écarts) jusqu'à ce qu'il contribue suffisamment à la vitesse du moteur pour le faire passer au point cible. La somme des écarts est multipliée par une constante (`Ki`) pour mettre à l'échelle le terme intégral du système.
3. `D` (différentiel) - cette valeur est le taux de variation des écarts. Elle est utilisée pour ralentir la vitesse du moteur s'il se déplace trop vite. Elle est calculée en prenant la différence entre la valeur d'écart actuelle et la valeur d'écart précédente. Elle est également multipliée par une constante (`Kd`) pour la mettre à l'échelle afin qu'elle corresponde au reste du système.

Réglage du contrôleur PID

Tuning the PID controller consists of adjusting constants for accurate results. Shuffleboard helps this process by displaying the details of a PID subsystem with a user interface for setting constant values and testing how well it operates. This is displayed while the robot is operating in test mode (done by setting « Test » in the driver station).



Il s'agit de l'image du mode Test d'un sous-système baptisé wrist qui représente un poignet articulé et qui a un potentiomètre comme capteur (pot) et un contrôleur de moteur connecté au moteur. Il a un certain nombre de champs qui correspondent au PIDSubsystem.

1. La valeur de la tension d'entrée analogique du potentiomètre. Il s'agit de la valeur d'entrée du capteur.
2. Un curseur qui déplace le moteur du poignet dans les deux sens avec 0 à l'arrêt. Les valeurs positives et négatives correspondent à un déplacement vers le haut et vers le bas.
3. Les constantes PID telles que décrites ci-dessus (F est une valeur à action directe utilisée pour les boucles PID de vitesse)

4. La valeur cible qui correspond à la valeur du pot lorsque le poignet a atteint la valeur souhaitée
5. Active le contrôleur PID - Ne fonctionne plus, voir ci-dessous.

Essayez différents gains PID pour obtenir les performances moteur souhaitées. Vous pouvez regarder la vidéo liée au début de cet article ou d'autres sources sur Internet pour obtenir les performances souhaitées.

Important : The enable option does not affect the PIDController introduced in 2020, as the controller is updated every robot loop. See the example below on how to retain this functionality.

Activer la fonctionnalité dans le nouveau PIDController

L'exemple suivant montre comment créer un bouton sur votre tableau de bord qui activera / désactivera le PIDController.

JAVA

```
ShuffleboardTab tab = Shuffleboard.getTab("Shooter");
GenericEntry shooterEnable = tab.add("Shooter Enable", false).getEntry();

// Command Example assumed to be in a PIDSubsystem
new NetworkButton(shooterEnable).onTrue(new InstantCommand(m_shooter::enable));

// Timed Robot Example
if (shooterEnable.getBoolean()) {
  // Calculates the output of the PID algorithm based on the sensor reading
  // and sends it to a motor
  motor.set(pid.calculate(encoder.getDistance(), setpoint));
}
```

C++

```
frc::ShuffleboardTab& tab = frc::Shuffleboard::GetTab("Shooter");
nt::GenericEntry& shooterEnable = *tab.Add("Shooter Enable", false).GetEntry();

// Command-based assumed to be in a PIDSubsystem
frc2::NetworkButton(shooterEnable).OnTrue(frc2::InstantCommand([&] { m_shooter.
↪Enable(); }));

// Timed Robot Example
if (shooterEnable.GetBoolean()) {
  // Calculates the output of the PID algorithm based on the sensor reading
  // and sends it to a motor
  motor.Set(pid.Calculate(encoder.GetDistance(), setpoint));
}
```

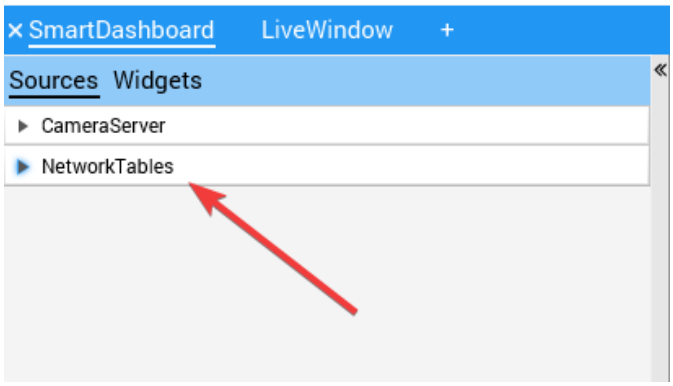
PYTHON

```
from wpilib.shuffleboard import Shuffleboard

tab = Shuffleboard.getTab("Shooter")
shooterEnable = tab.add("Shooter Enable", false).getEntry()

# Command Example assumed to be in a PIDSubsystem
NetworkButton(shooterEnable).onTrue(InstantCommand(m_shooter.enable()))

# Timed Robot Example
if (shooterEnable.getBoolean()):
  # Calculates the output of the PID algorithm based on the sensor reading
  # and sends it to a motor
  motor.set(pid.calculate(encoder.getDistance(), setpoint))
```

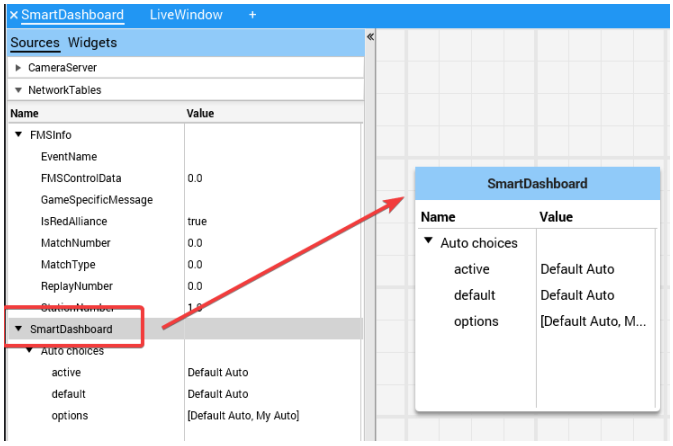
Affichage des hiérarchies de données

Faire glisser une clé avec d'autres clés en dessous (plus profondément dans la hiérarchie) affiche la hiérarchie dans une arborescence, similaire aux sources NetworkTables sur la gauche.

Sélectionnez la source de données :



Cliquez et faites glisser la clé NetworkTables dans l'onglet préféré.



11.2.4 Shuffleboard - Widgets personnalisés

Plugins intégrés

Shuffleboard fournit un certain nombre de plugins intégrés qui gèrent les tâches courantes pour FRC® use, comme les flux de caméra, tous les widgets et les connexions *NetworkTables*.

Plugin de base

Le plugin de base définit tous les types de données, widgets et mises en page nécessaires à l'utilisation de FRC. Il ne définit *pas* aucun des types de source, ni aucun type de données ou widget spécial pour ces types de source. Ceux-ci sont gérés par le *NetworkTables Plugin* et le *CameraServer Plugin*. Cette séparation des préoccupations permet aux équipes de créer plus facilement des plugins pour les types de sources ou protocoles personnalisés (par exemple HTTP, ZeroMQ) pour les types de données FRC sans avoir besoin d'un client NetworkTables.

Plug-in CameraServer

Le plugin de serveur de caméra fournit des sources et des widgets pour visualiser les flux de caméras de la classe WPILib `CameraServer`.

Ce plugin dépend du *NetworkTables Plugin* afin de découvrir les flux de caméras disponibles.

Découverte de flux

Les sources CameraServer sont automatiquement découvertes en consultant le NetworkTable `/CameraPublisher`.

```
/CameraPublisher
  /<camera name>
    streams=["url1", "url2", ...]
```

Par exemple, une caméra nommée « Camera » avec un serveur à `roborio-0000-frc.local` aurait cette disposition de table :

```
/CameraPublisher
  /Camera
    streams=["mjpeg:http://roborio-0000-frc.local:1181/?action=stream"]
```

Cette configuration découvrira automatiquement tous les flux de caméra hébergés sur un roboRIO par la classe CameraServer dans WPILib. Tous les projets non-WPILib qui souhaitent que les flux de caméra apparaissent dans le shuffleboard devront définir l'entrée de flux pour le serveur de caméra.

Plug-in NetworkTables

The NetworkTables plugin provides data sources backed by ntcore. Since the `LiveWindow`, `SmartDashboard`, and `Shuffleboard` classes in WPILib use NetworkTables to send the data to the driver station, this plugin will need to be loaded in order to use those classes.

Ce plugin gère automatiquement la connexion et la reconnexion à NetworkTables, les utilisateurs de shuffleboard et les auteurs de plugins personnalisés n'auront pas à se soucier des subtilités du protocole NetworkTables.

Création d'un plugin

Aperçu

Les plugins offrent la possibilité de créer des widgets personnalisés, des mises en page, des sources/types de données et des thèmes personnalisés. Shuffleboard fournit les éléments *plugins intégrés* suivants.

- Plugin NetworkTables : pour se connecter aux données publiées sur NetworkTables
- Base Plugin : Pour afficher des types de données FRC® personnalisés dans des gadgets personnalisés
- Plugin CameraServer : pour afficher les flux du CameraServer

---

Astuce : Un exemple de plugin Shuffleboard personnalisé qui crée un type de données personnalisé et un widget simple pour l'afficher peut être trouvé ici.

Créer un plugin personnalisé

Afin de définir un plugin, la classe de plugin doit être une sous-classe de edu.wpi.first.shuffleboard.api.Plugin ou l'une de ses sous-classes. Un exemple de classe plugin serait le suivant.

JAVA

```
import edu.wpi.first.shuffleboard.api.plugin.Description;
import edu.wpi.first.shuffleboard.api.plugin.Plugin;

@Description(group = "com.example", name = "MyPlugin", version = "1.2.3", summary =
↪"An example plugin")
public class MyPlugin extends Plugin {

}
```

Des explications supplémentaires sur la façon dont ces attributs sont utilisés, y compris les numéros de version, peuvent être trouvées ici.

Notez que l'annotation `@Description` est nécessaire pour indiquer au chargeur de plug-in les propriétés de la classe de plug-in personnalisée. Les classes de plugins sont autorisées à avoir un constructeur par défaut mais il ne peut accepter aucun argument.

Construction d'un plugin

The easiest way to build plugins is to utlize the *example-plugins* folder in the shuffleboard source tree. Clone Shuffleboard with `git clone https://github.com/wpilibsuite/shuffleboard.git`, and checkout the version that corresponds to the WPILib version you have installed (e.g. 2023.2.1). `git checkout v2023.2.1`

Put your plugin in the `example-plugins\PLUGIN-NAME` directory. Copy the `custom-data-and-widget.gradle` from `example-plugins\custom-data-and-widget` and rename to match your plugin name. Edit `settings.gradle` in the shuffleboard root directory to add `include "example-plugins:PLUGIN-NAME"`

Les plugins sont autorisés à avoir des dépendances avec d'autres plugins et librairies, cependant, ils doivent être inclus correctement dans le fichier de construction Maven ou Gradle. Lorsqu'un plugin dépend d'autres plugins, il est recommandé de définir ces dépendances afin que le plugin ne se charge pas lorsque les dépendances ne se chargent pas également. Cela peut être fait en utilisant l'annotation `@Requires` comme indiqué ci-dessous :

```
@Requires(group = "com.example", name = "Good Plugin", minVersion = "1.2.3")
@Requires(group = "edu.wpi.first.shuffleboard", name = "Base", minVersion = "1.0.0")
@Description(group = "com.example", name = "MyPlugin", version = "1.2.3", summary =
↪"An example plugin")
public class MyPlugin extends Plugin {

}
```

La `minVersion` spécifie la version minimale autorisée du plugin qui peut être chargée. Par exemple, si la `minVersion` est 1.4.5 et que le plugin avec la version 1.4.7 est chargé, il sera autorisé à le faire. Cependant, si le plugin avec la version 1.2.4 est chargé, il ne sera pas autorisé à le faire car il est inférieur à la `minVersion`.

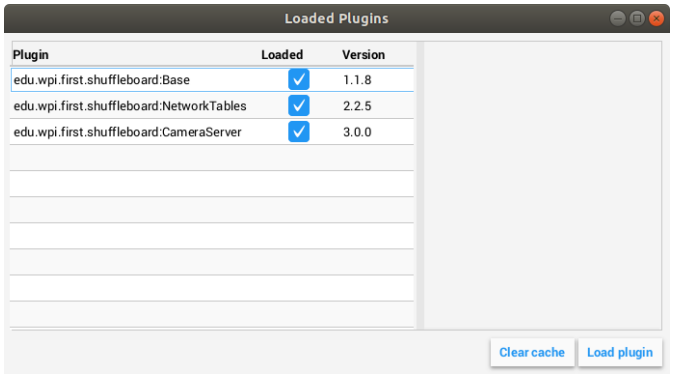
Déploiement du plugin au Shuffleboard

In order to load a plugin in Shuffleboard, you will need to generate a jar file of the plugin and put it in the `~/Shuffleboard/plugins` folder. This can be done automatically by running from the shuffleboard root `gradlew :example-plugins:PLUGIN-NAME:installPlugin`

Après le déploiement, le Shuffleboard mettra en cache le chemin du plugin afin qu'il puisse être automatiquement chargé lors du prochain chargement du Shuffleboard. Il peut être nécessaire de cliquer sur `Clear Cache` dans le menu des plugins pour supprimer un plugin ou recharger un plugin dans le Shuffleboard.

Ajout manuel d'un Plugin

The other way to add a plugin to Shuffleboard is to compile it to a jar file and add it from Shuffleboard. The jar file is located in `example-plugins\PLUGIN-NAME\build\libs` after running `gradlew build` in the shuffleboard root Open Shuffleboard, click on the file tab in the top left, and choose Plugins from the drop down menu.



Dans la fenêtre des plugins, choisissez le bouton « Load plugin » en bas à droite et sélectionnez votre fichier .jar.

Création de types de données personnalisés

Les widgets nous permettent de contrôler et de visualiser différents types de données. Ces données peuvent être des nombres entiers, doubles ou même des objets Java. Afin d'afficher ces types de données à l'aide de widgets, il est utile de créer une classe conteneur pour eux. Il n'est pas nécessaire de créer votre propre classe de données si le widget gère des types de données en champ unique tels que des doubles, des tableaux ou des chaînes de caractères..

Création de la classe de données

Dans cet exemple, nous allons créer un type de données personnalisé pour un point 2D et ses coordonnées x et y. Pour créer une classe de type de données personnalisée, elle doit étendre la classe abstraite ComplexData. Votre classe de données personnalisée doit également implémenter la méthode `asMap()` qui renvoie les données représentées sous la forme d'une carte simple, comme indiqué ci-dessous avec l'annotation `@Override` :

```
import edu.wpi.first.shuffleboard.api.data.ComplexData;

import java.util.Map;

public class MyPoint2D extends ComplexData<MyPoint2D> {

   private final double x;
   private final double y;

   //Constructor should take all the different fields needed and assign them their␣
↪corresponding instance variables.
   public MyPoint2D(double x, double y) {
      this.x = x;
      this.y = y;
   }

   @Override
   public Map<String, Object> asMap() {
      return Map.of("x", x, "y", y);
   }
}
```

Il est également recommandé de remplacer les méthodes par défaut `equals` et `hashcode` pour garantir que différents objets sont considérés comme équivalents lorsque leurs champs sont identiques. La méthode `asMap()` doit renvoyer les données représentées dans un simple objet Map car il sera mappé à l'entrée NetworkTables à laquelle il correspond. Dans ce cas, nous pouvons représenter le point comme ses coordonnées X et Y et renvoyer une `Map` les contenant.

```
import edu.wpi.first.shuffleboard.api.data.ComplexData;

import java.util.Map;

public final class MyPoint2D extends ComplexData<MyPoint2D> {

   private final double x;
   private final double y;

   // Constructor should take all the different fields needed and assign them to␣
↪their corresponding instance variables.
   public Point(double x, double y) {
      this.x = x;
      this.y = y;
   }

   @Override
   public Map<String, Object> asMap() {
      return Map.of("x", this.x, "y", this.y);
   }
 }
```

D'autres méthodes peuvent être ajoutées pour récupérer ou modifier des champs et des variables d'instance, mais il est recommandé de rendre ces classes immuables pour éviter de changer les objets de données source. Au lieu de cela, vous pouvez créer une copie du nouvel objet au lieu de manipuler l'objet existant. Par exemple, si nous voulions changer la coordonnée y de notre point, nous pourions définir la méthode suivante :

```
public MyPoint2D withY(double newY) {
   return new MyPoint2D(this.x, newY);
}
```

Cela crée un nouvel objet `MyPoint2D` et le renvoie avec la nouvelle coordonnée y. La même chose peut être faite pour changer la coordonnée x.

Création d'un type de données

Il existe deux types de données différents : les types de données simples qui n'ont qu'un seul champ (c'est-à-dire un seul nombre ou chaîne de caractères) et les types de données complexes, qui ont plusieurs champs de données (c'est-à-dire plusieurs chaînes, plusieurs nombres).

Afin de définir un type de données simple, la classe doit étendre la classe `SimpleDataType<DataType>` avec le type de données requis et implémenter la méthode `getDefaultValue()`. Dans cet exemple, nous utiliserons un double comme type de données simple.

```
public final class MyDoubleDataType extends SimpleDataType<Double> {
```

(suite sur la page suivante)

(suite de la page précédente)

```
   private static final String NAME = "Double";

   private MyDataType() {
      super(NAME, Double.class);
   }

   @Override
   public Double getDefaultValue() {
      return 0.0;
   }

}
```

Le constructeur de classe est défini comme private pour garantir qu'une seule instance du type de données existera.

Afin de définir un type de données complexe, la classe doit étendre la classe `ComplexDataType` et remplacer les méthodes `fromMap()` et `getDefaultValue()`. Nous utiliserons notre classe MyPoint2D comme exemple pour voir à quoi ressemblerait une classe de type de données complexe.

```
public final class PointDataType extends ComplexDataType<MyPoint2D> {

   private static final String NAME = "MyPoint2D";
   public static final PointDataType Instance = new PointDataType();

   private PointDataType() {
      super(NAME, MyPoint2D.class);
   }

   @Override
   public Function<Map<String, Object>, MyPoint2D> fromMap() {
      return map -> {
         return new MyPoint2D((double) map.getOrDefault("x", 0.0), (double) map.
↪getOrDefault("y", 0.0));
      };
   }

   @Override
   public MyPoint2D getDefaultValue() {
      // use default values of 0 for X and Y coordinates
      return new MyPoint2D(0, 0);
   }

}
```

Le code suivant ci-dessus fonctionne comme indiqué :

La méthode `fromMap()` crée un nouveau MyPoint2D en utilisant les valeurs de l'entrée NetworkTables à laquelle il est lié. La méthode `getOrDefault` renverra 0.0 si elle ne peut pas obtenir les valeurs d'entrée. Le `getDefaultValue` renverra un nouvel objet `MyPoint2D` si aucune source n'est présente.

Exportation du type de données vers le plugin

Afin que le type de données soit reconnu par le Shuffleboard, le plugin doit les exporter en remplaçant la méthode `getDataTypes`. Par exemple,

```
public class MyPlugin extends Plugin {

   @Override
   public List<DataType> getDataTypes() {
      return List.of(PointDataType.Instance);
   }

}
```

Création d'un widget

Les widgets nous permettent de visualiser, de modifier et d'interagir avec les données publiées via différentes sources de données. Les plug-ins CameraServer, NetworkTables et Base fournissent les widgets pour contrôler les types de données de base (y compris les types de données spécifiques à FRC). Cependant, les widgets personnalisés nous permettent de contrôler nos types de données personnalisés que nous avons créés dans les sections précédentes ou les objets Java.

L'interface de base `Widget` hérite des interfaces `Component` et `Sourced` . `Component` est le bloc de construction le plus élémentaire des composants à afficher dans Shuffleboard. `Sourced` est une interface pour les choses qui peuvent gérer et interagir avec des sources de données pour afficher ou modifier des données. Les widgets qui ne prennent pas en charge les liaisons de données mais qui ont simplement des nœuds-enfants n'utiliseraient pas l'interface « Sourced » mais simplement l'interface `Component` . Les deux sont des éléments de base pour créer des widgets et nous permettent de modifier et d'afficher des données.

Un bon widget permet à l'utilisateur de personnaliser le widget en fonction de ses besoins. Un exemple pourrait être de permettre à l'utilisateur de contrôler la plage d'un curseur numérique, c'est-à-dire son maximum et son minimum ou l'orientation du curseur lui-même. La vue du widget ou son apparence est définie à l'aide de FXML. FXML est un langage basé sur XML qui est utile pour définir la disposition statique du widget (volets, étiquettes et contrôles).

Plus d'informations sur FXML peuvent être trouvées ici.

Définition du FXML d'un widget

Dans cet exemple, nous allons créer deux curseurs pour nous aider à contrôler les coordonnées X et Y de notre type de données Point2D que nous avons créé dans les sections précédentes. Il est utile de placer le fichier FXML dans le même package que la classe Java.

Afin de créer une fenêtre vide et initiale pour notre widget, nous devons créer un volet, ou `Pane`. Un volet est un nœud-parent qui contient d'autres nœuds-enfants, dans ce cas, 2 curseurs. Il existe de nombreux types de volets différents, comme indiqué :

- Volet de pile -Stack-
  - Les panneaux de pile permettent de superposer des éléments. En outre, StackPanes par défaut des nœuds enfants centraux.
- Volet de grille

    - Les volets de grille sont extrêmement utiles pour définir des éléments- enfants à l'aide d'un système de coordonnées en créant une grille flexible de lignes et de colonnes sur le volet.
- Volet de flux
    - Les volets de flux enveloppent tous les nœuds enfants-dans un ensemble de limites. Les nœuds-enfants peuvent circuler verticalement (enveloppés à la limite de hauteur du volet) ou horizontalement (enveloppés à la limite de largeur du volet).
- Volet d'ancrage
    - Les volets d'ancrage permettent de placer des éléments-enfants en haut, en bas, à gauche, à droite ou au centre du volet.

Les volets de disposition sont extrêmement utiles pour placer des nœuds- enfants sur une ligne horizontale à l'aide d'un HBox ou d'une verticale à l'aide d'une colonne VBox.

La syntaxe de base pour définir un volet à l'aide de FXML serait la suivante :

```
<?import javafx.scene.layout.*?>
<StackPane xmlns:fx="http://javafx.com/fxml/1" fx:controller="/path/to/widget/class"␣
↪fx:id="root">
   ...
</StackPane>
```

L'attribut `fx:controller` contient le nom de la classe de widget. Une instance de cette classe est créée lors du chargement du fichier FXML. Pour que cela fonctionne, la classe de contrôleur doit avoir un constructeur sans argument.

Création d'une classe de widgets

Maintenant que nous avons un volet, nous pouvons maintenant ajouter des éléments-enfants à ce volet. Dans cet exemple, nous pouvons ajouter deux objets curseur. N'oubliez pas d'ajouter un `fx:id` à chaque élément afin qu'ils puissent être référencés dans notre classe Java que nous ferons plus tard. Nous utiliserons une `VBox` pour positionner notre curseur l'un sur l'autre.

```
<?import javafx.scene.layout.*?>
<StackPane xmlns:fx="http://javafx.com/fxml/1" fx:controller="/path/to/widget/class"␣
↪fx:id="root">

   <VBox>
      <Slider fx:id = "xSlider"/>
      <Slider fx:id = "ySlider"/>
   </VBox>

</StackPane>
```

Maintenant que nous avons fini de créer notre fichier FXML, nous pouvons maintenant créer une classe de widget. La classe de widget doit inclure une annotation `@Description` qui indique les types de données pris en charge du widget et le nom du widget. Si une annotation `@Description` n'est pas présente, la classe du plugin doit implémenter la méthode `get()` pour retourner ses widgets.

Elle doit également inclure une annotation `@ParametrizedController` qui pointe vers le fichier FXML contenant la mise en page du widget. Si la classe qui ne prend en charge qu'une seule source de données, elle doit étendre la classe `SimpleAnnotatedWidget`. Si la classe prend en charge plusieurs sources de données, elle doit étendre la classe `ComplexAnnotatedWidget`. Pour plus d'informations, voir *Types de widgets*.

```
import edu.wpi.first.shuffleboard.api.widget.Description;
import edu.wpi.first.shuffleboard.api.widget.ParametrizedController;
import edu.wpi.first.shuffleboard.api.widget.SimpleAnnotatedWidget;

/*
 * If the FXML file and Java file are in the same package, that is the Java file is␣
↪in src/main/java and the
 * FXML file is under src/main/resources or your code equivalent package, the␣
↪relative path will work
 * However, if they are in different packages, an absolute path will be required.
*/

@Description(name = "MyPoint2D", dataTypes = MyPoint2D.class)
@ParametrizedController("Point2DWidget.fxml")
public final class Point2DWidget extends SimpleAnnotatedWidget<MyPoint2D> {

}
```

Si vous n'utilisez pas de type de données personnalisées, vous pouvez référencer n'importe quel type de données Java (par exemple, `Double.class`), ou si le widget n'a pas besoin de liaison de données, vous pouvez passer `NoneType.class`.

Maintenant que nous avons créé notre classe, nous pouvons créer des champs pour les widgets que nous avons déclarés dans notre fichier FXML en utilisant l'annotation `@FXML`. Pour nos deux curseurs, un exemple serait :

```
import edu.wpi.first.shuffleboard.api.widget.Description;
import edu.wpi.first.shuffleboard.api.widget.ParametrizedController;
import edu.wpi.first.shuffleboard.api.widget.SimpleAnnotatedWidget;
import javafx.fxml.FXML;

@Description(name = "MyPoint2D", dataTypes = MyPoint2D.class)
@ParametrizedController("Point2DWidget.fxml")
public final class Point2DWidget extends SimpleAnnotatedWidget<MyPoint2D> {

   @FXML
   private Pane root;

   @FXML
   private Slider xSlider;

   @FXML
   private Slider ySlider;
}
```

Afin d'afficher notre volet sur notre widget personnalisé, nous devons remplacer la méthode `getView()` et renvoyer notre `StackPane`.

```
import edu.wpi.first.shuffleboard.api.widget.Description;
import edu.wpi.first.shuffleboard.api.widget.ParametrizedController;
import edu.wpi.first.shuffleboard.api.widget.SimpleAnnotatedWidget;
import javafx.fxml.FXML;

@Description(name = "MyPoint2D", dataTypes = MyPoint2D.class)
@ParametrizedController("Point2DWidget.fxml")
public final class Point2DWidget extends SimpleAnnotatedWidget<MyPoint2D> {

```

(suite sur la page suivante)

(suite de la page précédente)

```
   @FXML
   private StackPane root;

   @FXML
   private Slider xSlider;

   @FXML
   private Slider ySlider;

   @Override
   public Pane getView() {
      return root;
   }

}
```

Lier des éléments et ajouter des écouteurs

Binding is a mechanism that allows JavaFX widgets to express direct relationships with the data source. For example, changing a widget will change its related NetworkTableEntry and vice versa.

Un exemple, dans ce cas, serait de changer les coordonnées X et Y de notre point 2D en changeant les valeurs de xSlider et ySlider respectivement.

Une bonne pratique consiste à définir des liaisons dans la méthode `initialize()` étiquetée avec l'annotation `@FXML` qui est nécessaire pour appeler la méthode depuis FXML si la méthode n'est pas déclarée `public`.

```
import edu.wpi.first.shuffleboard.api.widget.Description;
import edu.wpi.first.shuffleboard.api.widget.ParametrizedController;
import edu.wpi.first.shuffleboard.api.widget.SimpleAnnotatedWidget;
import javafx.fxml.FXML;

@Description(name = "MyPoint2D", dataTypes = MyPoint2D.class)
@ParametrizedController("Point2DWidget.fxml")
public final class Point2DWidget extends SimpleAnnotatedWidget<MyPoint2D> {

   @FXML
   private StackPane root;

   @FXML
   private Slider xSlider;

   @FXML
   private Slider ySlider;

   @FXML
   private void initialize() {
      xSlider.valueProperty().bind(dataOrDefault.map(MyPoint2D::getX));
      ySlider.valueProperty().bind(dataOrDefault.map(MyPoint2D::getY));
   }

   @Override
   public Pane getView() {
```

(suite sur la page suivante)

(suite de la page précédente)

```
      return root;
   }

 }
```

The above `initialize` method binds the slider's value property to the `MyPoint2D` data class" corresponding X and Y value. Meaning, changing the slider will change the coordinate and vice versa. The `dataOrDefault.map()` method will get the data source's value, or, if no source is present, will return the default value.

L'utilisation d'un écouteur est une autre façon de modifier les valeurs lorsque le curseur ou la source de données a changé. Par exemple, un écouteur pour notre curseur serait :

```
xSlider.valueProperty().addListener((observable, oldValue, newValue) ->␣
↪setData(getData().withX(newValue));
```

Dans ce cas, la méthode `setData()` définit la valeur de la source de données du widget sur la `newValue`.

Explorer les composants personnalisés

Les widgets ne sont pas automatiquement trouvés lors du chargement des plugins ; le plugin de définition doit l'exporter explicitement pour qu'il soit utilisable. Cette approche est adoptée pour permettre à plusieurs plugins d'être définis dans le même JAR.

```
@Override
public List<ComponentType> getComponents() {
  return List.of(WidgetType.forAnnotatedWidget(Point2DWidget.class));
}
```

Définir le widget par défaut pour le type de données

Afin de définir votre widget par défaut pour votre type de données personnalisé, vous pouvez remplacer le `getDefaultComponents()` dans votre classe de plugin qui stocke une carte pour tous les widgets par défaut comme indiqué ci-dessous :

```
@Override
public Map<DataType, ComponentType> getDefaultComponents() {
   return Map.of(Point2DType.Instance, WidgetType.forAnnotatedWidget(Point2DWidget.
↪class));
}
```

Thèmes personnalisés

Since shuffleboard is a JavaFX application, it has support for custom themes via Cascading Stylesheets (**CSS** for short). These are commonly used on webpages for making HTML look nice, but JavaFX also has support, albeit for a different language subset (see here for documentation on how to use it).

Shuffleboard est livré avec trois thèmes par défaut : Material Light, Material Dark et Midnight. Ce sont des variations de couleur sur la même feuille de style de conception matérielle. De plus, ils héritent d'une feuille de style `base.css` qui définit les styles des composants personnalisés, définis dans Shuffleboard ou les librairies qu'il utilise ; la feuille de style de conception du matériau de base s'applique uniquement aux composants d'interface utilisateur intégrés à JavaFX.

Il existe deux façons de définir un thème personnalisé : placer les feuilles de style dans un répertoire avec le nom du thème dans `~/Shuffleboard/themes` ; par exemple, un thème théorique « Yellow » pourrait être placé dans

```
~/Shuffleboard/themes/Yellow/yellowtheme.css
```

Toutes les feuilles de style du répertoire seront traitées comme faisant partie du thème.

Chargement de thèmes via des plugins

Les thèmes personnalisés peuvent également être définis par des plugins. Cela les rend plus faciles à partager et à regrouper avec des widgets personnalisés, mais est légèrement plus difficile à définir. L'objet de thème aura besoin d'une référence à une classe définie dans le plugin afin que le chargeur de plugin puisse déterminer où se trouvent les feuilles de style. Si une classe est transmise qui n'est *pas* présente dans le JAR dans lequel se trouve le plugin, le thème ne pourra pas être utilisé.

```
@Description(group = "com.example", name = "My Plugin", version = "1.2.3", summary = "
↪")
class MyPlugin extends Plugin {

  private static final Theme myTheme = new Theme(MyPlugin.class, "My Theme Name", "/
↪path/to/stylesheet", "/path/to/stylesheet", ...);

  @Override
  public List<Theme> getThemes() {
    return ImmutableList.of(myTheme);
  }

}
```

Modifier ou étendre les thèmes par défaut du Shuffleboard

Les thèmes Material Light et Material Dark de Shuffleboard fournissent une grande partie du cadre pour les thèmes clairs et sombres, respectivement, ainsi que de nombreux styles spécifiques aux composants de shuffleboard, ControlsFX et Medusa UI pour s'adapter à la conception selon le style du matériau.

Les thèmes qui souhaitent modifier ceci doivent ajouter des instructions `import` pour ces feuilles de style :

```
@import "/edu/wpi/first/shuffleboard/api/material.css"; /* Material design CSS for␣
↪JavaFX components */
@import "/edu/wpi/first/shuffleboard/api/base.css";  /* Material design CSS for␣
↪shuffleboard components */
@import "/edu/wpi/first/shuffleboard/app/light.css"; /* CSS for the Material Light␣
↪theme */
@import "/edu/wpi/first/shuffleboard/app/dark.css";  /* CSS for the Material Dark␣
↪theme */
@import "/edu/wpi/first/shuffleboard/app/midnight.css";  /* CSS for the Midnight␣
↪theme */
```

Notez que `base.css` importe en interne `material.css`, et `light.css`, `dark.css`, et `midnight.css` importent tous `base.css`, donc l'importation de `light.css` importera implicitement à la fois `base.css` and `material.css`.

Code source pour les fichiers CSS

- _material.css : https://github.com/wpilibsuite/shuffleboard/blob/main/api/src/main/resources/edu/wpi/first/shuffleboard/api/material.css
- _base.css : https://github.com/wpilibsuite/shuffleboard/blob/main/api/src/main/resources/edu/wpi/first/shuffleboard/api/base.css
- _light.css : https://github.com/wpilibsuite/shuffleboard/blob/main/app/src/main/resources/edu/wpi/first/shuffleboard/app/light.css
- _dark.css : https://github.com/wpilibsuite/shuffleboard/blob/main/app/src/main/resources/edu/wpi/first/shuffleboard/app/dark.css
- _midnight.css : https://github.com/wpilibsuite/shuffleboard/blob/main/app/src/main/resources/edu/wpi/first/shuffleboard/app/midnight.css

Échantillons de couleur de conception matérielle

Le CSS de conception matérielle utilise des variables d'échantillons de couleurs pour presque tout. Ces variables peuvent être définies à partir de fichiers CSS personnalisés, ce qui réduit la quantité de code personnalisé nécessaire.

Les variables `-swatch- <100 | 200 | 300 | 400 | 500>` définissent des nuances progressivement plus foncées de la même couleur primaire. Le thème clair utilise les nuances de bleu par défaut définies dans `material.css`, mais le thème sombre les remplace par des nuances de rouge. `-swatch-<|light|dark>-gray` définit trois niveaux de gris à utiliser pour diverses couleurs d'arrière-plan ou de texte.

Remplacer les couleurs de l'échantillon

Remplacement du bleu par du rouge (clair)

```
@import "/edu/wpi/first/shuffleboard/app/light.css"

.root {
    -swatch-100: hsb(0, 80%, 98%);
    -swatch-200: hsb(0, 80%, 88%);
    -swatch-300: hsb(0, 80%, 78%);
    -swatch-400: hsb(0, 80%, 68%);
    -swatch-500: hsb(0, 80%, 58%);
}
```

Remplacement du rouge par du bleu (foncé)

```
@import "/edu/wpi/first/shuffleboard/app/dark.css"

.root {
    -swatch-100: #BBDEFB;
    -swatch-200: #90CAF9;
    -swatch-300: #64BEF6;
    -swatch-400: #42A5F5;
    -swatch-500: #2196F3;
}
```

Types de widgets

Bien qu'un `Widget` soit assez simple pour ce qui concerne son interface, il existe cependant plusieurs implémentations intermédiaires pour faciliter la définition du widget.

Classe	Description
`AbstractWidget`	Implemente `getProperties()`, `getSources()`, et `titleProperty()`
`SingleTypeWidget1`	Ajoute des propriétés pour les widgets qui ne prennent en charge qu'un seul type de données
`AnnotatedWidget`	Ajoute des implémentations par défaut pour `getName()` et `getDataTypes()` pour les widgets avec une annotation `@Description`
`SingleSourceWidget`	Pour les widgets avec une seule source (par défaut, les widgets prennent en charge plusieurs sources)
`SimpleAnnotatedWidget1<T>`	Combine `SingleTypeWidget1`, `AnnotatedWidget`, et `SingleSourceWidget`

Il existe également deux annotations pour aider à définir les widgets :

Nom	Description
`@ParametrizedController`	Permet aux widgets d'être des contrôleurs FXML pour les vues JavaFX définies via FXML
`@Description`	Permet de définir le nom et les types de données pris en charge sur une seule ligne

AbstractWidget

Cette classe implémente `getProperties()`, `getSources()`, `addSource()`, et `titleProperty()`. Elle définit également une méthode `exportProperties(Property<?>...)` de telle sorte que les sous-classes peuvent facilement ajouter des propriétés widget personnalisé, ou des propriétés pour les composants JavaFX dans le widget. La plupart des widgets dans le plugin de base l'utilisent.

SingleTypeWidget

Un type de widget qui ne prend en charge qu'un seul type de données. Cette interface est paramétrée et possède des méthodes pour définir ou obtenir les données, ainsi qu'une méthode pour obtenir le type de données (unique) du widget.

AnnotatedWidget

This interface implements `getDataTypes()` and `getName()` by looking at the `@Description` annotation on the implementing class. This *requires* the annotation to be present, or the widget will not be able to be loaded and used.

```
// No @Description annotation!
public class WrongImplementation implements AnnotatedWidget {
  // ...
}
```

```
@Description(name = ..., dataTypes = ...)
public class CorrectImplementation implements AnnotatedWidget {
  // ...
}
```

SingleSourceWidget

Un type de widget qui n'utilise qu'une seule source.

SimpleAnnotatedWidget

Une combinaison de `SingleTypeWidget<T>`, `AnnotatedWidget`, et `SingleSourceWidget`. La plupart des widgets du plugin de base proviennent de cette classe. Cela a également un champ `protected` appelé `dataOrDefault` qui permet aux sous-classes d'utiliser une valeur de données par défaut si le widget n'a pas de source, ou si la source fournit `null`.

@ParametrizedController

Cette annotation peut être placée sur une classe de widget pour indiquer au shuffleboard qu'il s'agit d'un contrôleur FXML pour une vue JavaFX définie via FXML. L'annotation prend un seul paramètre qui définit où le fichier FXML *par rapport à la classe sur laquelle il est placé*. Par exemple, un widget dans le répertoire `src/main/java/com/acme` qui est un contrôleur FXML pour un fichier FXML dans `src/main/resources/com/acme` peut utiliser l'annotation comme

```
@ParametrizedController("MyWidget.fxml")
```

ou comme

```
@ParametrizedController("/com/acme/MyWidget.fxml")
```

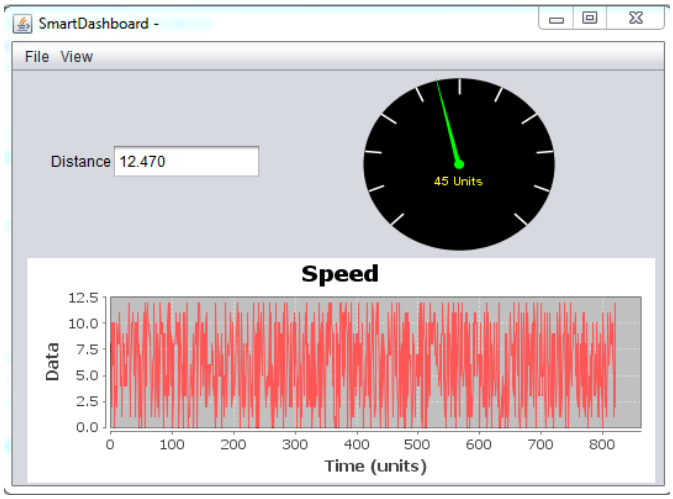
@Description

Cela permet aux widgets d'avoir leur nom et les types de données pris en charge définis par une seule annotation, lorsqu'ils sont utilisés avec *AnnotatedWidget*.

11.3 SmartDashboard

SmartDashboard is a simple and efficient dashboard that uses relatively few computer resources. It does not have the fancy look or some of the features Shuffleboard has, but it displays network tables data with a variety of widgets without bogging down the driver station computer.

11.3.1 Introduction à SmartDashboard

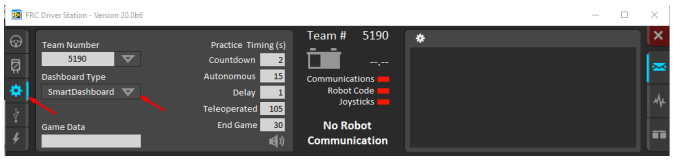


SmartDashboard est une application conçue en Java qui permet l'affichage en temps réel des données du robot. SmartDashboard est utile dans les situations suivantes :

- L'affichage, pendant l'exécution du programme, des données d'intérêt de votre robot. Ces données peuvent être affichées sous forme de simples champs de texte ou, de manière plus élaborée, sous forme de nombreux autres types d'affichage comme des graphiques, des cadrans, etc.
- L'affichage de l'état du programme du robot comme les commandes en cours d'exécution et l'état de tous les sous-systèmes
- L'affichage des boutons que vous pouvez actionner pour faire varier des valeurs de paramètres de votre robot.
- La sélection d'options de démarrage depuis le tableau de bord pouvant ensuite être lues par le programme du robot

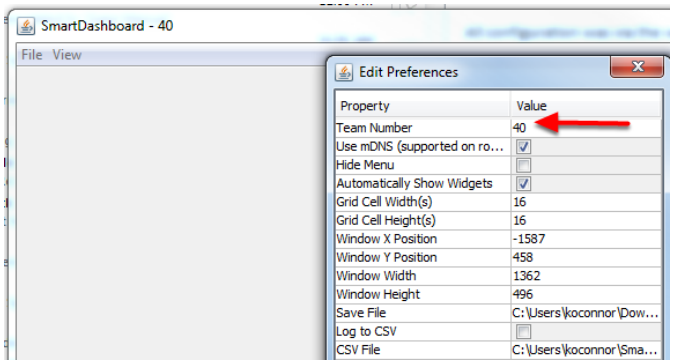
Les données affichées sont automatiquement mises en forme en temps réel lorsqu'elles sont envoyées à partir du robot, mais vous pouvez modifier le format ou les types de widget d'affichage, puis vous pouvez enregistrer les nouvelles dispositions d'écran en vue d'une utilisation ultérieure. SmartDashboard demeure extrêmement simple à utiliser. Pour afficher une donnée au tableau de bord, il suffit d'appeler l'une des méthodes de la classe SmartDashboard avec la donnée et son nom, et la valeur apparaîtra automatiquement au tableau de bord.

Installation de SmartDashboard



SmartDashboard est intégré dans l'installateur WPILib et peut être lancé directement à partir de Driver Station en sélectionnant le bouton **SmartDashboard** sous l'onglet Setup.

Configuration du numéro d'équipe



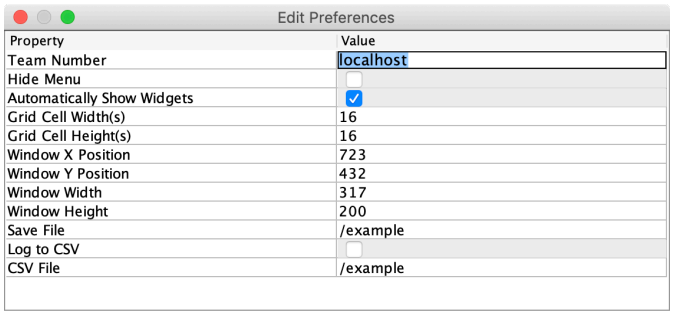
La première fois que vous lancez le SmartDashboard, vous êtes invité à configuer votre numéro d'équipe. Pour modifier le numéro d'équipe par la suite : cliquez sur **File > Preferences** pour ouvrir le dialogue Preferences. Double-cliquez sur la boîte à droite de **Team Number** et entrez votre FRC ® Numéro d'équipe, puis cliquez en dehors de la boîte pour enregistrer.

Note : SmartDashboard prendra un peu de temps pour finaliser la configuration du numéro d'équipe, ne vous inquiétez pas.

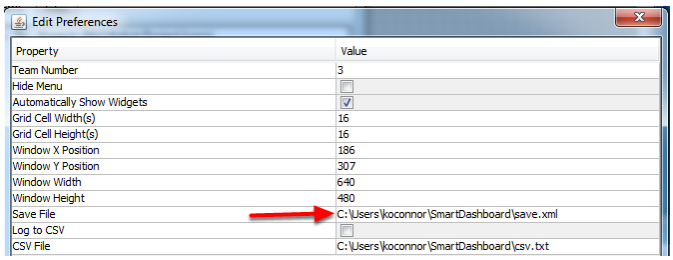
Définition d'un emplacement de serveur NetworkTables personnalisé

Par défaut, SmartDashboard recherchera les instances NetworkTables s'exécutant sur un RoboRIO connecté, mais il est parfois utile de rechercher des NetworkTables à une adresse IP différente. Pour vous connecter à SmartDashboard depuis un hôte autre que le roboRIO, ouvrez les préférences de SmartDashboard dans le menu `File` et dans le champ `Team Number`, saisissez l'adresse IP ou le nom d'hôte de l'hôte NetworkTables.

Cette option est incroyablement utile pour utiliser SmartDashboard avec WPILib simulation. Ajoutez simplement `localhost` au champ `Team Number` et SmartDashboard détectera votre robot localement !

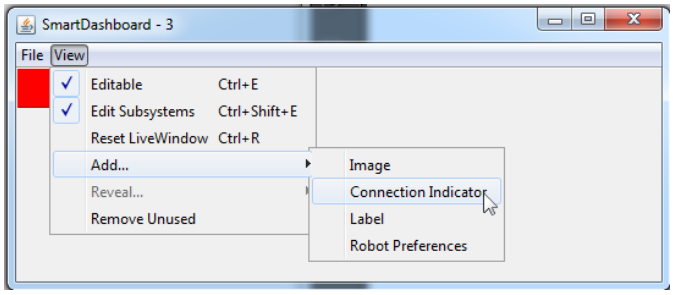


Localisation du fichier sauvegardé



L'utilisateur peut déterminer un repertoire de sauvegarde de SmartDashboard autre que le repertoire par défaut. Pour ce faire, cliquez sur la case voisine de **Save File** puis accédez au dossier dans lequel vous souhaitez placer la configuration. Les fichiers enregistrés dans les répertoires d'installation des composants WPILib seront probablement remplacés pendant les mises à jour de ces composants.

Ajout d'un indicateur de connexion



Il est souvent utile de vérifier si SmartDashboard est connecté au robot ou pas. Pour ajouter un indicateur de connexion, sélectionnez **View > Add > Connection Indicator**. Cet indicateur sera rouge lorsqu'il est déconnecté et vert lorsqu'il est connecté. Pour déplacer ou redimensionner cet indicateur, sélectionnez **View > Editable** pour basculer SmartDashboard en mode modifiable, puis faites glisser le centre de l'indicateur pour le déplacer ou les bords à redimensionner. Sélectionnez à nouveau l'élément **Editable** pour le verrouiller en place.

Ajout de widgets à SmartDashboard

Des widgets sont automatiquement ajoutés à SmartDashboard pour chaque « clé » envoyée par le code robot. Pour des instructions sur l'ajout de code robot à écrire pour SmartDashboard voir *Displaying Expressions from Within the Robot Program*.

11.3.2 Affichage des expressions à partir du programme du robot

---

Note : Souvent, le débogage ou la surveillance de l'état du robot implique d'écrire un certain nombre de valeurs à la console et de les visionner par streaming. Grâce à SmartDashboard, vous pouvez afficher des valeurs dans un GUI qui est automatiquement généré par votre programme. Au fur et à mesure que les valeurs sont mises à jour, les éléments correspondants du GUI changent de valeurs - il n'est pas nécessaire d'intercepter ces valeurs en streaming à la console.

Écriture de valeurs sur SmartDashboard

JAVA

```
protected void execute() {
   SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge());
   SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition());
   SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle());
   SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder());
   SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder());
   SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle());
   SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage());
   SmartDashboard.putNumber("RPM", shooter.getRPM());
}
```

C++

```
void Command::Execute() {
   frc::SmartDashboard::PutBoolean("Bridge Limit", BridgeTipper.AtBridge());
   frc::SmartDashboard::PutNumber("Bridge Angle", BridgeTipper.GetPosition());
   frc::SmartDashboard::PutNumber("Swerve Angle", Drivetrain.GetSwerveAngle());
   frc::SmartDashboard::PutNumber("Left Drive Encoder", Drivetrain.GetLeftEncoder());
   frc::SmartDashboard::PutNumber("Right Drive Encoder", Drivetrain.
↪GetRightEncoder());
   frc::SmartDashboard::PutNumber("Turret Pot", Turret.GetCurrentAngle());
   frc::SmartDashboard::PutNumber("Turret Pot Voltage", Turret.GetAverageVoltage());
   frc::SmartDashboard::PutNumber("RPM", Shooter.GetRPM());
}
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge())
SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition())
SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle())
SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder())
SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder())
SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle())
SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage())
SmartDashboard.putNumber("RPM", shooter.getRPM())
```

Vous pouvez écrire des valeurs booléennes, numériques ou des chaînes de caractères sur SmartDashboard en appelant simplement la méthode correspondant au type de la donnée appelée et en incluant le nom et la valeur de cette donnée, aucun code supplémentaire n'est requis. Dès que votre programme écrit une valeur pour un nom déclaré, cette variable apparaîtra dans le même élément de la station de pilotage ou sur la console de l'ordinateur de développement. Comme vous pouvez l'imaginer c'est un excellent moyen de débogage et d'obtenir l'état de votre robot pendant qu'il est en opération.

Création de widgets sur SmartDashboard

Les widgets sont automatiquement remplis sur le SmartDashboard, aucune intervention de l'utilisateur n'est requise. Notez que les widgets ne sont remplis que lorsque la valeur est écrite pour la première fois, vous devrez peut-être activer votre robot dans un mode particulier ou déclencher une routine de code particulière pour qu'un élément apparaisse. Pour modifier l'apparence du widget, voir les deux sections suivantes *Changer les propriétés d'affichage* et *Changer le type de widget pour une valeur*.

Données obsolètes

SmartDashboard utilise les *NetworkTables* pour communiquer des valeurs entre le robot et l'ordinateur portable de la station de conduite. NetworkTables agit comme une table distribuée de paires de noms et de valeurs. Si une paire nom / valeur est ajoutée au client (ordinateur portable) ou au serveur (robot), elle est répliquée sur l'autre. Si une paire nom/valeur est supprimée, par exemple, du robot mais que le SmartDashboard ou OutlineViewer est toujours en cours d'exécution, alors lorsque le robot est redémarré, les anciennes valeurs apparaîtront toujours dans SmartDashboard et OutlineViewer car elles n'ont jamais cessé de fonctionner et continuent à ces valeurs dans leurs tableaux. Lorsque le robot redémarre, ces anciennes valeurs seront répliquées sur le robot.

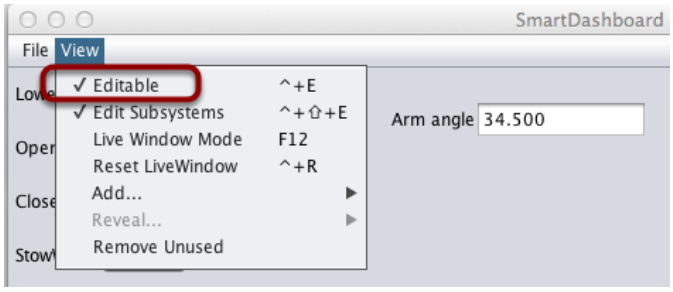
Pour garantir que SmartDashboard et OutlineViewer affichent les valeurs actuelles, il est nécessaire de redémarrer les clients et le robot NetworkTables en même temps. De cette façon, les anciennes valeurs que l'on détient ne seront pas répliquées aux autres.

Ce n'est généralement pas un problème si le programme ne change pas constamment, mais si le programme est phase de développement et l'ensemble de clés ajoutées à NetworkTables changent constamment, il peut alors s'avérer nécessaire de tout redémarrer afin de voir précisément ce qui est à jour.

11.3.3 Modification des propriétés d'affichage d'une valeur

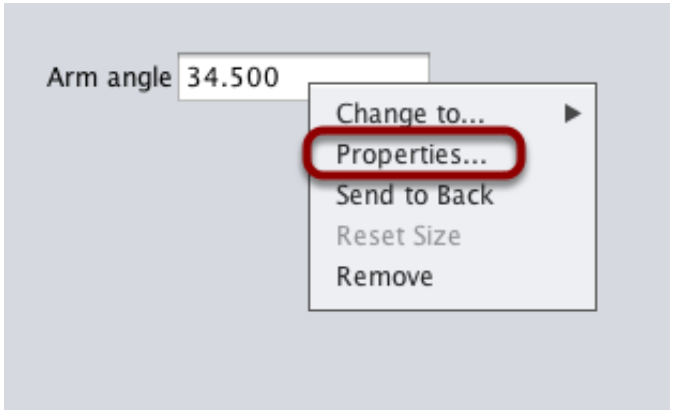
Each value displayed with SmartDashboard has a set of properties that effect the way it's displayed.

Configuration de l'affichage de SmartDashboard en mode édition



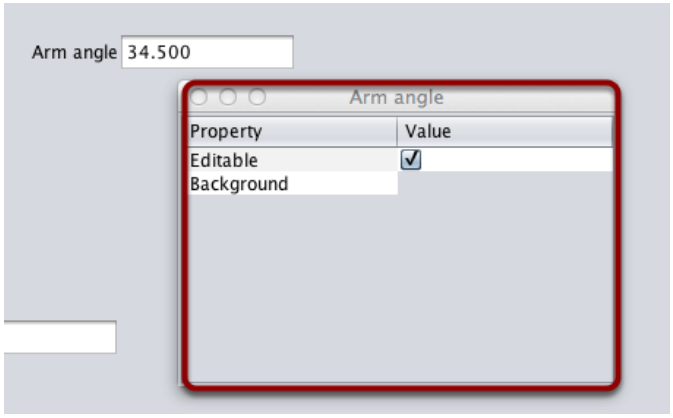
SmartDashboard dispose de deux modes de fonctionnement : affichage et édition. En mode édition, vous pouvez déplacer les widgets à l'écran et modifier leurs propriétés. Pour placer SmartDashboard en mode édition, cliquez sur le menu « View », puis sélectionnez « Editable » pour effectivement activer le mode édition.

Propriétés d'édition d'un widget



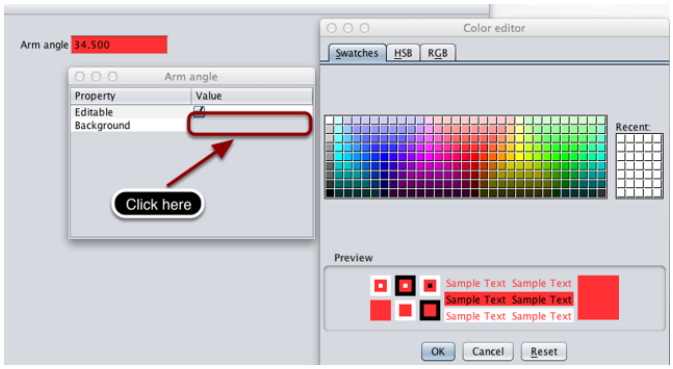
Une fois en mode édition, vous pouvez afficher et modifier les propriétés d'un widget. Cliquez avec le bouton droit sur le widget et sélectionnez « Properties... ».

Modification des propriétés d'un champ



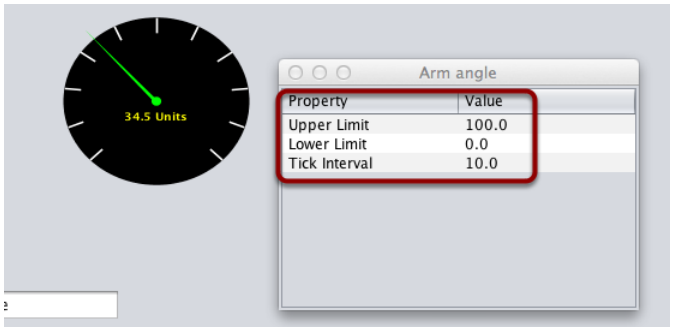
Une boîte de dialogue s'affiche en réponse à l'item menu « Properties… » du menu contextuel des widgets.

Modification de la couleur d'arrière-plan des widgets



Pour modifier la valeur d'une propriété, par exemple, la couleur de l'arrière-plan, cliquez sur la couleur de l'arrière-plan affichée (gris dans ce cas-ci), puis choisissez une couleur dans la palette de couleurs qui apparaît. Ceci sera utilisé comme couleur de l'arrière-plan des widgets.

Modification des propriétés d'autres widgets

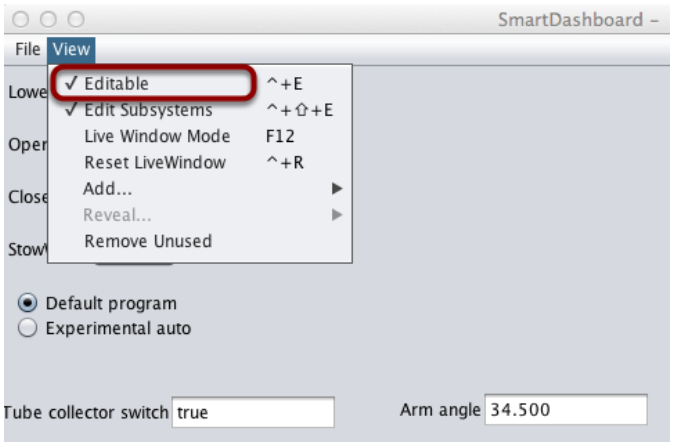


Différents types de widget ont différents ensembles de propriétés modifiables pour en changer l'apparence. Dans cet exemple, les paramètres Upper Limit, Lower Limit et Tick interval sont tous modifiables.

11.3.4 Modification du type de widget d'affichage pour une valeur

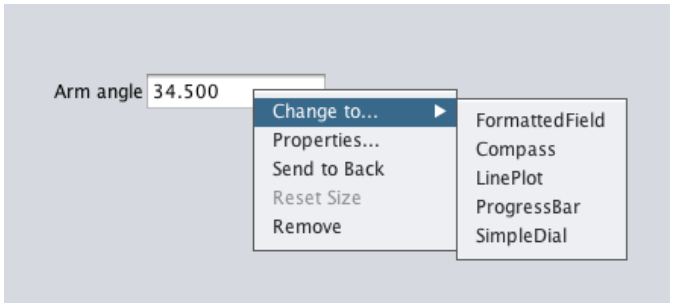
On peut changer le type de widget qui affiche les valeurs avec le SmartDashboard. Les widgets autorisés dépendent du type de valeur affichée.

Réglage du mode d'édition



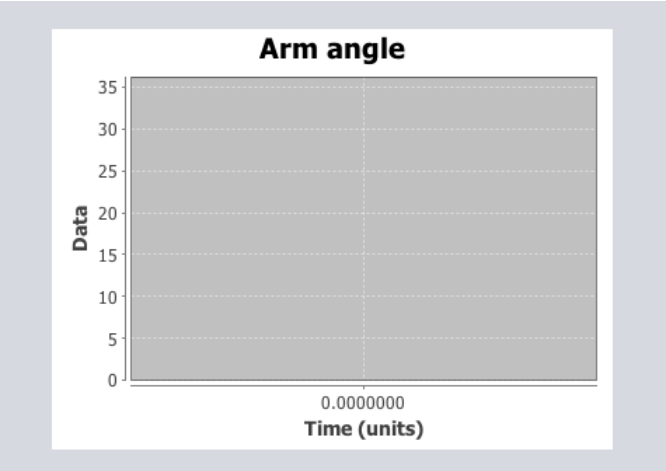
Assurez-vous que le SmartDashboard est en mode édition. Cela se fait en sélectionnant `Editable` dans le `View menu`.

Choix du type de widget



Faites un clic droit sur le widget et sélectionnez `Change to...`. Ensuite, choisissez le type de widget à utiliser pour la valeur particulière. Dans ce cas, nous choisissons `LinePlot`.

Affichage du nouveau type de widget



Le nouveau type de widget s'affiche. Dans ce cas, un tracé linéaire affichera les valeurs de l'angle du bras au fil du temps. Vous pouvez définir les propriétés du graphique pour mieux l'adapter à vos données en cliquant avec le bouton droit de la souris et en sélectionnant `Properties...`. Voir : *Changer les propriétés*.

11.3.5 Choisir un programme autonome

Souvent, les équipes ont plus d'un programme autonome, soit pour des raisons de concurrence, soit pour tester de nouveaux logiciels. Les programmes varient souvent en ajoutant des éléments tels que des délais, des stratégies différentes, etc. Les méthodes pour choisir la stratégie à exécuter impliquent généralement des commutateurs, des boutons de joystick, des boutons ou d'autres entrées matérielles.

With the SmartDashboard you can simply display a widget on the screen to choose the autonomous program that you would like to run. And with command based programs, that program is encapsulated in one of several commands. This article shows how to select an autonomous program with only a few lines of code and a nice looking user interface, with examples for both TimedRobot and Command-Based Robots.

TimedRobot

Note : The code snippets shown below are part of the TimedRobot template (Java, C++) :

Creating SendableChooser Object

In `Robot.java` / `Robot.h`, create a variable to hold a reference to a `SendableChooser` object. Two or more auto modes can be added by creating strings to send to the chooser. Using the `SendableChooser`, one can choose between them. In this example, `Default` and `My Auto` are shown as options. You will also need a variable to store which auto has been chosen, `m_autoSelected`.

Java

```
  private static final String kDefaultAuto = "Default";
  private static final String kCustomAuto = "My Auto";
  private String m_autoSelected;
  private final SendableChooser<String> m_chooser = new SendableChooser<>();
```

C++

```
  frc::SendableChooser<std::string> m_chooser;
  const std::string kAutoNameDefault = "Default";
  const std::string kAutoNameCustom = "My Auto";
  std::string m_autoSelected;
```

Python

```
import wpilib

self.defaultAuto = "Default"
self.customAuto = "My Auto";
self.chooser = wpilib.SendableChooser()
```

Setting Up Options

The chooser allows you to pick from a list of defined elements, in this case the strings we defined above. In `robotInit`, add your options created as strings above using `setDefaultOption` or `addOption`. `setDefaultOption` will be the one selected by default when the dashboard starts. The `putData` function will push it to the dashboard on your driver station computer.

Java

```
  public void robotInit() {
    m_chooser.setDefaultOption("Default Auto", kDefaultAuto);
    m_chooser.addOption("My Auto", kCustomAuto);
    SmartDashboard.putData("Auto choices", m_chooser);
  }
```

C++

```
void Robot::RobotInit() {
  m_chooser.SetDefaultOption(kAutoNameDefault, kAutoNameDefault);
  m_chooser.AddOption(kAutoNameCustom, kAutoNameCustom);
  frc::SmartDashboard::PutData("Auto Modes", &m_chooser);
}
```

Python

```
from wpilib import SmartDashboard

self.chooser.setDefaultOption("Default Auto", self.defaultAuto)
self.chooser.addOption("My Auto", self.customAuto)
SmartDashboard.putData("Auto choices", self.chooser)
```

Running Autonomous Code

Now, in `autonomousInit` and `autonomousPeriodic`, you can use the `m_autoSelected` variable to read which option was chosen, and change what happens during the autonomous period.

Java

```
  @Override
  public void autonomousInit() {
    m_autoSelected = m_chooser.getSelected();
    System.out.println("Auto selected: " + m_autoSelected);
  }

  /** This function is called periodically during autonomous. */
  @Override
  public void autonomousPeriodic() {
    switch (m_autoSelected) {
      case kCustomAuto:
        // Put custom auto code here
        break;
      case kDefaultAuto:
      default:
        // Put default auto code here
        break;
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }
  }
```

C++

```
void Robot::AutonomousInit() {
  m_autoSelected = m_chooser.GetSelected();
  fmt::print("Auto selected: {}\n", m_autoSelected);

  if (m_autoSelected == kAutoNameCustom) {
    // Custom Auto goes here
  } else {
    // Default Auto goes here
  }
}

void Robot::AutonomousPeriodic() {
  if (m_autoSelected == kAutoNameCustom) {
    // Custom Auto goes here
  } else {
    // Default Auto goes here
  }
}
```

Python

```
def autonomousInit(self):
   self.autoSelected = self.chooser.getSelected()
   print("Auto selected: " + self.autoSelected)

def autonomousPeriodic(self):
   match self.autoSelected:
      case self.customAuto:
         # Put custom auto code here
      case _:
         # Put default auto code here
```

Command-Based

Note : The code snippets shown below are part of the HatchbotTraditional example project (Java, C++, Python) :

Création de l'objet SendableChooser

Dans `RobotContainer`, créez une variable pour contenir une référence à un objet `SendableChooser`. Deux commandes ou plus peuvent être créées et stockées dans de nouvelles variables. En utilisant le `SendableChooser`, on peut choisir l'une ou l'autres des deux variables. Dans cet exemple, `SimpleAuto` et `ComplexAuto` sont affichés comme des options.

Java

```
  // A simple auto routine that drives forward a specified distance, and then stops.
  private final Command m_simpleAuto =
      new DriveDistance(
          AutoConstants.kAutoDriveDistanceInches, AutoConstants.kAutoDriveSpeed, m_
↪robotDrive);

  // A complex auto routine that drives forward, drops a hatch, and then drives␣
↪backward.
  private final Command m_complexAuto = new ComplexAuto(m_robotDrive, m_
↪hatchSubsystem);

  // A chooser for autonomous commands
  SendableChooser<Command> m_chooser = new SendableChooser<>();
```

C++ (using raw pointers)

```
  // The autonomous routines
  DriveDistance m_simpleAuto{AutoConstants::kAutoDriveDistanceInches,
                             AutoConstants::kAutoDriveSpeed, &m_drive};
  ComplexAuto m_complexAuto{&m_drive, &m_hatch};

  // The chooser for the autonomous routines
  frc::SendableChooser<frc2::Command*> m_chooser;
```

C++ (using `CommandPtr`)

```
  // The autonomous routines
  frc2::CommandPtr m_simpleAuto = autos::SimpleAuto(&m_drive);
  frc2::CommandPtr m_complexAuto = autos::ComplexAuto(&m_drive, &m_hatch);

  // The chooser for the autonomous routines
  frc::SendableChooser<frc2::Command*> m_chooser;
```

Python

```
        # A simple auto routine that drives forward a specified distance, and then␣
↪stops.
        self.simpleAuto = DriveDistance(
            constants.kAutoDriveDistanceInches, constants.kAutoDriveSpeed, self.drive
        )

        # A complex auto routine that drives forward, drops a hatch, and then drives␣
↪backward.
        self.complexAuto = ComplexAuto(self.drive, self.hatch)

        # Chooser
        self.chooser = wpilib.SendableChooser()
```

Configurer SendableChooser

Imaginez que vous ayez le choix entre deux programmes autonomes et qu'ils soient encapsulés dans les commandes `SimpleAuto` et `ComplexAuto`. Pour choisir entre eux :

Dans `RobotContainer`, créez un objet `SendableChooser` et ajoutez-y des instances des deux commandes. Il peut y avoir n'importe quel nombre de commandes, et celle ajoutée par défaut (`setDefaultOption`), devient celle qui est initialement sélectionnée. Notez que chaque commande est incluse dans un appel de méthode `setDefaultOption()` ou `addOption()` sur l'instance `SendableChooser`.

Java

```
    // Add commands to the autonomous command chooser
    m_chooser.setDefaultOption("Simple Auto", m_simpleAuto);
    m_chooser.addOption("Complex Auto", m_complexAuto);
```

C++ (using raw pointers)

```
  // Add commands to the autonomous command chooser
  m_chooser.SetDefaultOption("Simple Auto", &m_simpleAuto);
  m_chooser.AddOption("Complex Auto", &m_complexAuto);
```

C++ (using `CommandPtr`)

```
  // Add commands to the autonomous command chooser
  // Note that we do *not* move ownership into the chooser
  m_chooser.SetDefaultOption("Simple Auto", m_simpleAuto.get());
  m_chooser.AddOption("Complex Auto", m_complexAuto.get());
```

Python

```
        # Add commands to the autonomous command chooser
        self.chooser.setDefaultOption("Simple Auto", self.simpleAuto)
        self.chooser.addOption("Complex Auto", self.complexAuto)
```

Then, publish the chooser to the dashboard :

Java

```
// Put the chooser on the dashboard
SmartDashboard.putData(m_chooser);
```

C++

```
// Put the chooser on the dashboard
frc::SmartDashboard::PutData(&m_chooser);
```

Python

```
from wpilib import SmartDashboard

# Put the chooser on the dashboard
SmartDashboard.putData(chooser)
```

Démarrer une commande autonome

Dans `Robot.java`, lorsque la période autonome démarre, l'objet `SendableChooser` est interrogé pour obtenir la commande sélectionnée et cette commande doit être planifiée.

Java

```
  public Command getAutonomousCommand() {
    return m_chooser.getSelected();
  }
```

```
  public void autonomousInit() {
    m_autonomousCommand = m_robotContainer.getAutonomousCommand();

    // schedule the autonomous command (example)
    if (m_autonomousCommand != null) {
      m_autonomousCommand.schedule();
    }
  }
```

C++ (Source)

```
frc2::Command* RobotContainer::GetAutonomousCommand() {
  // Runs the chosen command in autonomous
  return m_chooser.GetSelected();
}
```

```
void Robot::AutonomousInit() {
  m_autonomousCommand = m_container.GetAutonomousCommand();

  if (m_autonomousCommand != nullptr) {
    m_autonomousCommand->Schedule();
  }
}
```

Python

```
    def getAutonomousCommand(self) -> commands2.Command:
        return self.chooser.getSelected()
```

```
    def autonomousInit(self) -> None:
        """This autonomous runs the autonomous command selected by your␣
↪RobotContainer class."""
        self.autonomousCommand = self.container.getAutonomousCommand()

        if self.autonomousCommand:
            self.autonomousCommand.schedule()
```

Exécution du planificateur en mode autonome

In `Robot.java`, this will run the scheduler every driver station update period (about every 20ms) and cause the selected autonomous command to run. In Python the scheduler runs automatically when `TimedCommandRobot` is used.

Note : L'exécution du planificateur peut se produire dans la fonction `autonomousPeriodic()` ou `robotPeriodic()`, les deux fonctionneront de la même manière en mode autonome.

Java

```
40  @Override
41  public void robotPeriodic() {
42    CommandScheduler.getInstance().run();
43  }
```

C++ (Source)

```
void Robot::RobotPeriodic() {
  frc2::CommandScheduler::GetInstance().Run();
}
```

Annulation de la commande autonome

Dans `Robot.java`, lorsque la période téléop commence, la commande autonome sera annulée.

Java

```
  @Override
  public void teleopInit() {
    // This makes sure that the autonomous stops running when
    // teleop starts running. If you want the autonomous to
    // continue until interrupted by another command, remove
    // this line or comment it out.
    if (m_autonomousCommand != null) {
      m_autonomousCommand.cancel();
    }
  }
```

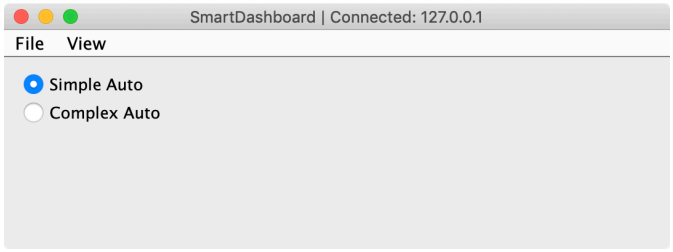
C++ (Source)

```
void Robot::TeleopInit() {
  // This makes sure that the autonomous stops running when
  // teleop starts running. If you want the autonomous to
  // continue until interrupted by another command, remove
  // this line or comment it out.
  if (m_autonomousCommand != nullptr) {
    m_autonomousCommand->Cancel();
    m_autonomousCommand = nullptr;
  }
}
```

Python

```
    def teleopInit(self) -> None:
        # This makes sure that the autonomous stops running when
        # teleop starts running. If you want the autonomous to
        # continue until interrupted by another command, remove
        # this line or comment it out.
        if self.autonomousCommand:
            self.autonomousCommand.cancel()
```

Affichage du tableau de bord intelligent

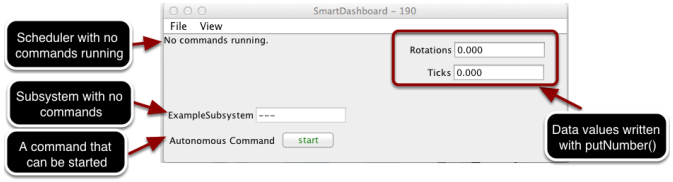


Lorsque le SmartDashboard est exécuté, les choix du `SendableChooser` sont automatiquement affichés. Vous pouvez simplement choisir une option avant le début de la période autonome et la commande correspondante s'exécutera.

11.3.6 Affichage de l'état des commandes et des sous-systèmes

Si vous utilisez les fonctionnalités de programmation basées sur les commandes de WPILib, vous constaterez qu'elles sont très bien intégrées à SmartDashboard. Celles-ci peuvent aider à diagnostiquer ce que fait le robot à tout moment et vous donner le contrôle et une vue de ce qui est en cours d'exécution.

Présentation des affichages de commande et de sous-système



Avec SmartDashboard, vous pouvez afficher l'état des commandes et des sous-systèmes dans votre programme de robot de différentes manières. Ces informations devraient réduire considérablement le temps de débogage de vos programmes. Sur cette image, vous pouvez voir un certain nombre d'affichages possibles. Voici les éléments suivants :

- Le planificateur dans l'état ou aucune commande en cours d'exécution, en anglais : `No commands running`. Dans l'exemple suivant, vous pouvez voir à quoi cela ressemble avec quelques commandes en cours d'exécution indiquant l'état du robot.
- Un sous-système `ExampleSubsystem` qui indique qu'il n'y a actuellement aucune commande en cours d'exécution qui « nécessite » ce sous-système. Lorsque les commandes sont en cours d'exécution, l'écran indiquera le nom des commandes qui utilisent le sous-système.
- Une commande écrite sur SmartDashboard qui montre un bouton `start` sur lequel vous pouvez appuyer pour exécuter la commande. C'est un excellent moyen de tester vos commandes une par une.
- Et quelques valeurs de données écrites dans le tableau de bord pour aider à déboguer le code en cours d'exécution.

Dans les exemples suivants, vous verrez à quoi ressemblerait l'écran lorsque des commandes sont en cours d'exécution et le code qui produit cet affichage.

Affichage de l'état du planificateur

JAVA

```
SmartDashboard.putData(CommandScheduler.getInstance());
```

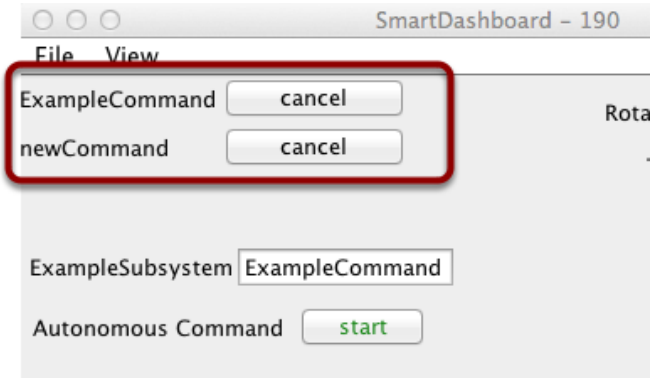
C++

```
frc::SmartDashboard::PutData(frc2::CommandScheduler::GetInstance());
```

PYTHON

```
from wpilib import SmartDashboard
from commands2 import CommandScheduler

SmartDashboard.putData(CommandScheduler.getInstance())
```

Vous pouvez afficher l'état du planificateur (le code qui planifie l'exécution de vos commandes). Cela se fait facilement en ajoutant une seule ligne à la méthode `RobotInit` dans votre programme comme indiqué ici. Dans cet exemple, l'instance Scheduler est écrite à l'aide de la méthode `putData` dans SmartDashboard. Cette ligne de code produit l'affichage dans l'image précédente.



Il s'agit de l'état du planificateur lorsque deux commandes sont en cours d'exécution, `ExampleCommand` et `newCommand`. Cela remplace le message `No commands running.` de l'image d'écran précédente. Vous pouvez voir les commandes affichées sur le tableau de bord lorsque le programme s'exécute et que diverses commandes sont déclenchées.

Affichage de l'état du sous-système

JAVA

```
SmartDashboard.putData(exampleSubsystem);
```

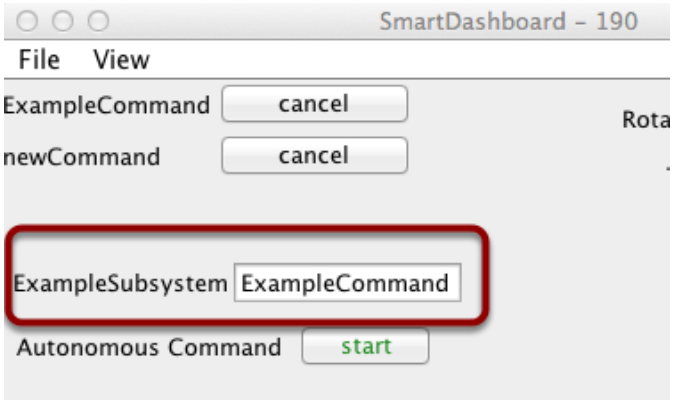
C++

```
frc::SmartDashboard::PutData(&exampleSubsystem);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData(exampleSubsystem)
```

Dans cet exemple, nous écrivons l'instance de commande, `exampleSubsystem` et l'instance de la classe `ExampleSubsystem` dans le SmartDashboard. Cela provoque l'affichage montré dans l'image précédente. Le champ de texte contiendra soit quelques tirets, `---` indiquant qu'aucune commande en cours utilise ce sous-système, ou encore le nom de la commande qui utilise actuellement ce sous-système.



Les commandes en cours « nécessitent » des sous-systèmes. C'est la commande qui réserve le sous-système à son usage exclusif. Si vous affichez un sous-système sur SmartDashboard, il affichera la commande qui l'utilise actuellement. Dans cet exemple, `ExampleSubsystem` est utilisé par `ExampleCommand`.

Activation des commandes avec un bouton

JAVA

```
SmartDashboard.putData("Autonomous Command", exampleCommand);
```

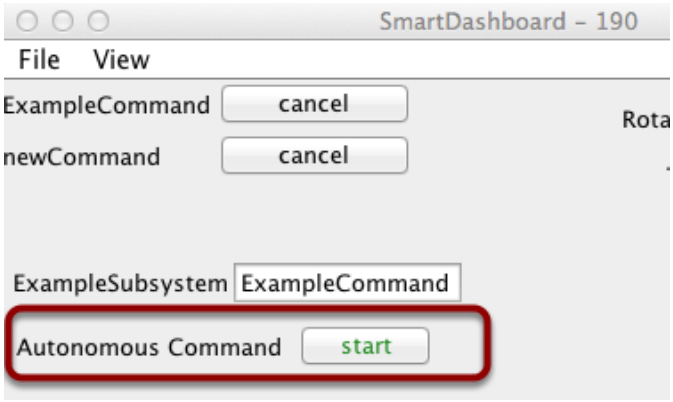
C++

```
frc::SmartDashboard::PutData("Autonomous Command", &exampleCommand);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData("Autonomous Command", exampleCommand)
```

This is the code required to create a button for the command on SmartDashboard. Pressing the button will schedule the command. While the command is running, the button label changes from `start` to `cancel` and pressing the button will cancel the command.

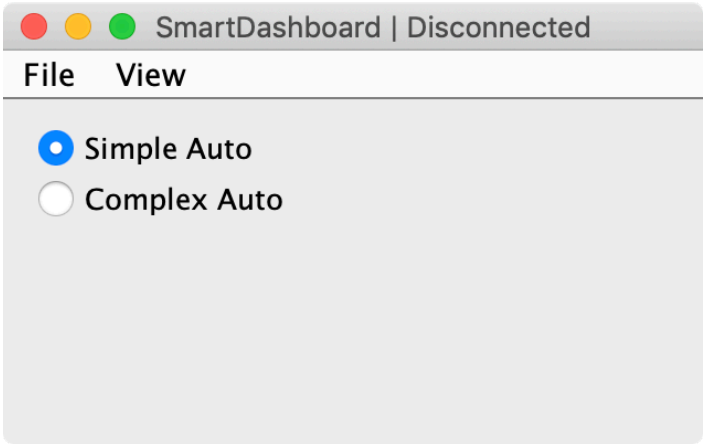


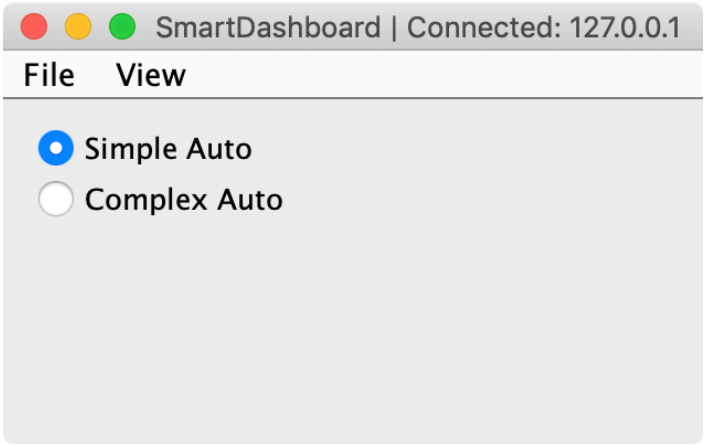
Dans cet exemple, vous pouvez voir un bouton intitulé `Autonomous Command`. Appuyez sur ce bouton pour exécuter la commande associée. Ceci est un excellent moyen de tester les commandes une par une sans avoir à ajouter du code de test supplémentaire à votre programme de robot. L'ajout de boutons pour chaque commande facilite le test du programme, une commande à la fois.

11.3.7 Vérifier que SmartDashboard fonctionne

Indicateur de connexion

SmartDashboard inclura automatiquement l'état de la connexion et l'adresse IP de la source NetworkTables dans le titre de la fenêtre.





Widget indicateur de connexion

SmartDashboard comprend un widget indicateur de connexion qui deviendra rouge ou vert en fonction de la connexion à NetworkTables, généralement fournie par le roboRIO. Pour obtenir des instructions sur l'ajout de ce widget, consultez *Ajout d'un indicateur de connexion* dans SmartDashboard Intro.

Exemple de programme de robot

JAVA

```
public class Robot extends TimedRobot {
   double counter = 0.0;

   public void teleopPeriodic() {
      SmartDashboard.putNumber("Counter", counter++);
   }
}
```

C++

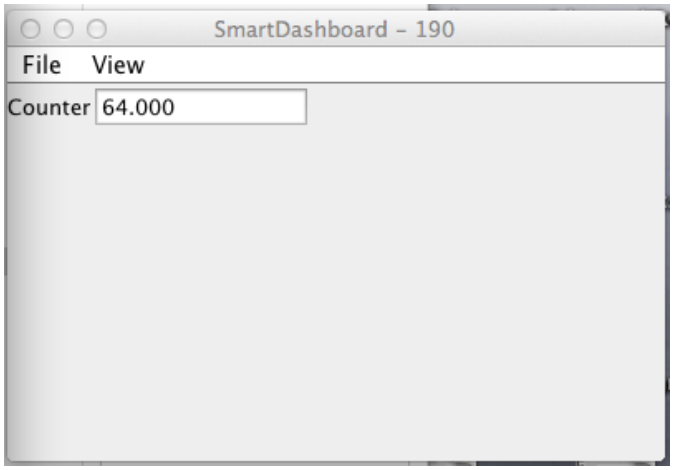
```
#include "Robot.h"
float counter = 0.0;

void Robot::TeleopPeriodic() {
   frc::SmartDashboard::PutNumber("Counter", counter++);
}
```

PYTHON

```
from wpilib import SmartDashboard

self.counter = 0.0

def teleopPeriodic(self) -> None:
   SmartDashboard.putNumber("Counter", self.counter += 1)
```

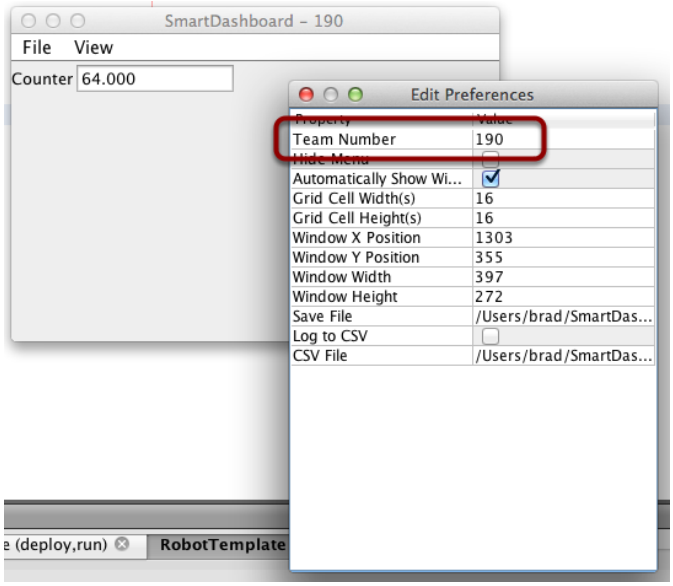
Il s'agit d'un programme de robot minimal qui écrit une valeur sur le SmartDashboard. Il incrémente simplement un compteur 50 fois par seconde pour vérifier que la connexion fonctionne. Cependant, pour minimiser l'utilisation de la bande passante, NetworkTables par défaut limitera les mises à jour à 10 fois par seconde.

Sortie SmartDashboard pour l'exemple de programme



L'affichage du SmartDashboard devrait ressembler à ceci après environ 6 secondes d'activation du robot en mode Teleop. Si ce n'est pas le cas, vous devez vérifier que la connexion est correctement configurée.

Vérification de l'adresse IP dans SmartDashboard

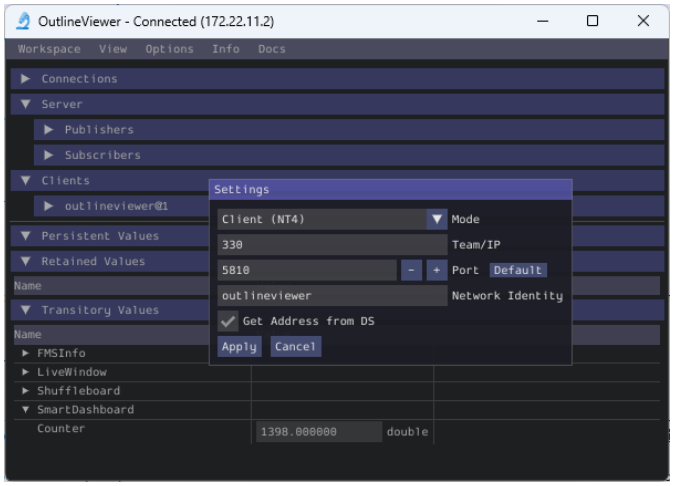


Si l'affichage de la valeur n'apparaît pas, vérifiez que le numéro d'équipe est correctement défini comme indiqué sur cette image. La boîte de dialogue des préférences peut être visualisée en sélectionnant `File`, puis `Preferences`.

Vérification du programme à l'aide d'OutlineViewer

You can verify that the robot program is generating SmartDashboard values by using the *OutlineViewer program*.

Expand the SmartDashboard row. The value `Counter` is the variable written to the SmartDashboard via NetworkTables. As the program runs you should see the value increasing (`1398.0` in this case). If you don't see this variable in the OutlineViewer, look for something wrong with the robot program or the network configuration.



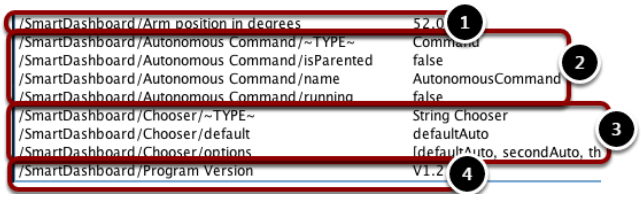
11.3.8 Espace pour noms dans SmartDashboard

SmartDashboard utilise NetworkTables pour envoyer des données entre le robot et l'ordinateur du tableau de bord (Driver Station). NetworkTables envoie des données sous forme de nom, de paires de valeurs, comme une table de hachage distribuée entre le robot et l'ordinateur. Lorsqu'une valeur est modifiée à un endroit, sa valeur est automatiquement mise à jour à l'autre endroit. Ce mécanisme et un ensemble standard de noms (clés) permettent d'afficher les données sur le SmartDashboard.

Il existe une structure hiérarchique dans l'espace pour noms créant un ensemble de tables et de sous-tables. Les données SmartDashboard se trouvent dans la sous-table SmartDashboard et les données LiveWindow sont dans la sous-table LiveWindow comme indiqué ci-dessous.

À des fins d'information, les noms et les valeurs peuvent être affichés à l'aide de l'application OutlineViewer installée au même emplacement que le SmartDashboard. Il affichera toutes les clés et valeurs NetworkTables au fur et à mesure de leur mise à jour.

Valeurs des données SmartDashboard

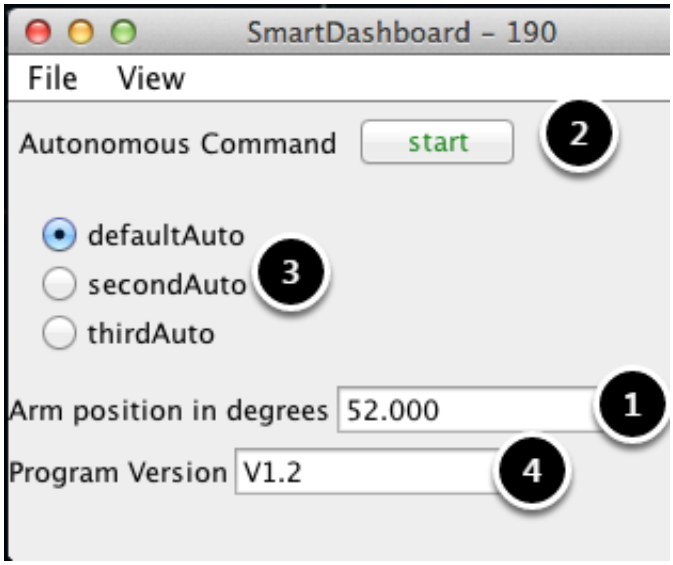


Les valeurs de SmartDashboard sont créées avec des noms de clé commençant par `SmartDashboard/`. Les valeurs ci-dessus affichées avec OutlineViewer correspondent aux données placées dans le SmartDashboard avec les instructions suivantes :

```
chooser = new SendableChooser();
chooser.setDefaultOption("defaultAuto", new AutonomousCommand());
chooser.addOption("secondAuto", new AutonomousCommand());
chooser.addOption("thirdAuto", new AutonomousCommand());
SmartDashboard.putData("Chooser", chooser);
SmartDashboard.putNumber("Arm position in degrees", 52.0);
SmartDashboard.putString("Program Version", "V1.2");
```

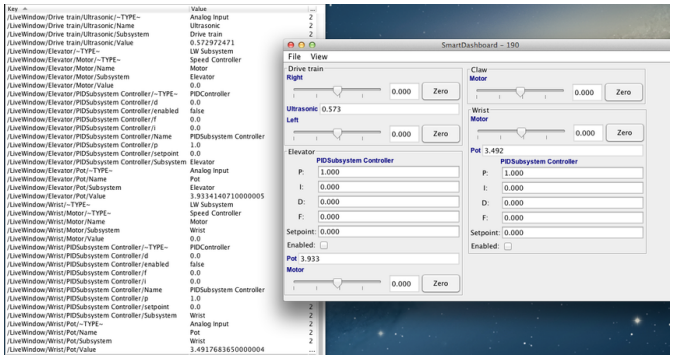
La `Arm position` est créee avec l'appel `putNumber()`. Le `AutonomousCommand` est écrit avec un `putData("Autonomous Command",command)` qui n'est pas montré dans le fragment de code ci-dessus. Le sélecteur est créé en tant qu'objet `SendableChooser` et la valeur de chaîne de caractères, `Program Version` est créée avec l'appel `putString()`.

Vue du SmartDashboard



Le code de l'étape précédente génère les valeurs de la table comme indiqué et l'affichage du SmartDashboard comme illustré ici. Les nombres correspondent aux variables NetworkTables affichées à l'étape précédente.

Valeurs des données LiveWindow



Les données LiveWindow sont automatiquement regroupées par sous-système. Les données peuvent être visualisées dans le SmartDashboard lorsque le robot est en mode Test (défini sur le Driver Station). Si vous n'écrivez pas un programme basé sur des commandes, vous pouvez toujours grouper les capteurs et les actionneurs pour une visualisation facile en spécifiant le nom du sous-système. Dans l'affichage ci-dessus, vous pouvez voir les noms des touches et la sortie résultante en mode Test sur le SmartDashboard. Toutes les chaînes de caractères commencent par `/LiveWindow` puis le nom du sous-système, puis un groupe de valeurs qui permettent d'afficher chaque élément. Le code qui génère cet affichage LiveWindow est indiqué ci-dessous :

```
drivetrainLeft = new PWMVictorSPX(1);
drivetrainLeft.setName("Drive train", "Left");

drivetrainRight = new PWMVictorSPX(1);
```

(suite sur la page suivante)

(suite de la page précédente)

```
drivetrainRight.setName("Drive train", "Right";

drivetrainRobotDrive = new DifferentialDrive(drivetrainLeft, drivetrainRight);
drivetrainRobotDrive.setSafetyEnabled(false);
drivetrainRobotDrive.setExpiration(0.1);

drivetrainUltrasonic = new AnalogInput(3);
drivetrainUltrasonic.setName("Drive train", "Ultrasonic");

elevatorMotor = new PWMVictorSPX(6);
elevatorMotor.setName("Elevator", "Motor");

elevatorPot = new AnalogInput(4);
elevatorPot.setName("Elevator", "Pot");

wristPot = new AnalogInput(2);
wristPot.setName("Wrist", "Pot");

wristMotor = new PWMVictorSPX(3);
wristMotor.setName("Wrist", "Motor");

clawMotor = new PWMVictorSPX(5);
clawMotor.setName("Claw", "Motor");
```

Les valeurs qui correspondent aux actionneurs ne sont pas seulement affichées, mais peuvent être définies à l'aide des curseurs créés dans le SmartDashboard en mode Test.

11.3.9 SmartDashboard : Test Mode et LiveWindow

Affichage de valeurs dans LiveWindow

LiveWindow ajoutera automatiquement vos capteurs et actionneurs pour vous. Il n'est pas nécessaire de le faire manuellement. Les valeurs LiveWindow peuvent également être affichées en écrivant le code vous-même et en l'ajoutant à votre programme de robot. Cela vous permet de personnaliser les noms et de les regrouper en sous-systèmes. Il s'agit d'une méthode pratique pour afficher s'il s'agit de sous-systèmes de programmes basés sur des commandes ou simplement d'un groupement que vous décidez d'utiliser dans votre programme.

Ajout du code nécessaire à votre programme

Pour chaque capteur ou actionneur créé, définissez le nom du sous-système et le nom d'affichage en appelant `setName` (`SetName` en C++). Lorsque le SmartDashboard est mis en mode LiveWindow, il affichera les capteurs et les actionneurs.

JAVA

```
Ultrasonic ultrasonic = new Ultrasonic(1, 2);
SendableRegistry.setName(ultrasonic, "Arm", "Ultrasonic");

Jaguar elbow = new Jaguar(1);
SendableRegistry.setName(elbow, "Arm", "Elbow");

Victor wrist = new Victor(2);
SendableRegistry.setName(wrist, "Arm", "Wrist");
```

C++

```
frc::Ultrasonic ultrasonic{1, 2};
SendableRegistry::SetName(ultrasonic, "Arm", "Ultrasonic");

frc::Jaguar elbow{1};
SendableRegistry::SetName(elbow, "Arm", "Elbow");

frc::Victor wrist{2};
SendableRegistry::SetName(wrist, "Arm", "Wrist");
```

PYTHON

```
from wpilib import Jaguar, Ultrasonic, Victor
from wpiutil import SendableRegistry

ultrasonic = Ultrasonic(1, 2)
SendableRegistry.setName(ultrasonic, "Arm", "Ultrasonic")

elbow = Jaguar(1)
SendableRegistry.setName(elbow, "Arm", "Elbow")

wrist = Victor(2)
SendableRegistry.setName(wrist, "Arm", "Wrist")
```

If your objects are in a `Subsystem`, this can be simplified using the addChild method of `SubsystemBase`

JAVA

```
Ultrasonic ultrasonic = new Ultrasonic(1, 2);
addChild("Ultrasonic", ultrasonic);

Jaguar elbow = new Jaguar(1);
addChild("Elbow", elbow);

Victor wrist = new Victor(2);
addChild("Wrist", wrist);
```

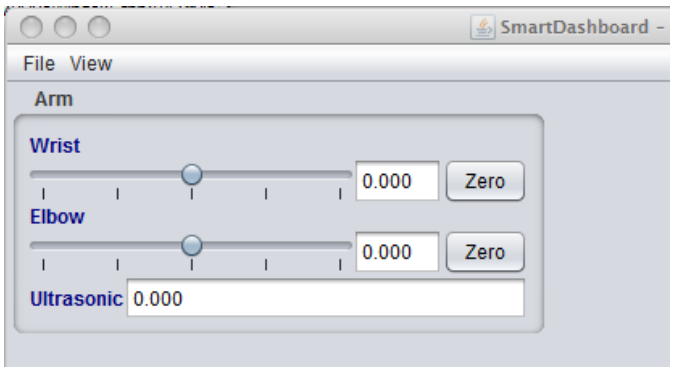
C++

```
frc::Ultrasonic ultrasonic{1, 2};
AddChild("Ultrasonic", ultrasonic);

frc::Jaguar elbow{1};
AddChild("Elbow", elbow);

frc::Victor wrist{2};
AddChild("Wrist", wrist);
```

PYTHON

```
from wpilib import Jaguar, Ultrasonic, Victor
from commands2 import SubsystemBase

ultrasonic = Ultrasonic(1, 2)
SubsystemBase.addChild("Ultrasonic", ultrasonic)

elbow = Jaguar(1)
SubsystemBase.addChild("Elbow", elbow)

wrist = Victor(2)
SubsystemBase.addChild("Wrist", wrist)
```

Affichage dans SmartDashboard



Les capteurs et actionneurs ajoutés au mode LiveWindow seront affichés regroupés par sous-système. Le nom du sous-système n'est qu'un regroupement arbitraire qui aide à organiser l'affichage des capteurs. Les actionneurs peuvent être activés à l'aide des curseurs correspondant aux deux contrôleurs de moteur.

Activation du mode Test (LiveWindow)

You may add code to your program to display values for your sensors and actuators while the robot is in Test mode. This can be selected from the Driver Station whenever the robot is not on the field (see *Activation du Mode Test* for details on how to do this). Test mode is designed to verify the correct operation of the sensors and actuators on a robot. In addition it can be used for obtaining setpoints from sensors such as potentiometers and for tuning PID loops in your code. When the robot is enabled in Test mode, the SmartDashboard display will switch to test mode (LiveWindow) and will display the status of any actuators and sensors used by your program.

Important : Since 2024, LiveWindow is not enabled by default in Test mode !

Enabling LiveWindow in Test Mode

To run LiveWindow in Test Mode, the following code is needed in the `Robot` class :

JAVA

```java
@Override
public void robotInit() {
   enableLiveWindowInTest(true);
}
```

C++

```cpp
void Robot::RobotInit() {
   EnableLiveWindowInTest(true);
}
```

PYTHON

```python
def robotInit(self) -> None:
   enableLiveWindowInTest(true)
```

Affichage en mode test explicite vs implicite

JAVA

```
PWMSparkMax leftDrive;
PWMSparkMax rightDrive;
PWMVictorSPX arm;
BuiltInAccelerometer accel;

@Override
public void robotInit() {
   leftDrive = new PWMSparkMax(0);
   rightDrive = new PWMSparkMax(1);
   arm = new PWMVictorSPX(2);
   accel = new BuiltInAccelerometer();
   SendableRegistry.setName(arm, "SomeSubsystem", "Arm");
   SendableRegistry.setName(accel, "SomeSubsystem", "Accelerometer");
}
```

C++

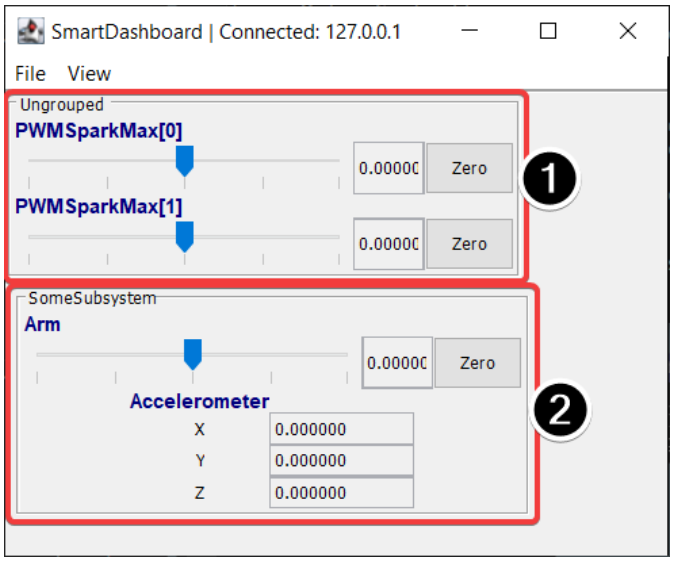
```
frc::PWMSparkMax leftDrive{0};
frc::PWMSparkMax rigthDrive{1};
frc::BuiltInAccelerometer accel{};
frc::PWMVictorSPX arm{3};

void Robot::RobotInit() {
   wpi::SendableRegistry::SetName(&arm, "SomeSubsystem", "Arm");
   wpi::SendableRegistry::SetName(&accel, "SomeSubsystem", "Accelerometer");
}
```

PYTHON

```
from wpilib import BuiltInAccelerometer, PWMSparkMax, PWMVictorSPX
from wpiutil import SendableRegistry

def robotInit(self) -> None:
   leftDrive = PWMSparkMax(0)
   rightDrive = PWMSparkMax(1)
   arm = PWMVictorSPX(2)
   accel = BuiltInAccelerometer()
   SendableRegistry.setName(arm, "SomeSubsystem", "Arm")
   SendableRegistry.setName(accel, "SomeSubsystem", "Accelerometer")
```

Tous les capteurs et actionneurs seront automatiquement affichés sur le SmartDashboard en mode test et seront nommés en utilisant le type d'objet (tel que PWMSparkMax, PWMVictorSPX, BuiltInAccelerometer, etc.) avec le numéro de canal avec lequel l'objet a été créé. De plus, le programme peut explicitement ajouter des capteurs et des actionneurs à l'affichage du mode test, auquel cas des noms de sous-système et d'objet définis par le programmeur peuvent être spécifiés, ce qui rend le programme plus clair. Cet exemple illustre la définition explicite de ces capteurs et actionneurs.

Comprendre ce qui s'affiche en mode Test



Il s'agit de la sortie de l'affichage SmartDashboard lorsque le robot est placé en mode test. Dans l'affichage ci-dessus, les objets répertoriés comme non groupés ont été implicitement créés par WPILib lors de la création des objets correspondants. Ces objets sont contenus dans un groupe de sous-système appelé « Ungrouped » **(1)** et sont nommés avec le type de périphérique (PWMSparkMax dans ce cas) et les numéros de canal. Les objets affichés dans le groupe « SomeSubsystem » **(2)** sont explicitement créés par le programmeur à partir de l'exemple de code de la section précédente. Ceux-ci sont nommés dans les appels à `SendableRegistry.setName()`. Les capteurs et actionneurs créés explicitement seront regroupés par le sous-système spécifié.

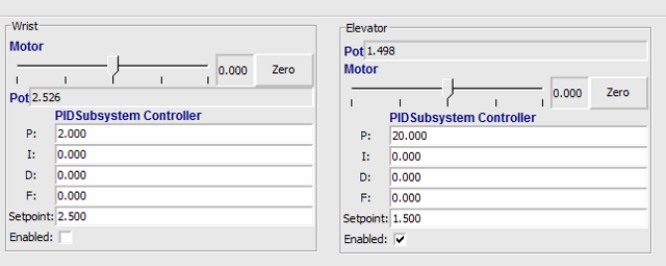
Réglage des paramètres PID avec SmartDashboard

Le PID (Proportionnel, Intégral, Dérivé) est un algorithme permettant de déterminer, par exemple, la vitesse d'un moteur en se servant d'une boucle de rétroaction à partir du signal d'un capteur pour atteindre une valeur désirée, la cible, le plus rapidement possible. Ainsi, un robot avec un mécanisme élévateur qui se déplace vers une position prédéterminée devrait s'y rendre aussi vite que possible, puis s'arrêter sans dépassement excessif conduisant à une oscillation. C'est par réglage du PID de l'élévateur qu'on peut arriver à ce résultat. L'idée derrière le PID est de calculer, pour un paramètre donné, la position d'un bras par exemple, une valeur d'écart qui est la différence entre la valeur actuelle de l'élément de rétroaction du mécanisme et la valeur souhaitée (cible). Dans le cas du bras, l'élément de rétroaction peut être un potentiomètre relié à un canal analogique qui fournit une tension qui est proportionnelle à la position du bras. La valeur souhaitée est la tension prédéterminée du potentiomètre pour que le bras se trouve à la position voulue, et la valeur actuelle est la tension correspondant à la position réelle du bras au moment précis où la mesure est prise.

Détermination des valeurs cibles avec LiveWindow

Créez un sous-système PID pour chaque mécanisme pourvu d'une boucle de rétroaction. Les sous-systèmes PID contiennent un actionneur (moteur) et un capteur de rétroaction (potentiomètre dans le cas présent). Vous pouvez utiliser le mode Test pour afficher les valeurs des capteurs et actionneurs du sous-système. À l'aide du curseur, contrôlez manuellement l'actionneur à chaque position souhaitée. Notez les valeurs du capteur (2) pour chacune des positions prédéterminées. Celles-ci serviront de cibles pour le contrôleur PID.

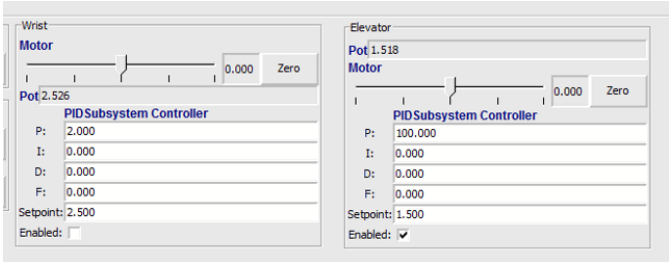
Affichage du PIDController dans LiveWindow



En mode Test, les sous-systèmes PID affichent leurs paramètres P, I et D qui ont été définies dans le code. Les valeurs P, I et D sont les poids appliqués à l'écart calculé (P), la somme des écarts au fil du temps (I) et le taux de changement des écarts (D). Chacun de ces termes est multiplié par le poids correspondant et, additionnés ensemble, ils forment la valeur envoyée au moteur. Choisir les valeurs optimales de P, I et D peut être difficile et nécessite beaucoup d'essais et erreurs. En opérant le robot en mode Test, on peut modifier ces valeurs et observer la réponse du mécanisme.

Important : The enable option does not affect the PIDController introduced in 2020, as the controller is updated every robot loop. See the example *here* on how to retain this functionality.

Réglage du Contrôleur PID



Tuning the PID controller can be difficult and there are many articles that describe techniques that can be used. It is best to start with the P value first. To try different values fill in a low number for P, enter a setpoint determined earlier in this document, and note how fast the mechanism responds. If it responds too slowly, perhaps never reaching the setpoint, increase P. If it responds too quickly, perhaps oscillating, reduce the P value. Repeat this process until you get a response that is as fast as possible without oscillation. It's possible that having a P term is all that's needed to achieve adequate control of your mechanism. Further information is located in the *Tuning a Flywheel Velocity Controller* document.

Une fois que vous avez déterminé les valeurs de P, I et D, elles peuvent alors être insérées dans votre programme. Plus précisément soit dans les propriétés du PIDSubsystem dans RobotBuilder, soit dans le constructeur du sous-système PID dans votre programme du robot.

Le terme F (feedforward) est utilisé pour le contrôle de la vitesse avec un contrôleur PID.

Plus d'informations peuvent être trouvées à *Contrôle PID dans WPILib*.

11.4 Glass

Glass est un nouveau tableau de bord et un nouvel outil de visualisation de données du robot. Son interface graphique est extrêmement similaire à celle de *Interface utilisateur graphique de Simulation*. Dans son état actuel, il est destiné à être utilisé comme un outil pour un programmeur plutôt que comme un tableau de bord approprié dans un environnement de compétition.

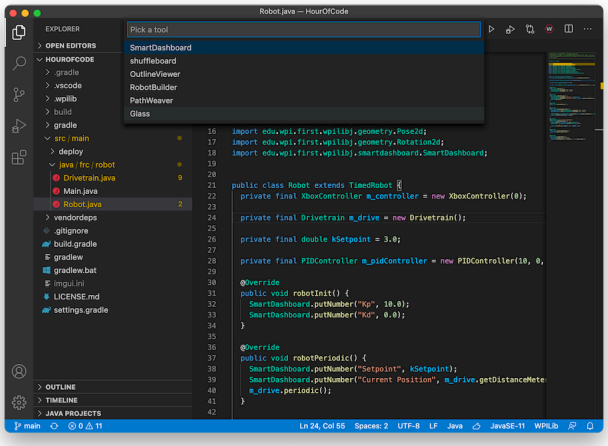
Note : Glass ne sera pas disponible dans la liste des dashboards de l'application Driver Station NI.

11.4.1 Introduction à Glass

Glass est un nouvel outil de visualisation de données de tableau de bord et de robot. Il prend en charge plusieurs des mêmes *widgets* que l'interface graphique de simulation prend en charge, y compris la visualisation de pose de robot et le traçage avancé. Dans son état actuel, il est destiné à être utilisé comme un outil de débogage du programmeur et non comme un tableau de bord pour une utilisation en compétition.

Lancement de Glass

Glass peut être lancé en sélectionnant le menu points de suspension (...) dans VS Code, en cliquant sur *Start Tool* puis en choisissant *Glass*.

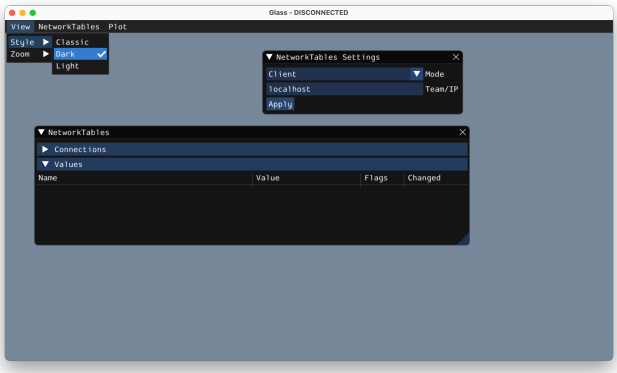


Note : Vous pouvez également lancer Glass directement en accédant à `~/wpilib/YYYY/tools` et en exécutant ``Glass.py``(Linux et macOS) ou en utilisant le raccourci à l'intérieur du dossier de bureau WPILib Tools (Windows).

Modification des paramètres d'affichage

L'élément de menu *View* contient les réglages *Zoom* et *Style* : qui peuvent être personnalisés. L'option *Zoom* dicte la taille du texte dans l'application tandis que l'option *Style* vous permet de choisir entre les options `Classic`, `Light`, et `Dark`.

Un exemple du paramètre de style `Dark` est ci-dessous :



Effacement des données d'application

Les données d'application pour Glass, y compris la taille et la position des widgets ainsi que d'autres informations personnalisées pour les widgets, sont stockées dans un fichier `glass.ini`. L'emplacement de ce fichier varie en fonction de votre système d'exploitation :

- Sous Windows, le fichier de configuration se trouve dans `%APPDATA%`.
- Sous macOS, le fichier de configuration se trouve dans `~/Library/Preferences`.
- Sous Linux, le fichier de configuration se trouve dans `$XDG_CONFIG_HOME` ou `~/.config` si le premier n'existe pas.

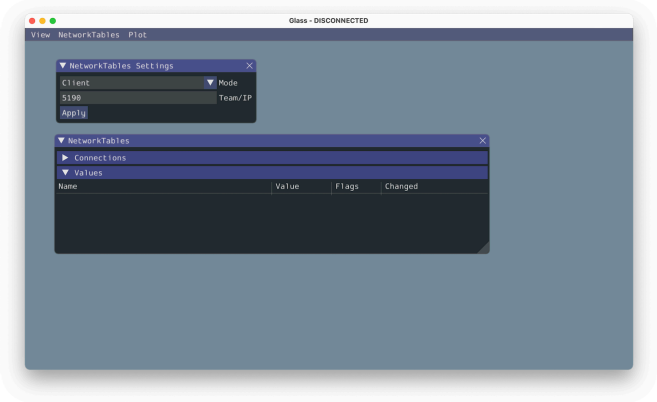
Le fichier de configuration `glass.ini` peut simplement être supprimé pour restaurer Glass à son état initial.

11.4.2 Établissement de connexions NetworkTables

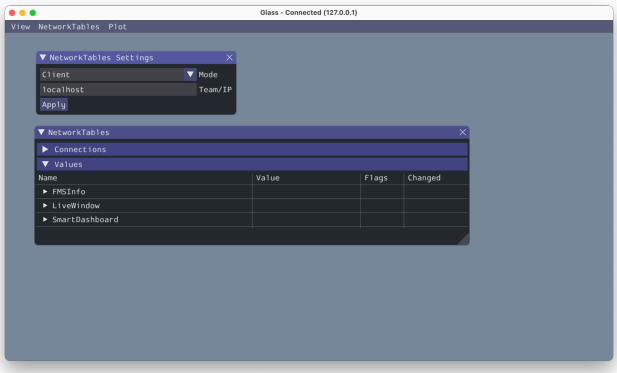
Glass utilise le protocole *NetworkTables* pour établir une connexion avec votre programme de robot. Il est également utilisé pour transmettre et recevoir des données vers et depuis le robot.

Connexion à un robot

Lorsque Glass est lancé pour la première fois, vous verrez deux widgets - *NetworkTables Settings* et *NetworkTables*. Pour vous connecter à un robot, sélectionnez *Client* sous *Mode* dans le widget *NetworkTables Settings*, entrez votre numéro d'équipe et cliquez sur *Apply*.



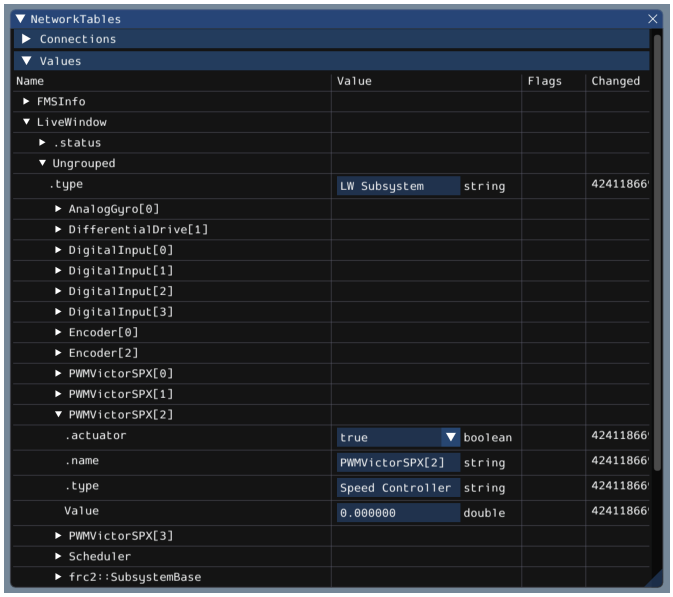
Vous pouvez également vous connecter à un robot qui s'exécute en simulation sur votre ordinateur (y compris les robots Romi) en tapant `localhost` dans la case *Team/IP*.



Important : L'état de la connexion NetworkTables est toujours visible sur la barre de titre de l'application Glass.

Affichage des entrées NetworkTables

Le widget *NetworkTables* peut être utilisé pour afficher toutes les entrées qui sont envoyées via NetworkTables. Ces entrées sont classées hiérarchiquement par table principale, sous-table, etc.



De plus, vous pouvez afficher tous les clients NetworkTables connectés sous le volet *Connections* du widget.

11.4.3 Widgets pour Glass

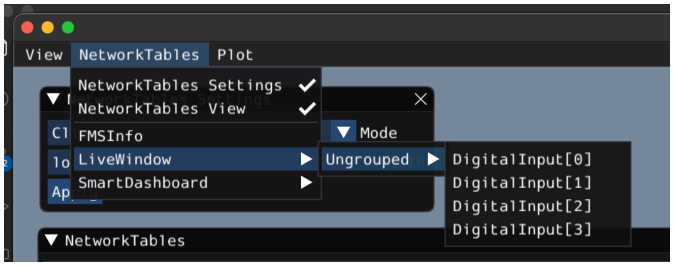
Des widgets spécialisés sont disponibles pour certains types qui existent dans le code du robot. Ceux-ci incluent des objets qui sont envoyés manuellement sur NetworkTables tels que les instances `SendableChooser`, ou le matériel qui est automatiquement envoyé sur LiveWindow.

Note : La prise en charge des widgets dans Glass en est encore à ses débuts - par conséquent, il n'y a qu'un petit ensemble de widgets disponibles. Cette liste augmentera au fur et à mesure que le travail de développement se poursuivra.

Note : Un widget peut être renommé en cliquant avec le bouton droit de la souris sur son en-tête et en spécifiant un nouveau nom.

Widgets matériels (Hardware)

Les widgets pour un matériel spécifique (tels que les contrôleurs de moteur) sont généralement disponibles via LiveWindow. On peut y accéder en sélectionnant l'option de menu *NetworkTables*, en cliquant sur *LiveWindow* et en choisissant le widget souhaité.



La liste du matériel (Hardware), qui est envoyé sur LiveWindow automatiquement et qui a des widgets est ci-dessous :

- `DigitalInput`
- `DigitalOutput`
- `SpeedController`
- `Gyro`

Voici un exemple de widget pour gyroscopes :

Widget Sélecteur envoyable

Le widget *Sendable Chooser* représente une instance `SendableChooser` à partir du code du robot. Il est souvent utilisé pour sélectionner des modes autonomes. Comme les autres tableaux de bord, votre instance `SendableChooser` doit simplement être envoyée à l'aide d'une API NetworkTables. Le plus simple est d'utiliser quelque chose comme `SmartDashboard` :

JAVA

```
SmartDashboard.putData("Auto Selector", m_selector);
```

C++

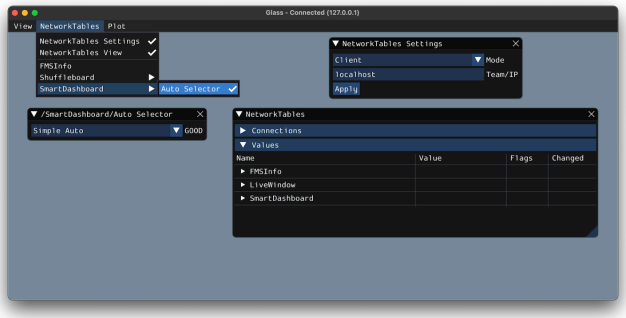
```
frc::SmartDashboard::PutData("Auto Selector", &m_selector);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData("Auto Selector", selector)
```

Note : Pour plus d'informations sur la création d'un `SendableChooser`, veuillez consulter *ce document*.

Le widget *Sendable Chooser* apparaîtra dans le menu *NetworkTables* et sous le nom de la table principale sur laquelle l'instance a été envoyée. Dans l'exemple ci-dessus, le nom de la table principale serait *SmartDashboard*.



Widget de contrôleur PID

Le widget *PID Controller* vous permet de régler rapidement les valeurs PID pour un certain contrôleur. Une instance `PIDController` doit être envoyée à l'aide d'une API NetworkTables. Le plus simple est d'utiliser quelque chose comme `SmartDashboard` :

JAVA

```
SmartDashboard.putData("Elevator PID Controller", m_elevatorPIDController);
```

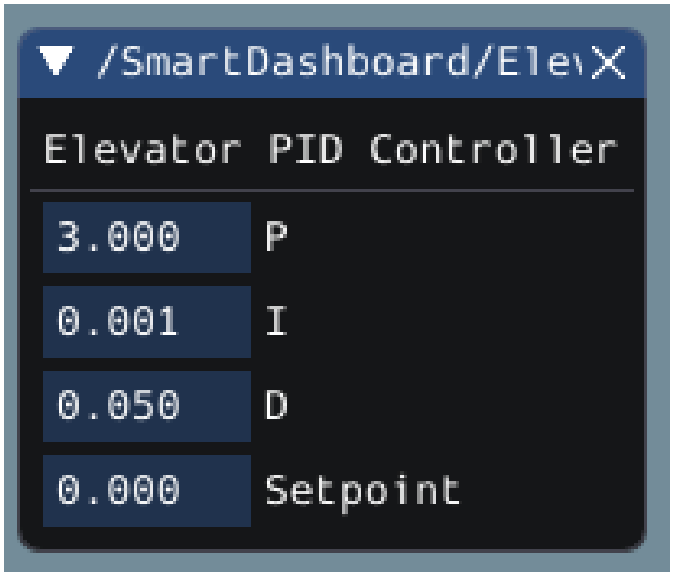
C++

```
frc::SmartDashboard::PutData("Elevator PID Controller", &m_elevatorPIDController);
```

PYTHON

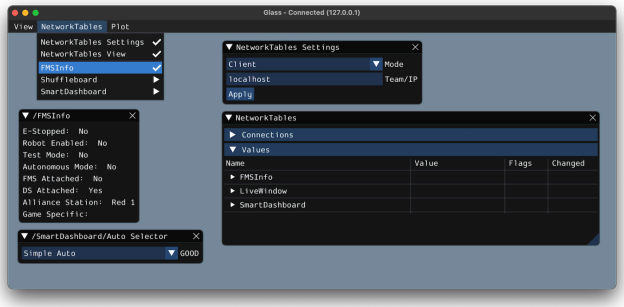
```
from wpilib import SmartDashboard

SmartDashboard.putData("Elevator PID Controller", elevatorPIDController)
```

Cela vous permet de régler rapidement les valeurs P, I et D pour divers points de consigne.



Widget FMSInfo

The *FMSInfo* widget is created by default when Glass connects to a robot. This widget displays basic information about the robot's enabled state, whether a Driver Station is connected, whether an *FMS* is connected, the game-specific data, etc. It can be viewed by selecting the *NetworkTables* menu item and clicking on *FMSInfo*.



11.4.4 Widgets pour l'infrastructure logicielle orientée commande

Glass a également plusieurs widgets spécifiques à *l'infrastructure logicielle orientée commande*. Ceux-ci incluent des widgets pour planifier les commandes, afficher les commandes en cours d'exécution sur un sous-système spécifique ou afficher l'état du *planificateur de commandes*.

Widget de sélection de commandes

Le widget *Command Selector* vous permet de démarrer et d'annuler une instance spécifique d'une commande (envoyée via NetworkTables) depuis Glass. Par exemple, vous pouvez créer une instance de `MyCommand` et l'envoyer à SmartDashboard :

JAVA

```
MyCommand command = new MyCommand(...);
SmartDashboard.putData("My Command", command);
```

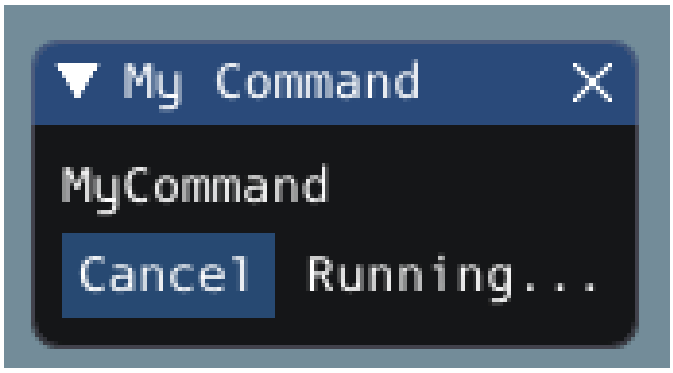
C++

```
#include <frc/smartdashboard/SmartDashboard.h>

...

MyCommand command{...};
frc::SmartDashboard::PutData("My Command", &command);
```

PYTHON

```
from wpilib import SmartDashboard

command = MyCommand(...)
SmartDashboard.putData("My Command", command)
```

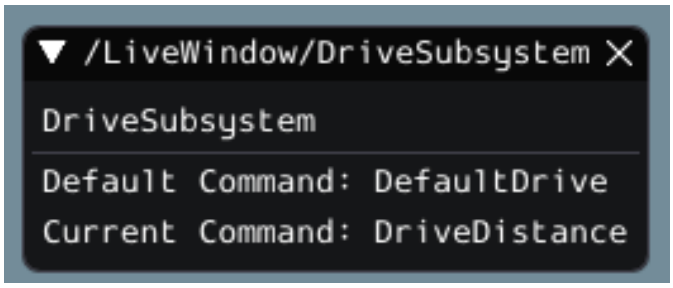
Note : L'instance `MyCommand` peut également être envoyée en utilisant une API NetworkTables de niveau inférieur ou en utilisant Shuffleboard API. Dans ce cas, l'API `SmartDashboard` a été utilisée, ce qui signifie que le widget *Command Selector* apparaîtra sous le nom de la table `SmartDashboard`.



Le widget a deux états. Lorsque la commande n'est pas en cours d'exécution, un bouton *Run* apparaît - en cliquant dessus, la commande sera programmée. Lorsque la commande est en cours d'exécution, un bouton *Cancel*, accompagné du texte *Running ...* apparaît (comme indiqué ci-dessus). Cela annulera la commande.

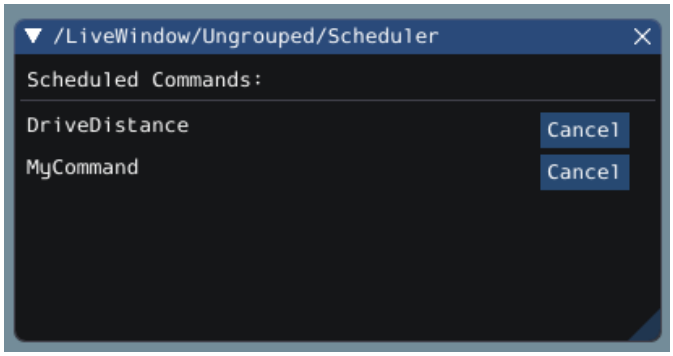
Widget de sous-système

The *Subsystem* widget can be used to see the default command and the currently scheduled command on a specific subsystem. If you are using the `SubsystemBase` base class, your subsystem will be automatically sent to NetworkTables over LiveWindow. To view this widget, look under the *LiveWindow* main table name in the *NetworkTables* menu.



Widget du planificateur de commandes

Le widget *Command Scheduler* vous permet de voir toutes les commandes actuellement programmées. De plus, n'importe laquelle de ces commandes peut être annulée à partir de l'interface graphique.



L'instance `CommandScheduler` est automatiquement envoyée à NetworkTables via LiveWindow. Pour afficher ce widget, regardez sous le nom de la table principale *LiveWindow* dans le menu *NetworkTables*.

11.4.5 Le widget Field2d

Glass prend en charge l'affichage de la position de votre robot sur le terrain à l'aide du widget *Field2d*. Une instance de la classe `Field2d` doit être créée, envoyée via NetworkTables et mise à jour périodiquement avec la dernière pose de robot dans votre code.

Envoi de la pose du robot à partir du code utilisateur

Pour envoyer la position de votre robot (généralement obtenue par *odométrie* ou un estimateur de pose), une instance `Field2d` doit être créée dans le code du robot et envoyée via NetworkTables. L'instance doit ensuite être mise à jour périodiquement avec la dernière pose du robot.

JAVA

```java
private final Field2d m_field = new Field2d();

// Do this in either robot or subsystem init
SmartDashboard.putData("Field", m_field);

// Do this in either robot periodic or subsystem periodic
m_field.setRobotPose(m_odometry.getPoseMeters());
```

C++

```cpp
#include <frc/smartdashboard/Field2d.h>
#include <frc/smartdashboard/SmartDashboard.h>

frc::Field2d m_field;

// Do this in either robot or subsystem init
frc::SmartDashboard::PutData("Field", &m_field);

// Do this in either robot periodic or subsystem periodic
m_field.SetRobotPose(m_odometry.GetPose());
```

PYTHON

```
from wpilib import SmartDashboard, Field2d

self.field = Field2d()

# Do this in either robot or subsystem init
SmartDashboard.putData("Field", self.field)

# Do this in either robot periodic or subsystem periodic
self.field.setRobotPose(self.odometry.getPose())
```

Note : L'instance `Field2d` peut également être envoyée en utilisant une API NetworkTables de niveau inférieur ou en utilisant *Shuffleboard API*. Dans ce cas, l'API `SmartDashboard` a été utilisée, ce qui signifie que le widget *Field2d* apparaîtra sous le nom de la table `SmartDashboard`.

Envoi de trajectoires à Field2d

La visualisation de votre trajectoire est une excellente étape de débogage pour vérifier que vos trajectoires ont été créées comme prévu. Les trajectoires peuvent être facilement visualisées dans *Field2d* en faisant usage des fonctions `setTrajectory()`/`SetTrajectory()`.

JAVA

```
44  public void robotInit() {
45    // Create the trajectory to follow in autonomous. It is best to initialize
46    // trajectories here to avoid wasting time in autonomous.
47    m_trajectory =
48        TrajectoryGenerator.generateTrajectory(
49            new Pose2d(0, 0, Rotation2d.fromDegrees(0)),
50            List.of(new Translation2d(1, 1), new Translation2d(2, -1)),
51            new Pose2d(3, 0, Rotation2d.fromDegrees(0)),
52            new TrajectoryConfig(Units.feetToMeters(3.0), Units.feetToMeters(3.0)));
53
54    // Create and push Field2d to SmartDashboard.
55    m_field = new Field2d();
56    SmartDashboard.putData(m_field);
57
58    // Push the trajectory to Field2d.
59    m_field.getObject("traj").setTrajectory(m_trajectory);
60  }
```

C++

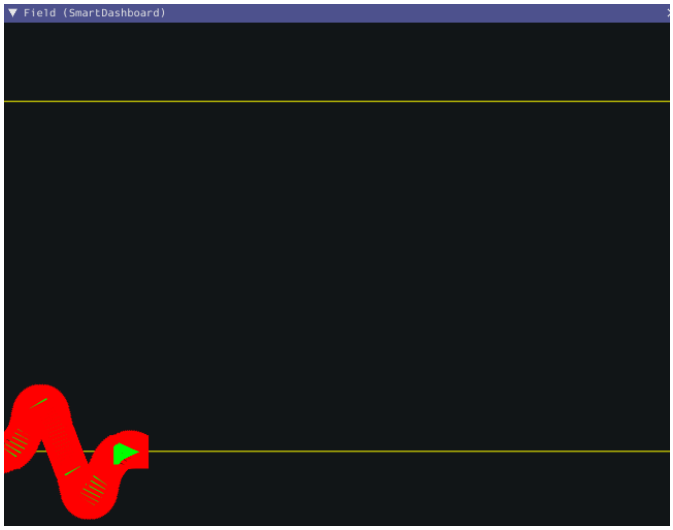
```
18  void AutonomousInit() override {
19    // Start the timer.
20    m_timer.Start();
21
22    // Send Field2d to SmartDashboard.
23    frc::SmartDashboard::PutData(&m_field);
24
25    // Reset the drivetrain's odometry to the starting pose of the trajectory.
26    m_drive.ResetOdometry(m_trajectory.InitialPose());
27
28    // Send our generated trajectory to Field2d.
29    m_field.GetObject("traj")->SetTrajectory(m_trajectory);
30  }
```

PYTHON

```
    def robotInit(self):

        # An example trajectory to follow during the autonomous period.
        self.trajectory = wpimath.trajectory.TrajectoryGenerator.generateTrajectory(
            wpimath.geometry.Pose2d(0, 0, wpimath.geometry.Rotation2d.fromDegrees(0)),
            [
                wpimath.geometry.Translation2d(1, 1),
                wpimath.geometry.Translation2d(2, -1),
            ],
            wpimath.geometry.Pose2d(3, 0, wpimath.geometry.Rotation2d.fromDegrees(0)),
            wpimath.trajectory.TrajectoryConfig(
                wpimath.units.feetToMeters(3.0), wpimath.units.feetToMeters(3.0)
            ),
        )

        # Create Field2d for robot and trajectory visualizations.
        self.field = wpilib.Field2d()

        # Create and push Field2d to SmartDashboard.
        wpilib.SmartDashboard.putData(self.field)

        # Push the trajectory to Field2d.
        self.field.getObject("traj").setTrajectory(self.trajectory)
```

Affichage des trajectoires à l'aide de Glass

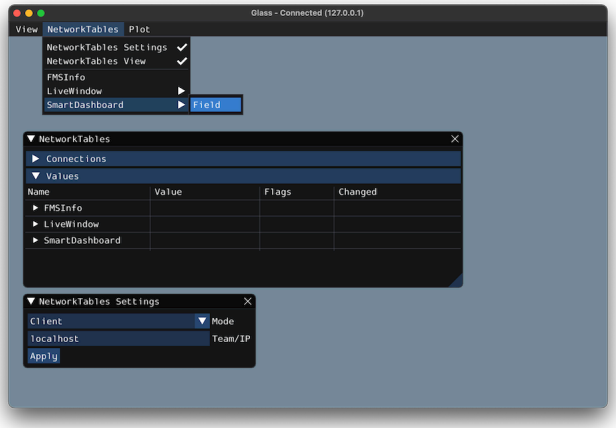
La trajectoire envoyée peut être visualisée avec *Glass* via la liste déroulante *NetworkTables -> SmartDashboard -> Field2d*.



Note : The above example which uses the RamseteController (Java / C++ / Python) will not show the sent trajectory until autonomous is enabled at least once.

Affichage de la pose du robot dans Glass

Après avoir envoyé l'instance `Field2d` sur NetworkTables, le widget *Field2d* peut être ajouté à Glass en sélectionnant *NetworkTables* dans la barre de menu, en choisissant le nom de la table sur laquelle l'instance a été envoyée, et puis en cliquant sur le bouton *Field*.

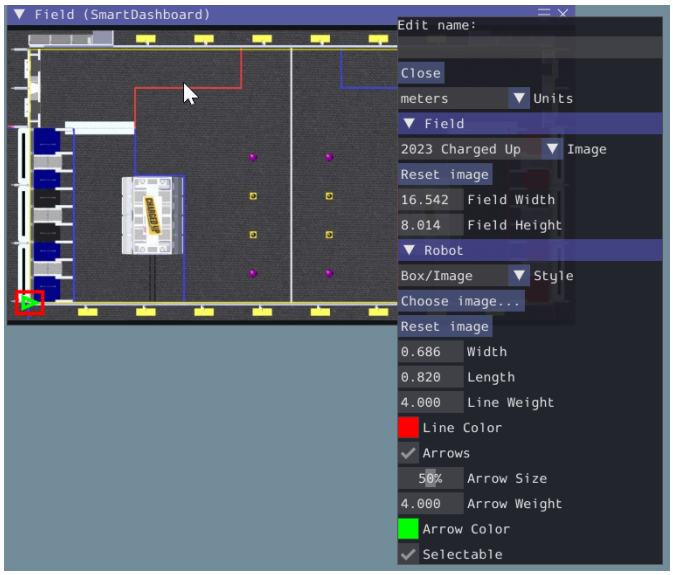


Une fois que le widget apparaît, vous pouvez le redimensionner et le placer sur l'espace de travail Glass comme vous le souhaitez. Un clic droit en haut du widget vous permettra de personnaliser le nom du widget, de sélectionner une image de champ personnalisée, de sélectionner une image de robot personnalisée et de choisir les dimensions du champ et du robot.

You can choose from an existing field layout using the *Image* drop-down. Or you can select a custom file by setting the *Image* to `Custom` and selecting *Choose image...*. You can choose to either select an image file or a PathWeaver JSON file as long as the image file is in the same directory. Choosing the JSON file will automatically import the correct location of the field in

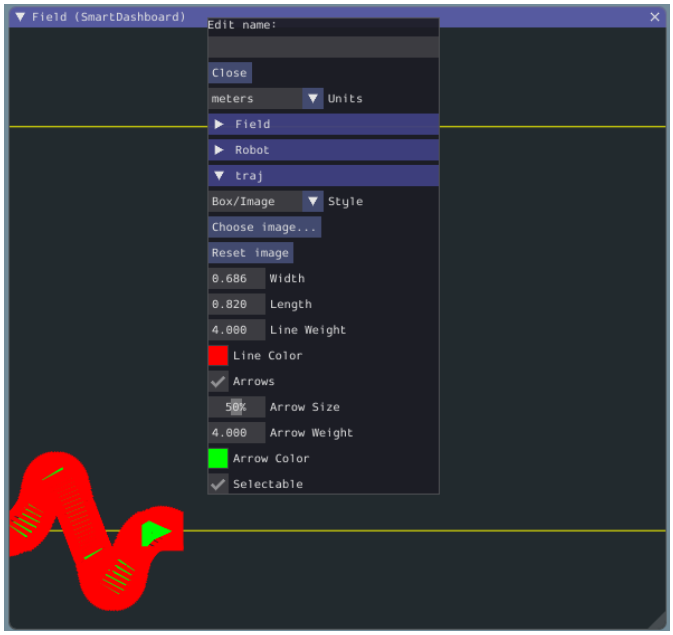
the image and the correct size of the field.

Note : You can retrieve the latest field image and JSON files from here. This is the same image and JSON that are used when generating paths using *PathWeaver*.



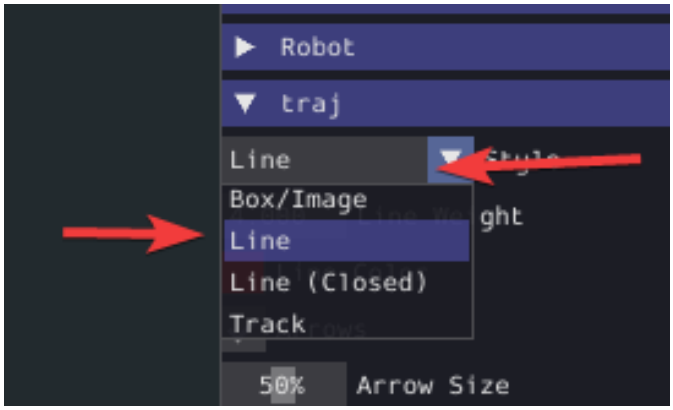
Modification du style de pose

Les Poses peuvent être personnalisées de nombreuses façons en cliquant avec le bouton droit de la souris sur la barre de menus Field2d. Exemples de personnalisation : largeur de la ligne, épaisseur de la ligne, style, largeur de la flèche, épaisseur de la flèche, couleur, etc.

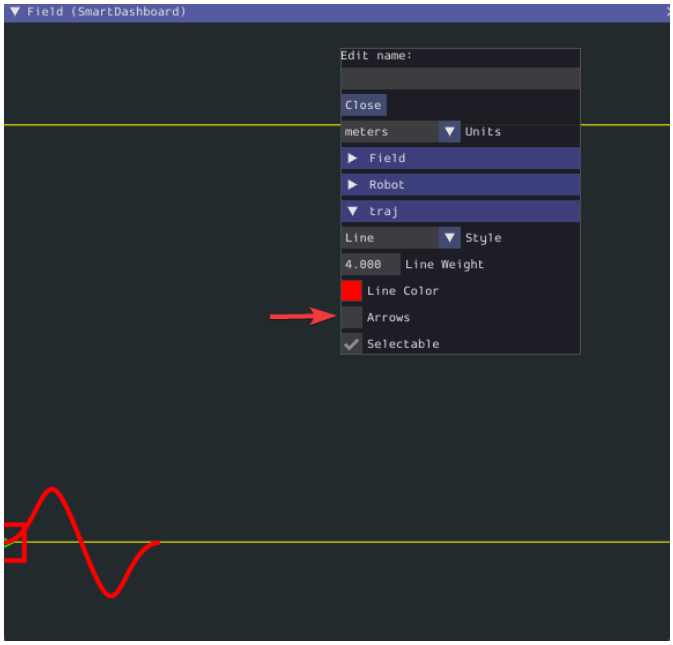


Une utilisation de la personnalisation du style de Pose consiste à convertir l'objet de Pose `traj` précédemment affiché en une ligne, plutôt qu'une liste de Poses. Cliquez sur la liste

déroulante *Style* et sélectionnez *Line*. Vous devriez remarquer un changement immédiat dans l'apparence de la trajectoire.

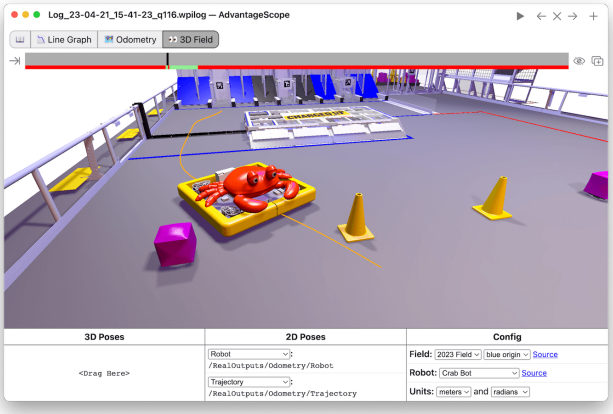


Now, uncheck the *Arrows* checkbox. This will cause our trajectory to look like a nice and fluid line !



Viewing Pose Data with AdvantageScope

AdvantageScope is an alternative option for viewing pose data from a `Field2d` object, including data recorded to a log file using *WPILib data logs*. Both 2D and 3D visualizations are supported. See the documentation for the odometry and 3D field tabs for more details.



11.4.6 Le widget Mechanism2d

Glass prend en charge l'affichage de représentations en forme de lignes droites des mécanismes de votre robot à l'aide du widget *Mechanism2d*. Il prend en charge les combinaisons de ligaments qui peuvent tourner et / ou s'étendre ou se rétracter, tels que les bras et les élevateurs et ils peuvent être combinés pour des mécanismes plus compliqués. Une instance de la classe `Mechanism2d` doit être créée et configuréé, envoyée via NetworkTables et mise à jour périodiquement selon les derniers états du mécanisme dans votre code robot. Il peut également être utilisé avec *Simulation physique* pour visualiser et programmer les mécanismes de votre robot avant sa construction effective.

Création et configuration de l'instance Mechanism2d

L'objet `Mechanism2d` est le « canvas » où le mécanisme est dessiné. Le nœud racine est l'endroit où le mécanisme est ancré à `Mechanism2d`. Pour un seul bras articulé, ce serait le point de pivot. Pour un élevateur, ce serait l'endroit où il est attaché à la base du robot. Pour obtenir un nœud racine (représenté par un objet `MechanismRoot2d` object), invoquez la méthode `getRoot(name, x, y)` sur l'objet conteneur `Mechanism2d`. Le nom est utilisé pour nommer la racine dans NetworkTables et doit être unique, mais n'est pas important. Le système de coordonnées `x / y` suit la même orientation que `Field2d` - `(0,0)` est en bas à gauche.

Dans les exemples ci-dessous, un élevateur est dessiné, avec un poignet rotatif au dessus de celui-ci. L'exemple complet de Mechanism2d est disponible dans Java / C++

JAVA

```
    // the main mechanism object
    Mechanism2d mech = new Mechanism2d(3, 3);
    // the mechanism root node
    MechanismRoot2d root = mech.getRoot("climber", 2, 0);
```

C++

```
  // the main mechanism object
  frc::Mechanism2d m_mech{3, 3};
  // the mechanism root node
  frc::MechanismRoot2d* m_root = m_mech.GetRoot("climber", 2, 0);
```

PYTHON

```
        # the main mechanism object
        self.mech = wpilib.Mechanism2d(3, 3)
        # the mechanism root node
        self.root = self.mech.getRoot("climber", 2, 0)
```

Chaque objet `MechanismLigament2d` représente un niveau du mécanisme. Il a trois paramètres requis, un nom, une longueur initiale à dessiner (par rapport à la taille de l'objet `Mechanism2d`), et un angle initial pour dessiner le ligament en degrés. Les angles des ligaments sont relatifs au ligament parent et suivent une notation mathématique - la même que *Rotation2d* (dans le sens inverse des aiguilles d'une montre, sens positif). Un ligament basé sur la racine avec un angle de zéro pointera vers la droite. Deux paramètres facultatifs vous permettent de modifier la largeur (également par rapport à la taille de l'objet Mechanism2d) et la couleur. Invoquez `append()`/`Append()` sur un nœud racine ou un nœud de ligament pour ajouter un autre nœud à la figure. En Java, passez un objet `MechanismLigament2d` déjà construit pour l'ajouter. En C++, passez les paramètres de construction afin de construire et d'ajouter un ligament.

JAVA

```
    // MechanismLigament2d objects represent each "section"/"stage" of the mechanism,␣
↪and are based
    // off the root node or another ligament object
    m_elevator = root.append(new MechanismLigament2d("elevator",␣
↪kElevatorMinimumLength, 90));
    m_wrist =
        m_elevator.append(
            new MechanismLigament2d("wrist", 0.5, 90, 6, new Color8Bit(Color.
↪kPurple)));
```

C++

```
// MechanismLigament2d objects represent each "section"/"stage" of the
// mechanism, and are based off the root node or another ligament object
frc::MechanismLigament2d* m_elevator =
    m_root->Append<frc::MechanismLigament2d>("elevator", 1, 90_deg);
frc::MechanismLigament2d* m_wrist =
    m_elevator->Append<frc::MechanismLigament2d>(
        "wrist", 0.5, 90_deg, 6, frc::Color8Bit{frc::Color::kPurple});
```

PYTHON

```
        # MechanismLigament2d objects represent each "section"/"stage" of the mechanism, and are based
        # off the root node or another ligament object
        self.elevator = self.root.appendLigament(
            "elevator", self.kElevatorMinimumLength, 90
        )
        self.wrist = self.elevator.appendLigament(
            "wrist", 0.5, 90, 6, wpilib.Color8Bit(wpilib.Color.kPurple)
        )
```

Ensuite, publiez l'objet `Mechanism2d` sur NetworkTables :

JAVA

```
    // post the mechanism to the dashboard
    SmartDashboard.putData("Mech2d", mech);
```

C++

```
    // publish to dashboard
    frc::SmartDashboard::PutData("Mech2d", &m_mech);
```

PYTHON

```
        # post the mechanism to the dashboard
        wpilib.SmartDashboard.putData("Mech2d", self.mech)
```

Note : L'instance `Mechanism2d` peut également être envoyée à l'aide d'une API NetworkTables de bas niveau ou à l'aide de l'API *Shuffleboard API*. Dans ce cas, l'API `SmartDashboard` a été utilisée, ce qui signifie que le widget *Mechanism2d* apparaîtra sous le nom de la table `SmartDashboard`.

Pour manipuler un angle ou une longueur de ligament, invoquer `setLength()` ou `setAngle()` sur l'objet `MechanismLigament2d`. Lorsque vous manipulez la longueur des ligaments en fonction des mesures du capteur, assurez-vous d'ajouter la longueur minimale pour éviter les ligaments de longueur 0 (et donc invisibles).

JAVA

```java
  @Override
  public void robotPeriodic() {
    // update the dashboard mechanism's state
    m_elevator.setLength(kElevatorMinimumLength + m_elevatorEncoder.getDistance());
    m_wrist.setAngle(m_wristPot.get());
  }
```

C++

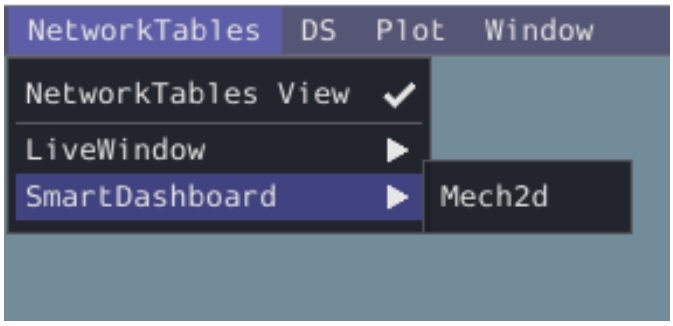
```cpp
  void RobotPeriodic() override {
    // update the dashboard mechanism's state
    m_elevator->SetLength(kElevatorMinimumLength +
                          m_elevatorEncoder.GetDistance());
    m_wrist->SetAngle(units::degree_t{m_wristPotentiometer.Get()});
  }
```

PYTHON

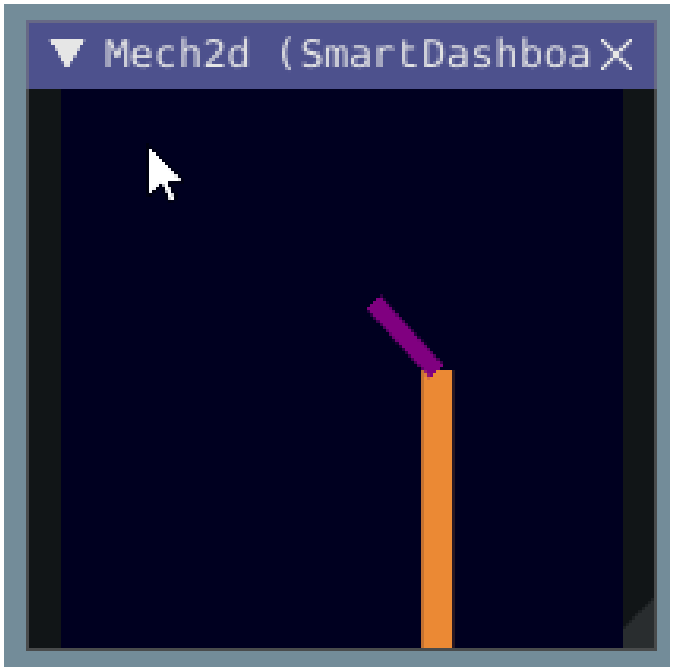
```python
    def robotPeriodic(self):
        # update the dashboard mechanism's state
        self.elevator.setLength(
            self.kElevatorMinimumLength + self.elevatorEncoder.getDistance()
        )
        self.wrist.setAngle(self.wristPot.get())
```

Affichage de Mechanism2d sous Glass

Après avoir envoyé l'instance `Mechanism2d` à traver NetworkTables, le widget *Mechanism2d* peut être ajouté à Glass en sélectionnant *NetworkTables* dans la barre de menus, en choisissant le nom de la table sur laquelle l'instance a été envoyée, puis en cliquant sur le bouton *Field*.

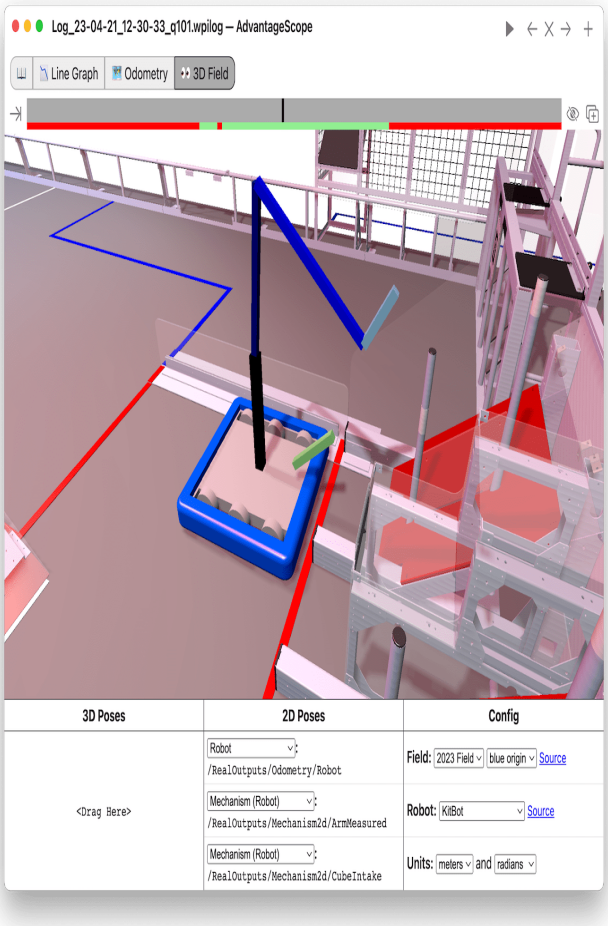


Une fois que le widget apparaît comme indiqué ci-dessous, vous pouvez le redimensionner et le placer sur l'espace de travail Glass comme vous le souhaitez. Un cliquant sur le bouton droit sur le haut du widget, vous pouver personnaliser le nom du widget. Au fur et à mesure que le potentiomètre du poignet et l'encodeur de l'élevateur changent, les paramètres du mécanisme se mettent à jour dans le widget.



Viewing the Mechanism2d in AdvantageScope

AdvantageScope is an alternative option for viewing a `Mechanism2d` object, including data recorded to a log file using *WPILib data logs*. Both 2D and 3D visualizations are supported. See the documentation for the mechanism and 3D field tabs for more details.



Prochaines étapes

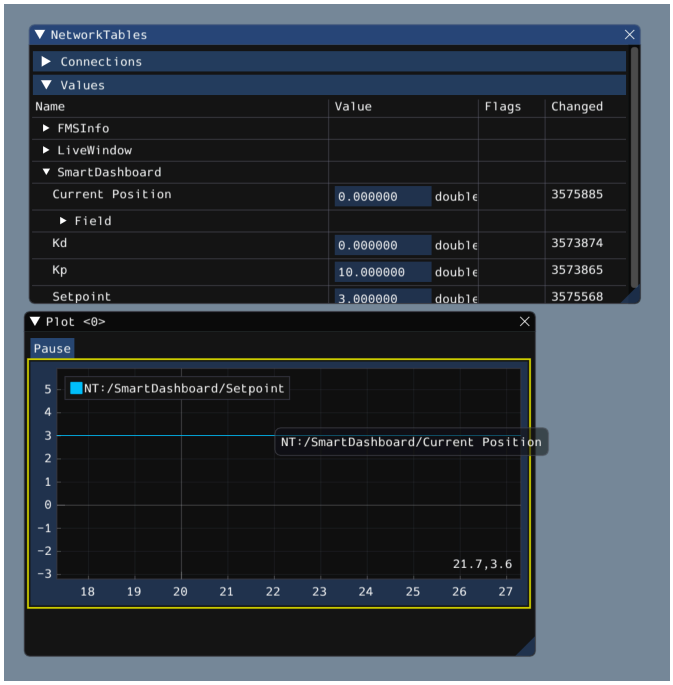
As mentioned above, the Mechanism2d visualization can be combined with *Physics Simulation* to help you program mechanisms before your robot is built. The ArmSimulation (Java / C++ / Python) and ElevatorSimulation (Java / C++ / Python) examples combine physics simulation and Mechanism2d visualization so that you can practice programming a single jointed arm and elevator without a robot.

11.4.7 Traçage de données

Glass excelle dans le traçage complet et performant des données de NetworkTables. Certaines fonctionnalités incluent des tracés redimensionnables, des tracés avec plusieurs axes y et la possibilité de mettre en pause, d'examiner et de reprendre les tracés.

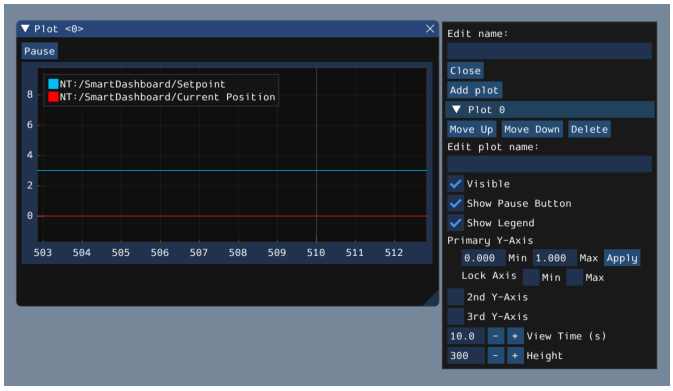
Créer un tracé

Un nouveau widget de tracé peut être créé en sélectionnant le bouton *Plot* dans la barre de menu principale puis en cliquant sur *New Plot Window*. Plusieurs tracés individuels peuvent être ajoutés à chaque fenêtre de tracé. Pour ajouter un tracé dans une fenêtre de tracé, cliquez sur le bouton *Add plot* à l'intérieur du widget. Ensuite, vous pouvez faire glisser diverses sources du widget *NetworkTables* dans le graphique :



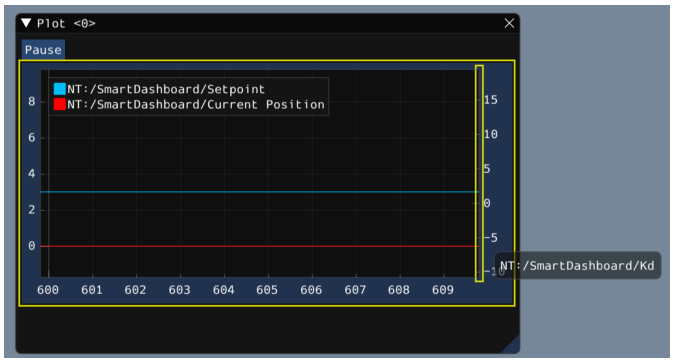
Manipulation des tracés

Vous pouvez cliquer et faire glisser sur le graphique pour vous déplacer et défiler au-dessus du graphique pour agrandir ou réduire les axes y. Double cliquer sur le graphique mettra automatiquement à l'échelle de sorte que les limites de l'agrandissement et des axes s'adaptent à toutes les données représentées. En outre, en cliquant à droite sur le graphique vous aurez accès à une pléthore d'options, y compris si vous voulez afficher des axes y secondaires et tertiaires, si vous souhaitez verrouiller certains axes, etc.

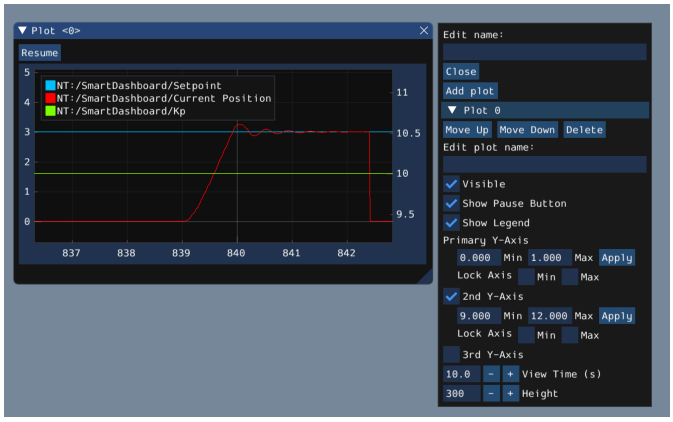


Si vous choisissez de rendre disponibles les axes y secondaires et tertiaires, vous pouvez faire glisser les sources de données sur ces axes pour que leurs lignes correspondent à l'axe

souhaité :

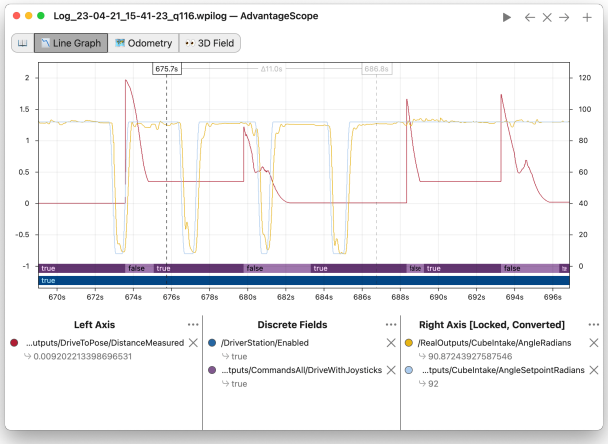


Ensuite, vous pouvez verrouiller certains axes afin que leur plage reste toujours constante, quel que soit le panoramique. Dans cet exemple, la plage de l'axe secondaire (avec l'entrée `/SmartDashboard/Kp`) a été verrouillée entre 9 et 12.



Plotting with AdvantageScope

AdvantageScope is an alternative option for creating plots, including from data recorded to a log file using *WPILib data logs*. See the documentation for the line graph tab for more details.



11.5 AdvantageScope

AdvantageScope is a data visualization tool for *NetworkTables*, *WPILib data logs*, and *Driver Station logs*. It is a programmer's tool (rather than a competition dashboard) and can be used to debug real or simulated robot code from a log file or live over the network.

In Visual Studio Code, press `Ctrl+Shift+P` and type `WPILib` or click the WPILib logo in the top right to launch the WPILib Command Palette. Select *Start Tool*, then select *AdvantageScope*. You can also open any supported log file in AdvantageScope using a standard file browser.

Note : Detailed documentation for AdvantageScope can be found here. It is also available offline by clicking the book icon in the tab bar.

The capabilities of AdvantageScope include :

- Display of numeric, textual, and boolean data in graphs and tables
- Visualization of pose and mechanism data in 2D and 3D, including custom 3D robot models
- Automatic synchronization of data sources, including log files, match videos, and Zebra MotionWorks tracking
- Specialized displays for joysticks, swerve module states, and console text
- Analysis of numeric fields using histograms and statistical measures
- Multiple export options, including CSV and WPILib data logs

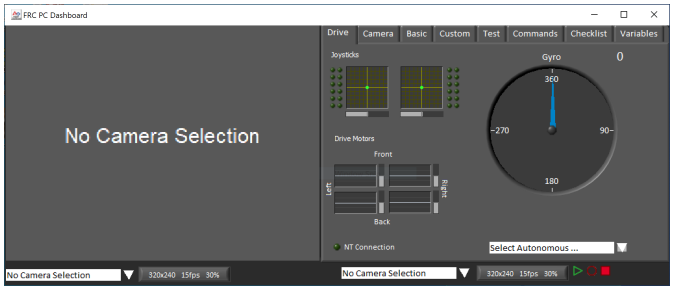
11.6 Dashboard LabVIEW

The LabVIEW Dashboard is easy to use and provides a lot of features straight out of the box like : camera streams, autonomous selection, and joystick feedback. It can be customized using LabVIEW by creating a new Dashboard project. While it *can be used* by Java, C++, or Python teams, they generally prefer SmartDashboard or Shuffleboard which can be customized in their respective language.

11.6.1 Dashboard FRC LabVIEW

L'application Dashboard installée et lancée par le FRC | reg | Driver Station est un programme LabVIEW conçu pour fournir aux équipes des informations de base sur leur robot, avec la possibilité d'étendre et de personnaliser les informations en fonction de leurs besoins. Cette application de tableau de bord utilise les *NetworkTables* et contient une variété d'outils que les équipes peuvent trouver utiles.

Dashboard LabVIEW

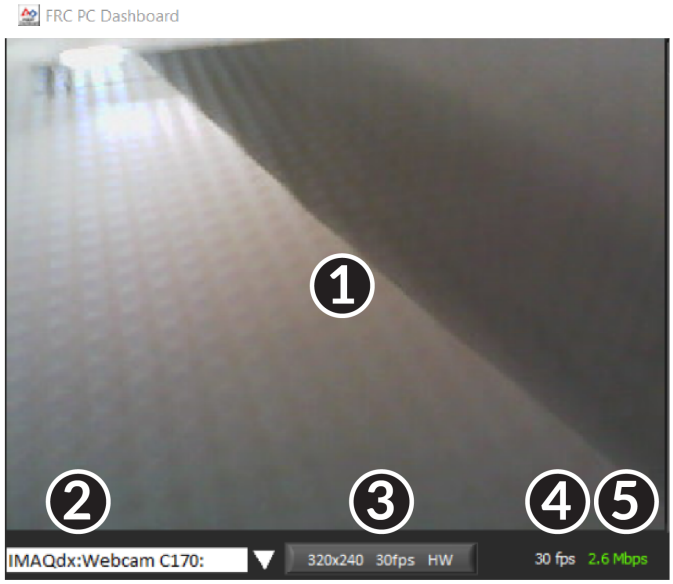


Le tableau de bord est divisé en deux sections principales. Le volet gauche est pour afficher une image de la caméra. Le volet droit contient :

- L'onglet Drive qui contient des indicateurs pour les valeurs du joystick et du moteur de déplacement (branché par défaut lorsqu'il est utilisé avec le code robot LabVIEW), un indicateur pour les données du gyroscope, une boîte de texte pour la sélection autonome, un indicateur de connexion et quelques contrôles et indicateurs pour la caméra
- L'onglet de Basic qui contient des contrôles et des indicateurs par défaut
- L'onglet camera qui contient une visionneuse de caméra secondaire, similaire à la visionneuse dans le volet gauche
- L'onglet Custom pour personnaliser le tableau de bord à l'aide de LabVIEW
- L'onglet Test pour une utilisation avec le mode Test dans l'infrastructure logicielle LabVIEW
- L'onglet Commands pour une utilisation avec la nouvelle infrastructure logicielle LabVIEW C&C
- L'onglet Checklist qui peut être utilisé pour créer des listes de tâches à remplir avant et/ou entre les matchs
- L'onglet Variables qui affiche les variables brutes NetworkTables dans un format de vue arborescente

L'application Dashboard LabVIEW inclut également des fonctionnalités d'enregistrement/lecture, situées en bas à droite. Pour plus de détails sur cette fonctionnalité, rendez-vous ci-dessous sous *Record/Playback*.

Image de la caméra et contrôles

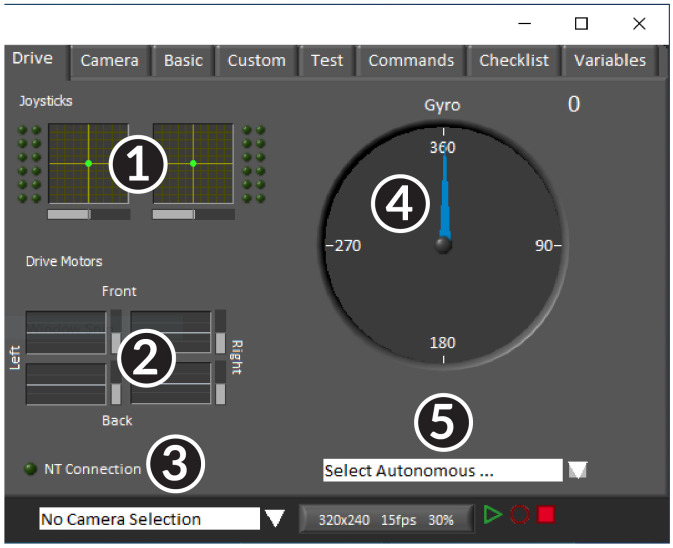


Le volet de gauche est utilisé pour afficher un flux vidéo de la caméra située sur le robot. Il existe également des commandes et des indicateurs liés à la caméra sous la zone des onglets :

1. Affichage de l'image de la caméra
2. Sélecteur de mode - Ce menu déroulant vous permet de sélectionner le type d'affichage de la caméra à utiliser. Les choix sont Camera Off, USB Camera SW (compression logicielle), USB Camera HW (compression matérielle) et IP Camera (caméra Axis). Notez que le paramètre caméra IP ne fonctionnera pas lorsque votre PC est connecté au roboRIO via USB
3. Paramètres de la caméra - Ce contrôle vous permet de modifier la résolution, la fréquence des images et la compression du flux d'images sur le tableau de bord, cliquez sur le contrôle sélectionner les paramètres de configuration.
4. Indicateur de bande passante - Indique l'utilisation approximative de la bande passante du flux d'images. L'indicateur affichera le vert pour une utilisation de bande passante « sûre », le jaune lorsque les équipes doivent faire preuve de prudence et le rouge si la bande passante du flux est au-delà des niveaux qui fonctionneront sur le terrain de compétition.
5. Fréquence d'images - Indique la fréquence d'images approximative reçue du flux d'images.

Astuce : L'indicateur de bande passante indique la bande passante combinée pour tous les flux de caméras ouverts.

Drive



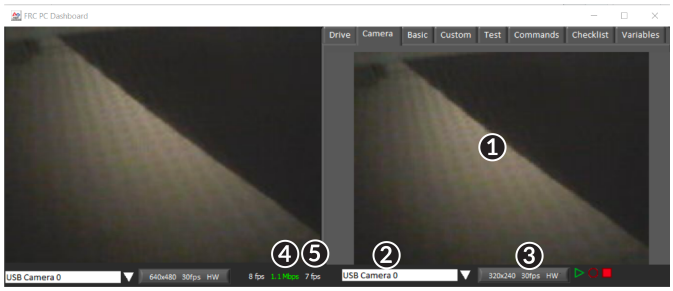
Le volet central contient une section qui fournit des informations sur les manettes et les commandes d'entraînement lorsqu'il est utilisé avec l'infrastructure logicielle LabVIEW et une section qui affiche l'état des NetworkTables et le sélecteur des modes autonomes :

1. Affiche des informations sur les axes X, Y et le Throttle et des valeurs de bouton pour jusqu'à 2 joysticks lors de l'utilisation de l'infrastructure logicielle LabVIEW
2. Affiche les valeurs envoyées aux contrôleurs de moteurs lors de l'utilisation de l'infrastructure logicielle LabVIEW
3. Affiche un indicateur de connexion pour les données NetworkTables du robot
4. Affiche une valeur du Gyro
5. Affiche une boîte de texte qui peut être utilisée pour sélectionner les modes autonomes. Les modèles de code de chaque langage ont des exemples d'utilisation de cette boîte pour faire une sélection parmi plusieurs programmes autonomes.

Ces indicateurs (autres que le Gyro) sont raccordés aux valeurs appropriées par défaut lors de l'utilisation de l'infrastructure logicielle LabVIEW. Pour plus d'informations sur leur utilisation avec le code C++/Java, consulter *Using the LabVIEW Dashboard with C++, Java, or Python Code*.

Camera

Astuce : Le volet gauche ne peut afficher qu'une seule sortie de caméra, donc pour afficher une deuxième sortie de caméra si nécessaire, utilisez l'onglet caméra sur le volet droit.

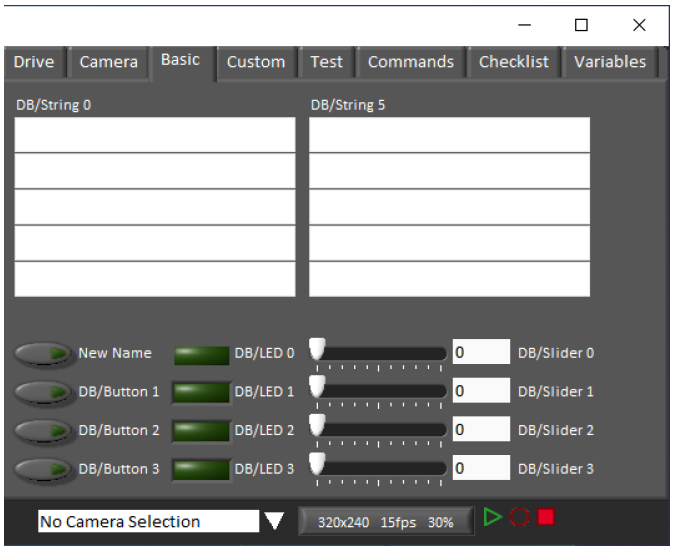


L'onglet caméra est utilisé pour afficher un flux vidéo d'une caméra située sur le robot. Il existe également des commandes et des indicateurs liés à la caméra sous la zone des onglets :

1. Affichage de l'image de la caméra
2. Sélecteur de mode - Ce menu déroulant vous permet de sélectionner le type d'affichage de la caméra à utiliser. Les choix sont Camera Off, USB Camera SW (compression logicielle), USB Camera HW (compression matérielle) et IP Camera (caméra Axis). Notez que le paramètre caméra IP ne fonctionnera pas lorsque votre PC est connecté au roboRIO via USB
3. Paramètres de la caméra - Ce contrôle vous permet de modifier la résolution, la fréquence des images et la compression du flux d'images sur le tableau de bord, cliquez sur le contrôle sélectionner les paramètres de configuration.
4. Indicateur de bande passante - Indique l'utilisation approximative de la bande passante du flux d'images. L'indicateur affichera le vert pour une utilisation de bande passante « sûre », le jaune lorsque les équipes doivent faire preuve de prudence et le rouge si la bande passante du flux est au-delà des niveaux qui fonctionneront sur le terrain de compétition.
5. Fréquence d'images - Indique la fréquence d'images approximative reçue du flux d'images.

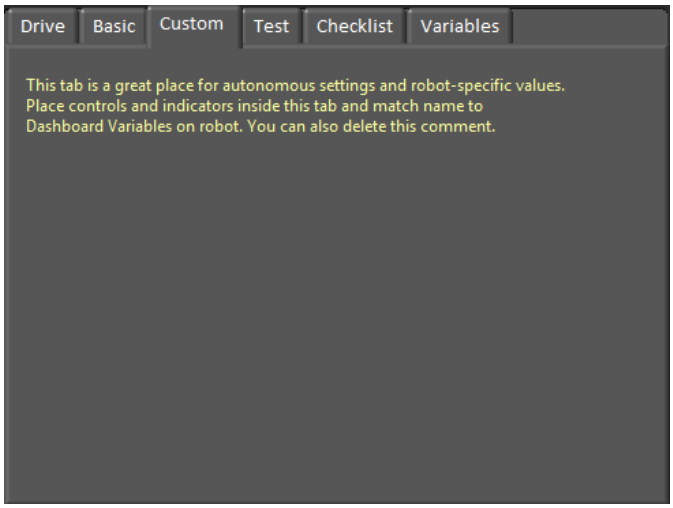
Astuce : L'indicateur de bande passante indique la bande passante combinée pour tous les flux de caméras ouverts.

Basic



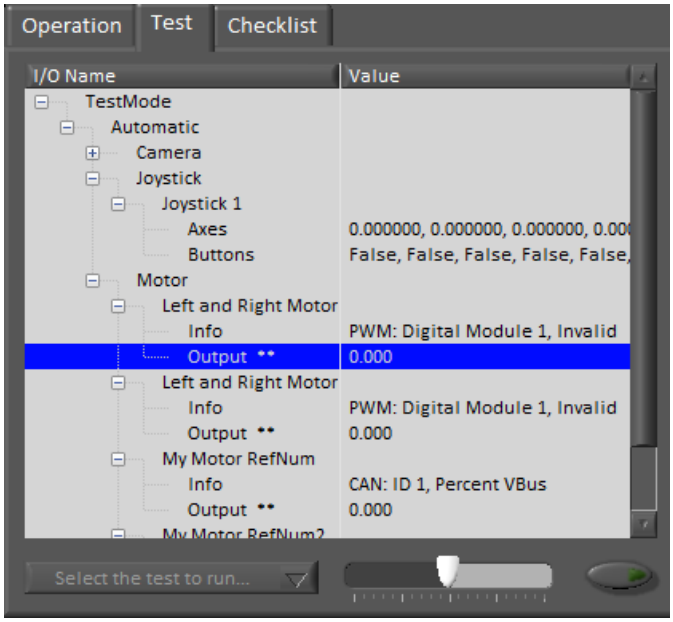
L'onglet Basic contient une variété de commandes/indicateurs bi-directionnels déjà remplis qui peuvent être utilisés pour contrôler le robot ou afficher les informations en provenance robot. Les noms des clés SmartDashboard associés à chaque élément sont étiquetés à côté de l'indicateur à l'exception des chaînes de caractères qui suivent le même modèle de nommage et l'incrément de DB/String 0 à DB/String 4 à gauche et DB/String 5 à DB/String 9 à droite. L'infrastructure logicielle LabVIEW contient un exemple de lecture des boutons et curseurs en mode Teleop. Il contient également un exemple de personnalisation des étiquettes dans Begin. Pour plus de détails sur l'utilisation de cet onglet avec le code C++Java, voir *Using the LabVIEW Dashboard with C++, Java, or Python Code*.

Custom



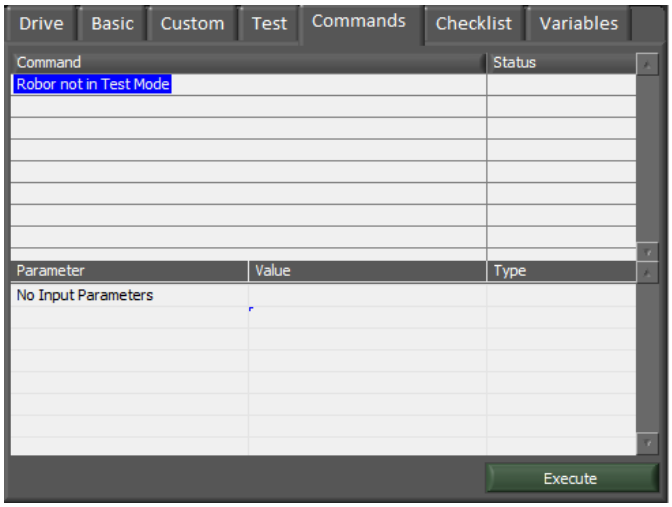
L'onglet Custom vous permet d'ajouter des contrôles/indicateurs supplémentaires au dashboard à l'aide de LabVIEW sans supprimer les fonctionnalités existantes. Pour personnaliser cet onglet, vous devrez créer un projet Dashboard dans LabVIEW.

Test



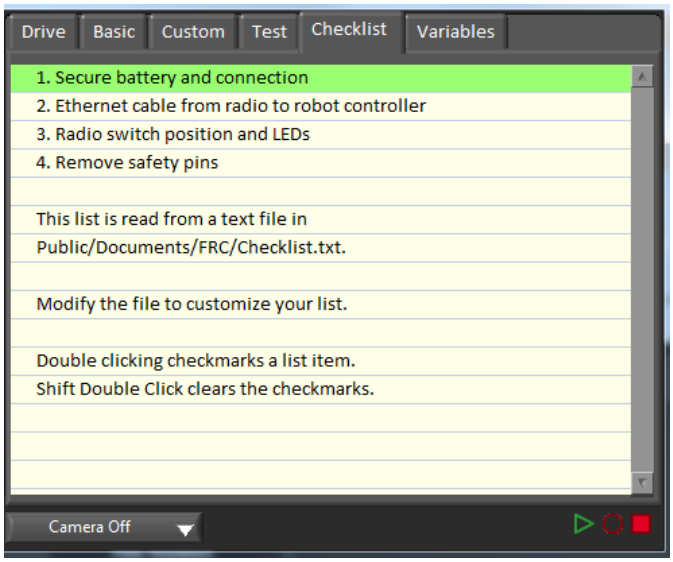
L'onglet Test est utilisé en mode Test pour les équipes utilisant LabVIEW (les équipes Java et C++ doivent utiliser SmartDashboard ou Shuffleboard lors de l'utilisation du mode Test). Pour de nombreux éléments dans les bibliothèques logicielles, les informations d'entrée/sortie seront remplies ici automatiquement. Tous les éléments qui ont ** à côté d'eux sont des sorties qui peuvent être contrôlées par le dashboard. Pour contrôler une sortie, cliquez dessus pour la sélectionner, faites glisser le curseur pour définir la valeur puis appuyez et maintenez le bouton vert pour activer la sortie. Dès que le bouton vert est libéré, la sortie sera désactivée. Cet onglet peut également être utilisé pour exécuter et surveiller les tests sur le robot. Un test d'exemple est fourni dans l »infrastructure logicielle LabVIEW. La sélection de ce test à partir de la liste déroulante affichera l'état du test à la place du curseur et activera les contrôles.

Commands



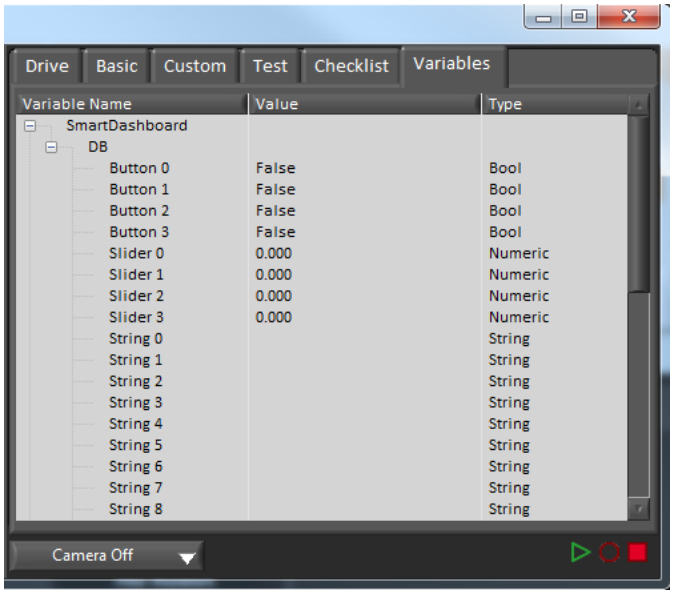
L'onglet Commands peut être utilisé avec le robot en mode Test pour voir quelles commandes sont en cours d'exécution et pour exécuter manuellement les commandes à des fins de test.

Checklist



L'onglet Checklist peut être utilisé par les équipes pour créer une liste de tâches à effectuer avant ou entre les matchs. Les instructions pour l'utilisation de l'onglet Checklist sont préalablement remplies dans le fichier de liste de contrôle par défaut.

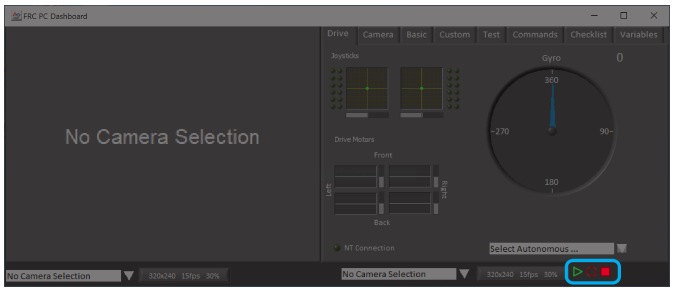
Variables



L'onglet Variables du volet gauche affiche toutes les variables NetworkTables dans une arborescence. Le nom de la variable (clé), la valeur et le type de données sont affichés pour chaque variable. Des informations sur l'utilisation de la bande passante NetworkTables sont

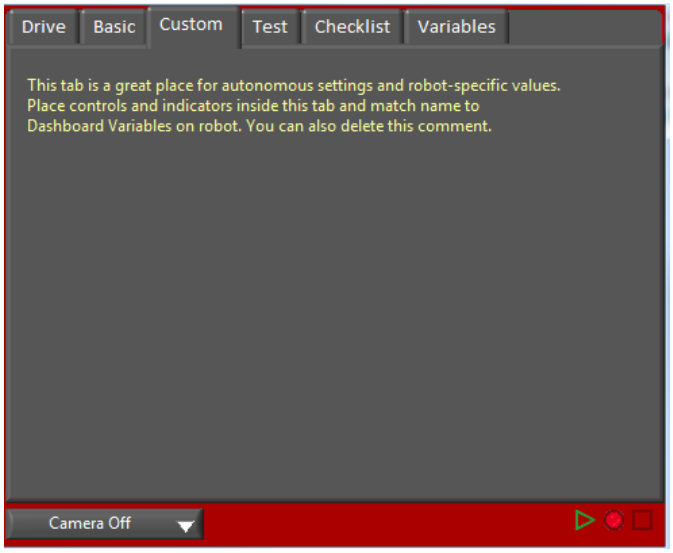
également affichées dans cet onglet. Les entrées seront affichées avec des losanges noirs si elles ne sont pas actuellement synchronisées avec le robot.

Enregistrement/Lecture



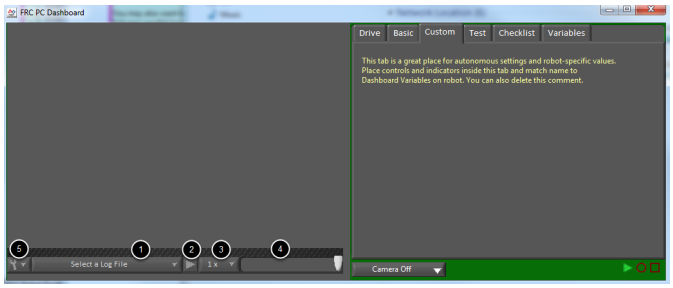
Le tableau de bord LabVIEW comprend une fonction d'enregistrement / lecture qui vous permet d'enregistrer des données vidéo et NetworkTables (telles que l'état des indicateurs de votre tableau de bord) et de les lire plus tard.

Enregistrement



Pour commencer l'enregistrement, cliquez sur le bouton Record de forme circulaire et de couleur rouge. L'arrière-plan du panneau droit deviendra rouge pour indiquer que vous enregistrez. Pour arrêter l'enregistrement, appuyez sur le bouton Stop de forme carrée et de couleur rouge.

Lecture



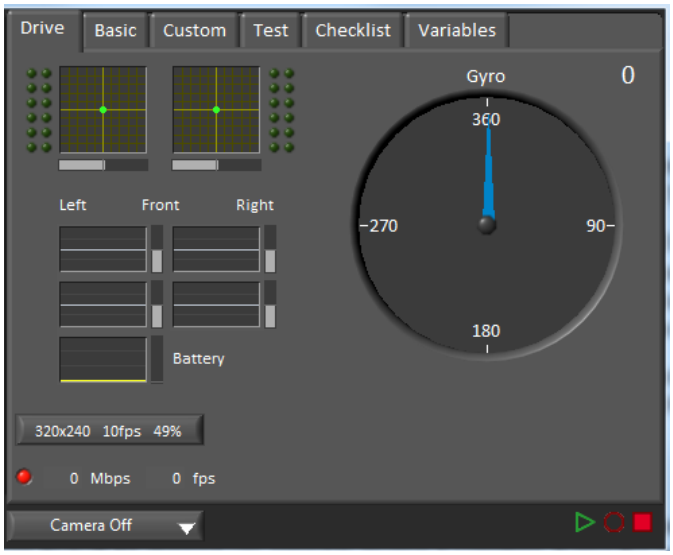
Pour lire un enregistrement, cliquez sur le bouton de lecture qui se présente sous la forme d'un triangle vert. L'arrière-plan du panneau droit commencera à clignoter en vert et les contrôles de lecture apparaîtront au bas du panneau de la caméra.

1. Sélecteur de fichiers - Le menu déroulant vous permet de sélectionner un fichier journal à lire. Les fichiers journaux sont nommés en utilisant la date et l'heure et le menu déroulant indiquera également la longueur du fichier. À la sélection d'un fichier journal, celui-ci commencera immédiatement à jouer.
2. Bouton Lecture/Pause - Ce bouton vous permet de mettre en pause et de reprendre la lecture du fichier journal.
3. Vitesse de lecture - Ce menu déroulant vous permet d'ajuster la vitesse de lecture de 1/10 de la vitesse à 10x la vitesse, la valeur par défaut correspond à la lecture en temps réel (1x)
4. Curseur de contrôle du temps - Ce curseur vous permet d'avancer rapidement ou de reculer à travers le fichier journal en cliquant sur un emplacement désiré ou en faisant glisser le curseur.
5. Paramètres - Avec un fichier journal sélectionné, ce menu déroulant vous permet de renommer ou de supprimer un fichier ou d'ouvrir le dossier contenant les journaux dans Windows Explorer (Typiquement `C:\Users\Public\Documents\FRC\Log Files\Dashboard`)

11.6.2 Using the LabVIEW Dashboard with C++, Java, or Python Code

The default LabVIEW Dashboard utilizes *NetworkTables* to pass values and is therefore compatible with C++, Java, and Python robot programs. This article covers the keys and value ranges to use to work with the Dashboard.

Onglet Drive



Le menu déroulant *Select Autonomous ...* peut être utilisé pour afficher les routines autonomes disponibles et en choisir une pour le match.

JAVA

```
SmartDashboard.putStringArray("Auto List", {"Drive Forwards", "Drive Backwards",
↪"Shoot"});

// At the beginning of auto
String autoName = SmartDashboard.getString("Auto Selector", "Drive Forwards") // This␣
↪would make "Drive Forwards the default auto
switch(autoName) {
   case "Drive Forwards":
   // auto here
   case "Drive Backwards":
   // auto here
   case "Shoot":
   // auto here
}
```

C++

```
frc::SmartDashboard::PutStringArray("Auto List", {"Drive Forwards", "Drive Backwards",
↪ "Shoot"});

// At the beginning of auto
String autoName = SmartDashboard.GetString("Auto Selector", "Drive Forwards") // This␣
↪would make "Drive Forwards the default auto
switch(autoName) {
   case "Drive Forwards":
   // auto here
   case "Drive Backwards":
```

(suite sur la page suivante)

(suite de la page précédente)

```
   // auto here
   case "Shoot":
   // auto here
}
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putStringArray("Auto List", ["Drive Forwards", "Drive Backwards",
↪"Shoot"])

# At the beginning of auto
autoName = SmartDashboard.getString("Auto Selector", "Drive Forwards") # This would␣
↪make "Drive Forwards the default auto
match autoName:
   case "Drive Forwards":
      # auto here
   case "Drive Backwards":
      # auto here
   case "Shoot":
      # auto here
```

L'envoi à l'entrée NetworkTables du « Gyro » remplira le gyro.

JAVA

```
SmartDashboard.putNumber("Gyro", drivetrain.getHeading());
```

C++

```
frc::SmartDashboard::PutNumber("Gyro", Drivetrain.GetHeading());
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putNumber("Gyro", self.drivetrain.getHeading())
```

Il y a quatre sorties qui indiquent la puissance du moteur délivré à la boîte de vitesses. Ceci est configuré pour 2 moteurs par côté et une transmission de type Tank. Cela se fait en paramétrant « RobotDrive Motors » comme dans l'exemple ci-dessous.

JAVA

```
SmartDashboard.putNumberArray("RobotDrive Motors", {drivetrain.getLeftFront(),␣
↪drivetrain.getRightFront(), drivetrain.getLeftBack(), drivetrain.getRightBack()});
```

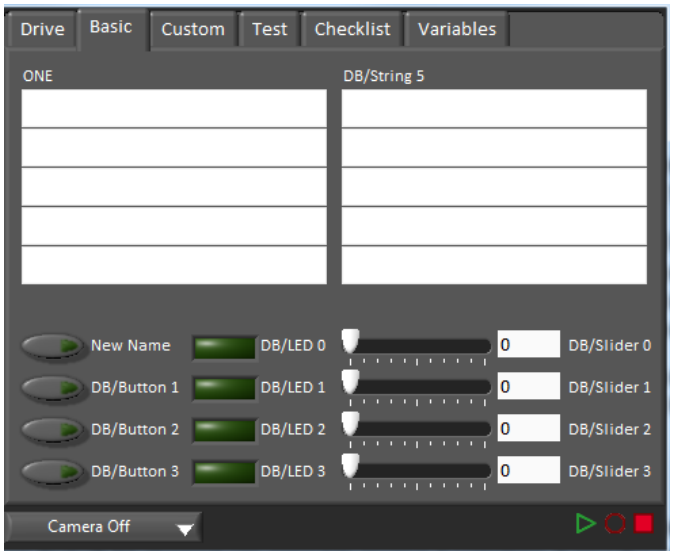
C++

```
frc::SmartDashboard::PutNumberArray("Gyro", {drivetrain.GetLeftFront(), drivetrain.
↪GetRightFront(), drivetrain.GetLeftBack(), drivetrain.GetRightBack()});
```

PYTHON

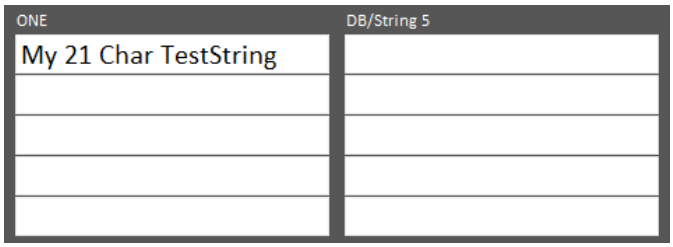
```
from wpilib import SmartDashboard

SmartDashboard.putNumberArray("RobotDrive Motors", [self.drivetrain.getLeftFront(),␣
↪self.drivetrain.getRightFront(), self.drivetrain.getLeftBack(), self.drivetrain.
↪getRightBack()])
```

Onglet Basic



L'onglet Basic utilise un certain nombre de clés dans le sous-tableau « DB » pour envoyer/recevoir des données du Dashboard. Les DEL sont des sorties seulement, les autres champs sont tous bi-directionnels (envoyer ou recevoir).

Les chaînes de caractères



Les chaînes de caractères sont étiquetées de haut en bas, de gauche à droite de « DB/String 0 » à « DB/String 9 ». Chaque champ de chaîne de caractères peut afficher au moins 21 caractères (le nombre exact dépend des caractères). Pour écrire dans ces chaînes :

JAVA

```
SmartDashboard.putString("DB/String 0", "My 21 Char TestString");
```

C++

```
frc::SmartDashboard::PutString("DB/String 0", "My 21 Char TestString");
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putString("DB/String 0", "My 21 Char TestString")
```

Pour lire les données des chaînes de caractères saisies sur le Dashboard :

JAVA

```
String dashData = SmartDashboard.getString("DB/String 0", "myDefaultData");
```

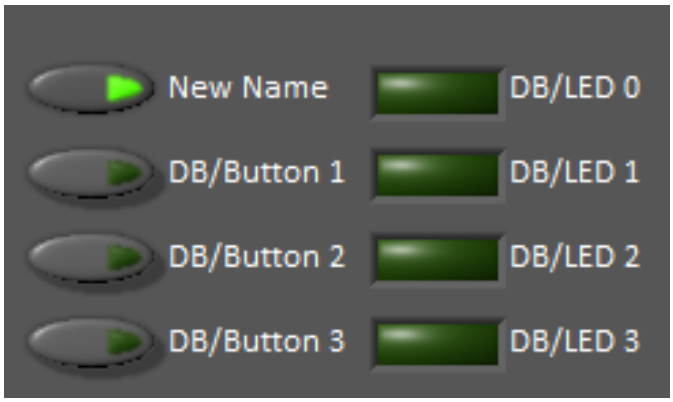
C++

```
std::string dashData = frc::SmartDashboard::GetString("DB/String 0", "myDefaultData");
```

PYTHON

```
from wpilib import SmartDashboard

dashData = SmartDashboard.getString("DB/String 0", "myDefaultData")
```

Boutons et DELS



Les boutons et les DEL sont des valeurs booléennes et sont étiquetés de haut en bas de « DB/Button 0 » à « DB/Button 3 » et « DB/LED 0 » à « DB/LED 3 ». Les boutons sont bi-directionnels, les DEL ne peuvent être écrites qu'à partir du robot et lues à partir du Dashboard. Pour écrire aux boutons ou dels :

JAVA

```
SmartDashboard.putBoolean("DB/Button 0", true);
```

C++

```
frc::SmartDashboard::PutBoolean("DB/Button 0", true);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putBoolean("DB/Button 0", true)
```

Pour lire à partir des boutons : (la valeur par défaut est false)

JAVA

```
boolean buttonValue = SmartDashboard.getBoolean("DB/Button 0", false);
```

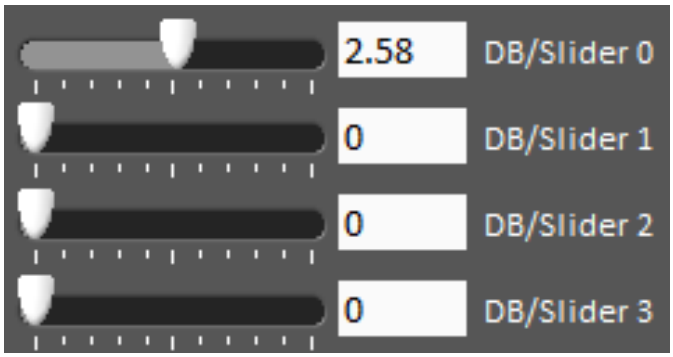
C++

```
bool buttonValue = frc::SmartDashboard::GetBoolean("DB/Button 0", false);
```

PYTHON

```
from wpilib import SmartDashboard

buttonValue = SmartDashboard.getBoolean("DB/Button 0", false)
```

Curseurs



Les curseurs sont des contrôles/indicateurs analogiques bi-directionnels (double) dont la plage varie de 0 à 5. Pour écrire des valeurs à ces indicateurs :

JAVA

```
SmartDashboard.putNumber("DB/Slider 0", 2.58);
```

C++

```
frc::SmartDashboard::PutNumber("DB/Slider 0", 2.58);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putNumber("DB/Slider 0", 2.58)
```

Pour lire des valeurs à partir du Dashboard dans le programme robot : (valeur par défaut de 0.0)

JAVA

```
double dashData = SmartDashboard.getNumber("DB/Slider 0", 0.0);
```

C++

```
double dashData = frc::SmartDashboard::GetNumber("DB/Slider 0", 0.0);
```

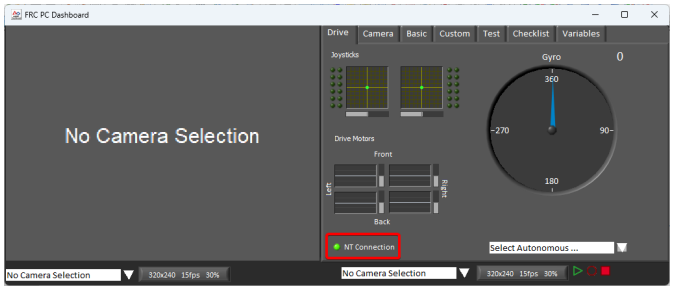
PYTHON

```
from wpilib import SmartDashboard

dashData = SmartDashboard.getNumber("DB/Slider 0", 0.0)
```

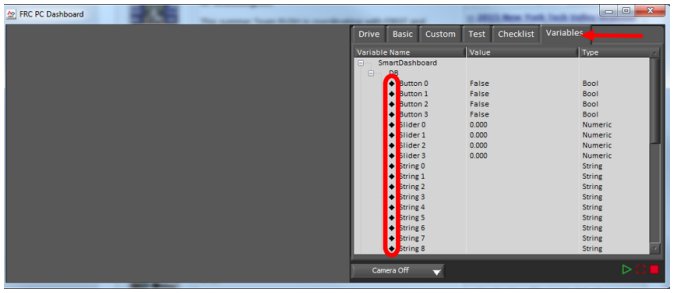
11.6.3 Troubleshooting Dashboard Connectivity

This document will help explain how to recognize if the Dashboard is not connected to your robot, steps to troubleshoot this condition and a code modification you can make.

Recognizing LabVIEW Dashboard Connectivity

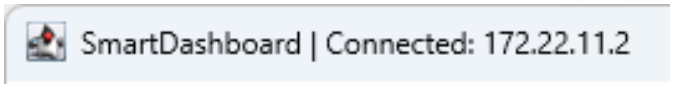


The LabVIEW Dashboard has a NetworkTables Connection indicator on the front panel.

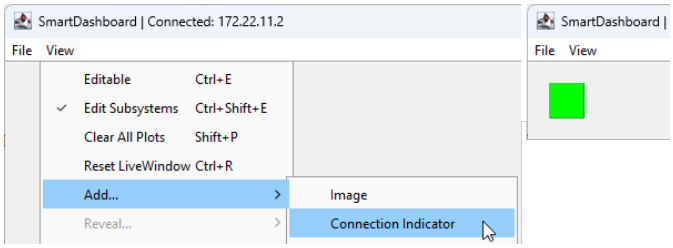


On the Variables tab of the Dashboard, the variables are shown with a black diamond when they are not synced with the robot. Once the Dashboard connects to the robot and these variables are synced, the diamond will disappear.

Recognizing SmartDashboard Connectivity



SmartDashboard indicates if it is connected or not in the title bar. It shows the IP address it is connected to. See *this page* for more on configuring the connection.

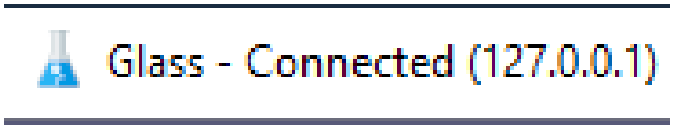


For more visibility, you can also add a Connection Indicator widget. The connection indicator can be moved or re-sized if the Editable checkbox is checked.

Recognizing Shuffleboard Connectivity

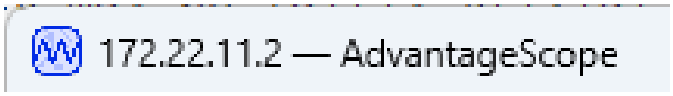
Shuffleboard indicates if it is connected or not in the bottom right corner of the application as shown in the image above. See *page* for more on configuring the connection.

Recognizing Glass Connectivity



Glass indicates if it is connected or not in the title bar. It shows the IP address it is connected to. See this *page* for more on configuring the connection.

Recognizing AdvantageScope Connectivity



AdvantageScope indicates if it is connected or not in the title bar. It shows the IP address it is connected to, or else the IP address it is attempting to connect to. See the AdvantageScope Documentation for more on configuring the connection.

Troubleshooting Connectivity

If the Dashboard does not connect to the Robot (after the Driver Station has connected to the robot) the recommended troubleshooting steps are :

1. Restart the Dashboard (there is no need to restart the Driver Station software)
2. If that doesn't work, restart the Robot Code using the Restart Robot Code button on the Diagnostics tab of the Driver Station
3. If it still doesn't connect, verify that the Team Number / Server is set properly in the Dashboard and that your Robot Code writes a value during initialization or disabled

12 Télémétrie

12.1 Telemetry : Recording and Sending Real-Time Data

Recording and viewing *telemetry* data is a crucial part of the engineering process - accurate telemetry data helps you tune your robot to perform optimally, and is indispensable for debugging your robot when it fails to perform as expected.

By default, no telemetry data is recorded (saved) on the robot. However, recording data on the robot can provide benefits over recording on a dashboard, namely that more data can be recorded (there are no bandwidth limitations), and all the recorded data can be very accurately timestamped. WPILib has integrated support for on-robot recording of telemetry data via the `DataLogManager` and `DataLog` classes and provides a tool for downloading data log files and converting them to CSV.

Note : In addition to on-robot recording of telemetry data, teams can record their telemetry data on their driver station computer with *Shuffleboard recordings*.

12.1.1 Adding Telemetry to Robot Code

WPILib supports several different ways to record and send telemetry data from robot code.

Au niveau le plus élémentaire, le *Riolog* gère l'affichage des instructions d'impression en provenance du code du robot. Cette fonctionnalité est utile pour le débogage à la volée d'un code présentant des erreurs, mais ne permet pas la mise à l'échelle car les interfaces de console ne conviennent pas aux flux importants de données.

WPILib prend en charge plusieurs *dashboards* qui permettent aux utilisateurs d'envoyer plus facilement de nombreuses données de télémesure vers l'ordinateur de pilotage. Tous les dashboards de la WPILib communiquent par l'intermédiare du protocole *NetworkTables*, et ils sont donc *dans une certaine mesure* interopérables (les données de télémesure enregistrée avec un dashboard seront visibles sur les autres, mais les widgets/formatages spécifiques ne seront généralement pas compatibles). NetworkTables (et donc tous les dashboards de la WPILib) prennent actuellement en charge les types de données suivants :

- `boolean`
- `boolean[]`
- `double`

- `double[]`
- `string`
- `string[]`
- `byte[]`

Telemetry data can be sent to a WPILib dashboard using an associated WPILib method (for more details, see the documentation for the individual dashboard in question), or by *directly publishing to NetworkTables*.

While NetworkTables does not yet support serialization of complex data types (this is tentatively scheduled for 2024), *mutable* types from user code can be easily extended to interface directly with WPILib dashboards via the `Sendable` interface, whose usage is described in the next article.

12.2 Télémesure du robot avec l'interface Sendable

Alors que les API des dashboards de la WPILib permettent aux utilisateurs d'envoyer facilement de petites quantités de données de leur code robot vers le dashboard, il est souvent fastidieux d'écrire manuellement du code pour publier des valeurs de télémesure à partir de la logique opérationnelle du code robot.

A cleaner approach is to leverage the existing object-oriented structure of user code to mark important data fields for telemetry logging in a *declarative programming* style. The WPILib framework can then handle the tedious/tricky part of correctly reading from (and, potentially, *writing to*) those fields for you, greatly reducing the total amount of code the user has to write and improving readability.

WPILib fournit cette fonctionnalité par l'intermédiaire de l'interface `Sendable`. Les classes qui implémentent `Sendable` sont capables d'enregistrer des données issues des détecteurs de valeurs qui envoient automatiquement des données au dashboard et, dans certains cas, de recevoir des valeurs en retour. Ces classes peuvent être envoyées de manière déclarative à n'importe lequel des dashboards de la WPILib (comme on le ferait pour un champ de données ordinaire), éliminant ainsi la nécessité pour les équipes d'écrire leur propre code pour envoyer/sonder pour les mises à jour.

12.2.1 Qu'est-ce qu'un Sendable ?

`Sendable` (Java, C++, `Python`) is an interface provided by WPILib to facilitate robot telemetry. Classes that implement `Sendable` can declaratively send their state to the dashboard - once declared, WPILib will automatically send the telemetry values every robot loop. This removes the need for teams to handle the iteration-to-iteration logic of sending and receiving values from the dashboard, and also allows teams to separate their telemetry code from their robot logic.

De nombreuses classes de la WPILib (telles que *Commands*) implémentent déjà l'interface `Sendable`, et peuvent donc être acheminées au dashboard sans aucune modification de la part de l'utilisateur. Les utilisateurs peuvent également étendre facilement leurs propres classes pour implémenter l'interface `Sendable`.

The `Sendable` interface contains only one method : `initSendable`. Implementing classes override this method to perform the binding of in-code data values to structured *JSON* data, which is then automatically sent to the robot dashboard via NetworkTables. Implementation of the `Sendable` interface is discussed in the *next article*.

12.2.2 Acheminement d'un objet Sendable au dashboard

Note : Contrairement aux types de données primitives, les objet de type Sendable sont automatiquement tenus à jour dans le dashboard par la WPILib, sans autre code utilisateur - « configurez-le et oubliez-le ». En conséquence, ils doivent généralement être envoyés au dashboard à travers un bloc d'initialisation ou un constructeur, *pas* dans une méthode périodique.

Pour envoyer un objet `Sendable` au dashboard, utilisez simplement la méthode `putData` du dashboard. Par exemple, une classe « arm » qui utilise un *contrôleur PID* peut automatiquement consigner la télémesure du contrôleur en appelant ce qui suit dans son constructeur :

JAVA

```
SmartDashboard.putData("Arm PID", armPIDController);
```

C++

```
frc::SmartDashboard::PutData("Arm PID", &armPIDController);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData("Arm PID", armPIDController)
```

De plus, certaines classes `Sendable` lient les setters ou métodes de réglage aux valeurs des données envoyées *à partir du dashboard vers le robot*, permettant ainsi le réglage à distance des paramètres du robot.

12.3 On-Robot Telemetry Recording Into Data Logs

By default, no telemetry data is recorded (saved) on the robot. The `DataLogManager` class provides a convenient wrapper around the lower-level `DataLog` class for on-robot recording of telemetry data into data logs. The WPILib data logs are binary for size and speed reasons. In general, the data log facilities provided by WPILib have minimal overhead to robot code, as all file I/O is performed on a separate thread–the log operation consists of mainly a mutex acquisition and copying the data.

12.3.1 Structure of Data Logs

Similar to NetworkTables, data logs have the concept of entries with string identifiers (keys) with a specified data type. Unlike NetworkTables, the data type cannot be changed after the entry is created, and entries also have metadata–an arbitrary (but typically JSON) string that can be used to convey additional information about the entry such as the data source or data schema. Also unlike NetworkTables, data log operation is unidirectional–the `DataLog` class can only write data logs (it does not support read-back of written values) and the `DataLogReader` class can only read data logs (it does not support changing values in the data log).

Data logs consist of a series of timestamped records. Control records allow starting, finishing, or changing the metadata of entries, and data records record data value changes. Timestamps are stored in integer microseconds ; when running on the RoboRIO, the FPGA timestamp is used (the same timestamp returned by `Timer.getFPGATimestamp()`).

Note : For more information on the details of the data log file format, see the WPILib Data Log File Format Specification.

12.3.2 Standard Data Logging using DataLogManager

The `DataLogManager` class (Java, C++, `Python`) provides a centralized data log that provides automatic data log file management. It automatically cleans up old files when disk space is low and renames the file based either on current date/time or (if available) competition match number. The data file will be saved to a USB flash drive in a folder called `logs` if one is attached, or to `/home/lvuser/logs` otherwise.

Note : USB flash drives need to be formatted as FAT32 to work with the roboRIO. NTFS or exFAT formatted drives will not work.

Log files are initially named `FRC_TBD_{random}.wpilog` until the DS connects. After the DS connects, the log file is renamed to `FRC_yyyyMMdd_HHmmss.wpilog` (where the date/time is UTC). If the *FMS* is connected and provides a match number, the log file is renamed to `FRC_yyyyMMdd_HHmmss_{event}_{match}.wpilog`.

On startup, all existing log files where a DS has not been connected will be deleted. If there is less than 50 MB of free space on the target storage, `FRC_` log files are deleted (oldest to newest) until there is 50 MB free OR there are 10 files remaining.

The most basic usage of DataLogManager only requires a single line of code (typically this would be called from `robotInit`). This will record all NetworkTables changes to the data log.

JAVA

```java
import edu.wpi.first.wpilibj.DataLogManager;

// Starts recording to data log
DataLogManager.start();
```

C++

```cpp
#include "frc/DataLogManager.h"

// Starts recording to data log
frc::DataLogManager::Start();
```

PYTHON

```python
from wpilib import DataLogManager

DataLogManager.start()
```

DataLogManager provides a convenience function (`DataLogManager.log()`) for logging of text messages to the `messages` entry in the data log. The message is also printed to standard output, so this can be a replacement for `System.out.println()`.

DataLogManager also records the current roboRIO system time (in UTC) to the data log every ~5 seconds to the `systemTime` entry in the data log. This can be used to (roughly) synchronize the data log with other records such as DS logs or match video.

For custom logging, the managed `DataLog` can be accessed via `DataLogManager.getLog()`.

Logging Joystick Data

DataLogManager by default does not record joystick data. The `DriverStation` class provides support for logging of DS control and joystick data via the `startDataLog()` function :

JAVA

```java
import edu.wpi.first.wpilibj.DataLogManager;
import edu.wpi.first.wpilibj.DriverStation;

// Starts recording to data log
DataLogManager.start();

// Record both DS control and joystick data
DriverStation.startDataLog(DataLogManager.getLog());

// (alternatively) Record only DS control data
DriverStation.startDataLog(DataLogManager.getLog(), false);
```

C++

```cpp
#include "frc/DataLogManager.h"
#include "frc/DriverStation.h"

// Starts recording to data log
frc::DataLogManager::Start();

// Record both DS control and joystick data
DriverStation::StartDataLog(DataLogManager::GetLog());

// (alternatively) Record only DS control data
DriverStation::StartDataLog(DataLogManager::GetLog(), false);
```

PYTHON

```python
from wpilib import DataLogManager, DriverStation

# Starts recording to data log
DataLogManager.start()

# Record both DS control and joystick data
DriverStation.startDataLog(DataLogManager.getLog())

# (alternatively) Record only DS control data
DriverStation.startDataLog(DataLogManager.getLog(), False)
```

12.3.3 Custom Data Logging using DataLog

The `DataLog` class (Java, C++, Python) and its associated LogEntry classes (e.g. `BooleanLogEntry`, `DoubleLogEntry`, etc) provides low-level access for writing data logs.

Note : Unlike NetworkTables, there is no change checking performed. **Every** call to a `LogEntry.append()` function will result in a record being written to the data log. Checking for changes and only appending to the log when necessary is the responsibility of the caller.

The LogEntry classes can be used in conjunction with DataLogManager to record values only to a data log and not to NetworkTables :

JAVA

```java
import edu.wpi.first.util.datalog.BooleanLogEntry;
import edu.wpi.first.util.datalog.DataLog;
import edu.wpi.first.util.datalog.DoubleLogEntry;
import edu.wpi.first.util.datalog.StringLogEntry;
import edu.wpi.first.wpilibj.DataLogManager;

BooleanLogEntry myBooleanLog;
DoubleLogEntry myDoubleLog;
```

(suite sur la page suivante)

(suite de la page précédente)

```
StringLogEntry myStringLog;

public void robotInit() {
  // Starts recording to data log
  DataLogManager.start();

  // Set up custom log entries
  DataLog log = DataLogManager.getLog();
  myBooleanLog = new BooleanLogEntry(log, "/my/boolean");
  myDoubleLog = new DoubleLogEntry(log, "/my/double");
  myStringLog = new StringLogEntry(log, "/my/string");
}

public void teleopPeriodic() {
  if (...) {
    // Only log when necessary
    myBooleanLog.append(true);
    myDoubleLog.append(3.5);
    myStringLog.append("wow!");
  }
}
```

C++

```
#include "frc/DataLogManager.h"
#include "wpi/DataLog.h"

wpi::log::BooleanLogEntry myBooleanLog;
wpi::log::DoubleLogEntry myDoubleLog;
wpi::log::StringLogEntry myStringLog;

void RobotInit() {
  // Starts recording to data log
  frc::DataLogManager::Start();

  // Set up custom log entries
  wpi::log::DataLog& log = frc::DataLogManager::GetLog();
  myBooleanLog = wpi::Log::BooleanLogEntry(log, "/my/boolean");
  myDoubleLog = wpi::log::DoubleLogEntry(log, "/my/double");
  myStringLog = wpi::log::StringLogEntry(log, "/my/string");
}

void TeleopPeriodic() {
  if (...) {
    // Only log when necessary
    myBooleanLog.Append(true);
    myDoubleLog.Append(3.5);
    myStringLog.Append("wow!");
  }
}
```

PYTHON

```python
from wpilib import DataLogManager, TimedRobot
from wpiutil.log import (
    DataLog,
    BooleanLogEntry,
    DoubleLogEntry,
    StringLogEntry,
)

class MyRobot(TimedRobot):
    def robotInit(self):
        # Starts recording to data log
        DataLogManager.start()

        # Set up custom log entries
        log = DataLogManager.getLog()
        self.myBooleanLog = BooleanLogEntry(log, "/my/boolean")
        self.myDoubleLog = DoubleLogEntry(log, "/my/double")
        self.myStringLog = StringLogEntry(log, "/my/string")

    def teleopPeriodic(self):
        if ...:
            # Only log when necessary
            self.myBooleanLog.append(True)
            self.myDoubleLog.append(3.5)
            self.myStringLog.append("wow!")
```

12.4 Downloading & Processing Data Logs

Data logs can be processed and viewed offline by AdvantageScope, the DataLogTool, or custom utilities. If data log files are being stored to the roboRIO integrated flash memory instead of a removable USB flash drive, it's important to periodically download and delete data logs to avoid the storage from filling up.

12.4.1 Managing Data Logs with AdvantageScope

AdvantageScope is an analysis tool that supports downloading, visualizing, and exporting data logs. See the relevant sections of the documentation for more details :

— Downloading log files
— Exporting to new formats

12.4.2 Managing Data Logs with the DataLogTool

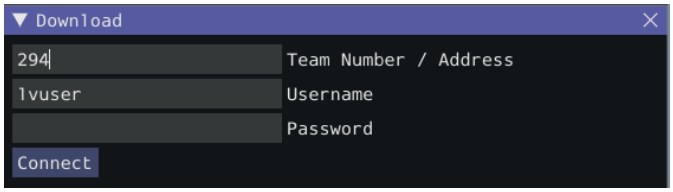
The DataLogTool desktop application integrates a SFTP client for downloading data log files from a network device (e.g. roboRIO or coprocessor) to the local computer.

This process consists of four steps :

1. Connect to roboRIO or coprocessor
2. Navigate to remote directory and select what files to download
3. Select download folder
4. Download files and optionally delete remote files after downloading

Connecting to RoboRIO

Note : The downloader uses SSH, so it will not be able to connect wirelessly if the radio firewall is enabled (e.g. when the robot is on the competition field).



Either a team number, IP address, or hostname can be entered into the *Team Number / Address* field. This field specifies the remote host to connect to. If a team number is entered, `roborio-TEAM-frc.local` is used as the connection address.

The remote username and password are also entered here. For the roboRIO, the username should be `lvuser` with a blank password.

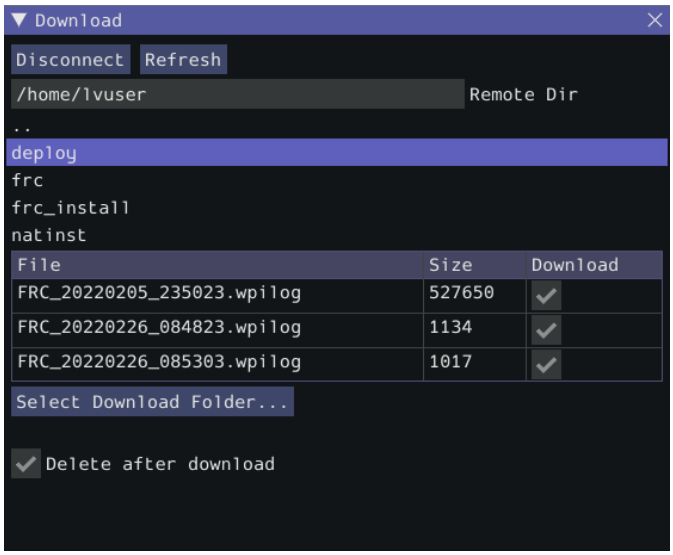
The tool also supports connecting to network devices other than the roboRIO, such as coprocessors, as long as the device supports SFTP password-based authentication.

Click *Connect* to connect to the remote device. This will attempt to connect to the device. The connection attempt can be aborted at any time by clicking *Disconnect*. If the application is unable to connect to the remote device, an error will be displayed above the *Team Number / Address* field and a new connection can be attempted.

Downloading Files

After the connection is successfully established, a simplified file browser will be displayed. This is used to navigate the remote filesystem and select which files to download. The first text box shows the current directory. A specific directory can be navigated to by typing it in this text box and pressing Enter. Alternatively, directory navigation can be performed by clicking on one of the directories that are listed below the remote dir textbox. Following the list of directories is a table of files. Only files with a `.wpilog` extension are shown, so the table will be empty if there are no log files in the current directory. The checkbox next to each data log file indicates whether the file should be downloaded.

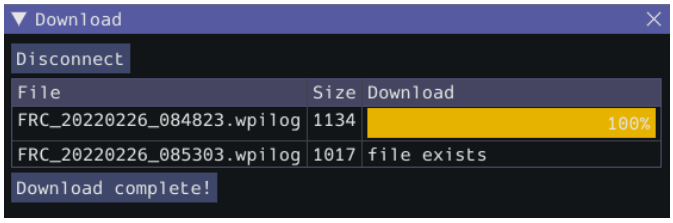
Note : On the roboRIO, log files are typically saved to either `/home/lvuser/logs` or `/u/logs` (USB stick location).



Click *Select Download Folder...* to bring up a file browser for the local computer.

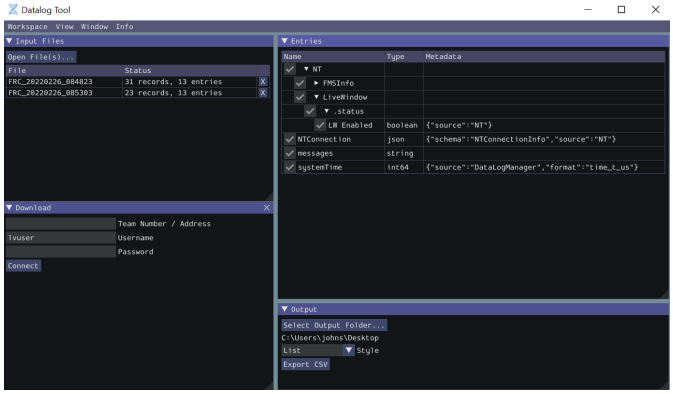
If you want to delete the files from the remote device after they are downloaded, check the *Delete after download* checkbox.

Once a download folder is selected, *Download* will appear. After clicking this button, the display will change to a download progress display. Any errors will be shown next to each file. Click *Download complete!* to return to the file browser.



Converting Data Logs to CSV

As data logs are binary files, the DataLogTool desktop application provides functionality to convert data logs into CSV files for further processing or analysis. Multiple data logs may be simultaneously loaded into the tool for batch processing, and partial data exports can be performed by selecting only the data that is desired to be output.

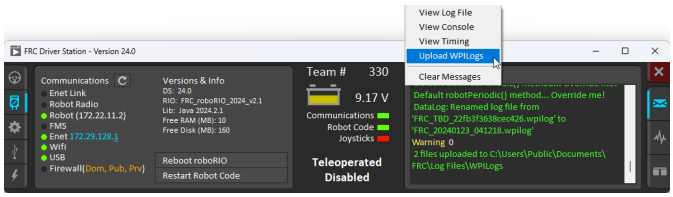


The conversion process is started by opening data log files in the « Input Files » window. Files are opened by clicking *Open File(s)...*. Summary status on each file (e.g. number of records and entries) is displayed. Clicking *X* in the table row closes the file.

After at least one file is loaded, the « Entries » window displays a tree view of the entries (this can be changed to a flat view by right clicking on the « Entries » window title bar and unchecking *Tree View*). Individual entries or entire subtrees can be checked or unchecked to indicate whether they should be included in the export. The data type information and initial metadata for each entry is also shown in the table. As the « Entries » view shows a merged view of all entries across all input files, if more than one input file is open, hovering over an entry's name will highlight what input files contain that entry.

The output window is used to specify the output folder (via *Select Output Folder...*) as well as the output style (list or table). The list output style outputs a CSV file with 3 columns (timestamp, entry name, and value) and a row for every value change (for every exported entry). The table output style outputs a CSV file with a timestamp column and a column for every exported entry; a row is output for every value change (for every exported entry), but the value is placed in the correct column for that entry. Clicking *Export CSV* will create a `.csv` file in the output folder corresponding to each input file.

12.4.3 Managing Data Logs with the Driver Station



The Driver Station software can download WPILogs. Click on the gear icon and select *Upload WPILogs*. The logs in `/home/lvuser/logs` or `/u/logs` will be downloaded automatically to `C:\Users\Public\Documents\FRC\Log Files\WPILogs`

12.4.4 Custom Processing of Data Logs

For more advanced processing of data logs (e.g. for processing of binary values that can't be converted to CSV), WPILib provides a `DataLogReader` class for reading data logs in Java, C++, or `Python`. There is also a pure python datalog reader (datalog.py). For other languages, the data log format is also documented.

DataLogReader provides a low-level view of a data log, in that it supports iterating over a data log's control and data records and decoding of common data types, but does not provide any higher level abstractions such as a NetworkTables-like map of entries. The printlog example in Java and C++ (and the Python `datalog.py`) demonstrates basic usage.

12.5 Conception de vos propres classes Sendable

Since the `Sendable` interface only has one method, writing your own classes that implement `Sendable` (and thus automatically log values to and/or consume values from the dashboard) is extremely easy : just provide an implementation for the overridable `initSendable` method, in which setters and getters for your class's fields are declaratively bound to key values (their display names on the dashboard).

Voici, par exemple, voici l'implémentation de `initSendable` de l'interface à `BangBangController` de la WPILib :

JAVA

```
151   @Override
152   public void initSendable(SendableBuilder builder) {
153     builder.setSmartDashboardType("BangBangController");
154     builder.addDoubleProperty("tolerance", this::getTolerance, this::setTolerance);
155     builder.addDoubleProperty("setpoint", this::getSetpoint, this::setSetpoint);
156     builder.addDoubleProperty("measurement", this::getMeasurement, null);
157     builder.addDoubleProperty("error", this::getError, null);
158     builder.addBooleanProperty("atSetpoint", this::atSetpoint, null);
159   }
```

C++

```
55  void BangBangController::InitSendable(wpi::SendableBuilder& builder) {
56    builder.SetSmartDashboardType("BangBangController");
57    builder.AddDoubleProperty(
58        "tolerance", [this] { return GetTolerance(); },
59        [this](double tolerance) { SetTolerance(tolerance); });
60    builder.AddDoubleProperty(
61        "setpoint", [this] { return GetSetpoint(); },
62        [this](double setpoint) { SetSetpoint(setpoint); });
63    builder.AddDoubleProperty(
64        "measurement", [this] { return GetMeasurement(); }, nullptr);
65    builder.AddDoubleProperty(
66        "error", [this] { return GetError(); }, nullptr);
67    builder.AddBooleanProperty(
68        "atSetpoint", [this] { return AtSetpoint(); }, nullptr);
69  }
```

Pour permettre la mise à jour automatique des valeurs par la WPILib « en arrière-plan », les noms de données `Sendable` sont liés à des méthodes getter et setter plutôt qu'à des valeurs de données spécifiques. Si un champ que vous souhaitez consigner n'a pas de setters et de getters définis, ils peuvent être définis inline à l'aide d'une expression lambda.

12.5.1 La classe SendableBuilder

As seen above, the `initSendable` method takes a single parameter, `builder`, of type `SendableBuilder` (Java, C++, Python). This builder exposes methods that allow binding of getters and setters to dashboard names, as well as methods for safely ensuring that values consumed *from* the dashboard do not cause unsafe robot behavior.

Liaison des données à l'aide des méthodes addProperty

Comme dans tout code d'un dashboard de la WPILib, les champs `Sendable` sont en fin de compte transmis via les *NetworkTables*, et donc les méthodes de liaison de données fournies par `SendableBuilder` correspondent aux types de données NetworkTables pris en charge :

- `boolean` : `addBooleanProperty`
- `boolean[]` : `addBooleanArrayProperty`
- `double` : `addDoubleProperty`
- `double[]` : `addDoubleArrayProperty`
- `string` : `addStringProperty`
- `string[]` : `addStringArrayProperty`
- `byte[]` : `addRawProperty`

Assurer la sécurité avec setSafeState et setActuator

Étant donné que `Sendable` permet aux utilisateurs de consommer des valeurs arbitraires lues du dashboard, il est possible pour les utilisateurs de diriger les commandes du dashboard directement vers les actionneurs du robot. Cette opération est extrêmement dangereuse si elle n'est pas faite avec soin ; Les dashboards ne sont pas une interface particulièrement efficace pour contrôler les mouvements du robot, et les utilisateurs ne s'attendent généralement pas à ce que le robot se déplace en réponse à un changement effectué à partir de valeur sur le dashboard.

Pour aider les utilisateurs à s'assurer d'une interfaçage sécuritaire avec les valeurs du dashboard, `SendableBuilder` expose une méthode `setSafeState`, qui est appelée pour placer tout mécanisme `Sendable` qui s'active en fonction d'une entrée du dashboard dans un état sécuritaire. Toute implémentation `Sendable` potentiellement dangereuse programmée par un utilisateur doit appeler `setSafeState` avec une implémentation d'état sécuritaire appropriée. Voici, par exemple, l'implémentation de la classe WPILib `PWMMotorController` :

JAVA

```
120   @Override
121   public void initSendable(SendableBuilder builder) {
122     builder.setSmartDashboardType("Motor Controller");
123     builder.setActuator(true);
124     builder.setSafeState(this::disable);
125     builder.addDoubleProperty("Value", this::get, this::set);
126   }
```

C++

```
56 void PWMMotorController::InitSendable(wpi::SendableBuilder& builder) {
57   builder.SetSmartDashboardType("Motor Controller");
58   builder.SetActuator(true);
59   builder.SetSafeState([=, this] { Disable(); });
60   builder.AddDoubleProperty(
61       "Value", [=, this] { return Get(); },
62       [=, this](double value) { Set(value); });
```

En outre, les utilisateurs peuvent appeler `builder.setActuator(true)` pour marquer tout mécanisme qui pourrait se déplacer à la suite de l'entrée `Sendable` en tant qu'actionneur. Actuellement, il est utilisé par *Shuffleboard* pour désactiver les widgets d'actionneur lorsqu'ils ne sont pas en mode *LiveWindow*

12.6 Librairies externes de télémesure

Astuce : Votre librairie n'est-elle pas répertoriée ici alors qu'elle devrait l'être ? Ouvrez une pull request pour l'ajouter !

Il existe plusieurs utilitaires et frameworks de journalisation des librairies tiers qui fournissent des fonctionnalités au-delà de ce qui est actuellement fourni par WPILib :

- AdvantageKit (Java uniquement) : Framework de journalisation basé sur « Log everything » ou « Consigne tout » avec des hooks pour rejouer les données enregistrées dans *simulation*.
- Monologue (Java only) : annotation-based logging library. Extensive telemetry and on-robot logging can be added to your robot code with minimal code footprint and design restrictions.
- DSLOG : An alternate driver station log viewer.

13

Programmation en LabVIEW FRC

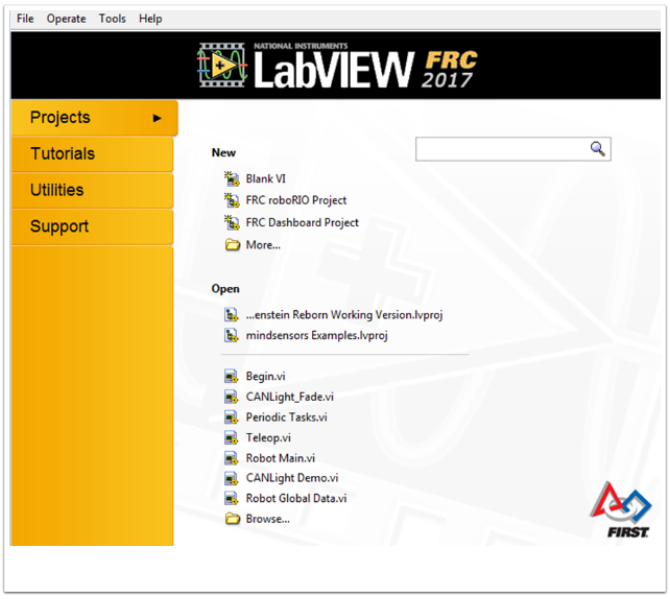
13.1 Création de programmes de robot

13.1.1 Didacticiel Tank Drive

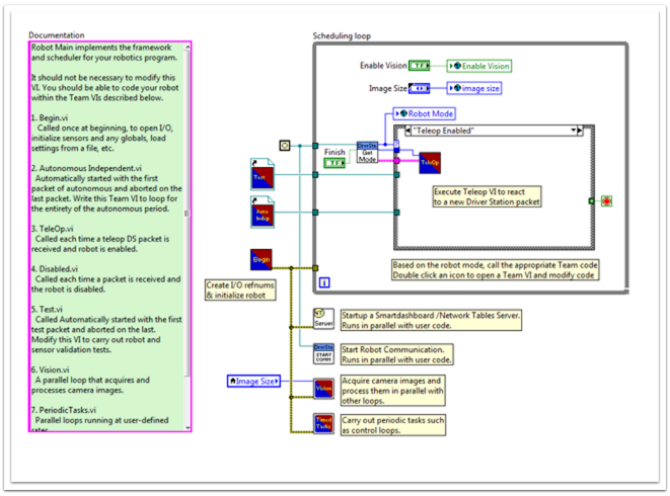
Question : Comment puis-je faire piloter mon robot avec deux joysticks à l'aide du mode de conduite tank drive ?

Solution : There are four components to consider when setting up tank drive for your robot. The first thing you will want to do is make sure the tank drive.vi is used instead of the arcade drive.vi or whichever drive VI you were utilizing previously. The second item to consider is how you want your joysticks to map to the direction you want to drive. In tank drive, the left joystick is used to control the left motors and the right joystick is used to control the right motors. For example, if you want to make your robot turn right by pushing up on the left joystick and down on the right joystick you will need to set your joystick's accordingly in LabVIEW (this is shown in more detail below). Next, you will want to confirm the *PWM* lines that you are wired into, are the same ones your joysticks will be controlling. Lastly, make sure your motor controllers match the motor controllers specified in LabVIEW. The steps below will discuss these ideas in more detail :

1. Ouvrez LabVIEW et double-cliquez sur `FRC roboRIO Project`.



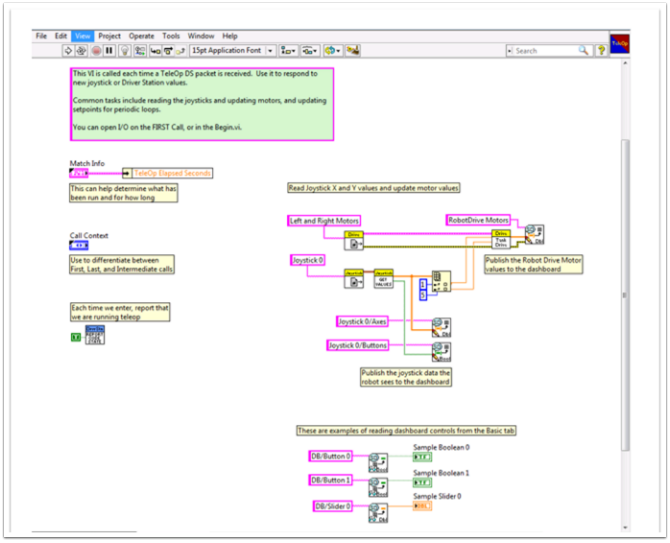
2. Donnez un nom à votre projet, ajoutez votre numéro d'équipe et sélectionnez `Arcade Drive Robot roboRIO`. Vous pouvez sélectionner une autre option, cependant, ce tutoriel traite de la façon de configurer le mode Tank drive pour ce projet.
3. Dans la fenêtre Explorateur de projets, ouvrez l'instrument virtuel `Robot Main.vi`.
4. Appuyez sur `Ctrl + E` pour voir le diagramme. Cela devrait ressembler à l'image suivante :



5. Double-cliquez sur l'instrument virtuel "Teleop" à l'intérieur de la Structure de condition Teleop Enabled. Observez son diagramme bloc. Vous y apporterez deux modifications :

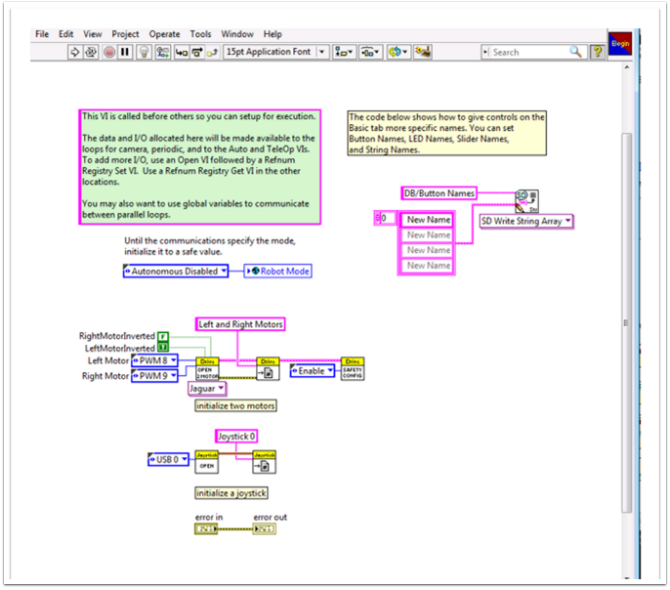
— Remplacez l'instrument virtuel Arcade Drive par l'instrument virtuel Tank drive. Vous le trouvez en cliquant à droite sur le diagramme bloc >> WPI Robotics Library >> Robot Drive >> et en cliquant sur le Tank Drive VI.
— Recherchez la fonction Index Array qui se trouve après l'instrument virtuel Get Values.vi . Vous devrez créer deux constantes numériques et relier chacune d'elles dans à l'une des entrées d'index. Vous pouvez déterminer quelles sont les valeurs de chaque index en regardant l'onglet USB Devices dans l'application FRC® Driver Station. Déplacez les deux joysticks pour déterminer les numéro des ports auxquels (index) ils sont liés. Il serait judicieux d'utiliser l'index de l'axe Y pour chaque joystick. C'est qu'il est intuitif

de pousser vers le haut sur le joystick quand vous voulez commander les moteurs de votre robot de manière à aller vers l'avant, et vers le bas quand vous voulez le commander pour aller en sens inverse. Si vous sélectionnez l'index de l'axe x pour chacun, alors vous devrez déplacer le joystick à gauche ou à droite (directions x-axe) pour commander les moteurs du robot et avoir les déplacements précedemment décrits. Dans ma configuration, j'ai sélectionné l'index 1 pour mes moteurs gauches Y-axe de contrôle et l'indice 5 comme les moteurs droits Y-axe de contrôle. Vous pouvez voir les ajustements sous LabVIEW dans l'image suivante :



6. Ensuite, revenez à votre « Robot Main.vi » et double-cliquez sur l'instrument virtuel « Begin.vi. »
7. La première chose à confirmer dans ce VI est que vos moteurs gauche et droit sont connectés aux mêmes câbles PWM aussi bien dans LabVIEW que sur votre PDP (Panneau de distribution de puissance).
8. La deuxième chose à confirmer dans ce VI est que l'instrument virtuel « Open 2 Motor.vi » a le bon contrôleur de moteurs sélectionné (Talon, Jaguar, Victor, etc.).

Par exemple, j'utilise des contrôleurs de moteurs de type Jaguar et mes moteurs sont reliés à PWM 8 et 9. L'image ci-dessous montre les changements que j'ai besoin de faire :



9. Enregistrez tous les Vis auxquels vous avez fait des ajustements et vous êtes maintenant en mesure de piloter votre robot en mode Tank drive !

13.1.2 Didacticiel de commande et de contrôle

Introduction

Command and Control est un nouveau modèle LabVIEW ajouté pour la saison 2016 qui organise le code robot en commandes et contrôleurs pour une collection de sous-systèmes spécifiques aux robots. Chaque sous-système dispose d'une boucle de contrôle indépendante ou d'une machine à état fonctionnant au taux approprié pour le mécanisme et les commandes de haut niveau qui mettent à jour les opérations souhaitées et définissent les points de consigne. Il est ainsi très facile dans le code autonome de construire des séquences synchrones de commandes. Pendant ce temps, TeleOp en tire profit, car il peut utiliser les mêmes commandes sans avoir besoin d'attendre la terminaison, permettant l'annulation facile et l'initiation de nouvelles commandes en fonction des entrées émanant de l'équipe de pilotage. Chaque sous-système dispose d'un panneau affichant ses valeurs de capteurs et de contrôle en fonction du temps, et de traçage des commandes pour faciliter le débogage.

Qu'entend-t-on par commande et contrôle ?

Command and Control reconnaît que les robots FRC® ont tendance à être construits de mécanismes relativement indépendants tels que Drive, Shooter, Arm, etc. Chacun d'eux est appelé sous-système et a besoin de code qui coordonnera les différents capteurs et actionneurs du sous-système afin de compléter les commandes demandées, ou des actions, telles que « Close Gripper » ou « Lower Arm ». L'un des principes clés de ce cadre est que les sous-systèmes auront chacun une boucle de contrôleur indépendante qui est uniquement responsable de la mise à jour des moteurs et autres actionneurs. Le code en dehors du contrôleur de sous-système peut émettre des commandes qui peuvent modifier la sortie du robot, mais ne doivent pas modifier directement les sorties. La différence est très subtile, mais cela signifie que les sorties ne peuvent être mises à jour qu'à partir d'un seul emplacement du projet. Cela accélère le débogage d'un robot se comportant de façon inattendue en vous donnant la possibilité

de regarder à travers une liste de commandes envoyées au sous-système plutôt que de rechercher votre projet pour dénicher l'endroit où une sortie peut avoir été modifiée. Il devient également plus facile d'ajouter un capteur supplémentaire, de changer d'engrenage ou de désactiver un mécanisme sans avoir besoin de modifier le code à l'extérieur du contrôleur.

Game code, composé principalement des modes Autonome et TeleOp, devra généralement mettre à jour les points de consigne et réagir à l'état de certains mécanismes. Pour le mode Autonome, il est très courant de définir le fonctionnement du robot comme une séquence d'opérations – déplacement, ramasser, transporter là, le tirer, etc. Les commandes peuvent être agencées séquentiellement avec une logique supplémentaire pour construire rapidement des routines complexes. Pour le monde teleOp, les mêmes commandes peuvent s'exécuter de façon asynchrone, ce qui permet au robot de toujours traiter les dernières entrées du pilote, et si elle sont implémentées correctement, de nouvelles commandes s'interrompent, permettant à l'équipe de pilotage de répondre rapidement aux conditions du terrain tout en profitant des commandes automatisées et des séquences de commandes.

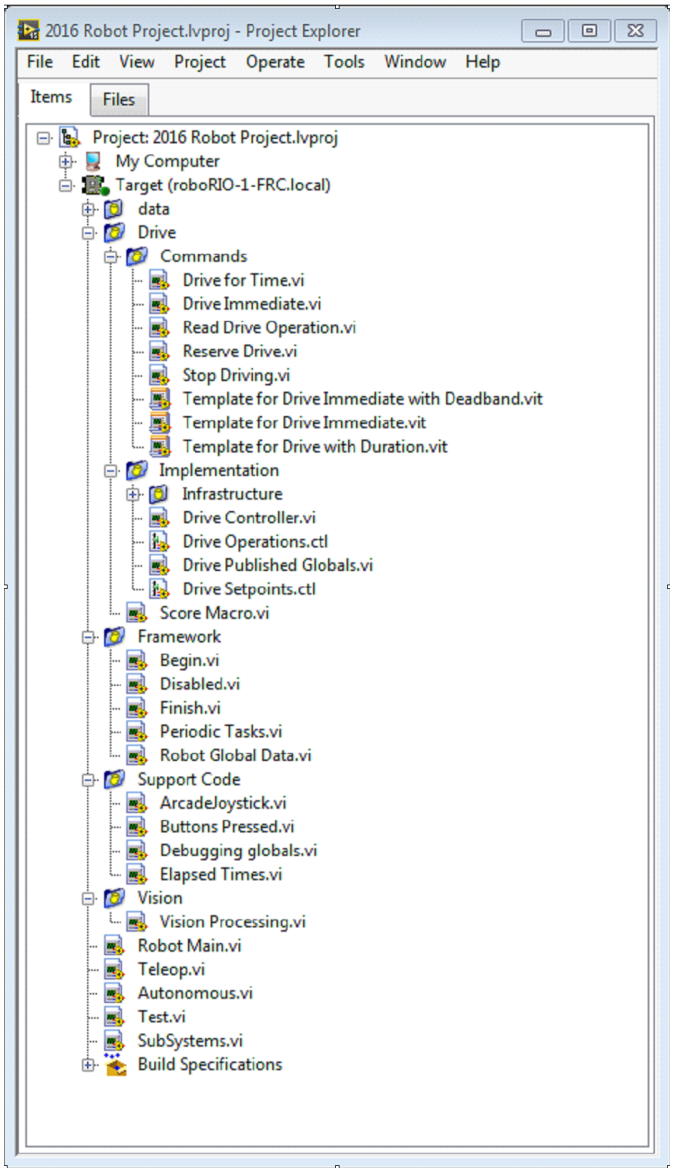
Pourquoi devrais-je utiliser la commande et le contrôle ?

Command and Control ajoute des fonctionnalités aux modèles de projet LabVIEW existants, permettant au code de mieux évoluer avec des robots et des code plus sophistiqués. Les sous-systèmes sont utilisés pour abstraire les détails de l'implémentation, et le code du jeu est construit à partir de séquences de commande Vi de haut niveau. Les commandes elles-mêmes sont des VIs qui peuvent mettre à jour les points de consigne, effectuer une mise à l'échelle numérique/correspondance entre les unités d'ingénierie et les unités de mécanisme et offrir des options de synchronisation. Si des modifications physiques sont apportées au robot, telles que la modification d'un rapport d'engrenage, des modifications peuvent être apportées à quelques commandes Vis pour refléter cette modification sur l'ensemble du code.

L'encapsulation des I/O permet un fonctionnement plus prévisible et un débogage plus rapide lorsque des conflits de ressources se produisent. Étant donné que chaque commande est un VI, vous pouvez passer une seule étape à travers les commandes ou utiliser la fonctionnalité intégrée *Trace* pour afficher une liste de toutes les commandes envoyées à chaque sous-système. Le cadre de développement utilise une notification asynchrone et une propagation cohérente des données, ce qui facilite le codage d'une séquence de commandes ou l'ajout d'une logique simple pour déterminer la commande adéquate à exécuter.

Partie 1 : Explorateur de projet

L'Explorateur de projets fournit une organisation pour tous les Vis et les fichiers que vous utiliserez pour votre système robot. Vous trouverez ci-dessous une description des principaux composants de l'Explorateur de projets pour aider à l'expansion de votre système. Les éléments les plus fréquemment utilisés ont été marqués en gras.



Mon ordinateur
: Les éléments qui définissent l'opération sur l'ordinateur sur lequel le projet a été chargé. Pour un projet de robot, il est utilisé comme cible de simulation et est rempli de fichiers de simulation.

Fichiers de support Sim
: The folder containing 3D *CAD* models and description files for the simulated robot.

Simulation de robot Readme.html
: Documents the *PWM* channels and robot info you will need in order to write robot code that matches the wiring of the simulated robot.

Dépendances
: Expose les fichiers utilisés par le code du robot simulé ; se remplira lorsque vous concevez le code de la cible du robot simulé.

Build Specifications
: Cela contiendra les fichiers qui spécifient comment compiler et déployer du code pour le robot simulé cible .

Cible (roborio-TEAM-FRC.local)
Les éléments qui définissent l'opération sur le roboRIO situé à (adresse).

Lecteur
L'implémentation du sous-système et les commandes de la base pilotable du robot. Elle sert une alternative personnalisée aux RobotDrive VIs de la WPILib .

Environnement de développement
Les Vis utilisés pour le code robot qui ne fait pas partie d'un sous-système et qui ne sont pas très souvent utilisés.

Begin
Appelé une fois quand le code robot démarre pour la première fois. Ceci est utile pour le code d'initialisation qui n'appartient pas à un sous-système en particulier.

Disabled
Appelé une fois pour chaque paquet désactivé et peut être utilisé pour déboguer les capteurs lorsque vous ne voulez pas que le robot se déplace.

Finish
Pendant l'exécution du code du robot, ce VI peut être appelé lorsque le code du robot se termine. Il n'est pas appelé lorsque le programme avorte ou lorsque l'alimentation est coupée.

Periodic Tasks
Un bon endroit pour les boucles périodiques ad hoc pour le débogage ou la surveillance

Robot Global Data
Utile pour partager des informations du robot qui n'appartiennent pas à un sous-système en particulier.

Support Code
Débogage et aides au développement du code.

Vision
Sous-système et commandes pour la caméra et le traitement d'image.

Robot Main.vi
C'est le VI que vous lancez pour l'execution du code au complet de votre robot.

Autonomous.vi
Ce VI est exécuté pendant le mode autonome.

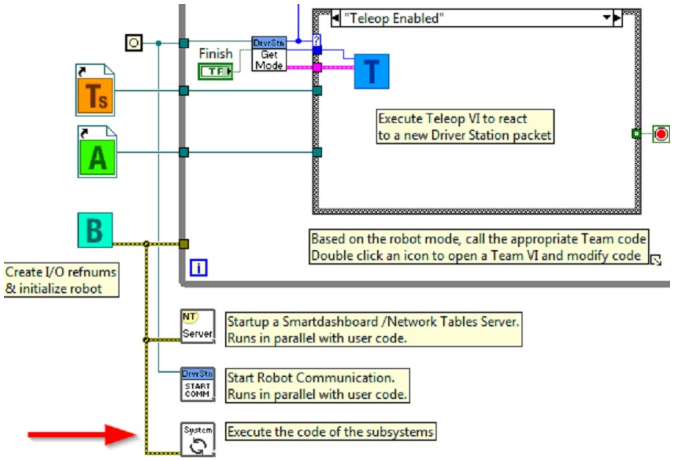
Teleop.vi
Ce VI est appelé pour chaque paquet du mode TeleOp.

Test.vi
Ce VI est exécuté lorsque la Driver Station est en mode test.

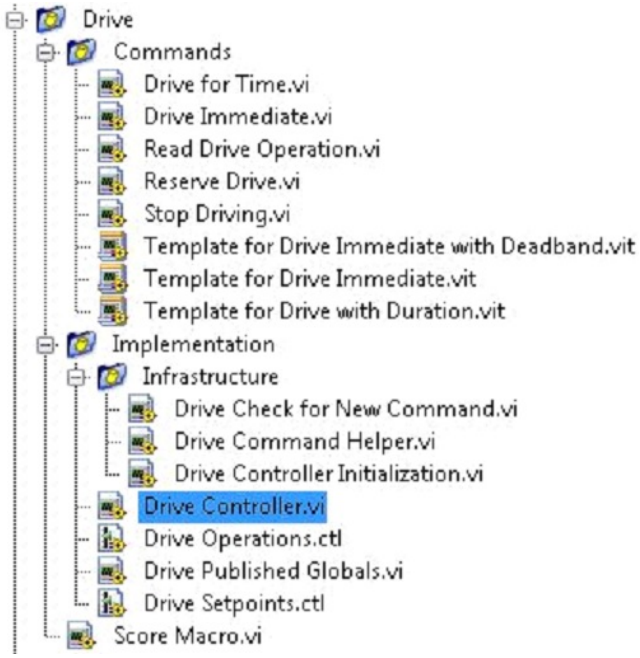
SubSystems.vi
Ce VI renferme et démarre tous les sous-système.

Dépendances
Affiche les fichiers utilisés par le code robot.

Build Specifications
Utilisé pour compiler et exécuter le code en tant qu'application de démarrage une fois que le code fonctionne correctement.



Explorateur de projets pour le sous-système lecteur



Commands :

Ce dossier contient les VIs de commandes qui commandent au contrôleur d'effectuer une opération. Il contient également des modèles pour la création de commandes de déplacement supplémentaires.

Note : Après avoir créé une nouvelle commande, vous pouvez avoir besoin de modifier `Drive Setpoints.ctl` pour ajouter ou mettre à jour les champs utilisés par le contrôleur pour définir la nouvelle fonctionnalité. Vous pouvez également avoir besoin d'entrer dans le Controller.vi pour le déplacement et modifier la **structure case** afin d'ajouter un cas pour chaque valeur.

Implémentation

Il s'agit des VIS et des contrôles utilisés pour construire le sous-système.

Infrastructure VIs

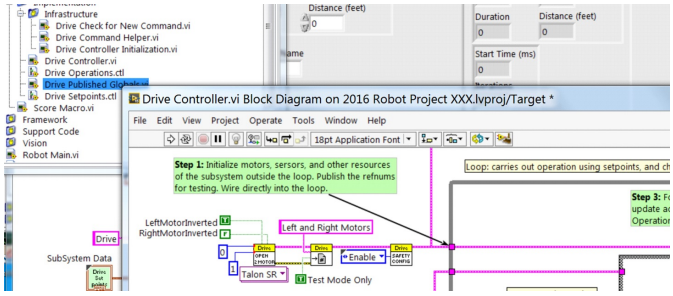
- Drive Check for New Command : Il est appelé à chaque itération de la boucle du contrôleur. Il vérifie les nouvelles commandes, met à jour les données de synchronisation et, une fois terminé, notifie une commande en attente.
- Drive Command Helper.vi : Les commandes appellent ce VI pour informer le contrôleur qu'une nouvelle commande a été émise.
- Drive Controller Initialization.vi : Il alloue le notifiant et combine la synchronisation, la commande par défaut et d'autres informations dans un seul câble de données.

- Drive Controller.vi : Ce VI contient la boucle pour le contrôle/la machine à états. Le panneau peut également contenir des affichages utiles pour le débogage.
- Drive Operation.ctl : Ce typedef définit les modes opérationnels du contrôleur. De nombreuses commandes peuvent partager une opération.
- Drive Setpoint.ctl : Il contient les champs de données utilisés par tous les modes d'opération du sous-système de déplacement.
- Drive Published Globals.vi : Une place utile pour publier des informations globales du sous-système de déplacement.

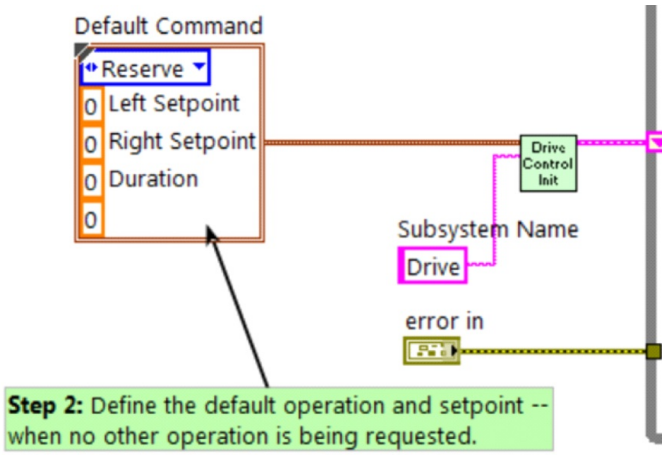
Partie 2 : Initialisation du sous-système de déplacement

Sur le diagramme bloc du contrôleur, il y a des commentaires en fond vert qui soulignent les zones clés qu'il serait judicieux de savoir comment modifier.

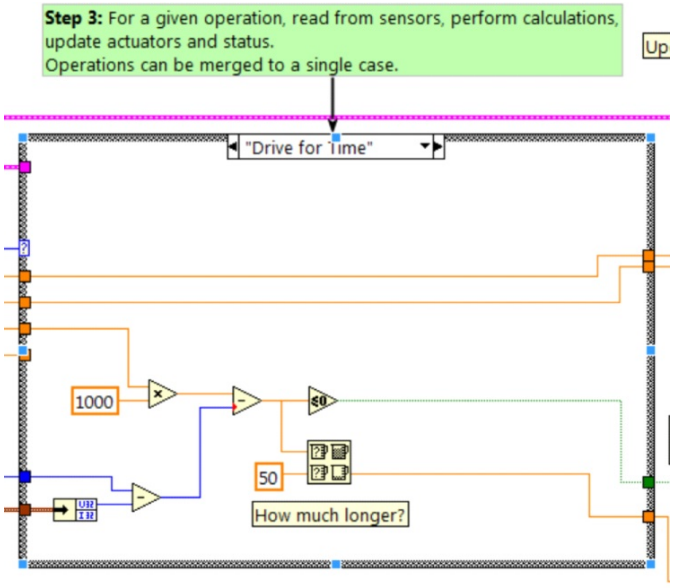
La zone située à gauche de la boucle de contrôle s'exécute une fois lorsque le sous-système démarre. C'est là que vous allouez et initialisez généralement toutes les données d'I/O et d'état. Vous pouvez publier les I/O refnums, ou vous pouvez les enregistrer en mode test uniquement pour les garder privés afin que d'autres codes externes ne puissent pas mettre à jour les moteurs sans utiliser une commande.



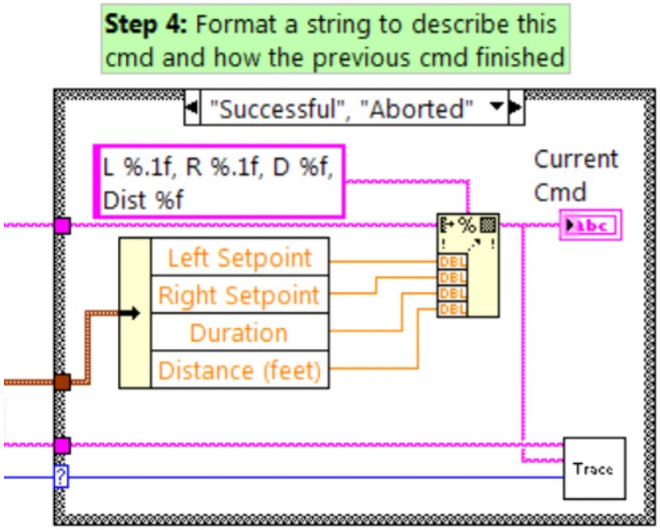
Note : L'initialisation des ressources pour chaque sous-système dans leur Controller.vi respectif plutôt que dans Begin.vi améliore l'encapsulation des I/O, réduisant les conflits de ressources potentiels et simplifie le débogage.



Une partie de l'initialisation consiste à sélectionner l'opération par défaut et à définir des valeurs de points de consigne lorsqu'aucune autre opération n'est en cours de traitement.

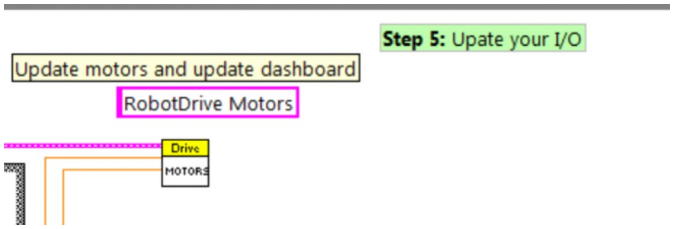


À l'intérieur de la boucle de contrôle se trouve une statement à conditions où les opérations sont effectivement implémentées. Les points de consigne, les délais d'itération, le nombre d'itérations et les capteurs peuvent tous avoir une influence sur la façon de fonctionner du sous-système. Cette structure à condition a une valeur pour chaque état d'opération du sous-système.



Chaque itération de la boucle du contrôleur mettra éventuellement à jour le *Trace VI*. Le cadre de développement intègre déjà le nom, l'opération et la description du sous-système, et vous trouverez peut-être utile de mettre en forme des valeurs des points de consigne dans les informations de trace. Ouvrez le Trace VI et cliquez sur Enable pendant que le code robot s'exécute pour atteindre les points de consigne actuels et les commandes envoyées à chaque sous-système.

L'objectif principal du contrôleur est de mettre à jour les actionneurs du sous-système. Cela peut se produire dans la structure à conditions, mais très souvent, il est avantageux de le faire en aval de la structure pour s'assurer que les valeurs sont toujours mises à jour avec la valeur correcte et dans un seul emplacement dans le code.



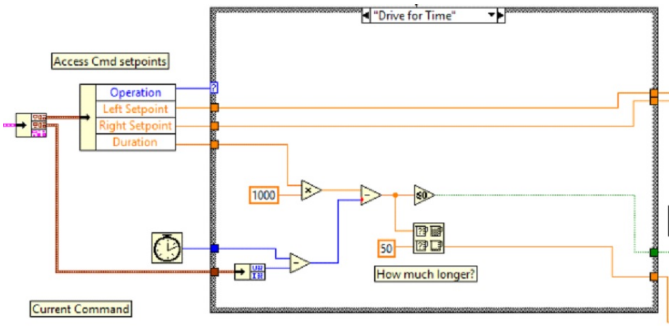
Partie 3 : Commandes intégrées dans le sous-système de déplacement

Il existe 3 commandes d'exemple jointes pour chaque nouveau sous-système :

Drive For Time.vi

Ce VI configure les moteurs de manière à fonctionner pendant un certain nombre de secondes. Il se synchronise éventuellement avec la fin de l'exécution de la commande.

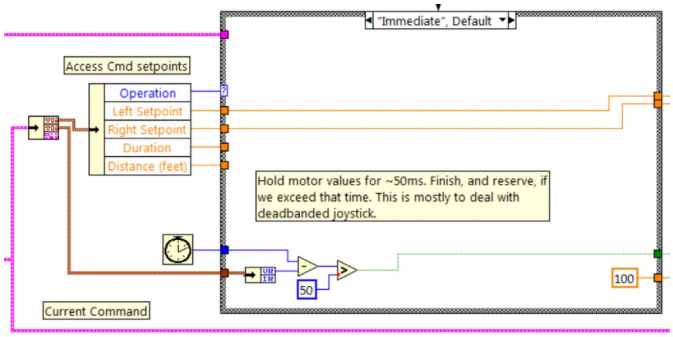
La condition *The Drive for Time* actionnera les moteurs au point de consigne jusqu'à ce que le délai s'écoule ou qu'une nouvelle commande soit émise. Si les moteurs ont le paramètre *safety timeout* activé, il est nécessaire de les mettre à jour au moins une fois tous les 100ms. C'est pourquoi le code attend la plus petite fraction du temps restant et 50ms.



Drive Immediate.vi

Obtient les vitesses souhaitées à gauche et à droite pour les moteurs et configurera ces moteurs immédiatement à ces points de consigne.

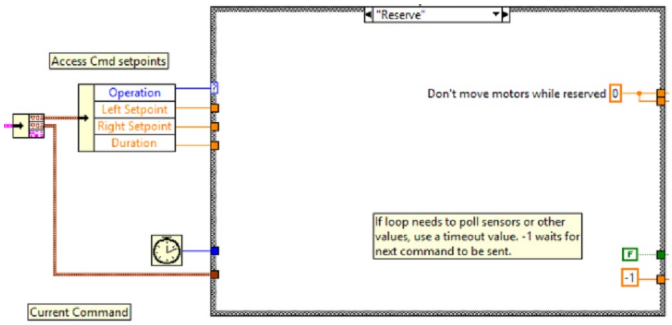
La condition *Immediate* met à jour les moteurs au point de consigne défini par la commande. La commande n'est pas considérée comme terminée puisque vous souhaitez que les moteurs conservent cette valeur jusqu'à ce qu'une nouvelle commande entre ou jusqu'à ce qu'un délai d'expiration soit atteint. Le délai d'attente est utile è chaque fois qu'une commande comprend une *zone morte*. De petites valeurs ne seront pas demandées si elles sont plus petites que la zone morte, et entraîneront un grongnement ou un fluage à moins que la commande ne temporise.



Stop Driving.vi

met les moteurs d'entraînement à zéro, ce qui immobilise le robot.

La commande Reserve éteint les moteurs et attend une nouvelle commande. Lorsqu'elle est utilisée avec une séquence de commande nommée, Reserve identifie que le sous-système de déplacement fait partie de cette séquence, même si elle n'est pas en train de déplacer le robot. Cela permet d'arbitrer la ressource sous-système entre les commandes simultanément en cours d'exécution.



Partie 4 : Création de nouvelles commandes

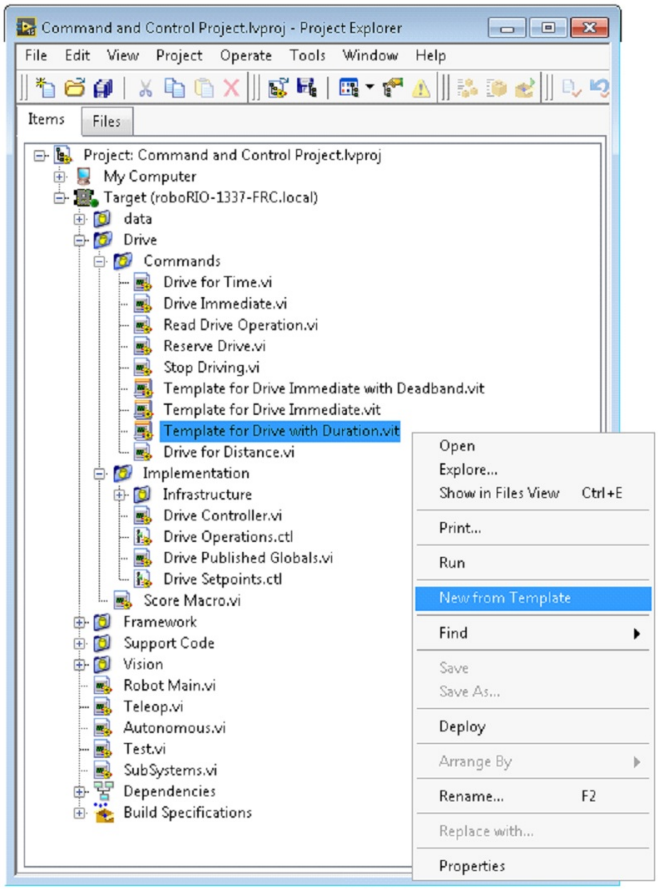
Le cadre de développement *Command and Control* permet aux utilisateurs de créer facilement de nouvelles commandes pour un sous-système. Pour créer une nouvelle commande ouvrez le dossier sous-système/Commands dans la fenêtre explorateur de projet, choisissez l'un des VI Templates à utiliser comme point de départ de votre nouvelle commande, cliquez avec le bouton droit et sélectionnez New From Template.

- **Immediate** : Ce VI informe le sous-système d'un nouveau point de consigne.
- **Immediate with deadband** : Ce VI compare la valeur d'entrée de la zone morte et informe éventuellement le sous-système du nouveau point de consigne. Ceci est très

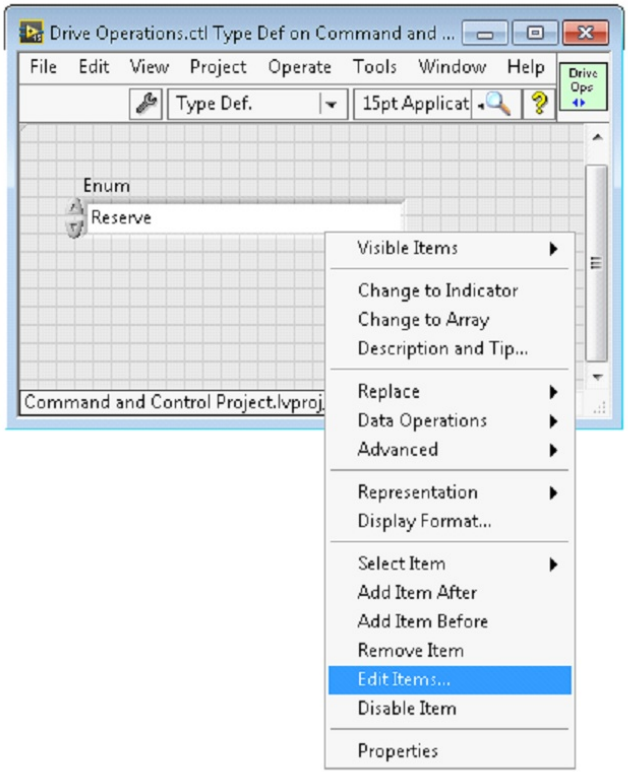
utile lorsque des valeurs continues joystick sont utilisées.

— **With duration** : Ce VI informe le sous-système d'exécuter cette commande pendant la durée donnée, puis de revenir à l'état par défaut. La synchronisation détermine si ce VI démarre l'opération et retourne immédiatement ou attend que l'opération se termine. La première option est couramment utilisée pour le mode TeleOp, et la seconde pour des séquences en mode autonome.

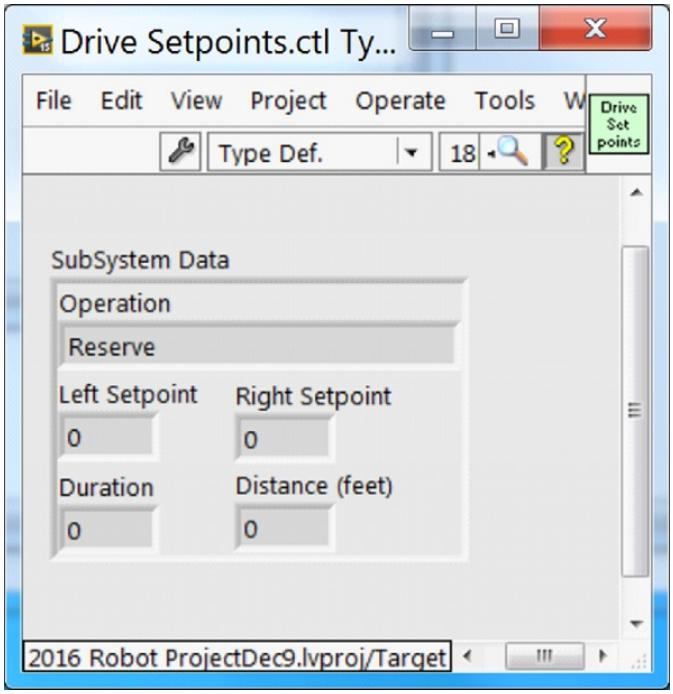
Dans cet exemple, nous allons ajouter la nouvelle commande « Drive for Distance ».



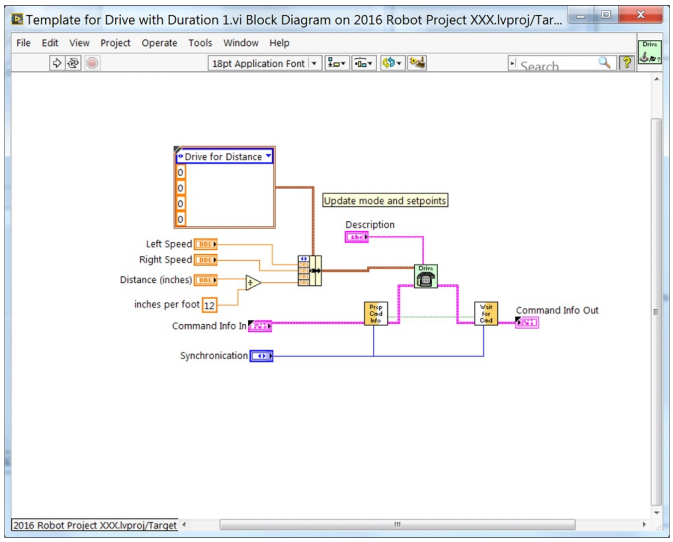
Premièrement, enregistrez le nouveau VI avec un nom descriptif tel que « Drive for Distance ». Ensuite, déterminez si la nouvelle commande a besoin d'une nouvelle valeur ajoutée à *enum typedef* des *Drive Operations*. Le code de projet initial a déjà une valeur enum de Drive for Distance, mais l'image suivante montre comment vous en ajouteriez une si nécessaire.



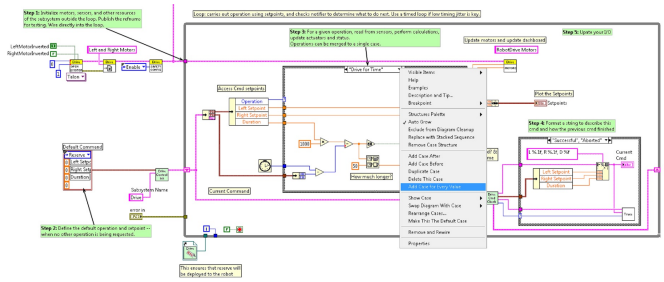
Si une commande a besoin d'informations supplémentaires pour les exécuter, ajoutez-les au contrôle des points de consigne. Par défaut, le sous-système de déplacement comporte des champs pour le point de consigne gauche, le point de consigne droit et la durée ainsi que l'opération à exécuter. La commande *Drive for Distance* peut réutiliser *Duration* en tant que distance, mais ajoutons un contrôle numérique au Drive Setpoints.ctl appelé Distance (pieds).



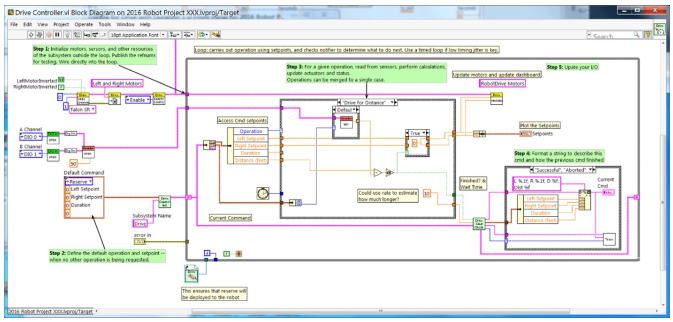
Une fois que nous avons tous les champs nécessaires pour spécifier notre commande, nous pouvons modifier *Drive for Distance.vi*, nouvellement créé. Comme indiqué ci-dessous, sélectionnez *Drive for Distance* à partir du menu déroulant de l'enum's et ajouter les paramètres au VI pour spécifier la distance, les vitesses, etc. Si les unités ne correspondent pas, la commande VI est un excellent endroit pour effectuer la correspondance entre les unités.



Ensuite, ajoutez du code au **Drive Controller** pour définir ce qui se passe lorsque la commande *Drive for Distance* s'exécute. Cliquez avec le bouton droit sur la Structure de conditions et le dupliquer ou ajouter d'autres cas chaque valeur. Cela créera une nouvelle condition « Drive for Distance ».

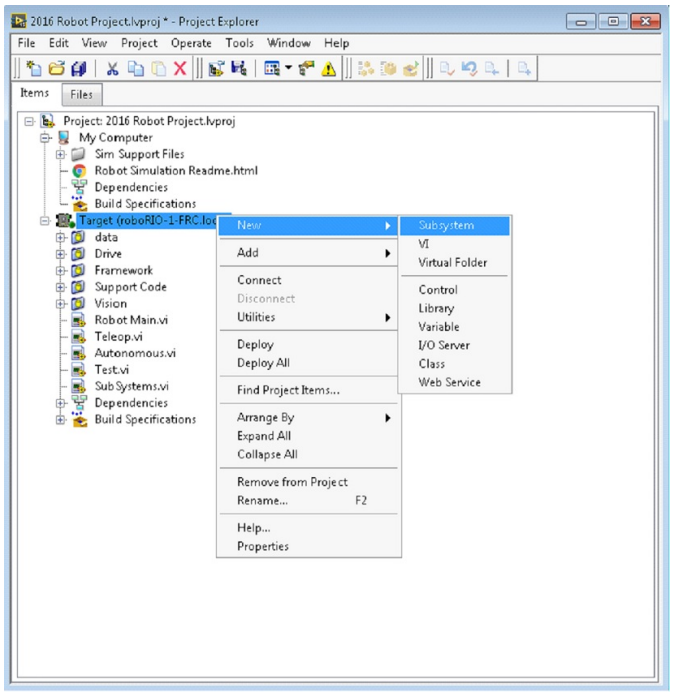


Afin d'accéder aux nouveaux champs de setpoint, développez le nœud de dissocation « Access Cmd setpoints ». Ouvrez votre encodeur(s) à l'extérieur, à gauche de la boucle. Dans le nouveau diagramme de la structure case, nous avons ajouté un appel pour réinitialiser l'encodeur sur la première itération de boucle et le lire autrement. Il y a aussi un simple code qui compare les valeurs de l'encodeur et met à jour la puissance du moteur. Si de nouveaux contrôles sont ajoutés au cluster des points de consigne, vous devez également envisager de les ajouter au Trace VI. Les modifications nécessaires sont indiquées dans l'image ci-dessous.

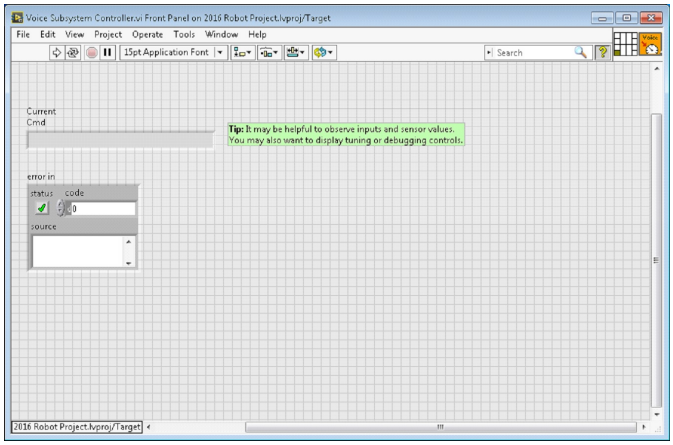


Partie 5 : Création d'un sous-système

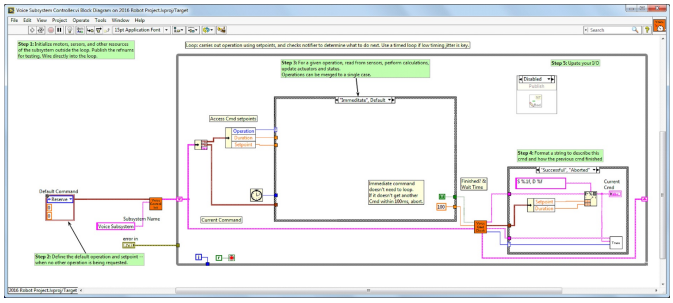
Afin de créer un nouveau sous-système, cliquez avec le bouton droit sur la cible roboRIO et sélectionnez New» Subsystem. Dans la boîte de dialogue contextuel, entrez le nom du sous-système, répertoriez les modes d'opération et spécifiez la couleur de l'icône.



Lorsque vous cliquez sur OK, le dossier du sous-système sera généré et ajouté au dossier et à l'arborescence du disque de projet. Il contiendra une implémentation de base des VIS et des contrôles qui constituent un sous-système. Un appel au nouveau contrôle sera inséré dans les sous-systèmes VI. Le contrôle VI s'ouvrira, prêt pour que vous ajoutiez des E/S et à implémenter une machine d'état ou un code de contrôle. Les icônes VI générées utilisent la couleur et le nom fournis dans la boîte de dialogue. Le code généré utilise des typedefs pour définir des champs et des points de consigne et des opérations.



Vous trouverez ci-dessous le diagramme bloc du sous-système nouvellement créé. Ce code est généré automatiquement lorsque vous créez le sous-système.



13.2 Ressources pour LabVIEW

13.2.1 Ressources pour LabVIEW

Note : To learn more about programming in LabVIEW and specifically programming FRC® robots in LabVIEW, check out the following resources.

Les bases de LabVIEW

NI provides tutorials on the basics of LabVIEW. These tutorials can help you get acquainted with the LabVIEW environment and the basics of the graphical, dataflow programming model used in LabVIEW.

Tutoriels NI FRC

NI héberge également de nombreux tutoriels et présentations spécifiques à la FRC allant du niveau de base au niveau avancé. Pour une ressource unique en profondeur, consultez des ateliers de formation FRC des niveaux de base et avancé dont le lien est situé près du bas de la page.

Didacticiels et exemples installés

Il existe également des tutoriels et des exemples pour toutes sortes de tâches et de composants fournis avec votre installation LabVIEW. Pour accéder aux didacticiels, à partir de l'écran LabVIEW Splash (l'écran qui apparaît lors du premier démarrage du programme), cliquez sur l'onglet `Tutorials` sur le côté gauche. Notez que les tutoriels sont tous dans un seul document, donc une fois qu'il est ouvert, vous pouvez naviguer vers d'autres tutoriels sans avoir à revenir à l'écran de démarrage.

Pour accéder aux exemples, cliquez sur l'onglet `Support`, puis `Find FRC Examples` ou chaque fois que vous travaillez sur un programme ouvrez le menu `Help`, sélectionnez `Find Examples` et ouvrez le dossier `FRC Robotics`.

Third Party Resources

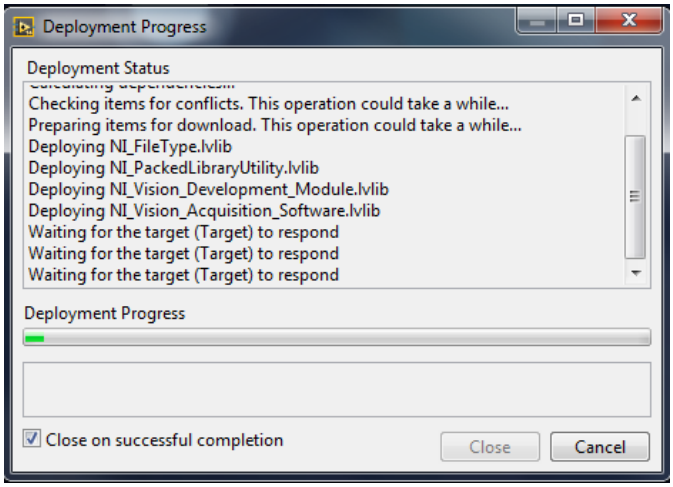
— FRC Control and Trajectory Library
— Secret Book Of FRC LabVIEW 2

13.2.2 En attendant que la cible réponde - Se remettre des boucles mal conçues

---

Note : Si vous téléchargez du code LabVIEW qui contient une boucle sans contrainte (une boucle sans délai), il est possible de mettre le roboRIO dans un état où LabVIEW est incapable de se connecter pour télécharger un nouveau code. Ce document explique le processus requis pour charger un nouveau code fixe pour récupérer de cet état.

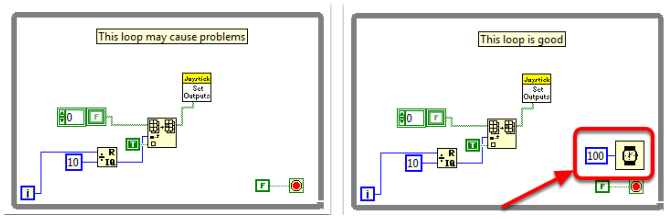
Le symptôme



Le symptôme principal de ce problème consiste aux tentatives de déploiement du nouveau code robot figé à l'étape « Waiting for the target (Target) to respond » comme indiqué ci-dessus. Notez qu'il existe d'autres causes possibles de ce symptôme (comme le passage d'un programme C++ ou Java à un programme LabVIEW), mais les étapes décrites ici devraient résoudre la plupart ou la totalité d'entre elles.

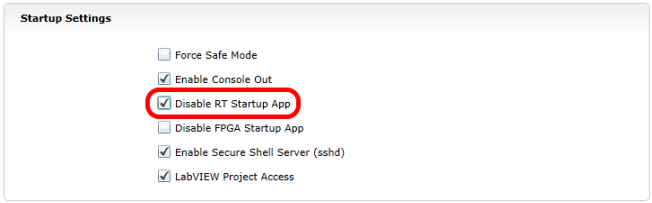
Cliquez sur `Cancel` pour fermer la boîte de dialogue du déploiement.

Le problème



L'une des sources courantes de ce problème est la présence de boucles non contraintes dans votre code LabVIEW. Une boucle sans contrainte est une boucle qui ne contient aucun élément de délai (comme celle de gauche). Si vous ne savez pas par où commencer à chercher, Disabled.VI, Periodic Tasks.VI et Vision Processing.VI sont les endroits communs où on retrouve ce type de boucle. Pour corriger ce problème, ajoutez dans votre code un élément de délai tel que le Wait(ms) VI de la palette Timing, présent dans la boucle droite.

Ne définir aucune App

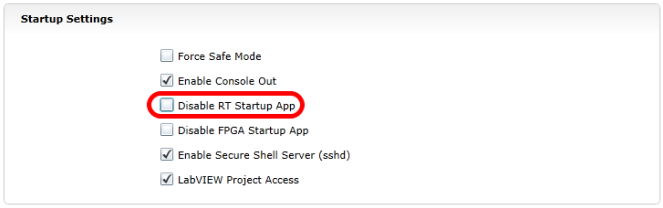


L'utilisation du serveur web du roboRIO (voir l'article *roboRIO Web Dashboard Startup Settings* pour plus de détails). **Cochez** la case « Disable RT Startup App ».

Redémarrer

Redémarrez le roboRIO, soit à l'aide du bouton Reset situé sur l'appareil, soit en cliquant sur Restart dans le coin supérieur droit de la page Web.

N'effacer aucune App



Utilisation du serveur web du roboRIO (voir l'article sur *roboRIO Web Dashboard Startup Settings*, pour plus de détails). **Décochez** la case « Disable RT Startup App ».

Charger le code LabVIEW

Chargez le code LabVIEW (à l'aide du bouton Run ou Run as Startup). Assurez-vous de configurer le code LabVIEW à « Startup before rebooting the roboRIO » ou vous devrez à nouveau suivre les instructions ci-dessus.

13.2.3 Comment basculer entre deux modes caméra

Ce code montre comment utiliser un bouton pour basculer entre deux modes caméra distincts. Le code se compose de quatre étapes.

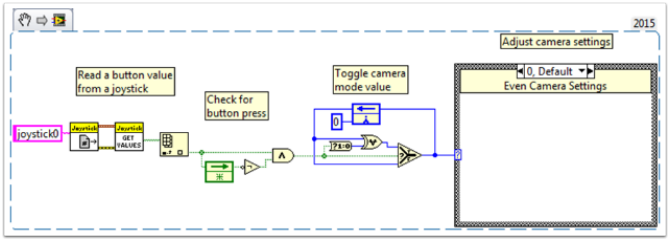
Dans la première étape, la valeur d'un bouton sur le joystick est lue.

Ensuite, la lecture actuelle est comparée à la lecture précédente à l'aide d'un **Feedback Node** et d'une certaine arithmétique booléenne. Ensemble, toutes ces opérations s'assurent que le mode caméra n'est basculé que lorsque le bouton est appuyé pour la première fois au lieu de basculer plusieurs fois pendant que le bouton est maintenu enfoncé.

Après cela, le mode caméra est basculé en masquant le résultat de la deuxième étape sur la valeur actuelle du mode caméra. C'est ce qu'on appelle le masquage de bits et en le faisant

à l'aide de la fonction **XOR**, le code basculera le mode caméra lorsque la deuxième étape renvoie la valeur logique VRAI et, autrement le code ne fera rien autrement.

Enfin, vous pouvez insérer le code pour chaque mode caméra dans la Structure case à la fin. Chaque fois que le code est exécuté, cette section exécute le code pour le mode caméra actuel.



13.2.4 Exemples et didacticiels LabVIEW

Didacticiels populaires

Didacticiel sur un déplacement chronométré en mode autonome
- Déplace votre robot de façon autonome selon différents intervalles de temps
- Pour en en savoir plus sur le déplacement en mode autonome

Didacticiel sur un contrôle de moteur simple
- Configuration matérielle et logicielle d'un moteur pour votre roboRIO
- Learn to setup the FRC® Control System and FRC Robot Project
- Pour en savoir plus sur le contrôle du moteur

Didacticiel sur le traitement d'images
- Apprendre les techniques de base de traitement d'images et comment utiliser NI Vision Assistant
- En savoir plus sur les caméras et le traitement d'images

Didacticiel sur la commande PID
- Qu'est-ce que c'est une commande PID et comment puis-je l'implémenter ?

Didacticiel de commande et de contrôle
- Qu'est-ce que sont la commande et le contrôle ?
- Comment puis-je l'implémenter ?

Didacticiel sur la Driver Station
- Découvrez la Driver Station FRC

Didacticiel sur le mode Test
- Apprendre à configurer et à utiliser le mode Test

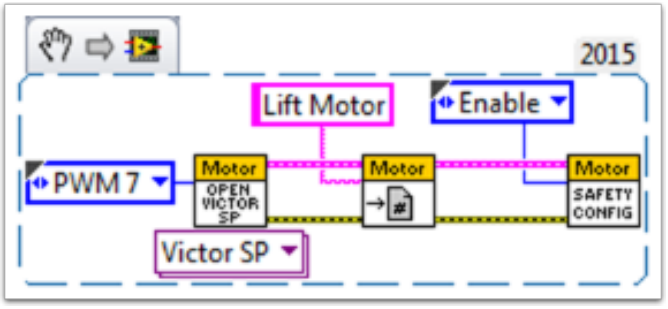
Vous cherchez d'autres exemples et discussions ? Recherchez dans d'autres documents ou publiez votre propre discussion, exemple de code ou tutoriel en cliquant ici !N'oubliez pas de marquer vos publications avec une balise !

13.2.5 Ajouter un moteur indépendant à un projet

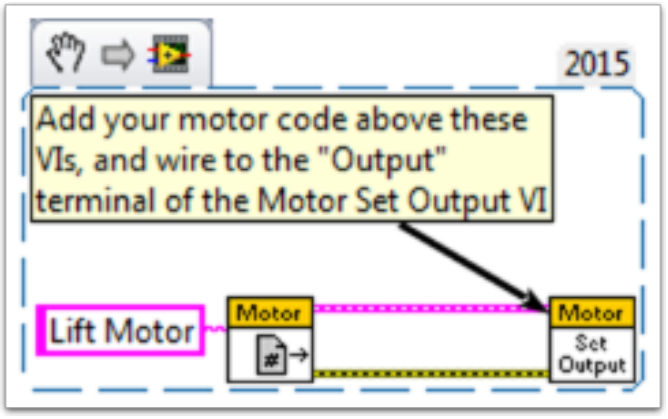
Une fois que le système d'entraînement qui contrôle vos roues de déplacement est tout configuré, vous pourriez avoir besoin d'ajouter un moteur supplémentaire pour contrôler un mécanisme complètement indépendant des roues, comme un bras. Puisque ce moteur ne fera pas partie de votre système d'entraînement Tank, Arcade, ou Mecanum, vous devrez le contrôler de manière indépendante.

Ces extraits de VI montrent comment configurer un seul moteur dans un projet qui peut déjà contenir un système d'entraînement multimoteur. Si vous voyez le symbole HAND>ARROW-LABVIEW, faites glisser l'image dans votre diagramme bloc, et voila : code ! Ok, voici comment vous le faites.

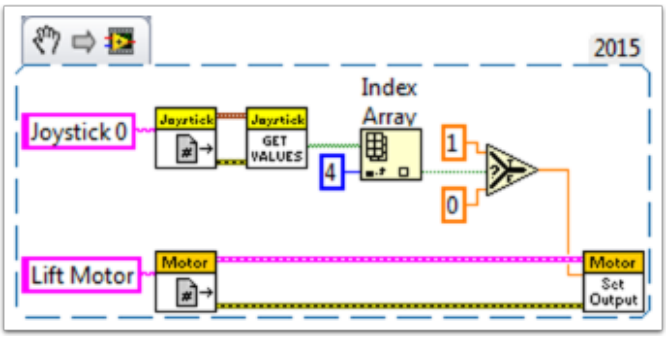
FIRST, create a motor reference in the **Begin.vi**, using the **Motor Control Open VI** and **Motor Control Refnum Registry Set VI**. These can be found by right-clicking in the block diagram and going to **WPI Robotics Library>>RobotDrive>>Motor Control**. Choose your *PWM* line and name your motor. I named mine « Lift Motor » and connected it to PWM 7. (I also included and enabled the Motor Control Safety Config VI, which automatically turns off the motor if it loses connection.)



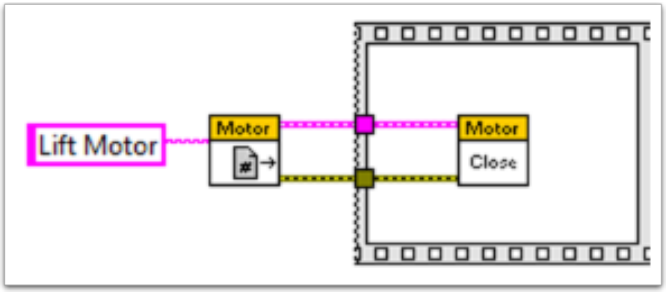
Maintenant, référencez votre moteur (le nom doit être exact) dans l'instrument virtuel **Teleop.vi** en utilisant l'instrument virtuel **Motor Control Refnum Registry Get VI** et configurez-le adéquatement à l'aide de **Motor Control Set Output VI**. Ces derniers sont au même endroit que les VI ci-dessus.



Par exemple, l'extrait de code suivant indique à Lift Motor d'avancer si le bouton 4 est appuyé sur le Joystick 0 et de rester immobile autrement. Pour moi, le bouton 4 correspond au bumper gauche sur ma manette de style Xbox (« Joystick 0 »). Pour savoir en profondeur sur les options de bouton joystick, consultez *Comment utiliser des boutons Joystick pour contrôler les moteurs ou les solénoïdes*.



Enfin, nous devons fermer les références dans l'instrument virtuel **Finish.vi** (tout comme nous le faisons avec le système de déplacement et le joystick), en utilisant les instruments virtuels **Motor Control Refnum Registry Get VI** et **Motor Control Close VI**. Bien que cette image montre l'instrument virtuel **Close VI** dans une structure de séquence plate par lui-même, nous voulons vraiment avoir tous les **Close VI** dans le même cadre. Vous pouvez simplement mettre ces deux VI sous les autres **Get VIs** et **Close VIs** (pour le joystick et le système d'entraînement).

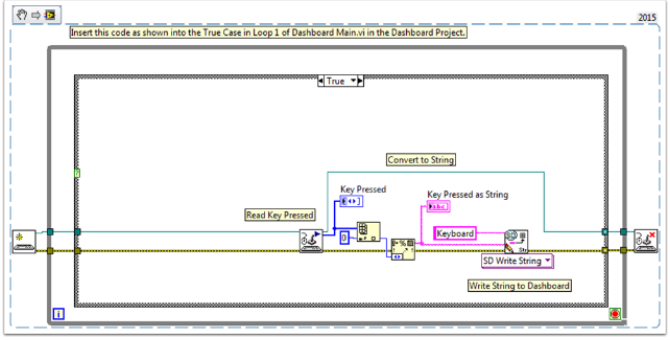


J'espère que cela vous aidera à programmer le meilleur robot qui soit ! Bonne chance !

13.2.6 Navigation au clavier avec le roboRIO

Cet exemple fournit quelques suggestions pour contrôler le robot à l'aide de la navigation du clavier à la place d'un joystick ou d'un autre contrôleur. Dans ce cas, nous utilisons les touches **A**, **W*, **S** et **D** pour contrôler deux moteurs d'entraînement dans une configuration Tank.

Le premier extrait VI est le code qui devra être inclus dans l'instrument virtuel `Main VI` du Dashboard. Vous pouvez insérer ce code dans le cas ""True"" de la *boucle 1*. Le code ouvre une connexion au clavier avant le début de l'exécution de la boucle, et à chaque itération il lit la touche pressée. Ces informations sont converties en une chaîne, qui est ensuite transmise au `Teleop VI` dans le projet du robot. Lorsque la *boucle 1* cèsse de s'exécuter, la connexion au clavier se ferme.

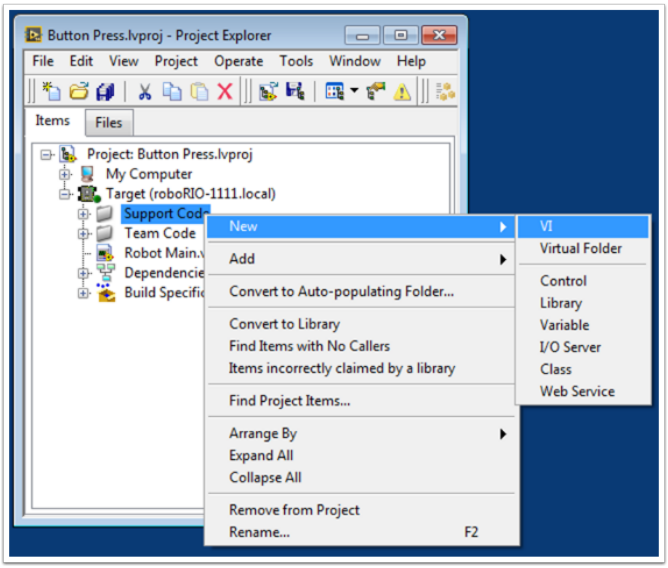


Le deuxième extrait VI est le code qui doit être inclus dans le `Teleop VI`. Il permet de lire la valeur de la chaîne du Dashboard qui indique quelle clé a été pressée. Une Structure case détermine ensuite quelles valeurs doivent être écrites aux moteurs gauche et droit, en fonction de la clé. Dans ce cas, **W** signifie en avant, **A** à gauche, **D** à droite, et **S** inverse. Dans cet exemple, chaque cas opère les moteurs à la moitié de leur vitesse maximale. Vous pouvez conserver cette logique dans votre code, modifier les valeurs ou ajouter du code supplémentaire pour permettre au conducteur d'ajuster la vitesse, de sorte que vous pouvez vous déplacer vite ou lentement au besoin. Une fois que les valeurs à envoyer aux moteurs sont sélectionnées, elles sont transmises aux moteurs d'entraînement et publiées sur Dashboard.

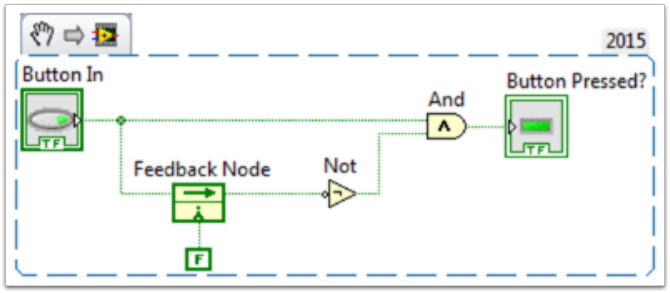
13.2.7 Concevoir un bouton à pression unique

Lorsque vous utilisez la fonction `Joystick Get Values`, en appuyant sur un bouton du joystick, le bouton est lu TRUE jusqu'à ce que le bouton soit libéré. Cela signifie que vous lirez probablement plusieurs valeurs TRUE tant que la pression sur le bouton est maintenue. Que faire si vous voulez lire une seule valeur TRUE chaque fois que l'on maintient la pression sur le bouton ? C'est ce qu'on appelle souvent un « One-Shot Button ». Le didacticiel suivant vous montrera comment créer un subVI que vous pouvez insérer dans votre `Teleop.vi` pour réaliser cette fonctionnalité.

Pour commencer, créez un nouveau VI dans le dossier *Code de support* de votre projet.



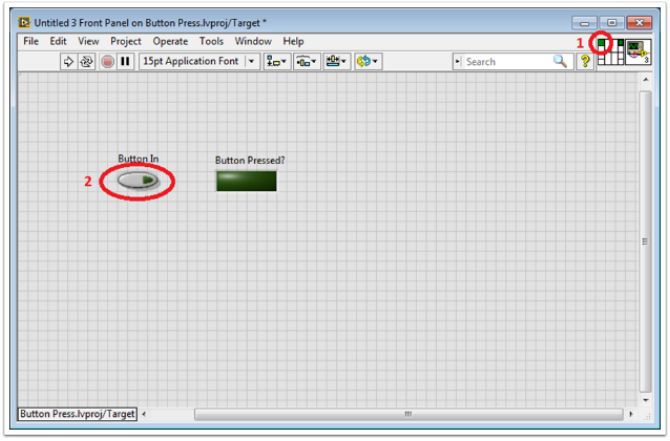
Maintenant, sur le diagramme bloc du nouveau VI, ajouter l'extrait de code suivant.



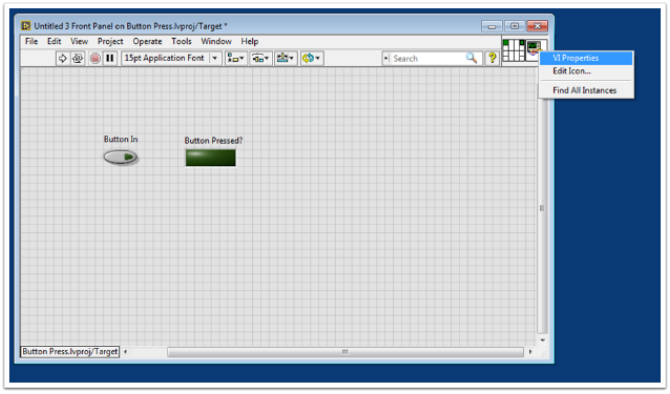
Ce code utilise une fonction appelée `Feedback Node`. Nous avons relié la valeur actuelle du bouton à l'entrée `Feedback Node`. Le fil sortant de la flèche `Feedback Node` représente la valeur précédente du bouton. Si la flèche de votre nœud de rétroaction va dans la direction opposée comme c'est indiqué ici, cliquez avec le bouton droit pour trouver l'option d'inversion de direction.

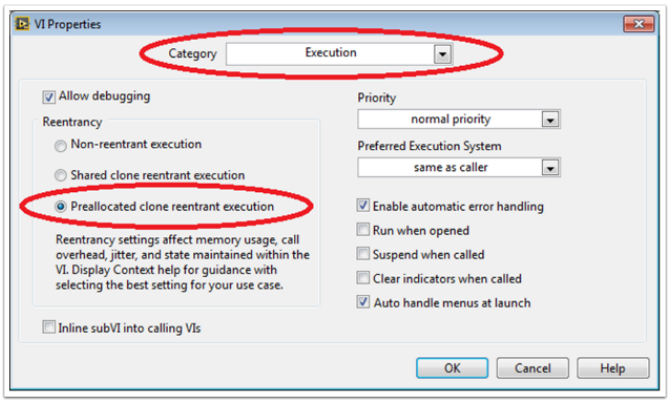
Lorsqu'un bouton est appuyé, la valeur du bouton passe de FALSE à TRUE. Nous voulons que la sortie de ce VI ne soit TRUE que lorsque la valeur actuelle du bouton est TRUE et que la valeur précédente du bouton est FALSE.

Puis, nous devons connecter le contrôle booléen et l'indicateur aux entrées et sorties du VI. Pour ce faire, cliquez d'abord sur le bloc dans le volet connecteur, puis cliquez sur le bouton pour connecter les deux (voir le diagramme ci-dessous). Répétez cette même opération avec l'indicateur.

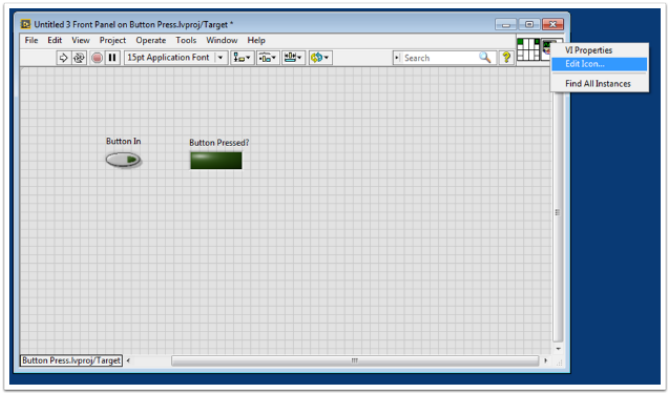


Ensuite, nous devons changer les propriétés de ce VI afin de pouvoir en utiliser plusieurs copies dans notre `TeleOp.vi`. Cliquez avec le bouton droit sur l'icône VI et accédez à `VI Properties`. Sélectionnez ensuite la catégorie « Execution » et sélectionnez « Preallocated clone reentrant execution ».



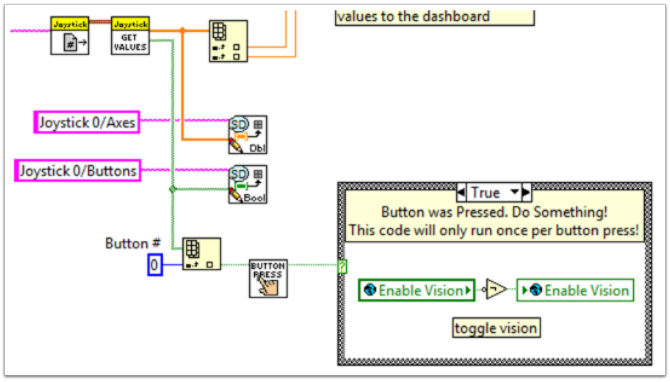


De plus, il serait judicieux de changer l'icône VI pour être plus descriptif de la fonction du VI. Cliquez avec le bouton droit sur l'icône et accédez à Edit Icon. Créez une icône.



Pour terminer, enregistrez le VI avec un nom descriptif. Vous pouvez maintenant faire glisser et déposer ce VI du dossier Fichiers de support dans votre `TeleOp.vi`. Voici une copie du VI : `Button_Press.vi` complété

Voici un exemple vous montrant la façon dont vous pourriez utiliser ce VI.



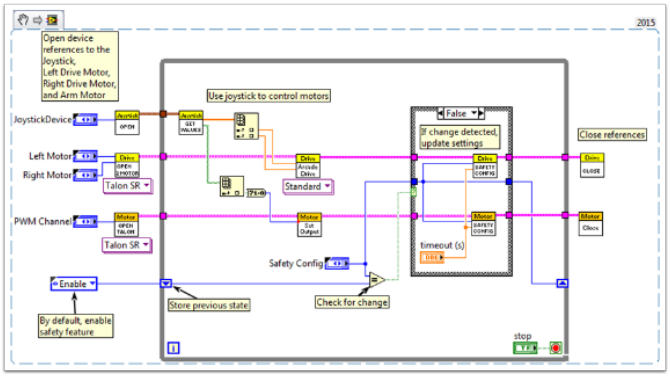
13.2.8 Ajout de fonctionnalités de sécurité à votre code robot

Un problème courant avec les projets complexes est de s'assurer que tout votre code est exécuté quand vous vous y attendez. Des problèmes peuvent survenir lorsque des tâches avec une priorité élevée, de longs temps d'exécution ou des appels fréquents monopolisent la puissance de traitement sur le roboRIO. Cela conduit à ce qu'on appelle « starvation » ou « famine » pour les tâches qui ne sont pas en mesure d'exécuter du que le processeur soit occupé. Dans la plupart des cas, cette situation ralentira simplement le temps de réaction à votre entrée des joysticks et autres dispositifs. Tout aussi grave, cette situation peut également amèner les moteurs de votre robot à conserver des valeurs de vitesse longtemps après que vous ayez envoyé une commande pour les arrêter. Pour éviter toute situation catastrophe en opérant votre robot, vous pouvez implémenter des fonctionnalités de sécurité qui vérifient la famine d'entrée de tâche et arrêter automatiquement les opérations potentiellement nocives.

Il existe des fonctions intégrées pour les moteurs qui permettent une mise en œuvre facile des contrôles de sécurité. Ces fonctions sont les suivantes :

— Robot Drive Safety Configuration
— Motor Drive Safety Configuration
— Relay Safety Configuration
— *PWM* Safety Configuration
— Solenoid Safety Configuration
— Robot Drive Delay and Update Safety

Dans toutes les fonctions configuration de sécurité, vous pouvez activer et désactiver les contrôles de sécurité pendant que votre programme est en cours d'exécution et configurer le délai d'expiration que vous pensez approprié. Les fonctions conservent un cache de tous les appareils dont la sécurité est activée et vérifieront si l'un d'entre eux a dépassé sa limite de temps. Le cas échéant, tous les appareils du cache seront désactivés et le robot s'arrêtera immédiatement ou ses sorties relais/PWM/solenoid seront désactivées. Le code ci-dessous montre comment utiliser les fonctions du *Drive Safety Configuration* pour définir une limite de temps maximale pendant lequel les moteurs ne recevant aucune nouvelle donnée d'entrée devront s'éteindre.



Pour tester l'arrêt de sécurité, essayez d'ajouter et une fonction Wait à la boucle et configuer cette fonction à un temps qui soit plus long que votre délai d'expiration !

La dernière fonction qui se rapporte à la mise en œuvre des contrôles de sécurité – Robot Drive Delay and Update Safety – vous permet de mettre le roboRIO en mode autonome sans dépasser la limite de temps. Il maintiendra la sortie actuelle du moteur sans faire d'appels coûteux aux fonctions de sortie du système d'entrainement, et s'assurera également que les contrôles de sécurité sont régulièrement mis à jour de sorte que les moteurs ne s'arrêteront pas soudainement.

Dans l'ensemble, il est fortement recommandé qu'une certaine forme de contrôle de sécurité

soit mise en œuvre dans votre projet pour s'assurer que votre robot ne soit pas involontairement laissé dans un état dangereux !

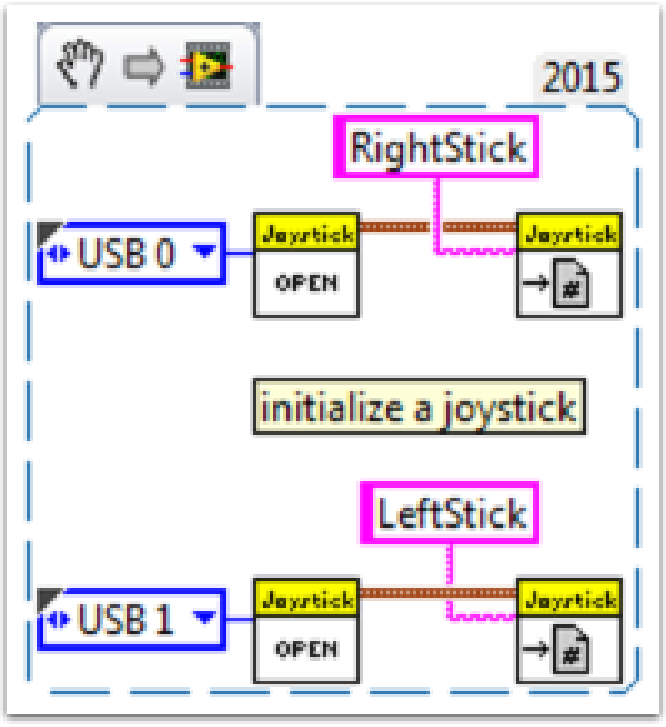
13.2.9 Comment utiliser des boutons du Joystick pour contrôler les moteurs ou les solénoïdes

Comme tous nous avons nos systèmes d'entraînement fonctionnels, nous allons à présent connecter nos périphériques tels que les moteurs et les solénoïdes. À cet effet, nous allons généralement utiliser des boutons de joystick afin de contrôler ces périphériques. Pour commencer, nous allons passer en revue plusieurs façons de contrôler les dispositfs avec des boutons de joystick.

Saviez-vous que vous pouvez cliquer et faire glisser un extrait de code VI d'un document comme celui-ci directement dans votre code LabVIEW ? Essayez-le avec les extraits de ce document.

Configuration :

Quelle que soit la configuration, vous devrez ajouter un, deux ou plusieurs (si vous êtes vraiment motivé) joysticks à l'instrument virtuel « Begin.vi ». Le premier exemple utilise 2 joysticks alors que les autres n'en utilisent qu'un. Donnez à chacun un nom unique afin que vous puissiez l'utiliser à d'autres endroits, comme l'extrait de code ci-dessous. Je les ai nommés « LeftStick » et « RightStick » parce qu'ils sont sur les côtés gauche et droit de mon bureau. Si vos joysticks sont déjà configurés, super ! Vous pouvez sauter cette étape.

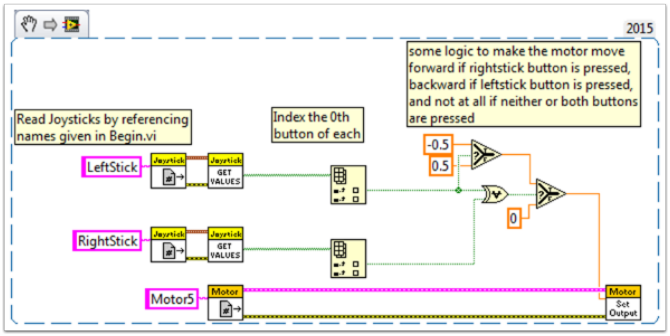


Le reste du code dans ce document sera placé dans le « Teleop.VI ». C'est là que nous allons programmer nos boutons de joystick pour contrôler différents aspects de nos moteurs ou solénoïdes.

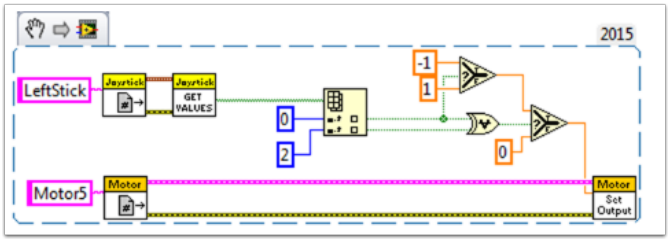
Scénario 1

« Je veux qu'un moteur tourne dans un sens quand j'appuie sur un bouton et l'autre façon quand j'appuie sur un autre bouton. »

Ce code utilise le bouton 0 sur deux joysticks différents pour contrôler le même moteur. Si le bouton 0 sur LeftStick est appuyé, le moteur se déplace vers l'arrière, et si le bouton 0 sur RightStick est appuyé, le moteur se déplace vers l'avant. Si les deux boutons sont appuyés ou si aucun des deux boutons n'est appuyé, le moteur ne bouge pas. Ici, j'ai nommé ma référence moteur « Motor5 », mais vous pouvez nommer votre moteur comme vous le voulez dans le « Begin.vi ».



Vous pouvez utiliser plusieurs boutons à partir du même joystick pour le contrôle. Pour en voir un exemple, regardez l'extrait de code VI suivant ou l'extrait de code VI dans le scénario 2.



Ici, j'ai utilisé les boutons 0 et 2 du joystick, mais sentez-vous à l'aise d'utiliser tous les boutons dont vous avez besoin.

Scénario 2

« Je veux des vitesses de déplacement variées pour différents boutons de joystick. »

Cet exemple pourrait être utile si vous avez besoin d'avoir un même moteur faire différentes choses en fonction des boutons que vous appuyez. Par exemple, disons que mon joystick a un déclencheur (bouton 0) et 4 boutons par dessus (boutons 1 à 4). Dans ce cas, les boutons suivants peuvent avoir les fonctions suivantes :

- bouton 1 - recule à mi-vitesse
- bouton 2 - avance à mi-vitesse
- bouton 3 - recule à 1/4 de vitesse
- bouton 4 - avance à 1/4 de vitesse
- déclencheur - pleine vitesse en avant ! (en avant à pleine vitesse)

Nous prenons ensuite le tableau booléen à partir de « JoystickGetValues.vi » et on le relie à un nœud « Boolean Array to Number » (Numeric Palette-Conversion Palette). Ce qui convertit le tableau booléen en un nombre que nous pouvons alors utiliser. Relier ce nombre à une structure case.

Chaque cas correspond à une représentation binaire des valeurs du tableau. Dans cet exemple, chaque cas correspond à une combinaison d'un seul bouton. Nous avons ajouté six cas : 0 (tous les boutons sont à OFF), 1 (bouton 0 sur ON), 2 (bouton 1 sur ON), 4 (bouton 2 sur ON), 8 (bouton 3 sur ON), et 16 (bouton 4 sur ON). Notez que nous avons sauté la valeur 3. 3 correspondrait aux boutons 0 et 1 appuyés en même temps. Nous ne l'avons pas défini dans nos requis, donc nous allons laisser le cas par défaut le gérer.

Il pourrait être utile d'examiner le document d'aide sur la structure case LabVIEW 2014 ici :

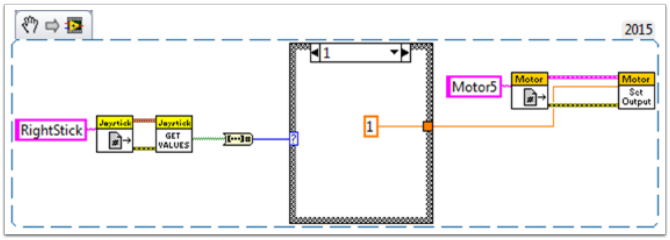
https://zone.ni.com/reference/en-XX/help/371361L-01/glang/case_structure/

Il ya aussi 3 tutoriels communautaires sur les structures case ici :

https://forums.ni.com/t5/Curriculum-and-Labs-for/Unit-3-Case-Structures-Lesson-1/ta-p/3505945?profile.language=en

https://forums.ni.com/t5/Curriculum-and-Labs-for/Unit-3-Case-Structures-Lesson-2/ta-p/3505933?profile.language=en

https://forums.ni.com/t5/Curriculum-and-Labs-for/Unit-3-Case-Structures-Lesson-3/ta-p/3505979?profile.language=en

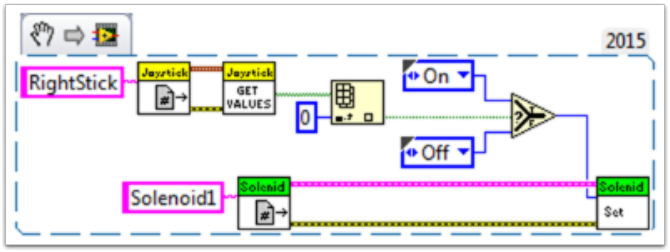


Comme nos requis étaient simples, nous n'avons besoin que d'une seule constante dans chaque cas. Pour le cas 1 (en avant toutes) nous utilisons un 1, pour le cas 2 (en arrière à moitié) nous utilisons un -0,5, etc. Nous pouvons utiliser n'importe quelle valeur constante entre 1 et -1. J'ai laissé le cas 0 comme cas par défaut, donc si plusieurs boutons sont appuyés (n'importe quel état indéfini a été atteint) le moteur s'arrêtera. Vous êtes bien sûr libre de personnaliser ces états comme vous le souhaitez.

Scénario 3

« Je veux contrôler un solénoïde à l'aide des boutons de mon joystick. »

Maintenant, nous sommes familiers avec la façon dont on peut regrouper les boutons d'un joystick dans un tableau de booleans. Nous devons indexer ce tableau pour obtenir le bouton qui nous intéresse, et relier ce boolean à un nœud sélectionné. Étant donné que le « Solenoid Set.vi » nécessite un Enum comme entrée, la façon la plus simple d'obtenir l'enum est de cliquer avec le bouton droit sur l'entrée « Value » de « Solenoid Set.vi » et sélectionnez « Create Constant ». Dupliquez cette constante et relier une copie avec le terminal True et une autre avec le terminal False du nœud sélectionné. Ensuite, relier la sortie du nœud sélectionné avec l'entrée « Value » du solenoid VI.



Ayez du plaisir en robotique !

13.2.10 Variables locales et globales en LabVIEW pour FRC

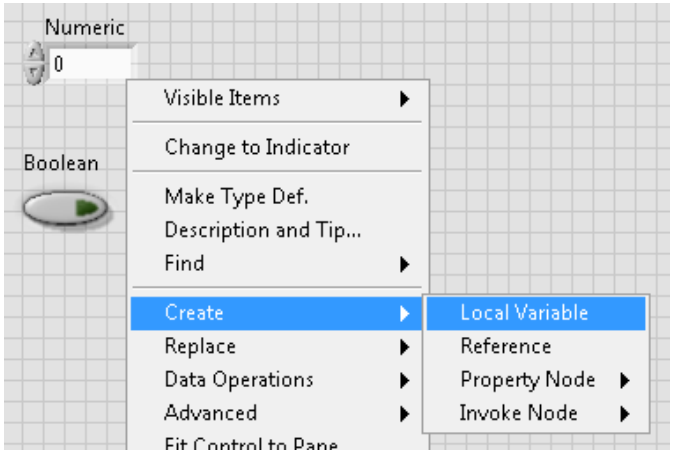
Cet exemple sert d'introduction aux variables locales et globales, comment elles sont utilisées dans un projet LabVIEW FRC® par défaut, et comment vous pouvez les utiliser dans votre projet.

Les variables locales et les variables globales peuvent être utilisées pour transférer des données entre les emplacements à l'intérieur du même VI (variables locales) ou dans différents VI's (variables globales), brisant le conventionnel Paradigme de flux de données pour lequel LabVIEW est reconnu. Ainsi, ils peuvent être utiles lorsque, pour une raison quelconque, vous ne pouvez pas transférer la valeur directement sur le nœud à un autre nœud.

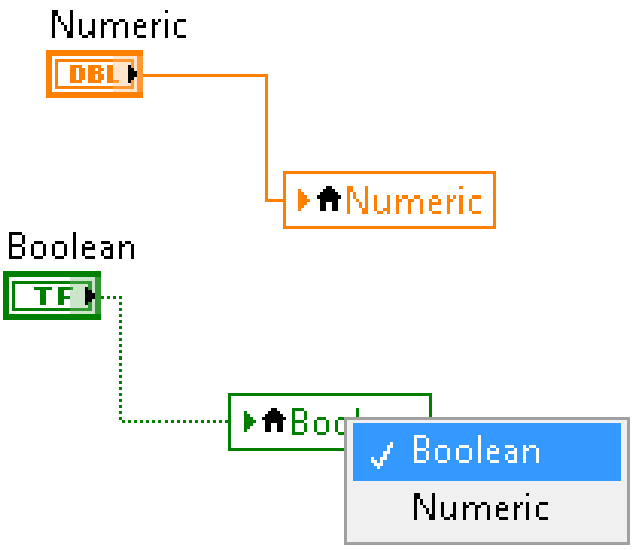
Note : Une raison possible peut être que vous devez transmettre des données entre les itérations consécutives d'une boucle ; Miro_T l'a couvert dans cette publication. Il convient également de noter que le nœud de rétroaction dans LabVIEW peut être utilisé comme un équivalent au registre à décalage, bien que cela puisse être un sujet pour un autre jour !

Introduction aux variables locales et globales

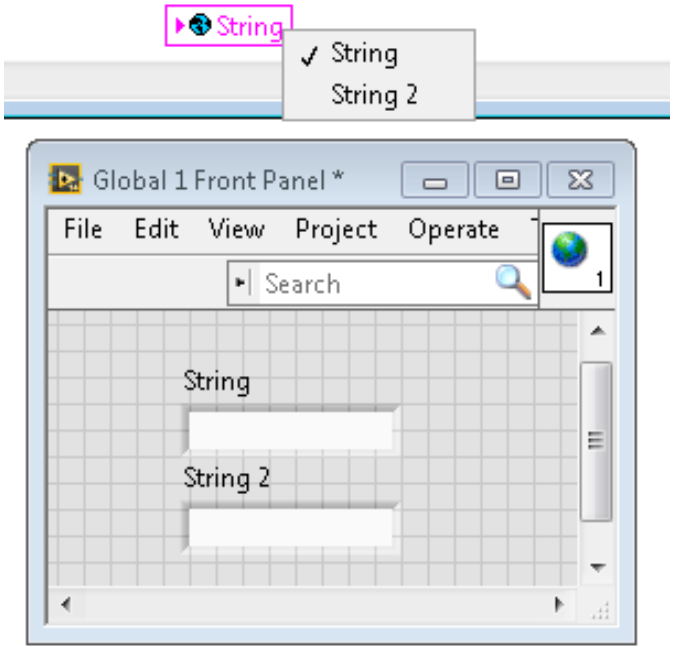
Les variables locales peuvent être utilisées dans le même VI. Créez une variable locale en cliquant avec le bouton droit sur un contrôle ou un indicateur sur votre panneau frontal :



Vous pouvez également créer une variable locale à partir de la Structures palette du diagramme bloc. Lorsque vous avez plusieurs variables locales dans un VI, vous pouvez cliquer à gauche pour choisir la bonne variable :



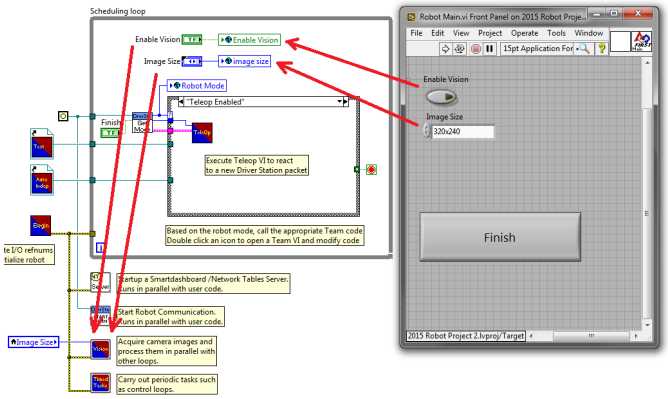
Les variables globales sont créées de manière légèrement différente. Ajoutez-y un diagramme bloc à partir de Structures palette et remarquez que lorsque vous cliquez deux fois, il ouvre un panneau avant séparé. Ce panneau avant n'a pas de diagramme bloc, mais vous pouvez ajouter autant d'entités au panneau avant que vous souhaitez et enregistrez-le sous forme de fichier *.vi :



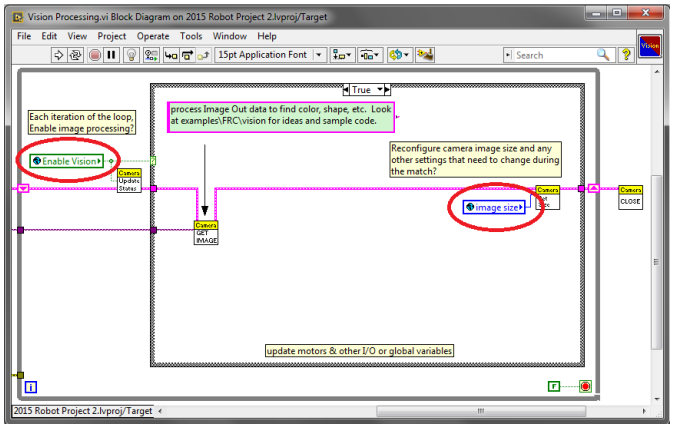
Note : Soyez très prudent afin d'éviter les conditions de course lorsque vous utilisez des variables locales et globales ! Essentiellement, assurez-vous que vous n'écrivez pas accidentellement à la même variable dans plusieurs endroits sans avoir un moyen de savoir à quel endroit elle a été écrite pour la dernière fois. Pour une explication plus approfondie, consulter ce document d'aide

Comment elles sont utilisées dans un projet de robot FRC LabVIEW par défaut

Des variables globales pour « Enable Vision » et « Image Size » sont écrites à chaque itération de Robot Main VI…



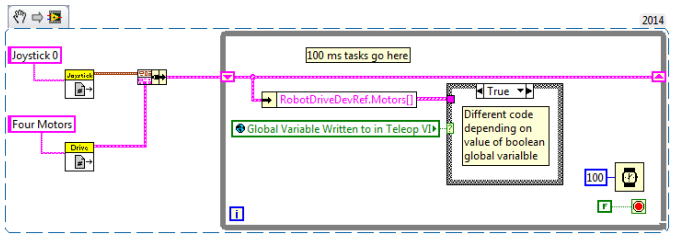
… Et puis lire à chaque itération de l'instrument virtuel Vision Processing VI :



Ce qui permet à l'utilisateur, lors du déploiement sur Robot Main VI à partir de l'environnement de développement LabVIEW, d'activer/désactiver la vision et de modifier la taille de l'image à partir du panneau avant de Robot Main.

Comment pouvez-vous les utiliser dans votre projet ?

Consultez le diagramme bloc de l'instrument virtuel Periodic Tasks VI. Il existe peut-être y a-t-il une valeur, par exemple un booléen, qui peut être écrite à une variable globale dans le Teleop VI, puis lue dans Periodic Tasks VI. Vous pouvez ensuite décider quel code ou valeurs utiliser dans Periodic Tasks VI, selon la variable globale booléenne :



13.2.11 Utilisation du compresseur en LabVIEW

Cet document montre comment configurer votre projet roboRIO pour utiliser le module de contrôle pneumatique (PCM). Le PCM démarre et arrête automatiquement le compresseur lorsque des pressions spécifiques sont mesurées dans le réservoir. Dans votre programme roboRIO, vous devrez ajouter les VI suivants.

Pour plus d'informations, consultez les liens suivants :

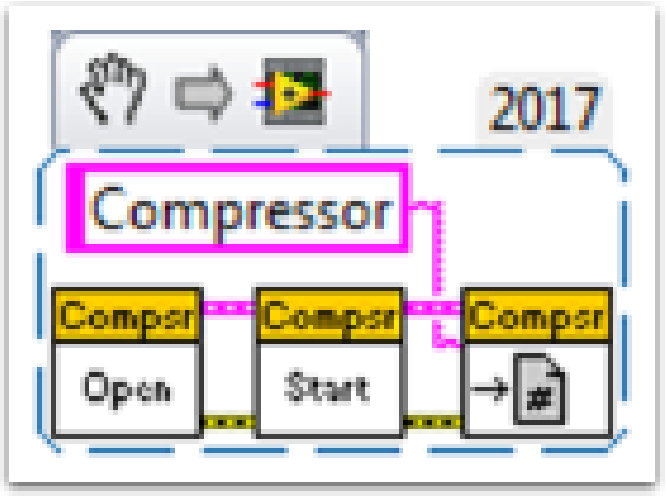
`FRC Pneumatics Manual`

Manuel d'utilisation du PCM

Pneumatiques pour le roboRIO étape par étape

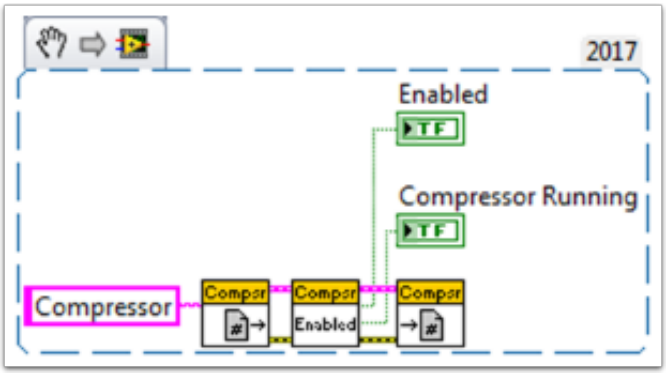
Begin VI

Placez cet extrait de code dans le Begin.vi.



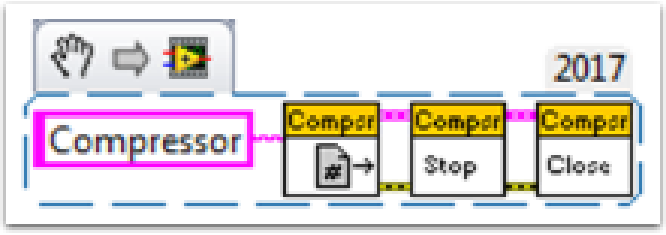
Teleop VI

Placez cet extrait de code dans Teleop.vi. Cette partie n'est requise que si vous utilisez les sorties pour d'autres processus.



Finish VI

Placez cet extrait de code dans le cadre Close Refs, enregistrez les données, etc. de Finish.vi.



14 FRC Python Programming

14.1 pyproject.toml usage

Note : RobotPy projects are not required to have a `pyproject.toml`, but when you run `robotpy sync` one will automatically be created for you.

`pyproject.toml` has become a standard way to store build and tooling configuration for Python projects. The `[tool.XXX]` section(s) of the TOML file is a place where tools can store their configuration information.

Currently RobotPy only stores deployment related information in `pyproject.toml`, in the `[tool.robotpy]` section. Users can customize the other sections however they want, and `robotpy` will ignore them.

The `pyproject.toml` file looks something like this :

```
#
# Use this configuration file to control what RobotPy packages are installed
# on your RoboRIO
#

[tool.robotpy]

# Version of robotpy this project depends on
robotpy_version = "2024.2.1.0"

# Which extra RobotPy components should be installed
# -> equivalent to `pip install robotpy[extra1, ...]
robotpy_extras = [
    # "all"
    # "apriltag"
    # "commands2"
    # "cscore"
    # "navx"
    # "pathplannerlib"
    # "phoenix5"
    # "phoenix6"
```

(suite sur la page suivante)

(suite de la page précédente)

```
    # "playingwithfusion"
    # "rev"
    # "romi"
    # "sim"
]

# Other pip packages to install
requires = []
```

Each of the following will instruct the deploy process to install packages to the roboRIO :

`robotpy_version` is the version of the `robotpy` PyPI package that this robot code depends on.

`robotpy_extras` defines extra RobotPy components that can be installed, as only the core RobotPy libraries are installed by default.

`requires` is a list of strings, and each item is equivalent to a line of a requirements.txt file. You can install any pure python packages on the roboRIO and they will likely work, but any packages that have binary dependencies must be cross-compiled for the roboRIO. For example, if you needed to use `numpy` in your robot code :

```
[tool.robotpy]

...

requires = ["numpy"]
```

The packages that can be installed are stored on the WPILib Artifactory server. If you find that you need a package that isn't available on artifactory, consult the roborio-wheels repository.

14.2 RobotPy subcommands

When you install RobotPy in your Python installation, it installs a package called `robotpy-cli`, which provides a `robotpy` command that can be used to perform tasks related to your robot and related code.

If you execute the command from the command line, it will show the various subcommands that are available :

Windows

```
py -3 -m robotpy
```

macOS

```
python3 -m robotpy
```

Linux

```
python3 -m robotpy
```

Note : If you don't see a list of commands but either see a RobotPy logo or an error saying `No module named robotpy.__main__; 'robotpy' is a package and cannot be directly executed`, you should uninstall the `robotpy` module and then reinstall it via pip.

This only affects users who upgraded from pre-2024 or the 2024 beta.

You can pass the `--help` argument to see more information about the subcommand. For example, to see help for the `sim` command you can do the following :

Windows

```
py -3 -m robotpy sim --help
```

macOS

```
python3 -m robotpy sim --help
```

Linux

```
python3 -m robotpy sim --help
```

This page has more detailed documentation for some of the subcommands :

14.2.1 Deploy Python program to roboRIO

Note : Before deploying the code to your robot, you must start by *installing RobotPy on your computer*

In particular, it is expected that you have ran `robotpy sync` to download all of the roboRIO python dependencies.

Windows

```
py -3 -m robotpy deploy
```

macOS

```
python3 -m robotpy deploy
```

Linux

```
python3 -m robotpy deploy
```

When you execute the `robotpy deploy` subcommand, it will do the following :

- Run `pytest` tests on your code (will exit if they fail)
- Install Python on the roboRIO (if not already present)
- Install python packages on the roboRIO as specified by your `pyproject.toml` (if not already present)
- Copy the entire robot project directory to the roboRIO and execute it

Avertissement : Avoid powering off the robot while deploying robot code. Interrupting the deployment process can corrupt the roboRIO filesystem and prevent your code from working until the roboRIO is *re-imaged*.

When successful, you will see a `SUCCESS: Deploy was successful!` message.

You can watch your robot code's output (and see any problems) with netconsole by using the Driver Station Log Viewer or pynetconsole. You can use netconsole and the normal FRC tools to interact with the running robot code.

Voir aussi :

Affichage de la sortie de la console

Immediate feedback via Netconsole

When deploying the code to the roboRIO, you can have immediate feedback by adding the option *–nc*. This will cause the deploy command to show your program's console output, by launching a netconsole listener.

Windows

```
py -3 -m robotpy deploy --nc
```

macOS

```
python3 -m robotpy deploy --nc
```

Linux

```
python3 -m robotpy deploy --nc
```

Note : Viewing netconsole output requires the driver station software to be connected to your robot

---

Skipping Tests

In the event that the tests are failing but you want to upload the code anyway, you can skip them by adding the option *–skip-tests*.

Windows

```
py -3 -m robotpy deploy --skip-tests
```

macOS

```
python3 -m robotpy deploy --skip-tests
```

Linux

```
python3 -m robotpy deploy --skip-tests
```

15 API du Matériel

Cette section traite du contrôle des moteurs et de la pneumatique via les contrôleurs de moteur, les solénoïdes et la pneumatique, et leur interface avec Java et C++ WPILib.

15.1 APIs pour moteurs

La programmation de vos moteurs est absolument indispensable pour que votre robot bouge ! Cette section présente des cours et des exemples utiles pour faire démarrer et déplacer votre robot !

15.1.1 Utilisation des contrôleurs de moteurs dans le code

Motor controllers come in two main flavors : *CAN* and *PWM*. A CAN controller can send more detailed status information back to the roboRIO, whereas a PWM controller can only be set to a value. For information on using these motors with the WPILib drivetrain classes, see *Utilisation des classes WPILib pour piloter votre robot*.

Utilisation des contrôleurs de moteurs PWM

PWM motor controllers can be controlled in the same way as a CAN motor controller. For a more detailed background on *how* they work, see *Contrôleurs de moteurs PWM en profondeur*. To use a PWM motor controller, simply use the appropriate motor controller class provided by WPILib and supply it the port the motor controller(s) are plugged into on the roboRIO. All approved motor controllers have WPILib classes provided for them.

Note : The `Spark` and `VictorSP` classes are used here as an example ; other PWM motor controller classes have exactly the same API.

JAVA

```
Spark spark = new Spark(0); // 0 is the RIO PWM port this is connected to

spark.set(-0.75); // the % output of the motor, between -1 and 1

VictorSP victor = new VictorSP(0); // 0 is the RIO PWM port this is connected to

victor.set(0.6); // the % output of the motor, between -1 and 1
```

C++

```
frc::Spark spark{0}; // 0 is the RIO PWM port this is connected to

spark.Set(-0.75); // the % output of the motor, between -1 and 1

frc::VictorSP victor{0}; // 0 is the RIO PWM port this is connected to

victor.Set(0.6); // the % output of the motor, between -1 and 1
```

PYTHON

```
spark = wpilib.Spark(0) # 0 is the RIO PWM port this is connected to

spark.set(-0.75) # the % output of the motor, between -1 and 1

victor = wpilib.VictorSP(0) # 0 is the RIO PWM port this is connected to

victor.set(0.6) # the % output of the motor, between -1 and 1
```

Contrôleurs de moteurs CAN

A handful of CAN motor controllers are available through vendors such as CTR Electronics, REV Robotics, and Playing with Fusion. See *Périphériques CAN provenant de tierce-partie*, *Librairies tierces*, and *Exemples de projets tiers* for more information.

15.1.2 Contrôleurs de moteurs PWM en profondeur

Indication : WPILib dispose d'un soutien important pour le contrôle moteur. Il existe un certain nombre de classes qui représentent différents types de contrôleurs de moteurs et servomoteurs. Il existe actuellement deux catégories principales de contrôleurs moteurs : les contrôleurs moteurs de type PWM et les contrôleurs moteurs de type CAN. WPILib contient également des classes composites (comme DifferentialDrive) qui vous permettent de contrôler plusieurs moteurs avec un seul objet. Cet article couvre en détails les contrôleurs moteurs PWM ; Les contrôleurs CAN et les classes composites seront couverts par des articles distincts.

Contrôleurs PWM, la théorie du fonctionnement

The acronym *PWM* stands for Pulse Width Modulation. For motor controllers, PWM can refer to both the input signal and the method the controller uses to control motor speed. To control the speed of the motor the controller must vary the perceived input voltage of the motor. To do this the controller switches the full input voltage on and off very quickly, varying the amount of time it is on based on the control signal. Because of the mechanical and electrical time constants of the types of motors used in FRC® this rapid switching produces an effect equivalent to that of applying a fixed lower voltage (50% switching produces the same effect as applying ~6V).

Le signal PWM que les contrôleurs utilisent comme entrée est un peu différent. Même aux limites de la plage du signal (avancer ou reculer au maximum), le signal n'approche jamais d'un rapport cyclique de 0% ou 100%. Au lieu de cela, les contrôleurs utilisent un signal avec une période de 5 mSec ou 10 mSec et une largeur d'impulsion médiane de 1,5 mSec, qui correspond à l'arrêt du moteur. En faisant varier la largeur de cette impulsion entre 1 mSec et 2 mSec, on fait varier la vitesse du moteur entre la marche arrière maximum et la marche avant maximum.

Valeurs de sortie brutes vs mises à l'échelle

In general, all of the motor controller classes in WPILib take a scaled -1.0 to 1.0 value as the output to an actuator. The PWM module in the FPGA on the roboRIO is capable of generating PWM signals with periods of 5, 10, or 20ms and can vary the pulse width in 4096 steps of 1us each . The raw values sent to this module are in this 0-4096 range with 0 being a special case which holds the signal low (disabled). The class for each motor controller contains information about what the typical bound values (min, max and each side of the deadband) are as well as the typical midpoint. WPILib can then use these values to map the scaled value into the proper range for the motor controller. This allows for the code to switch seamlessly between different types of controllers and abstracts out the details of the specific signaling.

Étalonnage des contrôleurs de moteurs

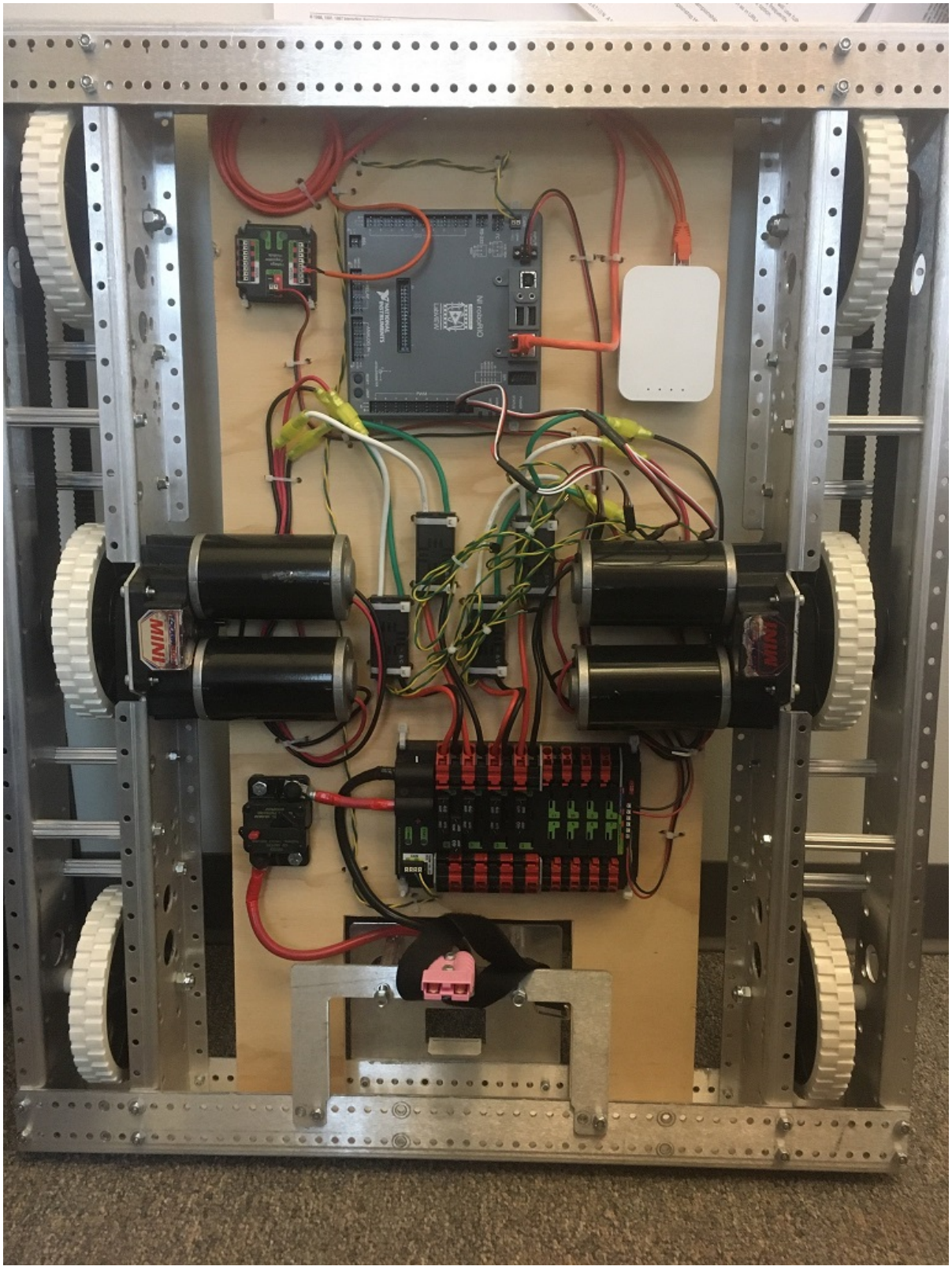
Donc, si WPILib gère toute cette mise à l'échelle, pourquoi auriez-vous alors besoin de calibrer votre contrôleur de moteur ? Les valeurs que WPILib utilise pour la mise à l'échelle sont des valeurs approximatives basées sur la mesure d'un certain nombre d'échantillons de chaque type de contrôleur. En raison d'une variété de facteurs, la synchronisation d'un contrôleur de moteur individuel peut varier légèrement. Afin d'éliminer définitivement le « bourdonnement » (signal médian interprété comme un léger mouvement dans une direction) et de conduire le contrôleur jusqu'à chaque extrême, il est toujours recommandé d'étalonner les contrôleurs. En général, la procédure d'étalonnage pour chaque contrôleur consiste à mettre le contrôleur en mode étalonnage, puis à diriger le signal d'entrée à chacune de ces extrémités, puis à revenir au point médian. Pour avoir des exemples relatifs à la façon d'utiliser ces contrôleurs de moteurs dans votre code, consulter *Using Motor Controllers in Code/Using PWM Motor Controllers*

15.1.3 Utilisation des classes WPILib pour piloter votre robot

WPILib comprend de nombreuses classes pour faciliter le développement de la programmation et obtenir un robot fonctionnel plus rapidement.

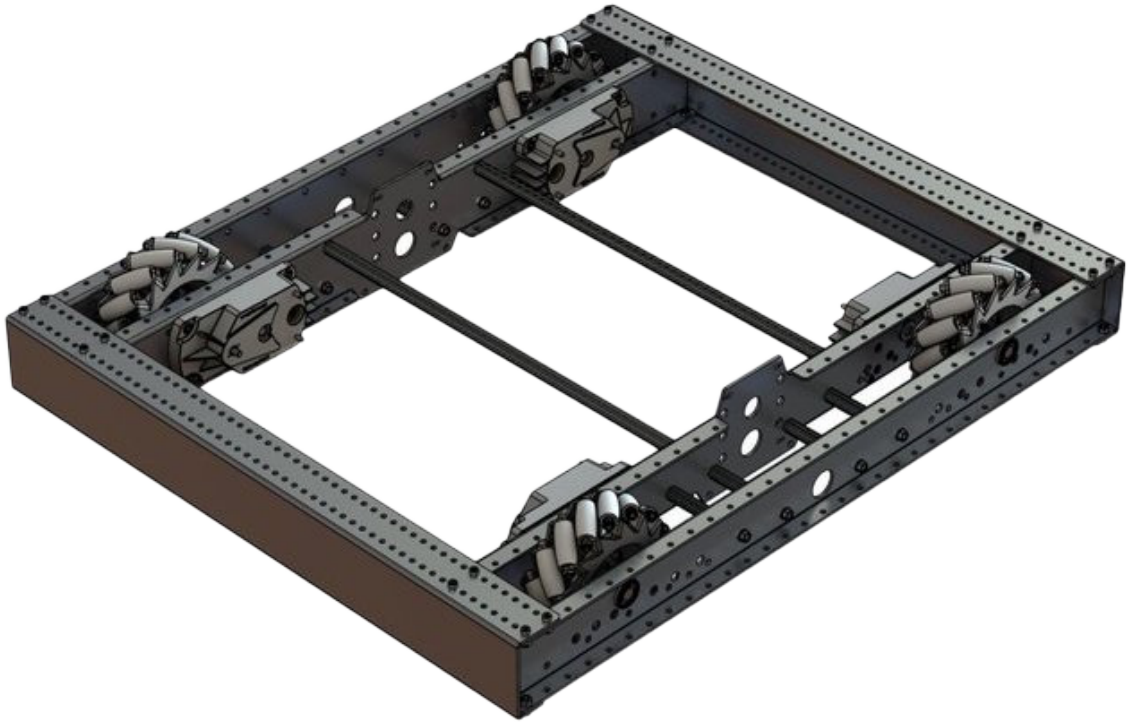
Transmissions standard

Robots à entraînement différentiel



Ces bases d'entraînement ont généralement deux ou plusieurs roues motrices en ligne ou omnidirectionnelles par côté (par exemple, 6WD ou 8WD) et peuvent également être appelées « skid-steer », « tank drive » ou « West Coast Drive ». La transmission du Kit de pièces est un exemple d'entraînement différentiel. Ces transmissions sont capables de faire avancer ou reculer le robot et peuvent le faire tourner en activant les moteurs de chaque côté dans des directions opposées, ce qui fait déraper les roues latéralement. Ces transmissions ne sont pas capables d'exécuter des mouvements de translation latéraux.

Entraînement de type Mécanum



L'entraînement Mécanum est une méthode de conduite utilisant des roues spécialement conçues qui permettent au robot de conduire dans n'importe quelle direction sans changer l'orientation du robot. Un robot avec une transmission conventionnelle (toutes les roues pointant dans la même direction) doit tourner dans la direction qu'il doit conduire. Un robot Mécanum peut se déplacer dans n'importe quelle direction sans avoir à tourner au préalable et est appelé un entraînement holonomique. Les roues (montrées sur ce robot) ont des rouleaux qui font que les forces de la conduite sont appliquées à un angle de 45 degrés plutôt que droit comme dans le cas d'un entraînement conventionnel.

Lorsqu'ils sont vus du haut, les rouleaux d'une transmission Mécanum doivent former un motif en "X". Il en résulte que les vecteurs de force (lors de la conduite de la roue vers l'avant) sur les deux roues avant pointent vers l'avant et vers l'intérieur et les deux roues arrière pointent vers l'avant et vers l'extérieur. En faisant tourner les roues dans différentes directions, divers composants des vecteurs de force s'annulent, entraînant le mouvement souhaité du robot. Un tableau rapide des différents mouvements a été fourni ci-dessous, en traçant les vecteurs de force pour chacun de ces mouvements peut aider à comprendre comment ces transmissions fonctionnent. En faisant varier les vitesses des roues en plus de la direction, les mouvements peuvent être combinés, ce qui entraîne une translation dans n'importe quelle direction et rotation, simultanément.

Conventions utilisées par les classes

Motor Inversion

Depuis 2022, le côté droit de la transmission n'est **plus** inversé par défaut. Il est de la responsabilité de l'utilisateur de gérer les inversions appropriées de sa transmission. Les utilisateurs peuvent inverser les moteurs en appelant `setInverted()`/`SetInverted()` sur leurs objets moteurs.

JAVA

```
PWMSparkMax m_motorRight = new PWMSparkMax(0);

@Override
public void robotInit() {
   m_motorRight.setInverted(true);
}
```

C++

```
frc::PWMSparkMax m_motorLeft{0};

public:
 void RobotInit() override {
   m_motorRight.SetInverted(true);
 }
```

PYTHON

```
def robotInit(self):
    self.motorRight = wpilib.PWMSparkMax(0)
    self.motorRight.setInverted(True)
```

Squaring Inputs

Lors de la conduite de robots, il est souvent souhaitable de manipuler les entrées du joystick de sorte que le robot ait un contrôle plus précis à basse vitesse tout en utilisant toute la plage de sortie disponible. Une façon d'y parvenir est d'élever au carré la valeur absolue de l'entrée du joystick, puis de réappliquer le signe. Par défaut, la classe Differential Drive mettra les entrées au carré. Si cela n'est pas souhaité (par exemple, si vous transmettez des valeurs à partir d'un PIDController), utilisez l'une des méthodes d'entraînement et initialiser le paramètre squaredInputs à False.

Input Deadband

Par défaut, la classe DifferentialDrive applique une zone morte d'entrée de 0.02. Cela signifie que toutes les valeurs d'entrée qui sont inférieures à 0.02 (après la mise au carré décrite ci-dessus) seront plutôt considérées comme ayant une valeur de zéro. Dans la plupart des cas, ces petites valeurs d'entrée résultent d'un centrage imparfait du joystick et ne sont pas suffisantes pour provoquer un mouvement du robot. En appliquant une zone morte, on réduit l'échauffement des moteurs provenant de l'énergie appliquée inutilement aux moteurs, lorsque le robot ne bouge pas. Pour modifier la zone morte, utilisez la méthode `setDeadband()`.

Maximum Output

Parfois, les pilotes ont l'impression que leur transmission roule trop vite et veulent limiter la sortie. Cela peut être accompli avec la méthode `setMaxOutput()`. Cette sortie maximale est multipliée par le résultat des fonctions d'entraînement précédentes comme deadband et squared inputs.

La sécurité moteur (Motor Safety)

Motor Safety est un mécanisme dans WPILib qui prend le concept d'un chien de garde (Watchdog) et crée une minuterie de sécurité pour chaque dispositif actionneur qui contrôle un moteur. Notez que ce mécanisme de protection s'ajoute au System Watchdog, celui-ci étant contrôlé par le code de communication réseau et le FPGA. Le Watchdog désactivera toutes les sorties du dispositif actionneur s'il ne reçoit pas de paquet de données valides pendant un délai de 125 ms.

The purpose of the Motor Safety mechanism is the same as the purpose of a watchdog timer, to disable mechanisms which may cause harm to themselves, people or property if the code locks up and does not properly update the actuator output. Motor Safety breaks this concept out on a per actuator basis so that you can appropriately determine where it is necessary and where it is not. Examples of mechanisms that should have motor safety enabled are systems like drive trains and arms. If these systems get latched on a particular value they could cause damage to their environment or themselves. An example of a mechanism that may not need motor safety is a spinning flywheel for a shooter. If this mechanism gets latched on a particular value it will simply continue spinning until the robot is disabled. By default Motor Safety is enabled for DifferentialDrive and MecanumDrive objects and disabled for all other motor controllers and servos.

La fonction Motor Safety d'un actionneur fonctionne en maintenant un timer qui surveille et enregistre le temps qui s'est écoulé depuis que le dernier appel de la méthode `feed()` pour cet actionneur. Un code dans la classe DriverStation initie une comparaison de ces timers aux valeurs de délai d'attente pour tout actionneur avec la sécurité activé tous les 5 paquets reçus (100ms nominal). Les méthodes set() de chaque classe de contrôleur de moteur et les méthodes `set()` et `setAngle()` de la classe servo font un appel à la méthode `feed()` pour indiquer que la sortie de l'actionneur a été mise à jour.

L'interface Motor Safety des contrôleurs moteurs peut être en interaction avec l'utilisateur par l'intermédiaire des méthodes suivantes :

JAVA

```
m_motorRight.setSafetyEnabled(true);
m_motorRight.setSafetyEnabled(false);
m_motorRight.setExpiration(.1);
m_motorRight.feed()
```

C++

```
m_motorRight->SetSafetyEnabled(true);
m_motorRight->SetSafetyEnabled(false);
m_motorRight->SetExpiration(.1);
m_motorRight->Feed();
```

PYTHON

```
m_motorRight.setSafetyEnabled(True)
m_motorRight.setSafetyEnabled(False)
m_motorRight.setExpiration(.1)
m_motorRight.feed()
```

Par défaut, tous les objets Drive activent la sécurité moteur. Selon le mécanisme et la structure de votre programme, vous pouvez souhaiter configurer la durée de temporisation de la sécurité moteur (en secondes). La durée du timeout est configurée par actionneur et n'est pas un paramètre global. La valeur par défaut (et minimale utile) est de 100 ms.

Conventions des axes

The drive classes use the NWU axes convention (North-West-Up as external reference in the world frame). The positive X axis points ahead, the positive Y axis points left, and the positive Z axis points up. We use NWU here because the rest of the library, and math in general, use NWU axes convention.

Joysticks follow NED (North-East-Down) convention, where the positive X axis points ahead, the positive Y axis points right, and the positive Z axis points down. However, it's important to note that axes values are rotations around the respective axes, not translations. When viewed with each axis pointing toward you, CCW is a positive value and CW is a negative value. Pushing forward on the joystick is a CW rotation around the Y axis, so you get a negative value. Pushing to the right is a CCW rotation around the X axis, so you get a positive value.

Note : See the *Coordinate System* section for more detail about the axis conventions and coordinate systems.

Utilisation de la classe DifferentialDrive pour contrôler les robots à entraînement différentiel (WCD)

Note : WPILib provides separate Robot Drive classes for the most common drive train configurations (differential and mecanum). The DifferentialDrive class handles the differential drivetrain configuration. These drive bases typically have two or more in-line traction or omni wheels per side (e.g., 6WD or 8WD) and may also be known as « skid-steer », « tank drive », or « West Coast Drive » (WCD). The Kit of Parts drivetrain is an example of a differential drive. There are methods to control the drive with 3 different styles (« Tank », « Arcade », or « Curvature »), explained in the article below.

DifferentialDrive est une méthode fournie pour le contrôle des transmissions « skid-steer » ou « West Coast », comme le châssis du Kit de pièces. L'instanciation d'un DifferentialDrive est aussi simple que cela :

Java

```
public class Robot extends TimedRobot {
  private DifferentialDrive m_robotDrive;
  private final PWMSparkMax m_leftMotor = new PWMSparkMax(0);
  private final PWMSparkMax m_rightMotor = new PWMSparkMax(1);

  @Override
  public void robotInit() {
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side␣
↪instead.
    m_rightMotor.setInverted(true);

    m_robotDrive = new DifferentialDrive(m_leftMotor::set, m_
↪rightMotor::set);
  }
```

C++ (Header ou en-tête)

```
  frc::PWMSparkMax m_leftMotor{0};
  frc::PWMSparkMax m_rightMotor{1};
  frc::DifferentialDrive m_robotDrive{
      [&](double output) { m_leftMotor.Set(output); },
      [&](double output) { m_rightMotor.Set(output); }};
```

C++ (Source)

```
  void RobotInit() override {
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side␣
↪instead.
    m_rightMotor.SetInverted(true);
  }
```

Python

```
    def robotInit(self):
        """Robot initialization function"""

        leftMotor = wpilib.PWMSparkMax(0)
        rightMotor = wpilib.PWMSparkMax(1)
        self.robotDrive = wpilib.drive.DifferentialDrive(leftMotor,␣
↪rightMotor)
        # We need to invert one side of the drivetrain so that positive␣
↪voltages
        # result in both sides moving forward. Depending on how your robot's
        # gearbox is constructed, you might have to invert the left side␣
↪instead.
        rightMotor.setInverted(True)
```

Multi-Motor DifferentialDrive

Many FRC® drivetrains have more than 1 motor on each side. Classes derived from `PWMMotorController` (Java / C++ / Python) have an `addFollower` method so that multiple follower motor controllers can be updated when the leader motor controller is commanded. CAN motor controllers have similar features, review the vendor's documention to see how to use them. The examples below show a 4 motor (2 per side) drivetrain. To extend to more motors, simply create the additional controllers and use additional `addFollower` calls.

Java

Class variables (e.g. in Robot.java or Subystem) :

```
  // The motors on the left side of the drive.
  private final PWMSparkMax m_leftLeader = new PWMSparkMax(DriveConstants.
↪kLeftMotor1Port);
  private final PWMSparkMax m_leftFollower = new PWMSparkMax(DriveConstants.
↪kLeftMotor2Port);

  // The motors on the right side of the drive.
  private final PWMSparkMax m_rightLeader = new PWMSparkMax(DriveConstants.
↪kRightMotor1Port);
  private final PWMSparkMax m_rightFollower = new PWMSparkMax(DriveConstants.
↪kRightMotor2Port);
```

In robotInit or Subsystem constructor :

```
    m_leftLeader.addFollower(m_leftFollower);
    m_rightLeader.addFollower(m_rightFollower);

    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side␣
↪instead.
    m_rightLeader.setInverted(true);
```

C++ (Header ou en-tête)

```
 private:
  // The motor controllers
  frc::PWMSparkMax m_left1;
  frc::PWMSparkMax m_left2;
  frc::PWMSparkMax m_right1;
  frc::PWMSparkMax m_right2;

  // The robot's drive
  frc::DifferentialDrive m_drive{[&](double output) { m_left1.Set(output); },
                                 [&](double output) { m_right1.Set(output); }
↪};
```

C++ (Source)

In robotInit or Subsystem constructor :

```
  m_left1.AddFollower(m_left2);
  m_right1.AddFollower(m_right2);

  // We need to invert one side of the drivetrain so that positive voltages
  // result in both sides moving forward. Depending on how your robot's
  // gearbox is constructed, you might have to invert the left side instead.
  m_right1.SetInverted(true);
```

Python

Note : MotorControllerGroup is *deprecated* in 2024. Can you help update this example ?

```
def robotInit(self):
    frontLeft = wpilib.Spark(1)
    rearLeft = wpilib.Spark(2)
    left = wpilib.MotorControllerGroup(frontLeft, rearLeft)
    left.setInverted(True) # if you want to invert the entire side you can␣
↪do so here
```

(suite sur la page suivante)

(suite de la page précédente)

```
    frontRight = wpilib.Spark(3)
    rearRight = wpilib.Spark(4)
    right = wpilib.MotorControllerGroup(frontLeft, rearLeft)

    self.drive = wpilib.drive.DifferentialDrive(left, right)
```

Modes de conduite

Note : La classe DifferentialDrive contient trois différents modes de pilotage (par défaut)
- Tank Drive, qui contrôle les côtés gauche et droit de façon indépendante
- Arcade Drive, qui contrôle la vitesse vers l'avant/arrière et le virage
- Curvature Drive, un sous-ensemble d'Arcade Drive, qui fait que votre robot se comporte comme une voiture avec des virages à courbure constante.

La classe DifferentialDrive contient trois méthodes par défaut pour contrôler les robots de dérapage ou de WCD. Notez que vous pouvez créer vos propres méthodes de contrôle de la conduite du robot et les faire appeler tankDrive() avec les entrées dérivées pour les moteurs gauche et droit.

Le mode Tank Drive est utilisé pour contrôler chaque côté du train d'entraînement de façon indépendante (généralement avec un axe de joystick (manette) individuel contrôlant chacun des côtés). Cet exemple montre comment utiliser l'axe Y de deux joysticks séparés pour un train d'entraînement opéré en mode Tank. La construction des objets a été omise, ci-dessus pour la transmission et ci-dessous pour la construction du joystick.

Le mode Arcade Drive est utilisé pour contrôler le train d'entraînement en utilisant la vitesse/l'accélérateur et le taux de rotation. Ceci est implémenté soit avec les deux axes provenant d'un seul joystick, soit divisé en joysticks (souvent sur une seule manette de jeu) avec l'accélérateur venant d'un joystick et la rotation d'un autre. Cet exemple montre comment utiliser un seul joystick avec le mode Arcade. La construction des objets a été omise, ci-dessus pour la transmission et c-dessous pour la construction du joystick.

Comme Arcade Drive, le mode Curvature Drive est utilisé pour contrôler la transmission en utilisant la vitesse/accélérateur et la vitesse de rotation. La différence est que l'entrée de commande de rotation contrôle le rayon de courbure au lieu du taux de changement de cap, un peu comme le volant d'une voiture. Ce mode prend également en charge la mise en place, qui est activée lorsque le troisième paramètre `boolean` est à l'état vrai (true).

JAVA

```
public void teleopPeriodic() {
    // Tank drive with a given left and right rates
    myDrive.tankDrive(-leftStick.getY(), -rightStick.getY());

    // Arcade drive with a given forward and turn rate
    myDrive.arcadeDrive(-driveStick.getY(), -driveStick.getX());

    // Curvature drive with a given forward and turn rate, as well as a button for␣
↪turning in-place.
    myDrive.curvatureDrive(-driveStick.getY(), -driveStick.getX(), driveStick.
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪getButton(1));
}
```

C++

```
void TeleopPeriodic() override {
    // Tank drive with a given left and right rates
    myDrive.TankDrive(-leftStick.GetY(), -rightStick.GetY());

    // Arcade drive with a given forward and turn rate
    myDrive.ArcadeDrive(-driveStick.GetY(), -driveStick.GetX());

    // Curvature drive with a given forward and turn rate, as well as a quick-turn␣
↪button
    myDrive.CurvatureDrive(-driveStick.GetY(), -driveStick.GetX(), driveStick.
↪GetButton(1));
}
```

PYTHON

```
def teleopPeriodic(self):
    # Tank drive with a given left and right rates
    self.myDrive.tankDrive(-self.leftStick.getY(), -self.rightStick.getY())

    # Arcade drive with a given forward and turn rate
    self.myDrive.arcadeDrive(-self.driveStick.getY(), -self.driveStick.getX())

    # Curvature drive with a given forward and turn rate, as well as a button for␣
↪turning in-place.
    self.myDrive.curvatureDrive(-self.driveStick.getY(), -self.driveStick.getX(),␣
↪self.driveStick.getButton(1))
```

Utilisation de la classe MecanumDrive pour contrôler les robots avec trains d'entraînement Mécanum

MecanumDrive est une méthode fournie pour le contrôle des transmissions holonomiques avec roues Mécanum, telles que le châssis du Kit de pièces, augmenté du kit Mécanum, comme illustré ci-dessus. Instancier un MecanumDrive est aussi simple que ceci :

JAVA

```
  private static final int kFrontLeftChannel = 2;
  private static final int kRearLeftChannel = 3;
  private static final int kFrontRightChannel = 1;
  private static final int kRearRightChannel = 0;

  @Override
  public void robotInit() {
    PWMSparkMax frontLeft = new PWMSparkMax(kFrontLeftChannel);
    PWMSparkMax rearLeft = new PWMSparkMax(kRearLeftChannel);
    PWMSparkMax frontRight = new PWMSparkMax(kFrontRightChannel);
    PWMSparkMax rearRight = new PWMSparkMax(kRearRightChannel);
    // Invert the right side motors.
    // You may need to change or remove this to match your robot.
    frontRight.setInverted(true);
    rearRight.setInverted(true);

    m_robotDrive = new MecanumDrive(frontLeft::set, rearLeft::set, frontRight::set,␣
↪rearRight::set);
  }
```

C++

```
private:
  static constexpr int kFrontLeftChannel = 0;
  static constexpr int kRearLeftChannel = 1;
  static constexpr int kFrontRightChannel = 2;
  static constexpr int kRearRightChannel = 3;

  frc::PWMSparkMax m_frontLeft{kFrontLeftChannel};
  frc::PWMSparkMax m_rearLeft{kRearLeftChannel};
  frc::PWMSparkMax m_frontRight{kFrontRightChannel};
  frc::PWMSparkMax m_rearRight{kRearRightChannel};
  frc::MecanumDrive m_robotDrive{
      [&](double output) { m_frontLeft.Set(output); },
      [&](double output) { m_rearLeft.Set(output); },
      [&](double output) { m_frontRight.Set(output); },
      [&](double output) { m_rearRight.Set(output); }};

  void RobotInit() override {
    // Invert the right side motors. You may need to change or remove this to
    // match your robot.
    m_frontRight.SetInverted(true);
    m_rearRight.SetInverted(true);
  }
```

PYTHON

```
    # Channels on the roboRIO that the motor controllers are plugged in to
    kFrontLeftChannel = 2
    kRearLeftChannel = 3
    kFrontRightChannel = 1
    kRearRightChannel = 0

    def robotInit(self):
        self.frontLeft = wpilib.PWMSparkMax(self.kFrontLeftChannel)
        self.rearLeft = wpilib.PWMSparkMax(self.kRearLeftChannel)
        self.frontRight = wpilib.PWMSparkMax(self.kFrontRightChannel)
        self.rearRight = wpilib.PWMSparkMax(self.kRearRightChannel)

        # invert the right side motors
        # you may need to change or remove this to match your robot
        self.frontRight.setInverted(True)
        self.rearRight.setInverted(True)

        self.robotDrive = wpilib.drive.MecanumDrive(
            self.frontLeft, self.rearLeft, self.frontRight, self.rearRight
        )

        self.stick = wpilib.Joystick(self.kJoystickChannel)
```

Modes d'entraînement Mécanum

Note : Les conventions d'axe d'entraînement sont différentes des conventions d'axe de joystick courantes. Voir les *Conventions d'axe* ci-dessus pour plus d'informations.

La classe MecanumDrive contient deux différents modes de pilotage (par défaut)

- driveCartesian : Les angles sont mesurés dans le sens horaire à partir de l'axe X positif. La vitesse du robot est indépendante de son angle ou de sa vitesse de rotation.
- drivePolar : Les angles sont mesurés dans le sens antihoraire à partir de la ligne droite. La vitesse à laquelle le robot roule (translation) est indépendante de son angle ou de sa vitesse de rotation.

JAVA

```
public void teleopPeriodic() {
    // Drive using the X, Y, and Z axes of the joystick.
    m_robotDrive.driveCartesian(-m_stick.getY(), -m_stick.getX(), -m_stick.getZ());
    // Drive at 45 degrees relative to the robot, at the speed given by the Y axis of␣
↪the joystick, with no rotation.
    m_robotDrive.drivePolar(-m_stick.getY(), Rotation2d.fromDegrees(45), 0);
}
```

C++

```
void TeleopPeriodic() override {
    // Drive using the X, Y, and Z axes of the joystick.
    m_robotDrive.driveCartesian(-m_stick.GetY(), -m_stick.GetX(), -m_stick.GetZ());
    // Drive at 45 degrees relative to the robot, at the speed given by the Y axis of␣
↪the joystick, with no rotation.
    m_robotDrive.drivePolar(-m_stick.GetY(), 45_deg, 0);
}
```

PYTHON

```
def teleopPeriodic(self):
    // Drive using the X, Y, and Z axes of the joystick.
    self.robotDrive.driveCartesian(-self.stick.getY(), -self.stick.getX(), -self.
↪stick.getZ())
    // Drive at 45 degrees relative to the robot, at the speed given by the Y axis of␣
↪the joystick, with no rotation.
    self.robotDrive.drivePolar(-self.stick.getY(), Rotation2d.fromDegrees(45), 0)
```

Conduite orientée sur le terrain

Un 4ème paramètre peut être fourni à la méthode `driveCartesian (double ySpeed, double xSpeed, double zRotation, double gyroAngle)`, soit l'angle renvoyé par un capteur gyroscopique. Cela ajustera la valeur de rotation fournie. Ceci est particulièrement utile avec le Mécanum car, aux fins de la direction, le robot n'a vraiment ni avant, ni arrière, ni côtés. Il peut aller dans n'importe quelle direction. L'ajout de l'angle en degrés par rapport à un objet gyroscopique entraînera le robot à s'éloigner des pilotes lorsque le joystick est poussé vers l'avant et vers les pilotes lorsqu'il est tiré vers eux, quelle que soit la direction dans laquelle le robot fait face.

L'utilisation de la conduite orientée sur le terrain rend souvent le robot beaucoup plus facile à conduire, en particulier par rapport à un système d'entraînement « orienté robot » où les commandes sont inversées lorsque le robot fait face aux conducteurs.

N'oubliez pas d'obtenir l'angle gyroscopique chaque fois que la méthode `driveCartesian()` est appelée.

Note : Beaucoup d'équipes aiment aussi augmenter linéairement les entrées des joysticks au fil du temps pour avoir une accélération en douceur et réduire le jerk. Ceci peut être accompli avec un *Slew Rate Limiter*.

15.1.4 Mouvement répétable à faible puissance - Contrôle des servos avec WPILib

Les servomoteurs sont un type de moteur qui intègre une contrôle rétroactif de la position dans le moteur afin de permettre à un seul moteur d'effectuer un mouvement reproductible et contrôlable, en prenant la position comme signal d'entrée. WPILib offre la possibilité de contrôler des servomoteurs qui correspondent à la spécification d'entrée de passe-temps communément rencontré (signal PWM (Pulse Width Modulation ou Modulation par largeur d'impulsion), 0.6 ms - 2.4 ms de largeur d'impulsion)

Construire un objet Servo

JAVA

```
Servo exampleServo = new Servo(1);
```

C++

```
frc::Servo exampleServo {1};
```

PYTHON

```
exampleServo = wpilib.Servo(1)
```

Un objet Servo est construit en passant un canal.

Définition des valeurs de Servo

JAVA

```
exampleServo.set(.5);
exampleServo.setAngle(75);
```

C++

```
exampleServo.Set(.5);
exampleServo.SetAngle(75);
```

PYTHON

```
exampleServo.set(.5)
exampleServo.setAngle(75)
```

Il existe deux méthodes de définition des valeurs pour Servo dans WPILib :

— Valeur mise à l'échelle - Définit la position du servo à l'aide d'une échelle de 0 à 1.0. La valeur 0 correspond à la fin de course dans un sens de rotation du servo et 1.0 correspond à la fin de course dans l'autre sens.

— Angle - Configurez la position du servo en spécifiant l'angle, en degrés de 0 à 180. Cette méthode fonctionnera pour les servos ayant la même portée que le servo de modèle Hitec HS-322HD . Toutes les valeurs transmises à cette méthode en dehors de la plage spécifiée seront ramenées dans cet intervalle.

15.2 API pneumatiques

15.2.1 Operating Pneumatic Cylinders

FRC teams can use a *solenoid valve* as part of performing a variety of tasks, including shifting gearboxes and moving robot mechanisms. A solenoid valve is used to electronically switch a pressurized air line « on » or « off ». Solenoids are controlled by a robot's Pneumatics Control Module, or Pneumatic Hub, which is in turn connected to the robot's roboRIO via *CAN*. The easiest way to see a solenoid's state is via the LEDs on the PCM or PH (which indicates if the valve is « on » or not). When un-powered, solenoids can be manually actuated with the small button on the valve body.

Single acting solenoids apply or vent pressure from a single output port. They are typically used either when an external force will provide the return action of the cylinder (spring, gravity, separate mechanism) or in pairs to act as a double solenoid. A double solenoid switches air flow between two output ports (many also have a center position where neither output is vented or connected to the input). Double solenoid valves are commonly used when you wish to control both the extend and retract actions of a cylinder using air pressure. Double solenoid valves have two electrical inputs which connect back to two separate channels on the solenoid breakout.

Single Solenoids in WPILib

Single solenoids in WPILib are controlled using the `Solenoid` class (Java / C++). To construct a Solenoid object, simply pass the desired port number (assumes default CAN ID) and pneumatics module type or CAN ID, pneumatics module type, and port number to the constructor. To set the value of the solenoid call `set(true)` to enable or `set(false)` to disable the solenoid output.

Java

```
// Solenoid corresponds to a single solenoid.
// In this case, it's connected to channel 0 of a PH with the default CAN ID.
private final Solenoid m_solenoid = new Solenoid(PneumaticsModuleType.REVPH, 0);
```

```
  /*
   * The output of GetRawButton is true/false depending on whether
   * the button is pressed; Set takes a boolean for whether
   * to retract the solenoid (false) or extend it (true).
   */
  m_solenoid.set(m_stick.getRawButton(kSolenoidButton));
```

C++ (Header)

```
// Solenoid corresponds to a single solenoid.
// In this case, it's connected to channel 0 of a PH with the default CAN
// ID.
frc::Solenoid m_solenoid{frc::PneumaticsModuleType::REVPH, 0};
```

C++ (Source)

```
/*
 * The output of GetRawButton is true/false depending on whether
 * the button is pressed; Set takes a boolean for whether
 * to retract the solenoid (false) or extend it (true).
 */
m_solenoid.Set(m_stick.GetRawButton(kSolenoidButton));
```

Double Solenoids in WPILib

Double solenoids are controlled by the `DoubleSolenoid` class in WPILib (Java / C++). These are constructed similarly to the single solenoid but there are now two port numbers to pass to the constructor, a forward channel (first) and a reverse channel (second). The state of the valve can then be set to `kOff` (neither output activated), `kForward` (forward channel enabled) or `kReverse` (reverse channel enabled). Additionally, the CAN ID can be passed to the DoubleSolenoid if teams have a non-default CAN ID.

Java

```
// DoubleSolenoid corresponds to a double solenoid.
// In this case, it's connected to channels 1 and 2 of a PH with the default CAN ID.
private final DoubleSolenoid m_doubleSolenoid =
    new DoubleSolenoid(PneumaticsModuleType.REVPH, 1, 2);
```

```
    m_doubleSolenoid.set(DoubleSolenoid.Value.kForward);
    m_doubleSolenoid.set(DoubleSolenoid.Value.kReverse);
```

C++ (Header)

```
  // DoubleSolenoid corresponds to a double solenoid.
  // In this case, it's connected to channels 1 and 2 of a PH with the default
  // CAN ID.
  frc::DoubleSolenoid m_doubleSolenoid{frc::PneumaticsModuleType::REVPH, 1, 2};
```

C++ (Source)

```
    m_doubleSolenoid.Set(frc::DoubleSolenoid::kForward);
    m_doubleSolenoid.Set(frc::DoubleSolenoid::kReverse);
```

Toggling Solenoids

Solenoids can be switched from one output to the other (known as toggling) by using the `.toggle()` method.

Note : Since a DoubleSolenoid defaults to off, you will have to set it before it can be toggled.

JAVA

```
Solenoid exampleSingle = new Solenoid(PneumaticsModuleType.CTREPCM, 0);
DoubleSolenoid exampleDouble = new DoubleSolenoid(PneumaticsModuleType.CTREPCM, 1, 2);

// Initialize the DoubleSolenoid so it knows where to start.  Not required for single␣
↪solenoids.
exampleDouble.set(kReverse);

if (m_controller.getYButtonPressed()) {
   exampleSingle.toggle();
   exampleDouble.toggle();
}
```

C++

```
frc::Solenoid exampleSingle{frc::PneumaticsModuleType::CTREPCM, 0};
frc::DoubleSolenoid exampleDouble{frc::PneumaticsModuleType::CTREPCM, 1, 2};

// Initialize the DoubleSolenoid so it knows where to start.  Not required for single␣
↪solenoids.
exampleDouble.Set(frc::DoubleSolenoid::Value::kReverse);

if (m_controller.GetYButtonPressed()) {
   exampleSingle.Toggle();
   exampleDouble.Toggle();
}
```

15.2.2 Generating and Storing Pressure

Pressure is created using a pneumatic compressor and stored in pneumatic tanks. The compressor must be on the robot and powered by the robot's pneumatics module. The « Closed Loop » mode on the Compressor is enabled by default, and it is *not* recommended that teams change this setting. When closed loop control is enabled the pneumatic module will automatically turn the compressor on when the digital pressure switch is closed (below the pressure threshold) and turn it off when the pressure switch is open (~120PSI). When closed loop control is disabled the compressor will not be turned on. Using the `Compressor` (Java / C++) class, users can query the status of the compressor. The state (currently on or off), pressure switch state, and compressor current can all be queried from the Compressor object, as shown by the following code from the Solenoid example project (Java, C++) :

Note : The Compressor object is only needed if you want the ability to turn off the compressor, change the pressure sensor (PH only), or query compressor status.

Construct a `Compressor` object :

REV Pneumatic Hub (PH)

Java

```
  // Compressor connected to a PH with a default CAN ID (1)
  private final Compressor m_compressor = new Compressor(PneumaticsModuleType.REVPH);
```

C++ (Header)

```
  // Compressor connected to a PH with a default CAN ID
  frc::Compressor m_compressor{frc::PneumaticsModuleType::REVPH};
```

CTRE Pneumatics Control Module (PCM)

Java

```
  // Compressor connected to a PCM with a default CAN ID (0)
  private final Compressor m_compressor = new Compressor(PneumaticsModuleType.
↪CTREPCM);
```

C++ (Header)

```
  // Compressor connected to a PH with a default CAN ID
```

Querying compressor current and state :

Java

```
        // Get compressor current draw.
        return m_compressor.getCurrent();
        // Get whether the compressor is active.
        return m_compressor.isEnabled();
        // Get the digital pressure switch connected to the PCM/PH.
        // The switch is open when the pressure is over ~120 PSI.
        return m_compressor.getPressureSwitchValue();
```

C++ (Source)

```
    // Get compressor current draw.
    units::ampere_t compressorCurrent = m_compressor.GetCurrent();
    return compressorCurrent.value();
    // Get whether the compressor is active.
    return m_compressor.IsEnabled();
    // Get the digital pressure switch connected to the PCM/PH.
    // The switch is open when the pressure is over ~120 PSI.
    return m_compressor.GetPressureSwitchValue();
```

Enable/disable digital closed-loop compressor control (enabled by default) :

Java

```
            // Disable closed-loop mode on the compressor.
            m_compressor.disable();
            // Enable closed-loop mode based on the digital pressure switch␣
↪connected to the
            // PCM/PH.
            // The switch is open when the pressure is over ~120 PSI.
            m_compressor.enableDigital();
```

C++ (Source)

```
        // Disable closed-loop mode on the compressor.
        m_compressor.Disable();
        // Enable closed-loop mode based on the digital pressure switch
        // connected to the PCM/PH. The switch is open when the pressure is over
        // ~120 PSI.
        m_compressor.EnableDigital();
```

The Pneumatic Hub also has methods for enabling compressor control using the REV Analog Pressure Sensor :

Java

```
        // Enable closed-loop mode based on the analog pressure sensor connected to␣
↪the PH.
        // The compressor will run while the pressure reported by the sensor is in␣
↪the
        // specified range ([70 PSI, 120 PSI] in this example).
        // Analog mode exists only on the PH! On the PCM, this enables digital␣
↪control.
        m_compressor.enableAnalog(70, 120);
        // Enable closed-loop mode based on both the digital pressure switch AND␣
↪the analog
        // pressure sensor connected to the PH.
        // The compressor will run while the pressure reported by the analog sensor␣
↪is in the
        // specified range ([70 PSI, 120 PSI] in this example) AND the digital␣
↪switch reports
        // that the system is not full.
        // Hybrid mode exists only on the PH! On the PCM, this enables digital␣
↪control.
        m_compressor.enableHybrid(70, 120);
```

C++ (Source)

```
    // Enable closed-loop mode based on the analog pressure sensor connected
    // to the PH. The compressor will run while the pressure reported by the
    // sensor is in the specified range ([70 PSI, 120 PSI] in this example).
    // Analog mode exists only on the PH! On the PCM, this enables digital
    // control.
    m_compressor.EnableAnalog(70_psi, 120_psi);
    // Enable closed-loop mode based on both the digital pressure switch AND the␣
↪analog
    // pressure sensor connected to the PH.
    // The compressor will run while the pressure reported by the analog sensor is␣
↪in the
    // specified range ([70 PSI, 120 PSI] in this example) AND the digital switch␣
↪reports
    // that the system is not full.
    // Hybrid mode exists only on the PH! On the PCM, this enables digital control.
    m_compressor.EnableHybrid(70_psi, 120_psi);
```

Pressure Transducers

A pressure transducer is a sensor where analog voltage is proportial to the measured pressure.

Pneumatic Hub

The Pneumatic Hub has analog inputs that may be used to read a pressure transducer using the Compressor class.

Java

```
// Compressor connected to a PH with a default CAN ID (1)
private final Compressor m_compressor = new Compressor(PneumaticsModuleType.REVPH);
```

```
        // Get the pressure (in PSI) from the analog sensor connected to the PH.
        // This function is supported only on the PH!
        // On a PCM, this function will return 0.
        return m_compressor.getPressure();
```

C++ (Header)

```
// Compressor connected to a PH with a default CAN ID
frc::Compressor m_compressor{frc::PneumaticsModuleType::REVPH};
```

C++ (Source)

```
  // Get the pressure (in PSI) from the analog sensor connected to the PH.
  // This function is supported only on the PH!
  // On a PCM, this function will return 0.
  units::pounds_per_square_inch_t pressure = m_compressor.GetPressure();
  return pressure.value();
```

roboRIO

A pressure transducer can be connected to the Analog Input ports on the roboRIO, and can be read by the `AnalogInput` or `AnalogPotentiometer` classes in WPILib.

Java

```
  // External analog pressure sensor
  // product-specific voltage->pressure conversion, see product manual
  // in this case, 250(V/5)-25
  // the scale parameter in the AnalogPotentiometer constructor is scaled from 1␣
↪instead of 5,
  // so if r is the raw AnalogPotentiometer output, the pressure is 250r-25
  static final double kScale = 250;
  static final double kOffset = -25;
  private final AnalogPotentiometer m_pressureTransducer =
      new AnalogPotentiometer(/* the AnalogIn port*/ 2, kScale, kOffset);
```

```
    // Get the pressure (in PSI) from an analog pressure sensor connected to the RIO.
    return m_pressureTransducer.get();
```

C++ (Header)

```
  // External analog pressure sensor
  // product-specific voltage->pressure conversion, see product manual
  // in this case, 250(V/5)-25
  // the scale parameter in the AnalogPotentiometer constructor is scaled from
  // 1 instead of 5, so if r is the raw AnalogPotentiometer output, the
  // pressure is 250r-25
  static constexpr double kScale = 250;
  static constexpr double kOffset = -25;
  frc::AnalogPotentiometer m_pressureTransducer{/* the AnalogIn port*/ 2,
                                                kScale, kOffset};
```

C++ (Source)

```
  // Get the pressure (in PSI) from an analog pressure sensor connected to
  // the RIO.
  return units::pounds_per_square_inch_t{m_pressureTransducer.Get()};
```

15.2.3 Utilisation du système de contrôle FRC pour contrôler la pneumatique

Il existe deux options pour activer les solénoïdes afin de contrôler les cylindres pneumatiques, le module de commande pneumatique de CTRE et le concentrateur pneumatique de REV Robotics.

Le module de commande pneumatique CTRE (PCM) est un composant qui se branche sur le réseau CAN et qui permet de contrôler le compresseur et jusqu'à 8 solénoïdes par module.

Le concentrateur pneumatique de REV (PH) est un composant qui se branche sur le réseau CAN et qui permet de contrôler le compresseur et jusqu'à 16 solénoïdes par module.

Ces composants sont intégrés à WPILib via une série de classes qui les rendent simples à utiliser. Le contrôle en boucle fermée du compresseur et du pressostat est géré par le ma-

tériel PCM et les solénoïdes sont gérés par la classe `Solénoid` qui contrôle les canaux des solénoïdes.

Ces modules sont chargés de réguler la pression du robot à l'aide d'un pressostat et d'un compresseur et de mettre en marche et d'arrêter les solénoïdes. Ils communiquent avec le roboRIO via le bus CAN. Pour plus d'informations, voir *Aperçu des composants matériels*.

15.2.4 Numéros de module

Les composants CAN sont identifiés par leur ID CAN. L'ID CAN par défaut pour les PCM est 0. L'ID CAN par défaut pour les PH est 1. Si vous utilisez un seul module sur le bus, il est recommandé de le laisser à l'ID CAN par défaut. Des modules supplémentaires peuvent être utilisés lorsque les modules solénoïdes correspondants sont différenciés par le numéro de module dans les constructeurs des classes `Solenoid`, `DoubleSolenoid` et `Compressor`.

15.3 Capteurs

Les capteurs font partie intégrante de la communication entre le matériel et le logiciel de votre robot. Cette section met en évidence l'interfaçage avec ces capteurs au niveau logiciel.

15.3.1 Un bref aperçu du logiciel associé aux capteurs

Note : Cette section couvre l'usage des capteurs dans le logiciel. Pour un guide sur le branchement électrique des capteurs, voir *Bref aperçu des capteurs*.

Note : Alors que les caméras peuvent certainement être considérées comme des « capteurs », le traitement de la vision est un sujet suffisamment compliqué pour être traité dans *sa propre section*, plutôt qu'ici.

Pour être efficace, il est souvent vital pour les robots de pouvoir obtenir des informations sur leur environnement. Les dispositifs qui fournissent au robot des informations sur l'état de son environnement sont appelés «capteurs». WPILib prend en charge de manière innée une grande variété de capteurs via des classes incluses dans ses librairies. Cette section fournira un guide à la fois sur l'utilisation des types de capteurs courants via WPILib, ainsi que sur l'écriture de code pour les capteurs sans support officiel.

Quels capteurs WPILIB prend-il en charge ?

The roboRIO includes an *FPGA* which allows accurate real-time measuring of a variety of sensor input. WPILib, in turn, provides a number of classes for accessing this functionality.

WPILib fournit un support intégré pour les capteurs suivants :

- *Accelerомètres*
- *Gyroscopes*
- *Capteurs ultrasoniques*
- *Potentiomètres*

— *Compteurs*
— *Encodeurs*
— *Interrupteurs fin de course*

De plus, WPILib comprend des classes de niveau inférieur pour une interface directe avec les entrées et sorties numériques et analogiques du FPGA.

15.3.2 Accéléromètres - Partie logicielle

Note : Cette section couvre le logiciel spécifique aux accéléromètres. Pour un guide qui explique le raccordement électrique des accéléromètres, voir *Accéléromètres*.

Un accéléromètre est un appareil qui mesure l'accélération.

Les accéléromètres se présentent généralement en deux types : axe unique et 3-axes. Un accéléromètre à axe unique mesure l'accélération le long d'une seule dimension spatiale ; un accéléromètre à 3 axes mesure l'accélération le long des trois dimensions spatiales à la fois.

WPILib prend en charge les accéléromètres à axe unique via la classe *AnalogAccelerometer*.

Les accéléromètres à trois axes nécessitent souvent des protocoles de communication plus complexes (tels que SPI ou I2C) pour envoyer des données multidimensionnelles. WPILib a supporte les accéléromètres 3 axes suivants :

— *ADXL345_I2C*
— *ADXL345_SPI*
— *ADXL362*
— *BuiltInAccelerometer*

La classe AnalogAccelerometer

The `AnalogAccelerometer` class (Java, C++) allows users to read values from a single-axis accelerometer that is connected to one of the roboRIO's analog inputs.

JAVA

```
// Creates an analog accelerometer on analog input 0
AnalogAccelerometer accelerometer = new AnalogAccelerometer(0);

// Sets the sensitivity of the accelerometer to 1 volt per G
accelerometer.setSensitivity(1);

// Sets the zero voltage of the accelerometer to 3 volts
accelerometer.setZero(3);

// Gets the current acceleration
double accel = accelerometer.getAcceleration();
```

C++

```
// Creates an analog accelerometer on analog input 0
frc::AnalogAccelerometer accelerometer{0};

// Sets the sensitivity of the accelerometer to 1 volt per G
accelerometer.SetSensitivity(1);

// Sets the zero voltage of the accelerometer to 3 volts
accelerometer.SetZero(3);

// Gets the current acceleration
double accel = accelerometer.GetAcceleration();
```

Si les utilisateurs disposent d'un accéléromètre analogique à 3 axes, ils peuvent utiliser trois instances de cette classe, une pour chaque axe.

Il existe des accesseurs en lecture (getters) pour l'accélération le long de chaque direction cardinale (x, y et z), ainsi qu'un accesseur en écriture (setter) pour la plage d'accélérations que l'accéléromètre mesurera.

JAVA

```
// Sets the accelerometer to measure between -8 and 8 G's
accelerometer.setRange(BuiltInAccelerometer.Range.k8G);
```

C++

```
// Sets the accelerometer to measure between -8 and 8 G's
accelerometer.SetRange(BuiltInAccelerometer::Range::kRange_8G);
```

La classe ADXL345_I2C

The ADXL345_I2C class (Java, C++) provides support for the ADXL345 accelerometer over the I2C communications bus.

JAVA

```
// Creates an ADXL345 accelerometer object on the MXP I2C port
// with a measurement range from -8 to 8 G's
ADXL345_I2C accelerometer = new ADXL345_I2C(I2C.Port.kMXP, ADXL345_I2C.Range.k8G);
```

C++

```
// Creates an ADXL345 accelerometer object on the MXP I2C port
// with a measurement range from -8 to 8 G's
frc::ADXL345_I2C accelerometer{I2C::Port::kMXP, frc::ADXL345_I2C::Range::kRange_8G};
```

La classe ADXL345_SPI

The ADXL345_SPI class (Java, C++) provides support for the ADXL345 accelerometer over the SPI communications bus.

JAVA

```
// Creates an ADXL345 accelerometer object on the MXP SPI port
// with a measurement range from -8 to 8 G's
ADXL345_SPI accelerometer = new ADXL345_SPI(SPI.Port.kMXP, ADXL345_SPI.Range.k8G);
```

C++

```
// Creates an ADXL345 accelerometer object on the MXP SPI port
// with a measurement range from -8 to 8 G's
frc::ADXL345_SPI accelerometer{SPI::Port::kMXP, frc::ADXL345_SPI::Range::kRange_8G};
```

La classe ADXL362

The ADXL362 class (Java, C++) provides support for the ADXL362 accelerometer over the SPI communications bus.

JAVA

```
// Creates an ADXL362 accelerometer object on the MXP SPI port
// with a measurement range from -8 to 8 G's
ADXL362 accelerometer = new ADXL362(SPI.Port.kMXP, ADXL362.Range.k8G);
```

C++

```
// Creates an ADXL362 accelerometer object on the MXP SPI port
// with a measurement range from -8 to 8 G's
frc::ADXL362 accelerometer{SPI::Port::kMXP, frc::ADXL362::Range::kRange_8G};
```

La classe BuiltInAccelerometer

The `BuiltInAccelerometer` class (Java, C++) provides access to the roboRIO's own built-in accelerometer :

JAVA

```
// Creates an object for the built-in accelerometer
// Range defaults to +- 8 G's
BuiltInAccelerometer accelerometer = new BuiltInAccelerometer();
```

C++

```
// Creates an object for the built-in accelerometer
// Range defaults to +- 8 G's
frc::BuiltInAccelerometer accelerometer;
```

Les accéléromètres provenant de tierce-parties

Alors que WPILib fournit un support intégré pour un certain nombre d'accéléromètres disponibles dans le kit de pièces ou via FIRST Choice, il existe certains dispositifs AHRS (Attitude and Heading Reference System) qui contiennent des accéléromètres. Ceux-ci sont généralement pris en charge à partir des librairies des manufacturiers. Par contre, si ces dispositifs ont une sortie analogique simple, ils peuvent être utilisées avec la classe *AnalogAccelerometer*.

Utilisation des accéléromètres dans le logiciel

Note : Les accéléromètres, comme leur nom l'indique, mesurent l'accélération. Des accéléromètres précis peuvent être utilisés pour déterminer la position grâce à une double intégration (puisque l'accélération est la deuxième dérivée de la position), de la même manière que les gyroscopes sont utilisés pour déterminer le cap. Cependant, les accéléromètres disponibles pour une utilisation en FRC n'ont pas le degré de performance nécessaire pour être utilisés de cette façon.

Il est recommandé d'utiliser des accéléromètres en FRC® pour toute application qui a besoin d'une mesure approximative de l'accélération actuelle. Cela peut inclure la détection de collisions avec d'autres robots ou éléments de terrain, de sorte que les mécanismes vulnérables puissent être automatiquement rétractés. Ils peuvent également être utilisés pour déterminer quand le robot passe sur un terrain accidenté pour une routine du mode autonome (comme traverser les défenses comme dans FIRST Stronghold).

Pour détecter les collisions, il est préférable de mesurer le paramètre de secousse (Jerk) plutôt que l'accélération. La secousse est la dérivée (ou le taux de changement) de l'accélération, et indique à quelle vitesse les forces sur le robot changent - l'impulsion soudaine d'une collision provoque une forte pointe dans la valeur de la secousse. La secousse peut être calculée en prenant simplement la différence entre deux mesures d'accélération subséquentes et en divisant par le temps entre elles :

JAVA

```java
double prevXAccel = 0.0;
double prevYAccel = 0.0;

BuiltInAccelerometer accelerometer = new BuiltInAccelerometer();

@Override
public void robotPeriodic() {
    // Gets the current accelerations in the X and Y directions
    double xAccel = accelerometer.getX();
    double yAccel = accelerometer.getY();

    // Calculates the jerk in the X and Y directions
    // Divides by .02 because default loop timing is 20ms
    double xJerk = (xAccel - prevXAccel) / 0.02;
    double yJerk = (yAccel - prevYAccel) / 0.02;

    prevXAccel = xAccel;
    prevYAccel = yAccel;
}
```

C++

```cpp
double prevXAccel = 0.0;
double prevYAccel = 0.0;

frc::BuiltInAccelerometer accelerometer;

void Robot::RobotPeriodic() {
    // Gets the current accelerations in the X and Y directions
    double xAccel = accelerometer.GetX();
    double yAccel = accelerometer.GetY();

    // Calculates the jerk in the X and Y directions
    // Divides by .02 because default loop timing is 20ms
    double xJerk = (xAccel - prevXAccel) / 0.02;
    double yJerk = (yAccel - prevYAccel) / 0.02;

    prevXAccel = xAccel;
    prevYAccel = yAccel;
}
```

Most accelerometers legal for FRC use are quite noisy, and it is often a good idea to combine them with the `LinearFilter` class (Java, C++) to reduce the noise :

JAVA

```
BuiltInAccelerometer accelerometer = new BuiltInAccelerometer();

// Create a LinearFilter that will calculate a moving average of the measured X␣
↪acceleration over the past 10 iterations of the main loop

LinearFilter xAccelFilter = LinearFilter.movingAverage(10);

@Override
public void robotPeriodic() {
    // Get the filtered X acceleration
    double filteredXAccel = xAccelFilter.calculate(accelerometer.getX());
}
```

C++

```
frc::BuiltInAccelerometer accelerometer;

// Create a LinearFilter that will calculate a moving average of the measured X␣
↪acceleration over the past 10 iterations of the main loop
auto xAccelFilter = frc::LinearFilter::MovingAverage(10);

void Robot::RobotPeriodic() {
    // Get the filtered X acceleration
    double filteredXAccel = xAccelFilter.Calculate(accelerometer.GetX());
}
```

15.3.3 Gyroscopes - Partie logicielle

Note : Cette section couvre le logiciel associé aux gyroscopes. Pour un guide sur le branchement électrique des gyroscopes, consulter *Gyroscopes*.

Un gyroscope, ou «gyro», est un capteur de vitesse angulaire généralement utilisé en robotique pour mesurer et/ou stabiliser le cap (direction angulaire souhaitée) d'un robot. WPILib fournit un support spécifique pour le gyroscope ADXRS450 disponible dans le kit de pièces, ainsi qu'un support plus général pour une plus grande variété de gyroscopes analogiques via la classe *AnalogGyro*.

Il existe des accesseurs (getters) de la vitesse angulaire et du cap actuels ainsi que des fonctions permettant de remettre à zéro le cap actuel et de calibrer le gyroscope.

Note : Il est primordial que le robot reste immobile pendant l'étalonnage d'un gyroscope.

ADIS16448

The ADIS16448 uses the `ADIS16448_IMU` class (Java, C++, Python). See the Analog Devices ADIS16448 documentation for additional information and examples.

> **Avertissement :** La documentation d'Analog Devices liée ci-dessus contient des instructions obsolètes pour l'installation du logiciel, car l'ADIS16448 est désormais intégré à WPILib.

JAVA

```
// ADIS16448 plugged into the MXP port
ADIS16448_IMU gyro = new ADIS16448_IMU();
```

C++

```
// ADIS16448 plugged into the MXP port
ADIS16448_IMU gyro;
```

PYTHON

```
from wpilib import ADIS16448_IMU

# ADIS16448 plugged into the MXP port
self.gyro = ADIS16448_IMU()
```

ADIS16470

The ADIS16470 uses the `ADIS16470_IMU` class (Java, C++, Python). See the Analog Devices ADIS16470 documentation for additional information and examples.

> **Avertissement :** La documentation d'Analog Devices liée ci-dessus contient des instructions obsolètes pour l'installation du logiciel, car l'ADIS16470 est désormais intégré à WPILib.

JAVA

```
// ADIS16470 plugged into the SPI port
ADIS16470_IMU gyro = new ADIS16470_IMU();
```

C++

```
// ADIS16470 plugged into the SPI port
ADIS16470_IMU gyro;
```

PYTHON

```
# ADIS16470 plugged into the SPI port
self.gyro = ADIS16470_IMU()
```

ADXRS450_Gyro

The `ADXRS450_Gyro` class (Java, C++, Python) provides support for the Analog Devices ADXRS450 gyro available in the kit of parts, which connects over the SPI bus.

Note : L'accumulation des données du gyroscope ADXRS450 est gérée par des circuits spéciaux dans le FPGA ; en conséquence, une seule instance de `ADXRS450_Gyro` peut être utilisée.

JAVA

```
// Creates an ADXRS450_Gyro object on the onboard SPI port
ADXRS450_Gyro gyro = new ADXRS450_Gyro();
```

C++

```
// Creates an ADXRS450_Gyro object on the onboard SPI port
frc::ADXRS450_Gyro gyro;
```

PYTHON

```
# Creates an ADXRS450_Gyro object on the onboard SPI port
self.gyro = ADXRS450_Gyro()
```

AnalogGyro

The `AnalogGyro` class (Java, C++, `Python`) provides support for any single-axis gyro with an analog output.

Note : L'accumulation des données du gyroscope est gérée par des circuits spéciaux dans le FPGA; en conséquence, `AnalogGyro`s ne peut être utilisé que sur les ports analogiques 0 et 1.

JAVA

```
// Creates an AnalogGyro object on port 0
AnalogGyro gyro = new AnalogGyro(0);
```

C++

```
// Creates an AnalogGyro object on port 0
frc::AnalogGyro gyro{0};
```

PYTHON

```
# Creates an AnalogGyro object on port 0
self.gyro = AnalogGyro(0)
```

navX

Le navX utilise la classe `AHRS`. Consultez la documentation navX pour d'autres types de connexion.

JAVA

```
// navX MXP using SPI
AHRS gyro = new AHRS(SPI.Port.kMXP);
```

C++

```
// navX MXP using SPI
AHRS gyro{SPI::Port::kMXP};
```

PYTHON

```
import navx

# navX MXP using SPI
self.gyro = navx.AHRS(SPI.Port.kMXP)
```

Pigeon

Le Pigeon doit utiliser la classe `WPI_PigeonIMU`. Le Pigeon peut être connecté avec le réseau CAN ou par câble de données à un TalonSRX. Le Guide de l'utilisateur de Pigeon IMU contient tous les détails sur l'utilisation du Pigeon.

JAVA

```
WPI_PigeonIMU gyro = new WPI_PigeonIMU(0); // Pigeon is on CAN Bus with device ID 0
// OR (choose one or the other based on your connection)
TalonSRX talon = new TalonSRX(0); // TalonSRX is on CAN Bus with device ID 0
WPI_PigeonIMU gyro = new WPI_PigeonIMU(talon); // Pigeon uses the talon created above
```

C++

```
WPI_PigeonIMU gyro{0}; // Pigeon is on CAN Bus with device ID 0
// OR (choose one or the other based on your connection)
TalonSRX talon{0}; // TalonSRX is on CAN Bus with device ID 0
WPI_PigeonIMU gyro{talon}; // Pigeon uses the talon created above
```

PYTHON

```
import phoenix5
import ctre.sensors

self.gyro = ctre.WPI_PigeonIMU(0); # Pigeon is on CAN Bus with device ID 0
# OR (choose one or the other based on your connection)
talon = ctre.TalonSRX(0); # TalonSRX is on CAN Bus with device ID 0
self.gyro = ctre.WPI_PigeonIMU(talon) # Pigeon uses the talon created above
```

Utilisation de gyroscopes dans le code

Note : Comme les gyroscopes mesurent le taux (variation) plutôt que la position angulaire, la position est calculée en intégrant (additionnant) les taux pour obtenir la valeur d'angle actuel. Par conséquant, les mesures d'angle gyroscopique sont toujours relatives à un angle zéro arbitraire (déterminé par l'angle du gyroscope lorsque le robot est mis en marche ou lorsqu'une méthode de mise à zéro a été appelée). Ces mesures sont également sujettes à

des erreurs accumulées (appelées « dérives ») qui vont en s'augmentant, au fur et à mesure que le gyroscope est utilisé. La quantité de dérive varie selon le type de gyroscope.

Les gyroscopes sont extrêmement utiles en FRC pour mesurer et contrôler le cap du robot. Étant donné que les jeux FRC sont de courte durée (moins de 3 minutes), la dérive totale du gyroscope au cours d'un jeu FRC a tendance à être gérable (de l'ordre de quelques degrés pour un gyroscope de bonne qualité). De plus, ce ne sont pas toutes les applications gyroscopiques qui nécessitent que la mesure du cap reste précise tout au long du jeu.

Affichage du cap du robot sur le tableau de bord

Shuffleboard inclut un widget pour afficher les données de cap d'un gyro sous la forme d'une boussole. Cela peut être utile pour visualiser le cap du robot lorsque les lignes de vue vers le robot sont obscurcies :

JAVA

```
// Use gyro declaration from above here

public void robotInit() {
    // Places a compass indicator for the gyro heading on the dashboard
    Shuffleboard.getTab("Example tab").add(gyro);
}
```

C++

```
// Use gyro declaration from above here

void Robot::RobotInit() {
    // Places a compass indicator for the gyro heading on the dashboard
    frc::Shuffleboard.GetTab("Example tab").Add(gyro);
}
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

def robotInit(self):
    # Use gyro declaration from above here

    # Places a compass indicator for the gyro heading on the dashboard
    Shuffleboard.getTab("Example tab").add(self.gyro)
```

Stabilisation du cap pendant la conduite

Une utilisation très courante d'un gyroscope est de stabiliser le cap du robot pendant la conduite, de sorte que le robot roule en ligne droite. Ceci est particulièrement important pour les entraînements holonomiques tels que Mécanum et Swerve, mais aussi utile pour les entraînements différentiels, style « char d'assaut ».

Ceci est généralement réalisé en utilisant une boucle de contrôle PID axée sur la vitesse de rotation angulaire ou le cap du robot. La sortie de cette boucle est dirigée vers les contrôleurs de moteurs, et le code génère alors un léger différentiel de vitesse entre les deux côtés de l'entraînement (gauche et droite) afin de corriger le cap du robot. Pour les robots avec Mécanum ou Swerve, cette correction de cap est gérée de façon différente.

Avertissement : Comme pour toutes les boucles de contrôle, les utilisateurs doivent veiller à ce que la direction du capteur et la direction de rotation soient cohérentes. Si ce n'est pas le cas, la boucle sera instable et le robot pivotera autour de son axe central de manière incontrôlée.

Exemple : stabilisation de l'entraînement de type différentiel à l'aide du taux de rotation

L'exemple suivant montre comment stabiliser le cap à l'aide d'une simple boucle P axée sur le taux de rotation angulaire. Étant donné qu'un robot qui ne tourne pas devrait avoir un taux de rotation égal à zéro, le point de consigne pour la boucle est implicitement nul, ce qui rend cette méthode très simple.

JAVA

```
// Use gyro declaration from above here

// The gain for a simple P loop
double kP = 1;

// Initialize motor controllers and drive
Spark leftLeader = new Spark(0);
Spark leftFollower = new Spark(1);

Spark rightLeader = new Spark(2);
Spark rightFollower = new Spark(3);

DifferentialDrive drive = new DifferentialDrive(leftLeader::set, rightLeader::set);

@Override
public void robotInit() {
    // Configures the encoder's distance-per-pulse
    // The robot moves forward 1 foot per encoder rotation
    // There are 256 pulses per encoder rotation
    encoder.setDistancePerPulse(1./256.);

    // Invert the right side of the drivetrain. You might have to invert the other␣
↪side
```

(suite sur la page suivante)

(suite de la page précédente)

```
    rightLeader.setInverted(true);

    // Configure the followers to follow the leaders
    leftLeader.addFollower(leftFollower);
    rightLeader.addFollower(rightFollower);
}

@Override
public void autonomousPeriodic() {
    // Setpoint is implicitly 0, since we don't want the heading to change
    double error = -gyro.getRate();

    // Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    drive.tankDrive(.5 + kP * error, .5 - kP * error);
}
```

C++

```
// Use gyro declaration from above here

// The gain for a simple P loop
double kP = 1;

// Initialize motor controllers and drive
frc::Spark leftLeader{0};
frc::Spark leftFollower{1};

frc::Spark rightLeader{2};
frc::Spark rightFollower{3};

frc::DifferentialDrive drive{[&](double output) { leftLeader.Set(output); },
                             [&](double output) { rightLeader.Set(output); }};

void Robot::RobotInit() {
    // Invert the right side of the drivetrain. You might have to invert the other␣
↪side
    rightLeader.SetInverted(true);

    // Configure the followers to follow the leaders
    leftLeader.AddFollower(leftFollower);
    rightLeader.AddFollower(rightFollower);
}

void Robot::AutonomousPeriodic() {
    // Setpoint is implicitly 0, since we don't want the heading to change
    double error = -gyro.GetRate();

    // Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    drive.TankDrive(.5 + kP * error, .5 - kP * error);
}
```

PYTHON

```python
from wpilib import Spark
from wpilib import MotorControllerGroup
from wpilib.drive import DifferentialDrive

def robotInit(self):
    # Use gyro declaration from above here

    # The gain for a simple P loop
    self.kP = 1

    # Initialize motor controllers and drive
    left1 = Spark(0)
    left2 = Spark(1)
    right1 = Spark(2)
    right2 = Spark(3)

    leftMotors = MotorControllerGroup(left1, left2)
    rightMotors = MotorControllerGroup(right1, right2)

    self.drive = DifferentialDrive(leftMotors, rightMotors)

    rightMotors.setInverted(true)

def autonomousPeriodic(self):
    # Setpoint is implicitly 0, since we don't want the heading to change
    error = -self.gyro.getRate()

    # Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    self.drive.tankDrive(.5 + self.kP * error, .5 - self.kP * error)
```

Note : MotorControllerGroup is *deprecated* in 2024. Can you help update the Python example ?

D'autres implémentations plus avancées peuvent utiliser une boucle de contrôle plus complexe. Les boucles PI sont particulièrement efficaces pour l'autorégulation de la direction basée sur le taux de rotation angulaire.

Exemple : stabilisation de l'entraînement du réservoir à l'aide du cap

L'exemple suivant montre comment stabiliser le cap à l'aide d'une simple boucle P axée sur le cap. Contrairement à l'exemple précédent, nous devrons régler le point de consigne sur le cap actuel du robot avant de commencer le mouvement, ce qui rend cette méthode légèrement plus compliquée.

JAVA

```
// Use gyro declaration from above here

// The gain for a simple P loop
double kP = 1;

// The heading of the robot when starting the motion
double heading;

// Initialize motor controllers and drive
Spark left1 = new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

MotorControllerGroup leftMotors = new MotorControllerGroup(left1, left2);
MotorControllerGroup rightMotors = new MotorControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);

@Override
public void robotInit() {
    rightMotors.setInverted(true);
}

@Override
public void autonomousInit() {
    // Set setpoint to current heading at start of auto
    heading = gyro.getAngle();
}

@Override
public void autonomousPeriodic() {
    double error = heading - gyro.getAngle();

    // Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    drive.tankDrive(.5 + kP * error, .5 - kP * error);
}
```

C++

```
// Use gyro declaration from above here

// The gain for a simple P loop
double kP = 1;

// The heading of the robot when starting the motion
double heading;

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};

frc::MotorControllerGroup leftMotors{left1, left2};
frc::MotorControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::RobotInit() {
  rightMotors.SetInverted(true);
}

void Robot::AutonomousInit() {
    // Set setpoint to current heading at start of auto
    heading = gyro.GetAngle();
}

void Robot::AutonomousPeriodic() {
    double error = heading - gyro.GetAngle();

    // Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    drive.TankDrive(.5 + kP * error, .5 - kP * error);
}
```

PYTHON

```
from wpilib import Spark
from wpilib import MotorControllerGroup
from wpilib.drive import DifferentialDrive

def robotInit(self):
    # Use gyro declaration from above here

    # The gain for a simple P loop
    self.kP = 1

    # Initialize motor controllers and drive
    left1 = Spark(0)
    left2 = Spark(1)
    right1 = Spark(2)
    right2 = Spark(3)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    leftMotors = MotorControllerGroup(left1, left2)
    rightMotors = MotorControllerGroup(right1, right2)

    self.drive = DifferentialDrive(leftMotors, rightMotors)

    rightMotors.setInverted(true)

def autonomousInit(self):
    # Set setpoint to current heading at start of auto
    self.heading = self.gyro.getAngle()

def autonomousPeriodic(self):
    error = self.heading - self.gyro.getAngle()

    # Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    self.drive.tankDrive(.5 + self.kP * error, .5 - self.kP * error)
```

Les implémentations plus avancées peuvent utiliser une boucle de contrôle plus complexe. Les boucles PD sont particulièrement efficaces pour l'autorégulation de la direction basée sur le cap.

Rotation vers un cap défini

Une autre application courante et très utile pour un gyroscope consiste à faire tourner le robot pour qu'il pointe vers une direction spécifiée. Cela peut être utile lors de la période de conduite autonome, ou encore, pendant le contrôle téléopéré pour aider à aligner le robot avec un élément du terrain de jeu.

Tout comme avec la stabilisation de cap, ceci est accompli avec une boucle PID - contrairement à la stabilisation, la boucle doit être axée sur le cap uniquement. L'exemple de code suivant fera tourner le robot de 90 degrés avec une simple boucle P :

JAVA

```
// Use gyro declaration from above here

// The gain for a simple P loop
double kP = 0.05;

// Initialize motor controllers and drive
Spark left1 = new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

MotorControllerGroup leftMotors = new MotorControllerGroup(left1, left2);
MotorControllerGroup rightMotors = new MotorControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);
```

(suite sur la page suivante)

(suite de la page précédente)

```java
@Override
public void robotInit() {
    rightMotors.setInverted(true);
}

@Override
public void autonomousPeriodic() {
    // Find the heading error; setpoint is 90
    double error = 90 - gyro.getAngle();

    // Turns the robot to face the desired direction
    drive.tankDrive(kP * error, -kP * error);
}
```

C++

```cpp
// Use gyro declaration from above here

// The gain for a simple P loop
double kP = 0.05;

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};

frc::MotorControllerGroup leftMotors{left1, left2};
frc::MotorControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::RobotInit() {
  rightMotors.SetInverted(true);
}

void Robot::AutonomousPeriodic() {
    // Find the heading error; setpoint is 90
    double error = 90 - gyro.GetAngle();

    // Turns the robot to face the desired direction
    drive.TankDrive(kP * error, -kP * error);
}
```

PYTHON

```
from wpilib import Spark
from wpilib import MotorControllerGroup
from wpilib.drive import DifferentialDrive

def robotInit(self):
    # Use gyro declaration from above here

    # The gain for a simple P loop
    self.kP = 0.05

    # Initialize motor controllers and drive
    left1 = Spark(0)
    left2 = Spark(1)
    right1 = Spark(2)
    right2 = Spark(3)

    leftMotors = MotorControllerGroup(left1, left2)
    rightMotors = MotorControllerGroup(right1, right2)

    self.drive = DifferentialDrive(leftMotors, rightMotors)

    rightMotors.setInverted(true)

def autonomousPeriodic(self):
    # Find the heading error; setpoint is 90
    error = 90 - self.gyro.getAngle()

    # Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    self.drive.tankDrive(self.kP * error, -self.kP * error)
```

Comme précédemment, les implémentations plus avancées peuvent utiliser des boucles de contrôle plus compliquées.

Note : Les boucles de rotation à angle peuvent être difficiles à régler correctement en raison de la force de friction existante dans la transmission, surtout si une simple boucle P est utilisée. Il existe plusieurs façons de contrebalancer cela ; l'une des plus courantes/efficaces consiste à ajouter une valeur de « sortie minimale » à la sortie de la boucle de contrôle. Une autre stratégie efficace consiste à pré-régler les contrôleurs de vitesse pour chaque moteur, sur chaque côté du robot.

15.3.4 Ultrasons - Partie logicielle

Note : Cette section couvre le logiciel relatif aux capteurs ultrasoniques. Pour un guide sur le branchement électrique des capteurs à ultrasons, voir *Ultrasons*.

Un capteur à ultrasons est couramment utilisé pour mesurer la distance à un objet à l'aide d'un son haute fréquence. En règle générale, les capteurs à ultrasons mesurent la distance de l'objet le plus rapproché dans leur «champ de vision».

Il existe deux principaux types d'ultrasons pris en charge nativement par WPILib :
- *Capteurs à ultrasons à réponse Ping*
- *Capteurs à ultrasons analogiques*

Capteurs à réponse ping

The `Ultrasonic` class (Java, C++) provides support for ping-response ultrasonics. As ping-response ultrasonics (per the name) require separate pins for both sending the ping and measuring the response, users must specify DIO pin numbers for both output and input when constructing an `Ultrasonic` instance :

Java

```
// Creates a ping-response Ultrasonic object on DIO 1 and 2.
Ultrasonic m_rangeFinder = new Ultrasonic(1, 2);
```

C++

```
// Creates a ping-response Ultrasonic object on DIO 1 and 2.
frc::Ultrasonic m_rangeFinder{1, 2};
```

The measurement can then be retrieved in either inches or millimeters in Java ; in C++ the *units library* is used to automatically convert to any desired length unit :

Java

```
  // We can read the distance in millimeters
  double distanceMillimeters = m_rangeFinder.getRangeMM();
  // ... or in inches
  double distanceInches = m_rangeFinder.getRangeInches();
```

C++

```
// We can read the distance
units::meter_t distance = m_rangeFinder.GetRange();
// units auto-convert
units::millimeter_t distanceMillimeters = distance;
units::inch_t distanceInches = distance;
```

Capteurs à ultrasons analogiques

Certains capteurs à ultrasons renvoient simplement une tension analogique correspondant à la distance mesurée. Ces capteurs peuvent être simplement utilisés avec la classe *AnalogPotentiometer*

Capteurs ultrasoniques en provenance de tierce partie

D'autres capteurs à ultrasons proposés par certains manufacturiers peuvent utiliser des protocoles de communication plus complexes (comme I2C ou SPI). WPILib ne fournit pas de support intégré pour ces capteurs à ultrasons ; ils seront généralement contrôlés avec les librairies des fournisseurs.

Utiliser les capteurs ultrasoniques dans le code

Ultrasonic sensors are very useful for determining spacing during autonomous routines. For example, the following code from the UltrasonicPID example project (Java, C++) will move the robot to 1 meter away from the nearest object the sensor detects :

Java

```
public class Robot extends TimedRobot {
  // distance the robot wants to stay from an object
  // (one meter)
  static final double kHoldDistanceMillimeters = 1.0e3;

  // proportional speed constant
  private static final double kP = 0.001;
  // integral speed constant
  private static final double kI = 0.0;
  // derivative speed constant
  private static final double kD = 0.0;

  static final int kLeftMotorPort = 0;
  static final int kRightMotorPort = 1;

  static final int kUltrasonicPingPort = 0;
  static final int kUltrasonicEchoPort = 1;

  // Ultrasonic sensors tend to be quite noisy and susceptible to sudden␣
↪outliers,
  // so measurements are filtered with a 5-sample median filter
  private final MedianFilter m_filter = new MedianFilter(5);

  private final Ultrasonic m_ultrasonic = new Ultrasonic(kUltrasonicPingPort,
↪ kUltrasonicEchoPort);
  private final PWMSparkMax m_leftMotor = new PWMSparkMax(kLeftMotorPort);
  private final PWMSparkMax m_rightMotor = new PWMSparkMax(kRightMotorPort);
  private final DifferentialDrive m_robotDrive =
      new DifferentialDrive(m_leftMotor::set, m_rightMotor::set);
  private final PIDController m_pidController = new PIDController(kP, kI,␣
↪kD);
```

(suite sur la page suivante)

(suite de la page précédente)

```
  public Robot() {
    SendableRegistry.addChild(m_robotDrive, m_leftMotor);
    SendableRegistry.addChild(m_robotDrive, m_rightMotor);
  }

  @Override
  public void autonomousInit() {
    // Set setpoint of the pid controller
    m_pidController.setSetpoint(kHoldDistanceMillimeters);
  }

  @Override
  public void autonomousPeriodic() {
    double measurement = m_ultrasonic.getRangeMM();
    double filteredMeasurement = m_filter.calculate(measurement);
    double pidOutput = m_pidController.calculate(filteredMeasurement);

    // disable input squaring -- PID output is linear
    m_robotDrive.arcadeDrive(pidOutput, 0, false);
  }

}
```

C++ (Header)

```
class Robot : public frc::TimedRobot {
 public:
  Robot();
  void AutonomousInit() override;
  void AutonomousPeriodic() override;

  // distance the robot wants to stay from an object
  static constexpr units::millimeter_t kHoldDistance = 1_m;

  static constexpr int kLeftMotorPort = 0;
  static constexpr int kRightMotorPort = 1;
  static constexpr int kUltrasonicPingPort = 0;
  static constexpr int kUltrasonicEchoPort = 1;

 private:
  // proportional speed constant
  static constexpr double kP = 0.001;
  // integral speed constant
  static constexpr double kI = 0.0;
  // derivative speed constant
  static constexpr double kD = 0.0;

  // Ultrasonic sensors tend to be quite noisy and susceptible to sudden
  // outliers, so measurements are filtered with a 5-sample median filter
  frc::MedianFilter<units::millimeter_t> m_filter{5};

  frc::Ultrasonic m_ultrasonic{kUltrasonicPingPort, kUltrasonicEchoPort};
  frc::PWMSparkMax m_left{kLeftMotorPort};
```

(suite sur la page suivante)

(suite de la page précédente)

```
  frc::PWMSparkMax m_right{kRightMotorPort};
  frc::DifferentialDrive m_robotDrive{
      [&](double output) { m_left.Set(output); },
      [&](double output) { m_right.Set(output); }};
  frc::PIDController m_pidController{kP, kI, kD};
};
```

C++ (Source)

```
void Robot::AutonomousInit() {
  // Set setpoint of the pid controller
  m_pidController.SetSetpoint(kHoldDistance.value());
}

void Robot::AutonomousPeriodic() {
  units::millimeter_t measurement = m_ultrasonic.GetRange();
  units::millimeter_t filteredMeasurement = m_filter.Calculate(measurement);
  double pidOutput = m_pidController.Calculate(filteredMeasurement.value());

  // disable input squaring -- PID output is linear
  m_robotDrive.ArcadeDrive(pidOutput, 0, false);
}
```

De plus, les ultrasons de réponse ping peuvent être envoyés à *Shuffleboard*, où ils seront affichés avec leurs propres widgets :

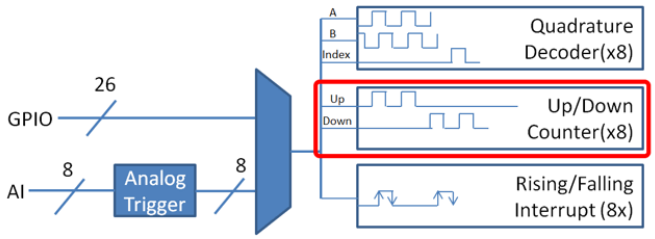
Java

```
    // Add the ultrasonic on the "Sensors" tab of the dashboard
    // Data will update automatically
    Shuffleboard.getTab("Sensors").add(m_rangeFinder);
```

C++

```
  // Add the ultrasonic on the "Sensors" tab of the dashboard
  // Data will update automatically
  frc::Shuffleboard::GetTab("Sensors").Add(m_rangeFinder);
```

15.3.5 Compteurs - Partie logicielle



The `Counter` class (Java, C++) is a versatile class that allows the counting of pulse edges on a digital input. `Counter` is used as a component in several more-complicated WPILib classes (such as *Encoder* and *Ultrasonic*), but is also quite useful on its own.

Note : Il y a un total de 8 dispositifs de type « compteur » dans le roboRIO FPGA, ce qui signifie pas plus de 8 objets `Counter` peuvent être instanciés à tout moment, y compris ceux contenus comme ressources dans d'autres objets WPILib. Pour des informations détaillées sur quand un : code :*Counter* peut être utilisé par un autre objet, reportez-vous à la documentation officielle de l'API.

Configuration d'un compteur

La classe `Counter` peut être configurée de différentes manières pour fournir des fonctionnalités différentes.

Modes de l'objet Counter

L'objet `Counter` peut être configuré pour fonctionner dans l'un des quatre modes différents :

1. *Two-pulse mode* : compte de haut en bas en fonction des fronts de deux canaux différents.
2. *Semi-period mode* : mesure la durée d'une impulsion sur un seul canal.
3. *Mode de longueur d'impulsion* : compte en haut et en bas en fonction des bords d'un canal, la direction étant déterminée par la durée de l'impulsion sur ce canal.
4. *External direction mode* : compte vers le haut et vers le bas en fonction des fronts d'un canal. Un autre canal séparé spécifie la direction.

Note : In all modes except semi-period mode, the counter can be configured to increment either once per edge (2X decoding), or once per pulse (1X decoding). By default, counters are set to two-pulse mode, though if only one channel is specified the counter will only count up.

Mode deux-impulsions (« Two-pulse mode »)

En mode deux-impulsions , le `Counter` comptera vers le haut pour chaque front / impulsion sur le « canal montant » spécifié et vers le bas pour chaque front / impulsion sur le « “canal descendant » spécifié. Un compteur peut être initialisé en deux impulsions avec le code suivant :

JAVA

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.k2Pulse);

@Override
public void robotInit() {
    // Set up the input channels for the counter
    counter.setUpSource(1);
    counter.setDownSource(2);

    // Set the decoding type to 2X
    counter.setUpSourceEdge(true, true);
    counter.setDownSourceEdge(true, true);
}
```

C++

```
// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::k2Pulse};

void Robot::RobotInit() {
    // Set up the input channels for the counter
    counter.SetUpSource(1);
    counter.SetDownSource(2);

    // Set the decoding type to 2X
    counter.SetUpSourceEdge(true, true);
    counter.SetDownSourceEdge(true, true);
```

Mode demi-période (« Semi-period mode »)

En mode demi-période, le `Counter` comptera la durée des impulsions sur un canal, soit d’un front montant au front descendant suivant, soit d’un front descendant au front montant suivant. Un compteur peut être initialisé en mode demi-période avec le code suivant :

JAVA

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.kSemiperiod);

@Override
public void robotInit() {
    // Set up the input channel for the counter
    counter.setUpSource(1);

    // Set the encoder to count pulse duration from rising edge to falling edge
    counter.setSemiPeriodMode(true);
}
```

C++

```
// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::kSemiperiod};

void Robot() {
    // Set up the input channel for the counter
    counter.SetUpSource(1);

    // Set the encoder to count pulse duration from rising edge to falling edge
    counter.SetSemiPeriodMode(true);
```

Pour obtenir la largeur d'impulsion, appelez la méthode `getPeriod ()` :

JAVA

```
// Return the measured pulse width in seconds
counter.getPeriod();
```

C++

```
// Return the measured pulse width in seconds
counter.GetPeriod();
```

Mode durée d'impulsion (« Pulse-length mode »)

En mode durée d'impulsion, le compteur comptera vers le haut ou vers le bas selon la durée de l'impulsion. Une impulsion au-dessous du temps seuil spécifié sera interprétée comme un compte vers le haut et une impulsion au-dessus du seuil est un compte vers le bas. (Notez l'inversion logique). Ceci est utile pour certains capteurs qui codent la direction de cette manière, comme ceux qui mesurent la position sur les dents d'un engrenage . Un compteur peut être initialisé dans ce mode comme suit :

JAVA

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.kPulseLength);

@Override
public void robotInit() {
    // Set up the input channel for the counter
    counter.setUpSource(1);

    // Set the decoding type to 2X
    counter.setUpSourceEdge(true, true);

    // Set the counter to count down if the pulses are longer than .05 seconds
    counter.setPulseLengthMode(.05)
}
```

C++

```
// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::kPulseLength};

void Robot::RobotInit() {
    // Set up the input channel for the counter
    counter.SetUpSource(1);

    // Set the decoding type to 2X
    counter.SetUpSourceEdge(true, true);

    // Set the counter to count down if the pulses are longer than .05 seconds
    counter.SetPulseLengthMode(.05)
```

Mode direction externe (« External direction mode »)

In external direction mode, the counter counts either up or down depending on the level on the second channel. If the direction source is low, the counter will increase ; if the direction source is high, the counter will decrease (to reverse this, see the next section). A counter can be initialized in this mode as follows :

JAVA

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.kExternalDirection);

@Override
public void robotInit() {
    // Set up the input channels for the counter
    counter.setUpSource(1);
    counter.setDownSource(2);

    // Set the decoding type to 2X
```

(suite sur la page suivante)

(suite de la page précédente)

```
    counter.setUpSourceEdge(true, true);
}
```

C++

```
// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::kExternalDirection};

void RobotInit() {
    // Set up the input channels for the counter
    counter.SetUpSource(1);
    counter.SetDownSource(2);

    // Set the decoding type to 2X
    counter.SetUpSourceEdge(true, true);
```

Configuration des paramètres des compteurs

Note : La classe `Counter` ne fait aucune hypothèse sur les unités de distance ; comme elle compte seulement les impulsions, elle retournera les valeurs dans le système d'unités utilisées pour calculer la distance. Les utilisateurs ont ainsi un contrôle total sur les unités de distance utilisées. Cependant, les unités de temps sont *toujours* en secondes.

Note : Le nombre d'impulsions utilisé dans le calcul de la distance par impulsion ne dépend *pas* du type de décodage - chaque « impulsion » doit toujours être considérée comme un cycle complet (montant et descendant).

Outre les configurations spécifiques au mode, la classe `Counter` propose un certain nombre de méthodes de configuration supplémentaires :

JAVA

```
// Configures the counter to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
counter.setDistancePerPulse(4./256.);

// Configures the counter to consider itself stopped after .1 seconds
counter.setMaxPeriod(.1);

// Configures the counter to consider itself stopped when its rate is below 10
counter.setMinRate(10);

// Reverses the direction of the counter
counter.setReverseDirection(true);

// Configures an counter to average its period measurement over 5 samples
```

(suite sur la page suivante)

(suite de la page précédente)

```
// Can be between 1 and 127 samples
counter.setSamplesToAverage(5);
```

C++

```
// Configures the counter to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
counter.SetDistancePerPulse(4./256.);

// Configures the counter to consider itself stopped after .1 seconds
counter.SetMaxPeriod(.1);

// Configures the counter to consider itself stopped when its rate is below 10
counter.SetMinRate(10);

// Reverses the direction of the counter
counter.SetReverseDirection(true);

// Configures an counter to average its period measurement over 5 samples
// Can be between 1 and 127 samples
counter.SetSamplesToAverage(5);
```

Lecture des informations des compteurs

Quel que soit le mode, il existe des informations que la classe `Counter` rend toujours disponibles aux utilisateurs :

Le compte

Les utilisateurs peuvent obtenir le compte actuel avec la méthode `get()` :

JAVA

```
// returns the current count
counter.get();
```

C++

```
// returns the current count
counter.Get();
```

La distance

Note : Les compteurs mesurent la distance *relative*, pas absolue ; la valeur de distance renvoyée dépendra de la position de l'encodeur lorsque le robot a été allumé ou la dernière valeur de l'encodeur reset.

Si la distance par impulsion a été configurée, les utilisateurs peuvent obtenir la distance totale parcourue par le capteur compté avec la méthode `getDistance( )` :

JAVA

```
// returns the current distance
counter.getDistance();
```

C++

```
// returns the current distance
counter.GetDistance();
```

Le taux

Note : Les unités de temps pour la classe `Counter` sont *toujours* en secondes.

Les utilisateurs peuvent obtenir le taux de changement actuel du compteur avec la méthode `getRate()` :

JAVA

```
// Gets the current rate of the counter
counter.getRate();
```

C++

```
// Gets the current rate of the counter
counter.GetRate();
```

Mode stationnaire

Les utilisateurs peuvent savoir si le compteur est stationnaire avec la méthode `getStopped()` :

JAVA

```
// Gets whether the counter is stopped
counter.getStopped();
```

C++

```
// Gets whether the counter is stopped
counter.GetStopped();
```

La direction

Les utilisateurs peuvent obtenir la direction vers laquelle le compteur incrémente ou décrémente avec le code :

JAVA

```
// Gets the last direction in which the counter moved
counter.getDirection();
```

C++

```
// Gets the last direction in which the counter moved
counter.GetDirection();
```

La période

Note : Dans le *mode semi-période*, cette méthode renvoie la durée de l'impulsion, pas de la période.

Les utilisateurs peuvent obtenir la durée (en secondes) de la période la plus récente avec la méthode `getPeriod()` :

JAVA

```
// returns the current period in seconds
counter.getPeriod();
```

C++

```
// returns the current period in seconds
counter.GetPeriod();
```

Réinitialisation d'un compteur

Pour réinitialiser un compteur à une lecture de distance de zéro, appelez la méthode `reset()`. Ceci est utile pour garantir que la distance mesurée correspond à la mesure physique réelle souhaitée.

JAVA

```
// Resets the encoder to read a distance of zero
counter.reset();
```

C++

```
// Resets the encoder to read a distance of zero
counter.Reset();
```

Utilisation de compteurs dans le code

Les compteurs sont utiles pour une grande variété d'applications robotiques - mais comme la classe `Counter` est si variée, il est difficile d'en fournir un bon résumé ici. Beaucoup de ces applications se chevauchent avec la classe `Encoder` - un simple compteur est souvent une alternative moins chère à un encodeur en quadrature. Pour un résumé des utilisations potentielles des encodeurs dans le code, voir *Encodeurs - Partie logicielle*.

15.3.6 Encodeurs - Partie logicielle

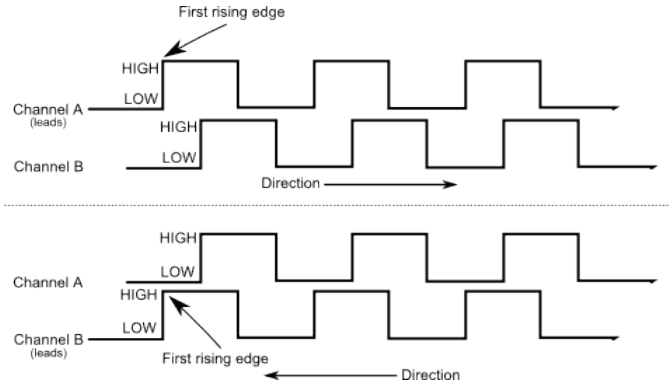
Note : Cette section couvre le logiciel associé aux encodeurs. Pour un guide sur le branchement électrique des encodeurs, voir *Encodeurs*.

Les encodeurs sont des composants utilisés pour mesurer le mouvement (généralement la rotation d'un arbre).

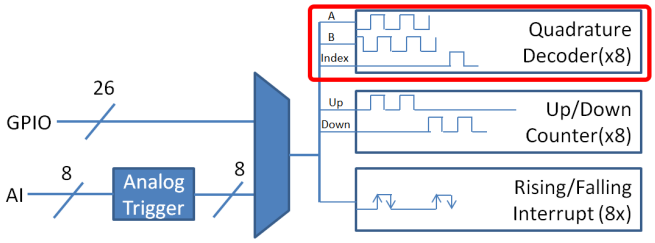
Important : Les classe dans ce document ne sont utilisées que pour les encodeurs qui sont branchés directement sur le roboRIO ! Veuillez consulter la documentation appropriée des fournisseurs pour l'utilisation d'encodeurs branchés sur des contrôleurs de moteur.

Encodeurs en quadrature - la classe `Encoder`

WPILib provides support for quadrature encoders through the `Encoder` class (Java, C++). This class provides a simple API for configuring and reading data from encoders.



Ces encodeurs produisent des signaux rectangulaires sur deux canaux qui sont déphasés d'un quart de période (d'où le terme « quadrature »). Les impulsions sont utilisées pour mesurer la rotation, et la direction du mouvement peut être déterminée à partir de quel canal « est en avance » par rapport à l'autre.



Le FPGA gère les encodeurs en quadrature via un module compteur ou un module encodeur, selon le *decoding type* - le choix est géré automatiquement par WPILib . Le FPGA contient 8 modules encodeurs.

Exemples d'encodeurs en quadrature :

- AMT103-V disponible via FIRST Choice
- CIMcoder
- CTRE Mag Encoder
- Grayhill 63r
- REV Through Bore Encoder
- US Digital E4T

Initialisation d'un encodeur en quadrature

Un encodeur en quadrature peut être instancié comme suit :

JAVA

```
// Initializes an encoder on DIO pins 0 and 1
// Defaults to 4X decoding and non-inverted
Encoder encoder = new Encoder(0, 1);
```

C++

```
// Initializes an encoder on DIO pins 0 and 1
// Defaults to 4X decoding and non-inverted
frc::Encoder encoder{0, 1};
```

Type de décodage

La classe WPILib `Encoder` peut décoder les signaux d'encodeur dans trois modes différents :

- **Décodage 1X** : Augmente la distance pour chaque période complète du signal de l'encodeur (une fois par quatre fronts).
- **Décodage 2X** : Augmente la distance pour chaque demi-période du signal de l'encodeur (une fois par deux fronts).
- **Décodage 4X** : Incrémente la distance pour chaque front du signal de l'encodeur (quatre fois par période).

Le décodage 4X offre la plus grande précision, mais au prix potentiel de « fluctuations » (jitter) accrue dans les mesures de débit. Pour utiliser un type de décodage différent, utilisez le constructeur suivant :

JAVA

```
// Initializes an encoder on DIO pins 0 and 1
// 2X encoding and non-inverted
Encoder encoder = new Encoder(0, 1, false, Encoder.EncodingType.k2X);
```

C++

```
// Initializes an encoder on DIO pins 0 and 1
// 2X encoding and non-inverted
frc::Encoder encoder{0, 1, false, frc::Encoder::EncodingType::k2X};
```

Configuration des paramètres de l'encodeur en quadrature

Note : La classe `Encoder` ne fait aucune hypothèse sur les unités de distance ; comme elle compte seulement les impulsions, elle retournera les valeurs dans le système d'unités utilisées pour calculer la distance. Les utilisateurs ont ainsi un contrôle total sur les unités de distance utilisées. Cependant, les unités de temps sont *toujours* en secondes.

Note : Le nombre d'impulsions utilisées dans le calcul de la distance par impulsion ne dépend *pas* du *type de décodage* - chaque « impulsion » doit toujours être considéré comme un cycle complet (quatre fronts).

La classe `Encoder` propose plusieurs méthodes de configuration :

JAVA

```
// Configures the encoder to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
encoder.setDistancePerPulse(4.0/256.0);

// Configures the encoder to consider itself stopped after .1 seconds
encoder.setMaxPeriod(0.1);

// Configures the encoder to consider itself stopped when its rate is below 10
encoder.setMinRate(10);

// Reverses the direction of the encoder
encoder.setReverseDirection(true);

// Configures an encoder to average its period measurement over 5 samples
// Can be between 1 and 127 samples
encoder.setSamplesToAverage(5);
```

C++

```
// Configures the encoder to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
encoder.SetDistancePerPulse(4.0/256.0);

// Configures the encoder to consider itself stopped after .1 seconds
encoder.SetMaxPeriod(0.1);

// Configures the encoder to consider itself stopped when its rate is below 10
encoder.SetMinRate(10);

// Reverses the direction of the encoder
encoder.SetReverseDirection(true);

// Configures an encoder to average its period measurement over 5 samples
// Can be between 1 and 127 samples
encoder.SetSamplesToAverage(5);
```

Lecture des informations des encodeurs en quadrature

La classe `Encoder` fournit une multitude d'informations à l'utilisateur sur le mouvement de l'encodeur.

La distance

Note : Les encodeurs en quadrature mesurent la distance *relative,* pas absolue ; la valeur de distance renvoyée dépendra de la position de l'encodeur lorsque le robot a été allumé ou la dernière valeur de l'encodeur *reset*.

Les utilisateurs peuvent obtenir la distance totale parcourue par l'encodeur avec la méthode `getDistance()` :

JAVA

```
// Gets the distance traveled
encoder.getDistance();
```

C++

```
// Gets the distance traveled
encoder.GetDistance();
```

Le taux

Note : Les unités de temps pour la classe `Encoder` sont *toujours* en secondes.

Les utilisateurs peuvent obtenir le taux de changement actuel de l'encodeur avec la méthode `getRate()` :

JAVA

```
// Gets the current rate of the encoder
encoder.getRate();
```

C++

```
// Gets the current rate of the encoder
encoder.GetRate();
```

Mode stationnaire

Les utilisateurs peuvent savoir si l'encodeur est stationnaire avec la méthode `getStopped()` :

JAVA

```
// Gets whether the encoder is stopped
encoder.getStopped();
```

C++

```
// Gets whether the encoder is stopped
encoder.GetStopped();
```

La direction

Les utilisateurs peuvent obtenir la direction dans laquelle l'encodeur s'est déplacé pour la dernière fois avec la méthode `getDirection()` :

JAVA

```
// Gets the last direction in which the encoder moved
encoder.getDirection();
```

C++

```
// Gets the last direction in which the encoder moved
encoder.GetDirection();
```

La période

Les utilisateurs peuvent obtenir la période des impulsions du codeur (en secondes) avec la méthode `getPeriod()` :

JAVA

```
// Gets the current period of the encoder
encoder.getPeriod();
```

C++

```
// Gets the current period of the encoder
encoder.GetPeriod();
```

Réinitialisation d'un encodeur en quadrature

Pour réinitialiser un encodeur en quadrature à une lecture de distance de zéro, appelez la méthode `reset()`. Ceci est utile pour s'assurer que la distance mesurée correspond à la mesure physique réelle souhaitée, et est souvent appelée lors d'une routine *de mise à zéro* :

JAVA

```
// Resets the encoder to read a distance of zero
encoder.reset();
```

C++

```
// Resets the encoder to read a distance of zero
encoder.Reset();
```

Encodeurs de rapport cyclique - La classe `DutyCycleEncoder`

WPILib provides support for duty cycle (also marketed as *PWM*) encoders through the `DutyCycleEncoder` class (Java, C++). This class provides a simple API for configuring and reading data from duty cycle encoders.

Le FPGA du roboRIO gère automatiquement les encodeurs de rapport cyclique.

Exemples d'encodeurs de rapport cyclique :
- AndyMark Mag Encoder
- CTRE Mag Encoder
- REV Through Bore Encoder
- Team 221 Lamprey2
- US Digital MA3

Initialisation d'un encodeur de rapport cyclique

Un encodeur de rapport cyclique («duty cycle») peut être instancié comme suit :

JAVA

```
// Initializes a duty cycle encoder on DIO pins 0
DutyCycleEncoder encoder = new DutyCycleEncoder(0);
```

C++

```
// Initializes a duty cycle encoder on DIO pins 0
frc::DutyCycleEncoder encoder{0};
```

Configuration des paramètres de l'encodeur de rapport cyclique

---

Note : La classe `DutyCycleEncoder` ne fait aucune hypothèse sur les unités de distance ; comme elle compte seulement les rotations, elle retournera les valeurs dans le système d'unités utilisées pour calculer la distance. Les utilisateurs ont ainsi un contrôle total sur les unités de distance utilisées.

La classe `DutyCycleEncoder` propose plusieurs méthodes de configuration :

JAVA

```
// Configures the encoder to return a distance of 4 for every rotation
encoder.setDistancePerRotation(4.0);
```

C++

```
// Configures the encoder to return a distance of 4 for every rotation
encoder.SetDistancePerRotation(4.0);
```

Lire une distance avec un encodeur de rapport cyclique

Note : L'encodeur de rapport cyclique mesure une distance absolue. Celle-ci ne dépend pas de la position de départ de l'encodeur.

Les utilisateurs peuvent obtenir la distance mesurée par l'encodeur avec la méthode `getDistance()` :

JAVA

```
// Gets the distance traveled
encoder.getDistance();
```

C++

```
// Gets the distance traveled
encoder.GetDistance();
```

Détecter si un encodeur de rapport cyclique est connecté

Comme un encodeur de rapport cyclique génère un ensemble continu d'impulsions, il est possible de détecter qu'un encodeur a été débranché.

JAVA

```
// Gets if the encoder is connected
encoder.isConnected();
```

C++

```
// Gets if the encoder is connected
encoder.IsConnected();
```

Réinitialisation d'un encodeur de rapport cyclique

Pour réinitialiser un encodeur afin que la distance actuelle soit de 0, appelez la méthode `reset()`. Ceci est utile pour garantir que la distance mesurée correspond à la mesure physique réelle souhaitée. Contrairement aux encodeurs en quadrature, les encodeurs de rapport cyclique n'ont pas besoin de retour au repos (homing). Cependant, après la réinitialisation, le décalage de position peut être stocké pour être défini au démarrage du programme afin d'éviter une nouvelle réinitialisation. La Preferences class 1 fournit une méthode pour enregistrer et récupérer les valeurs sur le roboRIO.

JAVA

```
// Resets the encoder to read a distance of zero at the current position
encoder.reset();

// get the position offset from when the encoder was reset
encoder.getPositionOffset();

// set the position offset to half a rotation
encoder.setPositionOffset(0.5);
```

C++

```
// Resets the encoder to read a distance of zero at the current position
encoder.Reset();

// get the position offset from when the encoder was reset
encoder.GetPositionOffset();

// set the position offset to half a rotation
encoder.SetPositionOffset(0.5);
```

Encodeurs analogiques - la classe `AnalogEncoder`

WPILib provides support for analog absolute encoders through the `AnalogEncoder` class (Java, C++). This class provides a simple API for configuring and reading data from duty cycle encoders.

Exemples d'encodeurs analogiques :
- Team 221 Lamprey2
- Thrifty Absolute Magnetic Encoder
- US Digital MA3

Initialisation d'un encodeur analogique

Un encodeur analogique peut être instancié comme suit :

JAVA

```
// Initializes a duty cycle encoder on Analog Input pins 0
AnalogEncoder encoder = new AnalogEncoder(0);
```

C++

```
// Initializes a duty cycle encoder on DIO pins 0
frc::AnalogEncoder encoder{0};
```

Configuration des paramètres de l'encodeur analogique

Note : La classe `AnalogEncoder` ne fait aucune hypothèse sur les unités de distance ; comme elle compte seulement les rotations, elle retournera les valeurs dans le système d'unités utilisées pour calculer la distance. Les utilisateurs ont ainsi un contrôle total sur les unités de distance utilisées.

La classe `AnalogEncoder` propose plusieurs méthodes de configuration :

JAVA

```
// Configures the encoder to return a distance of 4 for every rotation
encoder.setDistancePerRotation(4.0);
```

C++

```
// Configures the encoder to return a distance of 4 for every rotation
encoder.SetDistancePerRotation(4.0);
```

Lire une distance avec un encodeur analogique

Note : L'encodeur analogique mesure une distance absolue. Celle-ci ne dépend pas de la position de départ de l'encodeur.

Les utilisateurs peuvent obtenir la distance mesurée par l'encodeur avec la méthode `getDistance()` :

JAVA

```
// Gets the distance measured
encoder.getDistance();
```

C++

```
// Gets the distance measured
encoder.GetDistance();
```

Réinitialisation d'un encodeur analogique

Pour réinitialiser un encodeur analogique afin que la distance actuelle soit de 0, appelez la méthode `reset()`. Ceci est utile pour garantir que la distance mesurée correspond à la mesure physique réelle souhaitée. Contrairement aux encodeurs en quadrature, les encodeurs de rapport cyclique («duty cycle») n'ont pas besoin de retour au repos (homing). Cependant, après la réinitialisation, le décalage de position peut être stocké pour être défini au démarrage du programme afin d'éviter une nouvelle réinitialisation. La Preferences class 1 fournit une méthode pour enregistrer et récupérer les valeurs sur le roboRIO.

JAVA

```
// Resets the encoder to read a distance of zero at the current position
encoder.reset();

// get the position offset from when the encoder was reset
encoder.getPositionOffset();

// set the position offset to half a rotation
encoder.setPositionOffset(0.5);
```

C++

```
// Resets the encoder to read a distance of zero at the current position
encoder.Reset();

// get the position offset from when the encoder was reset
encoder.GetPositionOffset();

// set the position offset to half a rotation
encoder.SetPositionOffset(0.5);
```

Utilisation des encodeurs dans le code

Les encodeurs sont parmi les capteurs les plus utiles de FRC ; ils sont essentiels pour automatiser un mouvement de robot qui nécessite une certaine complexité. Les applications potentielles des encodeurs dans le code robot sont trop nombreuses pour être toutes énumérées ici, mais un exemple est fourni ci-dessous :

Parcourir une certaine distance

Les encodeurs peuvent être utilisés pour créer une routine simple de type « parcourir une certaine distance ». Ceci est utile en mode autonome, mais présente l'inconvénient que l'élan du robot l'amènera au-delà de la distance prévue. Les meilleures méthodes incluent l'utilisation d'un PID Controller 1 ou l'utilisation de Path Planning 2

Note : L'exemple suivant utilise la classe *Encoder*, mais est similaire si un autre type (*DutyCycleEncoder* ou *AnalogEncoder*) est utilisé. Cependant, les encodeurs en quadrature sont généralement mieux adaptés aux transmissions car ils retournent à zéro (roll over) plusieurs fois et n'ont pas de position absolue.

JAVA

```
// Creates an encoder on DIO ports 0 and 1
Encoder encoder = new Encoder(0, 1);

// Initialize motor controllers and drive
Spark leftLeader = new Spark(0);
Spark leftFollower = new Spark(1);

Spark rightLeader = new Spark(2);
Spark rightFollower = new Spark(3);

DifferentialDrive drive = new DifferentialDrive(leftLeader::set, rightLeader::set);

@Override
public void robotInit() {
    // Configures the encoder's distance-per-pulse
    // The robot moves forward 1 foot per encoder rotation
    // There are 256 pulses per encoder rotation
    encoder.setDistancePerPulse(1./256.);

    // Invert the right side of the drivetrain. You might have to invert the other␣
↪side
    rightLeader.setInverted(true);

    // Configure the followers to follow the leaders
    leftLeader.addFollower(leftFollower);
    rightLeader.addFollower(rightFollower);
}

@Override
public void autonomousPeriodic() {
    // Drives forward at half speed until the robot has moved 5 feet, then stops:
    if(encoder.getDistance() < 5) {
        drive.tankDrive(0.5, 0.5);
    } else {
        drive.tankDrive(0, 0);
    }
}
```

C++

```
 // Creates an encoder on DIO ports 0 and 1.
frc::Encoder encoder{0, 1};

// Initialize motor controllers and drive
frc::Spark leftLeader{0};
frc::Spark leftFollower{1};

frc::Spark rightLeader{2};
frc::Spark rightFollower{3};

frc::DifferentialDrive drive{[&](double output) { leftLeader.Set(output); },
                             [&](double output) { rightLeader.Set(output); }};
```

(suite sur la page suivante)

(suite de la page précédente)

```
void Robot::RobotInit() {
    // Configures the encoder's distance-per-pulse
    // The robot moves forward 1 foot per encoder rotation
    // There are 256 pulses per encoder rotation
    encoder.SetDistancePerPulse(1.0/256.0);

    // Invert the right side of the drivetrain. You might have to invert the other␣
↪side
    rightLeader.SetInverted(true);

    // Configure the followers to follow the leaders
    leftLeader.AddFollower(leftFollower);
    rightLeader.AddFollower(rightFollower);
}

void Robot::AutonomousPeriodic() {
    // Drives forward at half speed until the robot has moved 5 feet, then stops:
    if(encoder.GetDistance() < 5) {
        drive.TankDrive(0.5, 0.5);
    } else {
        drive.TankDrive(0, 0);
    }
}
```

La mise à zéro d'un mécanisme

Comme les encodeurs en quadrature mesurent la distance *relative*, il est souvent important de s'assurer que leur « point zéro » est au bon endroit. Une façon typique de le faire est une « routine de référencement », dans laquelle un mécanisme est déplacé jusqu'à ce qu'il atteigne une position connue (généralement accomplie avec un interrupteur de fin de course), ou « point zéro », puis l'encodeur est réinitialisé. Le code suivant fournit un exemple de base :

Note : La mise à zéro n'est pas nécessaire pour les encodeurs absolus comme les encodeurs de rapport cyclique et les encodeurs analogiques.

JAVA

```
Encoder encoder = new Encoder(0, 1);

Spark spark = new Spark(0);

// Limit switch on DIO 2
DigitalInput limit = new DigitalInput(2);

public void autonomousPeriodic() {
    // Runs the motor backwards at half speed until the limit switch is pressed
    // then turn off the motor and reset the encoder
    if(!limit.get()) {
        spark.set(-0.5);
    } else {
```

(suite sur la page suivante)

(suite de la page précédente)

```
        spark.set(0);
        encoder.reset();
    }
}
```

C++

```
frc::Encoder encoder{0,1};

frc::Spark spark{0};

// Limit switch on DIO 2
frc::DigitalInput limit{2};

void AutonomousPeriodic() {
    // Runs the motor backwards at half speed until the limit switch is pressed
    // then turn off the motor and reset the encoder
    if(!limit.Get()) {
        spark.Set(-0.5);
    } else {
        spark.Set(0);
        encoder.Reset();
    }
}
```

15.3.7 Entrées analogiques - Partie logicielle

Note : Cette section couvre le logiciel associé aux entrées analogiques. Pour un guide sur le branchement électrique aux entrées analogiques, voir *Entrées analogiques*.

Le FPGA du roboRIO prend en charge jusqu'à 8 canaux d'entrée analogiques qui peuvent être utilisés pour lire la valeur d'une tension analogique à partir d'un capteur. Les entrées analogiques peuvent être utilisées pour tout capteur qui produit une tension simple.

Les entrées analogiques du FPGA retournent par défaut un entier de 12 bits proportionnel à la tension, de 0 à 5 volts.

La classe AnalogInput

Note : Il est souvent plus pratique d'utiliser la classe wrapper *Potentiomètres analogiques* que d'utiliser `AnalogInput` directement, car elle supporte la mise à l'échelle en unités significatives.

Support for reading the voltages on the FPGA analog inputs is provided through the `AnalogInput` class (Java, C++).

Initialisation d'une entrée analogique

Un `AnalogInput` peut être initialisé comme suit :

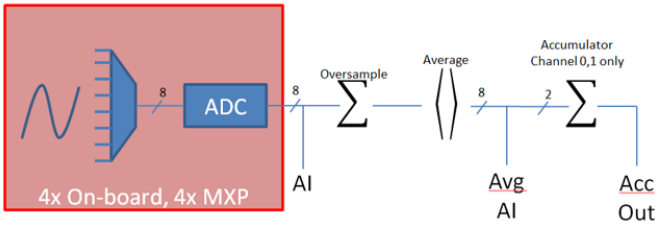
JAVA

```
// Initializes an AnalogInput on port 0
AnalogInput analog = new AnalogInput(0);
```

C++

```
// Initializes an AnalogInput on port 0
frc::AnalogInput analog{0};
```

Suréchantillonnage et moyenne



Les modules d'entrée analogique du FPGA prennent en charge à la fois le suréchantillonnage et la moyenne. Ces comportements sont très similaires, mais diffèrent de plusieurs façons importantes. Les deux peuvent être utilisés en même temps.

Suréchantillonnage

Lorsque le suréchantillonnage est activé, le FPGA additionne ensemble plusieurs échantillons consécutifs et renvoie la valeur cumulée. Les utilisateurs peuvent spécifier le nombre de *bits* de suréchantillonnage - pour n bits de suréchantillonnage, le nombre d'échantillons additionnés est 2^n :

JAVA

```
// Sets the AnalogInput to 4-bit oversampling.  16 samples will be added together.
// Thus, the reported values will increase by about a factor of 16, and the update
// rate will decrease by a similar amount.
analog.setOversampleBits(4);
```

C++

```
// Sets the AnalogInput to 4-bit oversampling.  16 samples will be added together.
// Thus, the reported values will increase by about a factor of 16, and the update
// rate will decrease by a similar amount.
analog.SetOversampleBits(4);
```

Moyenne

La moyenne se comporte comme un suréchantillonnage, sauf que les valeurs accumulées sont divisées par le nombre d'échantillons de sorte que la mise à l'échelle des valeurs renvoyées ne change pas. C'est souvent plus pratique, mais parfois l'erreur d'arrondi supplémentaire introduite par l'arrondi n'est pas souhaitable.

JAVA

```
// Sets the AnalogInput to 4-bit averaging.  16 samples will be averaged together.
// The update rate will decrease by a factor of 16.
analog.setAverageBits(4);
```

C++

```
// Sets the AnalogInput to 4-bit averaging.  16 samples will be averaged together.
// The update rate will decrease by a factor of 16.
analog.SetAverageBits(4);
```

Note : Lorsque le suréchantillonnage et la moyenne sont utilisés en même temps, le suréchantillonnage est appliqué *en premier*, puis la moyenne est calculée sur les valeurs suréchantillonnées. Ainsi, un suréchantillonnage à 2 bits et une moyenne à 2 bits utilisés en même temps augmenteront l'échelle des valeurs renvoyées par approximativement un facteur de 2, mais par contre, réduiront le taux de mise à jour par un facteur de 4.

Lecture des valeurs à partir d'une entrée analogique

Les valeurs peuvent être lues à partir d'une entrée analogique en utilisant une des quatre méthodes suivantes :

getValue

La méthode `getValue` renvoie la valeur instantanée mesurée à l'entrée analogique, sans appliquer d'étalonnage et en ignorant les paramètres de suréchantillonnage et de moyenne. La valeur renvoyée est un entier.

JAVA

```
analog.getValue();
```

C++

```
analog.GetValue();
```

getVoltage

La méthode `getVoltage` renvoie la tension instantanée mesurée à l" l'entrée analogique. Les paramètres de suréchantillonnage et de moyenne sont ignorés, mais la valeur est redimensionnée pour représenter une tension. La valeur renvoyée est un double.

JAVA

```
analog.getVoltage();
```

C++

```
analog.GetVoltage();
```

getAverageValue

La méthode `getAverageValue` renvoie la valeur moyenne de l'entrée analogique. La valeur n'est pas redimensionnée, mais un suréchantillonnage et une moyenne sont tous deux appliqués. La valeur renvoyée est un entier.

JAVA

```
analog.getAverageValue();
```

C++

```
analog.GetAverageValue();
```

getAverageVoltage

La méthode `getAverageVoltage` renvoie la tension moyenne de l'entrée analogique. Le redimensionnement, le suréchantillonnage et la moyenne sont tous appliqués. La valeur renvoyée est un double.

JAVA

```
analog.getAverageVoltage();
```

C++

```
analog.GetAverageVoltage();
```

Accumulateur

Note : Les méthodes d'accumulateur ne prennent actuellement pas en charge le retour d'une valeur en unités de volts - la valeur retournée sera toujours un entier (en particulier, un `long`).

Les canaux d'entrée analogique 0 et 1 prennent également en charge un accumulateur qui intègre (additionne) le signal indéfiniment, de sorte que la valeur renvoyée est la somme de toutes les valeurs mesurées dans le passé. Le suréchantillonnage et la moyenne sont appliqués avant l'accumulation.

JAVA

```
// Sets the initial value of the accumulator to 0
// This is the "starting point" from which the value will change over time
analog.setAccumulatorInitialValue(0);

// Sets the "center" of the accumulator to 0.  This value is subtracted from
// all measured values prior to accumulation.
analog.setAccumulatorCenter(0);
```

(suite sur la page suivante)

(suite de la page précédente)

```
// Returns the number of accumulated samples since the accumulator was last started/
↪reset
analog.getAccumulatorCount();

// Returns the value of the accumulator.  Return type is long.
analog.getAccumulatorValue();

// Resets the accumulator to the initial value
analog.resetAccumulator();
```

C++

```
// Sets the initial value of the accumulator to 0
// This is the "starting point" from which the value will change over time
analog.SetAccumulatorInitialValue(0);

// Sets the "center" of the accumulator to 0.  This value is subtracted from
// all measured values prior to accumulation.
analog.SetAccumulatorCenter(0);

// Returns the number of accumulated samples since the accumulator was last started/
↪reset
analog.GetAccumulatorCount();

// Returns the value of the accumulator.  Return type is long.
analog.GetAccumulatorValue();

// Resets the accumulator to the initial value
analog.ResetAccumulator();
```

Obtention du nombre et de la valeur synchronisés

Parfois, il est nécessairement d'obtenir des mesures appariées du comptage et de la valeur. Cela peut être fait en utilisant la méthode `getAccumulatorOutput` :

JAVA

```
// Instantiate an AccumulatorResult object to hold the matched measurements
AccumulatorResult result = new AccumulatorResult();

// Fill the AccumulatorResult with the matched measurements
analog.getAccumulatorOutput(result);

// Read the values from the AccumulatorResult
long count = result.count;
long value = result.value;
```

C++

```
// The count and value variables to fill
int_64t count;
int_64t value;

// Fill the count and value variables with the matched measurements
analog.GetAccumulatorOutput(count, value);
```

Utilisation d'entrées analogiques dans le code

La classe `AnalogInput` peut être utilisée pour écrire du code afin de lire les données d'une grande variété de capteurs (y compris des potentiomètres, des accéléromètres, des gyroscopes, des ultrasons, etc.) qui renvoient leurs données sous forme de tension analogique. Cependant, si possible, il est préférable d'utiliser les autres classes WPILib existantes qui appelent et utilisent déjà le code de bas-niveau (lecture des tensions analogiques et conversion en unités significatives). Les programmeurs doivent utiliser `AnalogInput` en « dernier recours », et privilégier les routines de haut-niveau d'abord.

En conséquence, pour des exemples d'utilisation efficace de capteurs analogiques dans du code, les utilisateurs doivent se référer aux autres pages de ce chapitre qui traitent de classes plus spécifiques.

15.3.8 Potentiomètres analogiques - Partie logicielle

Note : Cette section couvre le logiciel associé aux potentiomètres analogiques. Pour un guide sur le branchement électrique des potentiomètres analogiques, voir *Potentiomètres analogiques*.

Les potentiomètres sont des résistances variables qui permettent de convertir des informations sur la position en un signal de tension analogique. Ce signal peut être lu par le roboRIO pour contrôler tout appareil connecté au potentiomètre.

While it is possible to read information from a potentiometer directly with an *Entrées analogiques - Partie logicielle*, WPILib provides an `AnalogPotentiometer` class (Java, C++) that handles re-scaling the values into meaningful units for the user. It is strongly encouraged to use this class.

En fait, le nom `AnalogPotentiometer` est un nom qui suggère un usage restraint - cette classe peut être aussi utilisée pour la grande majorité des capteurs qui envoient leur signal sous la forme d'une tension analogique simple à échelle linéaire.

La classe AnalogPotentiometer

Note : Les paramètres « full range » ou « scale » dans le constructeur `AnalogPotentiometer` sont des facteurs d'échelle dans une plage comprenant les valeurs entre 0 à 1 , et non *pas* la valeur réelle (en voltage) de la lecture, comprise entre 0 à 5 volts. Autrement dit, ils représentent une échelle fractionnaire normalisée, plutôt qu'une échelle de tension.

AnalogPotentiometer peut être initialisé comme suit :

JAVA

```
// Initializes an AnalogPotentiometer on analog port 0
// The full range of motion (in meaningful external units) is 0-180 (this could be␣
↪degrees, for instance)
// The "starting point" of the motion, i.e. where the mechanism is located when the␣
↪potentiometer reads 0v, is 30.

AnalogPotentiometer pot = new AnalogPotentiometer(0, 180, 30);
```

C++

```
// Initializes an AnalogPotentiometer on analog port 0
// The full range of motion (in meaningful external units) is 0-180 (this could be␣
↪degrees, for instance)
// The "starting point" of the motion, i.e. where the mechanism is located when the␣
↪potentiometer reads 0v, is 30.

frc::AnalogPotentiometer pot{0, 180, 30};
```

Personnalisation de AnalogInput

Note : Si l'utilisateur modifie l'échelle de `AnalogInput` avec un suréchantillonnage, cela doit se refléter dans le paramètre d'échelle transmis au `AnalogPotentiometer`.

Si l'utilisateur souhaite appliquer des paramètres personnalisés au `AnalogInput` utilisé par le `AnalogPotentiometer`, un constructeur alternatif peut être utilisé dans lequel le `AnalogInput` est injecté :

JAVA

```
// Initializes an AnalogInput on port 0, and enables 2-bit averaging
AnalogInput input = new AnalogInput(0);
input.setAverageBits(2);

// Initializes an AnalogPotentiometer with the given AnalogInput
// The full range of motion (in meaningful external units) is 0-180 (this could be␣
↪degrees, for instance)
// The "starting point" of the motion, i.e. where the mechanism is located when the␣
↪potentiometer reads 0v, is 30.

AnalogPotentiometer pot = new AnalogPotentiometer(input, 180, 30);
```

C++

```
// Initializes an AnalogInput on port 0, and enables 2-bit averaging
frc::AnalogInput input{0};
input.SetAverageBits(2);

// Initializes an AnalogPotentiometer with the given AnalogInput
// The full range of motion (in meaningful external units) is 0-180 (this could be␣
↪degrees, for instance)
// The "starting point" of the motion, i.e. where the mechanism is located when the␣
↪potentiometer reads 0v, is 30.

frc::AnalogPotentiometer pot{input, 180, 30};
```

Lecture des valeurs à partir de AnalogPotentiometer

La valeur mise à l'échelle peut être lue en appelant simplement la méthode `get` :

JAVA

```
pot.get();
```

C++

```
pot.Get();
```

Utilisation de AnalogPotentiometers dans le code

Les capteurs analogiques peuvent être utilisés dans le code de la même façon qu'un autre type de capteur qui effectue la même fonction. Si le capteur analogique est un potentiomètre mesurant un angle de bras, il peut être utilisé de manière similaire à un *encodeur*. S'il s'agit d'un capteur à ultrasons, il peut être utilisé de manière similaire à d'autres *ultrasonics*.

Il est très important de garder à l'esprit que les potentiomètres ont des limites physiques réelles, et par conséquant, ont généralement une amplitude de mouvement limitée. Des précautions doivent être prises à la fois dans le mécanisme physique et dans le code pour garantir que le mécanisme n'abîme pas le capteur en dépassant sa portée maximale.

15.3.9 Entrées numériques - Partie logicielle

Note : Cette section couvre le logiciel relatif aux entrées numériques. Pour un guide sur le branchement électrique aux entrées numériques, voir *Entrées numériques*.

The roboRIO's FPGA supports up to 26 digital inputs. 10 of these are made available through the built-in DIO ports on the RIO itself, while the other 16 are available through the *MXP* breakout port.

Digital inputs read one of two states - « high » or « low. » By default, the built-in ports on the RIO will read « high » due to internal pull-up resistors (for more information, see *Entrées numériques*). Accordingly, digital inputs are most-commonly used with switches of some sort. Support for this usage is provided through the `DigitalInput` class (Java, C++).

La classe DigitalInput

Une entrée `DigitalInput` peut être initialisée comme suit :

JAVA

```
// Initializes a DigitalInput on DIO 0
DigitalInput input = new DigitalInput(0);
```

C++

```
// Initializes a DigitalInput on DIO 0
frc::DigitalInput input{0};
```

Lecture de la valeur du DigitalInput

L'état de `DigitalInput` peut être interrogé avec la méthode `get` :

JAVA

```
// Gets the value of the digital input.  Returns true if the circuit is open.
input.get();
```

C++

```
// Gets the value of the digital input.  Returns true if the circuit is open.
input.Get();
```

Création d'une entrée numérique à partir d'une entrée analogique

---

Note : Un `AnalogTrigger` construit avec un argument de numéro de port peut partager ce port analogique avec un `AnalogInput` séparé, mais deux objets *AnalogInput* peuvent ne pas partager le même port.

Sometimes, it is desirable to use an analog input as a digital input. This can be easily achieved using the `AnalogTrigger` class (Java, C++).

Un `AnalogTrigger` peut être initialisé comme suit. Comme avec `AnalogPotentiometer`, un `AnalogInput` peut être transmis explicitement si l'utilisateur souhaite personnaliser les paramètres d'échantillonnage :

JAVA

```
// Initializes an AnalogTrigger on port 0
AnalogTrigger trigger0 = new AnalogTrigger(0);

// Initializes an AnalogInput on port 1 and enables 2-bit oversampling
AnalogInput input = new AnalogInput(1);
input.setAverageBits(2);

// Initializes an AnalogTrigger using the above input
AnalogTrigger trigger1 = new AnalogTrigger(input);
```

C++

```
// Initializes an AnalogTrigger on port 0
frc::AnalogTrigger trigger0{0};

// Initializes an AnalogInput on port 1 and enables 2-bit oversampling
frc::AnalogInput input{1};
input.SetAverageBits(2);

// Initializes an AnalogTrigger using the above input
frc::AnalogTrigger trigger1{input};
```

Définition des points de déclenchement

Note : Pour plus de détails sur la mise à l'échelle des données « brutes » `AnalogInput`, voir *Entrées analogiques - Partie logicielle*.

Pour convertir le signal analogique en signal numérique, il est nécessaire de spécifier à quelles valeurs le déclencheur sera activé et désactivé. Ces valeurs sont préférablement différentes, pour que les changement d'états autour du point de transition se fassent de façon nette (sans oscillations) :

JAVA

```
// Sets the trigger to enable at a raw value of 3500, and disable at a value of 1000
trigger.setLimitsRaw(1000, 3500);

// Sets the trigger to enable at a voltage of 4 volts, and disable at a value of 1.5␣
↪volts
trigger.setLimitsVoltage(1.5, 4);
```

C++

```
// Sets the trigger to enable at a raw value of 3500, and disable at a value of 1000
trigger.SetLimitsRaw(1000, 3500);

// Sets the trigger to enable at a voltage of 4 volts, and disable at a value of 1.5␣
↪volts
trigger.SetLimitsVoltage(1.5, 4);
```

Utilisation de DigitalInputs dans le code

As almost all switches on the robot will be used through a `DigitalInput`. This class is extremely important for effective robot control.

Limiter le mouvement d'un mécanisme

Presque tous les mécanismes motorisés (tels que les bras et les élévateurs) en FRC | reg | devrait recevoir une certaine forme d'interrupteur de « fin de course » pour éviter qu'ils ne s'endommagent à la fin de leur plage de mouvements. Un court exemple est donné ci-dessous :

JAVA

```
Spark spark = new Spark(0);

// Limit switch on DIO 2
DigitalInput limit = new DigitalInput(2);

public void autonomousPeriodic() {
    // Runs the motor forwards at half speed, unless the limit is pressed
    if(!limit.get()) {
        spark.set(.5);
    } else {
        spark.set(0);
    }
}
```

C++

```
// Motor for the mechanism
frc::Spark spark{0};

// Limit switch on DIO 2
frc::DigitalInput limit{2};

void AutonomousPeriodic() {
    // Runs the motor forwards at half speed, unless the limit is pressed
    if(!limit.Get()) {
        spark.Set(.5);
    } else {
        spark.Set(0);
    }
}
```

La mise à zéro d'un mécanisme

Limit switches are very important for being able to « home » a mechanism with an encoder. For an example of this, see *La mise à zéro d'un mécanisme*.

15.3.10 Programmation des interrupteurs de fin de course

Limit switches are often used to control mechanisms on robots. While limit switches are simple to use, they only can sense a single position of a moving part. This makes them ideal for ensuring that movement doesn't exceed some limit but not so good at controlling the speed of the movement as it approaches the limit. For example, a rotational shoulder joint on a robot arm would best be controlled using a potentiometer or an absolute encoder. A limit switch could make sure that if the potentiometer ever failed, the limit switch would stop the robot from going too far and causing damage.

Les interrupteurs de fin de course peuvent avoir des sorties « normalement ouvertes » ou « normalement fermées ». Cela permettra de contrôler si un signal au niveau haut signifie que l'interrupteur est ouvert ou fermé. Pour en savoir plus sur la partie matérielle de l'interrupteur de fin de course, consultez cet *article*.

Contrôle d'un moteur à l'aide de deux interrupteurs de fin de course

JAVA

```
DigitalInput toplimitSwitch = new DigitalInput(0);
DigitalInput bottomlimitSwitch = new DigitalInput(1);
PWMVictorSPX motor = new PWMVictorSPX(0);
Joystick joystick = new Joystick(0);

@Override
public void teleopPeriodic() {
    setMotorSpeed(joystick.getRawAxis(2));
}

public void setMotorSpeed(double speed) {
    if (speed > 0) {
        if (toplimitSwitch.get()) {
            // We are going up and top limit is tripped so stop
            motor.set(0);
        } else {
            // We are going up but top limit is not tripped so go at commanded speed
            motor.set(speed);
        }
    } else {
        if (bottomlimitSwitch.get()) {
            // We are going down and bottom limit is tripped so stop
            motor.set(0);
        } else {
            // We are going down but bottom limit is not tripped so go at commanded␣
↪speed
            motor.set(speed);
        }
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }
}
```

C++

```
frc::DigitalInput toplimitSwitch {0};
frc::DigitalInput bottomlimitSwitch {1};
frc::PWMVictorSPX motor {0};
frc::Joystick joystick {0};

void TeleopPeriodic() {
    SetMotorSpeed(joystick.GetRawAxis(2));
}

void SetMotorSpeed(double speed) {
    if (speed > 0) {
        if (toplimitSwitch.Get()) {
            // We are going up and top limit is tripped so stop
            motor.Set(0);
        } else {
            // We are going up but top limit is not tripped so go at commanded speed
            motor.Set(speed);
        }
    } else {
        if (bottomlimitSwitch.Get()) {
            // We are going down and bottom limit is tripped so stop
            motor.Set(0);
        } else {
            // We are going down but bottom limit is not tripped so go at commanded␣
↪speed
            motor.Set(speed);
        }
    }
}
```

15.4 Divers API matériel.

Cette section met en évidence divers API matériels autonomes.

15.4.1 DELs adressables

LED strips have been commonly used by teams for several years for a variety of reasons. They allow teams to debug robot functionality from the audience, provide a visual marker for their robot, and can simply add some visual appeal. WPILib has an API for controlling WS2812 LEDs with their data pin connected via *PWM*.

Instanciation de l'objet adressableLED

You first create an `AddressableLED` object that takes the PWM port as an argument. It *must* be a PWM header on the roboRIO. Then you set the number of LEDs located on your LED strip, which can be done with the `setLength()` function.

> **Avertissement :** Il est important de noter que le réglage de la longueur décrit ci-haut est une tâche relativement longue à exécuter pour le roboRIO, et il n'est **pas** recommandé de l'exécuter périodiquement.

Une fois la longueur de la bande définie, vous devrez créer un objet `AddressableLEDBuffer` qui prendra le nombre de LED comme entrée. Vous appellerez ensuite `myAddressableLed.setData(myAddressableLEDBuffer)` pour définir les données correspondant à chaque LED. Enfin, vous pouvez appeler `myAddressableLed.start()` pour écrire la sortie en continu. Voici un exemple complet du processus d'initialisation.

Note : C++ ne supporte pas un AddressableLEDBuffer et utilise à la place un tableau.

Java

```
17   @Override
18   public void robotInit() {
19     // PWM port 9
20     // Must be a PWM header, not MXP or DIO
21     m_led = new AddressableLED(9);
22
23     // Reuse buffer
24     // Default to a length of 60, start empty output
25     // Length is expensive to set, so only set it once, then just update data
26     m_ledBuffer = new AddressableLEDBuffer(60);
27     m_led.setLength(m_ledBuffer.getLength());
28
29     // Set the data
30     m_led.setData(m_ledBuffer);
31     m_led.start();
32   }
```

C++

```
11 class Robot : public frc::TimedRobot {
12  private:
13   static constexpr int kLength = 60;
14
15   // PWM port 9
16   // Must be a PWM header, not MXP or DIO
17   frc::AddressableLED m_led{9};
18   std::array<frc::AddressableLED::LEDData, kLength>
19       m_ledBuffer;  // Reuse the buffer
20   // Store what the last hue of the first pixel is
21   int firstPixelHue = 0;
```

```
void Robot::RobotInit() {
  // Default to a length of 60, start empty output
  // Length is expensive to set, so only set it once, then just update data
  m_led.SetLength(kLength);
  m_led.SetData(m_ledBuffer);
  m_led.Start();
}
```

Note : The roboRIO only supports only 1 `AddressableLED` object. As WS2812B LEDs are connected in series, you can drive several strips connected in series from from `AddressableLED` object.

Définition de la bande entière sur une seule couleur

La couleur peut être définie sur chaque DEL spécifique sur la bande en utilisant deux méthodes. `setRGB()` qui prend les valeurs RGB comme entrée et `setHSV()` qui prend les valeurs HSV comme entrée.

Utilisation des valeurs RGB

RGB signifie rouge, vert et bleu. Il s'agit d'un modèle de couleur assez courant car il est assez facile à comprendre. Les LED peuvent être réglées avec la méthode `setRGB` qui prend 4 arguments : index (ou position de la LED sur la bande), valeur pour le rouge, valeur pour le vert, valeur pour le bleu. Les valeurs de rouge, vert et bleu sont des entiers compris entre 0 et 255.

Java

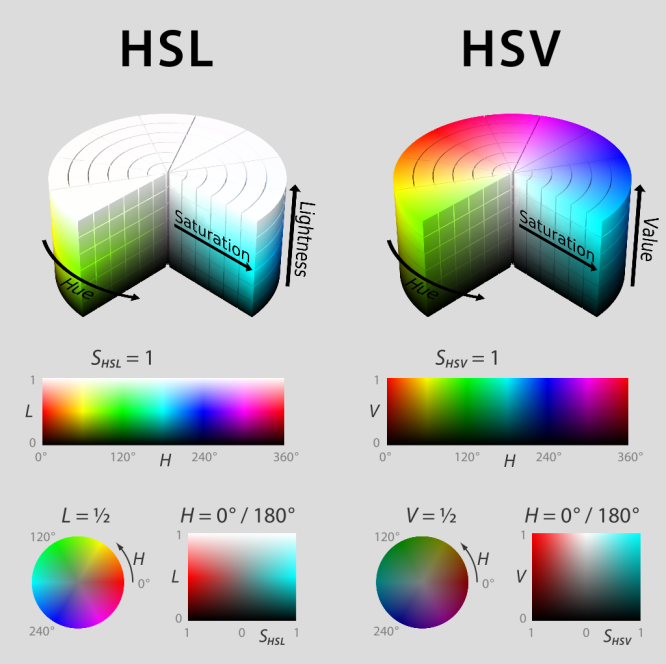
```
for (var i = 0; i < m_ledBuffer.getLength(); i++) {
   // Sets the specified LED to the RGB values for red
   m_ledBuffer.setRGB(i, 255, 0, 0);
}

m_led.setData(m_ledBuffer);
```

C++

```
for (int i = 0; i < kLength; i++) {
   m_ledBuffer[i].SetRGB(255, 0, 0);
}

m_led.SetData(m_ledBuffer);
```

Utilisation des valeurs HSV

HSV signifie Hue, Saturation et Value, ou Teinte, Saturation et Valeur. La teinte correspond à la couleur, la saturation étant la quantité de gris et la valeur étant la luminosité. Dans WPILib, Hue est un entier compris entre 0 et 180. Saturation et Value sont des nombres entiers compris entre 0 et 255. Si vous regardez un sélecteur de couleurs comme celui de Google, La teinte sera de 0 à 360 et la saturation et la valeur de 0% à 100%. C'est de la même façon qu'OpenCV gère les couleurs HSV. Assurez-vous que les valeurs HSV saisies dans WPILib sont correctes, sinon la couleur pourrait différer de celle attendue.



Les LED peuvent être réglées avec la méthode `setHSV` qui prend 4 arguments : index de la LED, Hue, Saturation et Value. Un exemple est illustré ci-dessous pour définir la couleur d'une bande LED sur rouge (teinte de 0).

Java

```
for (var i = 0; i < m_ledBuffer.getLength(); i++) {
   // Sets the specified LED to the HSV values for red
   m_ledBuffer.setHSV(i, 0, 100, 100);
}

m_led.setData(m_ledBuffer);
```

C++

```
for (int i = 0; i < kLength; i++) {
   m_ledBuffer[i].SetHSV(0, 100, 100);
}

m_led.SetData(m_ledBuffer);
```

Création d'un effet arc-en-ciel

La méthode ci-dessous fait plusieurs opérations. À l'intérieur de la boucle *for*, la teinte (ou Hue) est distribuée de façon uniforme sur la bande de LED et stocke la teinte individuelle des LED dans une variable appelée `hue`. Ensuite, la boucle *for* définit la valeur HSV de ce pixel spécifié en utilisant la valeur `hue`.

À l'extérieur de la boucle *for*, le `m_rainbowFirstPixelHue` itère ensuite le pixel qui contient la teinte « initiale » créant l'effet arc-en-ciel. `m_rainbowFirstPixelHue` vérifie ensuite pour s'assurer que la teinte est à l'intérieur des limites de teinte de 180. En effet, la teinte HSV est une valeur de 0 à 180.

Note : C'est une bonne pratique de programmation de garder la méthode `robotPeriodic()` aussi claire et concise que possible, nous allons donc créer une méthode pour gérer la configuration de nos données LED. Nous appellerons cette méthode `rainbow()` et l'appellerons de `robotPeriodic()`.

Java

```
42  private void rainbow() {
43    // For every pixel
44    for (var i = 0; i < m_ledBuffer.getLength(); i++) {
45      // Calculate the hue - hue is easier for rainbows because the color
46      // shape is a circle so only one value needs to precess
47      final var hue = (m_rainbowFirstPixelHue + (i * 180 / m_ledBuffer.getLength()))
    ↪% 180;
48      // Set the value
49      m_ledBuffer.setHSV(i, hue, 255, 128);
50    }
51    // Increase by to make the rainbow "move"
52    m_rainbowFirstPixelHue += 3;
53    // Check bounds
54    m_rainbowFirstPixelHue %= 180;
55  }
```

C++

```
void Robot::Rainbow() {
  // For every pixel
  for (int i = 0; i < kLength; i++) {
    // Calculate the hue - hue is easier for rainbows because the color
    // shape is a circle so only one value needs to precess
    const auto pixelHue = (firstPixelHue + (i * 180 / kLength)) % 180;
    // Set the value
    m_ledBuffer[i].SetHSV(pixelHue, 255, 128);
  }
  // Increase by to make the rainbow "move"
  firstPixelHue += 3;
  // Check bounds
  firstPixelHue %= 180;
}
```

Maintenant que nous avons créé notre méthode rainbow(), nous devons l'appeler et définir les données qui iront à la bande de LED.

Java

```
  @Override
  public void robotPeriodic() {
    // Fill the buffer with a rainbow
    rainbow();
    // Set the LEDs
    m_led.setData(m_ledBuffer);
  }
```

C++

```
void Robot::RobotPeriodic() {
  // Fill the buffer with a rainbow
  Rainbow();
  // Set the LEDs
  m_led.SetData(m_ledBuffer);
}
```

15.5 Contrôleurs de moteurs

Un contrôleur de moteur est chargé sur votre robot de faire bouger les moteurs. Pour les moteurs à courant continu à balais tels que le *CIM* ou 775, le contrôleur de moteur régule la tension que le moteur reçoit, un peu comme une ampoule. Pour les contrôleurs de moteur sans balais tels que le Spark MAX, le contrôleur régule la puissance délivrée à chaque « phase » du moteur.

Note : Un autre nom pour un contrôleur de moteur est un contrôleur de vitesse.

Indication : On peut fabriquer un contrôleur de moteur rapidement (mais non-conforme au normes FRC) en retirant le moteur d'une perceuse sans fil avec balais (non brushless) et en attachant des PowerPoles ou des équivalents aux fils du moteur. Assurez-vous que la tension fournie par la perceuse n'endommagera pas le moteur, mais notez que le 775 fonctionne bien jusqu'à 24 volts.

Avertissement : Attention ! Brancher un moteur de type « Brushless » directement à l'alimentation, comme un moteur à balais conventionnel, détruira le moteur !

15.5.1 Contrôleurs de moteur légaux FRC

Motor controllers come in lots of shapes, sizes and feature sets. This is the full list of FRC® Legal motor controllers as of 2024 :

- DMC 60/DMC 60c (P/N : 410-334-1, 410-334-2)
- Jaguar Motor Controller (P/N : MDL-BDC, MDL-BDC24, and 217-3367) branché seulement sur *PWM*
- Moteur Nidec Dynamo BLDC avec contrôleur (P/N 840205-000, am-3740)
- SD540 (P/N : SD540x1, SD540x2, SD540x4, SD540Bx1, SD540Bx2, SD540Bx4, SD540C)
- Spark Flex Motor Controller (P/N REV-11-2159, am-5276)
- Spark Motor Controller (P/N : REV-11-1200, am-4260)
- Spark MAX Motor Controller (P/N : REV-11-2158, am-4261)
- Talon FX Motor Controller (P/N 217-6515, 19-708850, am-6515, am-6515_Short, WCP-0940) for controlling integral Falcon 500 or Kraken X60 only,
- Talon (P/N : CTRE_Talon, CTRE_Talon_SR, and am-2195)
- Talon SRX (P/N : 217-8080, am-2854, 14-838288)
- Moteur Venom avec contrôleur (P/N BDC-10001) pour contrôler uniquement le moteur correspondant
- Victor 884 (P/N : VICTOR-884-12/12)
- Victor 888 (P/N : 217-2769)
- Victor SP (P/N : 217-9090, am-2855, 14-868380)
- Victor SPX (P/N : 217-9191, 17-868388, am-3748)

15.6 La pneumatique

La pneumatique est un moyen rapide et facile de fabriquer quelque chose qui se trouve dans un état ou un autre en utilisant de l'air comprimé. Pour plus d'informations sur le fonctionnement des systèmes pneumatiques, voir *API pneumatiques*.

15.6.1 Dispositifs pneumatiques légaux pour la FRC

— Module de commande pneumatique (P/N : am-2858, 217-4243)
— Concentrateur pneumatique (P/N REV-11-1852)

15.7 Relais

Un relais contrôle la puissance d'un moteur ou d'un système électronique personnalisé selon le mode on/off (tout ou rien).

15.7.1 Modules de relais légaux pour la FRC

— Relais Spike H-Bridge (P/N : 217-0220 et SPIKE-RELAY-H)
— Relais direct d'automatisation (P/N : AD-SSR6M12-DC200D, AD-SSR6M25-DC200D, AD-SSR6M40-DC200D)
— Power Distribution Hub (PDH) switched channel (P/N REV-11-1850)

16

Dispositifs avec bus CAN

16.1 Utilisation de périphériques CAN

CAN has many advantages over other methods of connection between the robot controller and peripheral devices.

- Les connexions CAN sont connectées en série d'un périphérique à l'autre, ce qui se traduit souvent par des câbles beaucoup plus courts en comparaison de câbler chaque périphérique directement au roboRIO.
- Much more data can be sent over a CAN connection than over a *PWM* connection - thus, CAN motor controllers are capable of a much more expansive feature-set than are PWM motor controllers.
- Le CAN est bidirectionnel, de sorte que les contrôleurs de moteur CAN peuvent renvoyer des données au roboRIO, facilitant à nouveau un ensemble de fonctionnalités plus étendu que celui offert par les contrôleurs PWM.

For instructions on wiring CAN devices, see the relevant section of the *robot wiring guide*.

Les périphériques CAN ont généralement leurs propres classes WPILib. Les sections suivantes décrivent l'utilisation de plusieurs de ces classes.

16.2 Module de commande pneumatique (PCM)

The Pneumatics Control Module (*PCM*) is a *CAN*-based device that provides complete control over the compressor and up to 8 solenoids per module. The PCM is integrated into WPILib through a series of classes that make it simple to use.

The closed loop control of the Compressor and Pressure switch is handled by the `Compressor` class (Java, C++, `Python`), and the Solenoids are handled by the `Solenoid` (Java, C++, `Python`) and `DoubleSolenoid` (Java, C++, `Python`) classes.

An additional PCM module can be used where the module's corresponding solenoids are differentiated by the module number in the constructors of the Solenoid and Compressor classes.

For more information on controlling the compressor, see *Operating a Compressor for Pneumatics*.

For more information on controlling solenoids, see *Operating Pneumatic Cylinders*.

16.3 Concentrateur pneumatique

The Pneumatic Hub (*PH*) is a *CAN*-based device that provides complete control over the compressor and up to 16 solenoids per module. The PH is integrated into WPILib through a series of classes that make it simple to use.

The closed loop control of the Compressor and Pressure switch is handled by the `Compressor` class (Java, C++, `Python`), and the Solenoids are handled by the `Solenoid` (Java, C++, `Python`) and `DoubleSolenoid` (Java, C++, `Python`) classes.

An additional PH module can be used where the module's corresponding solenoids are differentiated by the module number in the constructors of the Solenoid and Compressor classes.

For more information on controlling the compressor, see *Operating a Compressor for Pneumatics*.

For more information on controlling solenoids, see *Operating Pneumatic Cylinders*.

16.4 Module de distribution d'alimentation (PDP)

The CTRE Power Distribution Panel (*PDP*) and Rev Power Distribution Hub (*PDH*) can use their *CAN* connectivity to communicate a wealth of status information regarding the robot's power use to the roboRIO, for use in user code. This has the capability to report current temperature, the bus voltage, the total robot current draw, the total robot energy use, and the individual current draw of each device power channel. This data can be used for a number of advanced control techniques, such as motor *torque* limiting and brownout avoidance.

16.4.1 Création d'un objet PowerDistribution

To use the either Power Distribution module, create an instance of the `PowerDistribution` class (Java, C++, Python). With no arguments, the Power Distribution object will be detected, and must use CAN ID of 0 for CTRE or 1 for REV. If the CAN ID is non-default, additional constructors are available to specify the CAN ID and type.

JAVA

```
PowerDistribution examplePD = new PowerDistribution();
PowerDistribution examplePD = new PowerDistribution(0, ModuleType.kCTRE);
PowerDistribution examplePD = new PowerDistribution(1, ModuleType.kRev);
```

C++

```
PowerDistribution examplePD{};
PowerDistribution examplePD{0, frc::PowerDistribution::ModuleType::kCTRE};
PowerDistribution examplePD{1, frc::PowerDistribution::ModuleType::kRev};
```

PYTHON

```
from wpilib import PowerDistribution

examplePD = PowerDistribution()
examplePD = PowerDistribution(0, PowerDistribution.ModuleType.kCTRE)
examplePD = PowerDistribution(1, PowerDistribution.ModuleType.kRev)
```

Remarque : il n'est pas nécessaire de créer un objet PowerDistribution, sauf si vous avez besoin d'en lire des valeurs. La carte fonctionnera et alimentera tous les canaux même si l'objet n'est jamais créé.

Avertissement : Pour activer l'enregistrement de la tension et du courant dans l'application Driver Station, l'ID CAN du panneau de distribution d'alimentation CTRE *doit* être configuré avec la valeur 0, et pour le concentrateur de distribution d'alimentation REV, il *doit* être configuré avec la valeur 1.

16.4.2 Lecture de la tension sur le système électrique

JAVA

```
32      // Get the voltage going into the PDP, in Volts.
33      // The PDP returns the voltage in increments of 0.05 Volts.
34      double voltage = m_pdp.getVoltage();
35      SmartDashboard.putNumber("Voltage", voltage);
```

C++

```
28    // Get the voltage going into the PDP, in Volts.
29    // The PDP returns the voltage in increments of 0.05 Volts.
30    double voltage = m_pdp.GetVoltage();
31    frc::SmartDashboard::PutNumber("Voltage", voltage);
```

PYTHON

```
34        # Get the voltage going into the PDP, in Volts.
35        # The PDP returns the voltage in increments of 0.05 Volts.
36        voltage = self.pdp.getVoltage()
37        wpilib.SmartDashboard.putNumber("Voltage", voltage)
```

La surveillance de la tension du système électrique peut être utile (entre autres) pour détecter quand la batterie du robot faiblit, et que la tension actuelle s'approche de la tension minimale d'opération. Des mesures de prévention peuvent être alors prises pour éviter la panne incontrôlée du robot. Voir le document *roboRIO Brownouts* pour plus d'informations.

16.4.3 Lecture de la température

JAVA

```
37    // Retrieves the temperature of the PDP, in degrees Celsius.
38    double temperatureCelsius = m_pdp.getTemperature();
39    SmartDashboard.putNumber("Temperature", temperatureCelsius);
```

C++

```
33    // Retrieves the temperature of the PDP, in degrees Celsius.
34    double temperatureCelsius = m_pdp.GetTemperature();
35    frc::SmartDashboard::PutNumber("Temperature", temperatureCelsius);
```

PYTHON

```
39        # Retrieves the temperature of the PDP, in degrees Celsius.
40        temperatureCelsius = self.pdp.getTemperature()
41        wpilib.SmartDashboard.putNumber("Temperature", temperatureCelsius)
```

La surveillance de la température peut être utile pour détecter si le robot a consommé trop d'énergie et doit être mis à l'arrêt pendant un certain temps, ou encore, s'il y a un court-circuit ou un autre problème de câblage qui crée un surchauffement.

16.4.4 Lecture du courant total, de la puissance et de l'énergie

JAVA

```java
    // Get the total current of all channels.
    double totalCurrent = m_pdp.getTotalCurrent();
    SmartDashboard.putNumber("Total Current", totalCurrent);

    // Get the total power of all channels.
    // Power is the bus voltage multiplied by the current with the units Watts.
    double totalPower = m_pdp.getTotalPower();
    SmartDashboard.putNumber("Total Power", totalPower);

    // Get the total energy of all channels.
    // Energy is the power summed over time with units Joules.
    double totalEnergy = m_pdp.getTotalEnergy();
    SmartDashboard.putNumber("Total Energy", totalEnergy);
```

C++

```cpp
    // Get the total current of all channels.
    double totalCurrent = m_pdp.GetTotalCurrent();
    frc::SmartDashboard::PutNumber("Total Current", totalCurrent);

    // Get the total power of all channels.
    // Power is the bus voltage multiplied by the current with the units Watts.
    double totalPower = m_pdp.GetTotalPower();
    frc::SmartDashboard::PutNumber("Total Power", totalPower);

    // Get the total energy of all channels.
    // Energy is the power summed over time with units Joules.
    double totalEnergy = m_pdp.GetTotalEnergy();
    frc::SmartDashboard::PutNumber("Total Energy", totalEnergy);
```

PYTHON

```python
        # Get the total current of all channels.
        totalCurrent = self.pdp.getTotalCurrent()
        wpilib.SmartDashboard.putNumber("Total Current", totalCurrent)

        # Get the total power of all channels.
        # Power is the bus voltage multiplied by the current with the units Watts.
        totalPower = self.pdp.getTotalPower()
        wpilib.SmartDashboard.putNumber("Total Power", totalPower)

        # Get the total energy of all channels.
        # Energy is the power summed over time with units Joules.
        totalEnergy = self.pdp.getTotalEnergy()
        wpilib.SmartDashboard.putNumber("Total Energy", totalEnergy)
```

La surveillance du courant total, de la puissance et de l'énergie peut être utile pour contrôler la quantité d'énergie tirée de la batterie, à la fois pour prévenir les pannes de courant et pour s'assurer que les mécanismes disposent d'une puissance suffisante pour effectuer les actions

requises. La puissance est obtenue en multipliant la tension du bus par le courant et pour unité le Watt. L'énergie est la puissance additionnée dans le temps et a pour unité le Joule.

16.4.5 Lecture des courants des canaux individuels

Le PDP/PDH permet également aux utilisateurs de lire le courant consommé par les canaux d'alimentation de chaque composant. Vous pouvez lire le courant de n'importe lequel des 16 canaux du PDP (0-15) ou des 24 canaux du PDH (0-23).

JAVA

```
    // Get the current going through channel 7, in Amperes.
    // The PDP returns the current in increments of 0.125A.
    // At low currents the current readings tend to be less accurate.
    double current7 = m_pdp.getCurrent(7);
    SmartDashboard.putNumber("Current Channel 7", current7);
```

C++

```
    // Get the current going through channel 7, in Amperes.
    // The PDP returns the current in increments of 0.125A.
    // At low currents the current readings tend to be less accurate.
    double current7 = m_pdp.GetCurrent(7);
    frc::SmartDashboard::PutNumber("Current Channel 7", current7);
```

PYTHON

```
        # Get the current going through channel 7, in Amperes.
        # The PDP returns the current in increments of 0.125A.
        # At low currents the current readings tend to be less accurate.
        current7 = self.pdp.getCurrent(7)
        wpilib.SmartDashboard.putNumber("Current Channel 7", current7)
```

La surveillance de la consommation de courant de chaque dispositif peut être utile pour détecter les courts-circuits ou les moteurs bloqués.

16.4.6 Utilisation du canal commutable (PDH)

Le REV PDH est pourvu d'un canal qui peut être activé ou désactivé pour contrôler des circuits personnalisés.

JAVA

```
examplePD.setSwitchableChannel(true);
examplePD.setSwitchableChannel(false);
```

C++

```
examplePD.SetSwitchableChannel(true);
examplePD.SetSwitchableChannel(false);
```

PYTHON

```
examplePD.setSwitchableChannel(True)
examplePD.setSwitchableChannel(False)
```

16.5 Périphériques CAN provenant de tierce-partie

Un certain nombre de fournisseurs FRC® proposent leurs propres périphériques *CAN*. Comme les appareils CAN offrent un ensemble étendu de fonctionnalités, les appareils CAN des fournisseurs nécessitent des librairies de codes tout aussi étendues pour fonctionner. Par conséquent, ces librairies ne sont pas gérées en tant que partie officielle de WPILib, mais sont plutôt gérées par les fournisseurs eux-mêmes. Pour un guide sur l'installation de librairies tierces, consultez *3rd Party Libraries*

Une liste des périphériques CAN tiers courants de divers fournisseurs, ainsi que des liens vers la documentation externe correspondante, sont fournis ci-dessous :

16.5.1 CTR Electronics

CTR Electronics (CTRE) propose plusieurs périphériques CAN avec des bibliothèques externes. Les ressources générales pour tous les composants CTRE incluent :

> Documentation accompagnant l'utilitaire phoenix

Contrôleurs de moteur CTRE

- **Talon FX (avec moteur Falcon 500)**
  - Documentation d'API (v5 : Java, C++ | Pro : Java, C++)
  - Manuel d'utilisation du matériel
  - Documentation logicielle (v5, Pro)
- **Talon SRX**
  - Documentation d'API (Java, C++)
  - Manuel d'utilisation du matériel
  - Documentation logicielle
- **Victor SPX**
  - Documentation d'API (Java, C++)

- Manuel d'utilisation du matériel
- Documentation logicielle

Capteurs CTRE

- **CANcoder**
  - Documentation d'API (v5 : Java, C++ | Pro : Java, C++)
  - Manuel d'utilisation du matériel
  - Documentation logicielle (v5, Pro)
- **Pigeon 2.0**
  - Documentation d'API (v5 : Java, C++ | Pro : Java, C++)
  - Manuel d'utilisation du matériel
  - Documentation logicielle (v5, Pro)
- **Pigeon IMU**
  - Documentation d'API (Java, C++)
  - Manuel d'utilisation du matériel
  - Documentation logicielle
- **CANifier**
  - Documentation d'API (Java, C++)
  - Manuel d'utilisation du matériel
  - Documentation logicielle

CTRE - autres composants CAN

- **CANdle LED Controller**
  - Documentation d'API (Java, C++)
  - Manuel d'utilisation du matériel
  - Documentation logicielle

16.5.2 REV Robotics

REV Robotics currently offers the SPARK MAX and SPARK Flex motor controllers which can be used for brushed and REV brushless (NEO, NEO 550, and NEO Vortex) motors.

Contrôleurs de moteur REV

- **SPARK MAX**
  - API Documentation (Java, C++)
  - Technical Manual
- **SPARK Flex**
  - API Documentation (Java, C++)
  - Technical Manual

16.5.3 Playing With Fusion

Playing With Fusion (PWF) propose le moteur/contrôleur intégré Venom ainsi qu'un capteur de distance de temps de vol :

Contrôleurs de moteur PWF

- **Venom**
  - Documentation de l'API (Java, C++)
  - Manuel technique

Capteurs PWF

- **Capteur de temps de vol**
  - Documentation de l'API (Java, C++)
  - Manuel technique

16.5.4 Redux Robotics

Redux Robotics currently offers the Canandcoder *CAN* + *PWM* magnetic encoder and the Canandcolor *CAN*-enabled color sensor.

Capteurs Redux

- **Canandcoder**
  - API Documentation (Java, C++)
  - **`Manuel technique https ://docs.reduxrobotics.com/canandcoder/index.html>`__**
- **Canandcolor**
  - API Documentation (Java, C++)
  - Technical Manual

16.6 Spécifications du périphérique FRC CAN

This document seeks to describe the basic functions of the current FRC® *CAN* system and the requirements for any new CAN devices seeking to work with the system.

16.6.1 Adressage

Les nœuds FRC CAN attribuent des numéros ID d'arbitrage en fonction d'un schéma prédéfini qui divise l'ID en 5 composants :

Type d'appareil

Il s'agit d'une valeur de 5 bits décrivant le type de périphérique adressé. Un tableau des types de dispositif actuellement attribués se trouve ci-dessous. Si vous souhaitez qu'un nouveau type dispositif soit attribué à partir du pool `Réservé`, veuillez soumettre une demande à FIRST.

Types de dispositif	
Diffusion de messages	0
Contrôleur de robot	1
Contrôleur de moteur	2
Contrôleur de relais	3
Capteur gyroscopique	4
Accéléromètre	5
Capteur à ultrasons	6
Capteur à dents d'engrenage (Gear Tooth)	7
Module de distribution d'alimentation (PDP)	8
Contrôleur pneumatique (PCM)	9
Divers	10
Breakout IO	11
Réservé	12-30
Mise à jour du firmware	31

Manufacturier

Il s'agit d'une valeur de 8 bits indiquant le fabricant du périphérique CAN. Les valeurs actuellement attribuées se trouvent dans le tableau ci-dessous. Si vous souhaitez avoir un ID de fabricant attribué à partir du pool `Réservé` veuillez soumettre une demande à FIRST.

Manufacturier	
Broadcast	0
NI	1
Luminary Micro	2
DEKA	3
CTR Electronics	4
REV Robotics	5
Grapple	6
MindSensors	7
Team Use	8
Kauai Labs	9
Copperforge	10
Playing With Fusion	11
Studica	12
The Thrifty Bot	13
Redux Robotics	14
AndyMark	15
Vivid Hosting	16
Réservé	17-255

API / identificateur de message

L'API ou l'identificateur de message est une valeur 10 bits qui identifie une commande ou un type de message particulier. Ces identifiants sont uniques pour chaque combinaison Fabricant + Type d'appareil (donc un identifiant API qui peut être un « Voltage Set » pour un contrôleur de micromoteur lumineux peut être un « Status Get » pour un contrôleur de moteur CTR Electronics ou un `Current Get` pour un module de distribution de puissance CTR).

L'identificateur de message est en outre divisé en 2 sous-champs : la classe API 6 bits et l'index API 4 bits.

Classe API

La classe API est un identifiant 6 bits pour un regroupement d'API. Des messages similaires sont regroupés dans une seule classe API. Un exemple des classes API pour le contrôleur de moteur Jaguar est présenté dans le tableau ci-dessous.

Classe API	
Voltage Control Mode	0
Speed Control Mode	1
Voltage Compensation Mode	2
Position Control Mode	3
Current Control Mode	4
Status	5
Periodic Status	6
Configuration	7
Ack	8

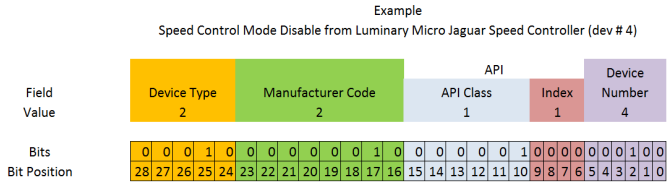
L'index API

L'index API est un identifiant 4 bits pour un message particulier au sein d'une classe API. Un exemple des valeurs d'index API pour la classe API de contrôle de vitesse du contrôleur de moteur Jaguar est présenté dans le tableau ci-dessous.

L'index API	
Enable Control	0
Disable Control	1
Set Setpoint	2
P Constant	3
I Constant	4
D Constant	5
Set Reference	6
Trusted Enable	7
Trusted Set No Ack	8
Trusted Set Setpoint No Ack	10
Set Setpoint No Ack	11

Numéro de périphérique

Le numéro de périphérique est une quantité de 6 bits indiquant le numéro du périphérique d'un type particulier. Les périphériques doivent avoir par défaut l'ID de périphérique 0 pour correspondre aux autres composants du système de contrôle FRC. Le périphérique 0x3F peut être réservé pour les messages de diffusion spécifiques au périphérique.



16.6.2 Cadres protégés

FRC CAN Les nœuds qui implémentent la capacité de contrôle du dispositif actionneur (contrôleurs de moteur, relais, contrôleurs pneumatiques, etc.) doivent mettre en œuvre un moyen de vérifier que le robot est activé et que les commandes proviennent du contrôleur de robot principal (c'est-à-dire le roboRIO).

16.6.3 Diffusion de messages

Les messages de diffusion sont des messages envoyés à tous les nœuds en définissant les champs « Type de dispositif » et « Manufacturier » à 0. La classe API pour les messages de diffusion est 0. Les messages de diffusion présentement définis sont listés dans le tableau ci-dessous :

Description	
Disable	0
System Halt	1
System Reset	2
Device Assign	3
Device Query	4
Heartbeat	5
Sync	6
Update	7
Firmware Version	8
Enumerate	9
System Resume (Reprise du système)	10

Devices should disable immediately when receiving the Disable message (arbID 0). Implementation of other broadcast messages is optional.

16.6.4 Configuration requise pour les nœuds CAN FRC

Pour que les nœuds CAN soient acceptés dans le système FRC, ils doivent :

- Communiquer en utilisant des ID d'arbitrage qui correspondent au format FRC prescrit :
  - Utiliser un type de périphérique CAN valide et légal (selon le tableau 1 - Types de périphériques CAN)
  - Utiliser un ID de fabricant valide et légal (selon le tableau 2 - Codes de fabricant CAN)
  - Utiliser la (les) classe (s) et index API attribués et documentés par le fabricant du périphérique
  - Avoir un numéro de dispositif qui peut être choisi par l'utilisateur dans le cas ou plusieurs unités du même type de dispositif sont destinées à coexister sur le même réseau.
- Prendre en charge les exigences minimales des messages de diffusion, comme indiqué dans la section Messages de diffusion.
- If controlling actuators, utilize a scheme to assure that the robot is issuing commands, is enabled, and is still present.
- Provide software library support for LabVIEW, C++, and Java or arrange with *FIRST*® or FIRST's Control System Partners to provide such interfaces.

16.6.5 Battement de cœur universel

The roboRIO provides a universal CAN heartbeat that any device on the bus can listen and react to. This heartbeat is sent every 20 ms. The heartbeat has a full CAN ID of `0x01011840` (which is the NI Manufacturer ID, RobotController type, Device ID 0 and API ID `0x061`). It is an 8 byte CAN packet with the following bitfield layout.

Description	Byte	Width (bits)
Match time (seconds)	8	8
Match number	6-7	10
Replay number	6	6
Red alliance	5	1
Enabled	5	1
Autonomous mode	5	1
Test mode	5	1
System watchdog	5	1
Tournament type	5	3
Time of day (year)	4	6
Time of day (month)	3-4	4
Time of day (day)	3	5
Time of day (seconds)	2-3	6
Time of day (minutes)	1-2	6
Time of day (hours)	1	5

```
struct [[gnu::packed]] RobotState {
  uint64_t matchTimeSeconds : 8;
  uint64_t matchNumber : 10;
  uint64_t replayNumber : 6;
  uint64_t redAlliance : 1;
  uint64_t enabled : 1;
```

(suite sur la page suivante)

(suite de la page précédente)

```
  uint64_t autonomous : 1;
  uint64_t testMode : 1;
  uint64_t systemWatchdog : 1;
  uint64_t tournamentType : 3;
  uint64_t timeOfDay_yr : 6;
  uint64_t timeOfDay_month : 4;
  uint64_t timeOfDay_day : 5;
  uint64_t timeOfDay_sec : 6;
  uint64_t timeOfDay_min : 6;
  uint64_t timeOfDay_hr : 5;
};
```

If the `System watchdog` flag is set, motor controllers are enabled. If 100 ms has passed since this packet was received, the robot program can be considered hung, and devices should act as if the robot has been disabled.

Note that all fields except `Enabled`, `Autonomous mode`, `Test mode`, and `System watchdog` will contain invalid values until an arbitrary time after the Driver Station connects.

17

Programmation de base

17.1 Introduction au contrôle de version Git

Important : Un guide plus détaillé sur Git est disponible sur le site Web de Git.

Git est un système de gestion de versions distribuée (VCS) créé par Linus Torvalds, également connu pour la création et la maintenance du noyau Linux. Un système de gestion de versions est un système de suivi des modifications de code pour les développeurs. Les avantages du système de gestion de versions Git sont les suivants :

- Separation of testing environments into *branches*
- Il offre la possibilité de naviguer vers un *commit* particulier sans supprimer l'historique
- Il offre la capacité à gérer *les commits* de différentes manières, y compris en les combinant
- Et comporte diverses autres fonctionnalités, voir ici

17.1.1 Conditions préalables

Important : Ce didacticiel utilise le système d'exploitation Windows

Vous devez télécharger et installer Git à partir des liens suivants :

- Windows
- macOS
- Linux

Note : Vous devrez peut-être ajouter Git à votre chemin de répertoire

17.1.2 Le vocabulaire Git

Git revolves around several core data structures and commands :

— **Repository (Référentiel de données) :** la structure des données de votre code, y compris un dossier `.git` dans le répertoire racine
— **Commit :** a particular saved state of the repository, which includes all files and additions
— **Branch :** a means of grouping a set of commits. Each branch has a unique history. This is primarily used for separating development and stable branches.
— **Push (Pousser) :** mettez à jour le référentiel distant avec vos modifications locales
— **Pull (Tirer) :** mettez à jour votre référentiel local avec les modifications à distance
— **Clone :** retrieve a local copy of a repository to modify
— **Fork :** duplicate a pre-existing repository to modify, and to compare against the original
— **Merge :** combine various changes from different branches/commits/forks into a single history

17.1.3 Le Référentiel de données

Un référentiel Git est une structure de données contenant la structure, l'historique et les fichiers d'un projet.

Les référentiels Git comprennent généralement :

— Un dossier `.git`. Ce dossier contient les différentes informations sur le référentiel.
— Un fichier `.gitignore`. Ce fichier contient les fichiers ou répertoires que vous ne voulez *pas* inclure lors de la validation.
— Fichiers et dossiers. Il s'agit du contenu principal du référentiel.

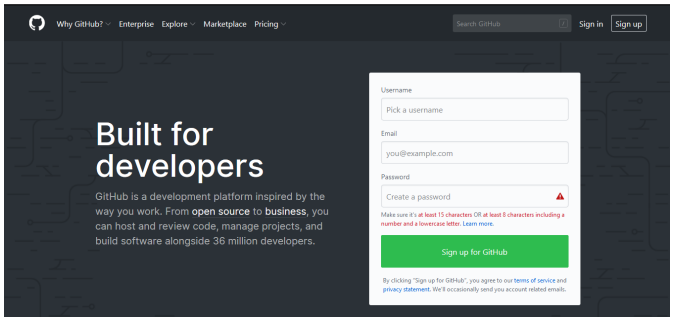
Création du Référentiel de données

You can store the repository locally, or through a remote – a remote being the cloud, or possibly another storage medium or server that hosts your repository. GitHub is a popular free hosting service. Numerous developers use it, and that's what this tutorial will use.

Note : Il existe différents fournisseurs qui peuvent héberger des dépôts. Gitlab et Bitbucket sont quelques alternatives à Github.

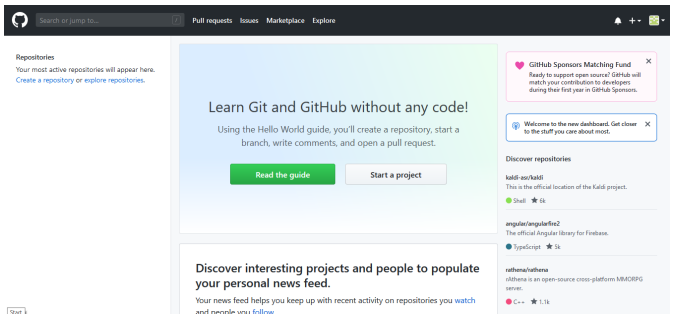
Création d'un compte GitHub

Go ahead and create a GitHub account by visiting the website and following the on-screen prompts.

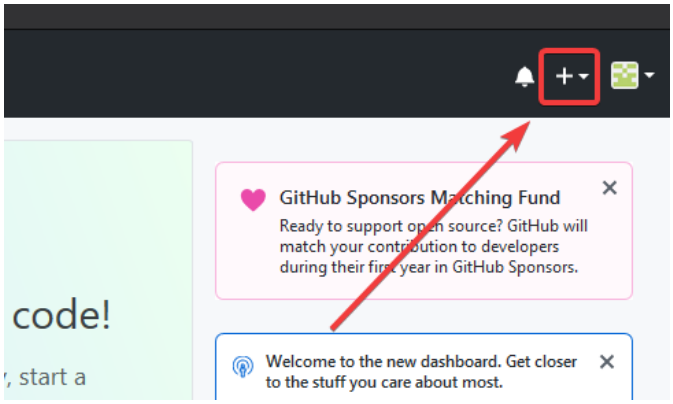


Création locale

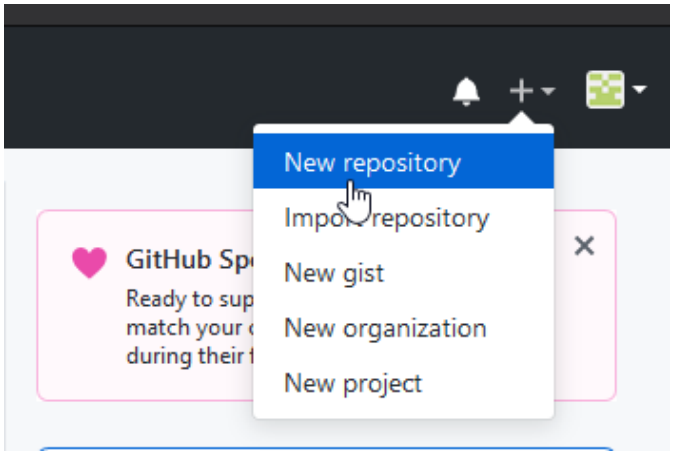
Après avoir créé et vérifié votre compte, vous irez ensuite visiter la page d'accueil. Cela ressemblera à l'image montrée ci-dessous.



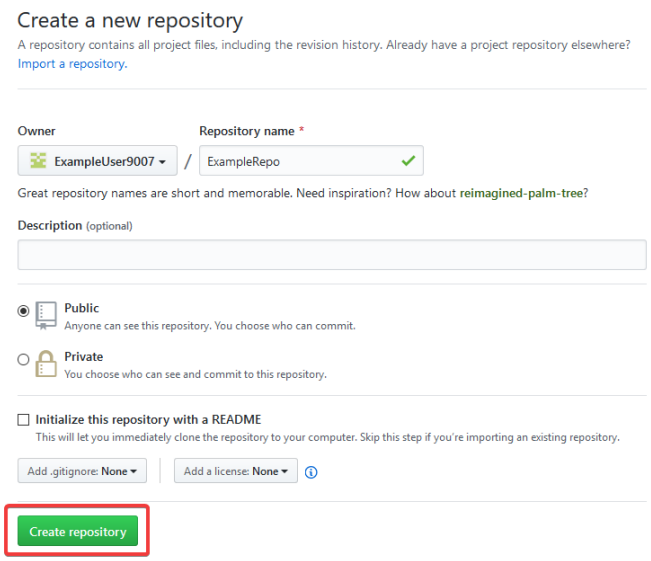
Cliquez sur l'icône « plus », ou + en haut à droite.



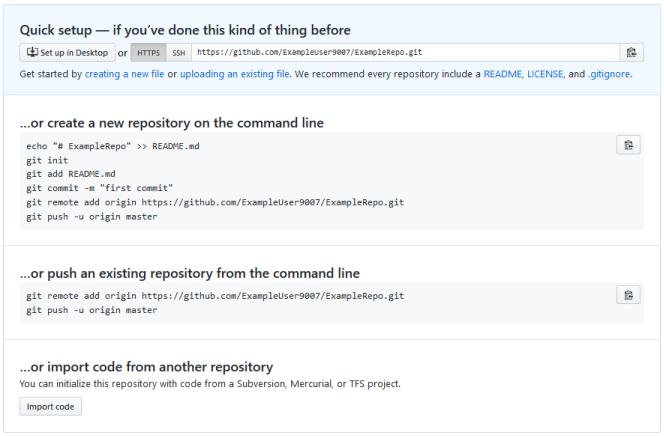
Cliquez ensuite sur « *New Repository* »



Remplissez les informations appropriées, puis cliquez sur « *Create repository* »

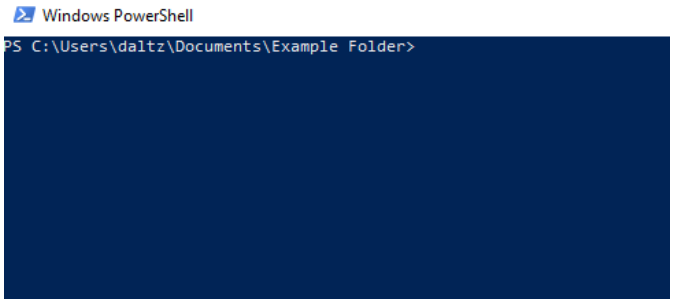


Vous devriez voir un écran similaire à celui-ci



Note : The keyboard shortcut `Ctrl+~` can be used to open a terminal in Visual Studio Code for Windows.

Vous aimerez à ce point-ci ouvrir une fenêtre PowerShell et accéder à votre répertoire de projets. Un excellent tutoriel sur PowerShell peut être trouvé ici. Veuillez utiliser votre moteur de recherche afin de trouver la façon d'ouvrir un terminal sur les systèmes d'exploitation alternatifs.



If a directory is empty, a file needs to be created in order for git to have something to track.

In the below Empty Directory example, we created a file called `README.md` with the contents of `# Example Repo`. For FRC® Robot projects, the below Existing Project commands should be run in the root of a project *created by the VS Code WPILib Project Creator*. More details on the various commands can be found in the subsequent sections.

Note : Remplacez le chemin d'accès `"C:\Users\ExampleUser9007\Documents\Example Folder"` par celui dans lequel vous souhaitez créer le repo, et remplacez l'URL distante `https://github.com/ExampleUser9007/ExampleRepo.git` par l'URL du repo que vous avez créé dans les étapes précédentes.

Répertoire vide

```
> cd "C:\Users\ExampleUser9007\Documents\Example Folder"
> git init
Initialized empty Git repository in C:/Users/ExampleUser9007/Documents/Example Folder/
↪.git/
> echo "# ExampleRepo" >> README.md
> git add README.md
> git commit -m "First commit"
[main (root-commit) fafafa] First commit
 1 file changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README.md
> git remote add origin https://github.com/ExampleUser9007/ExampleRepo.git
> git push -u origin main
```

Projet existant

```
> cd "C:\Users\ExampleUser9007\Documents\Example Folder"
> git init
Initialized empty Git repository in C:/Users/ExampleUser9007/Documents/Example Folder/
↪.git/
> git add .
> git commit -m "First commit"
[main (root-commit) fafafa] First commit
 1 file changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README.md
> git remote add origin https://github.com/ExampleUser9007/ExampleRepo.git
> git push -u origin main
```

17.1.4 Les Commits, ou Validés

Les référentiels sont principalement composés de Commits. Ceux-ci sont des états enregistrés ou des *versions* de code.

Dans l'exemple précédent, nous avons créé un fichier appelé README.md. Ouvrez ce fichier dans votre éditeur de texte préféré et modifiez-en quelques lignes. Après avoir joué avec le fichier pendant un moment, il suffit d'enregistrer et de fermer. Accédez à PowerShell et tapez les commandes suivantes.

```
> git add README.md
> git commit -m "Adds a description to the repository"
[main bcbcbc] Adds a description to the repository
 1 file changed, 2 insertions(+), 0 deletions(-)
> git push
```

Note : Writing good commit messages is a key part of a maintainable project. A guide on writing commit messages can be found here.

La commande Git Pull

Note : `git fetch` peut être utilisé lorsque l'utilisateur ne souhaite pas fusionner automatiquement dans la branche de travail actuelle

---

Cette commande récupère l'historique ou valide à partir du référentiel distant. Lorsque la télécommande contient du travail que vous n'avez pas, elle tentera de fusionner automatiquement. Voir *Le fusionnement (Merge)*.

Run : `git pull`

La commande Git Add

This command « stages » the specified file(s) so that they will be included in the next commit.

For a single file, run `git add FILENAME.txt` where FILENAME.txt is the name and extension of the file to add. To add every file/folder that isn't excluded via *gitignore*, run `git add .`. When run in the root of the repository this command will stage every untracked, unexcluded file.

La commande Git Commit

This command creates the commit and stores it locally. This saves the state and adds it to the repository's history. The commit will consist of whatever changes (« diffs ») were made to the staged files since the last commit. It is required to specify a « commit message » explaining why you changed this set of files or what the change accomplishes.

Exécuter : `git commit -m "type message here"`

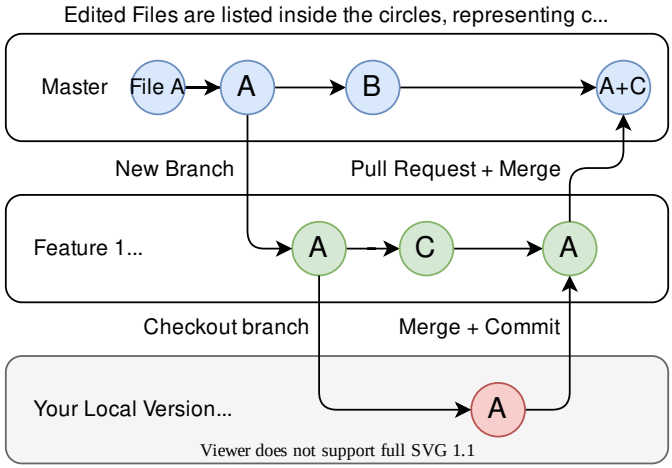
La commande Git Push

Téléchargez (Push) vos modifications locales sur le « Cloud » (à distance)

Exécuter : `git push`

17.1.5 Les Branches

Branches in Git are similar to parallel worlds. They start off the same, and then they can « branch » out into different varying paths. Consider the Git control flow to look similar to this.



Dans l'exemple ci-dessus, la branche principale a été fusionnée (ou dupliquée) avec la branche dévolue à la Fonctionnalité 1 et quelqu'un a dupliqué la branche pour en avoir une copie locale. Ensuite, une personne fait un commit (ou téléchargé) de ses modifications, les fusionnant avec la branche de Fonctionnalité 1 de la branche. Vous « fusionnez » les changements d'une branche à l'autre.

Création d'une branche

Exécutez : `git branch branch-name` où branch-name est le nom de la branche à créer. Le nouvel historique de branche sera créé à partir de la branche active actuelle.

Entrer dans une branche

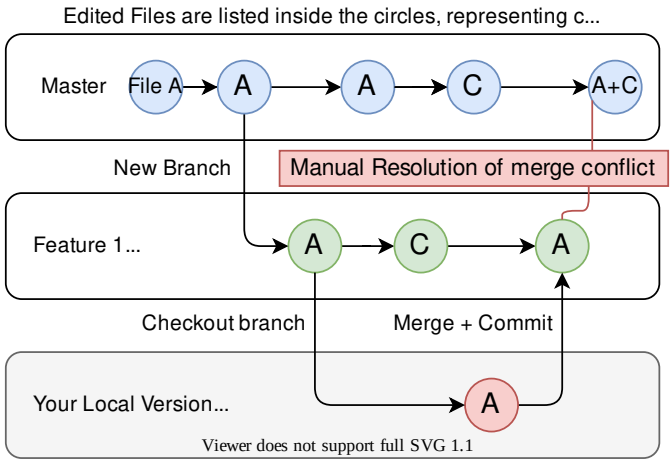
Une fois qu'une branche est créée, vous devez ensuite entrer dans la branche.

Exécutez : `git checkout branch-name` où branch-name est la branche qui a été créée précédemment.

17.1.6 Le fusionnement (Merge)

In scenarios where you want to copy one branches history into another, you can merge them. A merge is done by calling `git merge branch-name` with branch-name being the name of the branch to merge from. It is automatically merged into the current active branch.

It's common for a remote repository to contain work (history) that you do not have. Whenever you run `git pull`, it will attempt to automatically merge those commits into your local copy. That merge may look like the below.



However, in the above example, what if File A was modified by both branch Feature1 and Feature2 ? This is called a **merge conflict**. A merge conflict can be resolved by editing the conflicting file. In the example, we would need to edit File A to keep the history or changes that we want. After that has been done, simply re-add, re-commit, and then push your changes.

17.1.7 Les réinitialisations (Resets)

Parfois, l'historique doit être réinitialisé ou un Commit doit être annulé. Cela peut se faire de plusieurs manières.

Revenir en arrière sur un Commit

Note : Vous ne pouvez pas annuler une fusion, car git ne sait pas quelle branche ou origine il doit choisir.

Pour revenir en arrière sur l'historique menant à un Commit, exécutez `git revert commit-id`. Les ID de validation peuvent être affichés à l'aide de la commande `git log`.

La commande de réinitialisation de la tête

Avertissement : Réinitialiser la tête est une commande dangereuse. Elle efface définitivement toute le travail que vous avez accompli, et qui n'a pas été validé (Commit). Vous êtes prévenus !

Exécutez : `git reset --hard commit-id`.

17.1.8 Fourches ou Forks

Les fourches peuvent être traitées de la même manière que les branches. Vous pouvez fusionner le référentiel d'origine, (celui en amont, ou upstream) dans le référentiel fourché (celui d'origine).

Clonage d'un Dépôt existant

Dans la situation où un dépôt a déjà été créé et stocké dans un serveur distant, vous pouvez le cloner à l'aide

```
git clone https://github.com/myrepo.git
```

where `myrepo.git` is replaced with your git repo. If you follow this, you can skip to *commits*.

Mettre à jour une fourche

1. Ajoutez l'amont : `git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git`
2. Confirmez qu'il a été ajouté via : `git remote -v`
3. Faire un « Fetch » sur les changements en amont : `git fetch upstream`
4. Fusionnez les modifications dans la tête : `git merge upstream/upstream-branch-name`

17.1.9 Gitignore

Important : Il est extrêmement important que les équipes **ne modifient pas** le fichier `.gitignore` qui est intégré dans leur projet de robot. Cela peut conduire à un déploiement hors connexion qui ne fonctionne pas.

Un fichier `.gitignore` est couramment utilisé comme liste de fichiers à ne pas valider automatiquement avec `git add`. Tous les fichiers ou répertoires énumérés dans ce fichier ne seront pas validés. Ils ne s'afficheront pas non plus avec la commande git status.

Additional Information can be found here.

Masquer un dossier

Ajoutez simplement une nouvelle ligne contenant le dossier à cacher, avec une barre oblique à la fin

EX : `répertoire à exclure/`

Masquer un fichier

Ajoutez une nouvelle ligne avec le nom du fichier à masquer, y compris tout répertoire précédant de la racine du référentiel.

EX : `répertoire/fichier-à-cacher.txt`

EX : `file-to-hide2.txt`

17.1.10 Information supplémentaire

Un didacticiel beaucoup plus approfondi peut être trouvé sur le site officiel de git.

Un guide pour corriger les erreurs courantes se trouve dans le dépôt git flight rules

17.2 La librairie d'unités C++

WPILib is coupled with a Units library for C++ teams. This library leverages the C++ type system to enforce proper dimensionality for method parameters, automatically perform unit conversions, and even allow users to define arbitrary defined unit types. Since the C++ type system is enforced at compile-time, the library has essentially no runtime cost.

17.2.1 Utilisation de la librairie d'unités

La librairie d'unités est une bibliothèque d'en-tête uniquement. Vous devez inclure l'en-tête approprié dans vos fichiers source pour les unités que vous souhaitez utiliser. Voici une liste des unités disponibles.

```
#include <units/acceleration.h>
#include <units/angle.h>
#include <units/angular_acceleration.h>
#include <units/angular_velocity.h>
#include <units/area.h>
#include <units/capacitance.h>
#include <units/charge.h>
#include <units/concentration.h>
#include <units/conductance.h>
#include <units/current.h>
#include <units/curvature.h>
#include <units/data.h>
#include <units/data_transfer_rate.h>
#include <units/density.h>
#include <units/dimensionless.h>
#include <units/energy.h>
```

(suite sur la page suivante)

(suite de la page précédente)

```
#include <units/force.h>
#include <units/frequency.h>
#include <units/illuminance.h>
#include <units/impedance.h>
#include <units/inductance.h>
#include <units/length.h>
#include <units/luminous_flux.h>
#include <units/luminous_intensity.h>
#include <units/magnetic_field_strength.h>
#include <units/magnetic_flux.h>
#include <units/mass.h>
#include <units/moment_of_inertia.h>
#include <units/power.h>
#include <units/pressure.h>
#include <units/radiation.h>
#include <units/solid_angle.h>
#include <units/substance.h>
#include <units/temperature.h>
#include <units/time.h>
#include <units/torque.h>
#include <units/velocity.h>
#include <units/voltage.h>
#include <units/volume.h>
```

L'en-tête `units/math.h` fournit des fonctions adaptées aux unités telles que `units::math::abs()`.

Types d'unités et types de conteneurs

La librairie d'unités C++ est basée sur deux sortes de définitions de types : les types d'unités et les types de conteneurs.

Types d'unités

Les types d'unités correspondent au concept abstrait d'une unité, sans aucune valeur stockée réelle. Les types d'unités sont le « bloc de construction » fondamental de la librairie d'unités - tous les types d'unités sont définis de manière constructive (en utilisant le modèle `compound_unit`) à partir d'un petit nombre de types d'unités « de base » (tels que `mètres`, `secondes`, etc.).

While unit types cannot contain numerical values, their use in building other unit types means that when a type or method uses a template parameter to specify its dimensionality, that parameter will be a unit type.

Types de conteneurs

Les types de conteneurs correspondent à une quantité réelle dimensionnée en fonction d'une unité, c'est-à-dire qu'ils détiennent réellement la valeur numérique. Les types de conteneurs sont construits à partir de types d'unités avec le modèle `unit_t`. La plupart des types d'unités ont un type de conteneur correspondant qui porte le même nom suffixé par `_t` - par exemple, le type d'unité `units :: meter` correspond au type de conteneur `units :: meter_t`.

Chaque fois qu'une quantité spécifique d'une unité est utilisée (en tant que variable ou paramètre de méthode), ce sera une instance du type de conteneur. Par défaut, les types de conteneurs stockent la valeur réelle sous forme de `double` - les utilisateurs plus avancés peuvent changer cela en appelant le modèle `unit_t` manuellement.

Vous trouverez une liste complète des types d'unités et de conteneurs dans la documentation.

Création d'instances d'unités

Pour créer une instance d'une unité spécifique, nous créons une instance de son type de conteneur :

```
// The variable speed has a value of 5 meters per second.
units::meter_per_second_t speed{5.0};
```

Alternativement, la librairie d'unités a des types de littéraux définis pour certains des types de conteneurs les plus courants. Ceux-ci peuvent être utilisés en conjonction avec l'inférence de type via `auto` pour définir une unité plus succinctement :

```
// The variable speed has a value of 5 meters per second.
auto speed = 5_mps;
```

Les unités peuvent également être initialisées à l'aide d'une valeur d'un autre type de conteneur, en autant que les types peuvent être convertis entre eux. Par exemple, une valeur `mètre_t` peut être créée à partir d'une valeur `pied_t`.

```
auto feet = 6_ft;
units::meter_t meters{feet};
```

En fait, tous les types de conteneurs représentant des types d'unités convertibles sont *implicitement convertibles*. Ainsi, ce qui suit est parfaitement légal :

```
units::meter_t distance = 6_ft;
```

En bref, nous pouvons utiliser *n'importe quelle* unité de longueur à la place de *toute autre* unité de longueur, n'importe où dans notre code ; la librairie d'unités effectuera automatiquement la conversion correcte pour nous.

Exécution de l'arithmétique avec des unités

Les types de conteneurs prennent en charge toutes les opérations arithmétiques ordinaires de leur type de données sous-jacent, avec la condition supplémentaire que l'opération doit être *dimensionnellement* valide. Ainsi, l'addition doit toujours être effectuée sur deux types de conteneurs compatibles :

```
// Add two meter_t values together
auto sum = 5_m + 7_m; // sum is 12_m

// Adds meters to feet; both are length, so this is fine
auto sum = 5_m + 7_ft;

// Tries to add a meter_t to a second_t, will throw a compile-time error
auto sum = 5_m + 7_s;
```

La multiplication peut être effectuée sur n'importe quelle paire de types de conteneurs et donne le type de conteneur d'une unité composée :

Note : Lorsqu'un calcul donne un type d'unité composé, ce type ne sera vérifié pour la validité au point d'opération que si le type de résultat est spécifié explicitement. Si `auto` est utilisé, cette vérification n'aura pas lieu. Par exemple, lorsque nous divisons la distance par le temps, nous pouvons vouloir nous assurer que le résultat est bien une vitesse (c'est-à-dire `units :: meter_per_second_t`). Si le type de retour est déclaré `auto`, cette vérification ne sera pas effectuée.

```
// Multiply two meter_t values, result is square_meter_t
auto product = 5_m * 7_m; // product is 35_sq_m
```

```
// Divide a meter_t value by a second_t, result is a meter_per_second_t
units::meter_per_second_t speed = 6_m / 0.5_s; // speed is 12_mps
```

Les fonctions <cmath>

Certaines fonctions `std` (telles que `clamp`) sont conçues pour accepter tout type sur lequel les opérations arithmétiques peuvent être effectuées. Les quantités stockées en tant que types de conteneurs fonctionneront sans problème avec ces fonctions.

Cependant, d'autres fonctions `std` ne fonctionnent que sur les types numériques ordinaires (par exemple `double`). L'espace de noms `units :: math` de la librairie d'unités contient des « wrappers » pour plusieurs de ces fonctions qui acceptent les unités. Des exemples de telles fonctions incluent `sqrt`, `pow`, etc.

```
auto area = 36_sq_m;
units::meter_t sideLength = units::math::sqrt(area);
```

Retrait du « Unit Wrapper »

Pour convertir un type de conteneur en sa valeur sous-jacente, utilisez la méthode `value()`. Cela sert de trappe d'évacuation du système de type d'unités, qui ne doit être utilisé que lorsque c'est nécessaire.

```
units::meter_t distance = 6.5_m;
double distanceMeters = distance.value();
```

17.2.2 Exemple de la librairie d'unités dans le code WPILib

Plusieurs arguments pour des méthodes dans les nouvelles fonctionnalités de WPILib (ex. *kinematics*) utilisent la librairie d'unités. Voici un exemple d' *échantillonnage d'une trajectoire*.

```
// Sample the trajectory at 1.2 seconds. This represents where the robot
// should be after 1.2 seconds of traversal.
Trajectory::State point = trajectory.Sample(1.2_s);

// Since units of time are implicitly convertible, this is exactly equivalent to the␣
↪above code
Trajectory::State point = trajectory.Sample(1200_ms);
```

Certaines classes WPILib représentent des objets qui pourraient naturellement fonctionner avec plusieurs choix de types d'unités - par exemple, un profil de mouvement peut fonctionner sur une distance linéaire (par exemple mètres) ou angulaire (par exemple radians). Pour ces classes, le type d'unité est requis comme paramètre de modèle :

```
// Creates a new set of trapezoidal motion profile constraints
// Max velocity of 10 meters per second
// Max acceleration of 20 meters per second squared
frc::TrapezoidProfile<units::meters>::Constraints{10_mps, 20_mps_sq};

// Creates a new set of trapezoidal motion profile constraints
// Max velocity of 10 radians per second
// Max acceleration of 20 radians per second squared
frc::TrapezoidProfile<units::radians>::Constraints{10_rad_per_s, 20__rad_per_s / 1_s};
```

Pour une documentation plus détaillée, veuillez visiter la page officielle GitHub pour la librairie des unités.

17.3 The Java Units Library

The units library is a tool that helps programmers avoid mistakes related to units of measurement. It does this by keeping track of the units of measurement, and by ensuring that all operations are performed with the correct units. This can help to prevent errors that can lead to incorrect results, such as adding a distance in inches to a distance in meters.

An added benefit is readability and maintainability, which also reduces bugs. By making the units of measurement explicit in your code, it becomes easier to read and understand what your code is doing. This can also help to make your code more maintainable, as it is easier to identify and fix errors related to units of measurement.

The units library has a number of features :

— A set of predefined units, such as meters, degrees, and seconds.
— The ability to convert between different units.
— Support for performing arithmetic and comparisons on quantities with units.
— Support for displaying quantities with units in a human-readable format.

17.3.1 Terminology

Dimension
: Dimensions represent the nature of a physical quantity, such as length, time, or mass. They are independent of any specific unit system. For example, the dimension of meters is length, regardless of whether the length is expressed in meters, millimeters, or inches.

Unit
: Units are specific realizations of dimensions. They are the way of expressing physical quantities. Each dimension has a base unit, such as the meter for length, the second for time, the kilogram for mass. Derived units are formed by combining base units, such as meters per second for velocity.

Measure
: Measures are the specific magnitude of physical quantities, expressed in a particular unit. For example, 5 meters is a measure of distance.

These concepts are used within the Units Library. For example, the **measure** *10 seconds* has a magnitude of 10, the **dimension** is time, and the **unit** is seconds.

17.3.2 Using the Units Library

The Java units library is available in the `edu.wpi.first.units` package. The most relevant classes are :
— The various classes for predefined dimensions, such as Distance and Time
— Units, which contains a set of predefined units. Take a look a the Units javadoc to browse the available units and their types.
— Measure, which is used to tag a value with a unit.

Note : It is recommended to static import `edu.wpi.first.units.Units.*` to get full access to all the predefined units.

Java Generics

Units of measurement can be complex expressions involving various dimension, such as distance, time, and velocity. Nested generic type parameters allow for the definition of units that can represent such complex expressions. Generics are used to keep the library concise, reusable, and extensible, but it tends to be verbose due to the syntax for Java generics.

For instance, consider the type `Measure<Velocity<Distance>>`. This type represents a measurement for velocity, where the velocity itself is expressed as a unit of distance per unit of time. This nested structure allows for the representation of units like meters per second or feet per minute. Similarly, the type `Measure<Per<Voltage, Velocity<Distance>>>` represents a measurement for a ratio of voltage to velocity. This type is useful for representing quantities like volts per meter per second, the unit of measure for some *feedforward* gains.

It's important to note that not all measurements require such complex nested types. For example, the type `Measure<Distance>` is sufficient for representing simple units like meters or feet. However, for more complex units, the use of nested generic type parameters is essential.

For local variables, you may choose to use Java's *var* keyword instead of including the full type name. For example, these are equivalent :

```
Measure<Per<Voltage, Velocity<Distance>>> v = VoltsPerMeterPerSecond.of(8);
var v = VoltsPerMeterPerSecond.of(8);
```

Creating Measures

The `Measure` class is a generic type that represents a magnitude (physical quantity) with its corresponding unit. It provides a consistent and type-safe way to handle different dimensions of measurements, such as distance, angle, and velocity, but abstracts away the particular unit (e.g. meter vs. inch). To create a `Measure` object, you call the `Unit.of` method on the appropriate unit object. For example, to create a `Measure<Distance>` object representing a distance of 6 inches, you would write :

```
Measure<Distance> wheelDiameter = Inches.of(6);
```

Other measures can also be created using their `Unit.of` method :

```
Measure<Mass> kArmMass = Kilograms.of(1.423);
Measure<Distance> kArmLength = Inches.of(32.25);
Measure<Angle> kMinArmAngle = Degrees.of(5);
Measure<Angle> kArmMaxTravel = Rotations.of(0.45);
Measure<Velocity<Distance>> kMaxSpeed = MetersPerSecond.of(2.5);
```

Performing Calculations

The `Measure` class also supports arithmetic operations, such as addition, subtraction, multiplication, and division. These are done by calling methods on the objects. These operations always ensure that the units are compatible before performing the calculation, and they return a new `Measure` object. For example, you can add two `Measure<Distance>` objects together, even if they have different units :

```
Measure<Distance> distance1 = Inches.of(10);
Measure<Distance> distance2 = Meters.of(0.254);

Measure<Distance> totalDistance = distance1.plus(distance2);
```

In this code, the units library will automatically convert the measures to the same unit before adding the two distances. The resulting `totalDistance` object will be a new `Measure<Distance>` object that has a value of 0.508 meters, or 20 inches.

This example combines the wheel diameter and gear ratio to calcualate the distance per rotation of the wheel :

```
Measure<Distance> wheelDiameter = Inches.of(3);
double gearRatio = 10.48;
Measure<Distance> distancePerRotation = wheelDiameter.times(Math.PI).
↪divide(gearRatio);
```

Avertissement : By default, arithmetic operations create new `Measure` instances for their results. See *Java Garbage Collection* for discussion on creating a large number of short-lived objects. See also, the *Mutability and Object Creation* section below for a possible workaround.

Converting Units

Unit conversions can be done by calling `Measure.in(Unit)`. The Java type system will prevent units from being converted between incompatible types, such as distances to angles. The returned values will be bare `double` values without unit information - it is up to you, the programmer, to interpret them correctly ! It is strongly recommended to only use unit conversions when interacting with APIs that do not support the units library.

```
Measure<Velocity<Distance>> kMaxVelocity = FeetPerSecond.of(12.5);
Measure<Velocity<Velocity<Distance>>> kMaxAcceleration = FeetPerSecond.per(Second).
↪of(22.9);

kMaxVelocity.in(MetersPerSecond); // => OK! Returns 3.81
kMaxVelocity.in(RadiansPerSecond); // => Compile error! Velocity<Angle> cannot be␣
↪converted to Unit<Velocity<Distance>>

// The WPILib math libraries use SI metric units, so we have to convert to meters:
TrapezoidProfile.Constraints kDriveConstraints = new TrapezoidProfile.Constraints(
  maxVelocity.in(MetersPerSecond),
  maxAcceleration.in(MetersPerSecondPerSecond)
);
```

Usage Example

Pulling all of the concepts together, we can create an example that calculates the end effector position of an arm mechanism :

```
Measure<Distance> armLength = Feet.of(3).plus(Inches.of(4.25));
Measure<Distance> endEffectorX = armLength.times(Math.cos(getArmAngle().in(Radians)));
Measure<Distance> endEffectorY = armLength.times(Math.sin(getArmAngle().in(Radians)));
```

Human-readable Formatting

The `Measure` class has methods that can be used to get a human-readable representation of the measure. This feature is useful to display a measure on a dashboard or in logs.

- `toString()` and `toShortString()` return a string representation of the measure in a shorthand form. The symbol of the backing unit is used, rather than the full name, and the magnitude is represented in scientific notation. For example, 1.234e+04 V/m
- `toLongString()` returns a string representation of the measure in a longhand form. The name of the backing unit is used, rather than its symbol, and the magnitude is represented in a full string, not scientific notation. For example, 1234 Volt per Meter

17.3.3 Mutability and Object Creation

To reduce the number of object instances you create, and reduce memory usage, a special `MutableMeasure` class is available. You may want to consider using mutable objects if you are using the units library repeatedly, such as in the robot's periodic loop. See *Java Garbage Collection* for more discussion on creating a large number of short-lived objects.

`MutableMeasure` allows the internal state of the object to be updated, such as with the results of arithmetic operations, to avoid allocating new objects. Special care needs to be taken when mutating a measure because it will change the value every place that instance is referenced. If the object will be exposed as part of a public method, have that method return a regular `Measure` in its signature to prevent the caller from modifying your internal state.

Extra methods are available on `MutableMeasure` for updating the internal value. Note that these methods all begin with the `mut_` prefix - this is to make it obvious that these methods will be mutating the object and are potentially unsafe ! For the full list of methods and API documentation, see the MutableMeasure API documentation

`mut_plus(double, Unit)`	Increments the internal value by an amount in another unit. The internal unit will stay the same
`mut_plus(Measure)`	Increments the internal value by another measurement. The internal unit will stay the same
`mut_minus(double, Unit)`	Decrements the internal value by an amount in another unit. The internal unit will stay the same
`mut_minus(Measure)`	Decrements the internal value by another measurement. The internal unit will stay the same
`mut_times(double)`	Multiplies the internal value by a scalar
`mut_divide(double)`	Divides the internal value by a scalar
`mut_replace(double Unit)`	Overrides the internal state and sets it to equal the given value and unit
`mut_replace(Measur`	Overrides the internal state to make it identical to the given measurement
`mut_setMagnitude(d`	Overrides the internal value, keeping the internal unit. Be careful when using this !

```
MutableMeasure<Distance> measure = MutableMeasure.zero(Feet);
measure.mut_plus(10, Inches);    // 0.8333 feet
measure.mut_plus(Inches.of(10)); // 1.6667 feet
measure.mut_minus(5, Inches);    // 1.25 feet
measure.mut_minus(Inches.of(5)); // 0.8333 feet
measure.mut_times(6);            // 0.8333 * 6 = 5 feet
measure.mut_divide(5);           // 5 / 5 = 1 foot
measure.mut_replace(6.2, Meters) // 6.2 meters - note the unit changed!
measure.mut_replace(Millimeters.of(14.2)) // 14.2mm - the unit changed again!
measure.mut_setMagnitude(72)     // 72mm
```

Revisiting the arm example from above, we can use `mut_replace` - and, optionally, `mut_times` - to calculate the end effector position

```
import edu.wpi.first.units.Measure;
import edu.wpi.first.units.MutableMeasure;
import static edu.wpi.first.units.Units.*;

public class Arm {
```

(suite sur la page suivante)

(suite de la page précédente)

```
  // Note the two ephemeral object allocations for the Feet.of and Inches.of calls.
  // Because this is a constant value computed just once, they will easily be garbage collected without
  // any problems with memory use or loop timing jitter.
  private static final Measure<Distance> kArmLength = Feet.of(3).plus(Inches.of(4.25));

  // Angle and X/Y locations will likely be called in the main robot loop, let's store them in a MutableMeasure
  // to avoid allocating lots of short-lived objects
  private final MutableMeasure<Angle> m_angle = MutableMeasure.zero(Degrees);
  private final MutableMeasure<Distance> m_endEffectorX = MutableMeasure.zero(Feet);
  private final MutableMeasure<Distance> m_endEffectorY = MutableMeasure.zero(Feet);

  private final Encoder m_encoder = new Encoder(...);

  public Measure<Distance> getEndEffectorX() {
    m_endEffectorX.mut_replace(
      Math.cos(getAngle().in(Radians)) * kArmLength.in(Feet), // the new magnitude to store
      Feet // the units of the new magnitude
    );
    return m_endEffectorX;
  }

  public Measure<Distance> getEndEffectorY() {
    // An alternative approach so we don't have to unpack and repack the units
    m_endEffectorY.mut_replace(kArmLength);
    m_endEffectorY.mut_times(Math.sin(getAngle().in(Radians)));
    return m_endEffectorY;
  }

  public Measure<Angle> getAngle() {
    double rawAngle = m_encoder.getPosition();
    m_angle.mut_replace(rawAngle, Degrees); // NOTE: the encoder must be configured with distancePerPulse in terms of degrees!
    return m_angle;
  }
}
```

Avertissement : `MutableMeasure` objects can - by definition - change their values at any time ! It is unsafe to keep a stateful reference to them - prefer to extract a value using the `Measure.in` method, or create a copy with `Measure.copy` that can be safely stored. For the same reason, library authors must also be careful about methods accepting `Measure`.

Can you spot the bug in this code ?

```
private Measure<Distance> m_lastDistance;

public Measure<Distance> calculateDelta(Measure<Distance> currentDistance) {
  if (m_lastDistance == null) {
    m_lastDistance = currentDistance;
    return currentDistance;
  } else {
```

(suite sur la page suivante)

(suite de la page précédente)

```
    Measure<Distance> delta = currentDistance.minus(m_lastDistance);
    m_lastDistance = currentDistance;
    return delta;
  }
}
```

If we run the `calculateDelta` method a few times, we can see a pattern :

```
MutableMeasure<Distance> distance = MutableMeasure.zero(Inches);
distance.mut_plus(10, Inches);
calculateDelta(distance); // expect 10 inches and get 10 - good!

distance.mut_plus(2, Inches);
calculateDelta(distance); // expect 2 inches, but get 0 instead!

distance.mut_plus(8, Inches);
calculateDelta(distance); // expect 8 inches, but get 0 instead!
```

This is because the `m_lastDistance` field is a reference to the *same* `MutableMeasure` object as the input ! Effectively, the delta is calculated as (currentDistance - currentDistance) on every call after the first, which naturally always returns zero. One solution would be to track `m_lastDistance` as a *copy* of the input measure to take a snapshot ; however, this approach does incur one extra object allocation for the copy. If you need to be careful about object allocations, `m_lastDistance` could also be stored as a `MutableMeasure`.

Immutable Copies

```
private Measure<Distance> m_lastDistance;

public Measure<Distance> calculateDelta(Measure<Distance> currentDistance) {
  if (m_lastDistance == null) {
    m_lastDistance = currentDistance.copy();
    return currentDistance;
  } else {
    var delta = currentDistance.minus(m_lastDistance);
    m_lastDistance = currentDistance.copy();
    return delta;
  }
}
```

Zero-allocation Mutables

```
private final MutableMeasure<Distance> m_lastDistance = MutableMeasure.zero(Meters);
private final MutableMeasure<Distance> m_delta = MutableMeasure.zero(Meters);

public Measure<Distance> calculateDelta(Measure<Distance> currentDistance) {
  // m_delta = currentDistance - m_lastDistance
  m_delta.mut_replace(currentDistance);
  m_delta.mut_minus(m_lastDistance);
  m_lastDistance.mut_replace(currentDistance);
  return m_delta;
}
```

17.3.4 Defining New Units

There are four ways to define a new unit that isn't already present in the library :
- Using the `Unit.per` or `Unit.mult` methods to create a composite of two other units ;
- Using the `Milli`, `Micro`, and `Kilo` helper methods ;
- Using the `derive` method and customizing how the new unit relates to the base unit ; and
- Subclassing `Unit` to define a new dimension.

New units can be defined as combinations of existing units using the `Unit.mult` and `Unit.per` methods.

```
Per<Voltage, Distance> VoltsPerInch = Volts.per(Inch);
Velocity<Mass> KgPerSecond = Kilograms.per(Second);
Mult<Mass, Velocity<Velocity<Distance>> Newtons = Kilograms.
↪mult(MetersPerSecondSquared);
```

Using `mult` and `per` will store the resulting unit. Every call will return the same object to avoid unnecessary allocations and garbage collector pressure.

```
@Override
public void robotPeriodic() {
  // Feet.per(Millisecond) creates a new unit on the first loop,
  // which will be reused on every successive loop
  SmartDashboard.putNumber("Speed", m_drivebase.getSpeed().in(Feet.per(Millisecond)));
}
```

Note : Calling `Unit.per(Time)` will return a `Velocity` unit, which is different from and incompatible with a `Per` unit !

New dimensions can also be created by subclassing `Unit` and implementing the two constructors. Note that `Unit` is also a parameterized generic type, where the generic type argument is self-referential ; `Distance` is a `Unit<Distance>`. This is what allows us to have stronger guarantees in the type system to prevent conversions between unrelated dimensions.

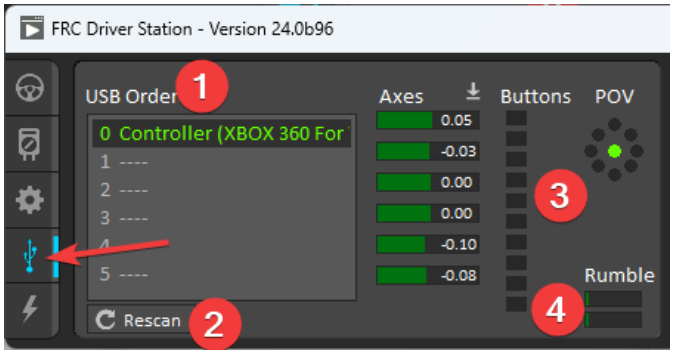
```
public class ElectricCharge extends Unit<ElectricCharge> {
  public ElectricCharge(double baseUnitEquivalent, String name, String symbol) {
    super(ElectricCharge.class, baseUnitEquivalent, name, symbol);
  }

  // required for derivation with Milli, Kilo, etc.
  public ElectricCharge(UnaryFunction toBaseConverter, UnaryFunction␣
↪fromBaseConverter, String name, String symbol) {
     super(ElectricCharge.class, toBaseConverter, fromBaseConverter, name, symbol);
  }
}

public static final ElectricCharge Coulomb = new ElectricCharge(1, "Coulomb", "C");
public static final ElectricCharge ElectronCharge = new ElectricCharge(1.60217646e-19,
↪ "Electron Charge", "e");
public static final ElectricCharge AmpHour = new ElectricCharge(3600, "Amp Hour", "Ah
↪");
public static final ElectricCharge MilliampHour = Milli(AmpHour);
```

17.4 Manettes (Joysticks)

Un joystick peut être utilisé avec le programme Driver Station pour contrôler le robot. Presque tous les « contrôleurs » reconnus par Windows peuvent être utilisés comme joystick. Les manettes sont accessibles en utilisant la classe `GenericHID`. Cette classe a trois sous-classes pertinentes pour les joysticks préconfigurés. Vous pouvez également implémenter le vôtre pour d'autres contrôleurs en étendant `GenericHID`. Le premier est `Joystick` qui est utile pour les joysticks de pilotage aérien standard. Le second est `XboxController` qui fonctionne pour la Xbox 360, Xbox One ou Logitech F310 (en mode XInput). Enfin, la classe `PS4Controller` est idéale pour utiliser ce contrôleur. Chaque axe du contrôleur va de -1 à 1.

The command based way to use the these classes is detailed in the section : *Liaison de commandes à des déclencheurs*.

17.4.1 Joysticks du poste de conduite



L'onglet *USB Devices* de Driver Station est utilisé pour installer et configurer le joystick à utiliser avec le robot. En appuyant sur un bouton d'un joystick, son entrée dans le tableau s'allumera en vert. La sélection du joystick affichera les valeurs des axes, des boutons et le PDV qui peuvent être utilisés pour déterminer le mappage entre les caractéristiques physiques du joystick et les numéros d'axe ou de bouton.

L'onglet Périphériques USB attribue également un index de joystick à chaque joystick. Pour réorganiser les joysticks, cliquez simplement et faites glisser. Le logiciel Driver Station essaiera de préserver l'ordre des périphériques entre les exécutions. C'est une bonne idée de noter dans quel ordre vos appareils doivent être et de vérifier chaque fois que vous démarrez le logiciel Driver Station qu'ils sont corrects.

Lorsque la Driver Station est en mode désactivé, elle recherche régulièrement les changements d'état sur les dispositifs joysticks. Les appareils débranchés sont supprimés de la liste et de nouveaux appareils sont ouverts et ajoutés. Lorsqu'il n'est pas connecté au FMS, le fait

de débrancher un joystick forcera la Driver Station en mode désactivé. Pour recommencer à utiliser le joystick : branchez le joystick, vérifiez qu'il s'affiche au bon endroit, puis réactivez le robot. Tant que la Driver Station est en mode activé, elle ne recherchera pas de nouveaux périphériques. Il s'agit d'une opération qui prend du temps et la mise à jour rapide des signaux des appareils connectés est prioritaire.

Note : Pour certains joysticks, la routine de démarrage lira quelle que soit la position des joysticks comme position centrale, par conséquent, lorsque l'ordinateur est allumé (ou lorsque le joystick est branché), les joysticks doivent être à leur position centrale.

Lorsque le robot est connecté au Field Management System en compétition, le mode Driver Station est dicté par le *FMS*. Cela signifie que vous ne pouvez pas désactiver votre robot et que le DS ne peut pas se désactiver lui-même afin de détecter les changements de joystick. Un rafraîchissement manuel complet des joysticks peut être initié en appuyant sur la touche F1 du clavier. Notez que cela fermera et rouvrira tous les manettes, donc celles-ci doivent être dans leur position centrale comme indiqué ci-dessus.

17.4.2 Classe `Joystick`

JAVA

```
Joystick exampleJoystick = new Joystick(0); // 0 is the USB Port to be used as␣
↪indicated on the Driver Station
```

C++

```
Joystick exampleJoystick{0}; // 0 is the USB Port to be used as indicated on the␣
↪Driver Station
```

PYTHON

```
exampleJoystick = wpilib.Joystick(0) # 0 is the USB Port to be used as indicated on␣
↪the Driver Station
```

La classe `Joystick` est conçue pour faciliter considérablement l'utilisation d'un joystick de pilotage aérien pour faire fonctionner le robot. Selon le joystick, l'utilisateur peut avoir besoin de définir les canaux X, Y, Z et Throttle spécifiques que votre joystick de pilotage aérien utilise. Cette classe propose des méthodes spéciales pour accéder à l'angle et à la magnitude du joystick.

Important : Due to differences in coordinate systems, teams usually negate the values when reading joystick axes. See the *Joystick and controller coordinate system* section for more detail.

17.4.3 Classe `XboxController`

JAVA

```
XboxController exampleXbox = new XboxController(0); // 0 is the USB Port to be used␣
↪as indicated on the Driver Station
```

C++

```
XboxController exampleXbox{0}; // 0 is the USB Port to be used as indicated on the␣
↪Driver Station
```

PYTHON

```
exampleXbox = wpilib.XboxController(0) # 0 is the USB Port to be used as indicated on␣
↪the Driver Station
```

The `XboxController` class provides named methods (e.g. `getXButton`, `getXButtonPressed`, `getXButtonReleased`) for each of the buttons, and the indices can be accessed with `XboxController.Button.kX.value`. The rumble feature of the controller can be controlled by using `XboxController.setRumble(GenericHID.RumbleType.kRightRumble, value)`. Many users do a split stick arcade drive that uses the left stick for just forwards / backwards and the right stick for left / right turning.

Important : Due to differences in coordinate systems, teams usually negate the values when reading joystick axes. See the *Joystick and controller coordinate system* section for more detail.

17.4.4 Classe `PS4Controller`

JAVA

```
PS4Controller examplePS4 = new PS4Controller(0); // 0 is the USB Port to be used as␣
↪indicated on the Driver Station
```

C++

```
PS4Controller examplePS4{0}; // 0 is the USB Port to be used as indicated on the␣
↪Driver Station
```

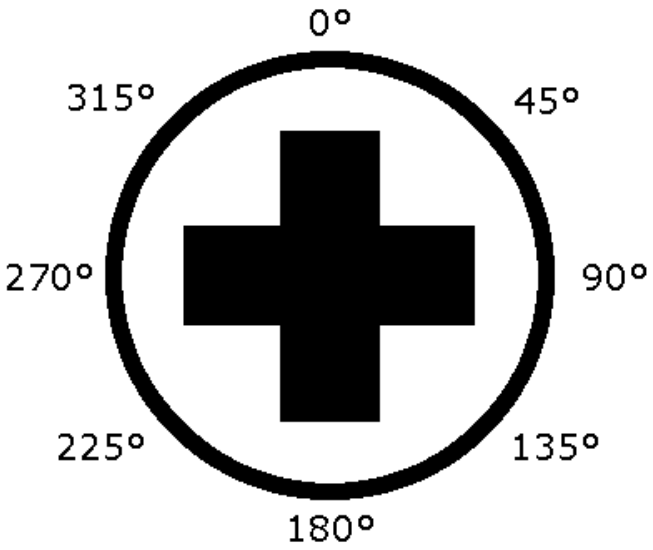
PYTHON

```
examplePS4 = wpilib.PS4Controller(0) # 0 is the USB Port to be used as indicated on␣
↪the Driver Station
```

The `PS4Controller` class provides named methods (e.g. `getSquareButton`, `getSquareButtonPressed`, `getSquareButtonReleased`) for each of the buttons, and the indices can be accessed with `PS4Controller.Button.kSquare.value`. The rumble feature of the controller can be controlled by using `PS4Controller.setRumble(GenericHID.RumbleType.kRightRumble, value)`.

Important : Due to differences in coordinate systems, teams usually negate the values when reading joystick axes. See the *Joystick and controller coordinate system* section for more detail.

17.4.5 POV



On joysticks, the POV is a directional hat that can select one of 8 different angles or read -1 for unpressed. The XboxController/PS4Controller D-pad works the same as a POV. Be careful

when using a POV with exact angle requirements as it is hard for the user to ensure they select exactly the angle desired.

17.4.6 Utilisation de GenericHID

An axis can be used with `.getRawAxis(int index)` (if not using any of the classes above) that returns the current value. Zero and one in this example are each the index of an axis as found in the Driver Station mentioned above.

JAVA

```
private final PWMSparkMax m_leftMotor = new PWMSparkMax(Constants.kLeftMotorPort);
private final PWMSparkMax m_rightMotor = new PWMSparkMax(Constants.kRightMotorPort);
private final DifferentialDrive m_robotDrive = new DifferentialDrive(m_leftMotor::set,
↪ m_rightMotor::set);
private final GenericHID m_stick = new GenericHID(Constants.kJoystickPort);

m_robotDrive.arcadeDrive(-m_stick.getRawAxis(0), m_stick.getRawAxis(1));
```

C++

```
frc::PWMVictorSPX m_leftMotor{Constants::kLeftMotorPort};
frc::PWMVictorSPX m_rightMotor{Constants::kRightMotorPort};
frc::DifferentialDrive m_robotDrive{[&](double output) { m_leftMotor.Set(output); },
                                    [&](double output) { m_rightMotor.Set(output); }};
frc::GenericHID m_stick{Constants::kJoystickPort};

m_robotDrive.ArcadeDrive(-m_stick.GetRawAxis(0), m_stick.GetRawAxis(1));
```

PYTHON

```
leftMotor = wpilib.PWMVictorSPX(LEFT_MOTOR_PORT)
rightMotor = wpilib.PWMVictorSPX(RIGHT_MOTOR_PORT)
self.robotDrive = wpilib.drive.DifferentialDrive(leftMotor, rightMotor)
self.stick = wpilib.GenericHID(JOYSTICK_PORT)

self.robotDrive.arcadeDrive(-self.stick.getRawAxis(0), self.stick.getRawAxis(1))
```

17.4.7 Utilisation des boutons

Note : Usage such as the following is for code not using the command-based framework. For button usage in the command-based framework, see *Liaison de commandes à des déclencheurs*.

Contrairement à un axe, vous souhaiterez généralement utiliser les méthodes `pressed` et `released` pour répondre à l'entrée du bouton. Ceux-ci retourneront vrai si le bouton a été

activé depuis la dernière vérification. Ceci est utile pour entreprendre une action une seule fois lorsque l'événement se produit, mais sans avoir à le faire continuellement pendant que le bouton est maintenu enfoncé.

JAVA

```
if (joystick.getRawButtonPressed(0)) {
   turnIntakeOn(); // When pressed the intake turns on
}
if (joystick.getRawButtonReleased(0)) {
   turnIntakeOff(); // When released the intake turns off
}

OR

if (joystick.getRawButton(0)) {
   turnIntakeOn();
} else {
   turnIntakeOff();
}
```

C++

```
if (joystick.GetRawButtonPressed(0)) {
   turnIntakeOn(); // When pressed the intake turns on
}
if (joystick.GetRawButtonReleased(0)) {
   turnIntakeOff(); // When released the intake turns off
}

OR

if (joystick.GetRawButton(0)) {
   turnIntakeOn();
} else {
   turnIntakeOff();
}
```

PYTHON

```
if joystick.getRawButtonPressed(0):
   turnIntakeOn() # When pressed the intake turns on

if joystick.getRawButtonReleased(0):
   turnIntakeOff() # When released the intake turns off

# OR

if joystick.getRawButton(0):
   turnIntakeOn()
else:
   turnIntakeOff()
```

Une demande courante consiste à activer et désactiver quelque chose en appuyant sur un bouton. Les bascules doivent être utilisées avec prudence, car elles obligent l'utilisateur à suivre l'état du robot.

JAVA

```java
boolean toggle = false;

if (joystick.getRawButtonPressed(0)) {
   if (toggle) {
      // Current state is true so turn off
      retractIntake();
      toggle = false;
   } else {
      // Current state is false so turn on
      deployIntake();
      toggle = true;
   }
}
```

C++

```cpp
bool toggle{false};

if (joystick.GetRawButtonPressed(0)) {
   if (toggle) {
      // Current state is true so turn off
      retractIntake();
      toggle = false;
   } else {
      // Current state is false so turn on
      deployIntake();
      toggle = true;
   }
}
```

PYTHON

```python
toggle = False

if joystick.getRawButtonPressed(0):
   if toggle:
      # current state is True so turn off
      retractIntake()
      toggle = False
   else:
      # Current state is False so turn on
      deployIntake()
      toggle = True
```

17.5 Coordinate System

Coordinate systems are used in FRC programming in several places. A few of the common places are : robot movement, joystick input, *pose* estimation, AprilTags, and path planning.

It is important to understand the basics of the coordinate system used throughout WPILib and other common tools for programming an FRC robot, such as PathPlanner. Many teams intuitively think of a coordinate system that is different from what is used in WPILib, and this leads to problems that need to be tracked down throughout the season. It is worthwhile to take a few minutes to understand the coordinate system, and come back here as a reference when programming. It's not very difficult to get robot movement with a joystick working without getting the coordinate system right, but it will be much more difficult to build on code using a different coordinate system to add *pose estimation* with *AprilTags* and path planning for autonomous.

17.5.1 WPILib coordinate system

In most cases, WPILib uses the NWU axes convention (North-West-Up as external reference in the world frame.) In the NWU axes convention, where the positive X axis points ahead, the positive Y axis points left, and the positive Z axis points up referenced from the floor. When viewed with each positive axis pointing toward you, counter-clockwise (CCW) is a positive value and clockwise (CW) is a negative value.

The figure above shows the coordinate system in relation to an FRC robot. The figure below shows this same coordinate system when viewed from the top (with the Z axis pointing toward you.) This is how you can think of the robot's coordinates in 2D.

17.5.2 Rotation conventions

In most cases in WPILib programming, 0° is aligned with the positive X axis, and 180° is aligned with the negative X axis. CCW rotation is positive, so 90° is aligned with the positive Y axis, and -90° is aligned with the negative Y axis.

The figure above shows the unit circle with common angles labeled in degrees (°) and radians (rad). Notice that rotation to the right is negative, and the range for the whole unit circle is -180° to 180° (-Pi radians to Pi radians).

Note : The range is (-180, 180], meaning it is exclusive of -180° and inclusive of 180°.

There are some places you may choose to use a different range, such as 0° to 360° or 0 to 1 rotation, but be aware that many core WPILib classes and FRC tools are built with the unit circle above.

Avertissement : Some *gyroscope* and *IMU* models use CW positive rotation, such as the NavX IMU. Care must be taken to handle rotation properly, sensor values may need to be inverted. Read the documentation and verify that rotation is CCW positive.

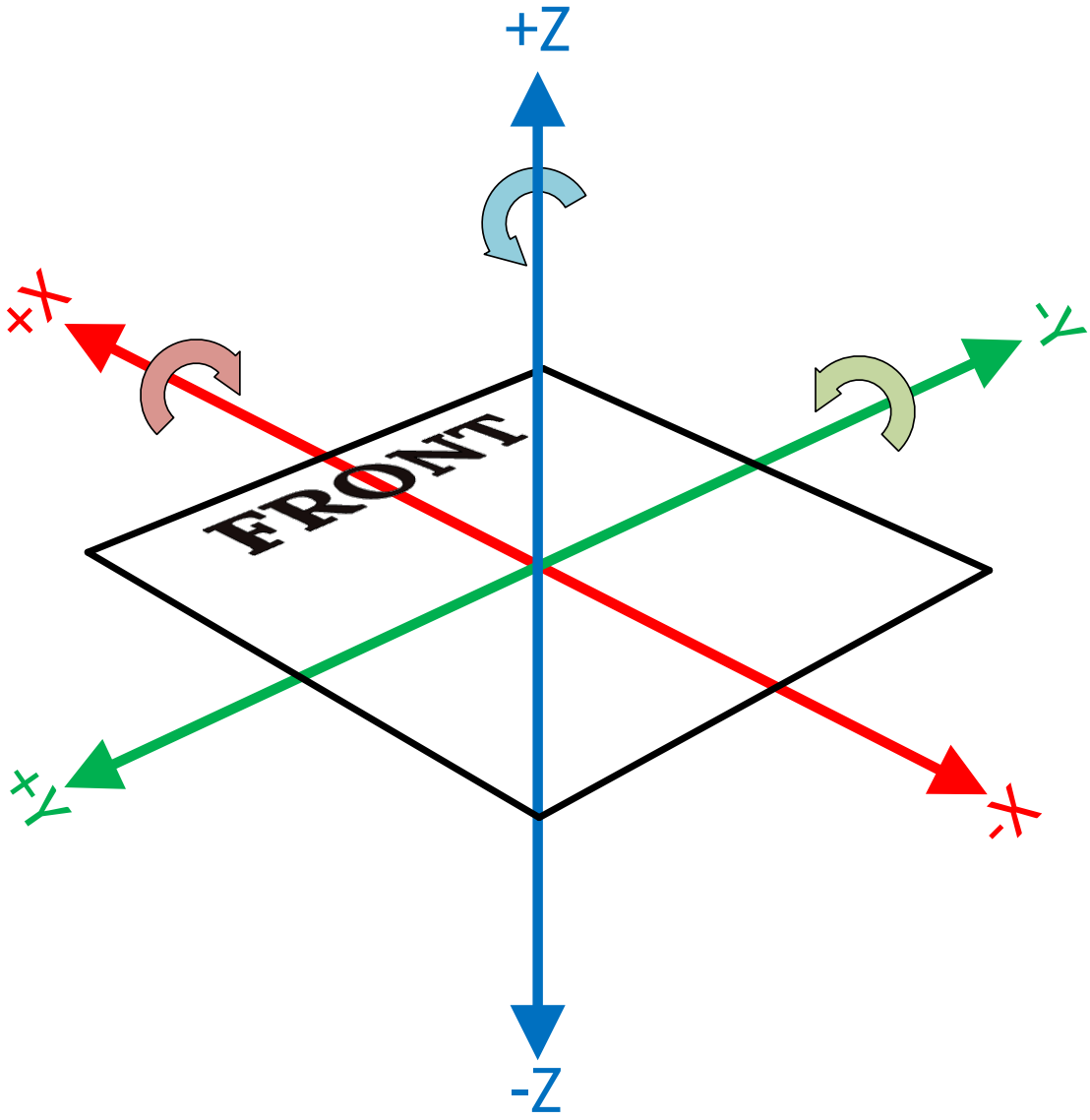


Fig. 1 – Robot coordinate system in three dimensions

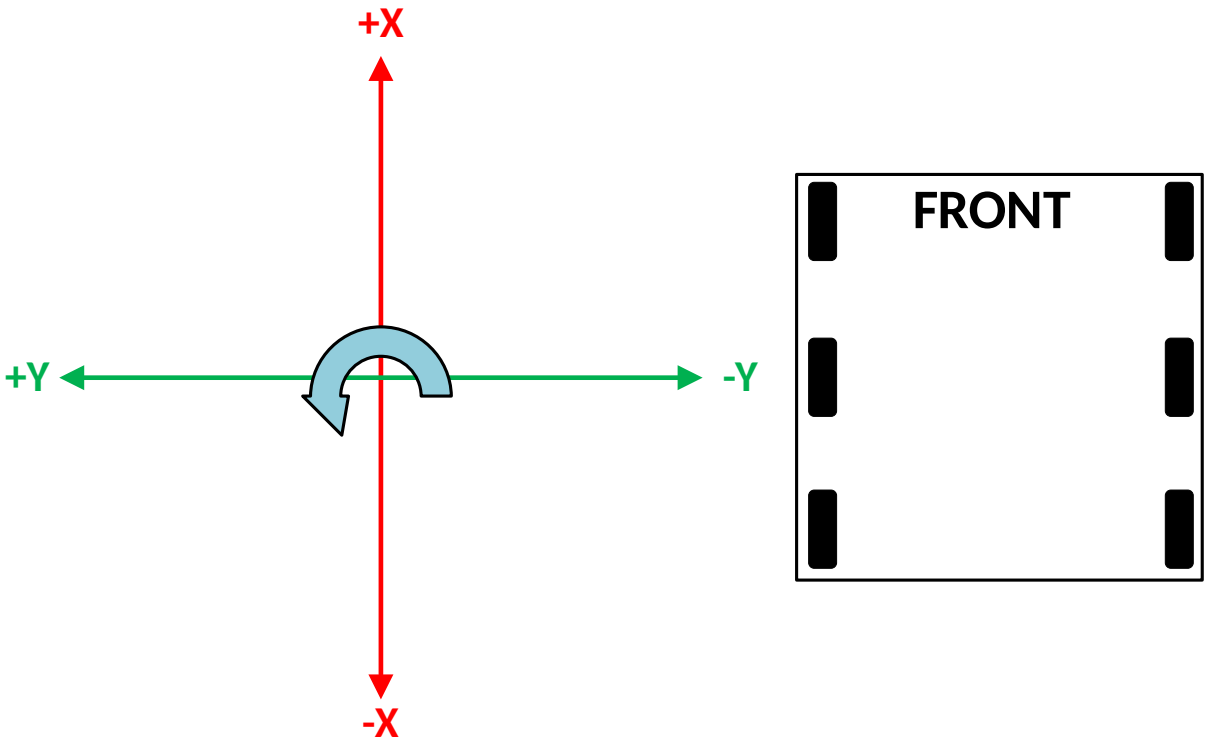


Fig. 2 – Robot coordinate system in two dimensions

Avertissement : Many sensors that read rotation around an axis, such as encoders and IMU's, read continuously. This means they read more than one rotation, so when rotating past 180° they read 181°, not -179°. Some sensors have configuration settings where you can choose their wrapping behavior and range, while others need to be handled in your code. Careful attention should be paid to make sure sensor readings are consistent and your control loop handles wrapping in the same way as your sensor.

17.5.3 Joystick and controller coordinate system

Joysticks, including the sticks on controllers, don't use the same NWU coordinate system. They use the NED (North-East-Down) convention, where the positive X axis points ahead, the positive Y axis points right, and the positive Z axis points down. When viewed with each positive axis pointing toward you, counter-clockwise (CCW) is a positive value and clockwise (CW) is a negative value.

It's important to note that joystick input values are rotations around an axis, not translations. In practical terms, this means :

- pushing forward on the joystick (toward the positive X axis) is a CW rotation around the Y axis, so you get a negative Y value.
- pushing to the right (toward the positive Y axis) is a CCW rotation around the X axis, so you get a positive X value.
- twisting the joystick CW (toward the positive Y axis) is a CCW rotation around the Z axis, so you get a positive Z value.

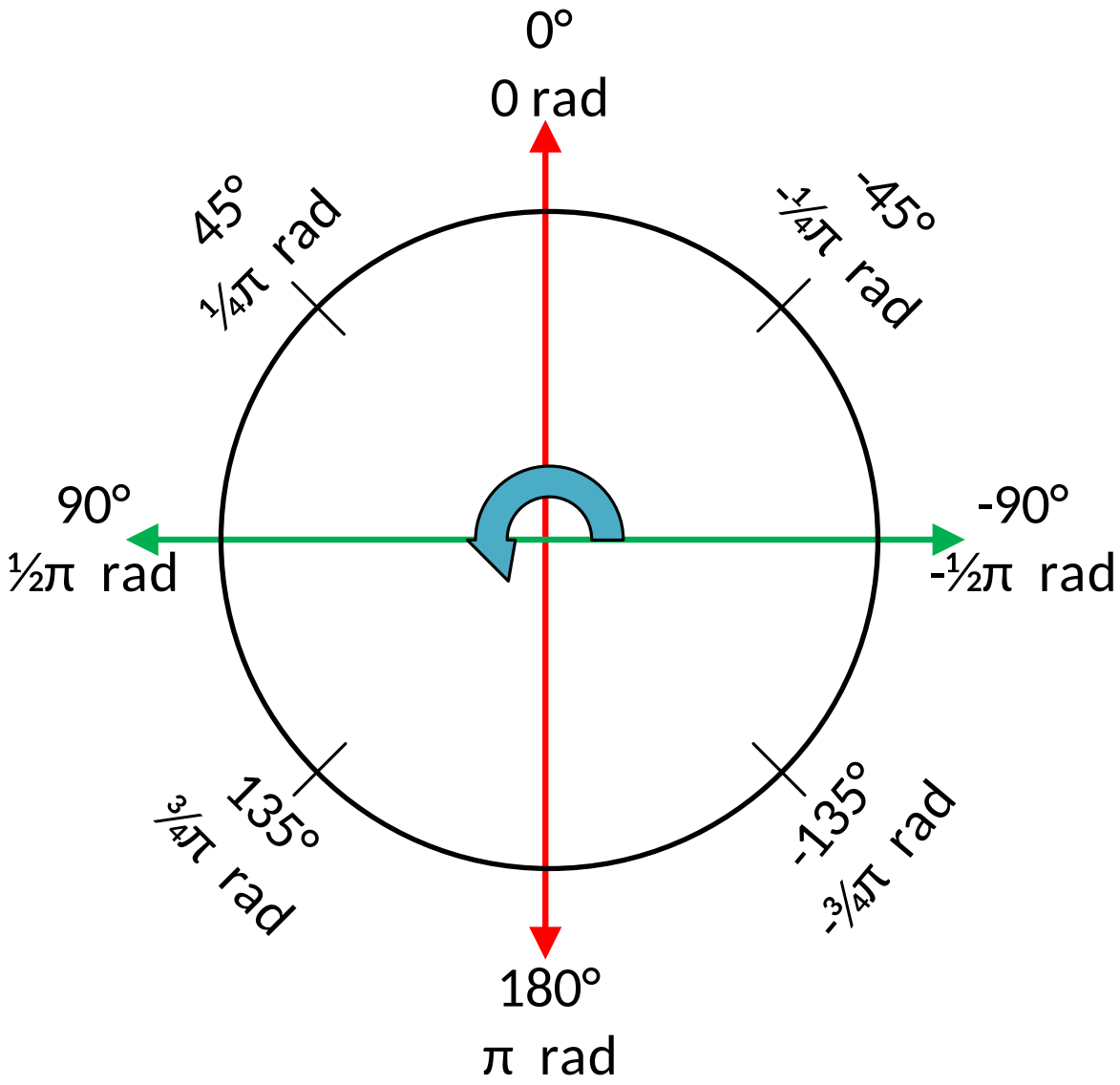


Fig. 3 – Unit circle with common angles

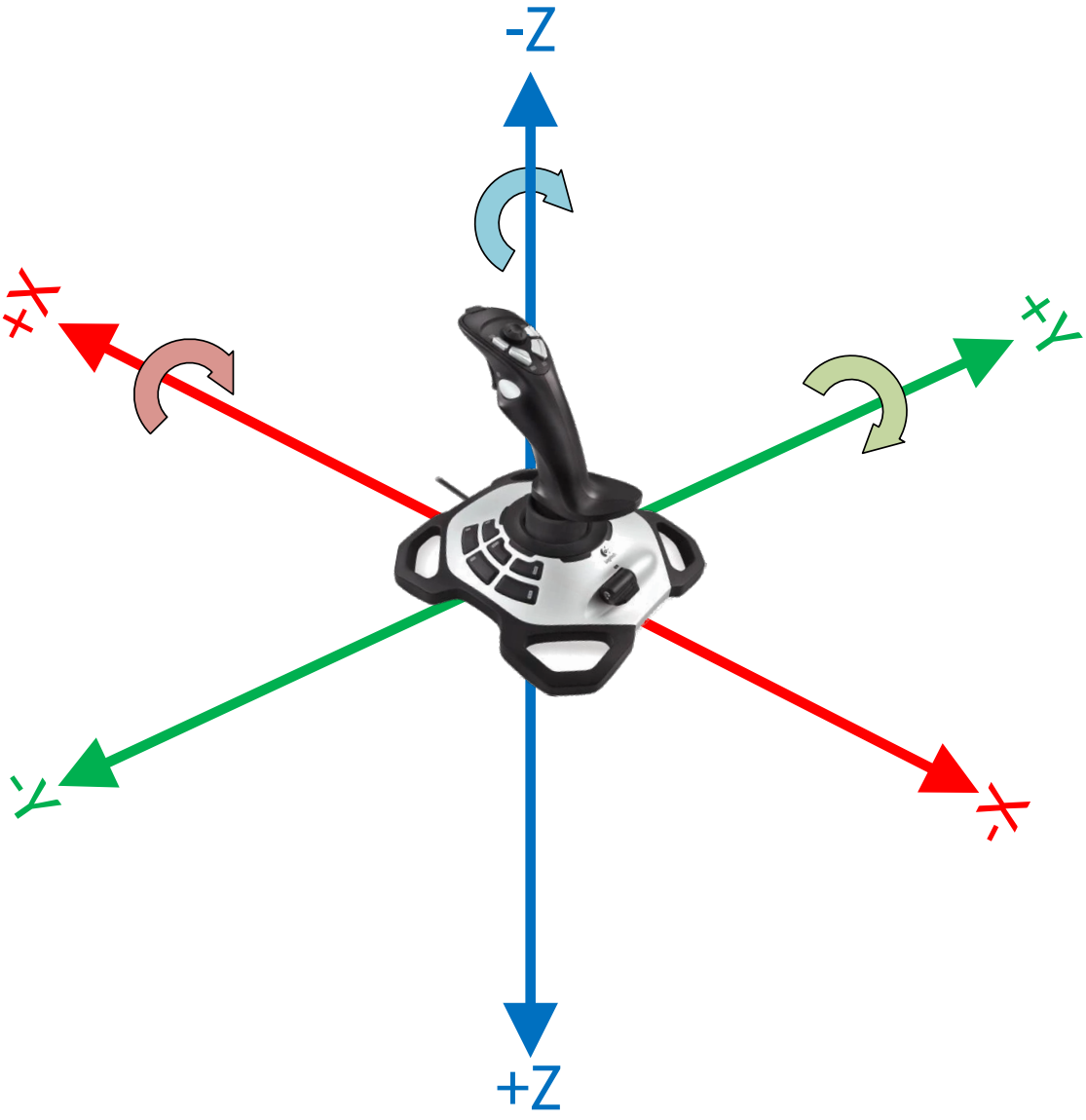


Fig. 4 – Joystick coordinate system

17.5.4 Using Joystick and controller input to drive a robot

You may have noticed, the coordinate system used by WPILib for the robot is not the same as the coordinate system used for joysticks and controllers. Care needs to be taken to understand the difference, and properly pass driver input to the drive subsystem.

Differential drivetrain example

Differential drivetrains are non-holonomic, which means the robot drivetrain cannot move side-to-side (strafe). This type of drivetrain can move forward and backward along the X axis, and rotate around the Z axis. Consider a common arcade drive scheme using a single joystick where the driver pushes the joystick forward/backward for forward/backward robot movement, and push the joystick left/right to rotate the robot left/right.

The code snippet below uses the `DifferentialDrive` and `Joystick` classes to drive the robot with the arcade scheme described above. `DifferentialDrive` uses the robot coordinate system defined above, and `Joystick` uses the joystick coordinate system.

JAVA

```
public void teleopPeriodic() {
    // Arcade drive with a given forward and turn rate
    myDrive.arcadeDrive(-driveStick.getY(), -driveStick.getX());
}
```

C++

```
void TeleopPeriodic() override {
    // Arcade drive with a given forward and turn rate
    myDrive.ArcadeDrive(-driveStick.GetY(), -driveStick.GetX());
}
```

PYTHON

```
def teleopPeriodic(self):
    # Arcade drive with a given forward and turn rate
    self.myDrive.arcadeDrive(-self.driveStick.getY(), -self.driveStick.getX())
```

The code calls the `DifferentialDrive.arcadeDrive(xSpeed, zRotation)` method, with values it gets from the `Joystick` class :

- The first argument is `xSpeed`
  - Robot : `xSpeed` is the speed along the robot's X axis, which is forward/backward.
  - Joystick : The driver sets forward/backward speed by rotating the joystick along its Y axis, which is pushing the joystick forward/backward.
  - Code : Moving the joystick forward is negative Y rotation, whereas moving the robot forward is along the positive X axis. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.
- The second argument is `zRotation`

- Robot : `zRotation` is the speed of rotation along the robot's Z axis, which is rotating left/right.
- Joystick : The driver sets rotation speed by rotating the joystick along its X axis, which is pushing the joystick left/right.
- Code : Moving the joystick to the right is positive X rotation, whereas robot rotation is CCW positive. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.

Mecanum drivetrain example

Mecanum drivetrains are holonomic, meaning they have the ability to move side-to-side. This type of drivetrain can move forward/backward and rotate around the Z axis like differential drivetrains, but it can also move side-to-side along the robot's Y axis. Consider a common arcade drive scheme using a single joystick where the driver pushes the joystick forward/backward for forward/backward robot movement, pushes the joystick left/right to move side-to-side, and twists the joystick to rotate the robot.

JAVA

```
public void teleopPeriodic() {
    // Drive using the X, Y, and Z axes of the joystick.
    m_robotDrive.driveCartesian(-m_stick.getY(), -m_stick.getX(), -m_stick.getZ());
}
```

C++

```
void TeleopPeriodic() override {
    // Drive using the X, Y, and Z axes of the joystick.
    m_robotDrive.driveCartesian(-m_stick.GetY(), -m_stick.GetX(), -m_stick.GetZ());
}
```

PYTHON

```
def teleopPeriodic(self):
    // Drive using the X, Y, and Z axes of the joystick.
    self.robotDrive.driveCartesian(-self.stick.getY(), -self.stick.getX(), -self.
↪stick.getZ())
```

The code calls the `MecanumDrive.driveCartesian(xSpeed, ySpeed, zRotation)` method, with values it gets from the `Joystick` class :

- The first argument is `xSpeed`
  - Robot : `xSpeed` is the speed along the robot's X axis, which is forward/backward.
  - Joystick : The driver sets forward/backward speed by rotating the joystick along its Y axis, which is pushing the joystick forward/backward.
  - Code : Moving the joystick forward is negative Y rotation, whereas robot forward is along the positive X axis. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.
- The second argument is `ySpeed`
  - Robot : `ySpeed` is the speed along the robot's Y axis, which is left/right.

  - Joystick : The driver sets left/right speed by rotating the joystick along its X axis, which is pushing the joystick left/right.
  - Code : Moving the joystick to the right is positive X rotation, whereas robot right is along the negative Y axis. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.
- The third argument is `zRotation`
  - Robot : `zRotation` is the speed of rotation along the robot's Z axis, which is rotating left/right.
  - Joystick : The driver sets rotation speed by twisting the joystick along its Z axis, which is twisting the joystick left/right.
  - Code : Twisting the joystick to the right is positive Z rotation, whereas robot rotation is CCW positive. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.

Swerve drivetrain example

Like mecanum drivetrains, swerve drivetrains are holonomic and have the ability to move side-to-side. Joystick control can be handled the same way for all holonomic drivetrains, but WPILib doesn't have a built-in robot drive class for swerve. Swerve coding is described in other sections of this documentation, but an example of using joystick input to set `ChassisSpeeds` values is included below. Consider the same common arcade drive scheme described in the mecanum section above. The scheme uses a single joystick where the driver pushes the joystick forward/backward for forward/backward robot movement, pushes the joystick left/right to move side-to-side, and twists the joystick to rotate the robot.

JAVA

```
// Drive using the X, Y, and Z axes of the joystick.
var speeds = new ChassisSpeeds(-m_stick.getY(), -m_stick.getX(), -m_stick.getZ());
```

C++

```
// Drive using the X, Y, and Z axes of the joystick.
frc::ChassisSpeeds speeds{-m_stick.GetY(), -m_stick.GetX(), -m_stick.GetZ()};
```

PYTHON

```
# Drive using the X, Y, and Z axes of the joystick.
speeds = ChassisSpeeds(-self.stick.getY(), -self.stick.getX(), -self.stick.getZ())
```

The three arguments to the `ChassisSpeeds` constructor are the same as `driveCartesian` in the mecanum section above; `xSpeed`, `ySpeed`, and `zRotation`. See the description of the arguments, and their joystick input in the section above.

17.5.5 Robot drive kinematics

Kinematics is a topic that is covered in a different section, but it's worth discussing here in relation to the coordinate system. It is critically important that kinematics is configured using the coordinate system described above. Kinematics is a common starting point for coordinate system errors that then cascade to basic drivetrain control, field oriented driving, pose estimation, and path planning.

When you construct a `SwerveDriveKinematics` or `MecanumDriveKinematics` object, you specify a translation from the center of your robot to each wheel. These translations use the coordinate system above, with the origin in the center of your robot.

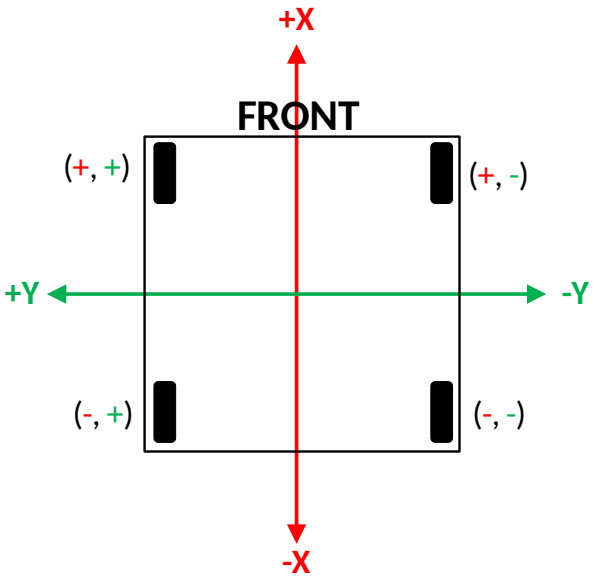


Fig. 5 – Kinematics with translation signs

For the robot in the diagram above, let's assume the distance between the front and rear wheels (wheelbase) is 2". Let's also assume the distance between the left and right wheels (trackwidth) is also 2". Our translations (x, y) would be like this :

- Front left : (1", 1")
- Front right : (1", -1")
- Rear left : (-1", 1")
- Rear right : (-1", -1")

Avertissement : A common error is to use an incorrect coordinate system where the positive Y axis points forward on the robot. The correct coordinate system has the positive X axis pointing forward.

17.5.6 Field coordinate systems

The field coordinate system (or global coordinate system) is an absolute coordinate system where a point on the field is designated as the origin. Two common uses of the field coordinate system will be explored in this document :

— Field oriented driving is a drive scheme for holonomic drivetrains, where the driver moves the controls relative to their perspective of the field, and the robot moves in that direction regardless of where the front of the robot is facing. For example, a driver on the red alliance pushes the joystick forward, the robot will move downfield toward the blue alliance wall, even if the robot's front is facing the driver.
— Pose estimation with odometry and/or AprilTags are used to estimate the robot's pose on the field.

Mirrored field vs. rotated field

Historically, FRC has used two types of field layouts in relation to the red and blue alliance.

Games such as Rapid React in 2022 used a rotated layout. A rotated layout means that, from your perspective from behind your alliance wall, your field elements and your opponent's elements are in the same location. Notice in the Rapid React field layout diagram below, whether you are on the red or blue alliance, your human player station is on your right and your hanger is on your left.

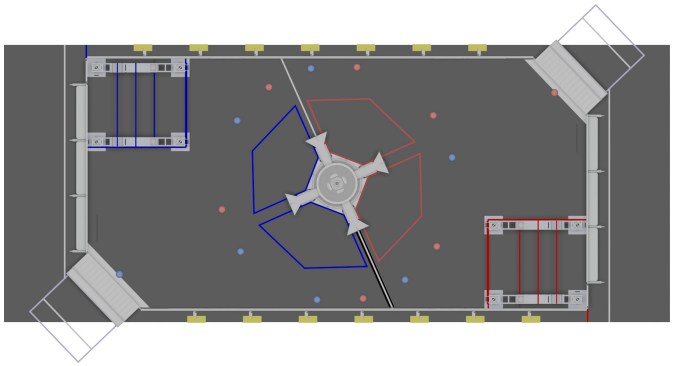


Fig. 6 – Rotated field from RAPID REACT in 2022[1]

Games such as CHARGED UP in 2023 and CRESCENDO in 2024 used a mirrored layout. A mirrored layout means that the red and blue alliance layout are mirrored across the center-point of the field. Refer to the CHARGED UP field diagram below. When you are standing behind the blue alliance wall, the charge station is on the right side of the field from your perspective. However, standing behind the red alliance wall, the charge station is on the left side of the field from your perspective.

1. Rapid React field image from MikLast on Chiefdelphi https://www.chiefdelphi.com/t/2022-top-down-field-renders/399031
2. CHARGED UP field image from MikLast on Chiefdelphi https://www.chiefdelphi.com/t/2023-top-down-field-renders/421365

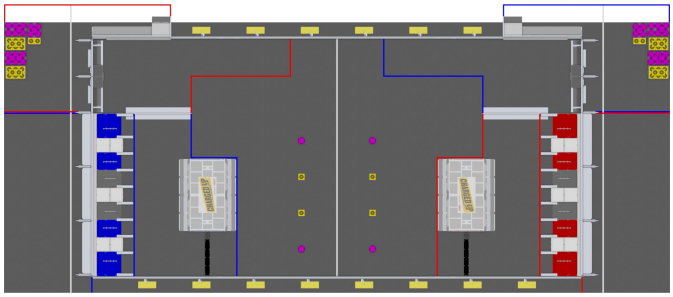


Fig. 7 – Mirrored field from CHARGED UP in 2023[2]

Dealing with red or blue alliance

There are two primary ways many teams choose to define the field coordinate system. In both methods, positive rotation (theta) is in the counter-clockwise (CCW) direction.

Avertissement : There are cases where your alliance may change (or appear to change) after the code is initialized. When you are not connected to the *FMS* at a competition, you can change your alliance station in the Driver Station application at any time. Even when you are at a competition, your robot will usually initialize before connecting to the FMS so you will not have alliance information.

Note : At competition events, the FMS will automatically report your Team Station and alliance color. When you are not connected to an FMS, you can choose your Team Station and alliance color on the Driver Station *Onglet Operation*.

Always blue origin

You may choose to define the origin of the field on the blue side, and keep it there regardless of your alliance color. With this solution, positive x-axis points away from the blue alliance wall.

Some advantages to this approach are :

- Pose estimation with AprilTags is simplified. AprilTags throughout the field are unique. If you keep the coordinate system the same regardless of alliance, there is no need for special logic to deal with the location of AprilTags on the field relative to your alliance.
- Many of the tools and libraries used in FRC follow this convention. Some of the tools include : PathPlanner, Choreo, and the ShuffleBoard and Glass Field2d widget.

In order to use this approach for field oriented driving, driver input needs to consider the alliance color. When your alliance is red and the driver is standing behind the red alliance wall, they will want the robot to move downfield toward the blue alliance wall. However, when your alliance is blue, the driver will want the robot to go downfield toward the red alliance wall.

A simple way to deal with field oriented driving is to check the alliance color reported by the *DriverStation* class, and invert the driver's controls based on the alliance. As noted above, your alliance color can change so it needs to be checked on every robot iteration.

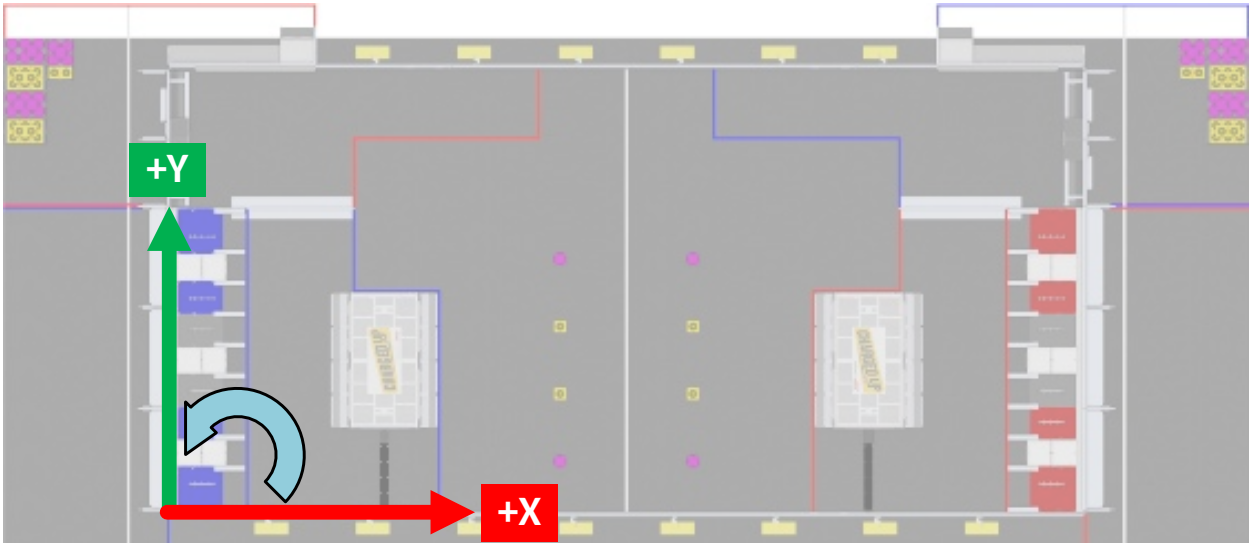


Fig. 8 – CHARGED UP with blue origin

JAVA

```
// The origin is always blue. When our alliance is red, X and Y need to be inverted
var alliance = DriverStation.getAlliance();
var invert = 1;
if (alliance.isPresent() && alliance.get() == Alliance.Red) {
    invert = -1;
}

// Create field relative ChassisSpeeds for controlling Swerve
var chassisSpeeds = ChassisSpeeds
        .fromFieldRelativeSpeeds(xSpeed * invert, ySpeed * invert, zRotation, imu.
↪getRotation2d());

// Control a mecanum drivetrain
m_robotDrive.driveCartesian(xSpeed * invert, ySpeed * invert, zRotation, imu.
↪getRotation2d());
```

C++

```
// The origin is always blue. When our alliance is red, X and Y need to be inverted
int invert = 1;
if (frc::DriverStation::GetAlliance() == frc::DriverStation::Alliance::kRed) {
    invert = -1;
}

// Create field relative ChassisSpeeds for controlling Swerve
frc::ChassisSpeeds chassisSpeeds =
        frc::ChassisSpeeds::FromFieldRelativeSpeeds(xSpeed * invert, ySpeed * invert,␣
↪zRotation, imu.GetRotation2d());

// Control a mecanum drivetrain
```

(suite sur la page suivante)

(suite de la page précédente)

```
m_robotDrive.driveCartesian(xSpeed * invert, ySpeed * invert, zRotation, imu.
↪GetRotation2d());
```

PYTHON

```
# The origin is always blue. When our alliance is red, X and Y need to be inverted
invert = 1
if wpilib.DriverStation.getAlliance() == wpilib.DriverStation.Alliance.kRed:
    invert = -1

# Create field relative ChassisSpeeds for controlling Swerve
chassis_speeds = wpilib.ChassisSpeeds.FromFieldRelativeSpeeds(
    xSpeed * invert, ySpeed * invert, zRotation, self.imu.GetAngle()
)

# Control a mecanum drivetrain
self.robotDrive.driveCartesian(xSpeed * invert, ySpeed * invert, zRotation, self.imu.
↪GetAngle())
```

Origin follows your alliance

You may choose to define the origin of the field based on the alliance you are one. With this approach, the positive x-axis always points away from your alliance wall.

When you are on the blue alliance, your origin looks like this :

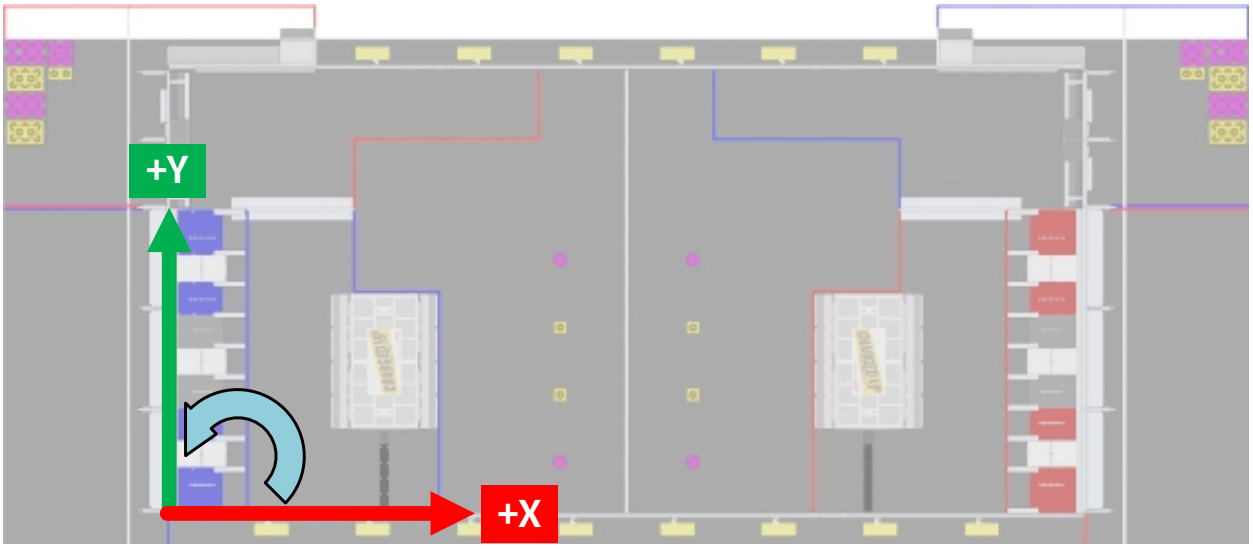


Fig. 9 – CHARGED UP field with blue alliance as origin

When you are on the red alliance, your origin looks like this :

This approach has a few more complications than the previous approach, especially in years when the field layout is mirrored between alliances.

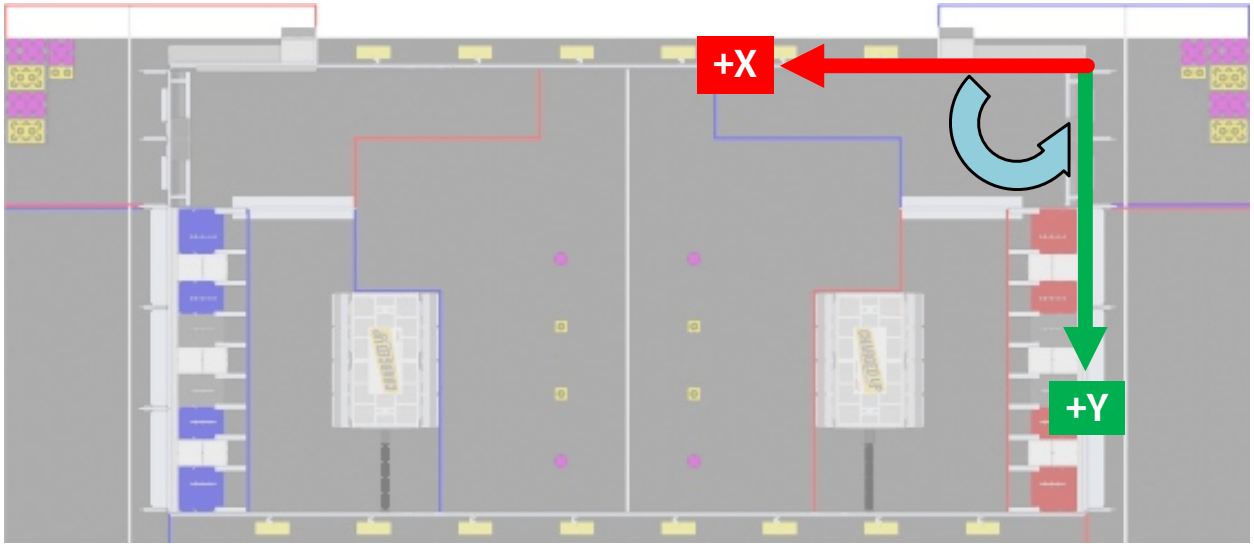


Fig. 10 – CHARGED UP field with red alliance as origin

In years when the field layout is rotated, this is a simple approach if you are not using AprilTags for pose estimation or doing other advanced techniques. When the field layout is rotated, the field elements appear at the same coordinates regardless of your alliance.

Some things you need to consider when using this approach are :

— As warned above, your alliance color can change after initialization. If you are not using AprilTags, you may not have anything to adjust when the alliance changes. However, if you are using AprilTags and your robot has seen a tag and used it for pose estimation, you will need to adjust your origin and reset your estimated pose.

— The field image in the ShuffleBoard and Glass Field2d widget follows the *Always blue origin* approach. Special handling is needed to display your robot pose correctly when your alliance is red. You will need to change the origin for your estimated pose to the blue alliance coordinate system before sending it to the dashboard.

17.6 Définition des préférences du robot

The Robot Preferences (Java, C++) class is used to store values in the flash memory on the roboRIO. The values might be for remembering preferences on the robot such as calibration settings for potentiometers, PID values, setpoints, etc. that you would like to change without having to rebuild the program. The values can be viewed on SmartDashboard or Shuffleboard and read and written by the robot program.

Cet exemple montre comment utiliser les Préférences pour modifier le point de consigne d'un régulateur PID et la constante P. Les exemples de code sont adaptés de l'exemple Arm Simulation (Java, C++). Vous pouvez exécuter l'exemple de simulation de bras dans le simulateur de robot pour voir comment utiliser la classe de préférences et interagir avec elle à l'aide des tableaux de bord sans avoir besoin d'un robot.

17.6.1 Initialisation des préférences

Java

```
  public static final String kArmPositionKey = "ArmPosition";
  public static final String kArmPKey = "ArmP";

  // The P gain for the PID controller that drives this arm.
  public static final double kDefaultArmKp = 50.0;
  public static final double kDefaultArmSetpointDegrees = 75.0;
```

```
  // The P gain for the PID controller that drives this arm.
  private double m_armKp = Constants.kDefaultArmKp;
  private double m_armSetpointDegrees = Constants.kDefaultArmSetpointDegrees;
  public Arm() {
    m_encoder.setDistancePerPulse(Constants.kArmEncoderDistPerPulse);
    // Set the Arm position setpoint and P constant to Preferences if the keys don't␣
↪already exist
    Preferences.initDouble(Constants.kArmPositionKey, m_armSetpointDegrees);
    Preferences.initDouble(Constants.kArmPKey, m_armKp);
  }
```

C++

```
inline constexpr std::string_view kArmPositionKey = "ArmPosition";
inline constexpr std::string_view kArmPKey = "ArmP";

inline constexpr double kDefaultArmKp = 50.0;
inline constexpr units::degree_t kDefaultArmSetpoint = 75.0_deg;
```

```
Arm::Arm() {
  // Set the Arm position setpoint and P constant to Preferences if the keys
  // don't already exist
  frc::Preferences::InitDouble(kArmPositionKey, m_armSetpoint.value());
  frc::Preferences::InitDouble(kArmPKey, m_armKp);
}
```

Python

```
    kArmPositionKey = "ArmPosition"
    kArmPKey = "ArmP"

    # The P gain for the PID controller that drives this arm.
    kDefaultArmKp = 50.0
    kDefaultArmSetpointDegrees = 75.0
```

```
        # The P gain for the PID controller that drives this arm.
        self.armKp = Constants.kDefaultArmKp
        self.armSetpointDegrees = Constants.kDefaultArmSetpointDegrees

        # Set the Arm position setpoint and P constant to Preferences if the keys don
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪'t already exist
        wpilib.Preferences.initDouble(
            Constants.kArmPositionKey, self.armSetpointDegrees
        )
        wpilib.Preferences.initDouble(Constants.kArmPKey, self.armKp)
```

Les préférences sont stockées sous un nom, la clé. Il est utile de stocker la clé dans une constante, comme `kArmPositionKey` et `kArmPKey` dans le code ci-dessus pour éviter de la taper plusieurs fois et d'éviter les fautes de frappe. Nous déclarons également des variables, `kArmKp` et `armPositionDeg` pour contenir les données récupérées à partir des préférences.

Dans `robotInit`, chaque clé est vérifiée pour voir si elle existe déjà dans la base de données des Préférences. La méthode `containsKey` prend un paramètre, la clé pour vérifier si les données pour cette clé existent déjà dans la base de données des préférences. S'il n'existe pas, une valeur par défaut est écrite. La méthode `setDouble` prend deux paramètres, la clé à écrire et les données à écrire. Il existe des méthodes similaires pour d'autres types de données comme les booléens, les entiers et les chaînes.

Si vous utilisez Command Framework, ce type de code peut être placé dans le constructeur d'un sous-système ou d'une commande.

17.6.2 Préférences de lecture

Java

```
  public void loadPreferences() {
    // Read Preferences for Arm setpoint and kP on entering Teleop
    m_armSetpointDegrees = Preferences.getDouble(Constants.kArmPositionKey, m_
↪armSetpointDegrees);
    if (m_armKp != Preferences.getDouble(Constants.kArmPKey, m_armKp)) {
      m_armKp = Preferences.getDouble(Constants.kArmPKey, m_armKp);
      m_controller.setP(m_armKp);
    }
  }
```

C++

```
void Arm::LoadPreferences() {
  // Read Preferences for Arm setpoint and kP on entering Teleop
  m_armSetpoint = units::degree_t{
      frc::Preferences::GetDouble(kArmPositionKey, m_armSetpoint.value())};
  if (m_armKp != frc::Preferences::GetDouble(kArmPKey, m_armKp)) {
    m_armKp = frc::Preferences::GetDouble(kArmPKey, m_armKp);
    m_controller.SetP(m_armKp);
  }
}
```

Python

```
    def loadPreferences(self):
        # Read Preferences for Arm setpoint and kP on entering Teleop
        self.armSetpointDegrees = wpilib.Preferences.getDouble(
            Constants.kArmPositionKey, self.armSetpointDegrees
        )
        if self.armKp != wpilib.Preferences.getDouble(Constants.kArmPKey, self.armKp):
            self.armKp = wpilib.Preferences.getDouble(Constants.kArmPKey, self.armKp)
            self.controller.setP(self.armKp)
```

Reading a preference is easy. The `getDouble` method takes two parameters, the key to read, and a default value to use in case the preference doesn't exist. There are similar methods for other data types like booleans, ints, and strings.

Selon les données stockées dans les préférences, vous pouvez les utiliser lorsque vous les lisez, comme la constante proportionnelle ci-dessus. Ou vous pouvez le stocker dans une variable et l'utiliser plus tard, comme le point de consigne, qui est utilisé dans `telopPeriodic` ci-dessous.

Java

```
  @Override
  public void teleopPeriodic() {
    if (m_joystick.getTrigger()) {
      // Here, we run PID control like normal.
      m_arm.reachSetpoint();
    } else {
      // Otherwise, we disable the motor.
      m_arm.stop();
    }
  }
```

```
  /** Run the control loop to reach and maintain the setpoint from the preferences. */
  public void reachSetpoint() {
    var pidOutput =
        m_controller.calculate(
            m_encoder.getDistance(), Units.degreesToRadians(m_armSetpointDegrees));
    m_motor.setVoltage(pidOutput);
  }
```

C++

```
void Robot::TeleopPeriodic() {
  if (m_joystick.GetTrigger()) {
    // Here, we run PID control like normal.
    m_arm.ReachSetpoint();
  } else {
    // Otherwise, we disable the motor.
    m_arm.Stop();
  }
}
```

```cpp
void Arm::ReachSetpoint() {
  // Here, we run PID control like normal, with a setpoint read from
  // preferences in degrees.
  double pidOutput = m_controller.Calculate(
      m_encoder.GetDistance(), (units::radian_t{m_armSetpoint}.value()));
  m_motor.SetVoltage(units::volt_t{pidOutput});
}
```

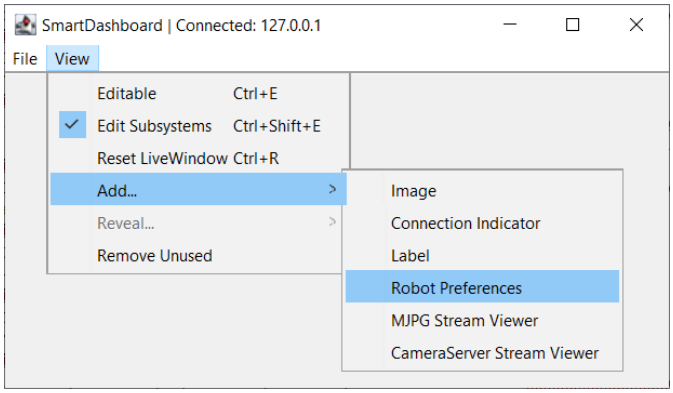
Python

```python
    def teleopPeriodic(self):
        if self.joystick.getTrigger():
            # Here, we run PID control like normal.
            self.arm.reachSetpoint()
        else:
            # Otherwise, we disable the motor.
            self.arm.stop()
```

```python
    def reachSetpoint(self):
        pidOutput = self.controller.calculate(
            self.encoder.getDistance(),
            units.degreesToRadians(self.armSetpointDegrees),
        )
        self.motor.setVoltage(pidOutput)
```

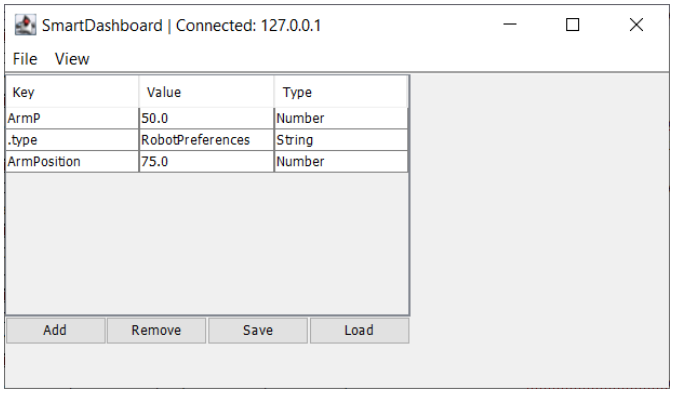
17.6.3 Utilisation des préférences dans SmartDashboard

Affichage des préférences dans SmartDashboard



Dans le SmartDashboard, l'affichage des Préférences peut être ajouté à l'affichage en sélectionnant *View* puis *Add...* puis *Robot Preferences*. Cela révèle le contenu du fichier de préférences stocké dans la mémoire flash du roboRIO.

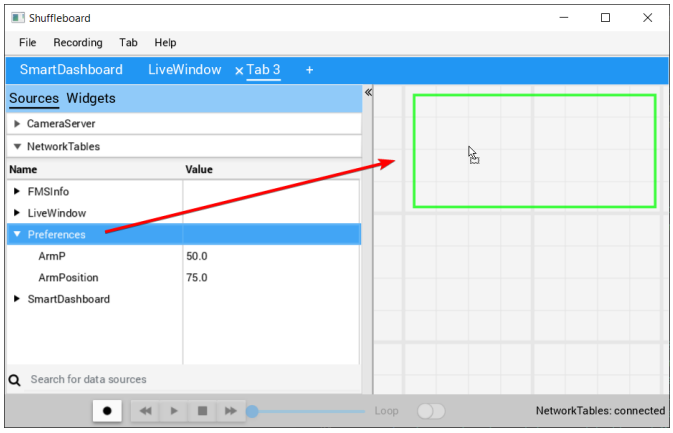
Modification des préférences dans SmartDashboard



Les valeurs sont affichées ici avec les valeurs par défaut du code. Si les valeurs doivent être ajustées, elles peuvent être modifiées ici et enregistrées.

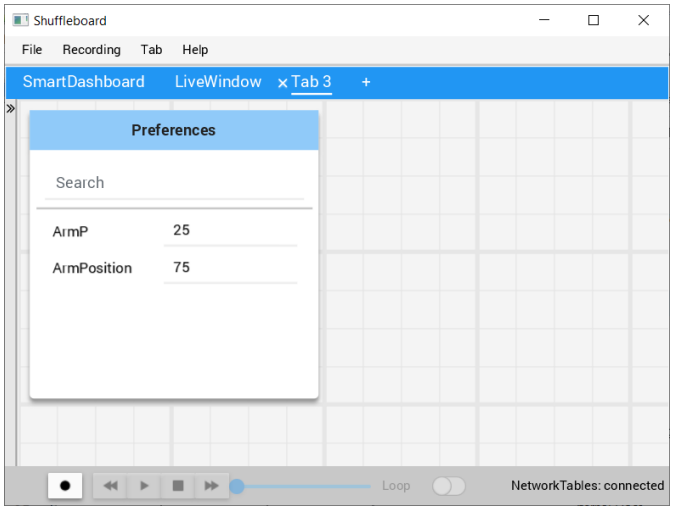
17.6.4 Utilisation des préférences dans Shuffleboard

Affichage des préférences dans Shuffleboard



Dans Shuffleboard, l'affichage des Préférences peut être ajouté à l'affichage en faisant glisser le champ des préférences depuis la fenêtre des sources. Cela révèle le contenu du fichier de préférences stocké dans la mémoire flash du roboRIO.

Modification des préférences dans Shuffleboard

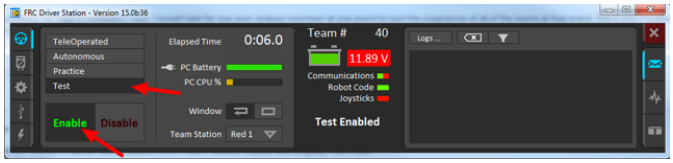


Les valeurs sont affichées ici avec les valeurs par défaut du code. Si les valeurs doivent être ajustées, elles peuvent être modifiées ici.

17.7 Utilisation du Mode Test

Le mode test est conçu pour permettre aux programmeurs d'avoir un endroit où mettre du code pour vérifier que tous les systèmes du robot fonctionnent. Dans chacun des modèles de programme de robot, il y a un endroit pour ajouter du code de test au robot.

17.7.1 Activation du Mode Test



Le mode test est conçu pour permettre aux programmeurs d'avoir un endroit où mettre du code pour vérifier que tous les systèmes du robot fonctionnent. Dans chacun des modèles de programme de robot, il y a un endroit pour ajouter du code de test au robot.

17.7.2 Ajout du code du Mode Test à votre code robot

When in test mode, the `testInit` method is run once, and the `testPeriodic` method is run once per tick, in addition to `robotPeriodic`, similar to teleop and autonomous control modes.

L'ajout du mode test peut être aussi simple que d'appeler vos méthodes Teleop déjà écrites à partir de Test. Ou vous pouvez écrire un code spécial pour essayer une nouvelle fonctionnalité qui ne s'exécute qu'en mode test, avant de l'intégrer dans votre code téléop ou autonome. Vous pouvez même écrire du code pour déplacer tous les moteurs et vérifier tous les capteurs pour aider l'équipe du stand !

17.7.3 LiveWindow en mode test

Important : Since 2024, LiveWindow in Test Mode is disabled by default ! See *Enabling LiveWindow in Test Mode* to enable it.

With LiveWindow, all actuator outputs can be controlled on the Dashboard and all sensor values can be seen. PID Controllers can also be tuned. The sensors and actuators are added automatically, no code is necessary. See *SmartDashboard : Test Mode et LiveWindow* for more details.

17.8 Lecture des traces de pile (StackTraces)

`Une erreur inattendue s'est produite.`

Lorsque le code de votre robot rencontre une erreur inattendue, vous verrez ce message s'afficher dans certaines sorties de la console (Driver Station ou RioLog). Vous remarquerez probablement aussi que votre robot s'arrête brusquement, ou ne bouge peut-être jamais. Ces erreurs inattendues sont appelées *exceptions non gérées*.

Lorsqu'une exception non gérée se produit, cela signifie que votre code contient un ou plusieurs bogues qui doivent être corrigés.

Cet article explorera certains des outils et techniques impliqués dans la recherche et la correction de ces bogues.

17.8.1 Qu'est-ce qu'une «trace de pile» ?

Le message `une erreur inattendue s'est produite` est un signal qu'une *trace de pile* a été imprimée.

In Java and C++, the *call stack* data structure is used to store information about which function or method is currently being executed.

Une *trace de pile* imprime des informations sur ce qui se trouvait sur cette pile lorsque l'exception non gérée s'est produite. Cela vous indique les lignes de code qui s'exécutaient juste avant que le problème ne se produise. Bien que cela ne vous indique pas toujours la *cause principale* exacte de votre problème, c'est généralement le meilleur endroit pour commencer à chercher.

17.8.2 Qu'est-ce qu'une «exception non gérée» ?

Une erreur irrécupérable est une condition qui survient dans laquelle le processeur ne peut pas continuer à exécuter du code. Cela implique presque toujours que, même si le code a été compilé et a commencé à s'exécuter, il n'est plus logique que l'exécution se poursuive.

Dans presque tous les cas, la cause première d'une exception non gérée est un code qui n'est pas correctement implémenté. Cela n'implique presque jamais qu'un matériel a mal fonctionné.

17.8.3 Alors, comment puis-je résoudre mon problème ?

Lire la trace de la pile

Pour commencer, recherchez au-dessus de `unexpected error has occurred` la trace de la pile.

Java

En Java, cela devrait ressembler à ceci :

```
Error at frc.robot.Robot.robotInit(Robot.java:24): Unhandled exception: java.lang.
↪NullPointerException
        at frc.robot.Robot.robotInit(Robot.java:24)
        at edu.wpi.first.wpilibj.TimedRobot.startCompetition(TimedRobot.java:94)
        at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:335)
        at edu.wpi.first.wpilibj.RobotBase.lambda$startRobot$0(RobotBase.java:387)
        at java.base/java.lang.Thread.run(Thread.java:834)
```

Il y a quelques éléments importants à retenir ici :
- Il y a eu une `Erreur`
- L'erreur était due à une `Exception non gérée` (Unhandled Exception)
- L'exception était une `java.lang.NullPointerException`
- L'erreur s'est produite lors de l'exécution de la ligne 24 à l'intérieur de `Robot.java`
  - `robotInit` était le nom de la méthode en cours d'exécution lorsque l'erreur s'est produite.
- `robotInit` est une fonction du package `frc.robot.Robot` (AKA, le code de votre équipe)
- `robotInit` a été appelé à partir d'un certain nombre de fonctions du package `edu.wpi.first.wpilibj` (AKA, les bibliothèques WPILib)

La liste des lignes en retrait commençant par le mot `at` représente l'état de la *pile* au moment où l'erreur s'est produite. Chaque ligne représente une méthode, qui a été *appelée par* la méthode juste en dessous.

Par exemple, si l'erreur s'est produite profondément dans votre base de code, vous pourriez voir plus d'entrées sur la pile :

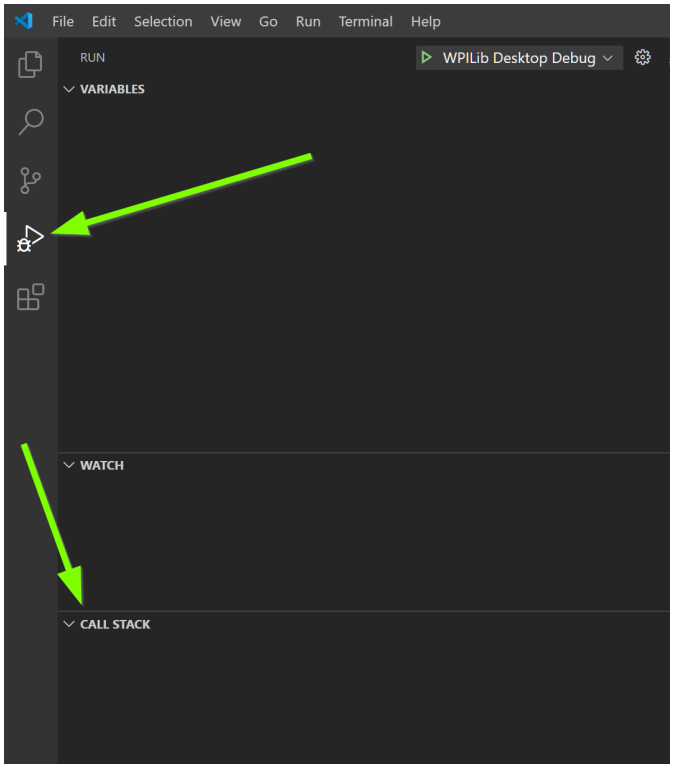
```
Error at frc.robot.Robot.buggyMethod(TooManyBugs.java:1138): Unhandled exception:␣
↪java.lang.NullPointerException
        at frc.robot.Robot.buggyMethod(TooManyBugs.java:1138)
        at frc.robot.Robot.barInit(Bar.java:21)
        at frc.robot.Robot.fooInit(Foo.java:34)
        at frc.robot.Robot.robotInit(Robot.java:24)
        at edu.wpi.first.wpilibj.TimedRobot.startCompetition(TimedRobot.java:94)
        at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:335)
        at edu.wpi.first.wpilibj.RobotBase.lambda$startRobot$0(RobotBase.java:387)
        at java.base/java.lang.Thread.run(Thread.java:834)
```

Dans ce cas : `robotInit` appelé `fooInit`, qui à son tour appelé `barInit`, qui à son tour appelé `buggyMethod`. Ensuite, lors de l'exécution de `buggyMethod`, l'"`NullPointerException` s'est produite.

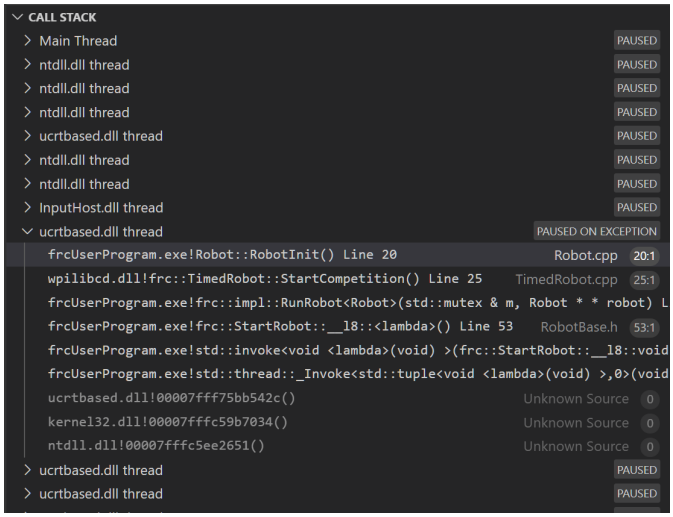
C++

Java produira généralement des traces de pile automatiquement lorsque les programmes rencontrent des problèmes. C++ nécessitera plus de fouilles pour extraire les mêmes informations. Habituellement, un débogueur en une seule étape devra être connecté au programme du robot en cours d'exécution.

Les traces de pile peuvent être trouvées dans l'onglet du débogueur de VS Code :



Les traces de pile en C++ ressembleront généralement à ceci :



Il y a quelques éléments importants à retenir ici :

— L'exécution du code est actuellement en pause.
— La raison pour laquelle il s'est arrêté était un thread ayant une `exception`

- L'erreur s'est produite lors de l'exécution de la ligne `20` à l'intérieur de `Robot.cpp`
  - `RobotInit` était le nom de la méthode en cours d'exécution lorsque l'erreur s'est produite.
- `RobotInit` est une fonction dans l'espace de noms `Robot ::` (AKA, le code de votre équipe)
- `RobotInit` a été appelé à partir d'un certain nombre de fonctions de l'espace de noms `frc::` (AKA, les bibliothèques WPILib)

Cette fenêtre « call stack » représente l'état de la *pile* au moment où l'erreur s'est produite. Chaque ligne représente une méthode, qui a été *appelée par* la méthode juste en dessous.

Les exemples de cette page supposent que vous exécutez des exemples de code en simulation, avec le débogueur connecté et surveillant les erreurs inattendues. Des techniques similaires devraient s'appliquer lors de l'exécution sur un vrai robot.

Effectuer une analyse de code

Une fois que vous avez trouvé la trace de la pile et trouvé les lignes de code qui déclenchent l'exception non gérée, vous pouvez commencer le processus de détermination de la cause première.

Souvent, le simple fait de regarder dans (ou à proximité) l'emplacement problématique dans le code sera fructueux. Vous remarquerez peut-être des choses que vous avez oubliées ou des lignes qui ne correspondent pas à un exemple auquel vous faites référence.

Note : Les développeurs qui ont beaucoup d'expérience avec le code auront souvent plus de chance d'examiner le code que les nouveaux. C'est bon, ne vous découragez pas ! L'expérience viendra avec le temps.

Une stratégie clé pour analyser le code consiste à poser les questions suivantes :

- When was the last time the code « worked » (I.e., didn't have this particular error) ?
- Qu'est-ce qui a changé dans le code entre la dernière version de travail et maintenant ?

Des tests fréquents et des modifications minutieuses du code contribuent à rendre cette stratégie particulière plus efficace.

Exécuter le débogueur en une seule étape

Parfois, il ne suffit pas de regarder le code pour détecter le problème. Le *débogueur en une seule étape* est une excellente option dans ce cas - il vous permet d'inspecter la série d'événements menant à l'exception non gérée.

Rechercher plus d'informations

Google is a phenomenal resource for understanding the root cause of errors. Searches involving the programming language and the name of the exception will often yield good results on more explanations for what the error means, how it comes about, and potential fixes.

Chercher de l'aide extérieure

Si tout le reste échoue, vous pouvez demander des conseils et de l'aide aux autres (en personne et en ligne). Lorsque vous travaillez avec des personnes qui ne connaissent pas votre base de code, il est très important de fournir les informations suivantes :

— Accès à votre code source, (Ex : *on github.com*)
— Le **texte complet** de l'erreur, y compris la trace de la pile complète.

17.8.4 Exemples et modèles courants

Il existe un certain nombre de problèmes courants qui entraînent des exceptions d'exécution.

Pointeurs nuls et références

C++ et Java ont tous deux le concept de « null » - ils l'utilisent pour indiquer quelque chose qui n'a pas encore été initialisé, et ne fait référence à rien de significatif.

La manipulation d'une référence « null » produira une erreur d'exécution.

Par exemple, considérons le code suivant :

JAVA

```java
PWMSparkMax armMotorCtrl;

@Override
public void robotInit() {
      armMotorCtrl.setInverted(true);
}
```

C++

```cpp
class Robot : public frc::TimedRobot {
   public:
      void RobotInit() override {
         motorRef->SetInverted(false);
      }

   private:
      frc::PWMVictorSPX m_armMotor{0};
      frc::PWMVictorSPX* motorRef;
};
```

Lors de l'exécution, vous verrez une sortie qui ressemble à ceci :

Java

```
********** Robot program starting **********
Error at frc.robot.Robot.robotInit(Robot.java:23): Unhandled exception: java.lang.
↪NullPointerException
        at frc.robot.Robot.robotInit(Robot.java:23)
        at edu.wpi.first.wpilibj.TimedRobot.startCompetition(TimedRobot.java:107)
        at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:373)
        at edu.wpi.first.wpilibj.RobotBase.startRobot(RobotBase.java:463)
        at frc.robot.Main.main(Main.java:23)

Warning at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:388): The robot␣
↪program quit unexpectedly. This is usually due to a code error.
  The above stacktrace can help determine where the error occurred.
  See https://wpilib.org/stacktrace for more information.
Error at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:395): The␣
↪startCompetition() method (or methods called by it) should have handled the␣
↪exception above.
```

En lisant la trace de la pile, vous pouvez voir que le problème s'est produit à l'intérieur de la fonction `robotInit()`, à la ligne 23, et que l'exception impliquait « Null Pointer ».

En allant à la ligne 23, vous pouvez voir qu'il n'y a qu'une seule chose qui pourrait être nulle - `armMotorCtrl`. En regardant plus haut, vous pouvez voir que l'objet `armMotorCtrl` est déclaré, mais jamais instancié.

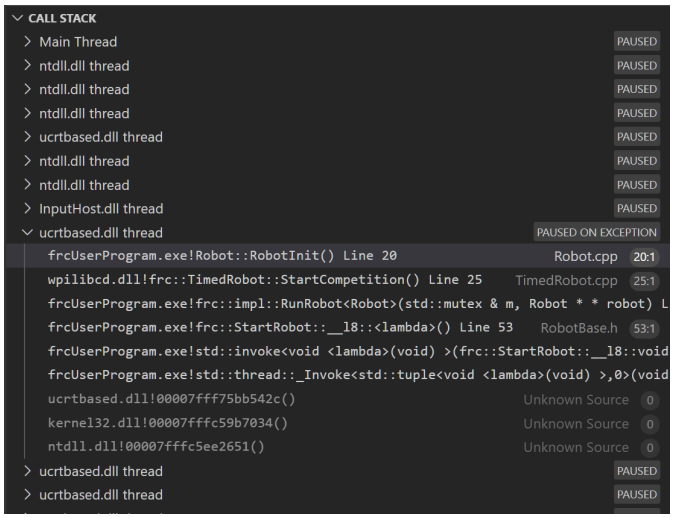
Alternativement, vous pouvez parcourir les lignes de code avec le débogueur en une seule étape et vous arrêter lorsque vous atteignez la ligne 23. L'inspection de l'objet `armMotorCtrl` à ce stade montrerait qu'il est nul.

C++

```
Exception has occurred: W32/0xc0000005
Unhandled exception thrown: read access violation.
this->motorRef was nullptr.
```

Dans Simulation, cela apparaîtra dans une fenêtre de débogueur qui pointe vers la ligne 20 dans le code bogué ci-dessus.

Vous pouvez afficher la trace complète de la pile en cliquant sur l'onglet du débogueur dans VS Code :



L'erreur est spécifique - notre variable membre `motorRef` a été déclarée, mais n'a jamais été affectée à une valeur. Par conséquent, lorsque nous essayons de l'utiliser pour appeler une méthode à l'aide de l'opérateur `->`, l'exception se produit.

L'exception indique que son type était `nullptr`.

Résoudre les problèmes d'objets nuls

En règle générale, vous voudrez vous assurer que chaque référence a été initialisée avant de l'utiliser. Dans ce cas, il manque une ligne de code pour instancier le `armMotorCtrl` avant d'appeler la méthode `setInverted()`.

Une implémentation fonctionnelle pourrait ressembler à ceci :

JAVA

```
19 PWMSparkMax armMotorCtrl;
20
21 @Override
22 public void robotInit() {
23       armMotorCtrl = new PWMSparkMax(0);
24       armMotorCtrl.setInverted(true);
25 }
```

C++

```
17 class Robot : public frc::TimedRobot {
18    public:
19       void RobotInit() override {
20          motorRef = &m_armMotor;
21          motorRef->SetInverted(false);
22       }
23
24    private:
```

(suite sur la page suivante)

(suite de la page précédente)

```
      frc::PWMVictorSPX m_armMotor{0};
      frc::PWMVictorSPX* motorRef;
};
```

Division par zéro

Il n'est généralement pas possible de diviser un nombre entier par zéro et d'attendre des résultats raisonnables. La plupart des processeurs (y compris le roboRIO) lèveront une exception non gérée.

Par exemple, considérons le code suivant :

JAVA

```
int armLengthRatio;
int elbowToWrist_in = 39;
int shoulderToElbow_in = 0; //TODO

@Override
public void robotInit() {
   armLengthRatio = elbowToWrist_in / shoulderToElbow_in;
}
```

C++

```
class Robot : public frc::TimedRobot {
   public:
   void RobotInit() override {
      armLengthRatio = elbowToWrist_in / shoulderToElbow_in;
   }

   private:
      int armLengthRatio;
      int elbowToWrist_in = 39;
      int shoulderToElbow_in = 0; //TODO

};
```

Lors de l'exécution, vous verrez une sortie qui ressemble à ceci :

Java

```
********** Robot program starting **********
Error at frc.robot.Robot.robotInit(Robot.java:24): Unhandled exception: java.lang.
↪ArithmeticException: / by zero
        at frc.robot.Robot.robotInit(Robot.java:24)
        at edu.wpi.first.wpilibj.TimedRobot.startCompetition(TimedRobot.java:107)
        at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:373)
        at edu.wpi.first.wpilibj.RobotBase.startRobot(RobotBase.java:463)
        at frc.robot.Main.main(Main.java:23)

Warning at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:388): The robot␣
↪program quit unexpectedly. This is usually due to a code error.
  The above stacktrace can help determine where the error occurred.
  See https://wpilib.org/stacktrace for more information.
Error at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:395): The␣
↪startCompetition() method (or methods called by it) should have handled the␣
↪exception above.
```

En regardant la trace de la pile, nous pouvons voir une exception `java.lang.ArithmeticException: / by zero` s'est produite à la ligne 24. Si vous regardez les deux variables qui sont utilisées à droite du `` =``, vous remarquerez peut-être que l'un d'entre eux a été initialisé à zéro. On dirait que quelqu'un a oublié de le mettre à jour ! De plus, la variable de valeur nulle est utilisée dans le dénominateur d'une opération de division. Par conséquent, l'erreur de division par zéro se produit.

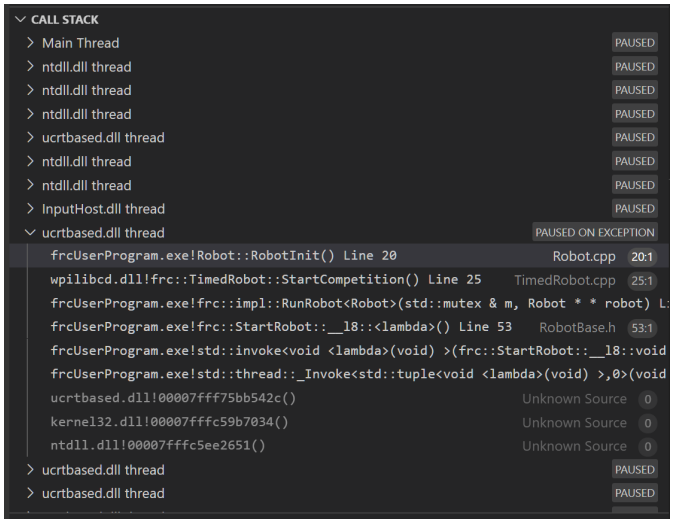
Alternativement, en exécutant le débogueur en une seule étape et en vous arrêtant à la ligne 24, vous pouvez inspecter la valeur de toutes les variables pour découvrir que `shoulderToElbow_in` a une valeur de `0`.

C++

```
Exception has occurred: W32/0xc0000094
Unhandled exception at 0x00007FF71B223CD6 in frcUserProgram.exe: 0xC0000094: Integer␣
↪division by zero.
```

Dans Simulation, cela apparaîtra dans une fenêtre de débogueur qui pointe vers la ligne 20 dans le code bogué ci-dessus.

Vous pouvez afficher la trace complète de la pile en cliquant sur l'onglet du débogueur dans VS Code :



En regardant le message, nous voyons que l'erreur est décrite comme `Integer division by zero`. Si vous regardez les deux variables qui sont utilisées à droite de l'opérateur = à la ligne 20, vous remarquerez peut-être que l'une d'entre elles a été initialisée à zéro. On dirait que quelqu'un a oublié de la mettre à jour ! De plus, la variable de valeur nulle est utilisée dans le dénominateur d'une opération de division. Par conséquent, l'erreur de division par zéro se produit.

Notez que les messages d'erreur peuvent être légèrement différents sur le roboRIO ou sur un système d'exploitation autre que Windows.

Résoudre les problèmes de division par zéro

Divide By Zero issues can be fixed in a number of ways. It's important to start by thinking about what a zero in the denominator of your calculation *means*. Is it plausible ? Why did it happen in the particular case you saw ?

Parfois, il vous suffit d'utiliser un nombre différent de 0.

Une implémentation fonctionnelle pourrait ressembler à ceci :

JAVA

```
18 int armLengthRatio;
19 int elbowToWrist_in = 39;
20 int shoulderToElbow_in = 3;
21
22 @Override
23 public void robotInit() {
24
25    armLengthRatio = elbowToWrist_in / shoulderToElbow_in;
26
27 }
```

C++

```
class Robot : public frc::TimedRobot {
   public:
   void RobotInit() override {
      armLengthRatio = elbowToWrist_in / shoulderToElbow_in;
   }

   private:
      int armLengthRatio;
      int elbowToWrist_in = 39;
      int shoulderToElbow_in = 3

};
```

Alternativement, si zéro *est* une valeur valide, l'ajout d'instructions `if/else` autour du calcul peut vous aider à définir un comportement alternatif pour éviter que le processeur effectue une division par zéro.

Enfin, changer les types de variables en `float` ou `double` peut vous aider à contourner le problème - les nombres à virgule flottante ont des valeurs spéciales comme `NaN` pour représenter les résultats d'une division par zéro opération. Cependant, vous devrez peut-être toujours gérer cela dans le code qui utilise la valeur de ce calcul.

Ressource HAL déjà allouée

A very common FRC-specific error occurs when the code attempts to put two hardware-related entities on the same HAL resource (usually, roboRIO IO pin).

Par exemple, considérons le code suivant :

JAVA

```
PWMSparkMax leftFrontMotor;
PWMSparkMax leftRearMotor;

@Override
public void robotInit() {
   leftFrontMotor = new PWMSparkMax(0);
   leftRearMotor = new PWMSparkMax(0);
}
```

C++

```
class Robot : public frc::TimedRobot {
   public:
      void RobotInit() override {
         m_frontLeftMotor.Set(0.5);
         m_rearLeftMotor.Set(0.25);
      }

   private:
```

(suite sur la page suivante)

(suite de la page précédente)

```
25      frc::PWMVictorSPX m_frontLeftMotor{0};
26      frc::PWMVictorSPX m_rearLeftMotor{0};
27
28    };
```

Lors de l'exécution, vous verrez une sortie qui ressemble à ceci :

Java

```
********** Robot program starting **********
Error at frc.robot.Robot.robotInit(Robot.java:25): Unhandled exception: edu.wpi.first.
↪hal.util.AllocationException: Code: -1029
PWM or DIO 0 previously allocated.
Location of the previous allocation:
        at frc.robot.Robot.robotInit(Robot.java:24)
        at edu.wpi.first.wpilibj.TimedRobot.startCompetition(TimedRobot.java:107)
        at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:373)
        at edu.wpi.first.wpilibj.RobotBase.startRobot(RobotBase.java:463)
        at frc.robot.Main.main(Main.java:23)

Location of the current allocation:
        at edu.wpi.first.hal.PWMJNI.initializePWMPort(Native Method)
        at edu.wpi.first.wpilibj.PWM.<init>(PWM.java:66)
        at edu.wpi.first.wpilibj.motorcontrol.PWMMotorController.<init>
↪(PWMMotorController.java:27)
        at edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax.<init>(PWMSparkMax.java:35)
        at frc.robot.Robot.robotInit(Robot.java:25)
        at edu.wpi.first.wpilibj.TimedRobot.startCompetition(TimedRobot.java:107)
        at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:373)
        at edu.wpi.first.wpilibj.RobotBase.startRobot(RobotBase.java:463)
        at frc.robot.Main.main(Main.java:23)

Warning at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:388): The robot␣
↪program quit unexpectedly. This is usually due to a code error.
  The above stacktrace can help determine where the error occurred.
  See https://wpilib.org/stacktrace for more information.
Error at edu.wpi.first.wpilibj.RobotBase.runRobot(RobotBase.java:395): The␣
↪startCompetition() method (or methods called by it) should have handled the␣
↪exception above.
```

Cette trace de pile montre qu'une `edu.wpi.first.hal.util.AllocationException` s'est produite. Il donne également le message utile : `PWM or DIO 0 previously allocated.` ou `PWM ou DIO préalablement alloué`.

Looking at our stack trace, we see two stack traces. The first stack trace shows that the first allocation occurred in `Robot.java:25`. The second stack trace shows that the error *actually* happened deep within WPILib. However, we should start by looking in our own code. Halfway through the stack trace, you can find a reference to the last line of the team's robot code that called into WPILib : `Robot.java:25`.

Taking a peek at the code, we see line 24 is where the first motor controller is declared and line 25 is where the second motor controller is declared. We can also note that *both* motor controllers are assigned to PWM output `0`. This doesn't make logical sense, and isn't physically possible. Therefore, WPILib purposely generates a custom error message and exception to alert the software developers of a non-achievable hardware configuration.

C++

En C++, vous ne verrez pas spécifiquement de trace de pile de ce problème. Au lieu de cela, vous recevrez des messages qui ressemblent à ce qui suit :

```
Error at PWM [C::31]: PWM or DIO 0 previously allocated.
Location of the previous allocation:
        at frc::PWM::PWM(int, bool) + 0x50 [0xb6f01b68]
        at frc::PWMMotorController::PWMMotorController(std::basic_string_view<char,␣
↪std::char_traits<char> >, int) + 0x70 [0xb6ef7d50]
        at frc::PWMVictorSPX::PWMVictorSPX(int) + 0x3c [0xb6e9af1c]
        at void frc::impl::RunRobot<Robot>(wpi::priority_mutex&, Robot**) + 0xa8␣
↪[0x13718]
        at int frc::StartRobot<Robot>() + 0x3d4 [0x13c9c]
        at __libc_start_main + 0x114 [0xb57ec580]

Location of the current allocation:: Channel 0
        at  + 0x5fb5c [0xb6e81b5c]
        at frc::PWM::PWM(int, bool) + 0x334 [0xb6f01e4c]
        at frc::PWMMotorController::PWMMotorController(std::basic_string_view<char,␣
↪std::char_traits<char> >, int) + 0x70 [0xb6ef7d50]
        at frc::PWMVictorSPX::PWMVictorSPX(int) + 0x3c [0xb6e9af1c]
        at void frc::impl::RunRobot<Robot>(wpi::priority_mutex&, Robot**) + 0xb4␣
↪[0x13724]
        at int frc::StartRobot<Robot>() + 0x3d4 [0x13c9c]
        at __libc_start_main + 0x114 [0xb57ec580]

Error at RunRobot: Error: The robot program quit unexpectedly. This is usually due to␣
↪a code error.
  The above stacktrace can help determine where the error occurred.
  See https://wpilib.org/stacktrace for more information.

        at void frc::impl::RunRobot<Robot>(wpi::priority_mutex&, Robot**) + 0x1c8␣
↪[0x13838]
        at int frc::StartRobot<Robot>() + 0x3d4 [0x13c9c]
        at __libc_start_main + 0x114 [0xb57ec580]

terminate called after throwing an instance of 'frc::RuntimeError'
  what():  PWM or DIO 0 previously allocated.
Location of the previous allocation:
        at frc::PWM::PWM(int, bool) + 0x50 [0xb6f01b68]
        at frc::PWMMotorController::PWMMotorController(std::basic_string_view<char,␣
↪std::char_traits<char> >, int) + 0x70 [0xb6ef7d50]
        at frc::PWMVictorSPX::PWMVictorSPX(int) + 0x3c [0xb6e9af1c]
        at void frc::impl::RunRobot<Robot>(wpi::priority_mutex&, Robot**) + 0xa8␣
↪[0x13718]
        at int frc::StartRobot<Robot>() + 0x3d4 [0x13c9c]
        at __libc_start_main + 0x114 [0xb57ec580]

Location of the current allocation:: Channel 0
```

La chose clé à noter ici est la chaîne de caractères,``PWM or DIO 0 previously allocated.``. Cette chaîne est votre principal indice que quelque chose dans le code a incorrectement « doublement assigné » l'utilisation de la broche 0.

L'exemple de message ci-dessus a été généré sur un roboRIO. Si vous l'exécutez en simulation, cela peut sembler différent.

Résoudre les problèmes de ressources HAL déjà allouées

`HAL : Resource already allocated` sont quelques-unes des erreurs les plus simples à corriger. Passez juste un peu de temps à regarder le câblage électrique du robot et comparez-le à ce qui est dans le code.

In the example, the left motor controllers are plugged into *PWM* ports `0` and `1`. Therefore, corrected code would look like this :

JAVA

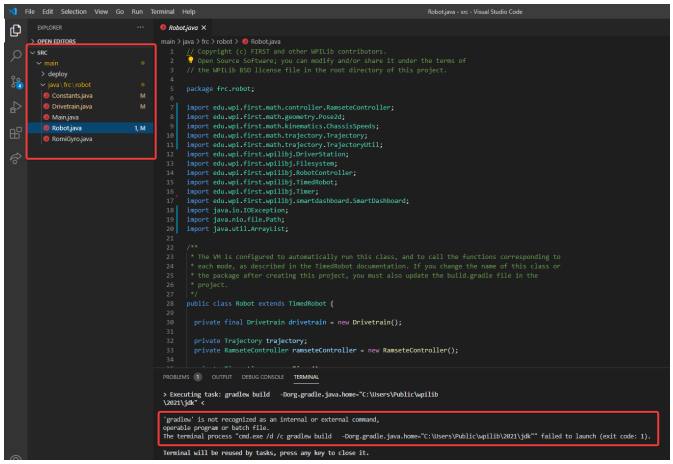
```
19 PWMSparkMax leftFrontMotor;
20 PWMSparkMax leftRearMotor;
21
22 @Override
23 public void robotInit() {
24
25    leftFrontMotor = new PWMSparkMax(0);
26    leftRearMotor = new PWMSparkMax(1);
27
28 }
```

C++

```
:lineno-start: 17

class Robot : public frc::TimedRobot {
   public:
      void RobotInit() override {
         m_frontLeftMotor.Set(0.5);
         m_rearLeftMotor.Set(0.25);
      }

   private:
      frc::PWMVictorSPX m_frontLeftMotor{0};
      frc::PWMVictorSPX m_rearLeftMotor{1};

   };
```

gradlew n'est pas reconnu...

`gradlew n'est pas reconnu comme une commande interne ou externe` est une erreur courante qui peut se produire lorsque le projet ou le répertoire dans lequel vous vous trouvez actuellement ne contient pas de fichier `gradlew`. Cela se produit généralement lorsque vous ouvrez le mauvais répertoire.



Dans la capture d'écran ci-dessus, vous pouvez voir que la barre latérale de gauche ne contient pas beaucoup de fichiers. Au minimum, VS Code a besoin de quelques fichiers pour créer et déployer correctement votre projet.

— `gradlew`
— `build.gradle`
— `gradlew.bat`

Si vous ne voyez aucun des fichiers ci-dessus dans votre répertoire de projet, alors vous avez deux causes possibles.

— Un projet corrompu ou mauvais.
— Vous êtes dans le mauvais répertoire.

La correction de gradlew n'est pas reconnue…

`gradlew n'est pas reconnu...` est un problème assez facile à résoudre. Identifiez d'abord la source du problème :

Êtes-vous dans le mauvais répertoire ? - Vérifiez que le répertoire du projet est le bon répertoire et ouvrez-le.

Votre projet manque-t-il des fichiers essentiels ? - Ce problème est plus complexe à résoudre. La solution recommandée est de *recréer votre projet* et de copier manuellement le code nécessaire.

17.9 Traiter les fonctions comme des données

Quel que soit le langage de programmation, l'une des premières choses que l'on apprend à faire lors de la programmation d'un ordinateur est d'écrire une fonction (également appelée « méthode » ou « sous-programme »). Les fonctions sont un élément fondamental du code organisé : l'écriture de fonctions nous permet d'éviter de dupliquer encore et encore le même morceau de code. Au lieu d'écrire des sections de code dupliquées, nous appelons une seule fonction qui contient le code que nous voulons exécuter à plusieurs endroits (à condition de bien nommer la fonction, le nom de la fonction est également plus facile à lire que le code lui-même !). Si la section de code a besoin d'informations supplémentaires sur son contexte environnant pour s'exécuter, nous les transmettons à la fonction en tant que « paramètres », et si elle doit renvoyer quelque chose au reste du code une fois terminé, nous appelons cela une « valeur de retour » (ensemble, les paramètres et la valeur de retour sont appelés la « signature » de la fonction) ;

Parfois, nous devons transmettre des fonctions d'une partie du code à une autre partie du code. Cela peut sembler un concept étrange si nous sommes habitués à considérer les fonctions comme faisant partie d'une définition de classe plutôt que comme des objets à part entière. Mais à la base, les fonctions ne sont que des données - de la même manière que nous pouvons stocker un « entier » ou un « double » comme variable et le transmettre à notre programme, nous pouvons faire la même chose avec une fonction. . Une variable dont la valeur est une fonction est appelée « interface fonctionnelle » en Java et « pointeur de fonction » ou « foncteur » en C++.

17.9.1 Pourquoi traiter les fonctions comme des données ?

En règle générale, le code qui appelle une fonction est couplé à (dépend de) la définition de la fonction. Bien que cela se produise tout le temps, cela devient problématique lorsque le code *appelant* la fonction (par exemple, WPILib) est développé indépendamment et sans connaissance directe du code qui *définit* la fonction (par exemple, le code d'une équipe FRC). Parfois, nous résolvons ce problème en utilisant des interfaces de classe, lesquelles définissent des collections de données et de fonctions destinées à être utilisées ensemble. Cependant, souvent, nous n'avons en réalité qu'une dépendance sur une *seule fonction*, plutôt que sur une *classe entière*.

Par exemple, WPILib propose aux utilisateurs plusieurs façons d'exécuter certains codes chaque fois qu'un bouton du joystick est enfoncé - l'un des moyens les plus simples et les plus propres de le faire est de permettre à l'utilisateur de *passer une fonction* à l'une des méthodes du joystick WPILib. De cette façon, l'utilisateur n'a qu'à écrire le code qui traite des événements intéressants et spécifiques à l'équipe (par exemple, « bouger mon bras de robot ») et non la chose ennuyeuse, sujette aux erreurs et universelle (« lire correctement les entrées des boutons depuis un joystick standard »).

Pour un autre exemple, le *framework basé sur des commandes* est construit sur des objets `Command` qui font référence aux méthodes définies sur diverses classes `Subsystem`. La plupart des types `Command` inclus (tels que `InstantCommand` et `RunCommand`) fonctionnent avec *n'importe quelle* fonction - pas seulement avec les fonctions associées à un seul `Subsystem`. Pour prendre en charge la construction de commandes de manière générique, nous devons prendre en charge le passage de fonctions d'un `Subsystem` (qui interagit avec le matériel) à une `Command` (qui interagit avec le planificateur).

Dans ces cas, nous voulons pouvoir transmettre une seule fonction sous forme de données, comme s'il s'agissait d'une variable - cela n'a pas de sens de demander à l'utilisateur de fournir une classe entière, alors que nous voulons simplement qu'il fournisse une seule fonction de conception appropriée.

Il est important de comprendre que *passer* une fonction n'est pas la même chose que *appeler* une fonction. Lorsque nous appelons une fonction, nous exécutons le code qu'elle contient et soit nous recevons une valeur de retour, soit nous provoquons des effets secondaires ailleurs dans le code, soit les deux. Lorsque nous *passons* une fonction, rien de particulier ne se produit *immédiatement*. Au lieu de cela, en passant la fonction, nous permettons à un *autre* code d'appeler la fonction *dans le futur*. Voir le nom d'une fonction dans le code ne signifie pas toujours que le code de la fonction est en cours d'exécution !

À l'intérieur du code qui transmet une fonction, nous verrons une syntaxe qui soit fait référence au nom d'une fonction existante d'une manière spéciale, soit définit une nouvelle fonction à transmettre à l'intérieur de l'expression d'appel. La syntaxe spécifique nécessaire (et les règles qui l'entourent) dépend du langage de programmation que nous utilisons.

17.9.2 Traiter les fonctions comme des données en Java

Java représente les fonctions en tant que données comme des instances d"interfaces fonctionnelles. Une « interface fonctionnelle » est un type particulier de classe qui n'a qu'une seule méthode. Puisque Java a été initialement conçu strictement pour la programmation orientée objet, il n'a aucun moyen de représenter une seule fonction détachée d'une classe. Au lieu de cela, il définit un groupe particulier de classes qui représentent *seulement* des fonctions uniques. Chaque type de signature de fonction possède sa propre interface fonctionnelle, qui est une interface avec une définition de fonction unique de cette signature.

Cela peut sembler compliqué, mais dans le contexte de WPILib, nous n'avons pas vraiment besoin de nous soucier de l'utilisation des interfaces fonctionnelles elles-mêmes : le code qui fait cela est interne à WPILib. Au lieu de cela, tout ce que nous devons savoir, c'est comment transmettre une fonction que nous avons écrite à une méthode qui prend une interface fonctionnelle comme paramètre. Pour un exemple simple, considérons la signature de `Commands.runOnce` (qui crée une `InstantCommand` qui, lorsqu'elle est planifiée, exécute la fonction donnée une seule fois puis se termine) :

Note : Le paramètre `requirements` est expliqué dans la *Documentation basée sur les commandes*, et ne sera pas abordé ici.

```
public static Command runOnce(Runnable action, Subsystem... requirements)
```

`runOnce` attend que nous lui donnions un paramètre `Runnable` (nommé `action`). Un `Runnable` est le terme Java désignant une fonction qui ne prend aucun paramètre et ne renvoie aucune valeur. Lorsque nous appelons `runOnce`, nous devons lui donner une fonction sans paramètres ni valeur de retour. Il existe deux manières de procéder : nous pouvons faire référence à une fonction existante en utilisant une « référence de méthode », ou nous pouvons définir la fonction souhaitée en ligne à l'aide d'une « expression lambda ».

Références de méthodes

Une référence de méthode nous permet de passer une fonction déjà existante comme notre `Runnable` :

```
// Create an InstantCommand that runs the `resetEncoders` method of the `drivetrain`␣
↪object
Command disableCommand = runOnce(drivetrain::resetEncoders, drivetrain);
```

L'expression `drivetrain::resetEncoders` est une référence à la méthode `resetEncoders` de l'objet `drivetrain`. Ce n'est pas un *appel* de méthode - cette ligne de code ne réinitialise pas *elle-même* les encodeurs de la transmission. Au lieu de cela, elle renvoie une ``Command`` qui le fera *quand cela est planifié.*

Rappelez-vous que pour que cela fonctionne, `resetEncoders` doit être un `Runnable` - c'est-à-dire qu'il ne doit prendre aucun paramètre et ne renvoyer aucune valeur. Ainsi, sa signature doit ressembler à ceci :

```
// void because it returns no parameters, and has an empty parameter list
public void resetEncoders()
```

Si la signature de la fonction ne correspond pas à ceci, Java ne pourra pas interpréter la référence de méthode comme un `Runnable` et le code ne sera pas compilé. Notez que tout

ce que nous devons faire est de nous assurer que la signature correspond à la signature de la méthode unique dans l'interface fonctionnelle `Runnable` - nous n'avons pas besoin de la nommer *explicitement* comme `Runnable`.

Expressions Lambda en Java

Si nous n'avons pas déjà une fonction nommée qui fait ce que nous voulons, nous pouvons définir une fonction « inline » - c'est-à-dire juste à l'intérieur de l'appel à `runOnce` ! Nous faisons cela en écrivant notre fonction avec une syntaxe spéciale qui utilise un symbole « flèche » pour lier la liste d'arguments au corps de la fonction :

```
// Create an InstantCommand that runs the drive forward at half speed
Command driveHalfSpeed = runOnce(() -> { drivetrain.arcadeDrive(0.5, 0.0); },␣
↪drivetrain);
```

Java appelle `() -> { drivetrain.arcadeDrive(0.5, 0.0); }` une « expression lambda » ; elle peut être appelée de manière moins déroutante « fonction flèche », « fonction inline » ou « fonction anonyme » (car elle n'a pas de nom). Bien que cela puisse paraître un peu étrange, il s'agit simplement d'une autre façon d'écrire une fonction : les parenthèses avant la flèche sont la liste des arguments de la fonction, et le code contenu entre parenthèses est le corps de la fonction. L'expression « lambda » représente ici une fonction qui appelle `drivetrain.arcadeDrive` avec un ensemble spécifique de paramètres - notez encore que cela n"*appelle pas* la fonction, mais la définit simplement et la transmet à la `Command` pour être exécuté plus tard lorsque la `Command` est planifiée.

Comme pour les références de méthode, nous n'avons pas besoin de nommer *explicitement* l'expression lambda comme `Runnable` - Java peut déduire que notre expression lambda est un `Runnable` tant que sa signature correspond à celle de la méthode unique. dans l'interface `Runnable`. En conséquence, notre lambda ne prend aucun argument et n'a pas d'instruction return - s'il ne correspondait pas au contrat `Runnable`, notre code ne pourrait pas être compilé.

Capture de l'état dans les expressions lambda en Java

Dans l'exemple ci-dessus, le corps de notre fonction fait référence à un objet qui vit en dehors de la fonction elle-même (à savoir, l'objet `drivetrain`). C'est ce qu'on appelle une « capture » d'une variable à partir du code environnant (qui est parfois appelé « portée externe » ou « portée englobante »). Habituellement, les variables capturées sont soit des variables locales du corps de la méthode englobante dans laquelle l'expression lambda est définie, soit des champs d'une définition de classe englobante dans laquelle cette méthode est définie.

En Java, la capture d'un état est une chose assez sûre à faire en général, avec toutefois une mise en garde majeure : nous ne pouvons capturer que l'état « effectivement final ». Cela signifie qu'il n'est légal de capturer une variable de la portée englobante que si cette variable n'est jamais réaffectée après l'initialisation. Notez que cela ne signifie pas que l'état capturé ne peut pas changer : rappelez-vous que les objets Java sont des références, donc l'objet vers lequel la référence *pointe* peut changer après la capture - mais la référence elle-même ne peut pas pointer vers un autre objet.

Cela signifie que nous ne pouvons capturer les types primitifs (comme `int`, `double` et `boolean`) que s'ils sont des constantes. Si nous voulons capturer une variable d'état qui peut changer, elle *doit être enveloppée dans un objet mutable*.

Sucre syntaxique pour les expressions lambda en Java

La syntaxe complète de l'expression lambda peut être inutilement verbeuse dans certains cas. Pour nous aider, Java nous permet de prendre quelques raccourcis (appelés « sucre syntaxique ») dans les cas où une partie de la notation est redondante.

Omission des crochets de fonction pour les lambdas à une ligne

Si le corps de la fonction de notre expression lambda ne comporte qu'une seule ligne, Java nous permet d'omettre les crochets autour du corps de la fonction. Lorsque nous omettons les parenthèses de fonction, nous omettons également les points-virgules de fin de ligne et le mot-clé *return*.

Ainsi, notre lambda `Runnable` ci-dessus pourrait à la place être écrit :

```
// Create an InstantCommand that runs the drive forward at half speed
Command driveHalfSpeed = runOnce(() -> drivetrain.arcadeDrive(0.5, 0.0), drivetrain);
```

Omission des parenthèses autour des paramètres lambda uniques

Si l'expression lambda est destinée à une interface fonctionnelle qui ne prend qu'un seul argument, nous pouvons omettre la parenthèse autour de la liste des paramètres :

```
// We can write this lambda with no parenthesis around its single argument
IntConsumer exampleLambda = (a -> System.out.println(a));
```

17.9.3 Traiter les fonctions comme des données en C++

C++ propose plusieurs façons de traiter les fonctions comme des données. Dans le cadre de cet article, nous ne parlerons que des parties pertinentes pour l'utilisation de WPILibC.

Dans WPILibC, les types de fonctions sont représentés avec la classe `std::function` (https://en.cppreference.com/w/cpp/utility/function/function). Cette classe de bibliothèque standard est calquée sur la signature de la fonction - cela signifie que nous devons lui fournir un type de fonction comme paramètre de modèle pour spécifier la signature de la fonction (comparez cela à *Java* ci-dessus, où nous avons un type d'interface distinct pour chaque type de signature).

Cela semble beaucoup plus compliqué que son utilisation en pratique. Regardons la signature d'appel de `cmd::RunOnce` (qui crée une `InstantCommand` qui, lorsqu'elle est planifiée, exécute la fonction donnée une seule fois puis se termine) :

Note : Le paramètre `requirements` est expliqué dans la *Documentation basée sur les commandes*, et ne sera pas abordé ici.

```
CommandPtr RunOnce(
 std::function<void()> action,
 Requirements requirements);
```

`runOnce` attend que nous lui donnions un paramètre `std::function<void()>` (nommé `action`). Un `std::function<void()>` est le type C++ pour une `std::function` qui ne prend

aucun paramètre et ne renvoie aucune valeur (le paramètre du modèle, `void()` est un type de fonction sans paramètres ni valeur de retour). Lorsque nous appelons `runOnce`, nous devons lui donner une fonction sans paramètres ni valeur de retour. C++ ne dispose pas d'un moyen simple de faire référence aux méthodes de classe existantes d'une manière qui puisse être automatiquement convertie en `std::function`, donc la manière typique de le faire est de définir une nouvelle fonction en ligne avec une « expression lambda ».

Expressions lambda en C++

Pour passer une fonction à `runOnce`, nous devons écrire une courte expression de fonction en ligne en utilisant une syntaxe spéciale qui ressemble aux déclarations de fonctions C++ ordinaires, mais qui varie de plusieurs manières importantes :

```
// Create an InstantCommand that runs the drive forward at half speed
CommandPtr driveHalfSpeed = cmd::RunOnce([this] { drivetrain.ArcadeDrive(0.5, 0.0); },
↪ {drivetrain});
```

C++ appelle `[captures] (params) { body; }` une « expression lambda ». Elle comporte trois parties : une *liste de capture* (crochets), une *liste de paramètres* facultative (parenthèses) et un *corps de fonction* (accolades). Cela peut paraître un peu étrange, mais la seule vraie différence entre une expression lambda et une fonction ordinaire (mis à part l'absence de nom de fonction) est l'ajout de la liste de capture.

Puisque `RunOnce` veut une fonction sans paramètres ni valeur de retour, notre expression lambda n'a pas de liste de paramètres ni d'instruction de retour. L'expression « lambda » représente ici une fonction qui appelle `drivetrain.ArcadeDrive` avec un ensemble spécifique de paramètres - notez encore que le code ci-dessus n"*appelle pas* la fonction, mais la définit simplement et la transmet à la `` Command`` à exécuter plus tard lorsque la `Command` est planifiée.

Capture de l'état dans les expressions lambda en C++

Dans l'exemple ci-dessus, le corps de notre fonction fait référence à un objet qui vit en dehors de la fonction elle-même (à savoir, l'objet `drivetrain`). C'est ce qu'on appelle une « capture » d'une variable à partir du code environnant (qui est parfois appelé « portée externe » ou « portée englobante »). Habituellement, les variables capturées sont soit des variables locales du corps de la méthode englobante dans laquelle l'expression lambda est définie, soit des champs d'une définition de classe englobante dans laquelle cette méthode est définie.

C++ a une sémantique un peu plus puissante que Java. L'un des coûts de cela est que nous devons généralement aider le compilateur C++ pour déterminer *comment exactement* nous voulons qu'il capture l'état de la portée englobante. C'est le but de la *liste de capture*. Pour utiliser le framework basé sur les commandes WPILibC, il suffit généralement d'utiliser une liste de capture de `[this]`, qui donne accès aux membres de la classe englobante en capturant le pointeur `this` de la classe englobante par valeur.

Les valeurs locales de méthodes ne peuvent pas être capturées avec le pointeur `this` et doivent être capturées explicitement soit par référence, soit par valeur en les incluant dans la liste de capture (ou implicitement en spécifiant à la place une sémantique de capture par défaut). Il est généralement plus sûr de capturer les valeurs locales par valeur, car une lambda peut survivre à la durée de vie d'un objet qu'elle capture par référence. Pour plus de détails, consultez la documentation de la bibliothèque standard C++ sur la sémantique de capture.

17.10 Obtenir la couleur d'alliance

La classe `DriverStation` (Java, C++, Python) possède de nombreuses fonctionnalités utiles pour obtenir des données à partir de l'ordinateur de la station de pilotage. L'une des fonctionnalités les plus importantes est `getAlliance` (Java & Python) / `GetAlliance` (C++).

Note that there are three cases : red, blue, and no color yet. It is important that code handles the third case correctly because the alliance color will not be available until the Driver Station connects. In particular, code should not assume that the alliance color will be available during constructor methods or *robotInit*, but it should be available by the time *autoInit* or *teleopInit* is called. FMS will set the alliance color automatically ; when not connected to FMS, the alliance color can be set from the Driver Station (see *« Team Station » on the Operation Tab*).

17.10.1 Obtenir votre couleur d'alliance et effectuer une action

JAVA

```
Optional<Alliance> ally = DriverStation.getAlliance();
if (ally.isPresent()) {
    if (ally.get() == Alliance.Red) {
        <RED ACTION>
    }
    if (ally.get() == Alliance.Blue) {
        <BLUE ACTION>
    }
}
else {
    <NO COLOR YET ACTION>
}
```

C++

```
using frc::DriverStation::Alliance;
if (auto ally = frc::DriverStation::GetAlliance()) {
    if (ally.value() == Alliance::kRed) {
        <RED ACTION>
    }
    if (ally.value() == Alliance::kBlue) {
        <BLUE ACTION>
    }
}
else {
    <NO COLOR YET ACTION>
}
```

PYTHON

```
from wpilib import DriverStation

ally = DriverStation.getAlliance()
if ally is not None:
    if ally == DriverStation.Alliance.kRed:
        <RED ACTION>
    elif ally == DriverStation.Alliance.kBlue:
        <BLUE ACTION>
else:
    <NO COLOR YET ACTION>
```

17.11 Java Garbage Collection

Java garbage collection is the process of automatically managing memory for Java objects. The Java Virtual Machine (JVM) is responsible for creating and destroying objects, and the garbage collector is responsible for identifying and reclaiming unused objects.

Java garbage collection is an automatic process, which means that the programmer does not need to explicitly deallocate memory. The garbage collector keeps track of which objects are in use and which are not, and it periodically reclaims unused objects.

17.11.1 Object Creation

Creating a large number of objects in Java can lead to memory and performance issues. While the Java Garbage Collector (GC) is designed to handle memory management efficiently, creating too many objects can overwhelm the GC and cause performance degradation.

Memory Concerns

When a large number of objects are created, it increases the overall memory footprint of the application. While the overhead for a single object may be insignificant, it can become substantial when multiplied by a large number of objects.

Note : *VisualVM* can be used to see where memory is allocated.

Performance Concerns

The GC's job is to periodically identify and reclaim unused objects in memory. While garbage collection is running on an FRC robot coded in Java, execution of the robot program is paused. When the GC has to collect a large number of objects, it has to pause the application to run more frequently or for longer periods of time. This is because the GC has to perform more work to collect and process each object.

GC-related performance degradation in robot programs can manifest as occasional pauses, freezes, or loop overruns as the GC works to reclaim memory.

Design Considerations

If you anticipate your application creating a large number of short-lived objects, it is important to consider design strategies to mitigate the potential memory and performance issues. Here are some strategies to consider :

— Minimize object creation : Carefully evaluate the need for each object creation. If possible, reuse existing objects or use alternative data structures, such as arrays or primitives, to avoid creating new objects.
— Efficient data structures : Use data structures that are well-suited for the type of data you are working with. For example, if you are dealing with a large number of primitive values, consider using arrays or collections specifically designed for primitives.

17.11.2 Diagnosing Out of Memory Errors with Heap Dumps

All objects in Java are retained in a section of memory called the *heap*. As objects typically consume the greatest amount of memory in a Java program, it is often useful to take a snapshot of the state of the heap—a heap dump—to analyze memory issues. Heap dumps only capture the state of a program's heap at a single point in time, so they are unlikely to be useful if not captured exactly at the time the program is experiencing memory issues.

Since `OutOfMemoryErrors` both crash the program and are a common reason to want a heap dump, the JVM can be configured to automatically take a heap dump the moment an `OutOfMemoryError` is caught by the JVM. To configure these options, locate the `frcJava` code block in your project's `build.gradle` :

```
deploy {
    targets {
        roborio(getTargetTypeClass('RoboRIO')) {
            // Team number is loaded either from the .wpilib/wpilib_preferences.json
            // or from command line. If not found an exception will be thrown.
            // You can use getTeamOrDefault(team) instead of getTeamNumber if you
            // want to store a team number in this file.
            team = project.frc.getTeamNumber()
            debug = project.frc.getDebugOrDefault(false)

            artifacts {
                // First part is artifact name, 2nd is artifact type
                // getTargetTypeClass is a shortcut to get the class type using a string

                frcJava(getArtifactTypeClass('FRCJavaArtifact')) {
                }

                // Static files artifact
                frcStaticFileDeploy(getArtifactTypeClass('FileTreeArtifact')) {
                    files = project.fileTree('src/main/deploy')
                    directory = '/home/lvuser/deploy'
                }
            }
        }
    }
}
```

Add to the code block so that it contains two `jvmArgs` commands, as shown below :

```
frcJava(getArtifactTypeClass('FRCJavaArtifact')) {
    // If you have other configuration here, you do not need to remove it.
    // Enable automatic heap dumps on OutOfMemoryError
    // Note: the heap dump path here is a path on a USB flash drive, see below
    jvmArgs.add("-XX:+HeapDumpOnOutOfMemoryError")
    jvmArgs.add("-XX:HeapDumpPath=/u/frc-usercode.hprof")
}
```

This will cause the JVM to write heap dumps to a file named `frc-usercode.hprof` at the root of a USB flash drive attached to the roboRIO when the code runs out of memory. It is recommended to save these heap dumps to a USB flash drive because heap dumps intrinsically consume the same amount of space on disk as the program heap did in memory when the program crashed, and are likely to be larger than the roboRIO's internal storage has capacity for. Once you have reproduced the `OutOfMemoryError`, redeploy your code without these options enabled, and use the USB flash drive to transfer the heap dump to a computer for analysis in a memory profiler such as *VisualVM*.

Avertissement : Configuring the JVM this way requires that the flash drive remain connected to the roboRIO while your code is running.

Larger SD cards may provide enough onboard storage to allow the use of these options on the roboRIO 2 without a USB flash drive. To do this, set the `-XX:HeapDumpPath` option to reference a path on the SD card, and use *FTP/SFTP to transfer the heap dump to a computer* before deleting it from the SD card.

Note that the JVM will **not** overwrite heap dumps with the exact path and filename specified by `-XX:HeapDumpPath` if they already exist, nor will it dump the process heap to a file with a different name. If a path to a directory is supplied instead of a path to a file, the JVM will instead write out heap dumps with unique filenames within the specified directory, with the name `java_pidNNNN.hprof`, where `NNNN` is the process ID of the JVM that ran out of memory. Note that this can cause large files to build up on disk if they are not cleaned out, so if you configure the JVM this way, be sure to frequently copy heap dumps to a computer and delete them from the flash drive/SD card afterward.

Prudence : Always be vigilant about the amount of available space on the underlying storage medium while you use this feature.

Use of this feature is not recommended during competitive play.

17.11.3 System Memory Tuning

If the JVM cannot allocate memory, the program will be terminated. As an embedded system with only a small amount of memory available (256 MB on the roboRIO 1, 512 MB on the roboRIO 2), the roboRIO is particularly susceptible to running out of memory.

No amount of system tuning can fix out of memory errors caused by out-of-control allocations.

If you are running out of memory, always investigate allocations with heap dumps and/or *VisualVM* first.

If you continue to run out of memory even after investigating with VisualVM and taking steps to minimize the number of allocated objects, a few different options are available to make additional memory available to the robot program.

- Disabling the system web server
- Setting sysctls (Linux kernel options)
- Periodically calling the garbage collector
- Setting up swap on a USB flash drive

Implementing most of these options require *connecting with SSH* to the roboRIO and running commands. If run incorrectly, it may require a reimage to recover, so be careful when following the instructions.

Disabling the System Web Server

The built-in NI system web server provides the webpage (the *roboRIO Web Dashboard*) seen when using a web browser to connect to the roboRIO, e.g. to change IP address settings. It also is used by the Driver Station's data log download functionality. However, it consumes several MB of RAM, so disabling it will free up that memory for the robot program to use. There are several ways to disable the web server :

The first and easiest is to use the *RoboRIO Team Number Setter* tool. Versions 2024.2.1 and later of the tool have a button to disable or enable the web server. However, a few teams have reported that this does not work or does not persist between reboots. There are two alternate ways to disable the web server ; both require connecting to the roboRIO with SSH and logging in as the `admin` user.

1. Run `/etc/init.d/systemWebServer stop; update-rc.d -f systemWebServer remove; sync`
2. Run `chmod a-x /usr/local/natinst/etc/init.d/systemWebServer; sync`

To revert the alternate ways and re-enable the web server, take the corresponding step :

1. Run `update-rc.d -f systemWebServer defaults; /etc/init.d/systemWebServer start; sync`
2. Run `chmod a+x /usr/local/natinst/etc/init.d/systemWebServer; sync`

Setting sysctls

Several Linux kernel options (called sysctls) can be set to tweak how the kernel allocates memory. Several options have been found to reduce out-of-memory errors :

- Setting `vm.overcommit_memory` to 1 (the default value is 2). This causes the kernel to always pretend there is enough memory for a requested memory allocation at the time of allocation ; the default setting always checks to see if there's actually enough memory to back an allocation at the time of allocation, not when the memory is actually used.
- Setting `vm.vfs_cache_pressure` to 1000 (the default value is 100). Increasing this causes the kernel to much more aggressively reclaim file system object caches ; it may slightly degrade performance.
- Setting `vm.swappiness` to 100 (the default value is 60). This causes the kernel to more aggressively swap process memory to the swap file. Changing this option has no effect unless you add a swap file.

You can set some or all of these options ; the most important one is `vm.overcommit_memory`. Setting these options requires connecting to the roboRIO with SSH and logging in as the `admin` user, then running the following commands :

```
echo "vm.overcommit_memory=1" >> /etc/sysctl.conf
echo "vm.vfs_cache_pressure=1000" >> /etc/sysctl.conf
echo "vm.swappiness=100" >> /etc/sysctl.conf
sync
```

The `/etc/sysctl.conf` file should contain the following lines at the end when done (to check, you can run the command `cat /etc/sysctl.conf`) :

```
vm.overcommit_memory=1
vm.vfs_cache_pressure=1000
vm.swappiness=100
```

To revert the change, edit `/etc/sysctl.conf` (this will require the use of the vi editor) and remove these 3 lines.

Periodically Calling the Garbage Collector

Sometimes the garbage collector won't run frequently enough to keep up with the quantity of allocations. As Java provides a way to trigger a garbage collection to occur, running it on a periodic basis may reduce peak memory usage. This can be done by adding a `Timer` and a periodic check :

```
Timer m_gcTimer = new Timer();

public void robotInit() {
  m_gcTimer.start();
}

public void periodic() {
  // run the garbage collector every 5 seconds
  if (m_gcTimer.advanceIfElapsed(5)) {
    System.gc();
  }
}
```

Setting Up Swap on a USB Flash Drive

A swap file on a Linux system provides disk-backed space that can be used by the system as additional virtual memory to put infrequently used data and programs when they aren't being used, freeing up physical RAM for active use such as the robot program. It is strongly recommended to not use the built-in non-replaceable flash storage on the roboRIO 1 for a swap file, as it has very limited write cycles and may wear out quickly. Instead, however, a FAT32-formatted USB flash drive may be used for this purpose. This does require the USB flash drive to always be plugged into the roboRIO before boot.

Prudence : Having a swap file on a USB stick means it's critical the USB stick stay connected to the roboRIO at all times it is powered.

This should be used as a last resort if none of the other steps above help. Generally needing swap is indicative of some other allocation issue, so use VisualVM first to optimize allocations.

A swap file can be set up by plugging the USB flash drive into the roboRIO USB port, connecting to the roboRIO with SSH and logging in as the `admin` user, and running the following commands. Note the vi step requires knowledge of how to edit and save a file in vi.

```
fallocate -l 100M /u/swapfile
mkswap /u/swapfile
swapon /u/swapfile
vi /etc/init.d/addswap.sh
chmod a+x /etc/init.d/addswap.sh
update-rc.d -v addswap.sh defaults
sync
```

The `/etc/init.d/addswap.sh` file contents should look like this :

```
#!/bin/sh
[ -x /sbin/swapon ] && swapon -e /u/swapfile
: exit 0
```

To revert the change, run `update-rc.d -f addswap.sh remove; rm /etc/init.d/addswap.sh; sync; reboot`.

18

Ressources d’assistance

En plus de la documentation fournie, FRC® vous propose plusieurs autres ressources pour vous aider à comprendre le système de contrôle et le logiciel.

18.1 Les autres documentations

En plus de ce site, il existe quelques autres endroits où les équipes peuvent obtenir de la documentation :

- Section des documents de la communauté NI FRC
- FIRST Inspires Technical Resources Page
- CTRE Software & Resources Page
- REV Robotics Documentation

18.2 Les forums

Vous avez une question à laquelle la documentation n’a pas répondue ? Un support officiel est fourni sur ces forums :

- NI FRC Support Forum (roboRIO, LabVIEW and Driver Station software questions, as well as roboRIO repairs)
- FIRST Inspires Control System Forum (câblage, questions relatives aux dispositifs électronique et la Driver Station)
- FIRST Inspires Programming Forum (questions de programmation pour C ++, Java, ou LabVIEW)

18.3 Assistance CTRE

Support for Cross The Road Electronics components (Pneumatics Control Module, Power Distribution Panel, Talon SRX, and Voltage Regulator Module) is provided via the email address support@crosstheroadelectronics.com.

18.4 Assistance REV Robotics

Support for REV Robotics components (SPARK MAX, Sensors, Pneumatic Hub, Power Distribution Hub, Radio Power Module) is provided via phone at 844-255-2267 or via the email address support@revrobotics.com.

18.5 Autres fournisseurs

Le support pour les fournisseurs en dehors du KOP peut être trouvé ci-dessous.

- Copperforge
- Kauai Labs (NavX)
- Limelight
- PhotonVision (Discord)
- Playing with Fusion

18.6 Soutien non officiel

There are useful forms of support provided by the community through various forums and services. The below links and websites are not endorsed by FIRST® and may be used at your own risk.

- Chief Delphi
- FRC Discord

18.7 Rapport de bogue

Found a bug ? Let us know by reporting it in the Issues section of the appropriate WPILibSuite project on GitHub : https://github.com/wpilibsuite

19 Glossaire FRC

accéléromètre
: Un capteur commun utilisé pour mesurer l'accélération dans un ou plusieurs axes.

AM
: AndyMark, Inc - strives to develop innovative products and outstanding service while inspiring our customers and making a positive impact in our community.

AprilTags
: Visual tags that provide low overhead, high accuracy localization. AprilTags are useful for helping your robot know where it is at on the field, so it can align itself to some goal position.

auto
: The first phase of each match is called Autonomous (auto) and consists of the robot's running pre-programmed instructions.

back-EMF
: In electric motors, the force generated by the interaction of spinning magnets in a coil of wire which opposes spinning motion.

boolean
: A form of data with only two possible values (true or false), intended to represent the two truth values of logic and Boolean algebra.

call stack
: A specially-organized region of memory which helps the program keep track of what function it is in. As each function calls another, the call point is recorded and added to the top of the structure, forming a « stack » of references. Additionally, local variables will also be stored in this stack. See call stack on Wikipedia for more info.

CAD
: Computer-Aided Design - software used to design an accurate model of an object. For FRC this is often used to design the robot to get accurate measurements and aid construction.

CAM
: Computer-Aided Manufacturing - the use of software to control machine tools in the manufacturing of work pieces.

CAN
: Controller Area Network - message-based protocol designed to allow microcontrollers and devices to communicate with each other's applications without a host computer.

CD
Chief Delphi - FRC team 47 inspired a popular community driven forum that today serves as an unofficial discussion hub for all things FRC.

central limit theorem
A core concept in probability which states that when many independent variables are added up, the result tends to look like a « normal » (or Gaussian) distribution, regardless of whether the independent variables themselves are normally distributed. See Central Limit Theorem on Wikipedia for more info.

CIM
CCL Industrial Motor, Limited - Chiaphua Components Limited is the company that made the commonly used, relatively powerful, brushed motor.

Classical Mechanics
The branch of physics which studies and describes the motion of relatively large, relatively slow objects. See Classical Mechanics on Wikipedia for more info.

COTS
Commercial off the shelf - a standard (i.e. not custom order) part commonly available from a vendor to all teams for purchase.

composition
A formal software term for building (or « composing ») software entities out of smaller component entities. See object composition on Wikipedia for more info.

CRTP
Continuously Recurring Template Pattern - A software idiom in which a class $X`$ derives from a class template instantiation using $X`$ itself as a template argument. See CRTP on Wikipedia for more info.

CSA
Control Systems Advisor - FRC volunteer position that assists teams with Robot Control System-related issues.

CTRE
Cross the Road Electronics LLC - is an engineering design, software development and electronics manufacturer based outside of Detroit in Macomb, MI. They primarily focus on high-performing, high-quality electronics communication, motor control, and control system products for FIRST teams and the EV industry. CTR Electronics was founded in 2006 by two FRC mentors who met through their robotics team : Mike Copioli and Omar Zrien and is staffed largely by FRC alumni, and active volunteers & mentors.

C++
One of the four officially supported programming languages.

declarative programming
A style of software which focuses on describing *what* a program should do, rather than *how* it gets done. See declarative programming on Wikipedia for more info.

dependency injection
A software design pattern where each class receives all objects it depends upon. Sometimes these are passed through the constructor, but not always. See dependency injection on Wikipedia for more info.

obsolète
Software that has been replaced and will no longer receive new features. Deprecated software will be maintained for at least 1 year, but may be removed after that. For example, if a method is deprecated prior to the 2022 season, it will be usable in the 2022 season, but may be removed prior to the 2023 season. Teams are encouraged to not use deprecated methods in new code. WPILib always deprecates features at least one year prior to removing them from the codebase.

design pattern
A particular, intentionally-chosen style of organizing code. A design pattern intentionally excludes using certain features of a programming language to constrain developers into solutions that are well-suited to a particular problem-space. See design pattern. on Wikipedia for more info.

DHCP
Dynamic Host Configuration Protocol - the protocol that allows a central device to assign unique IP addresses to all other devices.

encapsulation
A software design pattern which uses a class to hide the implementation details of other classes. See encapsulation on Wikipedia for more info.

entry
In *NetworkTables*, a combined *publisher* and *subscriber*. The subscriber is always active, but the publisher is not created until a publish operation is performed (e.g. a value is « set », aka published, on the entry). This may be more convenient than maintaining a separate publisher and subscriber.

enumeration
A list of all elements of a set, typically used to refer to a set of pre-defined values.

EPA
Expected Points Added - builds upon the Elo rating system, but transforms ratings to point units and makes several modifications.

event-driven programming
A style of programming where certain parts of code generate « events » as a result of some input (sensors, user interaction, etc). Then, other parts of code listen for and respond to « handle » these events. See event-based on Wikipedia for more info.

FIRST
For Inspiration and Recognition of Science and Technology - a global nonprofit organization that prepares young people for the future through a suite of life-changing youth robotics programs that build skills, confidence, and resilience.

FLL
FIRST Lego League - Introduces science, technology, engineering, and math (STEM) to children ages 4-16 through fun, exciting hands-on learning.

floating point
A method for approximating real numbers in computer-based arithmetic, using a fixed precision integer scaled by an integer exponent. Typically computer systems support both « single » precision (32-bit storage) and « double » precision (64-bit storage) floating point values, as defined by IEEE 754.

FMS
Field Management System - the electronics core responsible for sensing and controlling the FIRST Robotics Competition field.

FPGA
Field-programmable gate array - a specialized integrated circuit consisting of many digital logic elements, which can be configured to act in different patterns. This allows its behavior to be changed after manufacturing. In the context of FRC, National Instruments provides a specific configuration for the RIO's FPGA which allows it to process the electrical inputs and outputs at a very high rate. See FPGA on Wikipedia for more info.

FRC
FIRST Robotics Competition - Combining the excitement of sport with the rigors of science and technology. The ultimate Sport for the Mind inspiring High-school students.

FTA
: FIRST Technical Advisor - FRC volunteer position that is responsible for ensuring FIRST Robotics Competition events run smoothly, safely, and in accordance with FIRST requirements, and ensuring a high-quality experience for all event participants and teams.

FTC
: FIRST Tech Challenge - Grades 7-12 are challenged to design, build, program, and operate robots to compete in a head-to-head challenge in an alliance format.

GDC
: Game Design Committee - designs the game for each year and develops the rules and field setup for each competition.

GP
: Gracious Professionalism - part of the ethos of FIRST. It's a way of doing things that encourages high-quality work, emphasizes the value of others, and respects individuals and the community.

GradleRIO
: Le mécanisme qui alimente le déploiement du code du robot sur le roboRIO.

gyroscope
: Un appareil qui mesure le taux de rotation. Il peut additionner les mesures de rotation pour déterminer : terme : «cap» du robot. («Gyro», pour faire court)

cap
: La direction dans laquelle le robot est pointé, généralement exprimée sous forme d'angle en degrés.

imperative programming
: A style of programming that focuses on *what* the code should be doing, step by step, every loop. See imperative programming on Wikipedia for more info.

IMU
: Inertial Measurement Unit - a sensor that combines both an *accelerometer* and a *gyroscope* into a single sensor.

I2C
: Inter-Integrated Circuit - a synchronous, multi-master/multi-slave (controller/target), single-ended, serial communication bus.

Java
: One of the four officially supported programming languages.

JSON
: JavaScript Object Notation - A standardized way of organizing data into named values. The organized data can be easily *serialized*. While the original usage was in Javascript, it can be used and interested by most modern programming languages. See JSON on Wikipedia for more info.

KOP
: Kit of Parts - the collection of items listed on the Kickoff Kit checklists, distributed to the team via FIRST Choice, or paid for completely (except shipping) with a Product Donation Voucher (PDV).

Châssis KOP
: The KOP contains a drive base (chassis) distributed to every team (that did not opt out) as part of the *KOP*. For the 2024 season, the KOP chassis is the AM14U5.

LabVIEW
: One of the four officially supported programming languages.

LED
: Light-Emitting Diode - a semiconductor device that emits light when current flows through it. Used on multiple robot parts to convey the status of the device.

mass
: the amount of matter in a physical object. Objects with more mass will resist changes in motion more than objects with less mass. See mass on Wikipedia for more info.

moment of inertia
: The property of an object that describes both how much mass it has, and how that mass is distributed relative to a certain axis of rotation. Objects with higher moments of inertia resist changes in rotational motion more than objects with lower moments of inertia. Increasing the moment of inertia is accomplished by adding more mass, or moving the mass further away from the axis of rotation. See moment of inertia on Wikipedia for more info.

mutable
: An object that can be modified after it is created.

MXP
: myRIO Expansion Port - Port in the center of the roboRIO designed to expand the traditional IO count by offering multiple different IO types through one connector.

NetworkTables
: A publish-subscribe messaging system to communicate data between programs.

no-op
: No-op is a computer instruction which means no operation. When the computer processor encounters a no-op instruction, it simply moves to the next sequential instruction. Read more about no-op on Wikipedia .

odometry
: Using sensors on the robot to create an estimate of the pose of the robot on the field.

OPR
: Offensive Power Rating - a system to attempt to deduce the average point contribution of a team to an alliance

PCM
: Pneumatic Control Module - provides an easy all-in-one interface for pneumatic components.

PDH
: REV Power Distribution Hub - latest evolution in power distribution for FRC. With 20 high-current (40A max) channels, 3 low-current (15A max), and 1 switchable low-current channel, the PDH gives teams more flexibility for overall power delivery.

PDP
: CTRE Power Distribution Panel - power distribution module with 8 high-current (40A max), 8 lower current (20A / 30A), 1 20A protected channel (for *PCM* and *VRM*), and 1 10A protected channel (for the roboRIO).

permanent-magnet DC motor
: The classification of all legal motors for the FIRST robotics competition. This type of motor takes direct current as input, and uses it to create a magnetic field. In turn, this magnetic field interacts with a physical magnet to create a force that turns the output shaft. Electrical (« brushless ») or mechanical (« brushed ») means are used to ensure the electrically-generated magnetic field always points in a direction that creates forces when it interacts with the physical magnet, even as the motor's shaft rotates. See permanent-magnet motor on Wikipedia for more info.

persistent
: In *NetworkTables*, a *topic* that is saved to a file by the server and restored at startup.

PH
: Pneumatic Hub - is a standalone module that is capable of switching both 12V and 24V pneumatic solenoid valves. The Pneumatic Hub features 16 solenoid channels which

allow for up to 16 single-acting solenoids, 8 double-acting solenoids, or a combination of the two types.

property
In *NetworkTables*, named information (metadata) about a *topic* stored and updated separately from the topic's data. A topic may have any number of properties. A property's value can be any data type that can be represented in JSON.

publisher
In *NetworkTables*, an object that defines a *topic* and creates and sends timestamped data values.

pose
The collection of position and rotation information that describes how a rigid body is oriented in space, relative to some fixed reference point.

pose estimation
The process of estimating the robot's pose, commonly with *odometry* and/or *AprilTags*. Also known as *on-field localization*.

PWM
Pulse-width modulation - a method of controlling the average power or amplitude delivered by an electrical signal. Used in FRC to control the output of motors not using the *CAN* bus.

Python
One of the four officially supported programming languages.

RAII
Resource Acquisition Is Initialization - a language behavior (in C++, but not in Java) where holding a resource is tied to object lifetime.

retro-reflection
The property of reflecting incoming light back at the same angle it came in at, rather than an incident angle (like a mirror), absorbing it, or scattering it. Most FRC vision processing targets are retro-reflective. See retroreflector on Wikipedia for more information.

recursive composition
A type of *composition* in which the composite object may contain components of the same type as itself. For example, a command group may contain one or more command groups. See recursive composition on Wikipedia for more info. See also *recursive composition*.

retained
In *NetworkTables*, a *topic* that is kept alive by the server even after all publishers stop publishing.

REV
REV Robotics - inspires innovation and creativity within the educational robotics community by offering comprehensive product lines, extensive educational resources, world-class customer service, and specialized sponsorship programs. With a global presence spanning over 190 countries, we empower the next generation of STEM professionals by providing cutting-edge solutions and essential tools for success. Founded in 2014 by robotics enthusiasts Greg Needel and David Yanoshak, REV Robotics is driven by the mission to inspire and support students as they explore the exciting world of robotics and unlock their full robotic design potential. A majority of our employees are FIRST Alumni who remain actively involved, serving as volunteers and mentors for the local FIRST Community. This deep engagement reflects our commitment to supporting and inspiring the next generation of STEM enthusiasts.

RPM
: Radio Power Module - is designed to keep one of the most critical system components, the OpenMesh OM5P-AC WiFi radio, powered in the toughest moments of the competition. Revolutions Per Minute - a unit of rotational speed often used when describing motors.

RSL
: Robot Signal Light - safety light on every FRC robot used to indicate its operational status.

serialized
: The property of a data organization scheme that allows the description of the data to be sent in order, byte by byte, over some communication channel. Reading or writing a file on disk is done in this serial fashion (IE, the data is read or written byte by byte, not all at once). Sending data over a *SPI* or *I2C* bus is also done byte by byte, again requiring the data can be serialized.

simulation
: Un moyen pour les équipes de tester leur code sans disposer d'un robot réel.

software library
: A collection of code that can be imported into and used by other software. See software library on Wikipedia for more info.

solenoid valve
: A airflow-controlling valve which is actuated by a small electromagnet. Strictly speaking, the *solenoid* is the coil of wire which forms the electromagnet, and the *valve* is the mechanism which actually redirects airflow. However, the set of solenoid and valve together is often simply called « a solenoid ». See solenoid valve. on Wikipedia for more info.

SPI
: Serial Peripheral Interface - protocol for synchronous serial communication, used primarily in embedded systems for short-distance wired communication between integrated circuits.

state machine
: A programming construct that divides a problem into many discrete, well-defined, mutually-exclusive « states », then defines how the problem is solved by moving between different states. See state machine on Wikipedia for more more info.

subscriber
: In *NetworkTables*, an object that receives timestamped data value updates to one or more *topic*s.

TBA
: The Blue Alliance - Website for looking up *FRC* team statistics and event information.

telemetry
: The process of recording and sending real-time data about the performance of your robot to a real-time readout or log file. For the linguists among us, the word's roots are « tele » (remote) and « metry » (measurement). See telemetry on Wikipedia for more info.

téléop
: La deuxième phase de chaque match est appelée la période de téléopération (téléopération) et consiste en des pilotes contrôlant leurs robots.

topic
: In *NetworkTables*, a named data channel.

torque
: A force applied at a distance from some axis of rotation

trajectoire
: Une trajectoire est une courbe lisse, avec des vitesses et des accélérations à chaque point le long de la courbe, reliant deux extrémités sur le terrain.

transitory
: In *NetworkTables*, a *topic* that will disappear after the last *publisher* stops publishing.

VRM
: Voltage Regulator Module - provides access to different constant voltages for custom sensors, cameras, or other unique applications. 12V DC Input Directly fed power from the Power Distribution Panel Designed to work with the roboRIO FRC control system.

WCP
: WestCoast Products & Design LLC - was founded in Fall of 2011 by Ranjit Chahal (R.C.) and Harvey Rico. WCP aims to provide FIRST Teams, Hobbyists, and educators top notch quality products and designs for their projects.

WFA
: Woodie Flowers Award - This award recognizes an individual who has done an outstanding job of motivation through communication while also challenging the students to be clear and succinct in their communications.

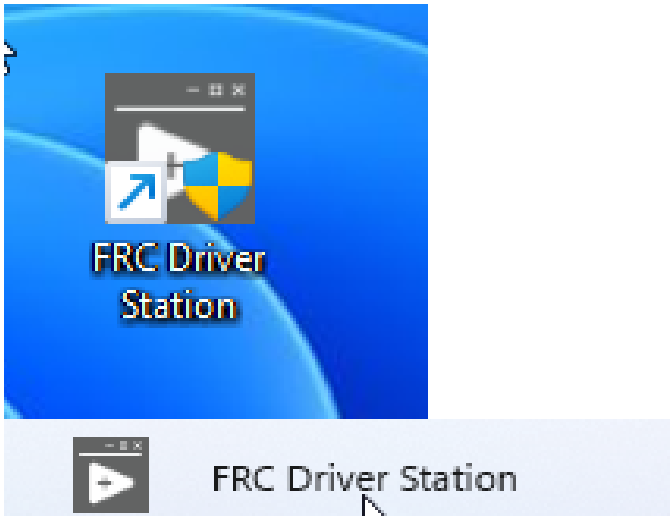
20 Driver Station

20.1 Driver Station FRC par NI LabVIEW

This article describes the use and features of the FRC® Driver Station Powered by NI LabVIEW.

Pour plus d'informations sur l'installation de l'application Driver Station, consulter *ce document*.

20.1.1 Démarrage de Driver Station FRC



The FRC Driver Station can be launched by double-clicking the icon on the Desktop or by selecting Start->All Apps->FRC Driver Station.

Note : By default the FRC Driver Station launches the *LabVIEW Dashboard*. It can also be configured on *Onglet Setup* to launch the other Dashboards : *SmartDashboard* and *Shuffleboard*. WPILib must be *installed* to use SmartDashboard and Shuffleboard.

20.1.2 Principaux raccourcis clavier de Driver Station

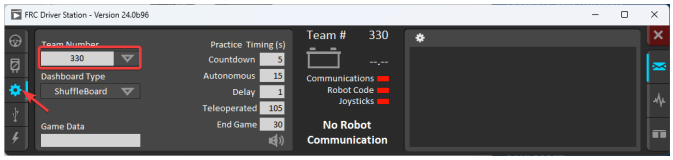
— `F1` - Force a Joystick refresh.
— `[ + ] + \` - Enable the robot (the 3 keys above Enter on most keyboards)
— `Enter` - Disable the Robot
— `Space` - Emergency Stop the robot. After an emergency stop is triggered the roboRIO will need to be rebooted before the robot can be enabled again.

Note : La barre d'espacement arrêtera en urgence le robot, que la fenêtre Driver Station soit active ou non

---

Avertissement : When connected to *FMS* in a match, teams must press the Team Station E-Stop button to emergency stop their robot as the DS enable/disable and E-Stop key shortcuts are ignored.

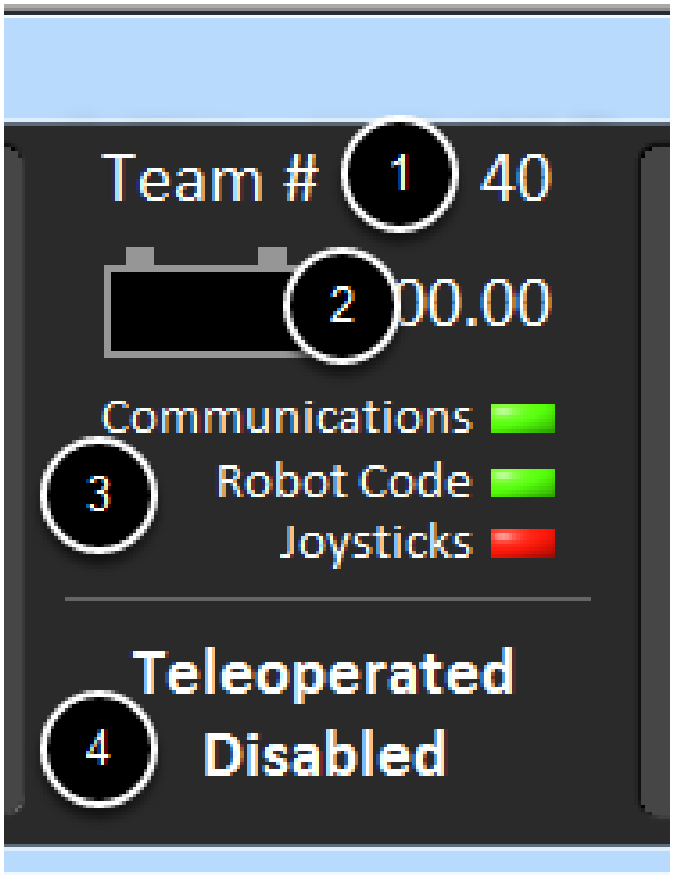
20.1.3 Configuration de Driver Station



Le DS doit être configuré sur votre numéro d'équipe afin de vous connecter à votre robot. Pour ce faire, cliquez sur l'onglet Setup puis entrez votre numéro d'équipe dans la boîte de numéro d'équipe. Appuyez sur return ou cliquez en dehors de la boîte pour que la configuration prenne effet.

PCs will typically have the correct network settings for the DS to connect to the robot already, but if not, make sure your Network adapter is set to *DHCP*.

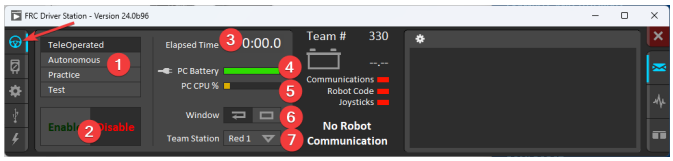
20.1.4 Volet Status



Le volet Status de Driver Station est situé au centre de l'interface et est toujours visible quel que soit l'onglet sélectionné. Il affiche une sélection d'informations essentielles sur l'état de DS et du robot :

1. Team # - Numéro d'équipe auquel DS est actuellement configuré. Cela devrait correspondre à votre numéro d'équipe FRC. Pour modifier le numéro d'équipe, voir l'onglet Setup .
2. Battery Voltage - Si DS est connecté et communique avec le roboRIO, la tension actuelle de la batterie s'affiche sous la forme d'un nombre accompagné d'un petit graphique de la tension en fonction du temps dans l'icône de la batterie. L'arrière-plan de l'indicateur numérique deviendra rouge lorsque l'avertissement de déclin de performance du roboRIO est déclenché. Voir *Baisse de tension du roboRIO et compréhension de la consommation de courant* pour plus d'information.
3. Major Status Indicators - Ces trois indicateurs affichent les trois principaux éléments concernant l'état de la DS. « Communications » indique si DS communique actuellement avec la tâche de communication réseau FRC du roboRIO (elle est divisée en deux, soit pour la communication TCP et pour UDP). L'indicateur « Robot Code » indique si le Programme du robot est en cours d'exécution (déterminé par si oui ou non la tâche Driver Station met à jour la tension de la batterie), l'indicateur « Joysticks » indique si au moins une manette est branchée et reconnue par DS.
4. Status String - The Status String provides an overall status message indicating the state of the robot. Some examples are « No Robot Communication », « No Robot Code », « Emergency Stopped », and « Teleoperated Enabled ». When the roboRIO brownout is triggered this will display « Voltage Brownout ».

20.1.5 Onglet Operation



L'onglet Operations est utilisé pour contrôler le mode du robot et fournir d'importants indicateurs d'état supplémentaires lorsque que le robot est activé.

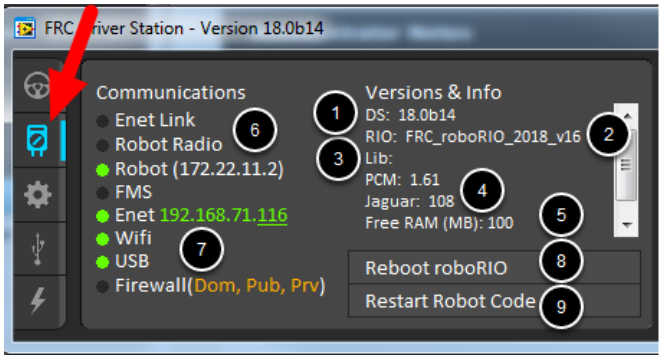
1. Mode robot - Cette section contrôle le mode robot.

— Le mode téléopéré amène le robot à exécuter le code dans la partie téléopérée du match.
— Le mode autonome oblige le robot à exécuter le code dans la partie autonome du match.
— Le mode d'entraînement amène le robot à parcourir les mêmes transitions qu'un match FRC après avoir appuyé sur le bouton Activer (la synchronisation du mode d'entraînement peut être trouvée dans l'onglet de configuration).
— Test Mode est un mode supplémentaire où le code de test qui ne s'exécute pas dans une correspondance régulière peut être testé.

2. Enable/Disable - These controls enable and disable the robot. See also *Driver Station Key Shortcuts*.
3. Elapsed Time - Indicates the amount of time the robot has been enabled.
4. PC Battery - Indicates current state of DS PC battery and whether the PC is plugged in.
5. PC CPU% - Indicates the CPU Utilization of the DS PC.
6. Window Mode - When not on the Driver account on the Classmate allows the user to toggle between floating (arrow) and docked (rectangle).
7. Team Station - Lorsque non connecté au FMS, configure la station d'équipe à transmettre au robot.

Note : When connected to the Field Management System the controls in sections 1 and 2 will be replaced by the words FMS Connected and the control in Section 7 will be greyed out.

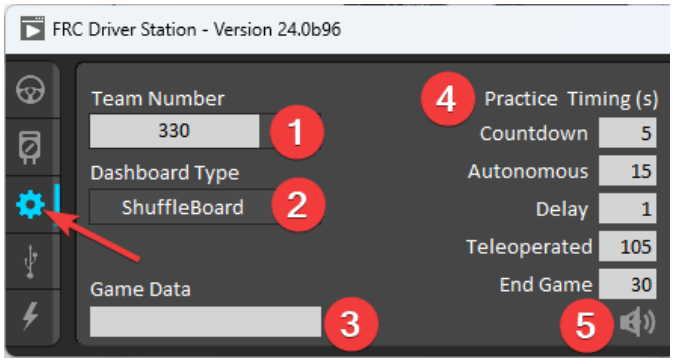
20.1.6 Onglet Diagnostics



L'onglet Diagnostics contient des indicateurs d'état supplémentaires que les équipes peuvent utiliser pour identifier les problèmes de leur robot :

1. DS Version - Indicates the Driver Station Version number.
2. roboRIO Image Version - String indicating the version of the roboRIO Image.
3. WPILib Version - String indicating the version of WPILib in use.
4. *CAN* Device Versions - String indicating the firmware version of devices connected to the CAN bus. These items may not be present if the CTRE Phoenix Framework has not been loaded.
5. Memory Stats - This section shows stats about the roboRIO memory.
6. Connection Indicators - La moitié supérieure de ces indicateurs montre l'état de connexion à divers composants.
   — « Enet Link » indique que l'ordinateur détecte une connexion quelconque à son port ethernet.
   — « Robot Radio » indique l'état de ping du pont sans fil du robot à 10.XX.YY.1.
   — « Robot » indique le résultat de la commande ping au roboRIO à l'aide de mDNS (avec un repli de l'adresse statique 10.TE.AM.2).
   — « FMS » indique si DS reçoit des paquets du FMS (ce N'EST PAS un indicateur de ping).
7. Network Indicators - The second section of indicators indicates status of network adapters and firewalls. These are provided for informational purposes ; communication may be established even with one or more unlit indicators in this section.
   — « Enet » indique l'adresse IP de l'adaptateur Ethernet détecté
   — « WiFi » indique si un adaptateur sans fil a été détecté comme activé
   — « USB » indique si une connexion USB du roboRIO a été détectée
   — « Firewall » indicates if any firewalls are detected as enabled. Enabled firewalls will show in orange (Dom = Domain, Pub = Public, Prv = Private)
8. *Reboot roboRIO* - This button attempts to perform a remote reboot of the roboRIO (after clicking through a confirmation dialog).
9. *Restart Robot Code* - This button attempts to restart the code running on the robot (but not restart the OS).

20.1.7 Onglet Setup

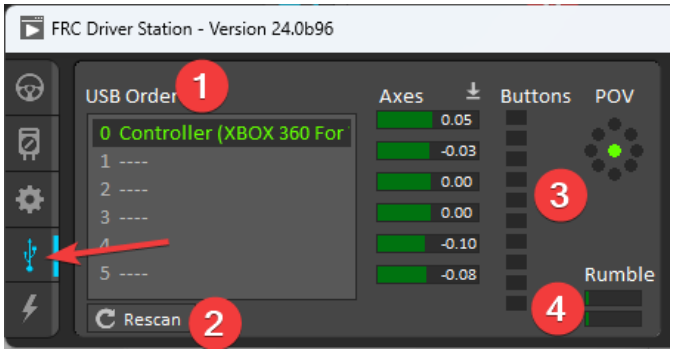


L'onglet Setup contient un certain nombre de boutons que les équipes peuvent utiliser pour contrôler le fonctionnement de Driver Station :

1. *Team Number* - Should contain your FRC Team Number. This controls the mDNS name that the DS expects the robot to be at. Shift clicking on the dropdown arrow will show all roboRIO names detected on the network for troubleshooting purposes.

2. *Dashboard Type* - Controls what Dashboard is launched by the Driver Station. *Default* launches the file pointed to by the « FRC DS Data Storage.ini » (for more information about setting a *custom dashboard*). By default this is Dashboard.exe in the Program Files (x86)\FRC Dashboard folder. *LabVIEW* attempts to launch a dashboard at the default location for a custom built LabVIEW dashboard, but will fall back to the default if no dashboard is found. *SmartDashboard* and *Shuffleboard* launch the respective dashboards included with the C++ and Java WPILib installation. *Remote* forwards LabVIEW dashboard data to the IP specified in *Dashboard IP* field.
3. *Game Data* - This box can be used for at home testing of the Game Data API. Text entered into this box will appear in the Game Data API on the Robot Side. When connected to FMS, this data will be populated by the field automatically.
4. Practice Mode Timing - Ces cases contrôlent le minutage de chaque partie de la séquence du Mode de Pratique. Lorsque le robot est activé en Mode Pratique, DS passe automatiquement par les modes indiqués de haut en bas.
5. Audio Control - Ce bouton contrôle si les signaux sonores doivent être joués en Mode Pratique.

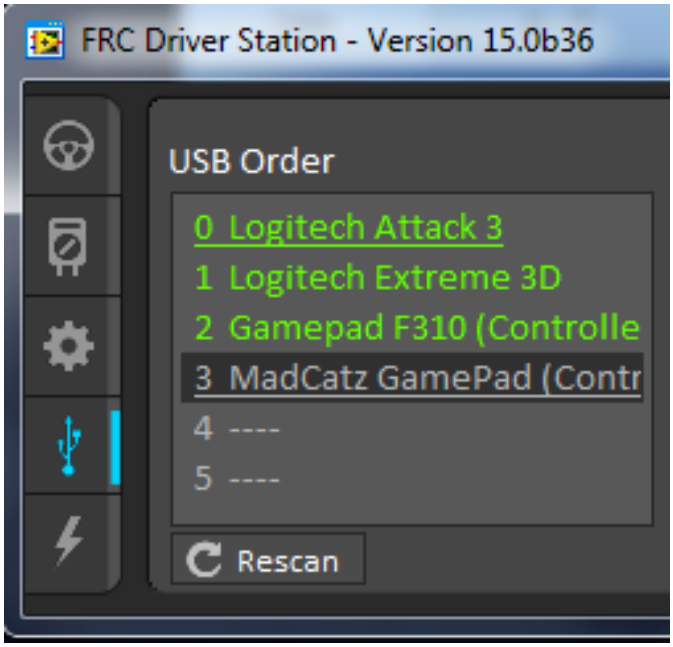
20.1.8 Onglet USB Devices



L'onglet USB Devices ou Périphériques USB inclut les informations relatives aux périphériques USB connectés à DS

1. USB Setup List - Contient une liste de tous les périphériques USB compatibles connectés à DS. Appuyer sur un bouton d'un périphérique mettra en surbrillance le nom de ce périphérique en vert précédé de 2 *s.
2. *Rescan* - This button will force a Rescan of the USB devices. While the robot is disabled, the DS will automatically scan for new devices and add them to the list. To force a complete re-scan or to re-scan while the robot is Enabled (such as when connected to FMS during a match) press F1 or use this button.
3. Indicateurs de périphériques - Ces indicateurs montrent l'état actuel des Axes, boutons et POV de la manette.
4. Rumble - Pour les périphériques XInput (tels que les contrôleurs X-Box), le contrôle Rumble s'affiche. Ceci peut être utilisé pour tester la fonction de vibration de l'appareil. La barre supérieure est « Left Rumble » et la barre inférieure est « Right Rumble ». Cliquer et tenir n'importe où le long de la barre activera proportionnellement la vibration (à gauche, sans vibration = 0, à droite pleine vibration = 1). Il s'agit d'un contrôle uniquement et n'indiquera pas la valeur Rumble définie dans le code du robot.

Réarrangement et ancrage des dispositifs



The Driver Station has the capability of « locking » a USB device into a specific slot. This is done automatically if the device is dragged to a new position and can also be triggered by double clicking on the device. « Locked » devices will show up with an underline under the device. A locked device will reserve its slot even when the device is not connected to the computer (shown as grayed out and underlined). Devices can be unlocked (and unconnected devices removed) by double clicking on the entry.

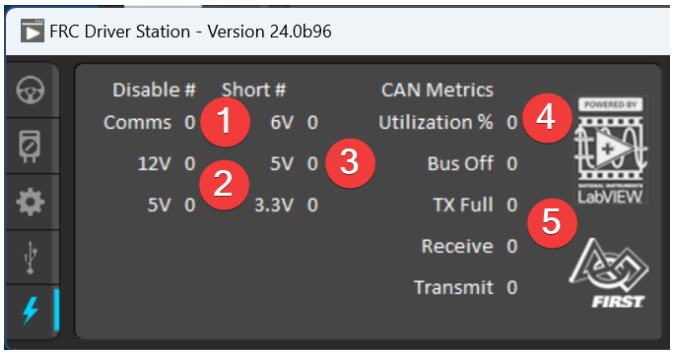
Note : If you have two or more of the same device, they should maintain their position as long as all devices remain plugged into the computer in the same ports they were locked in. If you switch the ports of two identical devices the lock should follow the port, not the device. If you re-arrange the ports (take one device and plug it into a new port instead of swapping) the behavior is not determinate (the devices may swap slots). If you unplug one or more of the set of devices, the positions of the others may move ; they should return to the proper locked slots when all devices are reconnected.

Exemple : L'image ci-dessus montre 4 périphériques :

- Une manette « Logitech Attack 3 » verrouillée. Ce périphérique restera dans cette position à moins d'être déplacé ailleurs ou déverrouillé
- Une manette « Logitech Extreme 3D » déverrouillée
- Un contrôleur « Gamepad F310 (Controller) » déverrouillé ; c'est un Gamepad Logitech F310
- Un contrôleur « MadCatz GamePad (Controller) » verrouillé mais déconnecté ; c'est un contrôleur MadCatz Xbox 360

Dans cet exemple, débrancher la manette Logitech Extreme 3D se traduira pour le F310 Gamepad à passer au port 1. Le branchement du Gamepad MadCatz (même si les périphériques aux ports 1 et 2 sont supprimés et que ces emplacements sont vides) se traduira par l'occupation du port 3.

20.1.9 Onglet CAN/Power

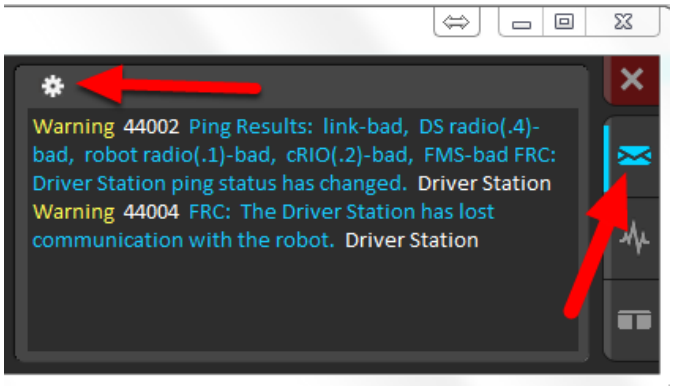


Le dernier onglet sur le côté gauche de DS est l'onglet CAN/Power. Cet onglet contient des informations sur l'état de l'alimentation du roboRIO et l'état du bus CAN :

1. Pannes Comms - Indique le nombre d'incidents de communication qui se sont produits depuis que DS a été connecté
2. Pannes 12V - Indique le nombre de pannes de puissance d'entrée (sous-tensions) qui se sont produites depuis que DS a été connecté
3. 6V/5V/3.3V Faults - Indicates the number of faults (typically caused by short circuits) that have occurred on the User Voltage Rails since the DS has been connected
4. Utilisation du Bus CAN - Indique le pourcentage d'utilisation du système bus CAN
5. Pannes CAN - Indique, pour chacun des 4 types de pannes CAN, le nombre de pannes produites depuis que DS a été connecté

Si une panne survient, l'indicateur de cet onglet (indiqué en bleu dans l'image ci-dessus) deviendra rouge.

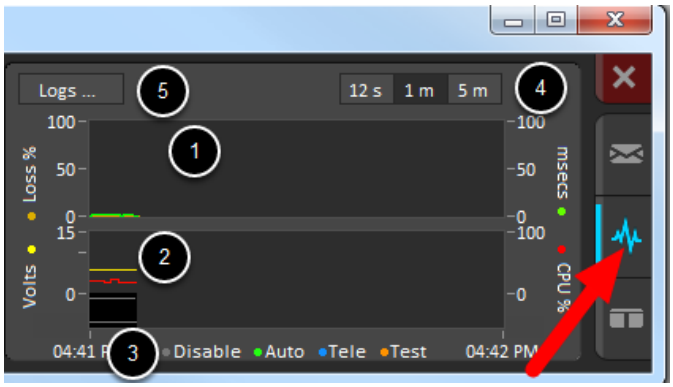
20.1.10 Onglet Messages



The Messages tab displays diagnostic messages from the DS, WPILib, User Code, and/or the roboRIO.

To access settings for the Messages tab, click the Gear icon. This will display a menu that will allow you to clear the box, launch a larger Console window for viewing messages, launch the DS Log Viewer, launch a viewer for program timing, or download log files from the robot.

20.1.11 Onglet Graphiques



L'onglet Graphiques trace et affiche des indicateurs avancés de l'état du robot pour aider les équipes à diagnostiquer les problèmes :

1. The top graph charts trip time in milliseconds in green (against the axis on the right) and lost packets per second in orange (against the axis on the left).
2. Le graphique en bas montre la tension de la batterie en jaune (contre l'axe à gauche), le %CPU du roboRIO en rouge (contre l'axe à droite), le mode DS Requested en ligne continue en bas du graphique et le mode du robot en ligne discontinue au-dessus.
3. Cette légende affiche les couleurs utilisées pour les modes DS Requested et Robot Reported dans le graphique inférieur.
4. Chart scale - These controls change the time scale of the DS Charts.
5. This button launches the *DS Log File Viewer*.

Le mode DS Requested est le mode dans lequel Driver Station commande au robot d'opérer. Le mode Robot Reported correspond au code en cours d'exécution selon les méthodes définies dans les programmes propres à chacun des langages.

20.1.12 Onglet Côte-à-côte

The last tab on the right side is the Both tab which displays Messages and Charts side by side.

20.2 Meilleures pratiques avec Driver Station

This document was created by Steve Peterson, with contributions from Juan Chong, James Cole-Henry, Rick Kosbab, Greg McKaskle, Chris Picone, Chris Roadfeldt, Joe Ross, and Ryan Sjostrand. The original post and follow-up posts can be found here.

Le poste de pilotage ne doit pas constituer un embêtement pour votre équipe sur le terrain de la Compétition de robotique FIRST (FRC). Construire une console solide et y configurer un ordinateur portable stable est un projet facile à réaliser avant la compétition. Lisez la suite pour découvrir les leçons apprises par de nombreuses équipes ayant disputé des milliers de matchs.

20.2.1 Avant le départ pour la compétition

1. Dédiez un ordinateur portable à utiliser uniquement comme station de pilotage. Beaucoup d'équipes le font. Un portable dédié vous permet de gérer sa configuration pour un seul objectif : être prêt à jouer votre match sur le terrain. Dédié signifie qu'aucun autre logiciel, à l'exception du logiciel Driver Station fourni par FRC et Dashboard, seront installés et en cours d'exécution.
2. Utilisez un ordinateur portable de classe affaires pour votre poste de pilotage. Pourquoi ? Ils sont beaucoup plus durables que les aubaines Black Friday à 300 $ chez Best Buy. Ils survivront à la compétition. Les ordinateurs portables de classe affaires ont des pilotes de périphérique de meilleure qualité et ces pilotes seront supportés plus longtemps par les manufacturiers que ceux trouvés dans les ordinateurs portables grand public. Cela prolonge la durée de votre investissement. Les séries Lenovo ThinkPad T et Dell Latitude sont deux marques de classe affaires populaires que vous verrez souvent en tournoi. Il y en a des milliers en vente chaque jour sur eBay. L'ordinateur portable fourni dans le kit de pièces des équipes recrues est une bonne machine d'entrée de gamme. Les équipes font le saut vers des écrans plus grands, car ils sont plus pratiques avec la vision et les tableaux de bord.
3. Consider used laptops rather than new. The FRC® Driver Station and dashboard software uses very few system resources, so you don't need to buy a new laptop – instead, buy a cheap 4-5 year old used one. You might even get one donated by a used computer store in your area.

Note : Before buying a used laptop ensure it is compatible with Windows 11. For example, only Intel 8th generation core processors and later are compatible.

---

4. Caractéristiques recommandées pour l'ordinateur portable
   a. RAM – 8GB of RAM or greater
   b. Une taille d'affichage de 13 po ou plus, avec une résolution minimale de 1440x1050.
   c. Ports
      i. Un port Ethernet intégré est très recommandé. Assurez-vous qu'il s'agit d'un port de pleine grandeur. Les ports Ethernet articulés ne tiennent pas quand ils sont trop souvent utilisés.
      ii. Utilisez un protecteur de port Ethernet pour créer votre connexion Ethernet. Il s'agit d'une petite extension Ethernet que vous laissez habituellement connectée sur l'ordinateur. Cela prolonge la durée de vie du port sur l'ordinateur portable. Ceci est particulièrement important si vous avez un ordinateur portable grand public avec un port Ethernet articulé.
      iii. Si le port Ethernet de votre ordinateur portable est douteux, remplacez l'ordinateur portable (recommandé) ou achetez un adaptateur Ethernet USB d'une marque réputée. Beaucoup d'équipes trouvent qu'un adaptateur Ethernet USB est moins fiable que le port Ethernet intégré, principalement en raison de matériel de mauvaise qualité et de mauvais pilotes. Les adaptateurs donnés aux recrues dans le kit de pièces ont la réputation de bien fonctionner.
      iv. 2 ports USB minimum
   d. Un clavier. Il est difficile de faire rapidement du dépannage sur le terrain sur les ordinateurs uniquement tactiles.
   e. A solid-state disk (SSD), 256 GB or larger. If the laptop has a rotating disk, spend $50 and replace it with a SSD.

   f. Updated to the current release of Windows 10 or 11.
   g. A laptop that supports Wi-Fi 6E (6 GHz) is recommended for use with the Wi-Fi 6E radio for 2025 and later.
5. Installez toutes les mises à jour Windows une semaine avant la compétition. Cela vous donne le temps de vous assurer que les mises à jour n'interféreront pas avec les fonctions du Driver Station. Pour ce faire, ouvrez la page Paramètres de Windows Update et voyez si vous êtes à jour. Si vous ne l'êtes pas, installez les mises à jour en attente. Redémarrez et vérifiez à nouveau pour vous assurer que vous êtes à jour.
6. Modifiez les « heures d'activité » pour les mises à jour Windows pour empêcher les mises à jour d'être installées pendant les heures de compétition. Accédez à Start -> Settings -> Update & Security -> Windows Update, puis sélectionnez Change active hours. Si vous voyagez à une compétition, prenez en compte les différences de fuseau horaire. Cela vous permettra de vous assurer que votre station de pilotage ne redémarre pas ou n'arrête pas de fonctionner en raison de l'installation d'une mise à jour sur place.
7. Supprimez tout logiciel antivirus ou antimalware tiers. Utilisez plutôt Windows Defender sur Windows 10 ou 11. Puisque vous vous connectez uniquement à Internet pour la mise à jour des logiciels Windows et FRC, le risque est faible. Installez uniquement les logiciels nécessaires au pilotage du robot. Votre but ici est d'éliminer les variables qui pourraient interférer avec le bon fonctionnement. Supprimez tout logiciel préinstallé inutile (« bloatware ») qui est venu avec la machine. N'utilisez pas l'ordinateur portable pour jouer à des jeux à l'hôtel la veille de l'événement. Beaucoup d'équipes vont aussi loin que se procurer un ordinateur portable dédié uniquement à la programmation.
8. Évitez les installations Windows 10 ou 11 gérées par le département informatique de l'école. Ces déploiements sont conçus pour l'environnement scolaire et viennent souvent avec des logiciels indésirables qui interfèrent avec le bon fonctionnement de votre robot.
9. Batterie d'ordinateur portable / alimentation
   a. Désactivez l'option « mettre le PC en veille » pour le mode batterie et le mode branché dans vos paramètres d'alimentation et veille.
   b. Désactiver la suspension sélective USB :
      i. Cliquez avec le bouton droit sur l'icône batterie/charge dans la barre des tâches, puis sélectionnez Options d'alimentation.
      ii. Choisissez un des modes de gestion de l'alimentation.
      iii. Dans le menu de gauche, cliquez sur Choisir quand éteindre l'écran. Ensuite, cliquez sur Modifier les paramètres d'alimentation avancés.
      iv. Faites défiler vers le bas dans les paramètres avancés et désactivez le paramètre de suspension sélective USB pour batterie et branché.
   c. Assurez-vous que la batterie de l'ordinateur portable peut contenir une charge pendant au moins une heure après avoir fait les changements ci-dessus. Cela laisse beaucoup de temps pour le robot et l'équipe de pilotage de passer par la file d'attente et d'atteindre la station d'alliance sans être branché.
10. Apportez un câble USB et Ethernet de confiance pour la connexion au roboRIO.
11. Ajoutez un dispositif de rétention, ou qui limite la traction sur les fils de vos manettes / contrôleurs pour les empêcher de tomber au sol ou se déconnecter des ports USB. Cela permet d'éviter les problèmes de connexions intermittentes de contrôleur.
12. Le compte d'utilisateur Windows que vous utilisez pour piloter doit faire partie du groupe Administrateur.

20.2.2 En tournoi

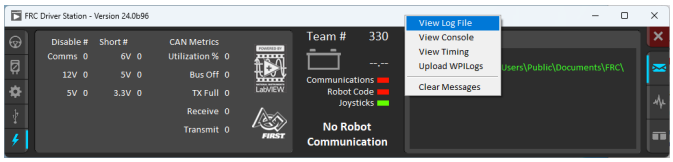
1. Turn off Windows firewall using *these instructions*.
2. Désactivez l'adaptateur Wi-Fi, soit à l'aide du bouton Wi-Fi physique, soit en le désactivant dans le panneau de configuration dans les Paramètres de l'adaptateur.
3. Chargez la station de pilotage quand elle est dans le puits.
4. Supprimez les mots de passe de connexion ou assurez-vous que tous les membres de l'équipe de terrain connaissent le mot de passe. Vous seriez surpris de voir combien de fois les pilotes arrivent sur le terrain sans connaître le mot de passe de l'ordinateur portable.
5. Assurez-vous que votre code LabView est déployé en permanence et défini sur « run as startup », en utilisant les instructions du tutoriel LabView. Si vous devez déployer le code chaque fois que vous allumez le robot, quelque chose ne fonctionne pas.
6. Limitez la navigation web aux sites web liés à FRC. Cela minimise les chances d'obtenir des logiciels malveillants pendant la compétition.
7. Ne comptez pas utiliser Internet pour effectuer des mises à jour logicielles. Il n'y en aura probablement pas dans la salle et le Wi-Fi de l'hôtel varie considérablement en qualité. Si vous avez besoin de mises à jour, contactez un conseiller en système de contrôle durant la compétition.

20.2.3 Avant chaque match

1. Assurez-vous que l'ordinateur portable est allumé et connecté sur le bon compte d'utilisateur avant la fin du match avant le vôtre.
2. Fermez les programmes qui ne sont pas nécessaires pendant le match – par exemple, Visual Studio Code ou LabView – lorsque vous êtes en compétition.
3. Apportez votre chargeur du portable sur le terrain. Des prises de courant sont fournies dans chaque station.
4. Attachez votre ordinateur portable avec du ruban velcro sur la tablette de la station de pilotage. Vous ne savez jamais quand votre partenaire d'alliance aura un problème de programmation autonome et que son robot ira frapper le mur de la station de pilotage.
5. Assurez-vous que les manettes et les contrôleurs sont affectés aux bons ports USB.
   a. Dans l'onglet USB du logiciel FRC Driver Station, faites glisser/déposer pour attribuer des manettes au besoin.
   b. Utilisez le bouton Rescan (F1) si les manettes/contrôleurs n'apparaissent pas en vert.
   c. Utilisez le bouton Rescan (F1) pendant la compétition si les manettes ou les contrôleurs se débranchent et se rebranchent ou apparaissent en gris pendant la compétition.
6. Ensure your *Dashboard is connected to the robot* after your driver station connects to the robot.

20.3 Visionneur de fichiers journaux de Driver Station

In an effort to provide information to aid in debugging, the FRC® Driver Station creates log files of important diagnostic data while running. These logs can be reviewed later using the FRC Driver Station Log Viewer. The Log Viewer can be found via the shortcut installed in the Start menu, in the FRC Driver Station folder in Program Files, or via the Gear icon in the Driver Station.



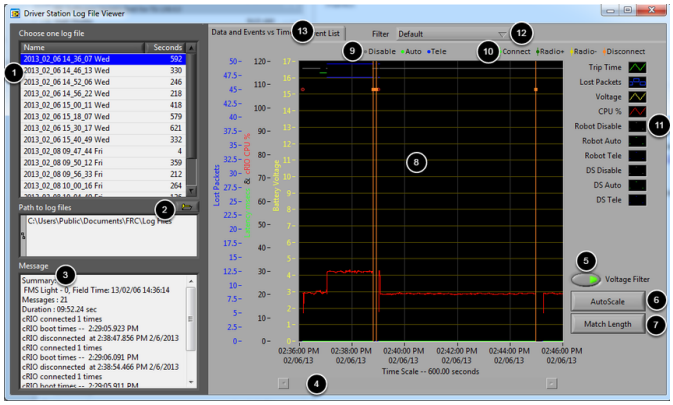
Note : Several alternative tools exist that provide similar functionality to the FRC Driver Station Log Viewer. *AdvantageScope* is an option included in WPILib, and DSLOG Reader is a third-party option. Note that WPILib offers no support for third-party projects.

20.3.1 Journaux d'événements

The Driver Station logs all messages sent to the Messages box on the Diagnostics tab (not the User Messages box on the Operation tab) into a new Event Log file. When viewing Log Files with the Driver Station Log File Viewer, the Event Log and Driver Station Log files are overlaid in a single display.

Log files are stored in `C:\Users\Public\Documents\FRC\Log Files`. Each log has date and timestamp in the file name and has two files with extension `.dslog` and `.dsevents`.

20.3.2 Interface du visionneur

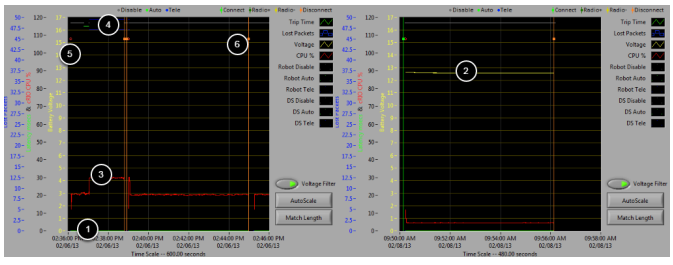


Le visionneur de journaux contient un certain nombre de contrôles et d'affichages pour faciliter l'analyse des fichiers journaux de Driver Station :

1. Zone de sélection des fichiers - Cette fenêtre affiche tous les fichiers journaux disponibles dans le dossier actuellement sélectionné. Cliquez sur un fichier journal dans la liste pour le sélectionner.

2. Chemin d'accès aux fichiers journaux - Cette zone affiche le dossier actuel dans lequel le visionneur recherche des fichiers journaux. Par défaut, ce dossier est le dossier dans lequel Driver Station range les fichiers journaux. Cliquez sur l'icône du dossier pour accéder à un autre emplacement.
3. Zone de message - Cette zone affiche un résumé de tous les messages du journal d'événements. Lorsque vous passez le curseur au dessus d'un événement sur le graphique, cette case change pour afficher les informations relatives à cet événement.
4. Barre de défilement - Lorsque le graphique est agrandi, cette barre de défilement permet de faire un défilement horizontal du graphique.
5. Filtre de tension - Ce contrôle active et désactive le filtre de tension (par défaut, actif). Le filtre de tension filtre les données telles que le % CPU, le Mode du Robot et le temps de surtension lorsqu'aucune tension de batterie n'est reçue (ce qui indique que DS n'est pas en communication avec le roboRIO).
6. AutoScale - Ce bouton effectue un zoom arrière du graphique pour afficher toutes les données du journal.
7. Match length - Ce bouton étale le graphique à environ la durée d'un match FRC (2 minutes et 30 secondes indiquées). Il ne localise pas automatiquement le début du match, vous devrez parcourir le graphe à l'aide de la barre de défilement pour localiser le début du mode autonome.
8. Graphique - Cet écran montre sous forme graphique les données du fichier journal DS (tension, surtension, %CPU du roboRIO, paquets perdus et mode du robot) ainsi que les données d'événements superposées (affichées sous forme de points sur le graphique avec certains événements montrant des lignes verticales sur l'ensemble du graphique). En passant le curseur sur les marqueurs d'événements du graphique, on affiche des informations sur l'événement dans la fenêtre Messages en bas à gauche de l'écran.
9. Légende du Mode Robot - Légende pour le Mode Robot affiché en haut de l'écran
10. Légende d'événement majeur - Légende pour les événements majeurs, affichés sous forme de lignes verticales sur le graphique
11. Légende graphique - Légende pour les données graphiques
12. Contrôle du filtrage - Liste déroulante pour sélectionner le mode de filtrage (modes de filtrage expliqués ci-dessous)
13. Contrôle d'onglet - Contrôle pour basculer entre les affichages graphique (données et événements vs temps) et liste d'événements.

20.3.3 Comment utiliser l'affichage graphique



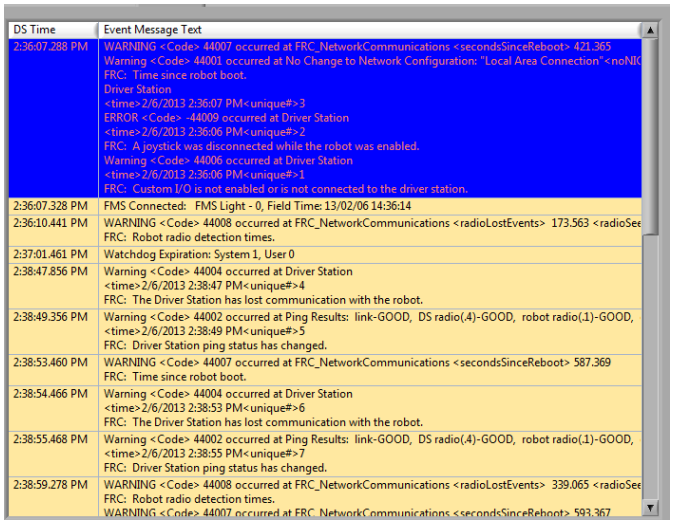
L'affichage graphique contient les informations suivantes :

1. Graphiques du temps de surtension en ms (ligne verte) et paquets perdus par seconde (affichés sous forme de barres verticales bleues). Dans ces exemples, la surtension est une ligne verte plate au bas du graphique et il n'y a pas de paquets perdus

2. Graphique de la tension de la batterie affichée sous la forme d'une ligne jaune.
3. Graphique du % CPU du roboRIO comme ligne rouge
4. Graphique des modes Robot et DS. L'ensemble supérieur de l'écran montre le mode dicté par Driver Station. L'ensemble inférieur montre le mode exécuté par le code du robot. Dans cet exemple, le robot ne communique pas son mode pendant les phases désactivé et autonome, mais il le communique pendant le Mode Teleop.
5. Les marqueurs d'événements sont affichés sur le graphique indiquant le moment où l'événement a eu lieu. Les erreurs s'affichent en rouge ; les avertissements en jaune. En passant le curseur par dessus un marqueur d'événement, vous afficherez des informations sur l'événement dans la zone Messages en bas à gauche de l'écran.
6. Les événements majeurs apparaissent sous forme de lignes verticales sur l'affichage du graphique.

Pour agrandir une partie du graphique, cliquez et faites glisser autour de la zone d'affichage souhaitée. Vous ne pouvez zoomer que sur l'axe de temps, vous ne pouvez pas zoomer verticalement.

20.3.4 Liste des événements



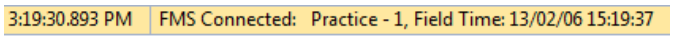
L'onglet Event List affiche une liste d'événements (avertissements et erreurs) enregistrés par Driver Station. Les événements et les détails affichés sont déterminés par le filtre actuellement actif (l'image montre le résultat obtenu du filtre actif « All Events, All Info »).

20.3.5 Filtres

Trois filtres sont disponibles dans le visionneur de journaux :

1. Default : ce filtre filtre bon nombre des erreurs et avertissements rapportés par Driver Station. Ce filtre est utile pour identifier les erreurs émises par le code sur le robot.
2. All Events and Time : ce filtre affiche tous les événements et l'heure à laquelle ils se sont produits
3. All Events, All Info : ce filtre affiche tous les événements et toutes les informations enregistrées. La différence principale entre ce filtre et « All Events and Time » est que cette option affiche le qualificatif « unique » pour la première occurrence d'un message particulier.

20.3.6 Identification des journaux des matches



A common task when working with the Driver Station Logs is to identify which logs came from competition matches. Logs which were taken during a match can now be identified using the *FMS* Connected event which will display the match type (Practice, Qualification or Elimination), match number, and the current time according to the FMS server. In this example, you can see that the FMS server time and the time of the Driver Station computer are fairly close, approximately 7 seconds apart.

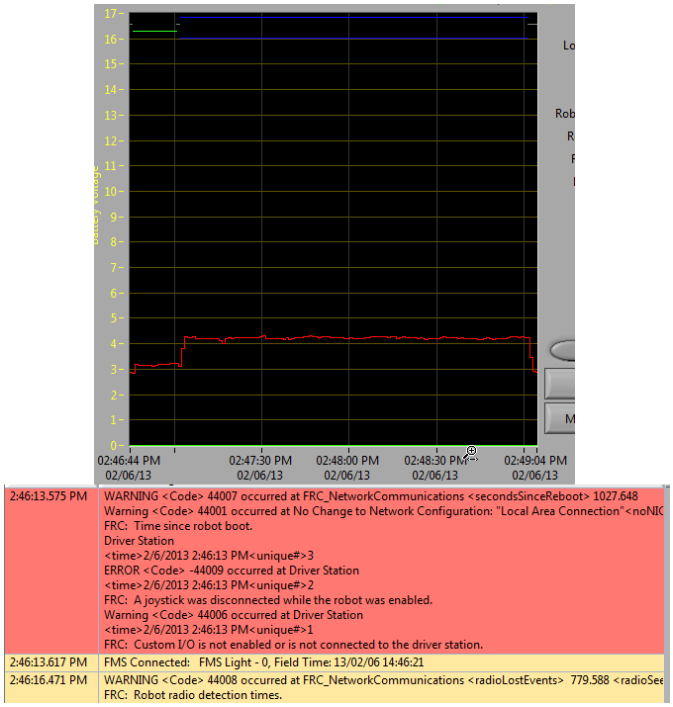
20.3.7 Identification avec le visionneur de journaux des problèmes de connexion courants

When diagnosing robot issues, there is no substitute for thorough knowledge of the system and a methodical debugging approach. If you need assistance diagnosing a connection problem at your events it is strongly recommended to seek assistance from your *FTA* and/or *CSA*. The goal of this section is to familiarize teams with how some common failures can manifest themselves in the DS Log files. Please note that depending on a variety of conditions a particular failure show slightly differently in a log file.

Note : Notez que tous les fichiers journaux affichés dans cette section ont été mis à l'échelle pour correspondre à la longueur d'un match à l'aide du bouton Match Length, puis parcourus jusqu'au début du Mode Autonome. En outre, la plateforme utilisée pour la production des journaux n'ayant pas été correctement câblée pour signaler la tension de la batterie, beaucoup de journaux ne contiennent pas d'informations sur la tension de la batterie.

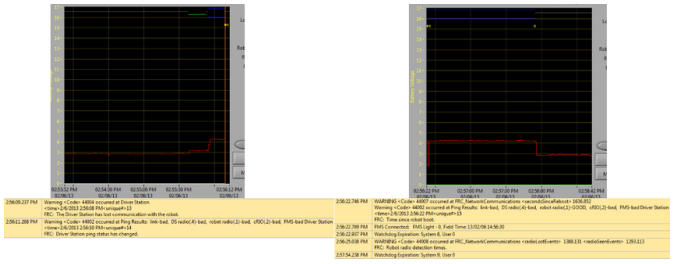
Astuce : Certains messages d'erreur trouvés dans le visionneur de journaux sont présentés ci-dessous et d'autres sont détaillés dans l'article *Erreurs/Avertissements de Driver Station*.

Journal « Normal »



Ceci représente un exemple du journal d'un match régulier. Les erreurs et les avertissements contenus dans la première zone proviennent du moment où DS a démarré et peuvent être ignorés. Ceci est confirmé en observant que ces événements se sont produits avant l'événement « FMS Connected : ». Le dernier événement montré peut également être ignoré, il est aussi lié à la première connexion du robot à DS (il se produit 3 secondes après la connexion au FMS) et se produit environ 30 secondes avant le début du match.

Déconnecté du FMS

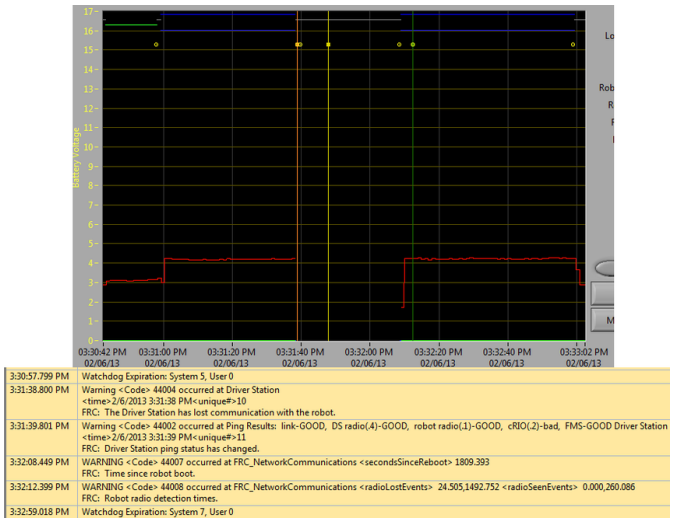


Lorsque DS se déconnecte du FMS, et donc du robot, pendant le match, le journal peut se subdiviser en morceaux. Les indicateurs clés de cette panne sont le dernier événement du premier journal, indiquant que la connexion au FMS est maintenant « bad » et le deuxième événement du 2ème journal qui est un nouveau message « FMS connected » suivi par DS transitant immédiatement vers Teleop Enabled. La cause la plus fréquente de ce type de défaillance est un câble ethernet sans griffe de verrouillage ou un port ethernet endommagé sur l'ordinateur de pilotage.

Redémarrage du roboRIO

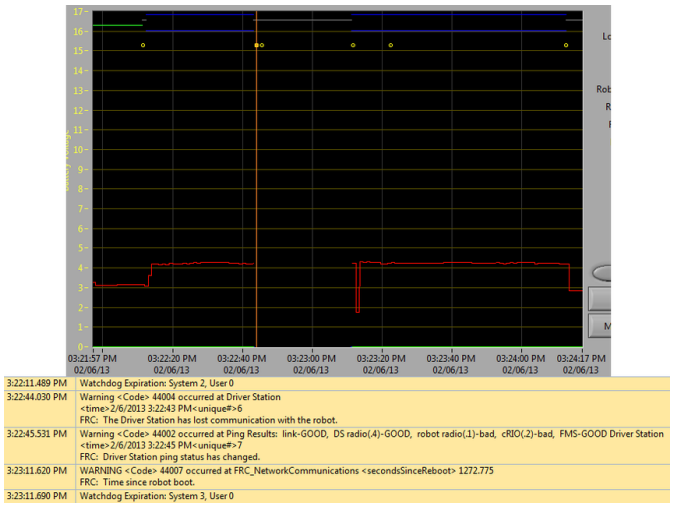
Le message « Time since robot boot » est le principal indicateur d'une défaillance de connexion causée par un redémarrage du roboRIO. Dans ce journal, DS perd la connexion avec le roboRIO à 3 :01 :36 tel qu'indiqué par le premier événement. Le deuxième événement indique que le ping initié après l'échec de la connexion a réussi avec tous les appareils excepté le roboRIO. À 3 :01 :47 le roboRIO commence à répondre à nouveau aux pings, un ping supplémentaire échoue à 3 :01 :52. À 3 :02 :02, Driver Station se connecte au roboRIO et le roboRIO rapporte qu'il a été en place pendant 3.682 secondes. Il s'agit d'un indicateur clair que le roboRIO a redémarré. Le code continue à se charger et à 3 :02 :24 le code signale une erreur de communication avec la caméra. Un avertissement est également signalé indiquant qu'aucun Code Robot n'est en cours d'exécution juste avant la fin du démarrage du programme.

Problème de câble Ethernet sur le robot



Un problème avec le câble ethernet sur le robot est principalement indiqué par un ping non concluant au roboRIO et l'apparition des événements *Radio Lost* et *Radio Seen* lorsque le roboRIO se reconnecte. Le message « Time since robot boot » lorsque le roboRIO se reconnecte indiquera également que le roboRIO n'a pas redémarré. Dans cet exemple, le câble Ethernet du robot a été déconnecté à 3 :31 :38. Le statut du ping indique que la radio est toujours connectée. Lorsque le robot se reconnecte à 3 :32 :08 l'évènement « Time since robot boot » est de 1809 secondes indiquant clairement que le roboRIO n'a pas redémarré. À 3 :32 :12, le robot indique qu'il a perdu la radio il y a 24.505 secondes et qu'il est revenu il y a 0.000 seconde. Ces points sont tracés sous forme de lignes verticales sur le graphique, jaune pour la radio perdue et vert pour la radio présente. Notez que les heures sont légèrement décalées par rapport aux événements réels tels qu'ils sont indiqués via la déconnexion et la connexion, mais ces heures aident à fournir des informations supplémentaires sur ce qui se passe.

Redémarrage de la radio



Un redémarrage de la radio du robot est généralement caractérisé par une perte de connexion avec la radio pendant environ 40-45 secondes. Dans cet exemple, la radio a brièvement perdu de la puissance à 3 :22 :44, ce qui l'a conduit à redémarrer. L'événement survenu à 3 :22 :45 indique que le ping à la radio a échoué. À 3 :23 :11, DS retrouve la communication avec le roboRIO et le roboRIO indique qu'il a été actif pendant 1272.775 secondes, excluant un redémarrage roboRIO. Notez que l'interrupteur réseau de la radio se rétablit très rapidement de sorte qu'une perte de puissance momentanée peut ne pas entraîner une séquence d'événements « radio lost »/ »radio seen ». Une perturbation plus longue peut entraîner l'enregistrement d'événements radio par DS. Dans ce cas, le facteur distinctif qui pointe vers un redémarrage radio est l'état ping de la radio dans DS. Si la radio se réinitialise, la radio sera inaccessible. Si le problème est un problème de câblage ou de connexion sur le robot, le ping radio doit demeure « GOOD ».

20.4 Erreurs/Avertissements de Driver Station

In an effort to provide both Teams and Volunteers (*FTA* / *CSA* / etc.) more information to use when diagnosing robot problems, a number of Warning and Error messages have been added to the Driver Station. These messages are displayed in the DS diagnostics tab when they occur and are also included in the DS Log Files that can be viewed with the Log File Viewer. This document discusses the messages produced by the DS (messages produced by WPILib can also appear in this box and the DS Logs).

20.4.1 Manette débranchée

```
ERROR<Code>-44009 occurred at Driver Station
<time>2/5/2013 4:43:54 PM <unique#>1
FRC: A joystick was disconnected while the robot was enabled.
```

Cette erreur est déclenchée lorsqu'une manette est débranchée. Contrairement au texte du message, cette erreur s'affichera même si le robot n'est pas activé, ou même connecté à DS. Vous verrez une seule instance de ce message se produire chaque fois que Driver Station est démarré, même si les manettes sont correctement connectées et fonctionnent.

Note : Joystick Unplugged warnings can be silenced by calling `DriverStation.silenceJoystickConnectionWarning(true)` (Java, C++)

20.4.2 Communication perdue

```
Warning<Code>44004 occurred at Driver Station
<time>2/6/2013 11:07:53 AM<unique#>2
FRC: The Driver Station has lost communication with the robot.
```

Ce message d'avertissement s'affiche à chaque fois que Driver Station perd la communication avec le robot (indicateur de communication passant du vert au rouge). Une seule instance de ce message s'affiche lorsque DS démarre, avant que la communication ne soit établie.

20.4.3 Statut ping

```
Warning<Code>44002 occurred at Ping Results: link-GOOD, DS radio(.4)-bad, robot␣
↪radio(.1)-GOOD, cRIO(.2)-bad, FMS- bad Driver Station
<time>2/6/2013 11:07:59 AM<unique#>5
FRC: Driver Station ping status has changed.
```

A Ping Status warning is generated each time the Ping Status to a device changes while the DS is not in communication with the roboRIO. As communications is being established when the DS starts up, a few of these warnings will appear as the Ethernet link comes up, then the connection to the robot radio, then the roboRIO (with *FMS* mixed in if applicable). If communications are later lost, the ping status change may help identify at which component the communication chain broke.

20.4.4 Temps écoulé depuis le démarrage du robot

```
WARNING<Code>44007 occurred at FRC_NetworkCommunications
**<secondsSinceReboot> 3.585**
FRC: Time since robot boot.
```

Ce message s'affiche à chaque fois que DS commence à communiquer avec le roboRIO. Le message indique la durée de fonctionnement, en secondes, du roboRIO et peut être utilisé pour déterminer si une perte de communication était due à un redémarrage du roboRIO.

20.4.5 Temps de détection de la radio

```
WARNING<Code>44008 occurred at FRC_NetworkCommunications
<radioLostEvents> 19.004<radioSeenEvents> 0.000
FRC: Robot radio dectection times

WARNING<Code>44008 occurred at FRC_NetworkCommunications
<radioLostEvents> 2.501,422.008<radioSeenEvents> 0.000,147.005
FRC: Robot radio dectection times.
```

Ce message peut s'afficher lorsque DS commence à communiquer avec le roboRIO et indique le temps, en secondes, depuis la dernière fois que la radio a été perdue et vue. Dans le premier exemple, l'image au-dessus du message indique que la connexion du roboRIO avec la radio a été perdue 19 secondes avant l'impression du message et que la radio a été revue à nouveau juste lorsque le message a été imprimé. Si plusieurs événements radioLost ou radioseen se sont produits depuis le démarrage du roboRIO, jusqu'à 2 événements de chaque type seront inclus, séparés par des virgules.

20.4.6 Pas de programme sur le robot

```
Warning<Code>44003 occurred at Driver Station
<time>2/8/2013 9:50:13 AM<unique#>8
FRC: No robot code is currently running.
```

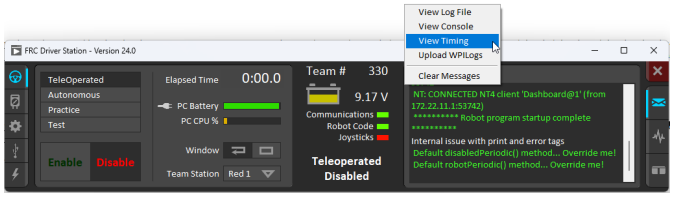
Ce message s'affiche lorsque DS commence à communiquer avec le roboRIO, mais ne détecte aucun code Robot en cours d'exécution. Une seule instance de ce message sera émis si Driver Station est déjà ouvert et en cours d'exécution pendant que le roboRIO est en train de démarrer tandis que DS commencera la communication avec le roboRIO avant la fin du chargement du programme Robot.

20.5 Driver Station Timing Viewer

The 2024 Driver Station has a a new window to help diagnose robot control issues.

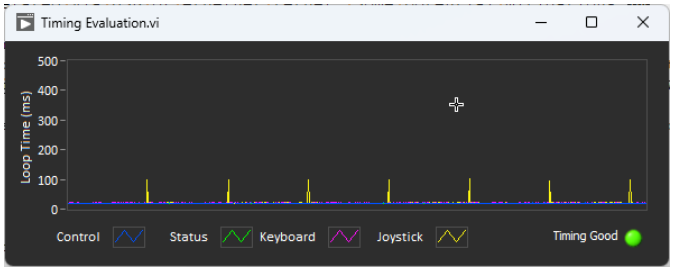
20.5.1 Opening the Timing Viewer

To start the Driver Station Timing Viewer, select the gear icon and then select *View Timing*.

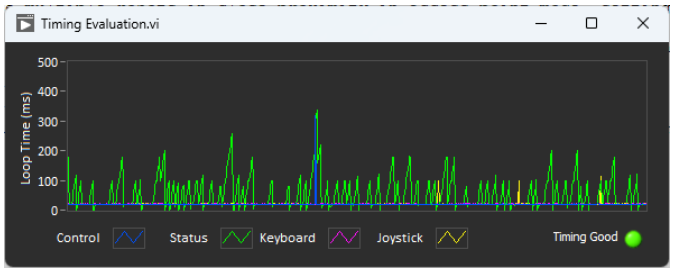


20.5.2 Viewing Timing

The Timing viewer shows the timing of the driver station loops measuring the joysick, keyboard, and control and status network packets. When timing is good, all values should be close to 0 ms. This can help diagnose what is causing robot control issues.



The next image shows what it looks like when network congestion causes network packets to be delayed and combined. In this example, the communication was so bad that the robot wouldn't stay enabled or connected for more then a second.



20.6 Programmer la radio pour FMS Offseason

When using the *FMS* Offseason software, the typical networking setup is to use a single access point with a single SSID and WPA key. This means that the radios should all be programmed to connect to this network, but with different IPs for each team. The Team version of the FRC® Bridge Configuration Utility has an FMS Offseason mode that can be used to do this configuration.

20.6.1 Prérequis

Installez l'utilitaire FRC® Radio Configuration Utility selon les instructions accessibles dans *Programmation de votre radio*

Avant de commencer à utiliser le logiciel :

1. Désactiver les connexions WiFi sur votre ordinateur, car ils peuvent empêcher l'utilitaire de configuration de communiquer correctement avec la passerelle
2. Branchez directement à partir de votre ordinateur dans le port Ethernet du pont sans fil le plus proche de la prise d'alimentation. Assurez-vous qu'aucun autre appareil n'est connecté à votre ordinateur via le port Ethernet. Si vous alimentez la radio via le port PoE, branchez un câble Ethernet du PC sur le côté de la prise de l'adaptateur PoE (où le roboRIO se brancherait). Si vous rencontrez des problèmes de configuration via l'adaptateur PoE, vous pouvez essayer de connecter le PC à l'autre port de la radio.

Configuration programmée

L'utilitaire de configuration radio, pendant son exécution, programme un certain nombre de paramètres de configuration dans la radio. Ces paramètres s'appliquent à la radio dans tous les modes (y compris lors d'événements). Il s'agit notamment de :

- Configurez une IP statique de `10.TE.AM.1`
- Définissez une adresse IP alternative du côté relié par fil de `192.168.1.1` pour la programmation future
- Relier en pont les ports câblés afin qu'ils puissent être utilisés de façon interchangeable
- La DEL témoin de la configuration notée dans le voyant d'état référencé ci-dessous.
- Limiter la bande passante à 4 Mo/s du côté sortie de l'interface sans fil (ceci peut être désactivé pour une utilisation à la maison)
- La qualité de service ou QS régularise la hiérarchisation interne des paquets (affecte la mémoire tampon interne et les paquets à éliminer si la limite de bande passante est atteinte). Les règles appliquées sont les suivantes :
  - Contrôle et état du robot (UDP `1110`, `1115`, `1150`)
  - Robot TCP & *NetworkTables* (TCP `1735`, `1740`)
  - En vrac (Tout autre trafic). (désactivé si la limite de largeur de bande (BW) est désactivée)
- *DHCP* server enabled. Serves out :
  - `10.TE.AM.11` - `10.TE.AM.111` du côté relié par fil
  - `10.TE.AM.138` - `10.TE.AM.237` du côté sans-fil
  - Masque Subnet de `255.255.255.0`
  - Adresse de diffusion `10.TE.AM.255`
- Serveur DNS activé. L'adresse IP du serveur DNS et le suffixe de domaine (`.lan`) sont servis dans le cadre du DHCP.

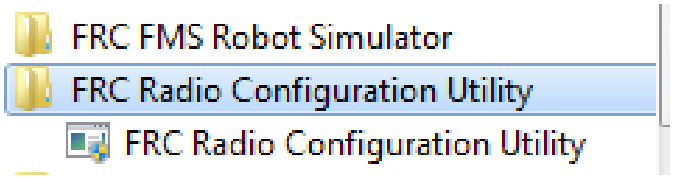
Astuce : Voir la :ref :` Référence sur les témoins d'état <docs/hardware/hardware-basics/status-lights-ref :OpenMesh Radio>` pour plus de détails sur le comportement des lumières d'indication d'état de la radio lorsqu'elle est configurée.

Lorsqu'ils sont programmés avec la version d'équipe de l'utilitaire Radio Configuration Utility, les comptes utilisateurs seront laissés (ou configurés) au défaut **pour les DAP uniquement** :

- Username : root
- Password : root

Note : Il n'est pas recommandé de modifier la configuration manuellement

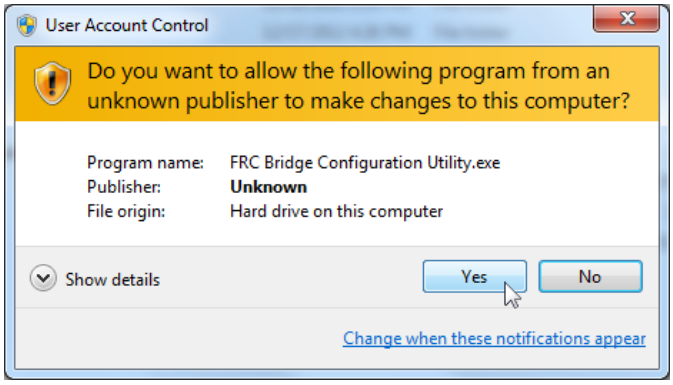
20.6.2 Lancer le logiciel



Utilisez le menu Démarrer ou le raccourci de bureau pour lancer le programme.

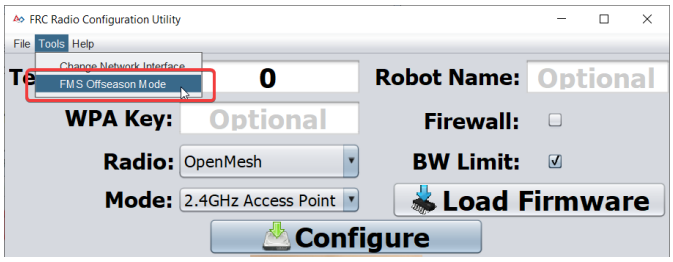
Note : Si vous avez besoin de localiser le programme, il est installé dans `C:\Program Files (x86)\FRC Radio Configuration Utility`. Pour les machines 32 bits, le chemin est `C:\Program Files\FRC Radio Configuration Utility`

20.6.3 Autoriser le programme à apporter des modifications, si vous y êtes invité



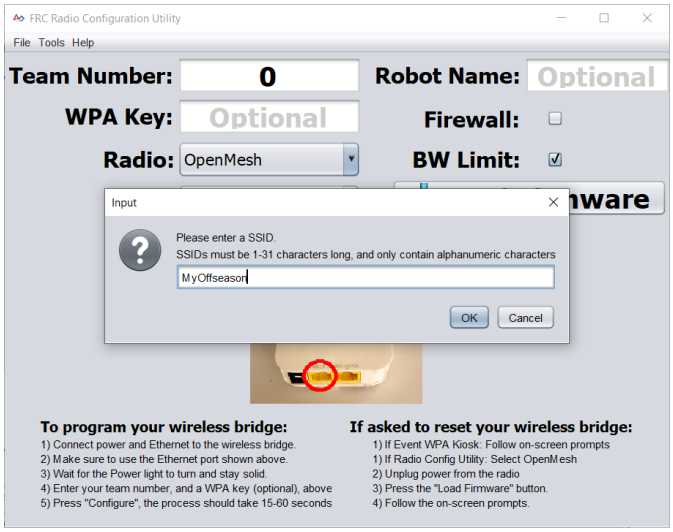
Une invite peut s'afficher pour autoriser l'utilitaire de configuration à apporter des modifications à l'ordinateur. Cliquez sur *Yes* si l'invite s'affiche.

20.6.4 Entrez en mode FMS Offseason



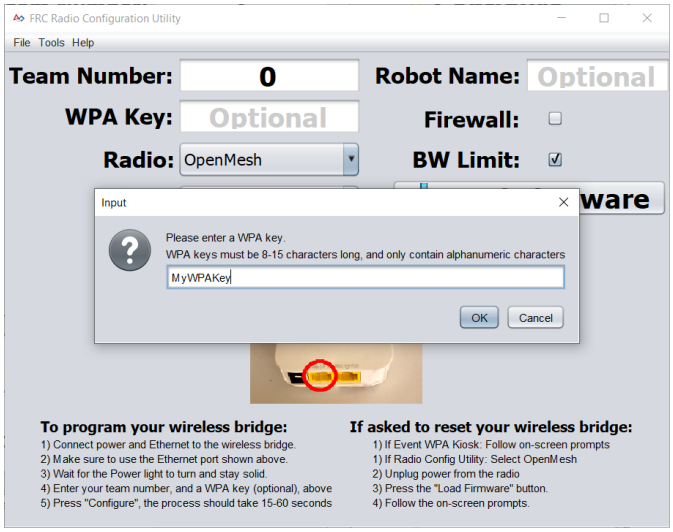
Cliquez sur ""Tools"" -> ""FMS-Lite Mode"" pour entrer en mode FMS-Lite.

20.6.5 Entrer le SSID



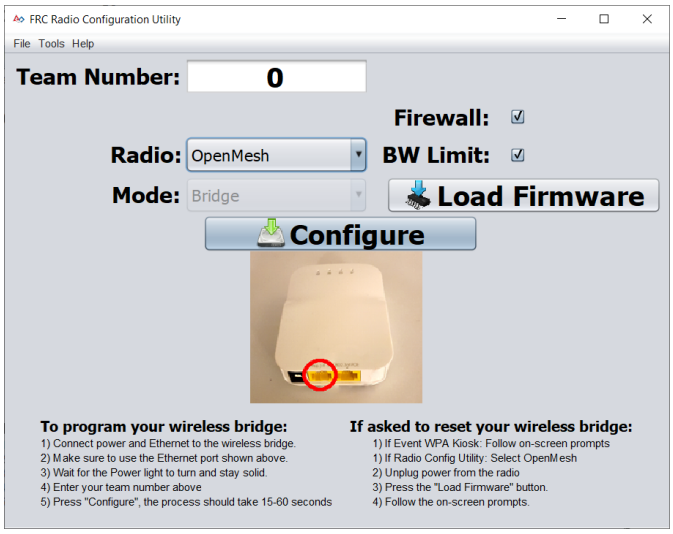
Entrez le SSID (nom) de votre réseau sans fil dans la zone et cliquez sur OK.

20.6.6 Entrer la clé WPA



Entrez la clé WPA de votre réseau dans la zone et cliquez sur OK. Laissez la boîte vide si vous utilisez un réseau non sécurisé.

20.6.7 Programmer les radios



Le kiosque est maintenant prêt à programmer autant de radios qu'on veut pour se connecter au réseau configuré. Pour programmer chaque radio, connectez la radio au kiosque, définissez le numéro d'équipe dans la zone, puis cliquez sur Configure.

Le kiosque programmera les radios OpenMesh, D-Link Rev A ou D-Link Rev B pour travailler sur un réseau FMS hors saison en sélectionnant l'option appropriée dans la liste déroulante « Radio ».

Note : Les limitations de bande passante et QoS ne seront pas configurées sur les radios D-Link dans ce mode.

20.6.8 Modification du SSID ou de la clé

Si vous entrez incorrectement un paramètre ou si vous devez modifier la clé SSID ou WPA, accédez au menu Tools et cliquez sur mode FMS-Lite pour sortir le kiosque du Mode FMS-Lite. Lorsque vous cliquez à nouveau pour remettre le kiosque en Mode FMS-Lite, vous serez ré-invité pour le SSID et la clé.

20.6.9 Dépannage

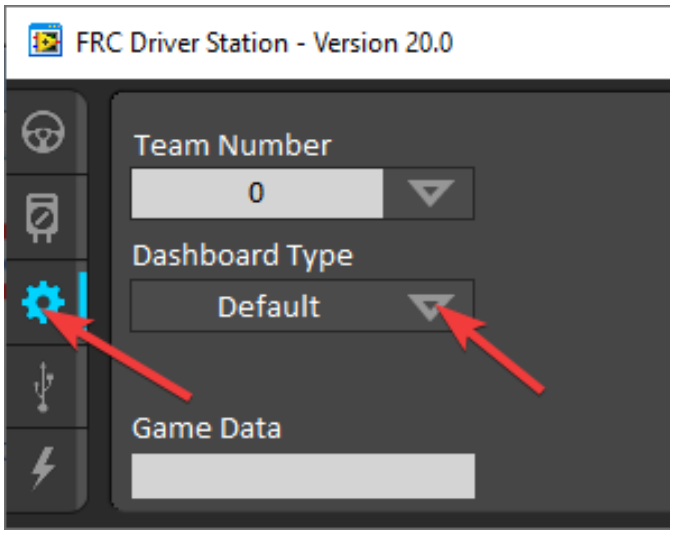
Consultez les étapes de dépannage dans la section *Programmation de votre radio*

20.7 Configuration manuelle de Driver Station pour lancer un Dashboard personnalisé

Note : Si WPILib n'est pas installé à l'emplacement par défaut (par exemple lorsque les fichiers sont copiés manuellement sur un PC), le Dashboard choisi peut ne pas être lancé correctement. Pour que le DS lance un Dashboard personnalisé lorsqu'il démarre, vous devez modifier manuellement les paramètres du Dashboard par défaut.

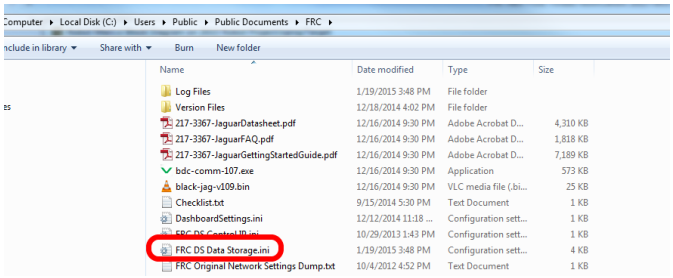
Avertissement : Ceci n'est pas nécessaire pour la plupart des installations, essayez d'abord d'utiliser le *type de Dashboard* défini pour votre langage de programmation.

20.7.1 Mettre la Driver Station sur Default



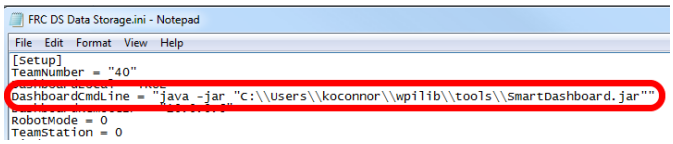
Ouvrez l'application Driver Station, cliquez sur l'onglet Setup et configurez le paramètre Dashboard à Default. **Fermez ensuite Driver Station !**

20.7.2 Ouvrir le fichier de stockage de données de la DS



Parcourez `C:\Users\Public\Documents\FRC` et cliquez deux fois sur `FRC DS Data Storage` pour l'ouvrir.

20.7.3 DashboardCmdLine



Localisez la ligne commençant par `DashboardCmdLine`. Modifiez-la pour qu'elle pointe vers le tableau de bord à lancer lorsque le DriverStation démarre

Dashboard personnalisé LabVIEW

Remplacez la chaîne après = avec `"C:\\PATH\\TO\\DASHBOARD.exe"` où le chemin spécifié est le chemin du fichier `exe` du Dashboard. Enregistrez le fichier `FRC DS Data Storage`.

Dashboard Java

Remplacez la chaîne de caractères après = par `java -jar "C:\\PATH\\TO\\DASHBOARD.jar"` où le chemin d'accès spécifié est le chemin d'accès au Dashboard `jar` file. Enregistrez le fichier `FRC DS Data Storage`.

Astuce : Shuffleboard et Smartdashboard nécessitent Java 11.

Dashboard du programme d'installation de WPILib

Remplacez la chaîne après = with `wscript "C:\\Users\\Public\\wpilib\\YYYY\\tools\\DASHBOARD.vbs"` où `YYYY` est l'année et `DASHBOARD.vbs` est soit `Shuffleboard.vbs` ou `Smartdashboard.vbs`. Enregistrez le fichier `FRC DS Data Storage`.

20.7.4 Lancer Driver Station

Driver Station devrait maintenant lancer l'application Dashboard configurée chaque fois qu'il est ouvert.

21 RobotBuilder

21.1 RobotBuilder - Une introduction

21.1.1 Aperçu de RobotBuilder

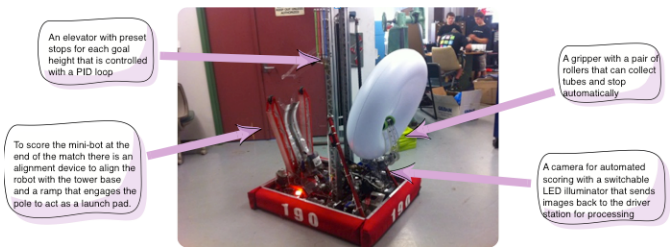
RobotBuilder est une application conçue pour faciliter le processus de développement du robot. RobotBuilder peut vous aider :

- Générer du code standard.
- Organisez votre robot et déterminez quels sont ses sous-systèmes clés.
- Vérifiez que vous avez suffisamment de canaux pour tous vos capteurs et actuateurs.
- Générer des diagrammes de câblage.
- Modifiez facilement votre interface opérateur.
- Plus…

La création d'un programme avec RobotBuilder est une procédure très simple en suivant quelques étapes qui sont les mêmes pour n'importe quel robot. Cette leçon décrit les étapes que vous pouvez suivre. Vous trouverez plus de détails sur chacune de ces étapes dans les sections suivantes du document.

Note : RobotBuilder génère du code à l'aide de la nouvelle infrastructure logicielle orientée commande . Pour plus de détails sur la nouvelle infrastructure logicielle, voir *Programmation orientée commande*.

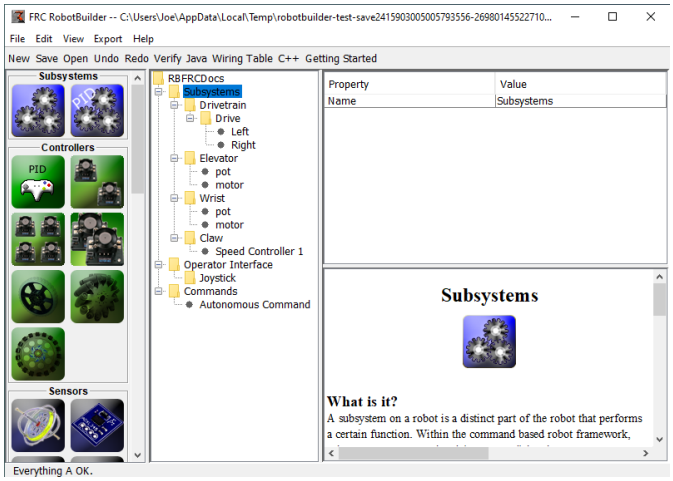
Diviser le robot en sous-systèmes



Votre robot est naturellement composé d'un certain nombre de systèmes plus petits tels que les bases pilotables, les bras, les tireurs, les collectionneurs, les manipulateurs, les poignets articulés, etc. Dans la conception de votre robot, pensez à diviser celui-ci en sous-systèmes plus petits et opérés séparément. Dans cet exemple particulier, il y a un élevateur, un dispositif d'alignement de minibot, une pince, et un système de caméra. Sans oublier la base pilotable. Chacune de ces parties du robot est contrôlée séparément et serait donc de bons candidats pour former les sous-systèmes de ce robot.

Pour plus d'informations, voir *Creating a Subsystem*.

Ajout de chaque sous-système au projet



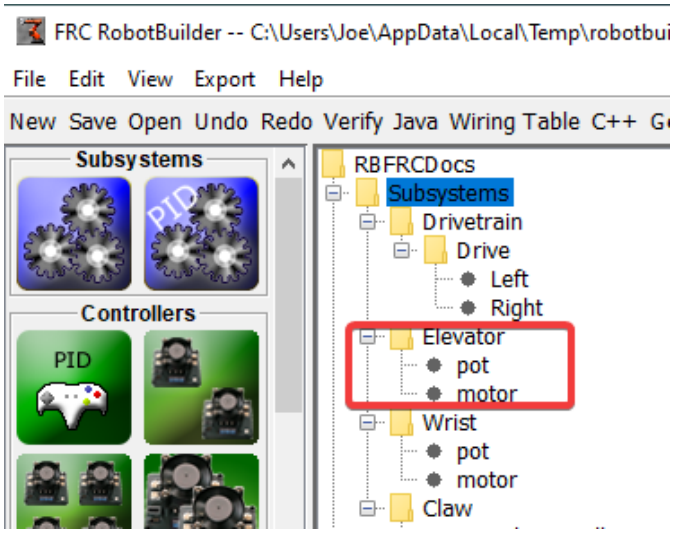
Chaque sous-système sera ajouté au dossier « Subsystems » dans le RobotBuilder et donné un nom significatif. Pour chacun des sous-systèmes, plusieurs attributs sont remplis pour des informations plus précises sur les sous-systèmes. En outre, il existe deux types de sous-systèmes d'intérêt :

1. PIDSubsystems - souvent il est souhaitable de contrôler l'opération d'un sous-systèmes à l'aide un contrôleur PID. Il s'agit de code dans votre programme qui permet d'amener un paramètre du sous-système, par exemple l'angle du sous-système bras, plus rapidement à une position désirée, puis d'arrêter la rotation lorsque cette position est atteinte. Les sous-systèmes PIDSubsystems ont un code de contrôleur PID intégré et sont souvent plus pratiques à utliser directement plutôt que d'écrire et d'ajouter vous-mêmes du code pour votre contrôleur PID. Les sous-systèmes PIDSubsystems ont un capteur qui détermine quand le dispositif a atteint la position cible et un actionneur (contrôleur de moteur) qui est conduit au point de consigne.
2. Sous-système régulier - ces sous-systèmes ne disposent pas d'un contrôleur PID intégré et sont utilisés pour les sous-systèmes sans contrôle PID pour la rétroaction ou pour les sous-systèmes nécessitant un contrôle plus complexe que ce qui peut être manipulé avec le contrôleur pid incorporé par défaut.

À mesure que vous avancerez dans la lecture de cette documentation, les différences entre les types de sous-systèmes vous apparaîtront plus évidentes.

Pour plus d'informations, voir *Creating a Subsystem* and *Writing Code for a Subsystem*.

Ajout de composants à chacun des sous-systèmes



Chaque sous-système se compose d'un certain nombre d'actionneurs, de capteurs et de contrôleurs qu'il utilise pour effectuer ses opérations. Ces capteurs et actionneurs sont ajoutés au sous-système auquel ils sont associés. Chacun des capteurs et actionneurs provient de la palette RobotBuilder et est traîné vers le sous-système approprié. Pour chacune d'elles, il y a généralement d'autres propriétés qui doivent être définies, telles que les numéros de port et d'autres paramètres spécifiques au composant.

Dans cet exemple, il y a un sous-système Elevateur qui utilise un moteur et un potentiomètre (moteur et pot) qui ont été déplacés vers le sous-système Elevateur.

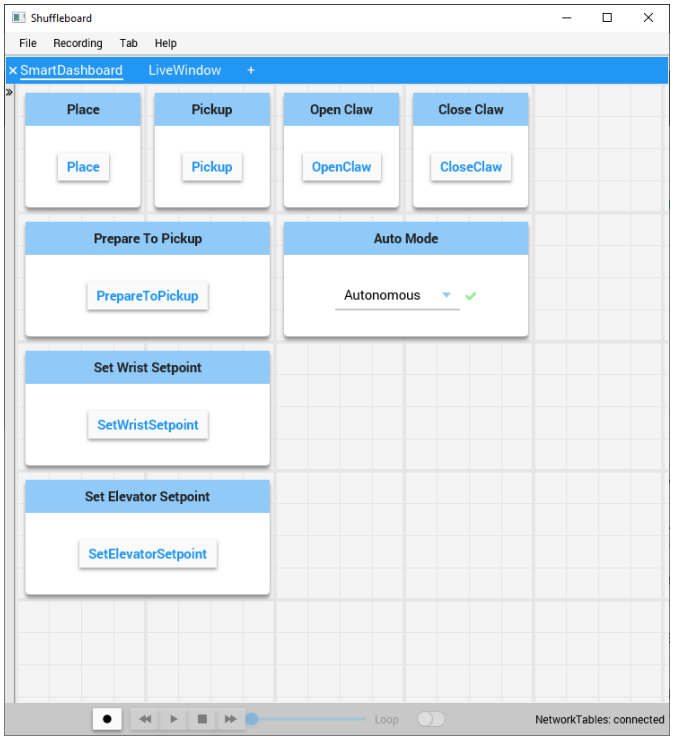
Ajout de commandes qui décrivent les objectifs du sous-système

Les commandes constituent les différents objectifs que le robot réalisera. Ces commandes sont ajoutées en faisant glisser la commande sous le dossier « Commands ». Lors de la création d'une commande, il y a 7 choix (indiqués sur la palette à gauche de l'image) :

- Normal commands - ce sont les commandes les plus flexibles, vous devez écrire tout le code pour effectuer les actions souhaitées et nécessaires pour atteindre l'objectif.
- Timed commands - ces commandes sont une version simplifiée d'une commande qui se termine après un délai d'attente ou timeout
- Instant commands - these commands are a simplified version of a command that runs for one iteration and then ends
- Command groups - ces commandes sont une combinaison d'autres commandes pouvant s'exécuter aussi bien dans un ordre séquentiel et que parallèle. Vous pouvez les utiliser pour créer des actions plus compliquées à partir d'un certain nombre de commandes de base.
- Setpoint commands - Les commandes valeur de consigne déplacent un sous-système PID vers une consigne fixe ou l'emplacement souhaité.
- PID commands - ces commandes ont un contrôleur PID intégré à utiliser avec un sous-système régulier.
- Conditional commands - ces commandes sélectionnent l'une des deux commandes à exécuter au moment de l'initialisation.

Pour plus d'informations, voir Creating a Command 1 et *Writing Command Code*.

Tester chaque commande

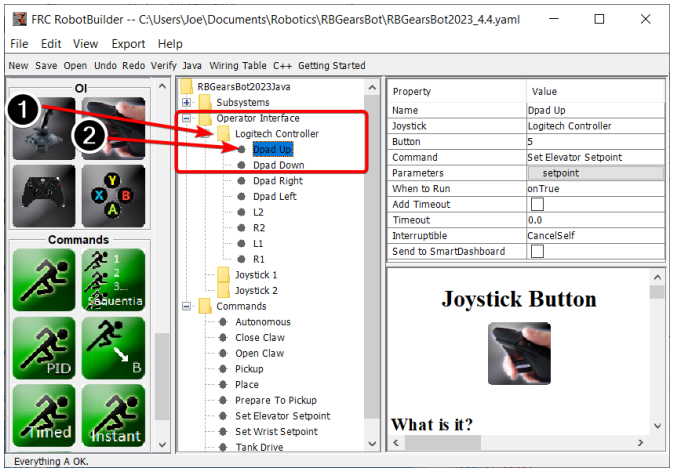


Chaque commande peut être exécutée à partir du Shuffleboard ou du SmartDashboard. Ceci est utile pour tester les commandes avant de les ajouter à l'interface de l'opérateur ou à un groupe de commandes. Tant que vous laissez la propriété « Button on SmartDashboard » cochée, un bouton sera créé sur le SmartDashboard. Lorsque vous appuyez sur le bouton de démarrage de la commande, la commande s'exécute et vous pouvez vérifier qu'elle effectue l'action souhaitée.

En créant des boutons, chaque commande peut être testée individuellement. Si toutes les commandes fonctionnent individuellement, vous pouvez être assez sûr que le robot fonctionnera dans son ensemble.

Pour plus d'informations, *Test avec Smartdashboard*.

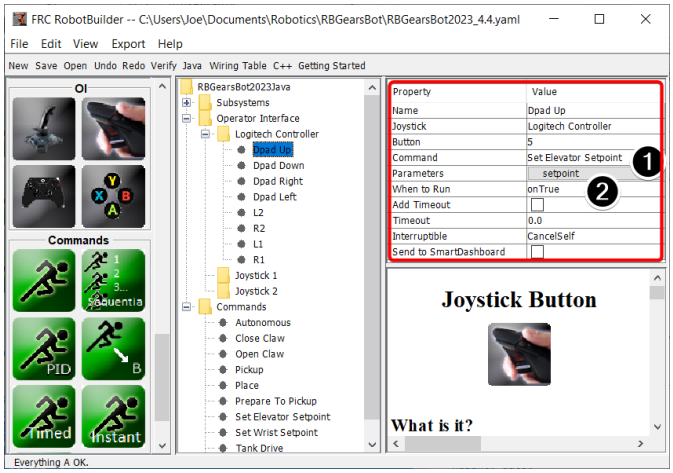
Ajout de composants de l'Interface Opérateur



L'interface Opérateur se compose de joysticks, de gamepads et d'autres dispositifs d'entrée HID. Vous pouvez ajouter des composants d'interface opérateur (joysticks, boutons joystick) à votre programme dans RobotBuilder. Celui-ci générera automatiquement du code qui initialisera tous les composants et leur permettra d'être connectés aux commandes.

Les composants de l'interface Opérateur sont déplacés de la palette vers le dossier »Operator Interface » dans le programme RobotBuilder. D'abord (1) ajouter des joysticks au programme, puis mettre des boutons sous les joysticks associés (2) et leur donner des noms significatifs, comme ShootButton.

Connection des commandes à l'Interface de Opérateur

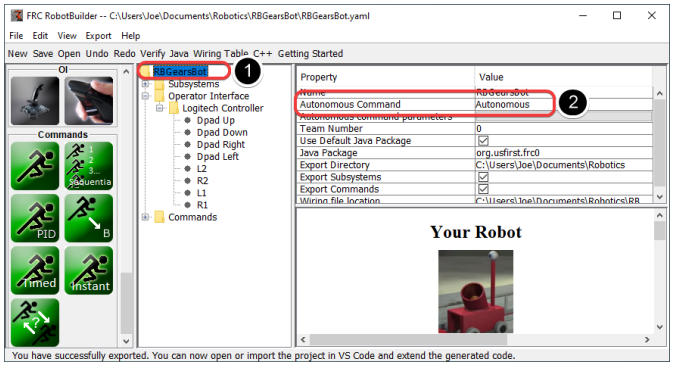


Les commandes peuvent être associées à des boutons de sorte que lorsqu'un bouton est appuyé sur la commande est cédulée pour exécution. Cela devrait, pour la plupart du temps, gérer une bonne portion de la partie téléopérée de votre programme de robot.

Cela se fait simplement en ajoutant (1) la commande à l'objet JoystickButton dans le programme RobotBuilder, puis (2) en définissant la condition dans laquelle la commande est planifiée.

Pour plus d'informations, voir *Connecting the Operator Interface to a Command*.

Développer des commandes autonomes

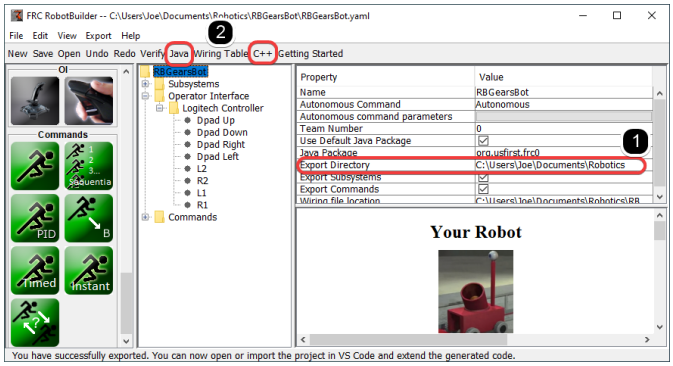


Les commandes facilitent le développement de programmes autonomes. Il vous suffit de spécifier quelle commande doit s'exécuter lorsque le robot entre dans la période autonome et elle sera automatiquement planifiée. Si vous avez testé les commandes tel que discuté ci-dessus, il s'agit simplement de choisir la commande qui doit s'exécuter.

Sélectionnez le robot à la racine du projet RobotBuilder (1), puis modifiez la propriété (2) Autonomous Command pour choisir la commande à exécuter. C'est aussi simple que ça !

Pour plus d'informations, voir *Setting the Default Autonomous Command*.

Générer le code



À tout moment du processus décrit ci-dessus, RobotBuilder peut vous générer un programme C++ ou Java qui représentera le projet que vous avez créé. Ceci se fait en spécifiant l'emplacement du projet dans les propriétés du projet (1), puis en cliquant sur le bouton de la barre d'outils appropriée pour générer le code (2).

For more information see *Generating RobotBuilder Code*.

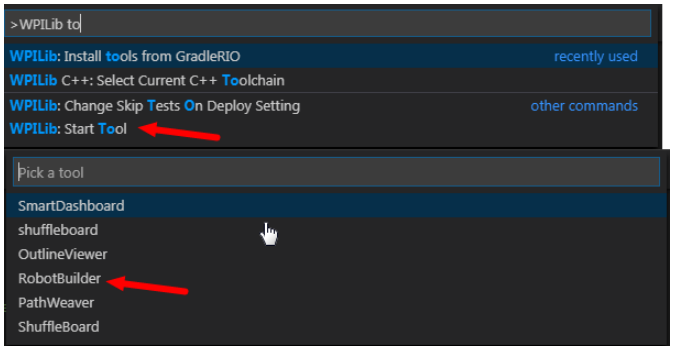
21.1.2 Le démarrage de RobotBuilder

Note : RobotBuilder est un programme Java et devrait pouvoir fonctionner sur n'importe quelle plate-forme qui supporte Java. Nous avons exécuté RobotBuilder sur macOS, Windows et diverses versions de Linux avec succès.

Obtenir une copie de RobotBuilder

RobotBuilder est téléchargé et installé par l'utilitaire WPILib Offline Installer. Pour plus d'informations, consultez le document *Windows/macOS/Linux installation guides*

Option 1 - Démarrer à partir du code Visual Studio

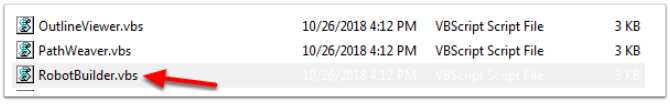


Appuyez sur `Ctrl+Shift+P` et tapez « WPILib » ou cliquez sur le logo WPILib en haut à droite pour lancer la palette de commande WPILib. Sélectionnez *Start Tool*, puis sélectionnez *Robot Builder*.

Option 2 - Raccourcis

Shortcuts are installed to the Windows Start Menu and the `2024 WPILib Tools` folder on the desktop.

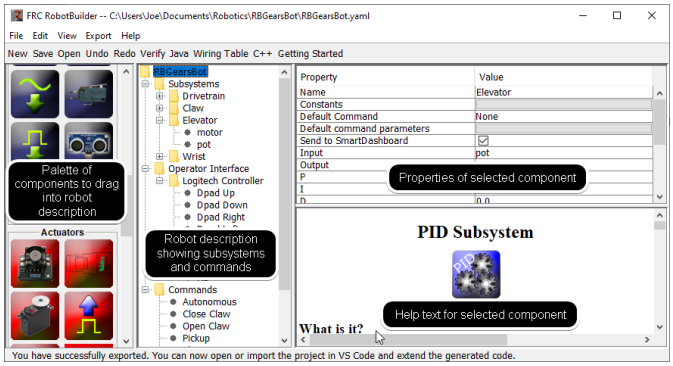
Option 3 - Exécution à partir du Script



Le processus d'installation installe les outils dans `~/wpilib/YYYY/tools` (où YYYY est l'année et `~` est `C:\Users\Public` sous Windows).

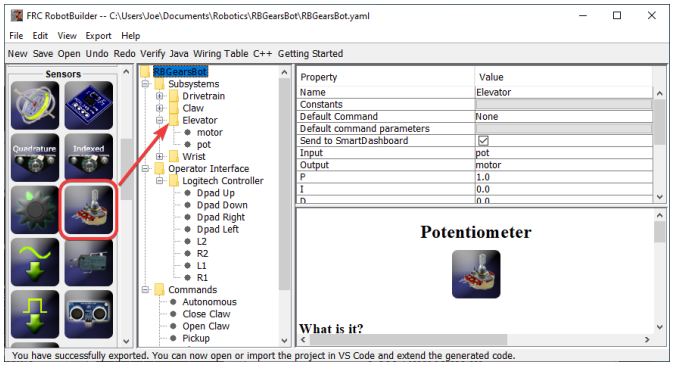
À l'intérieur de ce dossier, vous trouverez des fichiers d'extension `.vbs` (Windows) et d'extension `.py` (macOS/Linux) que vous pouvez utiliser pour lancer chaque outilitaire. Ces scripts permettent de lancer les outilitaires à l'aide de la version JDK adéquate et sont ceux que vous devez utiliser pour lancer les outilitaires.

21.1.3 L'interface utilisateur de RobotBuilder



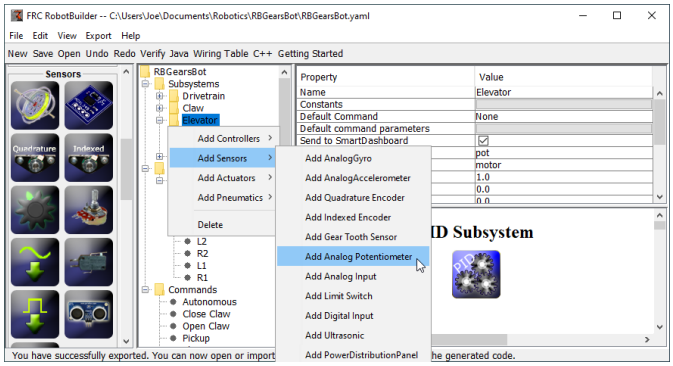
RobotBuilder possède une interface utilisateur conçue pour le développement rapide de programmes de robot. Presque toutes les opérations sont effectuées avec la souris, par des gestes de « glisser-déposer » ou en sélectionnant des options dans les listes déroulantes.

Glissement des éléments de la palette vers la description du robot



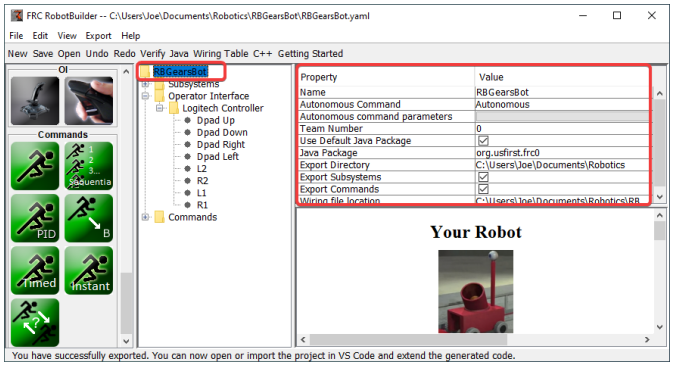
Vous pouvez faire glisser des éléments de la palette vers la description du robot en commençant le glisser sur l'élément de la palette et en terminant sur le conteneur où vous souhaitez que l'élément se trouve. Dans cet exemple, déposer un Potentiometer dans le sous-système Elevator.

Ajout de composants à l'aide du menu contextuel du clic droit



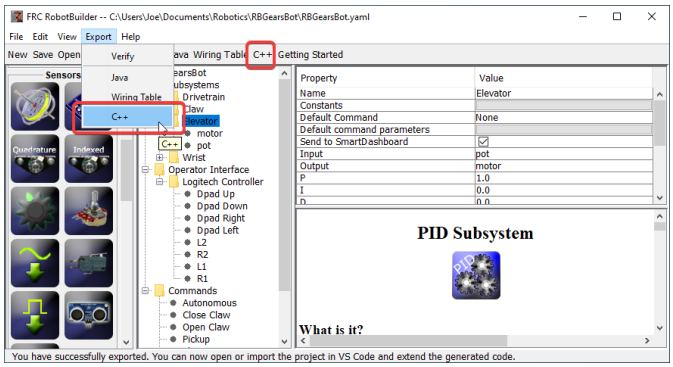
Une méthode de raccourci pour ajouter des éléments à la description du robot consiste à cliquer avec le bouton droit sur l'objet conteneur (ascenseur) et à sélectionner l'élément à ajouter (potentiomètre). Ceci est identique à l'utilisation du glisser-déposer, mais pourrait être plus facile pour certaines personnes.

Modification des propriétés des éléments de description du robot



Les propriétés d'un élément sélectionné apparaissent dans le visualiseur de propriétés. Les propriétés peuvent être modifiées en sélectionnant la valeur dans la colonne de droite.

Utilisation du système de menus



Les opérations pour RobotBuilder peuvent être sélectionnées via le système de menus ou son élément équivalent (s'il est disponible) dans la barre d'outils.

21.1.4 Configuration du projet Robot

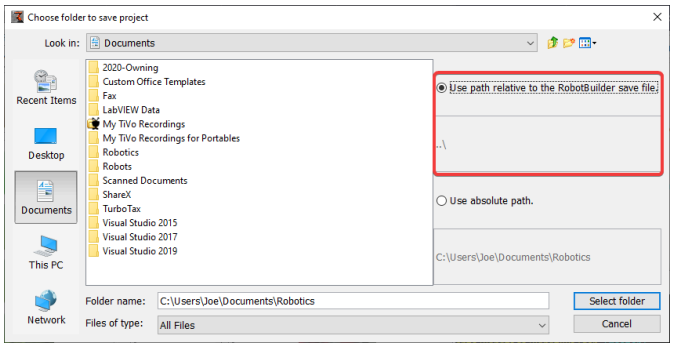
Le programme RobotBuilder possède certaines propriétés par défaut qui doivent être configurées pour que le programme et les autres fichiers générés fonctionnent correctement. Ces informations de configuration sont stockées dans les propriétés de description du robot (la première ligne).

Propriétés du projet de robot

Les propriétés qui décrivent le robot sont :

- **Name** - Le nom du projet de robot qui est créé
- **Autnomous Command** - la commande qui s'exécutera par défaut lorsque le programme sera placé en mode autonome
- **Autonomous Command Parameters** - Paramètres de la commande autonome
- **Team Number** : Ce paramètre définit le numéro d'équipe du projet, qui sera utilisé pour localiser le robot pendant le déploiement du code.
- **Use Default Java Package** - Si cette case est cochée, RobotBuilder utilisera le package par défaut (frc.robot). Sinon, vous pouvez spécifier un nom de package personnalisé à utiliser.
- **Java Package** - Le nom du package Java généré utilisé lors de la génération du code de projet
- ** Export Directory ** - Le dossier dans lequel le projet est généré lorsque l'option Export to Java ou C++ est sélectionnée
- **Export Subsystems** - Vérifié si RobotBuilder doit exporter les classes Subsystem de votre projet
- **Export Commands** - Vérifié si RobotBuilder doit exporter les classes de commandes de votre projet
- **Wiring File location** - l'emplacement du fichier html à générer qui contient le schéma de câblage de votre robot
- **Desktop Support** - Permet de faire les tests unitaires et la simulation. Bien que WPILib soit pourvue cette fonctionnalité, les librairies logicielles tierces, elles, peuvent ne pas l'avoir. Si les libairies ne prennent pas en charge les applications de bureau, votre code peut ne pas se compiler ou planter. Cette option ne doit pas être cochée, sauf si des tests unitaires ou une simulation sont nécessaires et que toutes les librairies la prennent en charge.

Utilisation d'un contrôle de code source avec le projet RobotBuilder

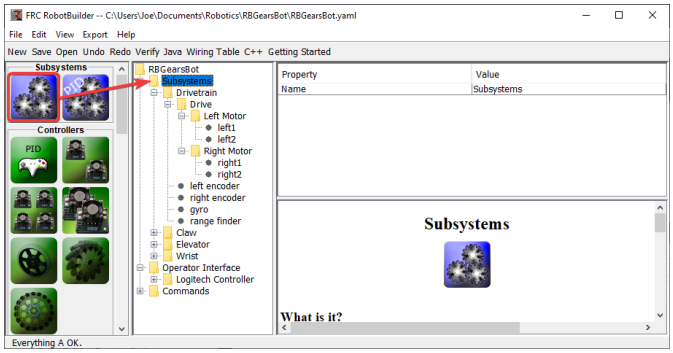


Lorsque vous utilisez un contrôle de code source, le projet sera généralement utilisé sur un certain nombre d'ordinateurs et le chemin d'accès (Path) au répertoire du projet peut être différent d'un ordinateur utilisateur à un autre. Si le fichier de projet RobotBuilder est stocké à l'aide d'un Path absolu, il contiendra généralement le nom d'utilisateur et ne sera pas utilisable sur plusieurs ordinateurs. Pour que cela fonctionne, sélectionnez « relative path » et spécifiez le Path comme un répertoire décalé par rapport aux fichiers du projet. Dans l'exemple ci-dessus, le fichier de projet est stocké dans le dossier juste au-dessus des fichiers de projet dans la hiérarchie de fichiers. Dans ce cas, le nom d'utilisateur ne fait pas partie du chemin d'accès et il sera portable sur tous vos ordinateurs.

21.1.5 Création d'un sous-système

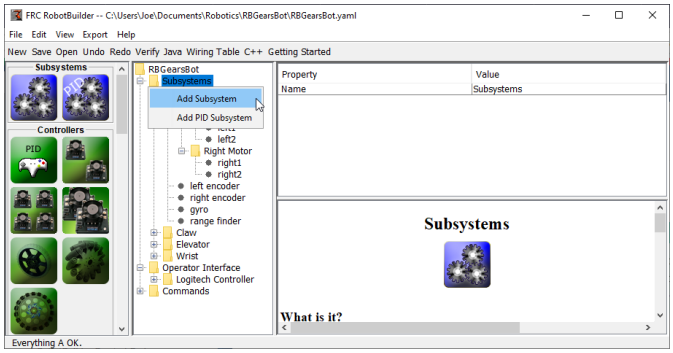
Les sous-systèmes sont des classes qui encapsulent (ou contiennent) toutes les données et le code qui font fonctionner un sous-système sur votre robot. La première étape de la création d'un programme de robot avec robotBuilder consiste à identifier et à créer tous les sous-systèmes du robot. Des exemples de sous-systèmes sont les ramasseurs, les collecteurs de billes, la base pilotable, les élevateurs, les bras, etc. Chaque sous-système contient tous les capteurs et actionneurs utilisés pour le faire fonctionner. Par exemple, un élevateur peut avoir un contrôleur de moteur Victor SPX et un potentiomètre pour fournir un signal de rétroaction sur la position du robot.

Création d'un sous-système à l'aide de la palette



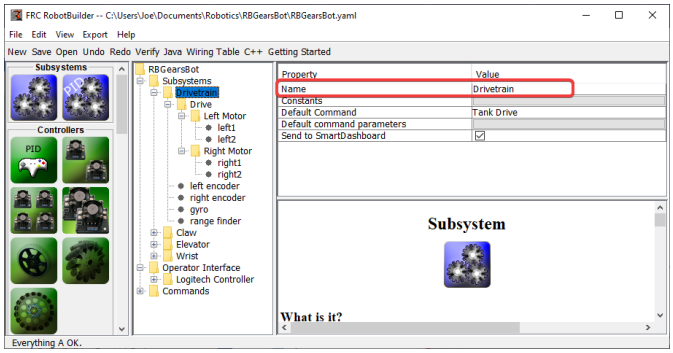
Faites glisser l'icône du sous-système de la palette vers le dossier Sous-systèmes dans la description du robot pour créer une classe de sous-système.

Création d'un sous-système à l'aide du menu contextuel



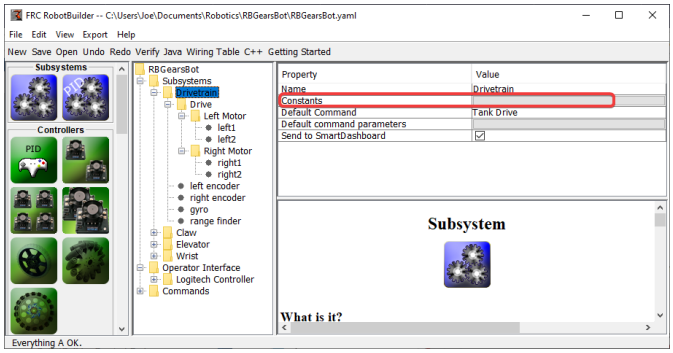
Cliquez avec le bouton droit sur le dossier Sous-système dans la description du robot pour ajouter un sous-système à ce dossier.

Nommer le sous-système



Après avoir créé le sous-système en faisant glisser ou en utilisant le menu contextuel décrit ci-dessus, tapez simplement le nom que vous souhaitez donner au sous-système. Le nom peut être composé de plusieurs mots séparés par des espaces, RobotBuilder concatenate les mots pour créer un nom qui est acceptable comme nom de classe Java ou C ++ .

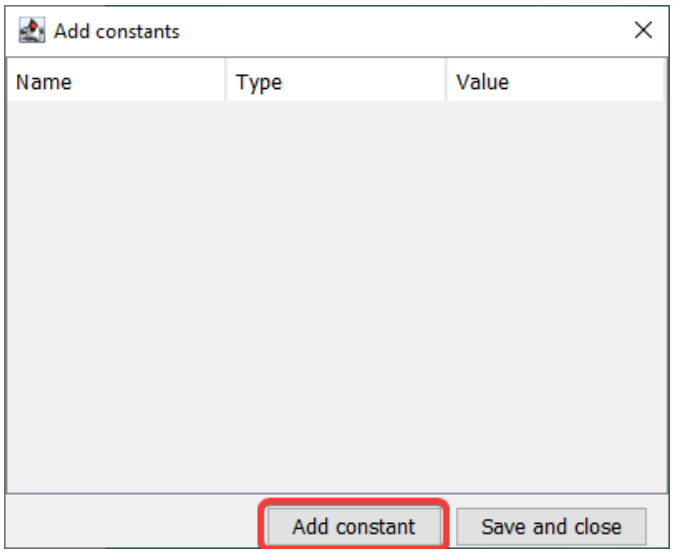
Ajout de constantes



Les constantes sont très utiles pour réduire la quantité de nombres « magiques » incrusté dans votre code. Dans les sous-systèmes, ils peuvent être utilisés pour surveiller certaines valeurs, comme les valeurs venant d'un capteur telles que la hauteur spécifique d'un mécanisme élevateur, ou la vitesse de déplacement du robot.

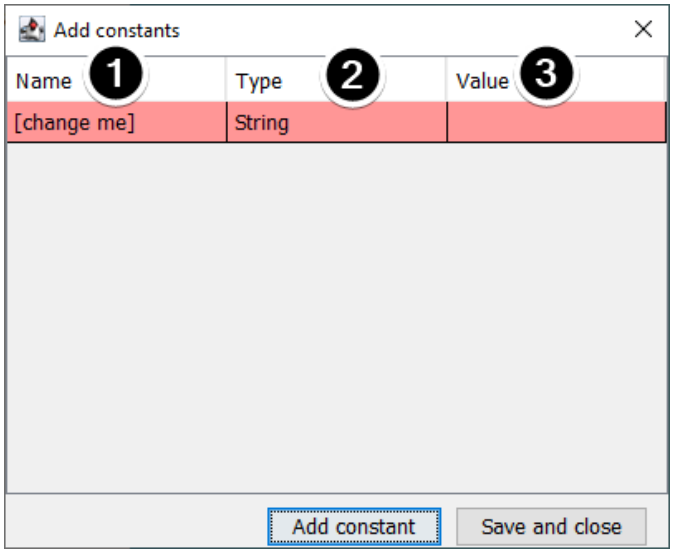
Par défaut, il n'y aura pas de constantes dans un sous-système. Pour créer une constante, appuyez sur le bouton à côté de « Constants » et ouvrir la boîte de dialogue requise.

Création de constantes



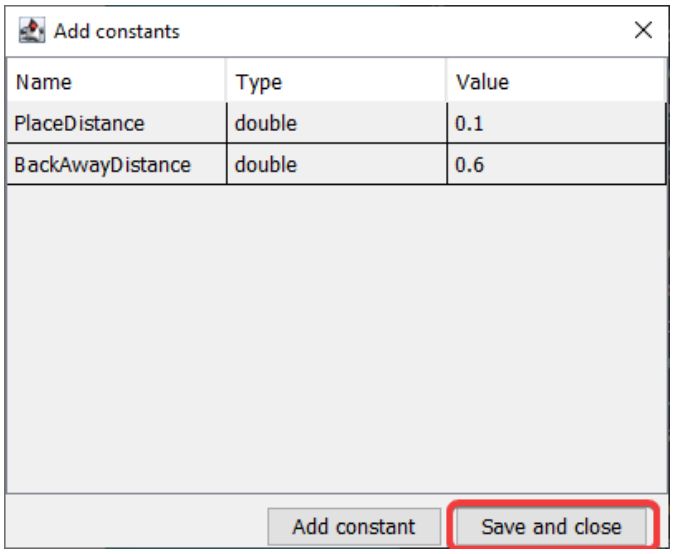
La table constantes sera vide dans un premier temps. Appuyez sur « Add constant » pour en ajouter une.

Ajout de constantes



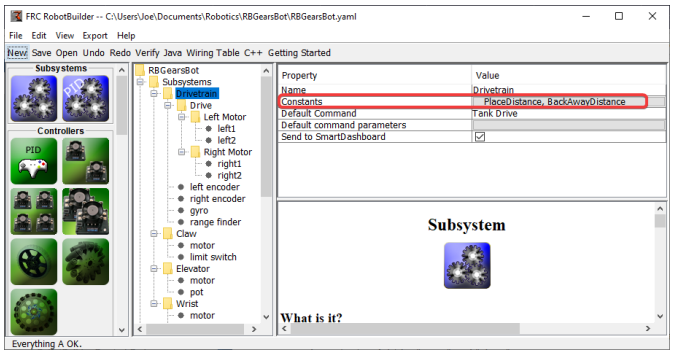
1. Le nom de la constante. Changez le pour quelque chose qui est plus descriptif. Dans cet exemple de base pilotable, certains bons noms de constantes peuvent être « Place-Distance » et « BackAwayDistance ».
2. Le type de la constante. Ce sera très probablement un double, mais vous pouvez choisir parmi les possibilités suivantes : String, double, int, long int, boolean, ou byte.
3. La valeur de la constante.

Sauvagarde de Constants



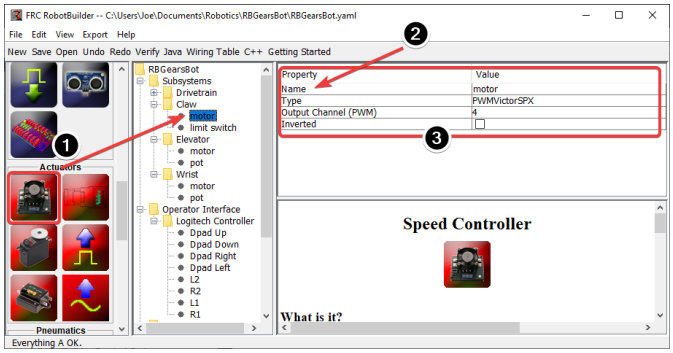
Après avoir ajouté des constantes et défini leurs valeurs, appuyez simplement sur « Enregistrer et fermer » pour enregistrer les constantes et fermer la boîte de dialogue. Si vous ne souhaitez pas enregistrer, appuyez sur le bouton de sortie en haut de la fenêtre.

Après la sauvegarde



Après avoir enregistré les constantes, les noms apparaîtront dans le bouton « Constantes » dans les propriétés du sous-système.

Ajout des dispositifs actionneurs / capteurs dans le sous-système



Il existe trois étapes pour ajouter des composants à un sous-système

1. Faites glisser les dispositifs actionneurs ou les capteurs en provenance de la palette vers le sous-système, selon vos besoins.
2. Nommer le dispositif actionneur ou le capteur
3. Modifier les propriétés telles que les numéros de module et les numéros de canal pour chaque élément du sous-système.

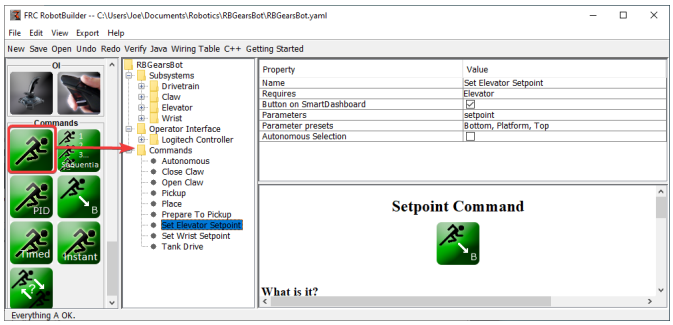
RobotBuilder incrémentera automatiquement les numéros de canaux utilisés pour chaque module du robot. Si vous n'avez pas encore câblé le robot, vous pouvez simplement laisser RobotBuilder faire le travail et attribuer automatiquement des numéros de canaux uniques pour chaque capteur ou dispositifs actionneur et plus tard, câbler le robot conformément au tableau ainsi généré.

Cela crée simplement un sous-système dans RobotBuilder et générera par la suite un code squelette pour le sous-système. Pour qu'il soit fonctionnel dans votre robot, veuillez vous référer à *Writing Code for a Subsystem*.

21.1.6 Création d'une commande

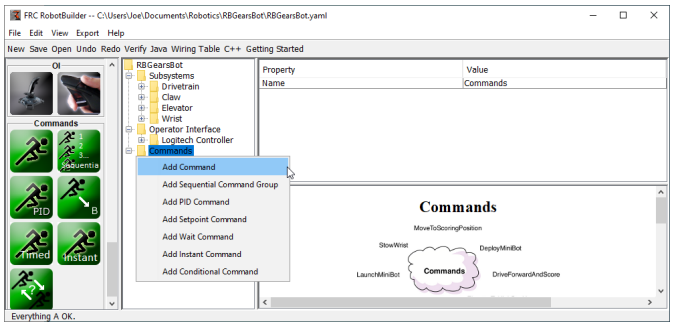
Les commandes sont des classes que vous créez qui fournissent des comportements ou des actions pour vos sous-systèmes. La classe de sous-système doit définir le fonctionnement du sous-système, comme MoveElevator pour démarrer le déplacement de d'un élévateur, ou ElevatorToSetPoint pour définir le point de consigne PID de l'élévateur. Les commandes lancent l'opération du sous-système et repèrent leur fins (lorsqu'elles se terminent).

Faire glisser la commande vers le dossier Commandes



Des commandes simples peuvent être glissées de la palette vers la description du robot. La commande sera créée sous le dossier Commandes.

Création de commandes à l'aide du menu contextuel



Vous pouvez également créer des commandes en utilisant le menu contextuel du clic droit sur le dossier Command dans la description du robot.

Configuration de la commande

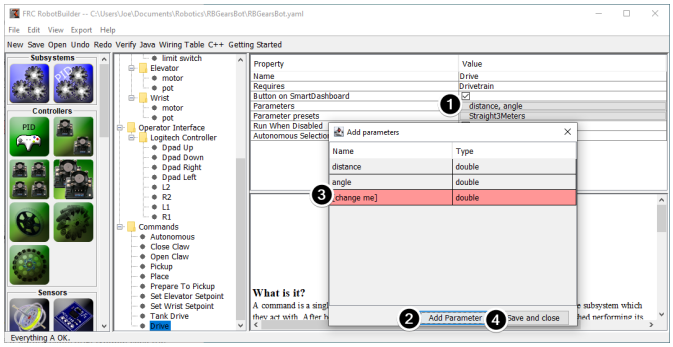
Property	Value
Name	Set Elevator Setpoint
Requires	Elevator
Button on SmartDashboard	☑
Parameters	setpoint
Parameter presets	Bottom, Platform, Top
Autonomous Selection	☐

1. Nommez la commande avec un nom significatif qui décrit ce que la commande fera. Les commandes doivent être nommées comme si elles faisaient partie du code, bien qu'il puisse y avoir des espaces entre les mots.
2. Définir le sous-système utilisé par cette commande. Lorsque cette commande est planifiée, elle arrêtera automatiquement toute commande en cours d'exécution qui nécessite également cette commande. Si une commande d'ouverture de la pince est en cours d'exécution (nécessitant le sous-système de la pince) et que la commande de fermeture de la pince est planifiée, elle arrêtera immédiatement l'ouverture et la fermeture s'amorcera.
3. Dire à RobotBuilder s'il doit créer des boutons sur le SmartDashboard pour la commande. Un bouton sera créé pour chaque paramètre prédéfini.
4. Définir les paramètres de cette commande. Une seule commande avec des paramètres peut faire la même chose que deux ou plusieurs commandes sans paramètres. Par exemple, les commandes « Drive Forward », « Drive Backward » et « Drive Distance » peuvent être consolidées en une seule commande qui prend des valeurs de direction et de distance.
5. Définissez des préréglages pour les paramètres. Ceux-ci peuvent être utilisés ailleurs dans RobotBuilder lors de l'utilisation de la commande, Quelques exemples : lier la commande à un bouton du joystick ou définir la commande par défaut pour un sous-système.
6. *Run When Disabled*. Permet à la commande de s'exécuter lorsque le robot est désactivé. Toutefois, tous les actionneurs commandés lorsqu'ils sont désactivés ne fonctionnent pas.

7. *Autonomous Selection*. Si la commande doit être ajoutée à l'objet Sendable Chooser afin qu'elle puisse être sélectionnée pour l'exécution du mode autonome.

Les commandes de Point de consigne sont définies avec un seul paramètre ("setpoint", de type double) ; les paramètres ne peuvent pas être ajoutés, modifiés ou supprimés pour les commandes de Points de consigne.

Ajout et modification de paramètres

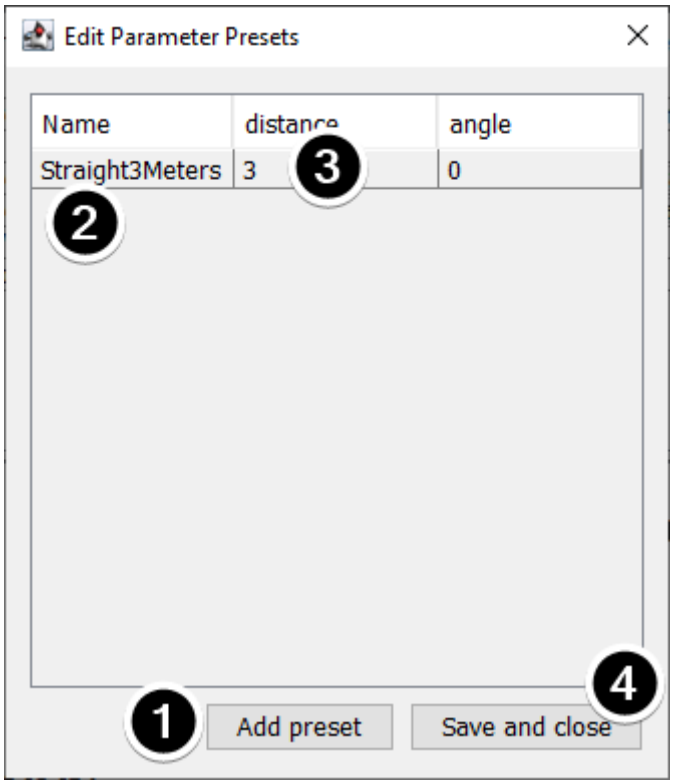


Pour ajouter ou modifier des paramètres :

1. Cliquez sur le bouton dans la colonne *Value* du tableau des propriétés
2. Appuyez sur le bouton *Add Parameter* pour ajouter un paramètre
3. Un paramètre qui vient d'être ajouté. Le nom par défaut est *[change me]* et le type par défaut est String (chaîne de caractères). Le nom par défaut n'est pas valide, vous devrez donc le modifier avant d'exporter. Double-cliquez sur la cellule *Name* pour commencer à changer le nom. Double-cliquez sur la cellule *Type* pour sélectionner le type.
4. Le bouton « Save and close » enregistre toutes les modifications et ferme la fenêtre.

Les lignes peuvent être réorganisées simplement en les faisant glisser et peuvent être supprimées en les sélectionnant et en appuyant sur la touche « Delete » ou « Backspace ».

Ajout et modification des préréglages de paramètres (Parameter Presets)

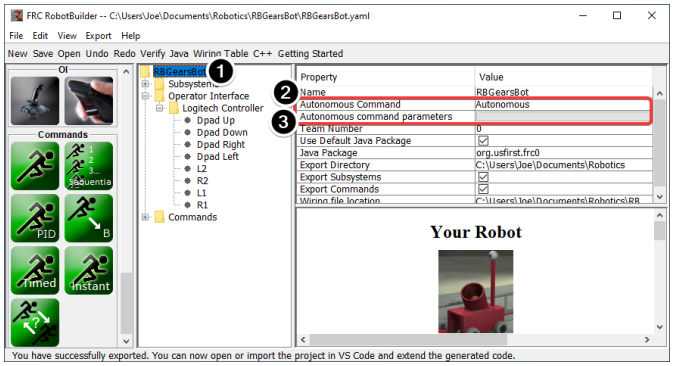


1. Cliquez sur *Add parameter set* pour ajouter un nouveau préréglage.
2. Remplacez le nom du préréglage par quelque chose de descriptif. Les préréglages de cet exemple permettent d'ouvrir et de fermer le sous-système de pince.
3. Modifiez la valeur du ou des paramètres du préréglage. Vous pouvez soit saisir une valeur (par exemple « 3.14 »), soit sélectionner parmi les constantes définies dans le sous-système requis par la commande. Notez que le type de la constante doit être du même type que le paramètre - vous ne pouvez pas faire passer une constante de type « int » à un paramètre de type « double », par exemple
4. Cliquez sur *Save and close* pour enregistrer les modifications et quitter la boîte de dialogue ; pour quitter sans enregistrer, appuyez sur le bouton Exit dans la barre supérieure de la fenêtre.

21.1.7 Définition des commandes autonomes

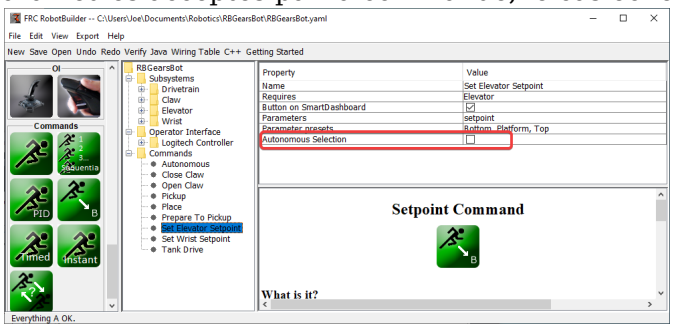
Puisqu'une commande est simplement une ou plusieurs actions (comportements) que le robot exécute, il est logique de décrire le fonctionnement autonome d'un robot comme une commande. Bien qu'il puisse s'agir d'une seule commande, il est plus probable que ce soit un groupe de commandes (un groupe de commandes qui se produisent ensemble).

RobotBuilder génère du code pour un *Sendable Chooser* qui permet à la commande autonome à exécuter d'être sélectionnée à partir du dashboard.



Pour désigner la commande autonome par défaut qui s'exécute si une autre commande n'est pas sélectionnée sur le dashboard :

— Sélectionnez le robot dans la description du programme robot
— Remplissez le champ de commande Autonome avec la commande qui doit fonctionner lorsque le robot est placé en mode autonome. Il s'agit d'un menu déroulant et vous donnera la possibilité de sélectionner n'importe quelle commande qui a été définie.
— Définissez les paramètres acceptés par la commande, le cas échéant.



Pour sélectionner les commandes à ajouter en tant qu'options au Sendable Chooser, sélectionnez la case à cochée Autonomous Selection.

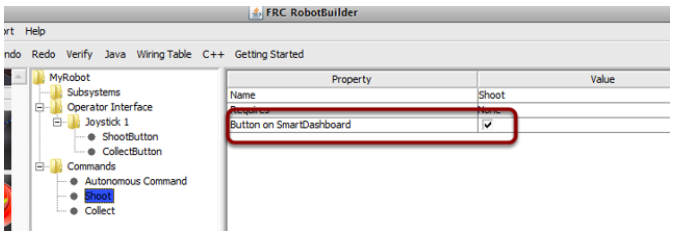
Lorsque le robot est mis en mode autonome, la commande autonome choisie sera planifiée.

21.1.8 Utilisation de Shuffleboard pour tester une commande

Les commandes sont facilement testées en ajoutant un bouton au Shuffleboard/SmartDashboard pour déclencher la commande. De cette façon, aucune intégration avec le reste du programme du robot n'est nécessaire et les commandes peuvent facilement être testées indépendamment. C'est le moyen le plus simple de vérifier les commandes car avec seulement une ligne de code dans votre programme, un bouton peut être créé sur le Shuffleboard qui exécutera la commande. Ces boutons peuvent ensuite être laissés en place pour vérifier les sous-systèmes et les opérations de commande dans le futur.

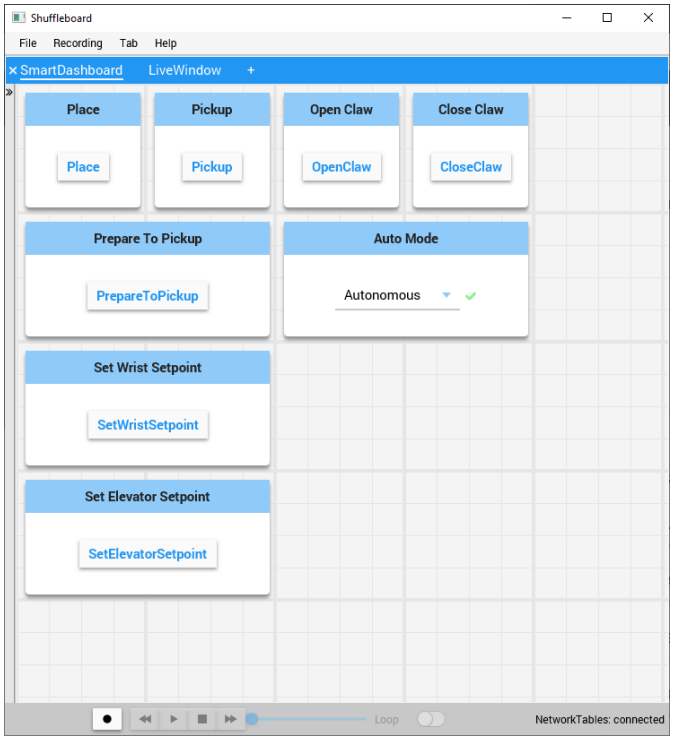
Cela a l'avantage supplémentaire de permettre à plusieurs programmeurs de travailler ensemble, chacun écrivant ses commandes. Lorsque le code est archivé dans le projet de robot principal, les commandes peuvent être testées individuellement.

Création de bouton sur le Shuffleboard



Le bouton est créé sur le Shuffleboard en plaçant une instance de la commande du programme robot dans le tableau de bord. Il s'agit d'une opération tellement courante qu'elle a été ajoutée à RobotBuilder en tant que case à cocher. Lorsque vous écrivez vos commandes, assurez-vous que la case est cochée et les boutons seront automatiquement générés pour vous.

utilisation des boutons



Les boutons seront générés automatiquement et apparaîtront sur l'écran du Shuffleboard. Vous pouvez réarranger les boutons sur le Shufflebaord. Dans cet exemple, il existe un certain nombre de commandes, chacune avec un bouton associé pour le test. Une pression sur le bouton d'une commande lancera l'excécution de celles-ci. Si un bouton associé à une commande est déjà enfoncé, l'appuyer à nouveau aura pour effet d'interrompre la commande car cette action a pour effet d'invoquer la méthode `Interrupted()`.

Ajout manuel de commandes

JAVA

```
SmartDashboard.putData("Autonomous Command", new AutonomousCommand());
SmartDashboard.putData("Open Claw", new OpenClaw(m_claw);
SmartDashboard.putData("Close Claw", new CloseClaw(m_claw));
```

C++

```
SmartDashboard::PutData("Autonomous Command", new AutonomousCommand());
SmartDashboard::PutData("Open Claw", new OpenClaw(&m_claw));
SmartDashboard::PutData("Close Claw", new CloseClaw(&m_claw));
```

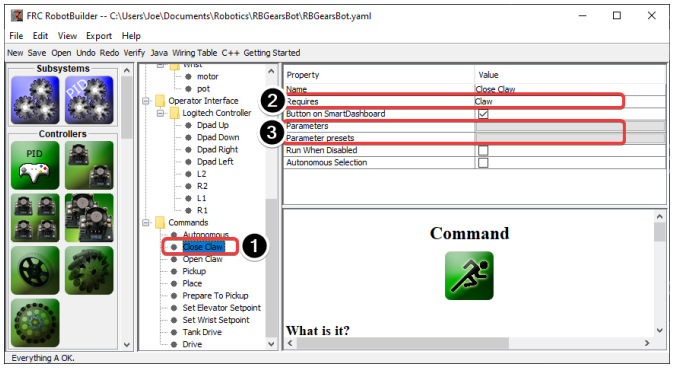
Les commandes peuvent être ajoutées manuellement au Shuffleboard en écrivant vous-même le code. Cela se fait en passant des instances de la commande à la méthode PutData avec le nom qui doit être associé au bouton sur le Shuffleboard. Ces instances sont alors planifiées pour être exécutées chaque fois que le bouton est enfoncé. Le résultat est exactement le même que le code généré par RobotBuilder, bien que cliquer sur la case à cocher dans RobotBuilder soit beaucoup plus facile que d'écrire tout le code à la main.

21.1.9 Connexion de l'interface opérateur à une commande

Les commandes gèrent les comportements de votre robot. La commande démarre un sous-système dans un mode de fonctionnement comme l'élévation et continue de fonctionner jusqu'à ce qu'il atteigne un certain point de consigne ou un certain délai. La commande gère ensuite l'attente de la fin du sous-système. De cette façon, les commandes peuvent s'exécuter en séquence pour développer des comportements plus complexes.

RobotBuilder générera également du code pour planifier l'exécution d'une commande chaque fois qu'un bouton de votre interface opérateur est enfoncé. Vous pouvez également écrire du code pour exécuter une commande lorsqu'une condition de déclenchement particulière s'est produite.

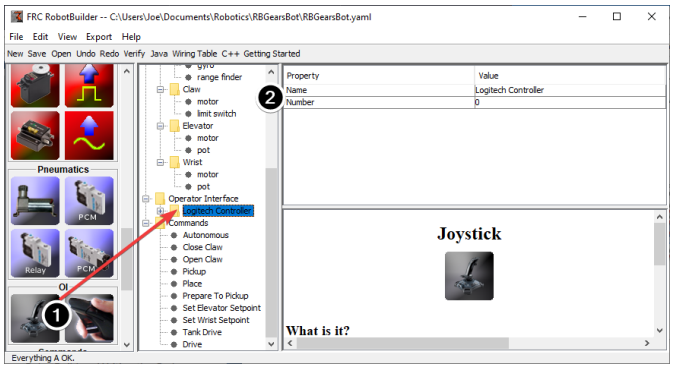
Exécuter une commande en appuyant sur un bouton



Dans cet exemple, nous voulons planifier l'exécution de la commande « Close Claw » chaque fois que le bouton de direction droit de la croix directionnelle ou dpad est appuyé, sur une manette de jeu logitech (bouton 6) est pressé.

1. La commande à exécuter est appelée « Close Claw » et sa fonction est de fermer la pince du robot
2. Notez que la commande nécessite le sous-système Claw. Cela garantira que cette commande commencera à s'exécuter même si une autre opération utilisait au même moment les ressources de la pince. Dans ce cas, la commande précédente serait interrompue.
3. Les paramètres permettent à une commande de faire plusieurs choses ; les préréglages vous permettent de définir les valeurs que vous passez à la commande et de les réutiliser

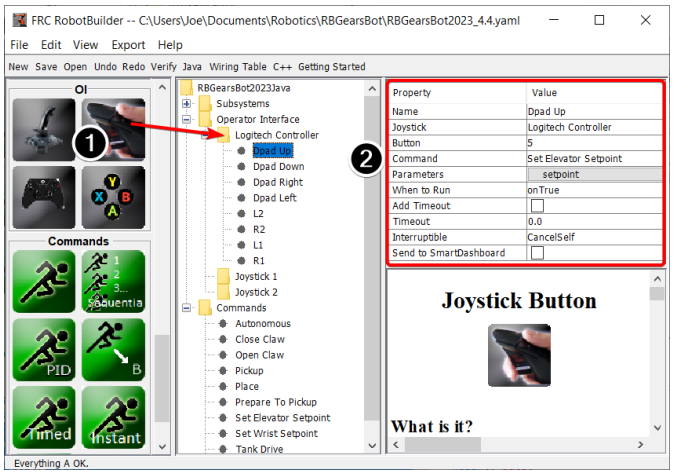
Ajout du joystick au programme Robot



Ajoutez le joystick au programme du robot

1. Faites glisser le joystick vers le dossier Interface opérateur dans le programme du robot
2. Nommez le joystick en utilisant un nom significatif et définissez le numéro de port USB

Faire un lien entre un bouton et la commande « Move Elevator »



Ajoutez le bouton dans le programme

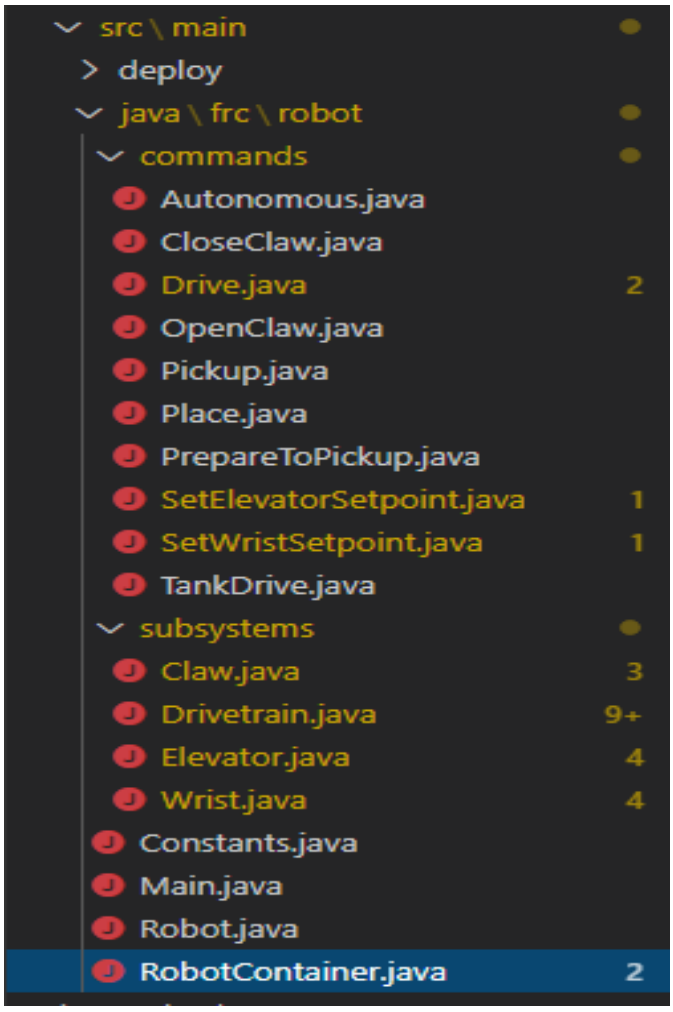
1. Faites glisser le bouton joystick sur le Joystick (Logitech Controller) de sorte qu'il soit sous le joystick

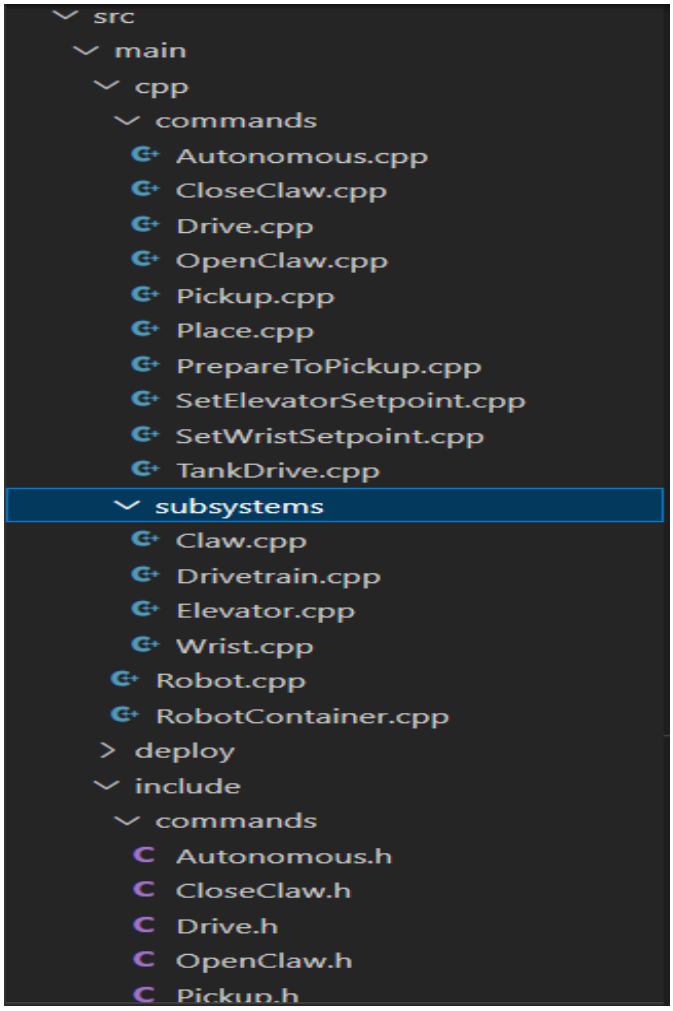
2. Set the properties for the button : the button number, the command to run when the button is pressed, parameters the command takes, and the *When to run* property to *on-True* to indicate that the command should run whenever the joystick button is pressed.

Note : Les boutons du joystick doivent être glissés vers (sous) un joystick. Vous devez avoir un joystick dans le dossier Operator Interface avant d'ajouter des boutons.

21.1.10 Code créé par RobotBuilder

La disposition d'un projet généré par RobotBuilder





Un projet généré par RobotBuilder se compose d'un package (en Java) ou d'un dossier (en C++) pour les commandes et un autre pour les sous-systèmes. Chaque objet Command ou de Subsystem est stocké sous ces conteneurs. Au niveau supérieur du projet, vous trouverez le programme principal du robot (RobotContainer.java/C++).

Pour plus d'informations sur l'organisation d'un robot orienté commande, voir *Structurer un projet de robot orienté commande*

Code généré automatiquement

JAVA

```
// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
m_chooser.setDefaultOption("Autonomous", new Autonomous());
// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS

SmartDashboard.putData("Auto Mode", m_chooser);
```

C++

```
// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
m_chooser.SetDefaultOption("Autonomous", new Autonomous());
// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS

frc::SmartDashboard::PutData("Auto Mode", &m_chooser);
```

Lorsque la description du robot est modifiée et que le code est réexporté, RobotBuilder est conçu pour ne pas modifier les modifications que vous avez apportées au fichier, préservant ainsi votre code. Cela fait de RobotBuilder un outil de cycle de vie complet. RobotBuilder indique les sections qui devront potentiellement être réécrites en les délimitant avec quelques commentaires spéciaux. Ces commentaires sont illustrés dans l'exemple ci-dessus. N'ajoutez aucun code dans ces blocs de commentaires, il sera réécrit la prochaine fois que le projet sera exporté de RobotBuilder.

Si le code à l'intérieur d'un de ces blocs doit être modifié, les commentaires peuvent être supprimés, mais cela empêchera d'autres mises à jour de se produire plus tard. Dans l'exemple ci-dessus, si les commentaires //BEGIN et //END étaient supprimés, puis un autre sous-système requis était ajouté dans RobotBuilder, il ne serait pas généré lors du prochain Export.

JAVA

```
// ROBOTBUILDER TYPE: Robot.
```

C++

```
// ROBOTBUILDER TYPE: Robot.
```

En outre, chaque fichier comporte un commentaire définissant le type de fichier. Si ce commentaire est modifié ou supprimé, RobotBuilder régénérera complètement le fichier en supprimant tout code ajouté à l'intérieur et à l'extérieur des blocs AUTOGENERATED CODE.

Programme de robot principal

Java

```
// ROBOTBUILDER TYPE: Robot.

package frc.robot;

import edu.wpi.first.hal.FRCNetComm.tInstances;
import edu.wpi.first.hal.FRCNetComm.tResourceType;
import edu.wpi.first.hal.HAL;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.CommandScheduler;

/**
 * The VM is configured to automatically run this class, and to call the
```

(suite sur la page suivante)

(suite de la page précédente)

```
 * functions corresponding to each mode, as described in the TimedRobot
 * documentation. If you change the name of this class or the package after
 * creating this project, you must also update the build.properties file in
 * the project.
 */
public class Robot extends TimedRobot { // (1)

    private Command m_autonomousCommand;

    private RobotContainer m_robotContainer;

    /**
     * This function is run when the robot is first started up and should be
     * used for any initialization code.
     */
    @Override
    public void robotInit() {
        // Instantiate our RobotContainer.  This will perform all our button bindings, and put our
        // autonomous chooser on the dashboard.
        m_robotContainer = RobotContainer.getInstance();
        HAL.report(tResourceType.kResourceType_Framework, tInstances.kFramework_RobotBuilder);
    }

    /**
    * This function is called every robot packet, no matter the mode. Use this for items like
    * diagnostics that you want ran during disabled, autonomous, teleoperated and test.
    *
    * <p>This runs after the mode specific periodic functions, but before
    * LiveWindow and SmartDashboard integrated updating.
    */
    @Override
    public void robotPeriodic() {
        // Runs the Scheduler.  This is responsible for polling buttons, adding newly-scheduled
        // commands, running already-scheduled commands, removing finished or interrupted commands,
        // and running subsystem periodic() methods.  This must be called from the robot's periodic
        // block in order for anything in the Command-based framework to work.
        CommandScheduler.getInstance().run();  // (2)
    }


    /**
    * This function is called once each time the robot enters Disabled mode.
    */
    @Override
    public void disabledInit() {
    }

    @Override
    public void disabledPeriodic() {
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }

    /**
    * This autonomous runs the autonomous command selected by your {@link␣
↪RobotContainer} class.
    */
    @Override
    public void autonomousInit() {
        m_autonomousCommand = m_robotContainer.getAutonomousCommand();  // (3)

        // schedule the autonomous command (example)
        if (m_autonomousCommand != null) {
            m_autonomousCommand.schedule();
        }
    }

    /**
    * This function is called periodically during autonomous.
    */
    @Override
    public void autonomousPeriodic() {
    }

    @Override
    public void teleopInit() {
        // This makes sure that the autonomous stops running when
        // teleop starts running. If you want the autonomous to
        // continue until interrupted by another command, remove
        // this line or comment it out.
        if (m_autonomousCommand != null) {
            m_autonomousCommand.cancel();
        }
    }

    /**
     * This function is called periodically during operator control.
     */
    @Override
    public void teleopPeriodic() {
    }

    @Override
    public void testInit() {
        // Cancels all running commands at the start of test mode.
        CommandScheduler.getInstance().cancelAll();
    }

    /**
    * This function is called periodically during test mode.
    */
    @Override
    public void testPeriodic() {
    }

}
```

C++ (Header ou en-tête)

```
// ROBOTBUILDER TYPE: Robot.
#pragma once

#include <frc/TimedRobot.h>
#include <frc2/command/Command.h>

#include "RobotContainer.h"

class Robot : public frc::TimedRobot { // {1}
 public:
  void RobotInit() override;
  void RobotPeriodic() override;
  void DisabledInit() override;
  void DisabledPeriodic() override;
  void AutonomousInit() override;
  void AutonomousPeriodic() override;
  void TeleopInit() override;
  void TeleopPeriodic() override;
  void TestPeriodic() override;

 private:
  // Have it null by default so that if testing teleop it
  // doesn't have undefined behavior and potentially crash.
  frc2::Command* m_autonomousCommand = nullptr;

  RobotContainer* m_container = RobotContainer::GetInstance();
};
```

C++ (Source)

```
// ROBOTBUILDER TYPE: Robot.

#include "Robot.h"

#include <frc/smartdashboard/SmartDashboard.h>
#include <frc2/command/CommandScheduler.h>

void Robot::RobotInit() {}

/**
 * This function is called every robot packet, no matter the mode. Use
 * this for items like diagnostics that you want to run during disabled,
 * autonomous, teleoperated and test.
 *
 * <p> This runs after the mode specific periodic functions, but before
 * LiveWindow and SmartDashboard integrated updating.
 */
void Robot::RobotPeriodic() { frc2::CommandScheduler::GetInstance().Run(); } // (2)

/**
 * This function is called once each time the robot enters Disabled mode. You
 * can use it to reset any subsystem information you want to clear when the
 * robot is disabled.
```

(suite sur la page suivante)

(suite de la page précédente)

```
 */
void Robot::DisabledInit() {}

void Robot::DisabledPeriodic() {}

/**
 * This autonomous runs the autonomous command selected by your {@link
 * RobotContainer} class.
 */
void Robot::AutonomousInit() {
  m_autonomousCommand = m_container->GetAutonomousCommand(); // {3}

  if (m_autonomousCommand != nullptr) {
    m_autonomousCommand->Schedule();
  }
}

void Robot::AutonomousPeriodic() {}

void Robot::TeleopInit() {
  // This makes sure that the autonomous stops running when
  // teleop starts running. If you want the autonomous to
  // continue until interrupted by another command, remove
  // this line or comment it out.
  if (m_autonomousCommand != nullptr) {
    m_autonomousCommand->Cancel();
    m_autonomousCommand = nullptr;
  }
}

/**
 * This function is called periodically during operator control.
 */
void Robot::TeleopPeriodic() {}

/**
 * This function is called periodically during test mode.
 */
void Robot::TestPeriodic() {}

#ifndef RUNNING_FRC_TESTS
int main() { return frc::StartRobot<Robot>(); }
#endif
```

Il s'agit du programme principal généré par RobotBuilder. Ce programme comprend plusieurs parties (sections mises en évidence) :

1. Cette classe étend TimedRobot. TimedRobot appellera toutes les 20 mSec vos méthodes `autonomousPeriodic()` et `teleopPeriodic()`.
2. Dans la méthode teleopPeriodic qui est appelée toutes les 20 ms, effectuez une seule passe de planification.
3. La commande autonome fournie est planifiée pour être exécutée au début du mode autonome dans la méthode `autonomousInit()` et annulée à la fin de la période autonome dans `teleopInit()`.

RobotContainer

Java

```
// ROBOTBUILDER TYPE: RobotContainer.

package frc.robot;

import frc.robot.commands.*;
import frc.robot.subsystems.*;
import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj2.command.Command.InterruptionBehavior;

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.InstantCommand;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj2.command.button.JoystickButton;
import frc.robot.subsystems.*;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS


/**
 * This class is where the bulk of the robot should be declared.  Since Command-based␣
↪is a
 * "declarative" paradigm, very little robot logic should actually be handled in the
↪{@link Robot}
 * periodic methods (other than the scheduler calls).  Instead, the structure of the␣
↪robot
 * (including subsystems, commands, and button mappings) should be declared here.
 */
public class RobotContainer {

  private static RobotContainer m_robotContainer = new RobotContainer();

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
// The robot's subsystems
    public final Wrist m_wrist = new Wrist(); // (1)
    public final Elevator m_elevator = new Elevator();
    public final Claw m_claw = new Claw();
    public final Drivetrain m_drivetrain = new Drivetrain();

// Joysticks
private final Joystick joystick2 = new Joystick(2); // (3)
private final Joystick joystick1 = new Joystick(1);
private final Joystick logitechController = new Joystick(0);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS


  // A chooser for autonomous commands
  SendableChooser<Command> m_chooser = new SendableChooser<>();

  /**
  * The container for the robot.  Contains subsystems, OI devices, and commands.
```

(suite sur la page suivante)

(suite de la page précédente)

```
61    */
62    private RobotContainer() {
63          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
64      // Smartdashboard Subsystems
65      SmartDashboard.putData(m_wrist); // (6)
66      SmartDashboard.putData(m_elevator);
67      SmartDashboard.putData(m_claw);
68      SmartDashboard.putData(m_drivetrain);
69
70
71      // SmartDashboard Buttons
72      SmartDashboard.putData("Close Claw", new CloseClaw( m_claw )); // (6)
73      SmartDashboard.putData("Open Claw: OpenTime", new OpenClaw(1.0, m_claw));
74      SmartDashboard.putData("Pickup", new Pickup());
75      SmartDashboard.putData("Place", new Place());
76      SmartDashboard.putData("Prepare To Pickup", new PrepareToPickup());
77      SmartDashboard.putData("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0,
    ↪ m_elevator));
78      SmartDashboard.putData("Set Elevator Setpoint: Platform", new␣
    ↪SetElevatorSetpoint(0.2, m_elevator));
79      SmartDashboard.putData("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3,␣
    ↪m_elevator));
80      SmartDashboard.putData("Set Wrist Setpoint: Horizontal", new SetWristSetpoint(0,␣
    ↪m_wrist));
81      SmartDashboard.putData("Set Wrist Setpoint: Raise Wrist", new SetWristSetpoint(-
    ↪45, m_wrist));
82      SmartDashboard.putData("Drive: Straight3Meters", new Drive(3, 0, m_drivetrain));
83      SmartDashboard.putData("Drive: Place", new Drive(Drivetrain.PlaceDistance,␣
    ↪Drivetrain.BackAwayDistance, m_drivetrain));
84
85      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
86      // Configure the button bindings
87      configureButtonBindings();
88
89      // Configure default commands
90          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SUBSYSTEM_DEFAULT_COMMAND
91      m_drivetrain.setDefaultCommand(new TankDrive(() -> getJoystick1().getY(), () ->␣
    ↪getJoystick2().getY(), m_drivetrain)); // (5)
92
93
94      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SUBSYSTEM_DEFAULT_COMMAND
95
96      // Configure autonomous sendable chooser
97          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
98
99      m_chooser.addOption("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0, m_
    ↪elevator));
100     m_chooser.addOption("Set Elevator Setpoint: Platform", new SetElevatorSetpoint(0.
    ↪2, m_elevator));
101     m_chooser.addOption("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3, m_
    ↪elevator));
102     m_chooser.setDefaultOption("Autonomous", new Autonomous()); // (2)
103
104     // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
105
106     SmartDashboard.putData("Auto Mode", m_chooser);
```

(suite sur la page suivante)

(suite de la page précédente)

```
  }

  public static RobotContainer getInstance() {
    return m_robotContainer;
  }

  /**
   * Use this method to define your button->command mappings.  Buttons can be created by
   * instantiating a {@link GenericHID} or one of its subclasses ({@link
   * edu.wpi.first.wpilibj.Joystick} or {@link XboxController}), and then passing it to a
   * {@link edu.wpi.first.wpilibj2.command.button.JoystickButton}.
   */
  private void configureButtonBindings() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=BUTTONS
// Create some buttons
final JoystickButton r1 = new JoystickButton(logitechController, 12);        // (4)
r1.onTrue(new Autonomous().withInterruptBehavior(InterruptionBehavior.kCancelSelf));

final JoystickButton l1 = new JoystickButton(logitechController, 11);
l1.onTrue(new Place().withInterruptBehavior(InterruptionBehavior.kCancelSelf));

final JoystickButton r2 = new JoystickButton(logitechController, 10);
r2.onTrue(new Pickup().withInterruptBehavior(InterruptionBehavior.kCancelSelf));

final JoystickButton l2 = new JoystickButton(logitechController, 9);
l2.onTrue(new PrepareToPickup().withInterruptBehavior(InterruptionBehavior.kCancelSelf));

final JoystickButton dpadLeft = new JoystickButton(logitechController, 8);
dpadLeft.onTrue(new OpenClaw(1.0, m_claw).withInterruptBehavior(InterruptionBehavior.kCancelSelf));

final JoystickButton dpadRight = new JoystickButton(logitechController, 6);
dpadRight.onTrue(new CloseClaw( m_claw ).withInterruptBehavior(InterruptionBehavior.kCancelSelf));

final JoystickButton dpadDown = new JoystickButton(logitechController, 7);
dpadDown.onTrue(new SetElevatorSetpoint(0, m_elevator).withInterruptBehavior(InterruptionBehavior.kCancelSelf));

final JoystickButton dpadUp = new JoystickButton(logitechController, 5);
dpadUp.onTrue(new SetElevatorSetpoint(0.3, m_elevator).withInterruptBehavior(InterruptionBehavior.kCancelSelf));



    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=BUTTONS
  }

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=FUNCTIONS
public Joystick getLogitechController() {
        return logitechController;
    }

```

(suite sur la page suivante)

(suite de la page précédente)

```
public Joystick getJoystick1() {
        return joystick1;
    }

public Joystick getJoystick2() {
        return joystick2;
    }


    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=FUNCTIONS

  /**
   * Use this to pass the autonomous command to the main {@link Robot} class.
   *
   * @return the command to run in autonomous
   */
  public Command getAutonomousCommand() {
    // The selected command will be run in autonomous
    return m_chooser.getSelected();
  }


}
```

C++ (Header ou en-tête)

```
// ROBOTBUILDER TYPE: RobotContainer.

#pragma once

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES
#include <frc/smartdashboard/SendableChooser.h>
#include <frc2/command/Command.h>

#include "subsystems/Claw.h"
#include "subsystems/Drivetrain.h"
#include "subsystems/Elevator.h"
#include "subsystems/Wrist.h"

#include "commands/Autonomous.h"
#include "commands/CloseClaw.h"
#include "commands/Drive.h"
#include "commands/OpenClaw.h"
#include "commands/Pickup.h"
#include "commands/Place.h"
#include "commands/PrepareToPickup.h"
#include "commands/SetElevatorSetpoint.h"
#include "commands/SetWristSetpoint.h"
#include "commands/TankDrive.h"
#include <frc/Joystick.h>
#include <frc2/command/button/JoystickButton.h>

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES

```

(suite sur la page suivante)

(suite de la page précédente)

```
class RobotContainer {

public:

    frc2::Command* GetAutonomousCommand();
    static RobotContainer* GetInstance();

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=PROTOTYPES
// The robot's subsystems
Drivetrain m_drivetrain; // (1)
Claw m_claw;
Elevator m_elevator;
Wrist m_wrist;


frc::Joystick* getJoystick2();
frc::Joystick* getJoystick1();
frc::Joystick* getLogitechController();

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=PROTOTYPES

private:

    RobotContainer();

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
// Joysticks
frc::Joystick m_logitechController{0}; // (3)
frc::Joystick m_joystick1{1};
frc::Joystick m_joystick2{2};

frc::SendableChooser<frc2::Command*> m_chooser;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS

Autonomous m_autonomousCommand;
    static RobotContainer* m_robotContainer;

    void ConfigureButtonBindings();
};
```

C++ (Source)

```
// ROBOTBUILDER TYPE: RobotContainer.

#include "RobotContainer.h"
#include <frc2/command/ParallelRaceGroup.h>
#include <frc/smartdashboard/SmartDashboard.h>



RobotContainer* RobotContainer::m_robotContainer = NULL;

RobotContainer::RobotContainer() : m_autonomousCommand(
```

(suite sur la page suivante)

(suite de la page précédente)

```
22      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR
23  ){
24
25
26
27      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR
28
29      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
30      // Smartdashboard Subsystems
31      frc::SmartDashboard::PutData(&m_drivetrain);
32      frc::SmartDashboard::PutData(&m_claw);
33      frc::SmartDashboard::PutData(&m_elevator);
34      frc::SmartDashboard::PutData(&m_wrist);
35
36
37      // SmartDashboard Buttons
38      frc::SmartDashboard::PutData("Drive: Straight3Meters", new Drive(3, 0, &m_
    ↪drivetrain)); // (6)
39      frc::SmartDashboard::PutData("Drive: Place", new Drive(Drivetrain::PlaceDistance,␣
    ↪Drivetrain::BackAwayDistance, &m_drivetrain));
40      frc::SmartDashboard::PutData("Set Wrist Setpoint: Horizontal", new␣
    ↪SetWristSetpoint(0, &m_wrist));
41      frc::SmartDashboard::PutData("Set Wrist Setpoint: Raise Wrist", new␣
    ↪SetWristSetpoint(-45, &m_wrist));
42      frc::SmartDashboard::PutData("Set Elevator Setpoint: Bottom", new␣
    ↪SetElevatorSetpoint(0, &m_elevator));
43      frc::SmartDashboard::PutData("Set Elevator Setpoint: Platform", new␣
    ↪SetElevatorSetpoint(0.2, &m_elevator));
44      frc::SmartDashboard::PutData("Set Elevator Setpoint: Top", new␣
    ↪SetElevatorSetpoint(0.3, &m_elevator));
45      frc::SmartDashboard::PutData("Prepare To Pickup", new PrepareToPickup());
46      frc::SmartDashboard::PutData("Place", new Place());
47      frc::SmartDashboard::PutData("Pickup", new Pickup());
48      frc::SmartDashboard::PutData("Open Claw: OpenTime", new OpenClaw(1.0_s, &m_claw));
49      frc::SmartDashboard::PutData("Close Claw", new CloseClaw( &m_claw ));
50
51      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
52
53      ConfigureButtonBindings();
54
55      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT-COMMANDS
56  m_drivetrain.SetDefaultCommand(TankDrive([this] {return getJoystick1()->GetY();},␣
    ↪[this] {return getJoystick2()->GetY();}, &m_drivetrain)); // (5)
57
58      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT-COMMANDS
59
60      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
61
62      m_chooser.AddOption("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0, &
    ↪m_elevator));
63      m_chooser.AddOption("Set Elevator Setpoint: Platform", new SetElevatorSetpoint(0.
    ↪2, &m_elevator));
64      m_chooser.AddOption("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3, &m_
    ↪elevator));
65
66      m_chooser.SetDefaultOption("Autonomous", new Autonomous()); // (2)
```

(suite sur la page suivante)

(suite de la page précédente)

```

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS

    frc::SmartDashboard::PutData("Auto Mode", &m_chooser);

}

RobotContainer* RobotContainer::GetInstance() {
    if (m_robotContainer == NULL) {
        m_robotContainer = new RobotContainer();
    }
    return(m_robotContainer);
}

void RobotContainer::ConfigureButtonBindings() {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=BUTTONS

frc2::JoystickButton m_dpadUp{&m_logitechController, 5}; // (4)
frc2::JoystickButton m_dpadDown{&m_logitechController, 7};
frc2::JoystickButton m_dpadRight{&m_logitechController, 6};
frc2::JoystickButton m_dpadLeft{&m_logitechController, 8};
frc2::JoystickButton m_l2{&m_logitechController, 9};
frc2::JoystickButton m_r2{&m_logitechController, 10};
frc2::JoystickButton m_l1{&m_logitechController, 11};
frc2::JoystickButton m_r1{&m_logitechController, 12};

m_dpadUp.OnTrue(SetElevatorSetpoint(0.3, &m_elevator).
 ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));

m_dpadDown.OnTrue(SetElevatorSetpoint(0, &m_elevator).
 ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));

m_dpadRight.OnTrue(CloseClaw( &m_claw ).
 ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));

m_dpadLeft.OnTrue(OpenClaw(1.0_s, &m_claw).
 ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));

m_l2.OnTrue(PrepareToPickup().
 ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));

m_r2.OnTrue(Pickup().
 ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));

m_l1.OnTrue(Place().
 ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));

m_r1.OnTrue(Autonomous().
 ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));


    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=BUTTONS
}

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=FUNCTIONS

```

(suite sur la page suivante)

(suite de la page précédente)

```
frc::Joystick* RobotContainer::getLogitechController() {
   return &m_logitechController;
}
frc::Joystick* RobotContainer::getJoystick1() {
   return &m_joystick1;
}
frc::Joystick* RobotContainer::getJoystick2() {
   return &m_joystick2;
}

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=FUNCTIONS


frc2::Command* RobotContainer::GetAutonomousCommand() {
  // The selected command will be run in autonomous
  return m_chooser.GetSelected();
}
```

Il s'agit du RobotContainer généré par RobotBuilder qui est l'endroit où les sous-systèmes et l'interface de l'opérateur sont définis. Ce programme comprend plusieurs parties (sections mises en évidence) :

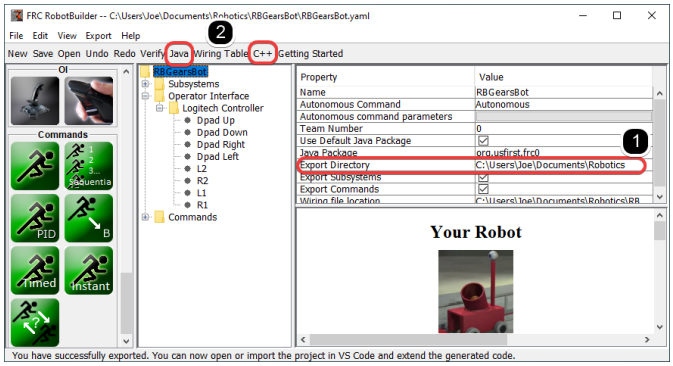
1. Chacun des sous-systèmes est déclaré ici. Ils peuvent être transmis comme paramètres à toutes les commandes qui en ont besoin.
2. S'il existe une commande autonome fournie dans les propriétés du fichier RobotBuilder du robot, elle est ajoutée à l'objet Sendable Chooser et selectionnable à partir du dashboard.
3. Le code pour tous les composants de l'interface opérateur est généré ici.
4. De plus, le code pour lier les boutons OI aux commandes qui devraient s'exécuter est également généré ici.
5. Commands to be run on a subsystem when no other commands are running are defined here.
6. Les commandes à exécuter via le dashboard sont définies ici.

21.2 RobotBuilder - Écriture du code

21.2.1 Génération du code pour un projet

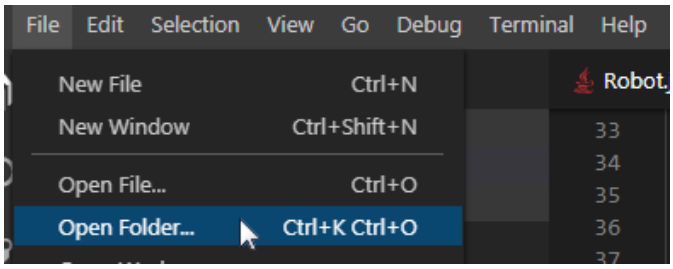
Après avoir configuré votre infrastructure de robot dans RobotBuilder, vous devrez exporter le code et le charger dans Visual Studio Code. Cet article décrit le processus pour ce faire.

Générer le code du projet



Vérifiez que le répertoire d'exportation pointe vers l'endroit où vous souhaitez (1), puis cliquez sur Java ou C ++ (2) pour générer un projet VS Code ou mettre à jour le code.

Ouvrez le projet dans Visual Studio Code

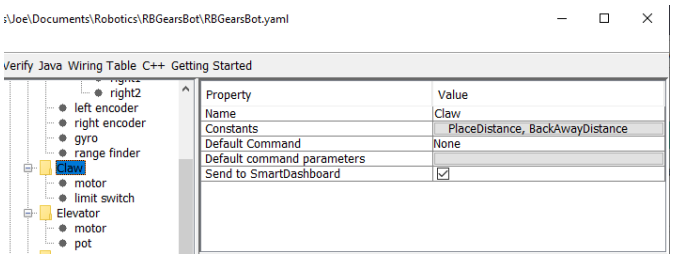


Ouvrez VS Code et sélectionnez **File -> Open Folder**. Accédez à votre emplacement d'exportation et cliquez sur **Select Folder**.

21.2.2 Écrire le code d'un sous-système

L'ajout de code pour créer un sous-système fonctionnel est très simple. Pour les sous-systèmes qui n'utilisent pas de rétroaction, cela s'avère extrêmement simple. Dans cette section, nous allons regarder un exemple de sous-système de pince ou *Claw* sur un bras articulé de robot. Le sous-système *Claw* dispose également d'un interrupteur de fin de course pour déterminer si un objet est bien saisi.

Représentation de RobotBuilder du sous-système Claw



La pince à l'extrémité du bras du robot est un sous-système actionné par un seul contrôleur de moteur VictorSPX. Il y a trois choses que nous voulons que la pince fasse, commence à s'ouvrir, commencer à se fermer, et s'arrêter. C'est la responsabilité du sous-système. La prise en charge pour l'ouverture et la fermeture sera assurée par une commande plus tard dans ce didacticiel. Nous définirons également une méthode pour savoir si la pince est en train de saisir un objet.

Ajout de capacités de sous-système

Java

```
// ROBOTBUILDER TYPE: Subsystem.

package frc.robot.subsystems;


import frc.robot.commands.*;
import edu.wpi.first.wpilibj.livewindow.LiveWindow;
import edu.wpi.first.wpilibj2.command.SubsystemBase;

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
import edu.wpi.first.wpilibj.DigitalInput;
import edu.wpi.first.wpilibj.motorcontrol.MotorController;
import edu.wpi.first.wpilibj.motorcontrol.PWMVictorSPX;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS


/**
 *
 */
public class Claw extends SubsystemBase {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS
public static final double PlaceDistance = 0.1;
public static final double BackAwayDistance = 0.6;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
private PWMVictorSPX motor;
private DigitalInput limitswitch;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS

    /**
    *
    */
    public Claw() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
motor = new PWMVictorSPX(4);
 addChild("motor",motor);
 motor.setInverted(false);

limitswitch = new DigitalInput(4);
 addChild("limit switch", limitswitch);
```

(suite sur la page suivante)

(suite de la page précédente)

```
55
56
57
58      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
59      }
60
61      @Override
62      public void periodic() {
63          // This method will be called once per scheduler run
64
65      }
66
67      @Override
68      public void simulationPeriodic() {
69          // This method will be called once per scheduler run when in simulation
70
71      }
72
73      public void open() {
74          motor.set(1.0);
75      }
76
77      public void close() {
78          motor.set(-1.0);
79      }
80
81      public void stop() {
82          motor.set(0.0);
83      }
84
85      public boolean isGripping() {
86          return limitswitch.get();
87      }
88
89  }
```

C++

```
11  // ROBOTBUILDER TYPE: Subsystem.
12
13  // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES
14  #include "subsystems/Claw.h"
15  #include <frc/smartdashboard/SmartDashboard.h>
16
17  // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES
18
19  Claw::Claw(){
20      SetName("Claw");
21      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
22      SetSubsystem("Claw");
23
24   AddChild("limit switch", &m_limitswitch);
25
26
```

(suite sur la page suivante)

(suite de la page précédente)

```
 AddChild("motor", &m_motor);
 m_motor.SetInverted(false);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
}

void Claw::Periodic() {
    // Put code here to be run every loop

}

void Claw::SimulationPeriodic() {
    // This method will be called once per scheduler run when in simulation

}

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS

// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS


void Claw::Open() {
    m_motor.Set(1.0);
}

void Claw::Close() {
    m_motor.Set(-1.0);
}

void Claw::Stop() {
    m_motor.Set(0.0);
}

bool Claw::IsGripping() {
    return m_limitswitch.Get();
}
```

Ajoutez des méthodes aux fichiers `claw.java` ou `claw.cpp` qui ouvriront, fermeront et empêcheront la pince de bouger et de lire l'état de l'interrupteur de fin de course de la pince. Ceux-ci seront utilisés par les commandes qui gèrent réellement le fonctionnement de la pince.

Note : Les commentaires ont été éliminés de ce fichier pour faciliter le suivi des changements de ce document.

Notez que les variables membres appelées `motor` et `limitswitch` sont créés par RobotBuilder afin qu'elles puissent être utilisées dans l'ensemble du sous-système. Chacun de vos éléments glissés dans la palette aura une variable membre avec le nom donné dans RobotBuilder.

Ajout des déclarations de méthode au fichier Header, ou d'en-tête (C ++ uniquement)

C++

```
// ROBOTBUILDER TYPE: Subsystem.
#pragma once

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES
#include <frc2/command/SubsystemBase.h>
#include <frc/DigitalInput.h>
#include <frc/motorcontrol/PWMVictorSPX.h>

// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES

/**
 *
 *
 * @author ExampleAuthor
 */
class Claw: public frc2::SubsystemBase {
private:
    // It's desirable that everything possible is private except
    // for methods that implement subsystem capabilities
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
frc::DigitalInput m_limitswitch{4};
frc::PWMVictorSPX m_motor{4};

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
public:
Claw();

    void Periodic() override;
    void SimulationPeriodic() override;
    void Open();
    void Close();
    void Stop();
    bool IsGripping();
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS
static constexpr const double PlaceDistance = 0.1;
static constexpr const double BackAwayDistance = 0.6;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS


};
```

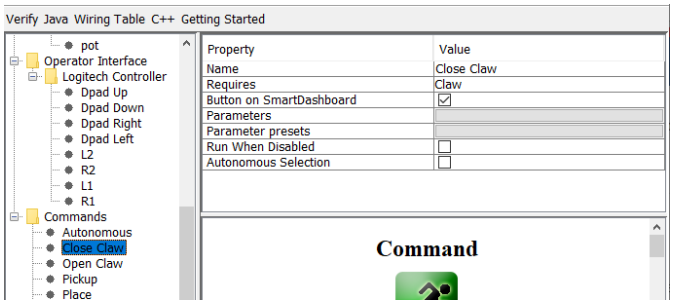
En plus d'ajouter les méthodes au fichier d'implémentation de classe, `Claw.cpp`, les déclarations des méthodes doivent être ajoutées au fichier d'en-tête, soit `Claw.h`. Les déclarations à ajouter sont indiquées ici.

Pour ajouter la fonctionnalité au sous-système Claw afin de gérer l'ouverture et la fermeture, vous devez *définir des commandes*.

21.2.3 Écrire le code d'une commande

Les classes Subsystem font bouger les mécanismes de votre robot, mais pour que les mouvements de ces mécanismes s'arrêtent au bon moment et se synchronisent à travers des opérations plus complexes, vous écrivez des commandes. Auparavant, dans *writing the code for a subsystem*, nous avons développé le code du sous-système *Claw* sur un robot pour démarrer l'ouverture de la pince, sa fermeture ou pour arrêter son mouvement. Maintenant, nous allons écrire le code pour une commande qui sera effectivement activer le moteur de la pince au moment opportun afin d'ouvrir ou de fermer la pince. Notre exemple de pince est un mécanisme très simple où nous activons le moteur pendant 1 seconde pour l'ouvrir ou jusqu'à ce que l'interrupteur de fin de course soit déclenché pour la fermer.

Commande Close Claw dans RobotBuilder



Ceci est la définition de la commande *CloseClaw* dans RobotBuilder. Notez qu'elle requiert le sous-système *Claw*. Cela s'explique à l'étape suivante.

Classe CloseClaw générée

JAVA

```
11 // ROBOTBUILDER TYPE: Command.
12
13 package frc.robot.commands;
14 import edu.wpi.first.wpilibj2.command.CommandBase;
15
16 // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
17 import frc.robot.subsystems.Claw;
18
19     // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
20
21 /**
22 *
23 */
24 public class CloseClaw extends CommandBase {
25
26     // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_DECLARATIONS
27         private final Claw m_claw;
28
29     // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_DECLARATIONS
30
31     // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
```

(suite sur la page suivante)

(suite de la page précédente)

```
32
33
34      public CloseClaw(Claw subsystem) {
35
36
37      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
38          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING
39
40      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING
41          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
42
43          m_claw = subsystem;
44          addRequirements(m_claw);
45
46      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
47      }
48
49      // Called when the command is initially scheduled.
50      @Override
51      public void initialize() {
52          m_claw.close(); // (1)
53      }
54
55      // Called every time the scheduler runs while the command is scheduled.
56      @Override
57      public void execute() {
58      }
59
60      // Called once the command ends or is interrupted.
61      @Override
62      public void end(boolean interrupted) {
63          m_claw.stop(); // (3)
64      }
65
66      // Returns true when the command should end.
67      @Override
68      public boolean isFinished() {
69          return m_claw.isGripping(); // (2)
70      }
71
72      @Override
73      public boolean runsWhenDisabled() {
74          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
75          return false;
76
77      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
78      }
79  }
```

C++

```
// ROBOTBUILDER TYPE: Command.

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR

#include "commands/CloseClaw.h"

CloseClaw::CloseClaw(Claw* m_claw)
:m_claw(m_claw){

    // Use AddRequirements() here to declare subsystem dependencies
    // eg. AddRequirements(m_Subsystem);
    SetName("CloseClaw");
    AddRequirements({m_claw});

// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR

}

// Called just before this Command runs the first time
void CloseClaw::Initialize() {
    m_claw->Close(); // (1)
}

// Called repeatedly when this Command is scheduled to run
void CloseClaw::Execute() {

}

// Make this return true when this Command no longer needs to run execute()
bool CloseClaw::IsFinished() {
    return m_claw->IsGripping(); // (2)
}

// Called once after isFinished returns true
void CloseClaw::End(bool interrupted) {
    m_claw->Stop(); // (3)
}

bool CloseClaw::RunsWhenDisabled() const {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
    return false;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
}
```

RobotBuilder générera les fichiers de classe pour la commande *CloseClaw*. La commande représente le comportement de la pince, c'est-à-dire son opération dans le temps. Pour faire fonctionner ce mécanisme de pince très simple, le moteur doit fonctionner pendant 1 seconde dans la direction de fermeture. Le sous-système *Claw* est pourvu de méthodes pour démarrer le moteur dans la bonne direction et pour l'arrêter. La responsabilité des commandes est de faire tourner le moteur pour le délai requis. Les lignes de code qui sont affichées dans les boîtes sont ajoutées pour créer ce comportement.

1. Démarrer le moteur qui controle la pince, pour qu'elle se déplaçe dans la direction de fermeture Ceci est accompli en appelant la méthode `Close()` qui a été ajoutée au sous-système *Claw* la méthode d'initialisation de *CloseClaw* .

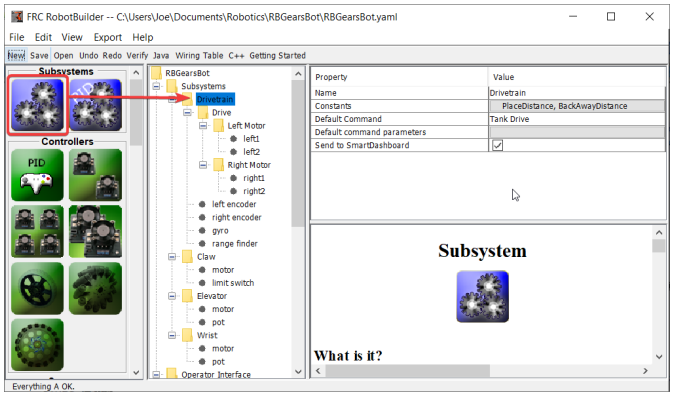
2. This command is finished when the limit switch in the *Claw* subsystem is tripped.
3. La méthode `End()` est alors appelée lorsque la commande est terminée et c'est un moment idéal pour réinitialiser certaines variables et relâcher les ressources qui ne seront plus utilisées. Dans ce cas, on arrête le moteur car le temps est écoulé.

21.2.4 Conduire le robot avec le mode Tank Drive et Joysticks

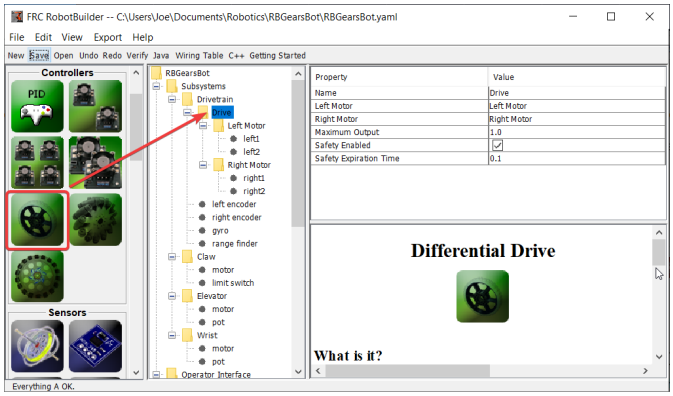
Un cas d'utilisation courant est d'avoir un joystick qui devrait entraîner certains dispositifs actionneurs qui font partie d'un sous-système. Le problème est que le joystick est créé dans la classe OI et que les moteurs à contrôler sont dans le sous-système. L'idée est de créer une commande qui, lorsqu'elle est planifiée, lit les entrées du joystick et appelle une méthode créée sur le sous-système qui entraîne les moteurs.

Dans cet exemple, un sous-système de base d'entraînement est montré qui fonctionne en entraînement de réservoir à l'aide d'une paire de joysticks.

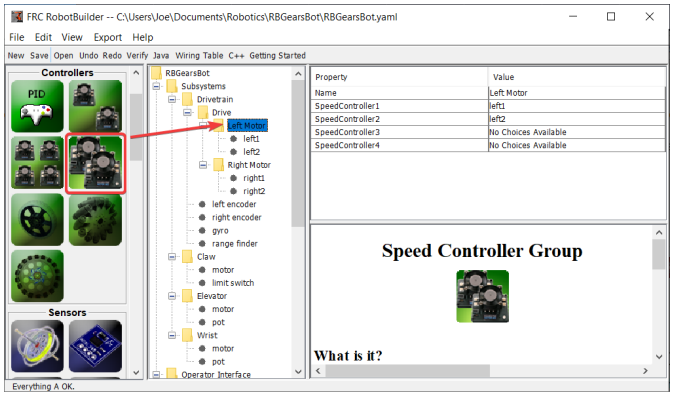
Créer un sous-système d'entraînement



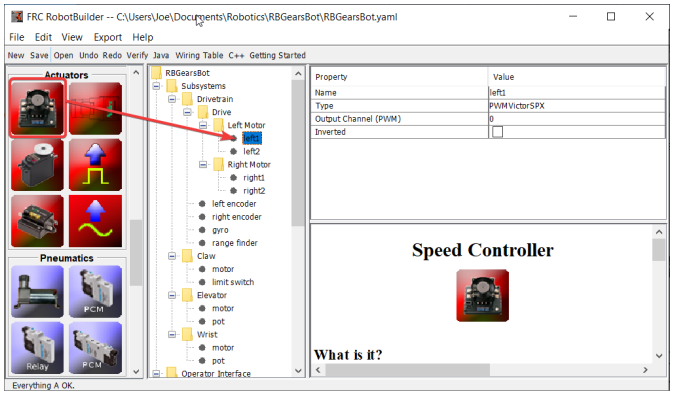
Créez un sous-système appelé Drive Train. Sa responsabilité sera de gérer la conduite de la base de robots.



À l'intérieur du Drive Train créer un objet Differential Drive pour un entraînement à deux moteurs. Il y a un moteur gauche et un moteur droit dans le cadre de la classe Differential Drive.

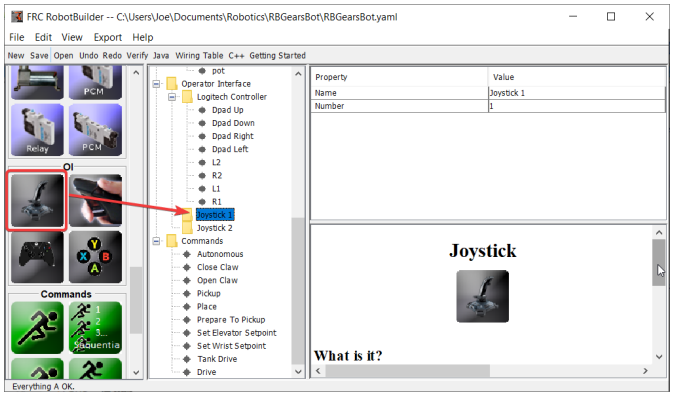


Puisque nous voulons utiliser plus de deux moteurs pour entraîner le robot, à l'intérieur de l'entraînement différentiel, créez deux groupes de contrôleurs de moteur. Ceux-ci regroupe-ront plusieurs contrôleurs de moteur afin qu'ils puissent être utilisés avec l'objet Differential Drive.



Enfin, créez deux contrôleurs de moteur dans chaque groupe de contrôleurs de moteur.

Ajouter les joysticks à l'interface opérateur



Ajoutez deux joysticks à l'interface opérateur, l'un est le joystick gauche et l'autre est le joystick droit. L'axe des y sur les deux joysticks est utilisé pour contrôler les moteurs gauche et droit du robot, respectivement.

Note : Assurez-vous d'exporter votre programme en C++ ou Java avant de passer à l'étape

suivante.

Créer une méthode pour écrire aux moteurs dans le sous-système

java

```
// ROBOTBUILDER TYPE: Subsystem.

package frc.robot.subsystems;


import frc.robot.commands.*;
import edu.wpi.first.wpilibj.livewindow.LiveWindow;
import edu.wpi.first.wpilibj2.command.SubsystemBase;

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
import edu.wpi.first.wpilibj.AnalogGyro;
import edu.wpi.first.wpilibj.AnalogInput;
import edu.wpi.first.wpilibj.CounterBase.EncodingType;
import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import edu.wpi.first.wpilibj.motorcontrol.MotorController;
import edu.wpi.first.wpilibj.motorcontrol.MotorControllerGroup;
import edu.wpi.first.wpilibj.motorcontrol.PWMVictorSPX;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS


/**
 *
 */
public class Drivetrain extends SubsystemBase {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS
public static final double PlaceDistance = 0.1;
public static final double BackAwayDistance = 0.6;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
private PWMVictorSPX left1;
private PWMVictorSPX left2;
private MotorControllerGroup leftMotor;
private PWMVictorSPX right1;
private PWMVictorSPX right2;
private MotorControllerGroup rightMotor;
private DifferentialDrive drive;
private Encoder leftencoder;
private Encoder rightencoder;
private AnalogGyro gyro;
private AnalogInput rangefinder;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS

    /**
    *
```

(suite sur la page suivante)

(suite de la page précédente)

```
    */
    public Drivetrain() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
left1 = new PWMVictorSPX(0);
 addChild("left1",left1);
 left1.setInverted(false);

left2 = new PWMVictorSPX(1);
 addChild("left2",left2);
 left2.setInverted(false);

leftMotor = new MotorControllerGroup(left1, left2  );
 addChild("Left Motor",leftMotor);


right1 = new PWMVictorSPX(5);
 addChild("right1",right1);
 right1.setInverted(false);

right2 = new PWMVictorSPX(6);
 addChild("right2",right2);
 right2.setInverted(false);

rightMotor = new MotorControllerGroup(right1, right2  );
 addChild("Right Motor",rightMotor);


drive = new DifferentialDrive(leftMotor, rightMotor);
 addChild("Drive",drive);
 drive.setSafetyEnabled(true);
drive.setExpiration(0.1);
drive.setMaxOutput(1.0);


leftencoder = new Encoder(0, 1, false, EncodingType.k4X);
 addChild("left encoder",leftencoder);
 leftencoder.setDistancePerPulse(1.0);

rightencoder = new Encoder(2, 3, false, EncodingType.k4X);
 addChild("right encoder",rightencoder);
 rightencoder.setDistancePerPulse(1.0);

gyro = new AnalogGyro(0);
 addChild("gyro",gyro);
 gyro.setSensitivity(0.007);

rangefinder = new AnalogInput(1);
 addChild("range finder", rangefinder);



    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
    }

    @Override
    public void periodic() {
```

(suite sur la page suivante)

(suite de la page précédente)

```
        // This method will be called once per scheduler run

    }

    @Override
    public void simulationPeriodic() {
        // This method will be called once per scheduler run when in simulation

    }

    // Put methods for controlling this subsystem
    // here. Call these from Commands.

    public void drive(double left, double right) {
        drive.tankDrive(left, right);
    }
}
```

C++ (Header ou en-tête)

```
// ROBOTBUILDER TYPE: Subsystem.
#pragma once

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES
#include <frc2/command/SubsystemBase.h>
#include <frc/AnalogGyro.h>
#include <frc/AnalogInput.h>
#include <frc/Encoder.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/motorcontrol/MotorControllerGroup.h>
#include <frc/motorcontrol/PWMVictorSPX.h>

// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES

/**
 *
 *
 * @author ExampleAuthor
 */
class Drivetrain: public frc2::SubsystemBase {
private:
    // It's desirable that everything possible is private except
    // for methods that implement subsystem capabilities
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
frc::AnalogInput m_rangefinder{1};
frc::AnalogGyro m_gyro{0};
frc::Encoder m_rightencoder{2, 3, false, frc::Encoder::k4X};
frc::Encoder m_leftencoder{0, 1, false, frc::Encoder::k4X};
frc::DifferentialDrive m_drive{m_leftMotor, m_rightMotor};
frc::MotorControllerGroup m_rightMotor{m_right1, m_right2  };
frc::PWMVictorSPX m_right2{6};
frc::PWMVictorSPX m_right1{5};
frc::MotorControllerGroup m_leftMotor{m_left1, m_left2  };
frc::PWMVictorSPX m_left2{1};
```

(suite sur la page suivante)

(suite de la page précédente)

```
frc::PWMVictorSPX m_left1{0};

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
public:
Drivetrain();

    void Periodic() override;
    void SimulationPeriodic() override;
    void Drive(double left, double right);
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS
static constexpr const double PlaceDistance = 0.1;
static constexpr const double BackAwayDistance = 0.6;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS


};
```

C++ (Source)

```
// ROBOTBUILDER TYPE: Subsystem.

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES
#include "subsystems/Drivetrain.h"
#include <frc/smartdashboard/SmartDashboard.h>

// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES

Drivetrain::Drivetrain(){
    SetName("Drivetrain");
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    SetSubsystem("Drivetrain");

 AddChild("range finder", &m_rangefinder);


 AddChild("gyro", &m_gyro);
 m_gyro.SetSensitivity(0.007);

 AddChild("right encoder", &m_rightencoder);
 m_rightencoder.SetDistancePerPulse(1.0);

 AddChild("left encoder", &m_leftencoder);
 m_leftencoder.SetDistancePerPulse(1.0);

 AddChild("Drive", &m_drive);
 m_drive.SetSafetyEnabled(true);
m_drive.SetExpiration(0.1_s);
m_drive.SetMaxOutput(1.0);


```

(suite sur la page suivante)

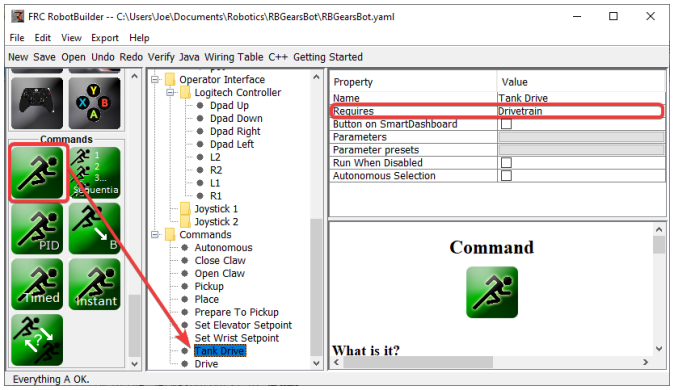
(suite de la page précédente)

```
 AddChild("Right Motor", &m_rightMotor);


 AddChild("right2", &m_right2);
 m_right2.SetInverted(false);

 AddChild("right1", &m_right1);
 m_right1.SetInverted(false);

 AddChild("Left Motor", &m_leftMotor);


 AddChild("left2", &m_left2);
 m_left2.SetInverted(false);

 AddChild("left1", &m_left1);
 m_left1.SetInverted(false);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
}

void Drivetrain::Periodic() {
    // Put code here to be run every loop

}

void Drivetrain::SimulationPeriodic() {
    // This method will be called once per scheduler run when in simulation

}

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS

// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS


// Put methods for controlling this subsystem
// here. Call these from Commands.

    void Drivetrain::Drive(double left, double right) {
        m_drive.TankDrive(left, right);
    }
```

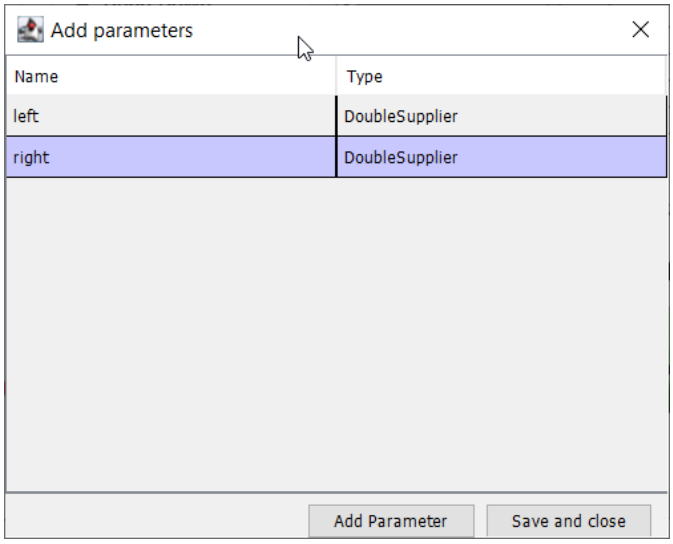
Create a method that takes the joystick inputs, in this case the left and right driver joystick. The values are passed to the DifferentialDrive object that in turn does tank steering using the joystick values. Also create a method called stop() that stops the robot from driving, this might come in handy later.

Note : Certaines sorties du RobotBuilder ont été supprimées dans cet exemple pour plus de clarté

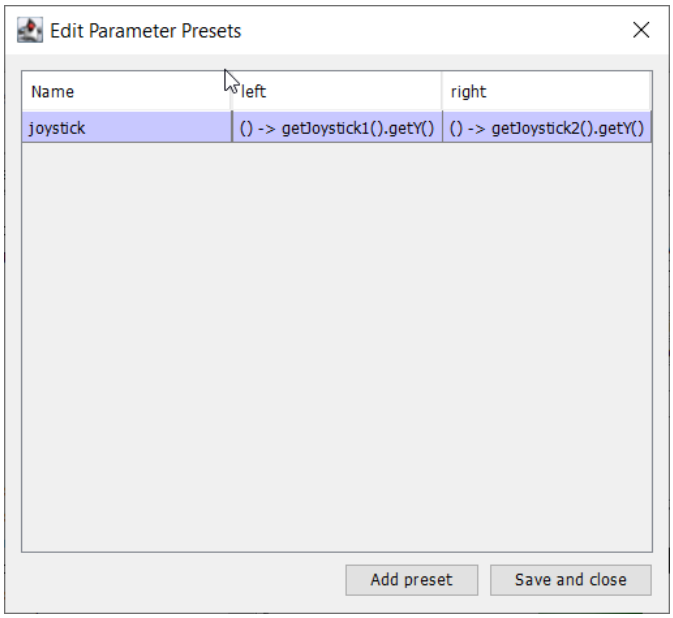
Lire les valeurs des joysticks et appeler les méthodes du sous-système



Créez une commande, dans ce cas appelée Tank Drive. Son but sera de lire une valeur de joystick et de la renvoyer au sous-système « Drive Base ». Notez que cette commande nécessite le sous-système « Drive Train ». Ce qui fera en sorte que cette commande cessera de fonctionner chaque fois que quelque chose d'autre tentera d'utiliser le « Drive Train »



Create two parameters (`DoubleSupplier` for Java or `std::function<double()>` for C++) for the left and right speeds.



Create a parameter preset to retrieve joystick values. Java : For the left parameter enter `() -> getJoystick1().getY()` and for right enter `() -> getJoystick2().getY()`. C++ : For the left parameter enter `[this] {return getJoystick1()->GetY();}` and for the right enter `[this] {return getJoystick2()->GetY();}`

Note : Assurez-vous d'exporter votre programme en C++ ou Java avant de passer à l'étape suivante.

Ajoutez le code pour conduite le robot

java

```
// ROBOTBUILDER TYPE: Command.

package frc.robot.commands;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj2.command.CommandBase;
import frc.robot.RobotContainer;
// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
import frc.robot.subsystems.Drivetrain;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS

/**
 *
 */
public class TankDrive extends CommandBase {

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_DECLARATIONS
        private final Drivetrain m_drivetrain;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_DECLARATIONS
```

(suite sur la page suivante)

(suite de la page précédente)

```

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS


    public TankDrive(Drivetrain subsystem) {


    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES

        m_drivetrain = subsystem;
        addRequirements(m_drivetrain);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
    }

    // Called when the command is initially scheduled.
    @Override
    public void initialize() {
    }

    // Called every time the scheduler runs while the command is scheduled.
    @Override
    public void execute() {
        m_drivetrain.drive(m_left.getAsDouble(), m_right.getAsDouble());
    }

    // Called once the command ends or is interrupted.
    @Override
    public void end(boolean interrupted) {
        m_drivetrain.drive(0.0, 0.0);
    }

    // Returns true when the command should end.
    @Override
    public boolean isFinished() {
        return false;
    }

    @Override
    public boolean runsWhenDisabled() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
        return false;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
    }
}
```

C++ (Header ou en-tête)

```
// ROBOTBUILDER TYPE: Command.

#pragma once

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES

#include <frc2/command/CommandHelper.h>
#include <frc2/command/CommandBase.h>

#include "subsystems/Drivetrain.h"

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES
#include "RobotContainer.h"
#include <frc/Joystick.h>

/**
 *
 *
 * @author ExampleAuthor
 */
class TankDrive: public frc2::CommandHelper<frc2::CommandBase, TankDrive> {
public:
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR
    explicit TankDrive(Drivetrain* m_drivetrain);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR

void Initialize() override;
void Execute() override;
bool IsFinished() override;
void End(bool interrupted) override;
bool RunsWhenDisabled() const override;


private:
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLES


Drivetrain* m_drivetrain;
frc::Joystick* m_leftJoystick;
frc::Joystick* m_rightJoystick;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLES
};
```

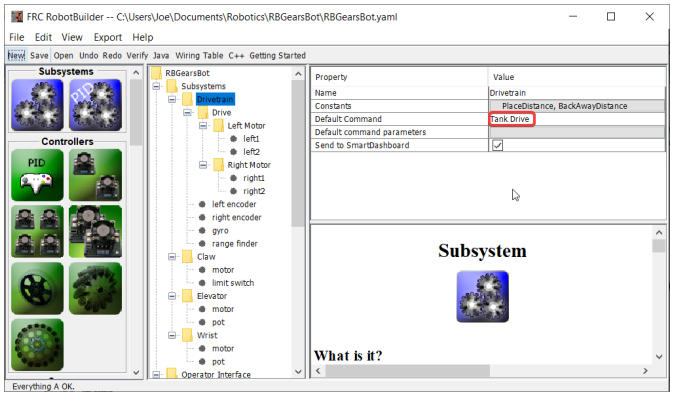
C++ (Source)

```cpp
// ROBOTBUILDER TYPE: Command.

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR

#include "commands/TankDrive.h"

TankDrive::TankDrive(Drivetrain* m_drivetrain)
:m_drivetrain(m_drivetrain){

    // Use AddRequirements() here to declare subsystem dependencies
    // eg. AddRequirements(m_Subsystem);
    SetName("TankDrive");
    AddRequirements({m_drivetrain});

// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR
}

// Called just before this Command runs the first time
void TankDrive::Initialize() {

}

// Called repeatedly when this Command is scheduled to run
void TankDrive::Execute() {
    m_drivetrain->Drive(m_left(),m_right());
}

// Make this return true when this Command no longer needs to run execute()
bool TankDrive::IsFinished() {
    return false;
}

// Called once after isFinished returns true
void TankDrive::End(bool interrupted) {
    m_drivetrain->Drive(0,0);
}

bool TankDrive::RunsWhenDisabled() const {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
    return false;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
}
```

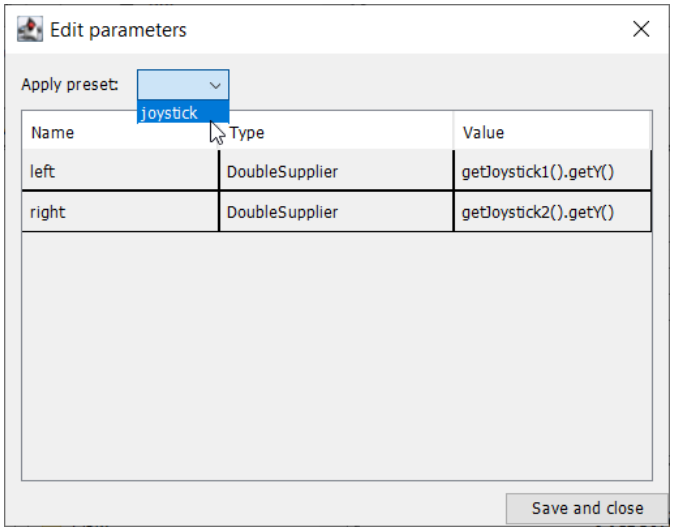
Add code to the execute method to do the actual driving. All that is needed is pass the for the left and right parameters to the Drive Train subsystem. The subsystem just uses them for the tank steering method on its DifferentialDrive object. And we get tank steering.

Nous avons également rempli la méthodes `end()` afin que lorsque cette commande est interrompue ou arrêtée, les moteurs soient arrêtés par mesure de sécurité.

Créer une commande par défaut



La dernière étape consiste à faire de la commande Tank Drive la « commande par défaut » du sous-système « Drive Train ». Cela signifie que les joysticks seront en contrôle tout le temps de la conduite, sauf lorsqu'une autre commande va utiliser « Drive Train ». C'est probablement le comportement souhaitable. Lorsque le code du mode autonome est en cours d'exécution, il nécessitera également « Drive Train » et interrompra la commande Tank Drive. Une fois le code autonome terminé, la commande Tank Drive redémarrera automatiquement (car il s'agit de la commande par défaut) et les opérateurs reprendront le contrôle. Si vous écrivez un code qui effectue une conduite automatisée (en se servant de capteurs, par exemple), ces commandes nécessitent également le DriveTrain et elles aussi vont interrompre la commande Tank Drive pour et prendre le contrôle du déplacement.



The final step is to choose the joystick parameter preset previously set up.

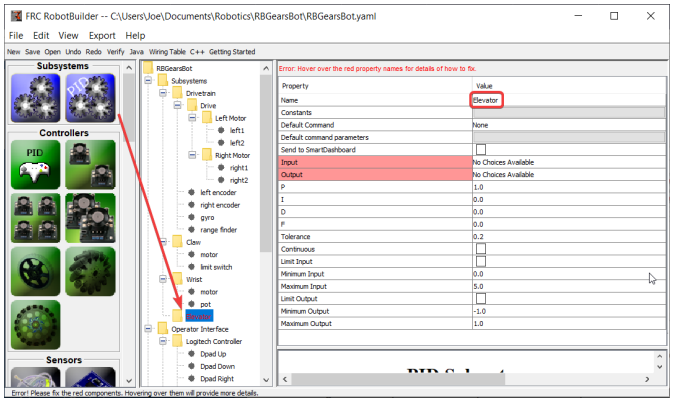
Note : Assurez-vous d'exporter votre programme en C++ ou Java avant de continuer.

21.3 RobotBuilder - Niveau avancé

21.3.1 Utilisation du sous-système PIDS pour contrôler les dispositifs actionneurs

Des sous-systèmes plus avancés utiliseront des capteurs avec rétroaction pour obtenir des résultats garantis pour des opérations telles que le réglage de la hauteur des ascenseurs ou des angles du poignet. Les sous-systèmes PID utilisent la rétroaction pour contrôler l'actionneur et l'amener à une position particulière. Dans cet exemple, nous utilisons un ascenseur avec un potentiomètre à 10 tours connecté pour donner un retour sur la hauteur. Le PIDSubsystem a un PIDController intégré pour contrôler automatiquement le mécanisme aux points de consigne corrects.

Créer un sous-système PIDS

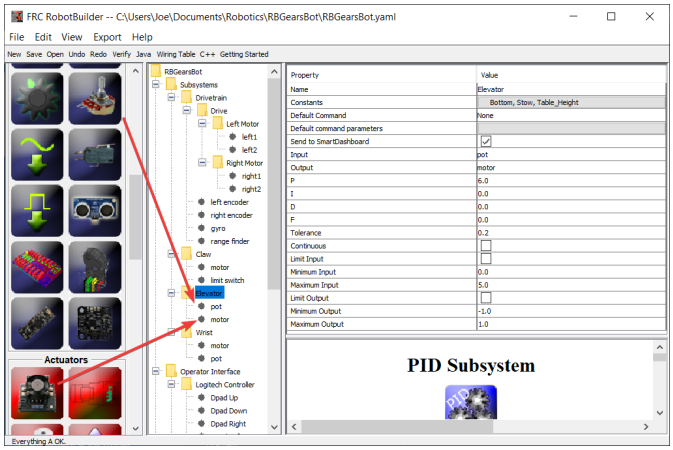


La création d'un sous-système qui utilise la rétroaction pour contrôler la position ou la vitesse d'un mécanisme est très facile.

1. Incorporer un sous-système PIDS dans la description du robot en faisant « glisser » (drag) son icône de la palette vers le dossier Sous-systèmes
2. Renommez le sous-système PID en utilisant un nom plus significatif, et descriptif pour le sous-système, dans ce cas-ci, Elevator

Notez que certaines parties de la description du robot sont maintenant devenues rouges. Cela indique que ces composants (le sous-système PIDS) doivent être complétés et leurs champs doivent être remplis. Les propriétés manquantes ou incorrectes sont affichées en rouge.

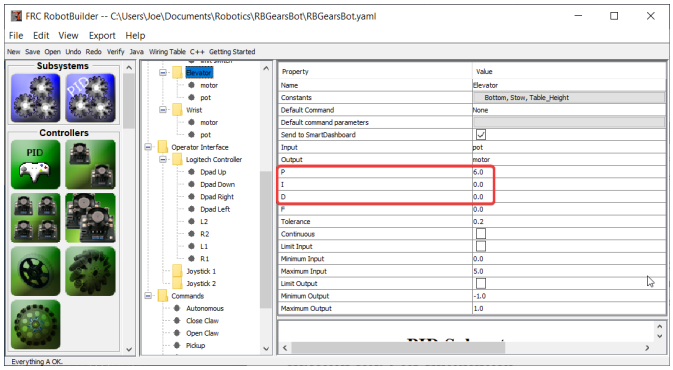
Ajout de capteurs et de dispositifs actionneurs au sous-système PIDS



Ajouter les composants manquants pour le PIDSubsystem

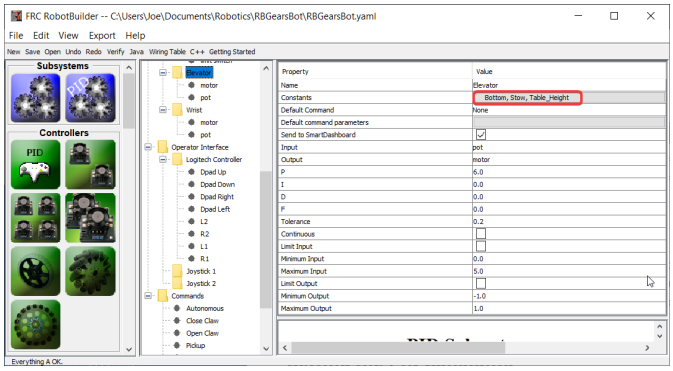
1. Faites glisser l'actionneur (un contrôleur de moteur) vers le sous-système concerné - dans ce cas, l'élevateur
2. Faites glisser le capteur qui sera utilisé pour la rétroaction vers le sous-système, dans ce cas, le capteur est un potentiomètre qui pourrait donner une rétroaction de la hauteur de l'ascenseur.

Remplissez les paramètres du PID

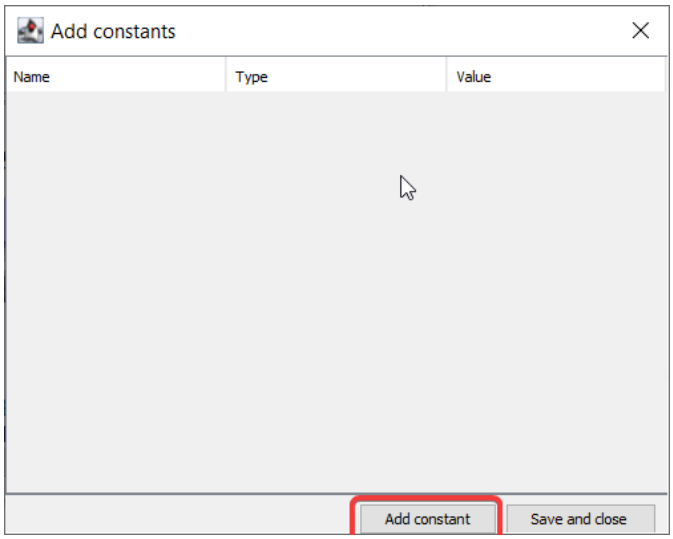


Les valeurs P, I et D doivent être définies pour obtenir la sensibilité et la stabilité souhaitées du mécanisme. Dans le cas de notre élevateur nous utilisons un gain proportionnel de 6.0 et 0 pour les termes I et D.

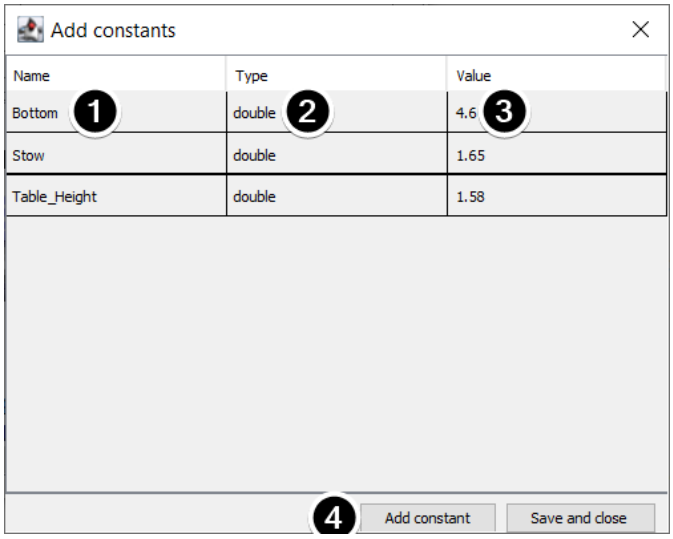
Création des constantes des points de consigne



Afin de faciliter la gestion des points de consigne de l'élevateur, nous créerons des constantes pour précisément gérer ces points de consigne. Cliquez sur la boîte de constants pour la faire faire apparaitre.



Cliquez sur le bouton *add constant*



1. Saisissez le nom de la constante, dans ce cas : Bottom
2. Sélectionnez, à partir du menu déroulant, un type pour la constante, dans ce cas : double
3. Sélectionnez une valeur pour la constante, dans ce cas : 4.65
4. Cliquez sur *add constant* pour continuer à ajouter d'autres constantes
5. Après avoir entré toutes les constantes, cliquez sur *Save and close*

21.3.2 Écrire le code pour PIDSubsystem

Le squelette du PIDSubsystem est généré par le RobotBuilder et nous devons remplir le reste du code pour fournir la valeur du potentiomètre et piloter le moteur avec la sortie du PIDController intégré.

Assurez-vous que le sous-système Elevator PID a été créé dans RobotBuilder. Une fois que tout est défini, générer le code Java/C ++ pour le projet à l'aide du menu Export ou du menu de la barre d'outils Java/C ++.

RobotBuilder génère les méthodes PIDSubsystem de sorte qu'aucun code supplémentaire n'est nécessaire pour le fonctionnement de base.

Définition des constantes PID

Les constantes de hauteur et les constantes PID sont générées automatiquement.

JAVA

```
public class Elevator extends PIDSubsystem {

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS
public static final double Bottom = 4.6;
public static final double Stow = 1.65;
public static final double Table_Height = 1.58;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
private AnalogPotentiometer pot;private PWMVictorSPX motor;
    //P I D Variables
    private static final double kP = 6.0;
    private static final double kI = 0.0;
    private static final double kD = 0.0;
    private static final double kF = 0.0;
    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
```

C++

```
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS
static constexpr const double Bottom = 4.6;
static constexpr const double Stow = 1.65;
static constexpr const double Table_Height = 1.58;

    static constexpr const double kP = 6.0;
    static constexpr const double kI = 0.0;
    static constexpr const double kD = 0.0;
    static constexpr const double kF = 0.0;
    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTANTS
```

Obtenir la mesure du potentiomètre

JAVA

```
@Override
public double getMeasurement() {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SOURCE
    return pot.get();

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SOURCE
}
```

C++

```
double Elevator::GetMeasurement() {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SOURCE
    return m_pot.Get();

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SOURCE
}
```

La méthode `getMeasurement()` est utilisée pour définir la valeur du capteur qui fournit la rétroaction pour le contrôleur PID. Dans ce cas, le code est généré automatiquement et renvoie la tension du potentiomètre telle que renvoyée par la méthode `get()`.

Calculer la sortie PID

JAVA

```
    @Override
    public void useOutput(double output, double setpoint) {
        output += setpoint*kF;
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=OUTPUT
motor.set(output);

        // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=OUTPUT
    }
```

C++

```
void Elevator::UseOutput(double output, double setpoint) {
    output += setpoint*kF;
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=OUTPUT
m_motor.Set(output);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=OUTPUT
}
```

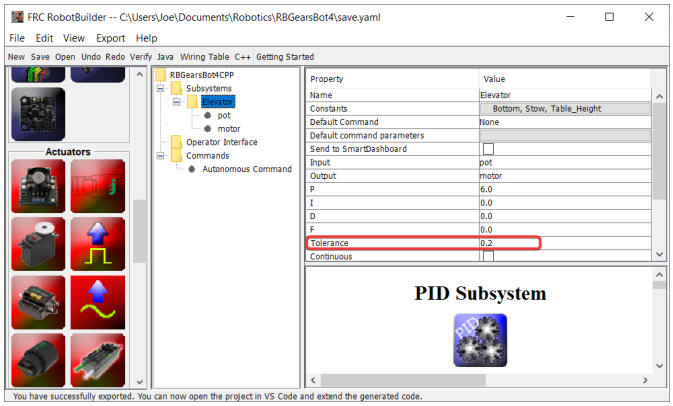
La méthode `useOutput` écrit la sortie PID calculée directement dans le moteur.

C'est tout ce qui est nécessaire pour créer le PIDSubsystem Elevator.

21.3.3 Commande de Point de consigne (Setpoint)

Une commande de point de consigne fonctionne en conjonction avec un sous-système PID pour entraîner un actionneur à un angle ou une position particulière qui est mesurée à l'aide d'un potentiomètre ou d'un encodeur. Cela arrive si souvent qu'il existe un raccourci dans RobotBuilder pour effectuer cette tâche.

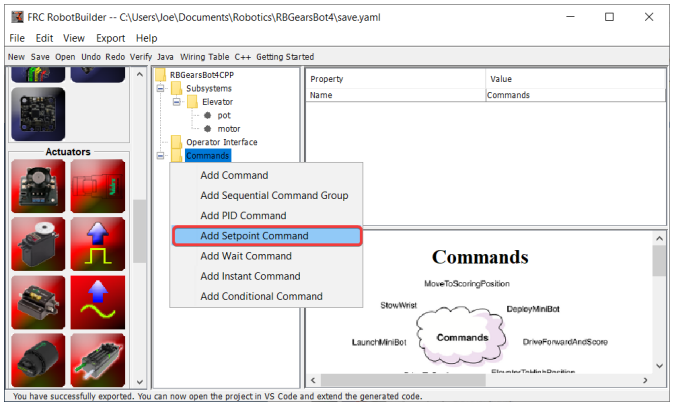
Commencer avec un PIDSubsystem



Supposons que dans un robot il y ait un bras articulé avec un potentiomètre qui mesure l'angle. En premier lieu *créer un PIDSubsystem* qui inclut le moteur qui déplace l'articulation et le potentiomètre qui mesure l'angle. Le sous-système PIDS doit avoir toutes les constantes PID pré-définies et doit pouvoir fonctionner correctement.

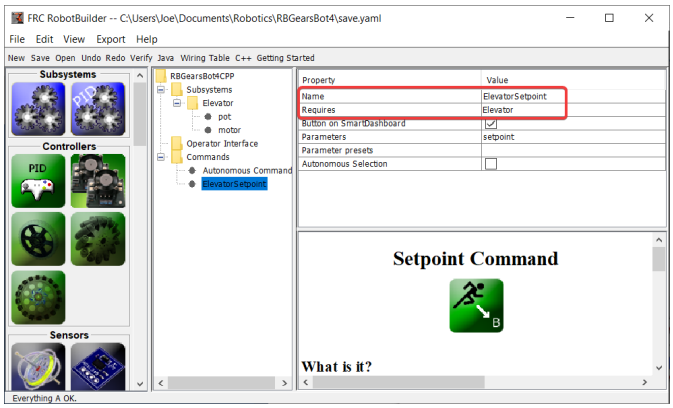
Il est important de définir le paramètre **Tolerance**. Celui-ci permet une distance admissible entre la valeur actuelle et le point de consigne et lorsque la valeur d'erreur est inférieure à « Tolerance », SetpointCommand passe alors à la commande suivante.

Création de la commande du Point de consigne

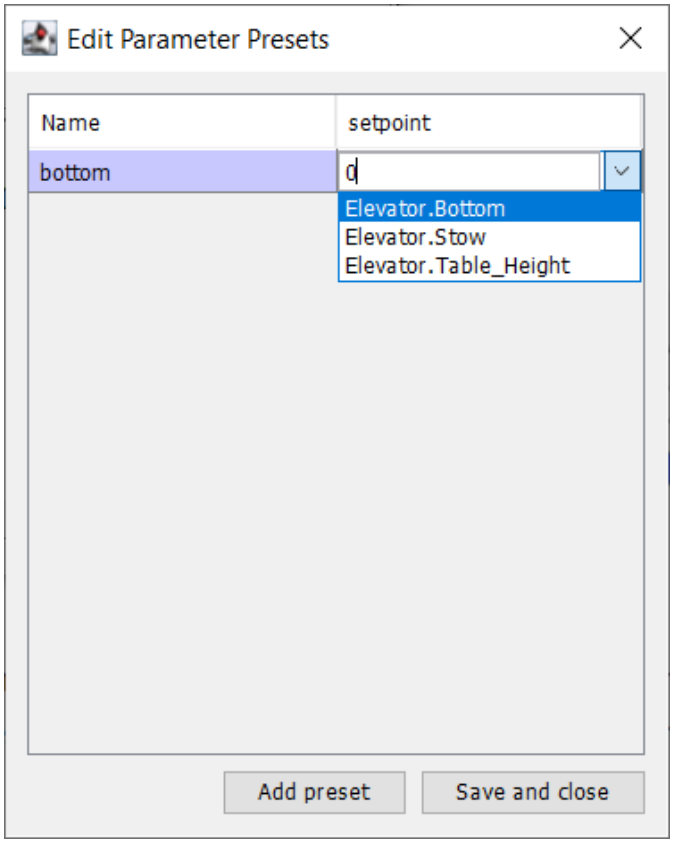


Cliquez avec le bouton droit sur le dossier Commandes dans la palette et sélectionnez « Add Setpoint command ».

Paramètres de commande du point de consigne



Remplissez le nom de la nouvelle commande. Le champ Requires est le PIDSubsystem qui est conduit à un point de consigne, dans ce cas le sous-système Elevator.



1. Cliquez sur Paramètres prédéfinis pour configurer les points de consigne.
2. Sélectionnez *Add Preset*
3. Entrez un nom prédéfini (dans ce cas “bottom”)
4. Cliquez sur la liste déroulante à côté de la zone de saisie du point de consigne
5. Select the Elevator.Bottom constant, that was created in the Elevator subsystem previously
6. Répétez les étapes 2 à 5 pour les autres points de consigne.
7. Cliquez sur *Save and close*

Il n’est pas nécessaire de remplir de code pour cette commande, elle est automatiquement créée par RobotBuilder.

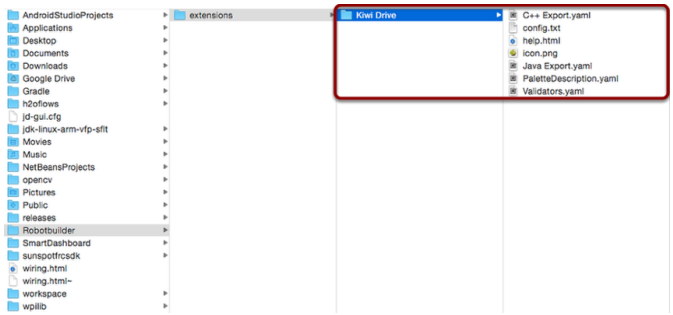
Chaque fois que cette commande est planifiée, elle conduira automatiquement le sous-système au point de consigne spécifié. Lorsque le point de consigne est atteint dans la tolérance spécifiée dans le PIDSubsystem, la commande se termine et la commande suivante démarre. Il est important de spécifier une tolérance acceptable dans le PIDSubsystem sinon cette commande pourrait ne jamais se terminer, faute d’atteindre la tolérance.

Note : Pour plus d’informations sur le contrôle PID, veuillez consulter **:réf :`Introduction aux contrôles avancés <docs/software/advanced-controls/introduction/index :Advanced Controls Introduction>`**.

21.3.4 Ajout de composants personnalisés

RobotBuilder fonctionne très bien pour la création de programmes de robots qui utilisent uniquement la WPILib pour les moteurs, les contrôleurs et les capteurs. Mais pour les équipes qui utilisent des classes personnalisées, RobotBuilder n'a pas de support pour ces classes, donc quelques étapes doivent être prises pour les utiliser dans RobotBuilder

Structure d'un composants personnalisé



Les composants personnalisés vont tous dans `~/wpilib/YYYY/Robotbuilder/extensions` où `~` est `C:\Users\Public` sous Windows et YYYY correspond à l'année de la saison FRC®.

Il y a sept fichiers et un dossier qui sont nécessaires pour un composant personnalisé. Le dossier contient les fichiers décrivant le composant et comment l'exporter. Il doit avoir le même nom que le composant (p. ex. »Kiwi Drive » pour une base pilotable de type kiwi, « Robot Drive 6 » pour une base pilotable à six moteurs, etc.). Les fichiers doivent avoir les mêmes noms et extensions que ceux indiqués ici. D'autres fichiers peuvent être dans le dossier en plus de ces sept, mais les sept doivent être présents pour que RobotBuilder reconnaisse le composant personnalisé.

PaletteDescription.yaml

PaletteDescription.yaml × Validators.yaml × C++ Export.yaml

```
1  !Component
2    name: Kiwi Drive
3    type: Controller
4    supports: {PIDOutput: 3}
5    help: A type of drivetrain with three omni wheels
6    properties:
7      - !ChildSelectionProperty
8          name: Motor 1
9          type: PIDOutput
10         validators: [KiwiDriveValidator, ChildDropdownSelected]
11     - !ChildSelectionProperty
12         name: Motor 2
13         type: PIDOutput
14         validators: [KiwiDriveValidator, ChildDropdownSelected]
15     - !ChildSelectionProperty
16         name: Motor 3
17         type: PIDOutput
18         validators: [KiwiDriveValidator, ChildDropdownSelected]
```

Ligne par ligne :

— !Component : déclare le début d'un nouveau composant
— name : Nom du composant. C'est ce qui s'affichera dans la palette/arbre - cela devrait également être le même que le nom du dossier contenant
— type : le type du composant (ceux-ci seront expliqués en profondeur plus tard)

- supports : une carte de la quantité de chaque type de composant qui peut prendre en charge. Les contrôleurs de moteurs de RobotBuilder sont tous des PIDOutputs, de sorte qu'une base kiwi peut prendre en charge trois PIDOutputs. Si un composant ne prend rien en charge (comme les capteurs ou les contrôleurs de moteurs), juste ignorer cette ligne
- help : une chaîne de caractères concise qui affiche un message utile lorsqu'on passe le curseur sur l'un de ces composants
- properties : liste des propriétés de ce composant. Dans cet exemple de base kiwi, il y a trois propriétés très similaires, une pour chaque moteur. Un ChildSelectionProperty permet à l'utilisateur de choisir un composant du type donné à partir des sous-composants de celui qui est en cours de modification (donc ici, s'afficherait une liste déroulante demandant un PIDOutput - c'est-à-dire un contrôleur de moteur - qui a été ajouté à la base kiwi)

Les types de composants que RobotBuilder prend en charge (ceux-ci sont sensibles à la casse) :

- Command
- Subsystem
- PIDOutput (contrôleur de moteur)
- PIDSource (capteur qui implémente PIDSource, par exemple un potentiomètre analogique, un encodeur)
- Sensor (capteur qui n'implémente pas PIDSource, par exemple un interrupteur de fin de course)
- Controller (déplacement du robot, contrôleur PID, etc.)
- Actuator (une sortie qui n'est pas un moteur, par exemple un solenoïde, un servo)
- Joystick
- Joystick Button

Propriétés

Propriétés pertinentes pour un composant personnalisé :

- StringProperty : utilisé lorsqu'un composant a besoin d'une chaîne de caractères, par exemple le nom du composant
- BooleanProperty : utilisé lorsqu'un composant a besoin d'une valeur booléenne, par exemple en mettant un bouton sur le SmartDashboard
- DoubleProperty : utilisé lorsqu'un composant a besoin d'une valeur numérique, par exemple PID constantsChoicesProperty
- ChildSelectionProperty : ce paramètre est utilisé lorsque vous devez choisir un composant enfant, par exemple des contrôleurs de moteurs dans un RobotDrive
- TypeSelectionProperty : utilisé lorsque vous devez choisir n'importe quel composant du type donné de n'importe où dans le programme, par exemple l'entrée et la sortie pour une commande PID

Les champs de chaque propriété sont décrits ci-dessous :

```
A property is one of:
- !StringProperty
  name: The name of this property, should be unique within this component
  validator: Optional. The validator that should be used to
             validate this property.
  default: The default value when no other is presented.
- !BooleanProperty
  name: The name of this property, should be unique within this component
  validator: Optional. The validator that should be used to
             validate this property.
  default: The default value when no other is presented.
- !DoubleProperty
  name: The name of this property, should be unique within this component
  validator: Optional. The validator that should be used to
             validate this property.
  default: The default value when no other is presented.
- !FileProperty
  name: The name of this property, should be unique within this component
  validator: Optional. The validator that should be used to
             validate this property.
  default: The default value when no other is presented.
  extension: The extension at the end of this file without the '.'
  folder: Whether or not to select folders instead of files
- !ChoicesProperty
  name: The name of this property, should be unique within this component
  validator: Optional. The validator that should be used to
             validate this property.
  default: The default value when no other is presented.
  choices: List of choices to present to the user.
- !ChildSelectionProperty
  name: The name of this property, should be unique within this component
  validator: Optional. The validator that should be used to
             validate this property.
  default: The default value when no other is presented.
  type: Type of the child to select.
- !TypeSelectionProperty
  name: The name of this property, should be unique within this component
  validator: Optional. The validator that should be used to
             validate this property.
  default: The default value when no other is presented.
  type: Type of component to select.
```

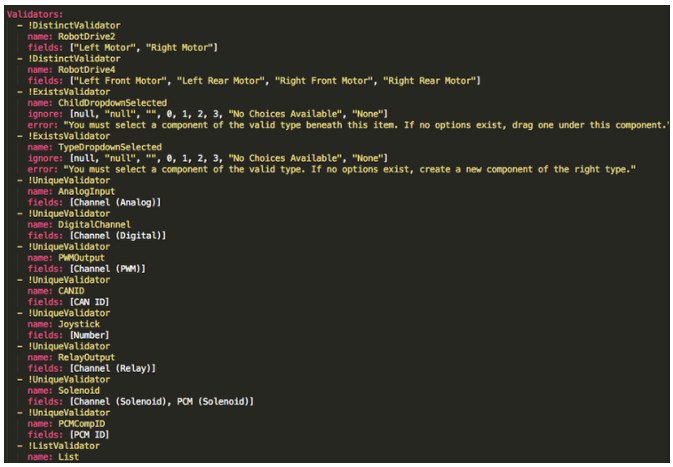
Validators.yaml

Validators.yaml

```
1  !DistinctValidator
2    name: KiwiDriveValidator
3    fields: ["Motor 1", "Motor 2", "Motor 3"]
```

Vous avez peut-être remarqué « KiwiDriveValidator » dans l'entrée validateurs de chacune des propriétés des moteurs dans PaletteDescription.yaml. Ce n'est pas un validateur intégré, il a donc dû être défini dans Validators.yaml. Cet exemple de validateur est très simple - il s'assure simplement que chacun des champs nommés a une valeur différente des autres.

Validateurs intégrés et types de validateurs



Les validateurs intégrés sont très utiles (en particulier les UniqueValidators pour l'utilisation port/canal), mais parfois un validateur personnalisé est nécessaire, comme dans l'étape précédente

- DistinctValidator : Garantit que les valeurs de chacun des champs donnés sont uniques
- ExistsValidator : Garantit qu'une valeur a été définie pour la propriété à l'aide de ce validateur
- UniqueValidator : S'assure que la valeur de la propriété est unique globalement pour les champs donnés
- ListValidator : S'assure que toutes les valeurs d'une liste propriété sont valides

C++ Export.yaml

Analyse du fichier ligne par ligne

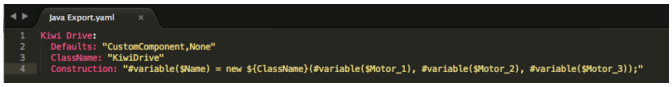
- Kiwi Drive : Le nom du composant exporté. Il s'agit du même nom que le nom défini dans PaletteDescription.yaml, et le nom du dossier contenant ce fichier
- Defaults : fournit certaines valeurs par défaut pour les fichiers d'inclusion nécessaires à ce composant, le nom de la classe, un modèle de construction, et plus encore. Le CustomComponent par défaut ajoute une inclusion pour `Custom/${ClassName}.h` à chaque fichier généré qui utilise le composant (par exemple `RobotDrive.h` aurait `#include Custom/KiwiDrive.h` en haut du fichier)
- ClassName : nom de la classe personnalisée que vous ajoutez.
- Construction : une instruction sur la façon dont le composant doit être construit. Les variables seront remplacées par leurs valeurs (« `${ClassName}` » sera remplacé par « KiwiDrive »), puis les macros seront évaluées (par exemple, `#variable($Name)` pourrait être remplacée par `drivebaseKiwiDrive`).

Cet exemple s'attend à une classe KiwiDrive avec le constructeur

```
KiwiDrive(SpeedController, SpeedController, SpeedController)
```

Si votre équipe utilise Java, ce fichier peut être laissé vide.

Java Export.yaml



Très similaire au fichier d'exportation C++ ; la seule différence devrait être la ligne de construction. Cet exemple s'attend à une classe KiwiDrive avec le constructeur

```
KiwiDrive(SpeedController, SpeedController, SpeedController)
```

Si votre équipe utilise C++, ce fichier peut être laissé vide.

Utilisation de macros et de variables

Les macros sont des fonctions simples que RobotBuilder utilise pour transformer des variables en texte qui seront insérées dans le code généré. Ils commencent toujours par le Symbole « # », et ont une syntaxe similaire aux fonctions : `<macro_name>( arg0, arg1, arg2, ...)`. La seule macro que vous aurez probablement besoin d'utiliser est `#variable( component_name )`

`#variable` accepte une chaîne de caractères, généralement la variable définie quelque part (c.-à-d. « Name » est le nom donné au composant dans RobotBuilder, tel que « Arm Motor »), et le transforme en nom d'une variable définie dans le code généré. Par exemple,``#variable(« Arm Motor »)`` se traduit par la chaîne de caractères `ArmMotor`

Les variables sont référencées en plaçant un signe dollar (« $ ») devant le nom de la variable, qui peut optionnellement être placé à l'intérieur des accolades pour distinguer facilement la variable d'un autre texte dans le fichier. Lorsque le fichier est analysé, le signe dollar, le nom de la variable et les accolades sont remplacés par la valeur de la variable (p. ex. `${ClassName}` est remplacé par `KiwiDrive`).

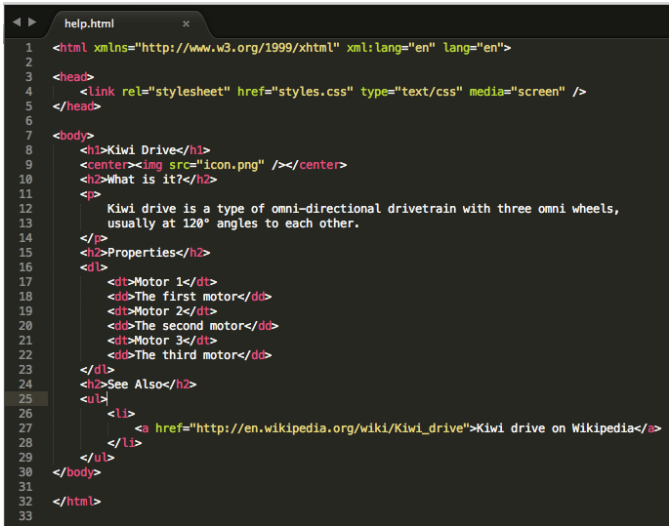
Les variables sont soit des propriétés de composants (p. ex. « Motor 1 », « Motor 2 », « Motor 3 » dans l'exemple de l'entraînement kiwi), soit l'un des suivants :

1. Short_Name : le nom donné au composant dans le panneau d'édition de RobotBuilder
2. Name : le nom complet du composant. Si le composant se trouve dans un sous-système, il s'agit d'un version écourtée du nom ajoutée au nom du sous-système
3. Export : Le nom du fichier dans lequel ce composant doit être créé, le cas échéant. Cela doit être « RobotMap » pour les composants comme les actionneurs, les contrôleurs et les capteurs ; ou « OI » pour des composants comme les gamepads ou d'autres composants OI personnalisés. Notez que le défaut « CustomComponent » s'exportera vers le RobotMap.
4. Import : Les fichiers qui doivent être inclus ou importés pour que ce composant puisse être utilisé.
5. Declaration : une instruction, similaire à Construction, pour savoir comment déclarer une variable de ce type de composant. Ceci est pris en charge par la valeur par défaut « None »
6. Construction : une instruction montrant comment créer une nouvelle instance de ce composant
7. LiveWindow : une instruction montrant comment ajouter ce composant au LiveWindow
8. Extra : les instructions relatives à toutes les fonctions ou méthodes exigent que ce composant se fonctionne correctement, tels que les encodeurs qui doivent définir le type d'encodage.

9. Prototype (C++ uniquement) : Le prototype pour la création d'une fonction dans le fichier dans lequel le composant est déclaré, généralement un getter de la classe OI
10. Function : Une fonction à créer dans le fichier dans lequel le composant est déclaré, généralement un getter dans la classe OI
11. PID : Une instruction montrant la façon d'obtenir la sortie PID du composant, si elle en a une (p. ex. `#variable($Short_Name)->PIDGet()`)
12. ClassName : Le nom de la classe que représente le composant (p. ex. ""KiwiDrive"" ou ""Joystick"")

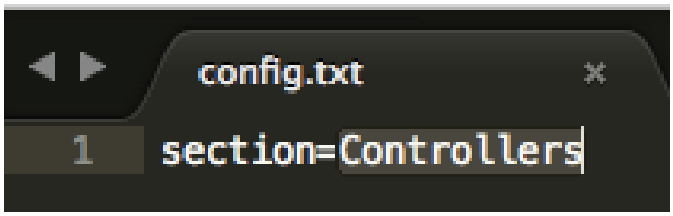
Si vous avez des variables avec des espaces dans le nom (tels que »Motor 1 », « Right Front Motor », etc.), les espaces doivent être remplacés par des soulignements lors de leur utilisation dans les fichiers d'exportation.

help.html



Un fichier HTML donnant des informations sur le composant. Il est préférable qu'il soit aussi détaillé que possible, mais il n'est certainement pas nécessaire si le programmeur (s) est ou sont assez familier.s avec le composant, ou si c'est si simple qu'il inutile d'avoir une description détaillée.

config.txt



Un fichier de configuration pour contenir diverses informations sur le composant. Actuellement, il y a que la section de la palette pour contenir le composant

Les sections de la palette (celles-ci sont sensibles à la casse) :
— Subsystems

— Controllers
— Sensors
— Actuators
— Pneumatics
— OI
— Commands

icon.png

L'Icône qui apparaît dans la palette et la page d'aide. Il devrait être un fichier 64x64 `.png`.

It should use the color scheme and general style of the section it's in to avoid visual clutter, but this is entirely optional. Photoshop `.psd` files of the icons and backgrounds are in src/main/icons/icons and png files of the icons and backgrounds are in src/main/resources/icons.

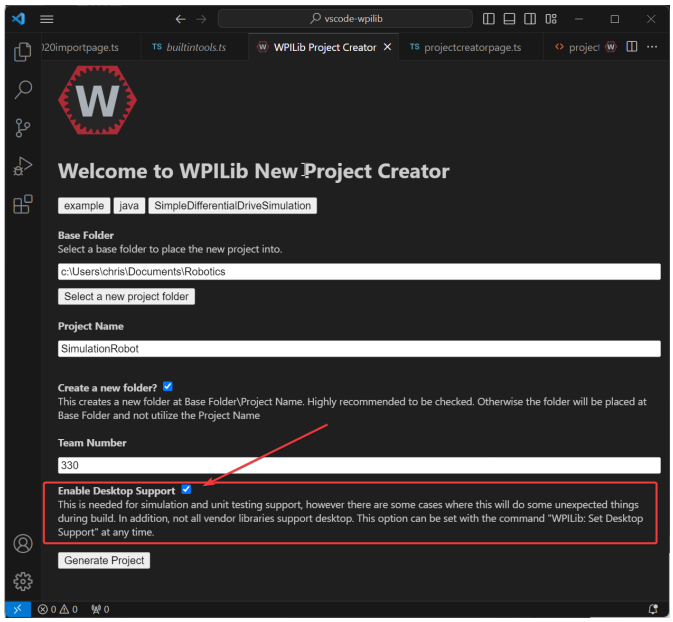
22 Simulation du Robot

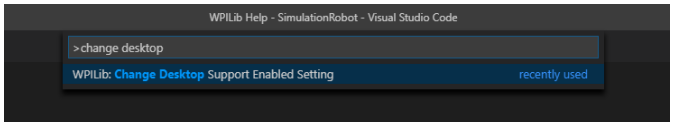
22.1 Introduction au simulateur de Robot

Le simulateur «Robot Simulation» rend possible la programmation d'un robot virtuel dans le cas où une équipe ne dispose d'un véritable robot. WPILib offre aux équipes la possibilité de simuler diverses fonctionnalités du robot virtuellement à l'aide de simples commandes Gradle.

Java/C++

L'utilisation du simulateur nécessite l'activation de *Desktop Support*. Cela peut se faire en cochant la case à cocher « Enable Desktop Support Checkbox » lors de la création de votre projet de robot ou en exécutant « WPILib : Change Desktop Support Enabled Setting » à partir de la palette de commandes Visual Studio Code.





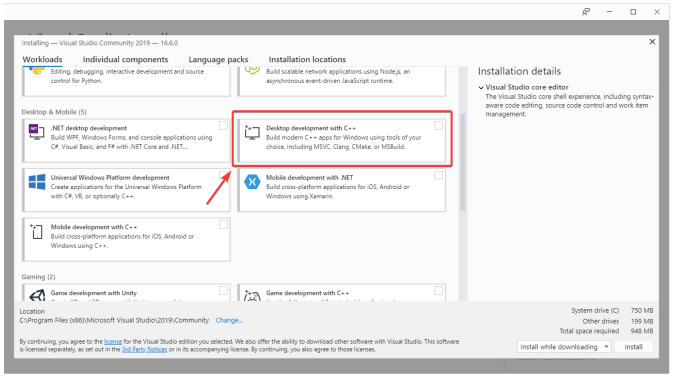
La prise en charge du bureau peut également être activée en modifiant manuellement votre fichier `build.gradle` situé à la racine de votre projet de robot. Il suffit de changer `includeDesktopSupport = false` pour `includeDesktopSupport = true`

Important : Il est important de noter que l'activation de la prise en charge de bureau/simulation peut avoir des conséquences imprévues. Tous les fournisseurs ne prennent pas en charge cette option, et le code qui utilise leurs bibliothèques peut même se planter lors de la tentative d'exécution de la simulation !

If at any point in time you want to disable Desktop Support, simply re-run the « WPILib : Change Desktop Support Enabled Setting » from the command palette or change `includeDesktopSupport` to false in build.gradle.

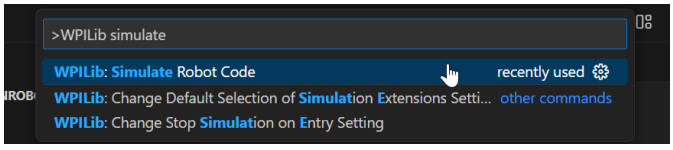
Note : C++ robot simulation requires that a native compiler to be installed. For Windows, this would be Visual Studio 2022 version 17.9 or later (**not** VS Code), macOS requires Xcode 14 or later, and Linux (Ubuntu) requires the `build-essential` package.

Assurez-vous que l'option *Desktop Development with C++* est cochée dans le programme d'installation de Visual Studio pour la prise en charge de la simulation.



Running Robot Simulation

La simulation sommaire du robot de base peut être réalisée à l'aide du code VS. Cela peut se faire sans utiliser de commandes autres que celles de la palette de commandes de VS Code.



La sortie de votre console dans Visual Studio Code doit ressembler à ce qui apparaît ci-dessus. Cependant, les équipes voudront probablement réellement *tester* leur code au lieu de simplement éxécuter la simulation. Cela peut être fait à l'aide de *WPILib's Simulation GUI*.

```
********** Robot program starting **********
Default disabledInit() method... Override me!
```

(suite sur la page suivante)

(suite de la page précédente)

```
Default disabledPeriodic() method... Override me!
Default robotPeriodic() method... Override me!
```

Important : La simulation peut également être exécutée en dehors de VS Code en utilisant `./gradlew simulateJava` pour Java ou `./gradlew simulateNative` pour C++.

Note : Some vendors support attaching hardware to your PC and using the hardware in desktop simulation (e.g. CANivore). See *vendor documentation* for more information about the command *WPILib : Hardware Sim Robot Code*.

Python

GUI simulation support is installed by default when you install RobotPy.

There is a `robotpy` subcommand that you can execute to run your code in simulation :

Windows

```
py -3 -m robotpy sim
```

macOS

```
python3 -m robotpy sim
```

Linux

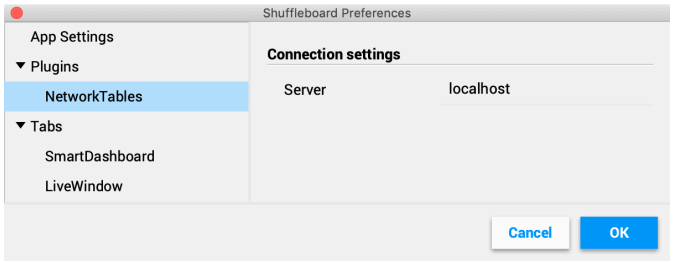
```
python3 -m robotpy sim
```

22.1.1 Exécution des tableaux de bord du robot

Shuffleboard, SmartDashboard, Glass, and AdvantageScope can be used with WPILib simulation when they are configured to connect to the local computer (i.e. `localhost`).

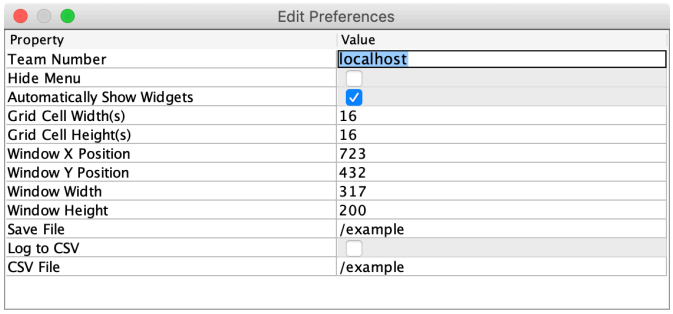
Shuffleboard

Shuffleboard is automatically configured to look for a NetworkTables instance from the roboRIO but **not from other sources**. To connect to a simulation, open Shuffleboard preferences from the *File* menu and select *NetworkTables* under *Plugins* on the left navigation bar. In the *Server* field, type in the IP address or hostname of the NetworkTables host. For a standard simulation configuration, use `localhost`.



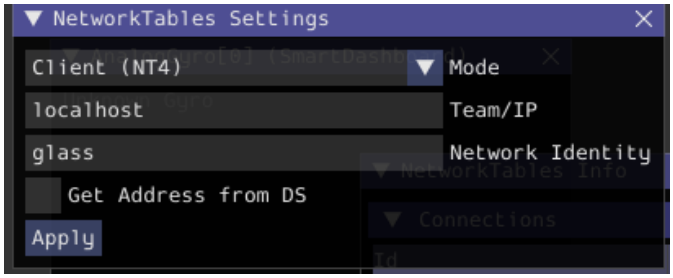
SmartDashboard

SmartDashboard is automatically configured to look for a NetworkTables instance from the roboRIO, but **not from other sources**. To connect to a simulation, open SmartDashboard preferences under the *File* menu and in the *Team Number* field, enter the IP address or hostname of the NetworkTables host. For a standard simulation configuration, use `localhost`.



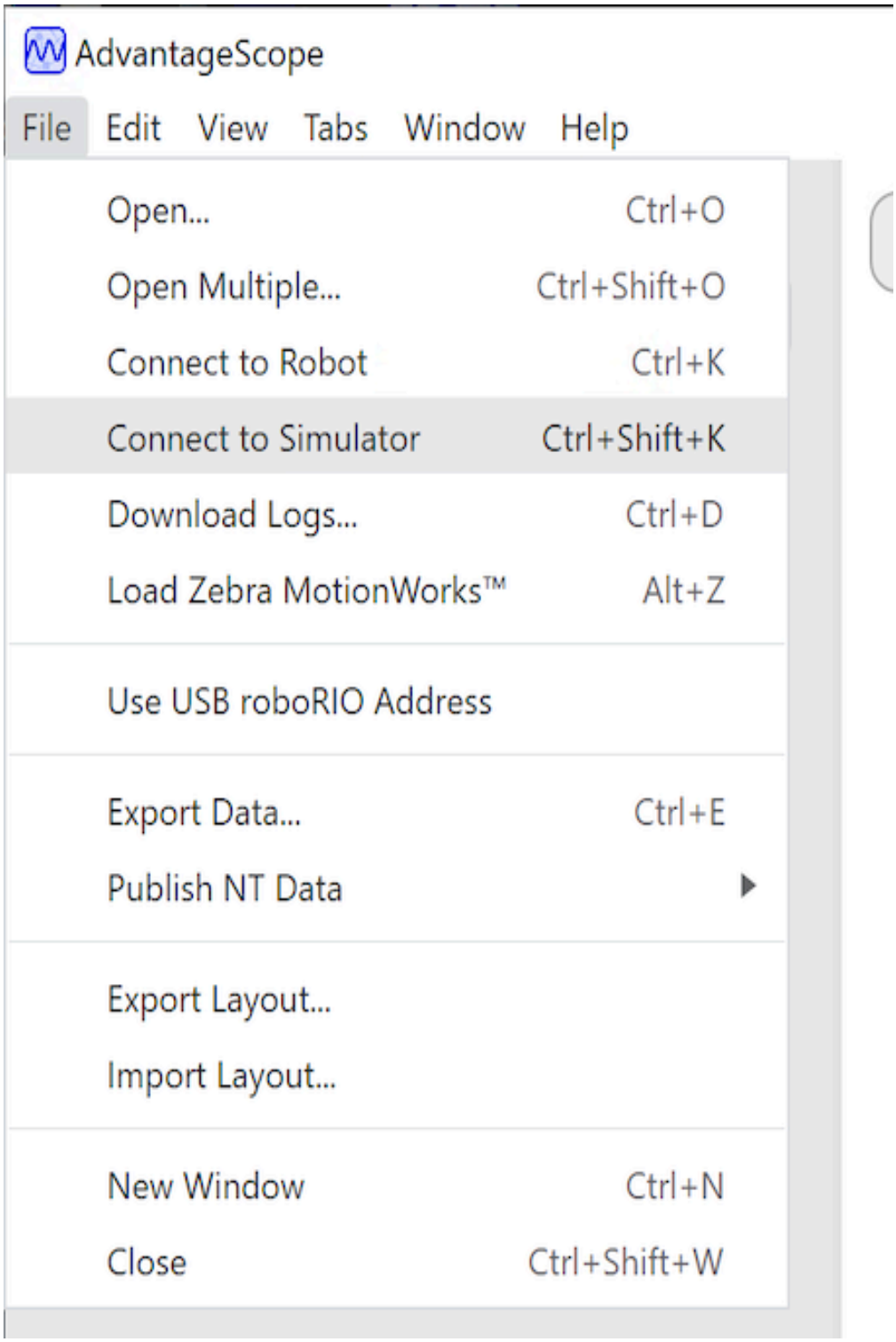
Glass

Glass is automatically configured to look for a NetworkTables instance from the roboRIO, but **not from other sources**. To connect to a simulation, open *NetworkTables Settings* under the *NetworkTables* menu and in the *Team/IP* field, enter the IP address or hostname of the NetworkTables host. For a standard simulation configuration, use `localhost`.



AdvantageScope

No configuration is required to connect to a NetworkTables instance running on the local computer. To connect to a simulation, click *Connect to Simulator* under the *File* menu or press `Ctrl+Shift+K`.

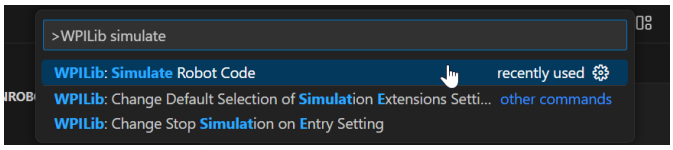


22.2 Simulation d'éléments spécifiques de l'interface utilisateur

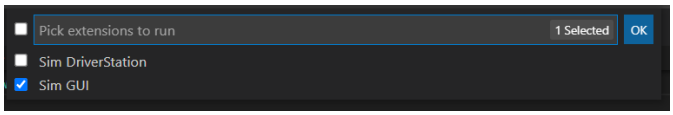
WPILib a renforcé la simulation du robot en introduisant un composant d'interface utilisateur graphique (GUI). Cela permet aux équipes de visualiser facilement les entrées et les sorties de leur robot.

Note : L'Interface Utilisateur Graphique de simulation est très similaire à bien des égards à *Glass*. Certaines des pages suivantes seront liées à des sections Glass qui décrivent des éléments communs aux deux interfaces graphiques.

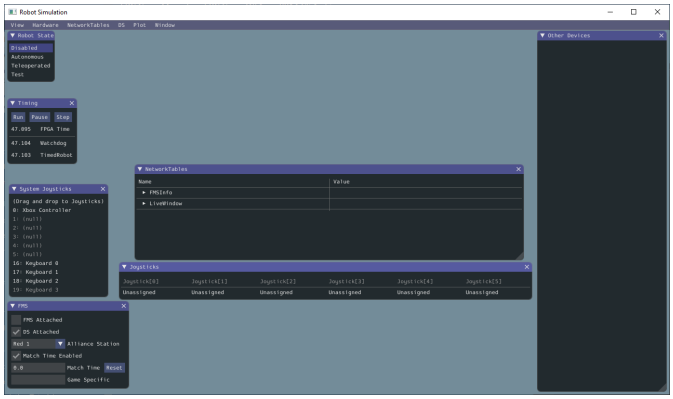
22.2.1 Exécution de l'interface graphique



Vous pouvez simplement démarrer l'interface graphique via l'option de palette de commandes **Run Simulation**.

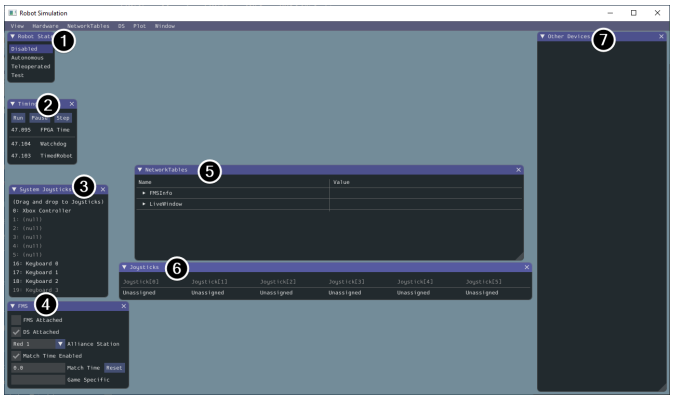


Et l'option `Sim GUI` devrait apparaître dans une nouvelle boîte de dialogue et sera sélectionnée par défaut. Appuyez sur *Ok*. Cette opération va à présent lancer l'interface graphique de simulation !



22.2.2 Utilisation de l'interface graphique

Comprendre la disposition graphique



Les éléments suivants sont affichés sur l'interface graphique de simulation par défaut :

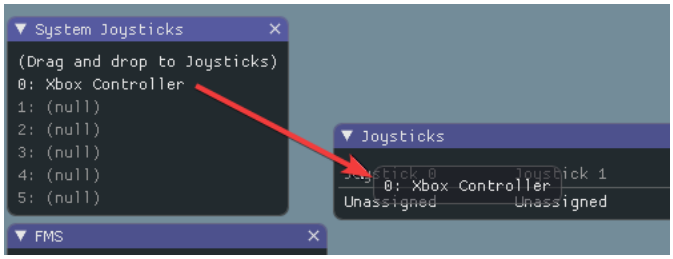
1. **Robot State** - Il s'agit de l'état actuel du robot ou « mode ». station de pilotage (si c'est l'emplacement physique des pilotes), ou driver station (si c'est le logiciel) Vous pouvez cliquer sur les étiquettes pour changer de mode comme vous le feriez sur la Driver Station.
2. **Timing** - Affiche les valeurs des minuteurs ou timers du robot et permet de gérer le timing.
3. **System Joysticks** - Il s'agit de la liste des joysticks actuellement connectés à votre système.
4. **FMS** - This is used for simulating many of the common *FMS* systems.
5. **NetworkTables** - Ceci affiche les données qui ont été publiées sur les NetworkTables.
6. **Joysticks** - Ce sont des joysticks dont le code robot peut tirer directement des données.
7. **Other Devices** - Cela inclut les dispositifs qui ne font partie d'aucune des autres catégories, tel que le gyroscope ADXRS450 qui est inclus dans le kit de pièces ou les périphériques tiers qui prennent en charge la simulation.

Les éléments suivants peuvent être ajoutés à partir du menu Hardware, mais ne sont pas affichés par défaut.

1. **Addressable LEDs** - Ceci montre les LED contrôlées par la classe `AddressableLED`
2. **Analog Inputs** - Cela inclut tous les dispositifs qui utiliseraient normalement le connecteur **ANALOG IN** sur le roboRIO, tels que n'importe quel gyros analogique.
3. **DIO** - (Digital Input Output ou Entrée Sortie Numérique) Cela inclut tous les dispositifs qui utilisent le connecteur **DIO** sur le roboRIO.
4. **Encoders** - Cela affichera tous les dispositfs instanciés qui prolongent ou utilisent la classe `Encoder`.
5. **PDPs** - Ceci montre l'objet Power Distribution Panel (Panneau de distribution de puissance)
6. **PWM Outputs** - This is a list of instantiated *PWM* devices. This will appear as many devices as you instantiate in robot code, as well as their outputs.
7. **Relays** - Cela inclut tous les dispositifs de relais. Cela inclut les relais VEX Spike.
8. **Solenoids** - Il s'agit de la liste des solénoïdes dans l'état « connected ». Lorsque vous créez un objet solenoid et poussez (extend) les sorties, celles-ci sont affichées ici.

Ajout d'un joystick du système aux joysticks

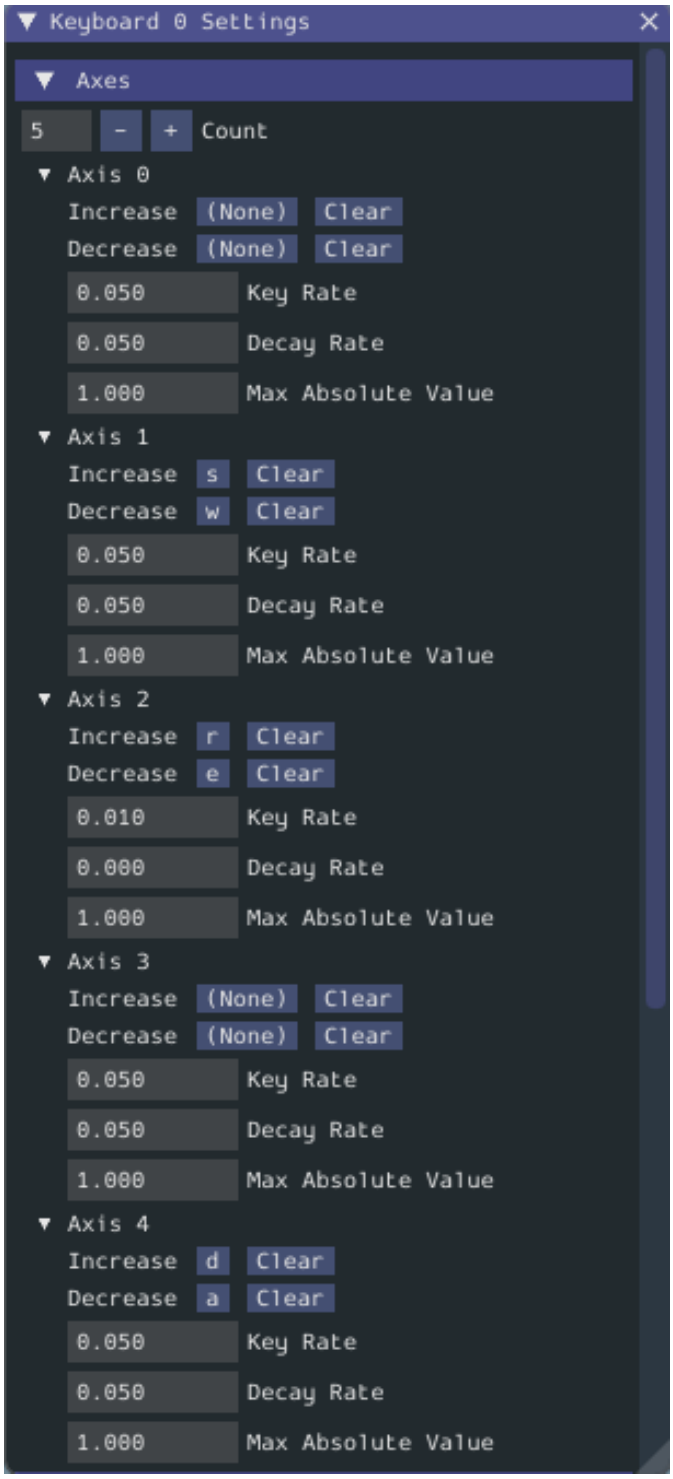
Pour ajouter un joystick à partir de la liste des joysticks du système, il suffit de cliquer et de faire glisser un joystick affiché sous le menu « System Joysticks » au menu « Joysticks ».



Note : L'application FRC® Driver Station établit une correspondance spéciale pour les manettes de jeu connectées et le simulateur WPILib ne fait pas par défaut cette « correspondance ». Vous pouvez activer ce comportement en appuyant sur la bascule « Map gamepad » sous le menu « Joysticks ».

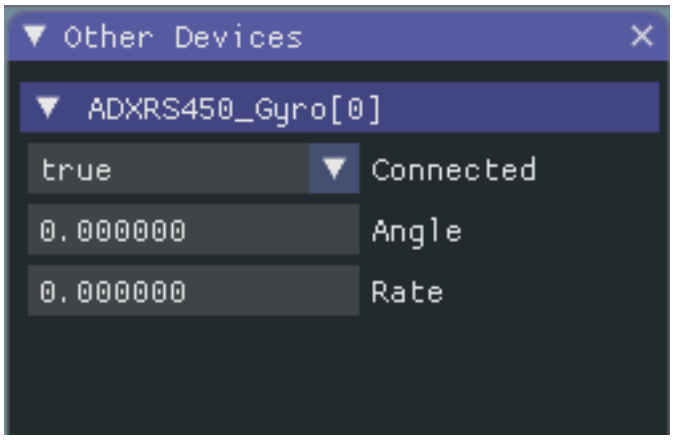
Utilisation du clavier comme joystick

Vous ajoutez un clavier à la liste des joysticks système en cliquant et en faisant glisser l'un des éléments du clavier (par exemple clavier 0) tout comme un joystick ci-dessus. Pour modifier les paramètres du clavier, rendez-vous sur l'élément *DS* dans la barre de menu, puis choisissez *Keyboard 0 Settings*. Cela vous permet de contrôler quels boutons de clavier contrôlent quel axe. Il s'agit d'un exemple courant montrant la façon de rendre le clavier semblable à un split sticks arcade drive sur une manette Xbox (utilise l'axe 1 & 4) :


Keyboard 0 Settings
Axes
5
-
+
Count
Axis 0
Increase
(None)
Clear
Decrease
(None)
Clear
0.050
Key Rate
0.050
Decay Rate
1.000
Max Absolute Value
Axis 1
Increase
s
Clear
Decrease
w
Clear
0.050
Key Rate
0.050
Decay Rate
1.000
Max Absolute Value
Axis 2
Increase
r
Clear
Decrease
e
Clear
0.010
Key Rate
0.000
Decay Rate
1.000
Max Absolute Value
Axis 3
Increase
(None)
Clear
Decrease
(None)
Clear
0.050
Key Rate
0.050
Decay Rate
1.000
Max Absolute Value
Axis 4
Increase
d
Clear
Decrease
a
Clear
0.050
Key Rate
0.050
Decay Rate
1.000
Max Absolute Value

Modification des entrées ADXRS450

L'utilisation de l'objet ADXRS450 est un excellent moyen de tester des sorties du gyroscope. Pour le voir, allez au menu « Other Devices ». Un menu déroulant est ensuite exposé qui affiche différentes options telles que « Connected », « Angle », et « Rate ». Toutes ces valeurs sont des valeurs que vous pouvez modifier, et que votre code robot peut utiliser à la volée.



22.2.3 Détermination de la simulation à partir du code robot

Dans les cas où les librairies de fournisseurs ne compilent pas lors de l'exécution de la simulation de robot, vous pouvez envelopper leur contenu avec `RobotBase.isReal()` laquelle renvoie un `boolean`.

JAVA

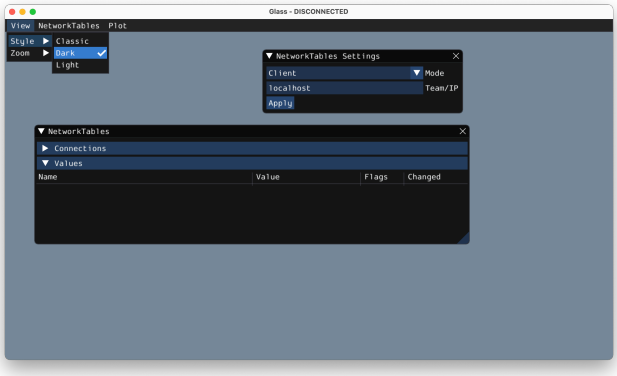
```
TalonSRX motorLeft;
TalonSRX motorRight;

public Robot() {
 if (RobotBase.isReal()) {
   motorLeft = new TalonSRX(0);
   motorRight = new TalonSRX(1);
 }
}
```

Note : La réaffectation des types de valeurs en C++ nécessite un déplacement ou une affectation de copie ; les classes venant de fournisseurs qui ne supportent pas la carte SIM et qui n'ont pas d'opérateur d'affectation de déplacement ou d'affectation de copie ne peuvent pas être prises en charge par une allocation conditionnelle à moins qu'un pointeur ne soit utilisé, au lieu d'un type de valeur.

22.2.4 Modification des paramètres d'affichage

L'élément de menu *View* contient les réglages *Zoom* et *Style* : qui peuvent être personnalisés. L'option *Zoom* dicte la taille du texte dans l'application tandis que l'option *Style* vous permet de choisir entre les options `Classic`, `Light`, et `Dark`.

Un exemple du paramètre de style `Dark` est ci-dessous :



22.2.5 Effacement des données d'application

Les données d'application pour l'interface graphique de simulation, y compris la taille et les positions des widgets ainsi que d'autres informations personnalisées pour les widgets, sont stockées dans un fichier `imgui.ini`. Ce fichier est situé dans la racine du dossier contenant le projet à partir duquel la simulation est lancée.

Le fichier de configuration `imgui.ini` peut simplement être supprimé pour restaurer l'interface graphique de simulation à un « état vierge ».

22.3 Simulation physique avec WPILib

Parce que *la notation de l'espace d'états* nous permet de représenter de manière compacte la *dynamique* d'un *système*, nous pouvons en tirer parti pour fournir une façon de simuler des systèmes physiques sur des robots. Le but de ces simulateurs est de simuler le mouvement des mécanismes du robot sans modifier le code utilisateur existant. L'algorithme de base de ces simulateurs est le suivant :

- En code utilisateur normal :
  - Le PID ou des algorithmes de contrôle similaires génèrent des commandes de tension à partir des lectures du codeur (ou d'un autre capteur)
  - Les sorties du moteur sont réglées
- Dans le code périodique de simulation :
  - La simulation *état* est mise à jour en utilisant les *Entrées*, généralement des tensions de moteurs réglés à partir d'une boucle PID
  - Les lectures de l'encodeur simulé (ou d'un autre capteur) sont définies pour le code utilisateur à utiliser dans le prochain incrément de temps

22.3.1 Classes de simulation de WPILib

Les classes de simulation physique suivantes sont disponibles dans WPILib :
- LinearSystemSim, pour la modélisation de systèmes avec dynamique linéaire
- Volant d'inertie
- DifférentielDrivetrainSim
- ElevatorSim, which models gravity in the direction of elevator motion
- SingleJointedArmSim, which models gravity proportional to the arm angle
- BatterySim, qui estime simplement l'affaissement de la tension de la batterie en fonction des courants tirés

Toutes les classes de simulation (à l'exception du simulateur d'entraînement différentiel) héritent de la classe `LinearSystemSim`. Par défaut, la dynamique est la dynamique du système linéaire $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$. Les sous-classes remplacent la méthode `UpdateX(x, u, dt)` pour fournir une dynamique non linéaire personnalisée, comme la modélisation de la gravité.

Note : La prise en charge de la simulation par Swerve est en cours d'élaboration, mais nous ne pouvons pas fournir d'ETA. Pour des mises à jour sur les progrès, veuillez suivre ce pull request.

22.3.2 Utilisation dans le code utilisateur

Ce qui suit est disponible à partir du projet exemple de `elevatorsimulation` de WPILib.

In addition to standard objects such as motors and encoders, we instantiate our elevator simulator using known constants such as carriage mass and gearing reduction. We also instantiate an `EncoderSim`, which sets the distance and rate read by our `Encoder`.

In the following example, we simulate an elevator given the mass of the moving carriage (in kilograms), the radius of the drum driving the elevator (in meters), the gearing reduction between motor and drum as output over input (so usually greater than one), the minimum and maximum height of the elevator (in meters), the starting height of the elevator, and some random noise to add to our position estimate.

Note : Les simulateurs d'élévateur et de bras empêcheront la position simulée de dépasser des hauteurs ou des angles minimum ou maximum donnés. Si vous souhaitez simuler un mécanisme avec une rotation ou un mouvement infini, `LinearSystemSim` peut être une meilleure option.

Java

```
47  // Simulation classes help us simulate what's going on, including gravity.
48  private final ElevatorSim m_elevatorSim =
49      new ElevatorSim(
50          m_elevatorGearbox,
51          Constants.kElevatorGearing,
52          Constants.kCarriageMass,
53          Constants.kElevatorDrumRadius,
54          Constants.kMinElevatorHeightMeters,
55          Constants.kMaxElevatorHeightMeters,
```

(suite sur la page suivante)

(suite de la page précédente)

```
        true,
        0,
        VecBuilder.fill(0.01));
  private final EncoderSim m_encoderSim = new EncoderSim(m_encoder);
```

C++

```
  // Simulation classes help us simulate what's going on, including gravity.
  frc::sim::ElevatorSim m_elevatorSim{m_elevatorGearbox,
                                      Constants::kElevatorGearing,
                                      Constants::kCarriageMass,
                                      Constants::kElevatorDrumRadius,
                                      Constants::kMinElevatorHeight,
                                      Constants::kMaxElevatorHeight,
                                      true,
                                      0_m,
                                      {0.01}};
  frc::sim::EncoderSim m_encoderSim{m_encoder};
```

Ensuite, `teleopPeriodic`/`TeleopPeriodic` (Java/C++) utilise une simple boucle de contrôle PID pour conduire notre elevateur à un point de consigne à 30 pouces (76 cm) du sol.

Java

```
  @Override
  public void teleopPeriodic() {
    if (m_joystick.getTrigger()) {
      // Here, we set the constant setpoint of 0.75 meters.
      m_elevator.reachGoal(Constants.kSetpointMeters);
    } else {
      // Otherwise, we update the setpoint to 0.
      m_elevator.reachGoal(0.0);
    }
  }
```

```
  public void reachGoal(double goal) {
    m_controller.setGoal(goal);

    // With the setpoint value we run PID control like normal
    double pidOutput = m_controller.calculate(m_encoder.getDistance());
    double feedforwardOutput = m_feedforward.calculate(m_controller.getSetpoint().
↪velocity);
    m_motor.setVoltage(pidOutput + feedforwardOutput);
  }
```

C++

```cpp
void Robot::TeleopPeriodic() {
  if (m_joystick.GetTrigger()) {
    // Here, we set the constant setpoint of 0.75 meters.
    m_elevator.ReachGoal(Constants::kSetpoint);
  } else {
    // Otherwise, we update the setpoint to 0.
    m_elevator.ReachGoal(0.0_m);
  }
}
```

```cpp
void Elevator::ReachGoal(units::meter_t goal) {
  m_controller.SetGoal(goal);
  // With the setpoint value we run PID control like normal
  double pidOutput =
      m_controller.Calculate(units::meter_t{m_encoder.GetDistance()});
  units::volt_t feedforwardOutput =
      m_feedforward.Calculate(m_controller.GetSetpoint().velocity);
  m_motor.SetVoltage(units::volt_t{pidOutput} + feedforwardOutput);
}
```

Puis, `simulationPeriodic/SimulationPeriodic` (Java/C++) utilise la tension appliquée au moteur pour mettre à jour la position simulée de l'élévateur. Nous utilisons `SimulationPeriodic` car il s'exécute seulement que pour les robots simulés. Cela signifie que notre code de simulation ne sera pas exécuté sur un vrai robot.

Note : Classes inheriting from command-based's `Subsystem` can override the inherited `simulationPeriodic()` method. Other classes will need their simulation update methods called from `Robot`'s `simulationPeriodic`.

Enfin, la lecture de la distance du codeur simulé est réglée en utilisant la position simulée de l'élévateur, et la tension de la batterie du robot est réglée en utilisant le courant estimé tiré par l'ascenseur.

Java

```java
  public void simulationPeriodic() {
    // In this method, we update our simulation of what our elevator is doing
    // First, we set our "inputs" (voltages)
    m_elevatorSim.setInput(m_motorSim.getSpeed() * RobotController.
↪getBatteryVoltage());

    // Next, we update it. The standard loop time is 20ms.
    m_elevatorSim.update(0.020);

    // Finally, we set our simulated encoder's readings and simulated battery voltage
    m_encoderSim.setDistance(m_elevatorSim.getPositionMeters());
    // SimBattery estimates loaded battery voltages
    RoboRioSim.setVInVoltage(
        BatterySim.calculateDefaultBatteryLoadedVoltage(m_elevatorSim.
↪getCurrentDrawAmps()));
  }
```

C++

```
void Elevator::SimulationPeriodic() {
  // In this method, we update our simulation of what our elevator is doing
  // First, we set our "inputs" (voltages)
  m_elevatorSim.SetInput(frc::Vectord<1>{
      m_motorSim.GetSpeed() * frc::RobotController::GetInputVoltage()});

  // Next, we update it. The standard loop time is 20ms.
  m_elevatorSim.Update(20_ms);

  // Finally, we set our simulated encoder's readings and simulated battery
  // voltage
  m_encoderSim.SetDistance(m_elevatorSim.GetPosition().value());
  // SimBattery estimates loaded battery voltages
  frc::sim::RoboRioSim::SetVInVoltage(
      frc::sim::BatterySim::Calculate({m_elevatorSim.GetCurrentDraw()}));
}
```

22.4 Périphérique de simulation

WPILib offre un moyen de gérer les données du périphérique de simulation sous la forme de l'API SimDevice

22.4.1 Simulation des classes fondamentales de l'infrastructure WPILib

Les classes fondamentales de l'infrastructure WPILib (c.-à-d. `Encoder`, `Ultrasonic`, etc.) ont des classes de simulation nommées `EncoderSim`, `UltrasonicSim`, etc. Ces classes permettent des interactions avec les données du simulateur qui ne seraient pas possibles ou valides en dehors de la simulation. Les construire en dehors de la simulation n'interférera probablement pas avec votre code, mais appeler leurs fonctions et autres est un comportement non défini - dans le meilleur des cas, ils ne feront rien, les cas pires quant à eux pourraient planter votre code ! Placez le code de simulation fonctionnelle dans des fonctions de simulation uniquement (telles que `simulationPeriodic()`) ou enveloppez-les avec `RobotBase.isReal()`/ `RobotBase::IsReal()` checks (qui sont `constexpr` en C++).

Note : Cet exemple utilisera la classe `EncoderSim` à titre d'exemple. L'utilisation d'autres classes de simulation se fera quasiment de la même manière.

Création d'objets pour le dispositif de simulation

L'objet du dispositif de simulation peut être construit de deux façons :

— Un constructeur qui accepte l'objet matériel ordinaire.
— Un constructeur ou une méthode par défaut qui accepte le numéro de port/index/canal à qui l'appareil est connecté. Ce serait le même numéro qui a été utilisé pour construire l'objet matériel régulier. Ceci est particulièrement utile pour *unit testing*.

JAVA

```
// create a real encoder object on DIO 2,3
Encoder encoder = new Encoder(2, 3);
// create a sim controller for the encoder
EncoderSim simEncoder = new EncoderSim(encoder);
```

C++

```
// create a real encoder object on DIO 2,3
frc::Encoder encoder{2, 3};
// create a sim controller for the encoder
frc::sim::EncoderSim simEncoder{encoder};
```

Lecture et écriture des données de simulation

Chaque classe de simulation a des fonctions getter (`getXxx()`/`GetXxx()`) et setter (`setXxx(value)`/`SetXxx(value)`) pour chaque champ `Xxx`. Les fonctions getter retourneront la même chose que le getter de la classe régulière pour le robot réel.

JAVA

```
simEncoder.setCount(100);
encoder.getCount(); // 100
simEncoder.getCount(); // 100
```

C++

```
simEncoder.SetCount(100);
encoder.GetCount(); // 100
simEncoder.GetCount(); // 100
```

Enregistrement des Callbacks

En plus des getters et des setters, chaque champ dispose également d'une fonction `registerXxxCallback()` qui enregistre un callback ou rappel à exécuter à chaque fois que la valeur du champ change et renvoie un objet `CallbackStore`. Les callbacks acceptent un paramètre de type chaîne de caractère correspondant au nom du champ et un objet `HALValue` contenant la nouvelle valeur. Avant de récupérer les valeurs d'une `HALValue`, vérifiez le type de valeur contenue. Les types possibles sont `HALValue.kBoolean/HAL_BOOL`, `HALValue.kDouble/HAL_DOUBLE`, `HALValue.kEnum/HAL_ENUM`, `HALValue.kInt/HAL_INT`, `HALValue.kLong/HAL_LONG`.

En Java, appelez `close()` sur l'objet `CallbackStore` pour annuler le callback. Gardez une référence à l'objet afin qu'il ne soit pas récupéré par le garbage collector - sinon le callback sera annulé par GC. Pour fournir des données arbitraires au callback, capturez-les dans le lambda ou utilisez une référence de méthode.

En C++, enregistrez l'objet `CallbackStore` dans la bonne portée - le callback sera annulé lorsque l'objet sort de sa portée et est détruit. Les données arbitraires peuvent être transmises aux callbacks via le paramètre `param`.

Avertissement : Tenter de récupérer une valeur d'un type à partir d'une `HALValue` contenant un type différent est un comportement non défini.

JAVA

```
NotifyCallback callback = (String name, HALValue value) -> {
  if (value.getType() == HALValue.kInt) {
    System.out.println("Value of " + name + " is " + value.getInt());
  }
}
CallbackStore store = simEncoder.registerCountCallback(callback);

store.close(); // cancel the callback
```

C++

```
HAL_NotifyCallback callback = [](const char* name, void* param, const HALValue*␣
↪value) {
  if (value->type == HAL_INT) {
    wpi::outs() << "Value of " << name << " is " << value->data.v_int << '\n';
  }
};
frc::sim::CallbackStore store = simEncoder.RegisterCountCallback(callback);
// the callback will be canceled when ``store`` goes out of scope
```

22.4.2 Simulation d'autres dispositifs - La classe SimDevicesim

Note : Les fournisseurs peuvent implémenter leur connexion à l'API SimDevice d'une manière légèrement différente de celle décrite ici. Ils peuvent également fournir une classe de simulation spécifique pour leurs classes de composants. Consultez la documentation de votre fournisseur pour plus d'informations sur ce qu'il prend en charge et la manière de le faire.

La classe `SimDeviceSim` (**pas** `SimDevice` !) est un objet de simulation de périphérique général pour les périphériques qui ne sont pas des périphériques WPILib de base et qui n'ont donc pas de classes de simulation spécifiques, telles que les périphériques des fournisseurs. Ces composants apparaîtront dans l'onglet *Other Devices* du document *SimGUI*.

L'objet `SimDeviceSim` est créé à l'aide d'une clé de chaîne identique à la clé utilisée par le fournisseur pour construire le `SimDevice` sous-jacent dans sa classe du composant. Cette clé est celle avec qui le composant apparaît dans l'onglet *Other Devices*, et est généralement de la forme `Prefix:Device Name[index]`. Si la clé contient des ports/index/numéros de canal, ils peuvent être transmis sous forme d'arguments distincts au constructeur `SimDeviceSim`. La clé contient un préfixe qui est caché par défaut dans le SimGUI, il peut être affiché en sélectionnant l'option *Show prefix*. Ne pas inclure ce préfixe dans la clé transmise à `SimDeviceSim` ne se référera pas au bon composant !

JAVA

```
SimDeviceSim device = new SimDeviceSim(deviceKey, index);
```

C++

```
frc::sim::SimDeviceSim device{deviceKey, index};
```

Une fois que nous avons le `SimDeviceSim`, nous pouvons obtenir des objets `SimValue` représentant les champs du composant. Il existe également des sous-classes de type spécifique `SimDouble`, `SimInt`, `SimLong`, `SimBoolean`, et `SimEnum` qui devraient être utilisées au lieu de la classe non fiable `SimValue`. Celles-ci sont construites à partir de `SimDeviceSim` à l'aide d'une clé identique à celle utilisée par le fournisseur pour définir le champ. Cette clé est celle inscrite dans le champ apparaissant telle que dans le SimGUI. Tenter de récupérer un objet `SimValue` en dehors du simulateur ou lorsque le composant ou les clés de champ n'ont pas d'objet retournera `null` - cela peut provoquer `NullPointerException` en Java ou un comportement non défini dans C++.

JAVA

```
SimDouble field = device.getDouble(fieldKey);
field.get();
field.set(value);
```

C++

```
hal::SimDouble field = device.GetDouble(fieldKey);
field.Get();
field.Set(value);
```

22.5 Tutoriel de simulation de la base pilotable

Il s'agit d'un tutoriel pour implémenter un modèle de simulation de votre transmission différentielle à l'aide des classes de simulation. Bien que le code que nous aborderons dans ce didacticiel soit indépendant du framework, il existe deux exemples complets disponibles - un pour chaque framework.

- `StateSpaceDifferentialDriveSimulation` (Java, C++) utilise le framework basé sur les commandes
- `SimpleDifferentialDriveSimulation` (Java, C++) utilise une approche plus traditionnelle du flux de données.

Ces deux exemples sont également disponibles dans la fenêtre VS Code *New Project*.

22.5.1 Aperçu de la simulation du système de transmission

Note : Le code de ce tutoriel n'utilise pas une infrastructure de développement spécifique (c.-à-d. orientée commande vs un simple flux de données dans un simple fichier) ; toutefois, des directives seront fournies dans certains domaines sur la meilleure façon d'implémenter certains éléments de code dans des types spécifiques d'infrastructure de dévoleppement.

L'objectif de ce tutoriel est de fournir des conseils sur l'implémentation des capacités de simulation pour un robot pourvu d'une transmission différentielle. À la fin de ce tutoriel, vous devriez être en mesure de :

1. Comprendre les concepts fondamentaux qui sous-tendent le cadre de simulation WPILib.
2. Créer un modèle de simulation du système de transmission à l'aide des paramètres physiques de votre robot.
3. Utiliser le modèle de simulation pour prédire comment votre vrai robot se déplacera compte tenu des tensions d'entrée spécifiques.
4. Réglez les constantes de rétroaction et corrigez les bogues courants (par exemple, l'inversion du moteur) avant d'avoir accès au matériel physique (Hardware).
5. Utilisez l'interface graphique de simulation pour visualiser le mouvement du robot sur un champ virtuel.

Pourquoi simuler la transmission d'une base pilotable ?

La transmission de la base pilotable d'un robot est l'un des mécanismes les plus importants - par conséquent, il est important de s'assurer que le programme logiciel qui contrôle votre transmission est aussi robuste que possible. En étant capable de simuler la réaction d'une vraie transmission physique, vous pouvez prendre une longueur d'avance sur l'écriture de programmes de qualité avant d'avoir accès au matériel physique. Avec le cadre de simulation, vous pouvez vérifier non seulement les fonctionnalités de base, comme vous assurer que les inversions sur les moteurs et les encodeurs sont correctes, mais également des capacités avancées telles que la vérification de la précision du suivi de trajectoire.

22.5.2 Étape 1 : Création d'instances simulées du matériel (Hardware)

Le framework de simulation WPILib contient plusieurs classes `XXXSim` , où `XXX` représente du matériel physique tel que des encodeurs ou des gyroscopes. Ces classes de simulation peuvent être utilisées pour définir des positions et des vitesses (pour les encodeurs) et des angles (pour les gyroscopes) à partir d'un modèle de votre base pilotable. Voir *the Article sur la simulation de composant*, pour plus d'informations sur ces classes de matériel de simulation et la simulation des périphériques des fournisseur.

Note : Simulation objects associated with a particular subsystem should live in that subsystem. An example of this is in the `StateSpaceDriveSimulation` (Java, C++) example.

Simulation d'encodeurs

La classe `EncoderSim` permet aux utilisateurs de définir les positions et les vitesses de l'encodeur sur un objet `Encoder` donné. Lorsqu'elle fonctionne sur du matériel réel, la classe `Encoder` interagit avec de vrais capteurs pour compter les révolutions (et les convertir automatiquement en unités de distance si elle est configurée pour le faire) ; cependant, en simulation, il n'y a pas de telles mesures à faire. La classe `EncoderSim` peut accepter ces lectures simulées à partir d'un modèle de votre transmission.

Note : Il n'est pas possible de simuler des encodeurs directement connectés à des contrôleurs de moteur CAN à l'aide des classes WPILib. Pour plus d'informations sur votre contrôleur de

moteur spécifique, veuillez lire la documentation de votre fournisseur.

JAVA

```
// These represent our regular encoder objects, which we would
// create to use on a real robot.
private Encoder m_leftEncoder = new Encoder(0, 1);
private Encoder m_rightEncoder = new Encoder(2, 3);

// These are our EncoderSim objects, which we will only use in
// simulation. However, you do not need to comment out these
// declarations when you are deploying code to the roboRIO.
private EncoderSim m_leftEncoderSim = new EncoderSim(m_leftEncoder);
private EncoderSim m_rightEncoderSim = new EncoderSim(m_rightEncoder);
```

C++

```
#include <frc/Encoder.h>
#include <frc/simulation/EncoderSim.h>

...

// These represent our regular encoder objects, which we would
// create to use on a real robot.
frc::Encoder m_leftEncoder{0, 1};
frc::Encoder m_rightEncoder{2, 3};

// These are our EncoderSim objects, which we will only use in
// simulation. However, you do not need to comment out these
// declarations when you are deploying code to the roboRIO.
frc::sim::EncoderSim m_leftEncoderSim{m_leftEncoder};
frc::sim::EncoderSim m_rightEncoderSim{m_rightEncoder};
```

Simulation de gyroscopes

Semblable à la classe `EncoderSim`, des classes de gyroscopes simulés existent également pour les gyroscopes WPILib couramment utilisés - `AnalogGyroSim` et `ADXRS450_GyroSim`. Ceux-ci sont également construits de la même manière.

Note : It is not possible to simulate certain vendor gyros (i.e. Pigeon *IMU* and NavX) using WPILib classes. Please read the respective vendors" documentation for information on their simulation support.

JAVA

```
// Create our gyro object like we would on a real robot.
private AnalogGyro m_gyro = new AnalogGyro(1);

// Create the simulated gyro object, used for setting the gyro
// angle. Like EncoderSim, this does not need to be commented out
// when deploying code to the roboRIO.
private AnalogGyroSim m_gyroSim = new AnalogGyroSim(m_gyro);
```

C++

```
#include <frc/AnalogGyro.h>
#include <frc/simulation/AnalogGyroSim.h>

...

// Create our gyro objectl ike we would on a real robot.
frc::AnalogGyro m_gyro{1};

// Create the simulated gyro object, used for setting the gyro
// angle. Like EncoderSim, this does not need to be commented out
// when deploying code to the roboRIO.
frc::sim::AnalogGyroSim m_gyroSim{m_gyro};
```

22.5.3 Étape 2 : Création d'un modèle de transmission pour une base pilotable

Afin de déterminer avec précision comment votre transmission physique répondra à des entrées de tension moteur données, un modèle précis de votre transmission doit être créé. Ce modèle est généralement créé en mesurant divers paramètres physiques de votre vrai robot. Dans WPILib, ce modèle de simulation de transmission est représenté par la classe `DifferentialDrivetrainSim`.

Création d'un `DifferentialDrivetrainSim` à partir de mesures physiques

One way to creating a `DifferentialDrivetrainSim` instance is by using physical measurements of the drivetrain and robot – either obtained through *CAD* software or real-world measurements (the latter will usually yield better results as it will more closely match reality). This constructor takes the following parameters :

- Le type et le nombre de moteurs d'un côté de la transmission.
- The gear ratio between the motors and the wheels as output *torque* over input *torque* (this number is usually greater than 1 for drivetrains).
- The moment of inertia of the drivetrain (this can be obtained from a *CAD* model of your drivetrain. Usually, this is between 3 and 8 kgm^2).
- La masse de la transmission (il est recommandé d'utiliser la masse de l'ensemble du robot lui-même, car elle modélisera plus précisément les caractéristiques d'accélération de votre robot pour le suivi de trajectoire).
- Le rayon des roues motrices.
- La largeur de voie (distance entre les roues gauche et droite).

— Les écarts types du bruit de mesure : cela représente le bruit de mesure que vous attendez de vos capteurs réels. Le bruit de mesure est un tableau avec 7 éléments, chaque élément représentant l'écart type du bruit de mesure en x, y, cap, vitesse gauche, vitesse droite, position gauche et position droite respectivement. Cette option peut être omise en C ++ ou définie sur `null` en Java si le bruit de mesure n'est pas souhaitable.

Vous pouvez calculer le bruit de mesure de vos capteurs en prenant plusieurs points de données de l'état que vous essayez de mesurer et en calculant l'écart type à l'aide d'un outil comme Python. Par exemple, pour calculer l'écart type dans l'estimation de la vitesse de vos encodeurs, vous pouvez déplacer votre robot à une vitesse constante, prendre plusieurs mesures et calculer leur écart type par rapport à la moyenne connue. Si ce processus est trop fastidieux, les valeurs utilisées dans l'exemple ci-dessous doivent être une bonne représentation du bruit moyen des codeurs.

Note : L'écart type du bruit pour une mesure a les mêmes unités que cette mesure. Par exemple, l'écart type du bruit de vitesse a des unités de m

---

Note : Il est très important d'utiliser des unités SI (c'est-à-dire des mètres et des radians) lors du passage de paramètres en Java. En C ++, la *librairie d'unités* peut être utilisée pour spécifier n'importe quel type d'unité.

JAVA

```
// Create the simulation model of our drivetrain.
DifferentialDrivetrainSim m_driveSim = new DifferentialDrivetrainSim(
  DCMotor.getNEO(2),       // 2 NEO motors on each side of the drivetrain.
  7.29,                    // 7.29:1 gearing reduction.
  7.5,                     // MOI of 7.5 kg m^2 (from CAD model).
  60.0,                    // The mass of the robot is 60 kg.
  Units.inchesToMeters(3), // The robot uses 3" radius wheels.
  0.7112,                  // The track width is 0.7112 meters.

  // The standard deviations for measurement noise:
  // x and y:          0.001 m
  // heading:          0.001 rad
  // l and r velocity: 0.1   m/s
  // l and r position: 0.005 m
  VecBuilder.fill(0.001, 0.001, 0.001, 0.1, 0.1, 0.005, 0.005));
```

C++

```
#include <frc/simulation/DifferentialDrivetrainSim.h>

...

// Create the simulation model of our drivetrain.
frc::sim::DifferentialDrivetrainSim m_driveSim{
  frc::DCMotor::GetNEO(2), // 2 NEO motors on each side of the drivetrain.
  7.29,                // 7.29:1 gearing reduction.
  7.5_kg_sq_m,         // MOI of 7.5 kg m^2 (from CAD model).
```

(suite sur la page suivante)

(suite de la page précédente)

```
  60_kg,              // The mass of the robot is 60 kg.
  3_in,               // The robot uses 3" radius wheels.
  0.7112_m,           // The track width is 0.7112 meters.

  // The standard deviations for measurement noise:
  // x and y:          0.001 m
  // heading:          0.001 rad
  // l and r velocity: 0.1   m/s
  // l and r position: 0.005 m
  {0.001, 0.001, 0.001, 0.1, 0.1, 0.005, 0.005}};
```

Création d'un `DifferentialDrivetrainSim` à partir de SysId Gains

You can also use the gains produced by *System Identification*, which you may have performed as part of setting up the trajectory tracking workflow outlined *here* to create a simulation model of your drivetrain and often yield results closer to real-world behavior than the method above.

Important : Vous devez avoir besoin de deux ensembles de gains Kv et Ka de l'outil d'identification - l'un provenant du mouvement en ligne droite et l'autre de la rotation sur place. Nous appellerons respectivement ces deux ensembles de gains gains linéaires et gains angulaires.

Ce constructeur prend les paramètres suivants :

- Un système linéaire représentant la transmission - cela peut être créé en utilisant les gains d'identification.
- La largeur de voie (distance entre les roues gauche et droite).
- Le type et le nombre de moteurs d'un côté de la transmission.
- The gear ratio between the motors and the wheels as output *torque* over input *torque* (this number is usually greater than 1 for drivetrains).
- Le rayon des roues motrices.
- Les écarts types du bruit de mesure : cela représente le bruit de mesure que vous attendez de vos capteurs réels. Le bruit de mesure est un tableau avec 7 éléments, chaque élément représentant l'écart type du bruit de mesure en x, y, cap, vitesse gauche, vitesse droite, position gauche et position droite respectivement. Cette option peut être omise en C ++ ou définie sur `null` en Java si le bruit de mesure n'est pas souhaitable.

Vous pouvez calculer le bruit de mesure de vos capteurs en prenant plusieurs points de données de l'état que vous essayez de mesurer et en calculant l'écart type à l'aide d'un outil comme Python. Par exemple, pour calculer l'écart type dans l'estimation de la vitesse de vos encodeurs, vous pouvez déplacer votre robot à une vitesse constante, prendre plusieurs mesures et calculer leur écart type par rapport à la moyenne connue. Si ce processus est trop fastidieux, les valeurs utilisées dans l'exemple ci-dessous doivent être une bonne représentation du bruit moyen des codeurs.

Note : L'écart type du bruit pour une mesure a les mêmes unités que cette mesure. Par exemple, l'écart type du bruit de vitesse a des unités de m

---

Note : Il est très important d'utiliser des unités SI (c'est-à-dire des mètres et des radians) lors du passage de paramètres en Java. En C ++, la *librairie d'unités* peut être utilisée pour

spécifier n'importe quel type d'unité.

JAVA

```
// Create our feedforward gain constants (from the identification
// tool)
static final double KvLinear = 1.98;
static final double KaLinear = 0.2;
static final double KvAngular = 1.5;
static final double KaAngular = 0.3;

// Create the simulation model of our drivetrain.
private DifferentialDrivetrainSim m_driveSim = new DifferentialDrivetrainSim(
  // Create a linear system from our identification gains.
  LinearSystemId.identifyDrivetrainSystem(KvLinear, KaLinear, KvAngular, KaAngular),
  DCMotor.getNEO(2),       // 2 NEO motors on each side of the drivetrain.
  7.29,                    // 7.29:1 gearing reduction.
  0.7112,                  // The track width is 0.7112 meters.
  Units.inchesToMeters(3), // The robot uses 3" radius wheels.

  // The standard deviations for measurement noise:
  // x and y:          0.001 m
  // heading:          0.001 rad
  // l and r velocity: 0.1   m/s
  // l and r position: 0.005 m
  VecBuilder.fill(0.001, 0.001, 0.001, 0.1, 0.1, 0.005, 0.005));
```

C++

```
#include <frc/simulation/DifferentialDrivetrainSim.h>
#include <frc/system/plant/LinearSystemId.h>
#include <units/acceleration.h>
#include <units/angular_acceleration.h>
#include <units/angular_velocity.h>
#include <units/voltage.h>
#include <units/velocity.h>

...

// Create our feedforward gain constants (from the identification
// tool). Note that these need to have correct units.
static constexpr auto KvLinear = 1.98_V / 1_mps;
static constexpr auto KaLinear = 0.2_V / 1_mps_sq;
static constexpr auto KvAngular = 1.5_V / 1_rad_per_s;
static constexpr auto KaAngular = 0.3_V / 1_rad_per_s_sq;
// The track width is 0.7112 meters.
static constexpr auto kTrackwidth = 0.7112_m;

// Create the simulation model of our drivetrain.
frc::sim::DifferentialDrivetrainSim m_driveSim{
  // Create a linear system from our identification gains.
  frc::LinearSystemId::IdentifyDrivetrainSystem(
    KvLinear, KaLinear, KvAngular, KaAngular, kTrackWidth),
```

(suite sur la page suivante)

(suite de la page précédente)

```
  kTrackWidth,
  frc::DCMotor::GetNEO(2), // 2 NEO motors on each side of the drivetrain.
  7.29,                // 7.29:1 gearing reduction.
  3_in,                // The robot uses 3" radius wheels.

  // The standard deviations for measurement noise:
  // x and y:          0.001 m
  // heading:          0.001 rad
  // l and r velocity: 0.1   m/s
  // l and r position: 0.005 m
  {0.001, 0.001, 0.001, 0.1, 0.1, 0.005, 0.005}};
```

Création d'un `DifferentialDrivetrainSim` du châssis KoP

La classe `DifferentialDrivetrainSim` renferme également une méthode statique `createKitbotSim()` (Java) / `CreateKitbotSim()` (C++) qui peut créer une instance du `DifferentialDrivetrainSim` en utilisant les paramètres standard du châssis du kit de pièces. Cette méthode accepte 5 arguments, dont deux sont facultatifs :

- Le type et le nombre de moteurs d'un côté de la transmission.
- The gear ratio between the motors and the wheels as output *torque* over input *torque* (this number is usually greater than 1 for drivetrains).
- Le diamètre des roues installées sur la base pilotable.
- Le moment d'inertie de la base d'entraînement (facultatif).
- Les écarts types du bruit de mesure : cela représente le bruit de mesure que vous attendez de vos capteurs réels. Le bruit de mesure est un tableau avec 7 éléments, chaque élément représentant l'écart type du bruit de mesure en x, y, cap, vitesse gauche, vitesse droite, position gauche et position droite respectivement. Cette option peut être omise en C ++ ou définie sur `null` en Java si le bruit de mesure n'est pas souhaitable.

Vous pouvez calculer le bruit de mesure de vos capteurs en prenant plusieurs points de données de l'état que vous essayez de mesurer et en calculant l'écart type à l'aide d'un outil comme Python. Par exemple, pour calculer l'écart type dans l'estimation de la vitesse de vos encodeurs, vous pouvez déplacer votre robot à une vitesse constante, prendre plusieurs mesures et calculer leur écart type par rapport à la moyenne connue. Si ce processus est trop fastidieux, les valeurs utilisées dans l'exemple ci-dessous doivent être une bonne représentation du bruit moyen des codeurs.

Note : L'écart type du bruit pour une mesure a les mêmes unités que cette mesure. Par exemple, l'écart type du bruit de vitesse a des unités de m

---

Note : Il est très important d'utiliser des unités SI (c'est-à-dire des mètres et des radians) lors du passage de paramètres en Java. En C ++, la *librairie d'unités* peut être utilisée pour spécifier n'importe quel type d'unité.

JAVA

```
private DifferentialDrivetrainSim m_driveSim = DifferentialDrivetrainSim.
↪createKitbotSim(
  KitbotMotor.kDualCIMPerSide, // 2 CIMs per side.
  KitbotGearing.k10p71,        // 10.71:1
  KitbotWheelSize.kSixInch,    // 6" diameter wheels.
  null                         // No measurement noise.
);
```

C++

```
#include <frc/simulation/DifferentialDrivetrainSim.h>

...

frc::sim::DifferentialDrivetrainSim m_driveSim =
  frc::sim::DifferentialDrivetrainSim::CreateKitbotSim(
    frc::sim::DifferentialDrivetrainSim::KitbotMotor::DualCIMPerSide, // 2 CIMs per␣
↪side.
    frc::sim::DifferentialDrivetrainSim::KitbotGearing::k10p71,       // 10.71:1
    frc::sim::DifferentialDrivetrainSim::KitbotWheelSize::kSixInch    // 6" diameter␣
↪wheels.
);
```

Note : Vous pouvez utiliser l'enum (Java) / struct (C++) `KitbotMotor`, `KitbotGearing`, et `KitbotWheelSize` pour obtenir des configurations couramment utilisées du châssis fourni dans le kit de pièces.

Important : Construire votre instance `DifferentialDrivetrainSim` de cette façon n'est qu'une approximation et vise à permettre aux équipes de mettre en place et d'exécuter des simulations. L'utilisation de valeurs empiriques mesurées à partir de votre robot physique donnera toujours des résultats plus précis.

22.5.4 Étape 3 : Mise à jour du modèle de transmission

Maintenant que le modèle de transmission a été créé, il doit être mis à jour périodiquement avec les dernières commandes de tension du moteur. Il est recommandé de faire cette étape dans une méthode séparée `simulationPeriodic()` / `SimulationPeriodic()` dans votre sous-système et d'appeler cette méthode uniquement en simulation.

Note : If you are using the command-based framework, every subsystem that extends `SubsystemBase` has a `simulationPeriodic()` / `SimulationPeriodic()` which can be overridden. This method is automatically run only during simulation. If you are not using the command-based library, make sure you call your simulation method inside the overridden `simulationPeriodic()` / `SimulationPeriodic()` of the main `Robot` class. These periodic methods are also automatically called only during simulation.

La mise à jour du modèle comporte trois étapes principales :

1. Définissez l' *entrée* du modèle de transmission. Ce sont les tensions du moteur des deux côtés de la transmission.
2. Avancez le modèle dans le temps du pas de temps périodique nominal (généralement 20 ms). Cela met à jour tous les états de la transmission (c'est-à-dire la pose, les positions de l'encodeur et les vitesses) comme si 20 ms s'étaient écoulés.
3. Mettez à jour les capteurs simulés avec de nouvelles positions, vitesses et angles.

JAVA

```
private PWMSparkMax m_leftMotor = new PWMSparkMax(0);
private PWMSparkMax m_rightMotor = new PWMSparkMax(1);

public Drivetrain() {
  ...
  m_leftEncoder.setDistancePerPulse(2 * Math.PI * kWheelRadius / kEncoderResolution);
  m_rightEncoder.setDistancePerPulse(2 * Math.PI * kWheelRadius / kEncoderResolution);
}

public void simulationPeriodic() {
  // Set the inputs to the system. Note that we need to convert
  // the [-1, 1] PWM signal to voltage by multiplying it by the
  // robot controller voltage.
  m_driveSim.setInputs(m_leftMotor.get() * RobotController.getInputVoltage(),
                       m_rightMotor.get() * RobotController.getInputVoltage());

  // Advance the model by 20 ms. Note that if you are running this
  // subsystem in a separate thread or have changed the nominal timestep
  // of TimedRobot, this value needs to match it.
  m_driveSim.update(0.02);

  // Update all of our sensors.
  m_leftEncoderSim.setDistance(m_driveSim.getLeftPositionMeters());
  m_leftEncoderSim.setRate(m_driveSim.getLeftVelocityMetersPerSecond());
  m_rightEncoderSim.setDistance(m_driveSim.getRightPositionMeters());
  m_rightEncoderSim.setRate(m_driveSim.getRightVelocityMetersPerSecond());
  m_gyroSim.setAngle(-m_driveSim.getHeading().getDegrees());
}
```

C++

```
frc::PWMSparkMax m_leftMotor{0};
frc::PWMSparkMax m_rightMotor{1};

Drivetrain() {
  ...
  m_leftEncoder.SetDistancePerPulse(2 * std::numbers::pi * kWheelRadius /␣
↪kEncoderResolution);
  m_rightEncoder.SetDistancePerPulse(2 * std::numbers::pi * kWheelRadius /␣
↪kEncoderResolution);
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
void SimulationPeriodic() {
  // Set the inputs to the system. Note that we need to convert
  // the [-1, 1] PWM signal to voltage by multiplying it by the
  // robot controller voltage.
  m_driveSim.SetInputs(
    m_leftMotor.get() * units::volt_t(frc::RobotController::GetInputVoltage()),
    m_rightMotor.get() * units::volt_t(frc::RobotController::GetInputVoltage()));

  // Advance the model by 20 ms. Note that if you are running this
  // subsystem in a separate thread or have changed the nominal timestep
  // of TimedRobot, this value needs to match it.
  m_driveSim.Update(20_ms);

  // Update all of our sensors.
  m_leftEncoderSim.SetDistance(m_driveSim.GetLeftPosition().value());
  m_leftEncoderSim.SetRate(m_driveSim.GetLeftVelocity().value());
  m_rightEncoderSim.SetDistance(m_driveSim.GetRightPosition().value());
  m_rightEncoderSim.SetRate(m_driveSim.GetRightVelocity().value());
  m_gyroSim.SetAngle(-m_driveSim.GetHeading().Degrees());
}
```

Important : Si le côté droit de votre transmission est inversé, vous DEVEZ inverser le signe de la tension dans l'appel `SetInputs()` pour vous assurer que les tensions positives correspondent au mouvement vers l'avant.

Important : Étant donné que le modèle de simulateur de transmission renvoie respectivement les positions et les vitesses en mètres et en m/s, celles-ci doivent être converties en ticks d'encodeur lors de l'appel de `SetDistance()` et `SetRate()`. Alternativement, vous pouvez configurer `SetDistancePerPulse` sur les encodeurs pour que l'objet `Encoder` s'en charge automatiquement - c'est l'approche qui est adoptée dans l'exemple ci-dessus.

Maintenant que les positions, les vitesses et les angles du gyroscope simulés ont été définis, vous pouvez appeler `m_leftEncoder.GetDistance()`, etc. dans le code de votre robot comme d'habitude et il se comportera exactement comme il le ferait sur un vrai robot. Cela implique d'effectuer des calculs d'odométrie, des boucles de rétroaction PID de vitesse pour le suivi de trajectoire, etc.

22.5.5 Étape 4 : Mise à jour de l'odométrie et visualisation de la position du robot

Maintenant que les positions, les vitesses et les angles du gyroscope simulés sont régulièrement mis à jour avec des informations précises, ces données peuvent être utilisées pour mettre à jour la pose du robot dans une boucle périodique (telle que la méthode `periodic()` dans un `Subsystem`). En simulation, la boucle périodique utilisera des lectures simulées d'encodeur et de gyroscope pour mettre à jour l'odométrie alors que sur le robot réel, le même code utilisera des lectures réelles provenant du matériel physique.

Note : Pour plus d'informations sur l'utilisation de l'odométrie, voir *ce document*.

Visualisation de la pose du robot

La pose du robot peut être visualisée sur l'interface graphique du simulateur (pendant la simulation) ou sur un tableau de bord tel que Glass (sur un vrai robot) en envoyant la pose d'odométrie sur un objet `Field2d` . Un `Field2d` peut être construit de manière triviale sans aucun argument de constructeur :

JAVA

```
private Field2d m_field = new Field2d();
```

C++

```
#include <frc/smartdashboard/Field2d.h>

..

frc::Field2d m_field;
```

Cette instance `Field2d` doit ensuite être envoyée via les NetworkTables. Le meilleur endroit pour faire cela est dans le constructeur de votre sous-système.

JAVA

```
public Drivetrain() {
  ...
  SmartDashboard.putData("Field", m_field);
}
```

C++

```
#include <frc/smartdashboard/SmartDashboard.h>

Drivetrain() {
  ...
  frc::SmartDashboard::PutData("Field", &m_field);
}
```

Note : L'instance `Field2d` peut également être envoyée en utilisant une API NetworkTables de niveau inférieur ou en utilisant *l'API Shuffleboard*.

Enfin, la pose de votre odométrie doit être mise à jour périodiquement dans l'objet `Field2d`. Rappelez-vous que cela devrait être une méthode générale `periodic()`, c'est-à-dire une méthode qui s'exécute à la fois pendant la simulation et pendant le fonctionnement réel du robot.

JAVA

```
public void periodic() {
  ...
  // This will get the simulated sensor readings that we set
  // in the previous article while in simulation, but will use
  // real values on the robot itself.
  m_odometry.update(m_gyro.getRotation2d(),
                    m_leftEncoder.getDistance(),
                    m_rightEncoder.getDistance());
  m_field.setRobotPose(m_odometry.getPoseMeters());
}
```

C++

```
void Periodic() {
  ...
  // This will get the simulated sensor readings that we set
  // in the previous article while in simulation, but will use
  // real values on the robot itself.
  m_odometry.Update(m_gyro.GetRotation2d(),
                    units::meter_t(m_leftEncoder.GetDistance()),
                    units::meter_t(m_rightEncoder.GetDistance()));
  m_field.SetRobotPose(m_odometry.GetPose());
}
```

Important : Il est important que ce code soit placé dans une méthode régulière `periodic()` - une méthode qui est appelée périodiquement quel que soit le mode de fonctionnement. Si vous utilisez la librairie basée sur les commandes, cette méthode existe déjà. Sinon, vous êtes responsable d'appeler cette méthode périodiquement à partir de la classe principale `Robot`.

Note : At this point we have covered all of the code changes required to run your code. You should head to the *Simulation User Interface page* for more info on how to run the simulation and the *Field2d Widget page* to add the field that your simulated robot will run on to the GUI.

22.6 Test unitaire

Unit testing is a method of testing code by dividing the code into the smallest « units » possible and testing each unit. In robot code, this can mean testing the code for each subsystem individually. There are many unit testing frameworks for most languages. Java robot projects have JUnit 5 available by default, and C++ robot projects have Google Test.

22.6.1 Écriture de code testable

Note : This example can be easily adapted to the command-based paradigm by having `Intake` inherit from `SubsystemBase`.

Notre sous-système sera un mécanisme Intake du défin Infinite Recharge constitué d'un piston et d'un moteur : le piston déploie/rétracte le Intake, et le moteur tirera les cellules de puissance à l'intérieur. Nous ne voulons pas que le moteur fonctionne si le mécanisme Intake n'est pas déployé parce qu'il n'aura aucun effet.

Pour fournir un « clean slate » pour chaque test, nous avons besoin d'avoir une fonction pour détruire l'objet et libérer toutes les allocations matérielles. En Java, cela se fait en implémentant l'interface `AutoCloseable` et sa méthode `.close()`, détruisant chaque objet membre en appelant la méthode membre `.close()` - un objet sans méthode `.close()` n'a probablement pas besoin d'être fermé. En C++, le destructeur par défaut sera appelé automatiquement lorsque l'objet sort de sa portée et appellera à son tour les destructeurs des objets membres.

Note : Les fournisseurs tiers peuvent ne pas prendre en charge la fermeture des ressources de la même manière que celle indiquée ici. Consultez la documentation de votre fournisseur pour plus d'informations sur ce qu'il prend en charge et comment.

Java

```
import edu.wpi.first.wpilibj.DoubleSolenoid;
import edu.wpi.first.wpilibj.PneumaticsModuleType;
import edu.wpi.first.wpilibj.examples.unittest.Constants.IntakeConstants;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;

public class Intake implements AutoCloseable {
  private final PWMSparkMax m_motor;
  private final DoubleSolenoid m_piston;

  public Intake() {
    m_motor = new PWMSparkMax(IntakeConstants.kMotorPort);
    m_piston =
        new DoubleSolenoid(
            PneumaticsModuleType.CTREPCM,
            IntakeConstants.kPistonFwdChannel,
            IntakeConstants.kPistonRevChannel);
  }

  public void deploy() {
    m_piston.set(DoubleSolenoid.Value.kForward);
  }

  public void retract() {
    m_piston.set(DoubleSolenoid.Value.kReverse);
    m_motor.set(0); // turn off the motor
  }

  public void activate(double speed) {
```

(suite sur la page suivante)

(suite de la page précédente)

```
    if (isDeployed()) {
      m_motor.set(speed);
    } else { // if piston isn't open, do nothing
      m_motor.set(0);
    }
  }

  public boolean isDeployed() {
    return m_piston.get() == DoubleSolenoid.Value.kForward;
  }

  @Override
  public void close() {
    m_piston.close();
    m_motor.close();
  }
}
```

C++ (Header ou en-tête)

```
#include <frc/DoubleSolenoid.h>
#include <frc/motorcontrol/PWMSparkMax.h>

#include "Constants.h"

class Intake {
 public:
  void Deploy();
  void Retract();
  void Activate(double speed);
  bool IsDeployed() const;

 private:
  frc::PWMSparkMax m_motor{IntakeConstants::kMotorPort};
  frc::DoubleSolenoid m_piston{frc::PneumaticsModuleType::CTREPCM,
                               IntakeConstants::kPistonFwdChannel,
                               IntakeConstants::kPistonRevChannel};
};
```

C++ (Source)

```
#include "subsystems/Intake.h"

void Intake::Deploy() {
  m_piston.Set(frc::DoubleSolenoid::Value::kForward);
}

void Intake::Retract() {
  m_piston.Set(frc::DoubleSolenoid::Value::kReverse);
  m_motor.Set(0);  // turn off the motor
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
void Intake::Activate(double speed) {
  if (IsDeployed()) {
    m_motor.Set(speed);
  } else {  // if piston isn't open, do nothing
    m_motor.Set(0);
  }
}

bool Intake::IsDeployed() const {
  return m_piston.Get() == frc::DoubleSolenoid::Value::kForward;
}
```

22.6.2 Écriture des Tests

Important : Les tests sont placés à l'intérieur de l'ensemble source `test` : `/src/test/java/` et `/src/test/cpp/` pour les tests Java et C++, respectivement. Les fichiers en dehors de cette racine source n'ont pas accès au cadre de test - cela fera échouer la compilation en raison de références non résolues.

In Java, each test class contains at least one test method marked with `@org.junit.jupiter.api.Test`, each method representing a test case. Additional methods for opening resources (such as our `Intake` object) before each test and closing them after are respectively marked with `@org.junit.jupiter.api.BeforeEach` and `@org.junit.jupiter.api.AfterEach`. In C++, test fixture classes inheriting from `testing::Test` contain our subsystem and simulation hardware objects, and test methods are written using the `TEST_F(testfixture, testname)` macro. The `SetUp()` and `TearDown()` methods can be overridden in the test fixture class and will be run respectively before and after each test.

Chaque méthode de test doit contenir au moins une *assertion* (`assert*()` en Java ou `EXPECT_*()` en C++). Ces assertions vérifient une condition à moment de l'exécution et échouent au test si la condition n'est pas remplie. S'il y a plus d'une assertion dans une méthode de test, la première assertion échouée plantera le test - l'exécution n'atteindra pas les assertions subséquentes.

Both JUnit and GoogleTest have multiple assertion types; the most common is equality : `assertEquals(expected, actual)`/`EXPECT_EQ(expected, actual)`. When comparing numbers, a third parameter - `delta`, the acceptable error, can be given. In JUnit (Java), these assertions are static methods and can be used without qualification by adding the static star import `import static org.junit.jupiter.api.Assertions.*`. In Google Test (C++), assertions are macros from the `<gtest/gtest.h>` header.

Note : La comparaison des valeurs en point-flottants n'est pas exacte, de sorte que leur comparaison doit être faite avec un paramètre d'erreur acceptable (`DELTA`).

Java

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import edu.wpi.first.hal.HAL;
import edu.wpi.first.wpilibj.DoubleSolenoid;
import edu.wpi.first.wpilibj.PneumaticsModuleType;
import edu.wpi.first.wpilibj.examples.unittest.Constants.IntakeConstants;
import edu.wpi.first.wpilibj.simulation.DoubleSolenoidSim;
import edu.wpi.first.wpilibj.simulation.PWMSim;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class IntakeTest {
  static final double DELTA = 1e-2; // acceptable deviation range
  Intake m_intake;
  PWMSim m_simMotor;
  DoubleSolenoidSim m_simPiston;

  @BeforeEach // this method will run before each test
  void setup() {
    assert HAL.initialize(500, 0); // initialize the HAL, crash if failed
    m_intake = new Intake(); // create our intake
    m_simMotor =
        new PWMSim(IntakeConstants.kMotorPort); // create our simulation PWM motor␣
↪controller
    m_simPiston =
        new DoubleSolenoidSim(
            PneumaticsModuleType.CTREPCM,
            IntakeConstants.kPistonFwdChannel,
            IntakeConstants.kPistonRevChannel); // create our simulation solenoid
  }

  @SuppressWarnings("PMD.SignatureDeclareThrowsException")
  @AfterEach // this method will run after each test
  void shutdown() throws Exception {
    m_intake.close(); // destroy our intake object
  }

  @Test // marks this method as a test
  void doesntWorkWhenClosed() {
    m_intake.retract(); // close the intake
    m_intake.activate(0.5); // try to activate the motor
    assertEquals(
        0.0, m_simMotor.getSpeed(), DELTA); // make sure that the value set to the␣
↪motor is 0
  }

  @Test
  void worksWhenOpen() {
    m_intake.deploy();
    m_intake.activate(0.5);
    assertEquals(0.5, m_simMotor.getSpeed(), DELTA);
  }

  @Test
```

(suite sur la page suivante)

(suite de la page précédente)

```
  void retractTest() {
    m_intake.retract();
    assertEquals(DoubleSolenoid.Value.kReverse, m_simPiston.get());
  }

  @Test
  void deployTest() {
    m_intake.deploy();
    assertEquals(DoubleSolenoid.Value.kForward, m_simPiston.get());
  }
}
```

C++

```
#include <frc/DoubleSolenoid.h>
#include <frc/simulation/DoubleSolenoidSim.h>
#include <frc/simulation/PWMSim.h>
#include <gtest/gtest.h>

#include "Constants.h"
#include "subsystems/Intake.h"

class IntakeTest : public testing::Test {
 protected:
  Intake intake;  // create our intake
  frc::sim::PWMSim simMotor{
      IntakeConstants::kMotorPort};  // create our simulation PWM
  frc::sim::DoubleSolenoidSim simPiston{
      frc::PneumaticsModuleType::CTREPCM, IntakeConstants::kPistonFwdChannel,
      IntakeConstants::kPistonRevChannel};  // create our simulation solenoid
};

TEST_F(IntakeTest, DoesntWorkWhenClosed) {
  intake.Retract();      // close the intake
  intake.Activate(0.5);  // try to activate the motor
  EXPECT_DOUBLE_EQ(
      0.0,
      simMotor.GetSpeed());  // make sure that the value set to the motor is 0
}

TEST_F(IntakeTest, WorksWhenOpen) {
  intake.Deploy();
  intake.Activate(0.5);
  EXPECT_DOUBLE_EQ(0.5, simMotor.GetSpeed());
}

TEST_F(IntakeTest, Retract) {
  intake.Retract();
  EXPECT_EQ(frc::DoubleSolenoid::Value::kReverse, simPiston.Get());
}

TEST_F(IntakeTest, Deploy) {
  intake.Deploy();
  EXPECT_EQ(frc::DoubleSolenoid::Value::kForward, simPiston.Get());
}
```

Pour une utilisation plus avancée de JUnit et Google Test, consultez les documents cadres.

22.6.3 Exécution des tests

Note : Les tests seront toujours exécutés en simulation sur votre bureau. Pour les conditions préalables et plus d'informations, consultez *Introduction à la simulation*.

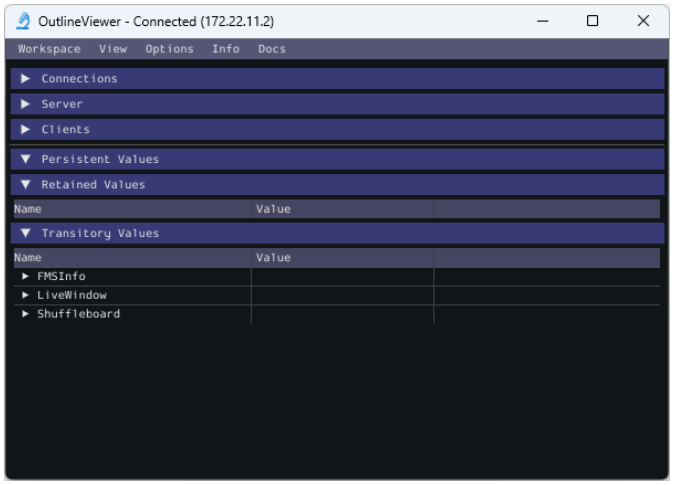
Pour que les tests Java s'exécutent, assurez-vous que votre fichier `build.gradle` contient le bloc suivant :

```
test {
    useJUnitPlatform()
    systemProperty 'junit.jupiter.extensions.autodetection.enabled', 'true'
}
```

Use *Test Robot Code* from the Command Palette to run the tests. Results will be reported in the terminal output, each test will have a `FAILED` or `PASSED/OK` label next to the test name in the output. JUnit (Java only) will generate a HTML document in `build/reports/tests/test/index.html` with a more detailed overview of the results; if there are any failed tests a link to render the document in your browser will be printed in the terminal output.

Par défaut, Gradle exécute les tests chaque fois que le code robot est compilé, y compris les déploiements. Cela augmentera le temps de déploiement, et des tests défaillants et provoquera l'échec de la compilation et du déploiement. Pour éviter que cela ne se produise, vous pouvez utiliser *Change Skip Tests On Deploy Setting* de la palette de commande pour configurer, s'il y a lieu, l'exécution des tests lors du déploiement.

23 OutlineViewer

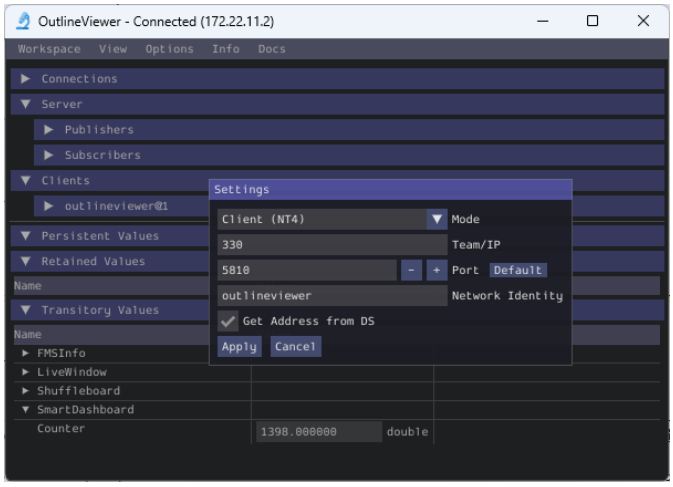


OutlineViewer est un utilitaire utilisé pour afficher, modifier et ajouter au contenu des NetworkTables à des fins de débogage. Il affiche toutes les paires clés/valeurs actuellement dans les NetworkTables et peut être utilisé pour modifier la valeur des clés existantes ou ajouter de nouvelles clés à la table. OutlineViewer est inclus dans les installations des langages Java et C++.

Sous Visual Studio Code, appuyez sur `Ctrl+Shift+P` et tapez `WPILib` ou cliquez sur le logo WPILib en haut à droite pour lancer la palette de commande WPILib. Sélectionnez *Start Tool*, puis sélectionnez *OutlineViewer*.

To connect to your robot, open OutlineViewer and slect *options* then *settings* and set the `Team/IP` to be your team number. After you click *Apply*, OutlineViewer will connect. If you have trouble connecting to OutlineViewer please see the *Dashboard Troubleshooting Steps*.

Note : You can use `localhost` instead of a team number to point OutlineViewer at a simulated robot, Romi or XRP.

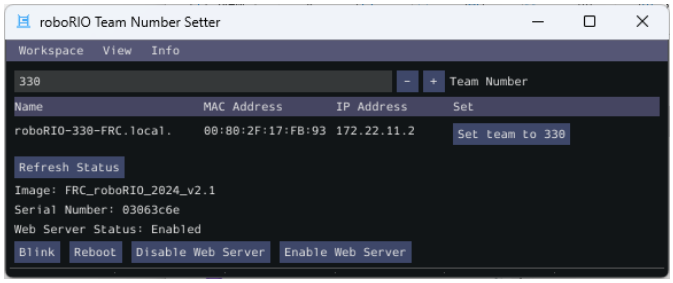


Pour ajouter des paires de clés/valeurs supplémentaires à NetworkTables, cliquez à droite sur un emplacement et choisissez le type de données correspondant.

Note : Les équipes LabVIEW peuvent utiliser l'onglet Variables du Dashboard LabVIEW pour obtenir les mêmes fonctionnalités que OutlineViewer.

---

24 roboRIO Team Number Setter



The roboRIO Team Number Setter is a cross-platform utility that can be used to set the team number on the roboRIO. It is an alternative to the roboRIO imaging tool for setting the team number.

In Visual Studio Code, press `Ctrl+Shift+P` and type `WPILib` or click the WPILib logo in the top right to launch the WPILib Command Palette. Select *Start Tool*, then select *roboRIOTeamNumberSetter*.

Connect to the roboRIO over USB to use the tool, as this is the simplest method when the team number hasn't been set.

24.1 Setting Team Number

Enter your team number in the *Team Number* field and select *Set team to xxxx*. This will take about a second, then press the *Reboot* button to reboot the roboRIO so the new team number takes effect.

24.2 Enabling/Disabling Webserver

The *roboRIO's webserver* provides some debuging and enables some configuration. However, it also takes memory away from the robot program. You can disable it by clicking on the *Disable Web Server* button. If you'd like to enable it again, you can click *Enable Web Server*.

24.3 roboRIO Identification

Clicking the *Blink* button will cause the roboRIO's Radio LED to blink a few times to help identify the roboRIO.

25 Traitement de la vision

25.1 Introduction à la vision

25.1.1 Qu'est-ce que la vision ?

La vision en FRC® utilise une caméra connectée au robot afin d'aider les équipes à marquer des points et d'aider à la conduite du robot, tant pendant les périodes autonomes que téléopérées.

Méthodes de vision

Il existe deux méthodes principales que la plupart des équipes utilisent pour la vision en FRC.

Diffusion

Cette méthode consiste à diffuser le signal vidéo de la caméra vers la station de pilotage afin que le conducteur et le manipulateur puissent obtenir des informations visuelles du point de vue du robot. Cette méthode est simple et prend peu de temps à mettre en œuvre, ce qui en fait une bonne option si vous n'avez pas besoin des fonctionnalités de traitement de la vision.

— *Streaming using the roboRIO*

Traitement de données

Instead of only streaming the camera to the Driver Station, this method involves using the frames captured by the camera to compute information, such as a game piece's or target's angle and distance from the camera. This method requires more technical knowledge and time in order to implement, as well as being more computationally expensive. However, this method can help improve autonomous performance and assist in « auto-scoring » operations during the teleoperated period. This method can be done using the roboRIO or a coprocessor such as the Raspberry Pi using OpenCV.

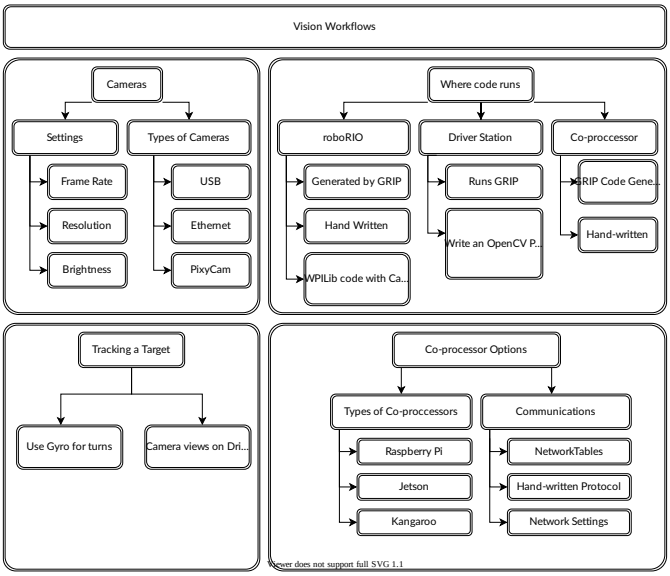
— *Vision Processing with Raspberry Pi*
— *Vision Processing with the roboRIO*

Pour plus d'informations sur les avantages et les inconvénients de l'utilisation d'un coprocesseur pour le traitement de la vision, consultez la page suivante : *Stratégies pour la programmation de la vision*.

25.1.2 Stratégies pour la programmation de la vision

L'utilisation de la vision par ordinateur est un excellent moyen de rendre votre robot sensible aux éléments sur le terrain et de le rendre beaucoup plus autonome. Souvent dans les jeux FRC®, il ya des points bonus pour les balles tirées en mode autonome ou d'autres pièces de jeu dans les buts ou la navigation vers des endroits précis sur le terrain. La vision par ordinateur est un excellent moyen de résoudre bon nombre de ces problèmes. Et si vous avez du code autonome qui peut relever le défi, alors il peut être utilisé pendant la période de téléopéré et ainsi assister les conducteurs humains lors du pilotage.

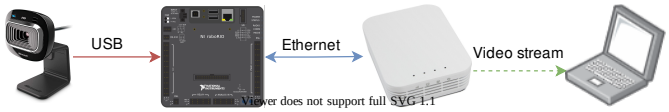
Il existe de nombreuses options pour choisir les composants pour le traitement de la vision et où le programme de vision va s'exécuter. WPILib et les outils associés prennent en charge un certain nombre d'options et offrent aux équipes une grande flexibilité pour décider quoi faire. Cet article va tenter de vous donner un aperçu de la plupart des choix et des compromis disponibles.



Librairie de vision par ordinateur OpenCV

OpenCV est une librairie de vision par ordinateur de type open-source largement utilisée dans le monde universitaire et l'industrie. Elle est supportée par les fabricants de matériel fournissant un traitement accéléré par GPU, et dispose de liens pour un certain nombre de langages, notamment C++, Java et Python. Elle est également bien documentée avec de nombreux sites Web, livres, vidéos et cours de formation Il existe donc de nombreuses ressources disponibles pour vous aider à apprendre à l'utiliser. Les versions C++ et Java de WPILib incluent les librairies OpenCV, celle-ci se charge de la capture, du traitement et de la visualisation de vidéos et comprend des outils pour vous aider à créer vos algorithmes de vision. Pour plus d'informations sur OpenCV, consultez https://opencv.org.

Vision Code sur roboRIO

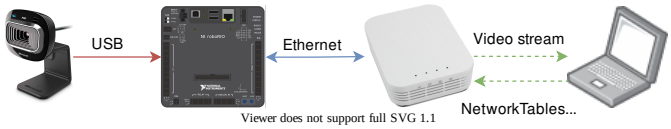


Vision code can be embedded into the main robot program on the roboRIO. Building and running the vision code is straightforward because it is built and deployed along with the robot program. The vision code can be written in C++, Java, or Python. The disadvantage of this approach is that having vision code running on the same processor as the robot program can cause performance issues. This is something you will have to evaluate depending on the requirements for your robot and vision program.

In this approach, the vision code simply produces results that the robot code directly uses. Be careful about synchronization issues when writing robot code that is getting values from a vision thread. The VisionRunner class in WPILib make this easier.

En utilisant les fonctions fournies par la classe CameraServer, le flux vidéo peut être envoyé à des tableaux de bord tels que Shuffleboard afin que les opérateurs puissent voir ce que la caméra voit. De plus, des annotations peuvent être ajoutées aux images à l'aide des commandes OpenCV afin que les cibles ou autres objets intéressants puissent être identifiés dans la disposition du tableau de bord.

Code de vision sur ordinateur DS

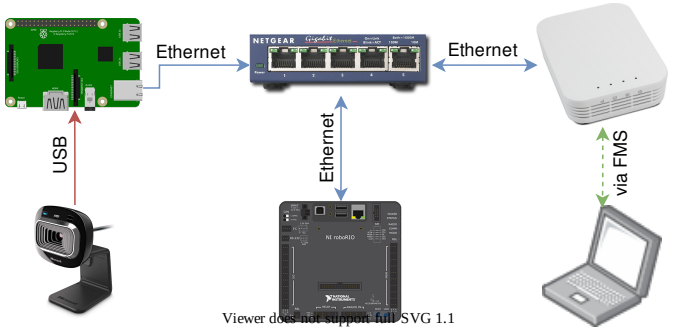


When vision code is running on the DS computer, the video is streamed back to the Driver Station laptop for processing. Even the older Classmate laptops are substantially faster at vision processing than the roboRIO. You can write your own vision program using a language of your choosing. Python makes a good choice since there is a native NetworkTables implementation and the OpenCV bindings are very good.

Une fois les images traitées, les valeurs clés telles que la position cible, la distance ou tout autre élément dont vous avez besoin peuvent être renvoyées au robot avec NetworkTables. Cette approche a généralement une latence plus élevée, car un délai est ajouté en raison des images devant être envoyées à l'ordinateur portable. Les limitations de bande passante limitent également la résolution maximale et le FPS des images utilisées pour le traitement.

The video stream can be displayed on Shuffleboard or SmartDashboard.

Code de vision sur coprocesseur



Les coprocesseurs tels que le Raspberry Pi sont idéaux pour prendre en charge le code de vision (voir *Utilisation du Raspberry Pi pour FRC*). L'avantage est qu'ils peuvent fonctionner à pleine vitesse et ne pas interférer avec le programme du robot. Dans ce cas, la caméra est probablement connectée au coprocesseur ou (dans le cas des caméras Ethernet) à un interrupteur Ethernet sur le robot. Le programme peut être écrit dans n'importe quelle langage ; Python est un bon choix en raison de ses liaisons simples à OpenCV et NetworkTables. Certaines équipes ont utilisé des coprocesseurs de vision haute performance tels que le Nvidia Jetson pour sa vitesse très rapide et sa plus haute résolution, bien que cette approche nécessite généralement des connaissances avancées sur Linux et en programmation.

Cette approche nécessite un peu plus d'expertise en programmation et ajoute un léger poids supplémentaire au robot, cependant, c'est le meilleur des deux mondes par rapport aux deux autres approches. Les coprocesseurs sont beaucoup plus rapides que le roboRIO et le traitement d'image peut être effectué avec latence minimale ou utilisation de la bande passante.

Les données peuvent être envoyées du programme de vision vers le coprocesseur via les NetworkTables ou par un protocole privé sur un réseau ou une connexion série.

Options de caméra

WPILib prend en charge un certain nombre d'options de caméra. Les caméras ont plusieurs paramètres qui influencent le fonctionnement ; par exemple, la fréquence d'images et la résolution d'image influencent la qualité des images reçues, mais lorsqu'elles sont réglés trop hautes, elles contribuent à un temps de traitement très élevé et, si elles sont envoyées à l'ordinateur portable Driver Station pour traitement, elles peuvent causer un dépassement de la bande passante disponible sur le terrain.

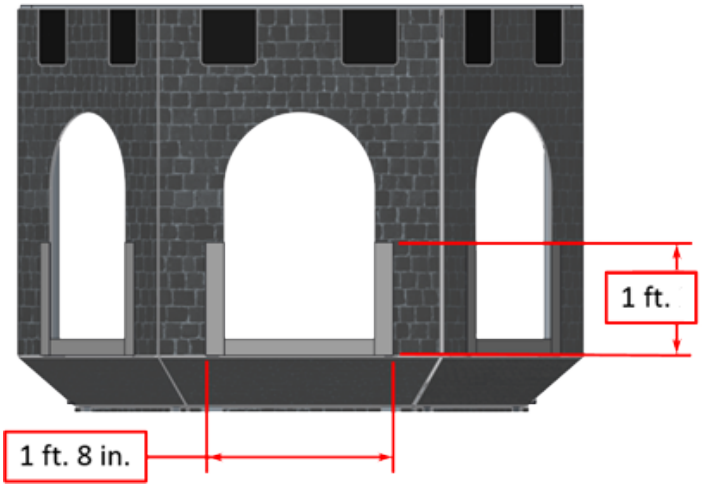
La classe CameraServer en C++ et Java est utilisée pour interfacer avec les caméras connectées au robot. Elle récupère les trames pour le traitement local via un objet Source et ensuite envoie le flux au poste de pilotage pour visualisation ou traitement.

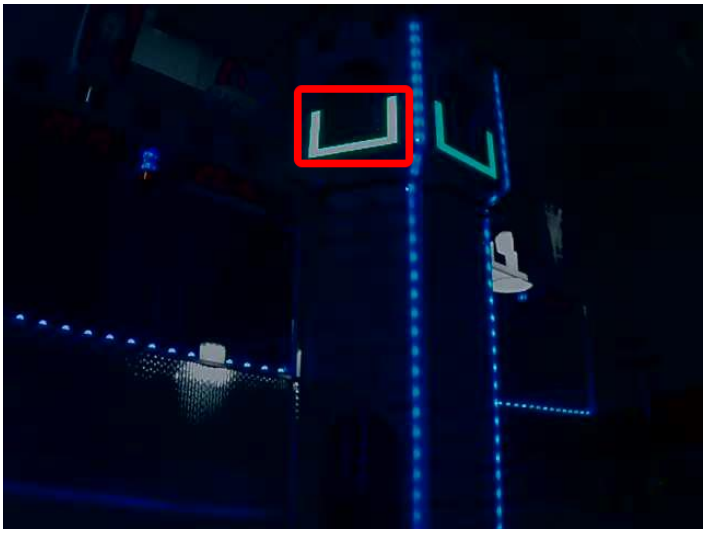
25.1.3 Informations sur la cible et rétroréflexion

Beaucoup de jeux FRC® ont un ruban rétroréfléchissant attaché aux éléments de terrain pour aider dans le traitement de vision. Ce document décrit les cibles utilisée en vision dans le jeu FRC 2016 et les propriétés visuelles du matériau dont sont consituées ces cibles.

Note : Pour les dimensions officielles et les dessins de tous les composants sur le terrain, veuillez consulter les Dessins Officiels du Terrain.

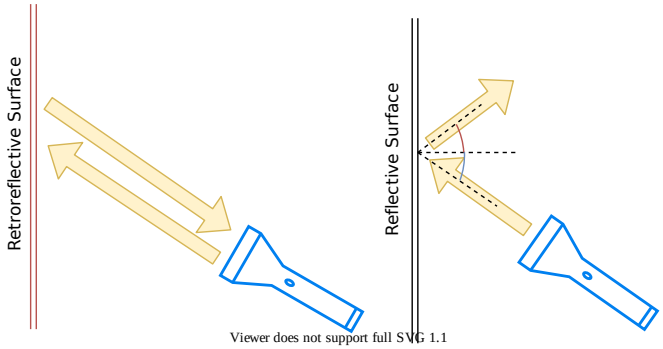
Cibles





Chaque cible de vision pour la saison 2016 consistait en un U de 1′ 8″ de largeur et 1′ de hauteur fabriqué avec un matériau rétroréfléchissant de 2″ de largeur (3M 8830 Silver Marking Film). Les cibles sont situées adjacentes au bas de chaque but. Lorsqu'elles sont correctement éclairées, les bandes rétroréfléchissantes produisent un marquage lumineux et/ou saturé de couleur.

Rétroréflectivité vs réflectivité



Les matériaux hautement réfléchissants sont généralement comme un miroir, de sorte que la lumière « rebondit » à un angle supplémentaire. Comme illustré ci-dessus à gauche, les angles bleu et rouge totalisent 180 degrés. Une explication similaire est que la lumière réfléchit sur la surface normale à la ligne verte tracée perpendiculairement à la surface. Notez qu'une lumière pointée vers la surface ne retournera à la source lumineuse que si l'angle bleu est de ~90 degrés.

Retro-reflective materials are not mirrored, but it will typically have either shiny facets across the surface, or it will have a pearl-like appearance. Not all faceted or pearl-like materials exhibit *retro-reflection*, however. Retro-reflective materials return the majority of light back to the light source, and they do this for a wide range of angles between the surface and the light source, not just the 90 degree case. Retro-reflective materials accomplish this using small prisms, such as found on a bicycle or roadside reflector, or by using small spheres with the appropriate index of refraction that accomplish multiple internal reflections. In nature, the eyes of some animals, including house cats, also exhibit the retro-reflective effect typically referred to as night-shine.

Exemples de rétroréflexion

Ce matériau est relativement familier, car il est souvent utilisé pour améliorer la visibilité nocturne des panneaux de signalisation, des vélos et des piétons.

Initialement, la rétroréflexion peut ne pas sembler être une propriété utile pour la sécurité nocturne, mais lorsque la lumière et l'œil sont proches l'un de l'autre, comme illustré ci-

dessus, la lumière réfléchie retourne vers l'œil et le matériau brille intensément, même à grande distance. En raison du petit angle entre les yeux du conducteur et les phares du véhicule, les matériaux rétroréfléchissants peuvent augmenter considérablement la visibilité des objets distants pendant la conduite de nuit.

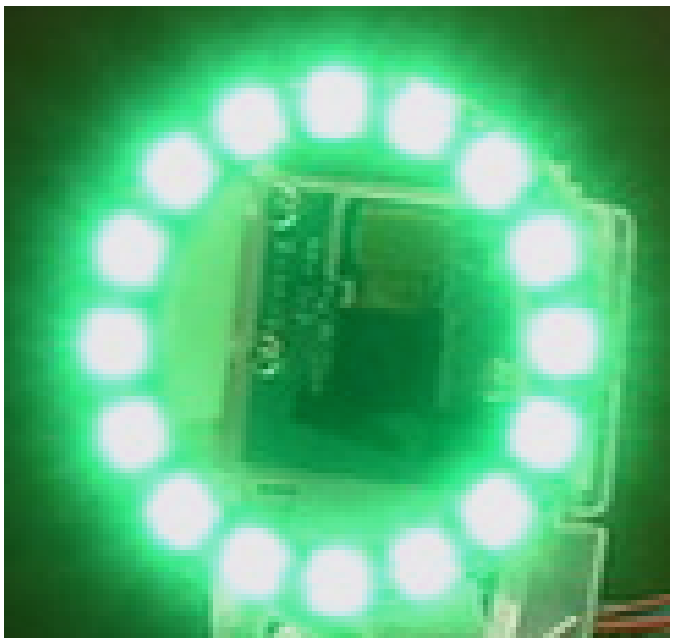
Démonstration

Pour explorer davantage les propriétés des matériaux rétroréfléchissants :

1. Placer un morceau du matériau sur un mur ou une surface verticale
2. Tenez-vous à 10-20 pieds (3 -6 mètres) de distance et braquez une petite lampe de poche sur le matériau.
3. Commencez par la lumière tenue au niveau de votre ceinture et levez-la lentement jusqu'à ce qu'elle soit au niveau de vos yeux. Lorsque la lumière se rapproche de vos yeux, l'intensité de la lumière renvoyée augmente rapidement.
4. Modifiez l'angle en vous déplaçant vers d'autres endroits de la pièce et en répétant. La réflexion lumineuse doit se produire sur une large gamme d'angles de vision, mais l'angle entre la source lumineuse et l'œil est la clé du succès, et doit être assez petit.

Expérimentez avec différentes sources de lumière. Le matériau est des centaines de fois plus réfléchissant que la peinture blanche ; les sources lumineuses faibles fonctionneront donc bien. Par exemple, un feu de sécurité de vélo rouge démontrera que la couleur de la source de lumière détermine la couleur de la lumière réfléchie. Si possible, placez plusieurs membres de l'équipe à différents endroits, chacun avec sa propre source de lumière. Cela montrera que les effets sont largement indépendants et que le matériau peut apparaître simultanément de différentes couleurs aux différents membres de l'équipe. Cela démontre également que le matériau est largement insensible à l'éclairage ambiant. La lumière renvoyée au membres de l'équipe est presque entièrement déterminée par une source lumineuse qu'ils contrôlent ou une située directement derrière eux. À l'aide de la lampe de poche, identifiez d'autres articles rétroréfléchissants déjà dans votre environnement : soit sur les vêtements, sacs à dos, chaussures, etc.

Éclairage



Nous avons vu que le ruban rétroréfléchissant ne brille seulement si une source de lumière est dirigée vers lui, et que la source de lumière doit passer très près de l'objectif de la caméra ou des yeux de l'observateur. Bien qu'il existe un certain nombre de façons d'accomplir cela, une source de lumière à considérer est le flash annulaire, ou anneau lumineux, illustré ci-dessus. Il place la source de lumière directement autour de l'objectif de la caméra et fournit un éclairage très uniforme. En raison de leur intensité lumineuse et de leur petite taille, les LED sont particulièrement utiles pour construire ce type d'appareil.

Comme indiqué ci-dessus, des anneaux lumineux peu coûteux de LED sont disponibles dans une variété de couleurs et de tailles et sont faciles à attacher aux caméras. Certains peuvent même être alimenté par l'entremise d'un Raspberry Pi. Bien qu'ils ne soient pas conçus pour un éclairage diffus et uniforme, ils fonctionnent assez bien pour faire briller le ruban rétro-réfléchissant. Un petit anneau LED vert est disponible via FIRST Choice. D'autres anneaux LED similaires sont disponibles auprès de fournisseurs, sous la rubrique SuperBrightLED.

Exemples d'images

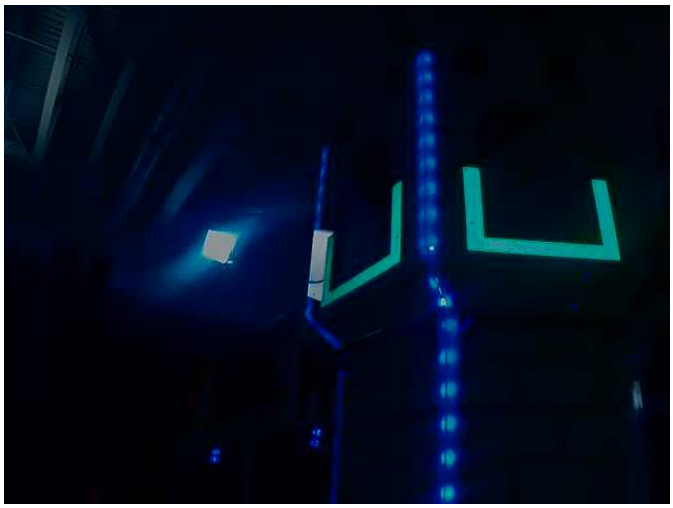
Des exemples d'images sont fournies avec les exemples de code pour chaque langage de programmation (déjà inclus avec LabVIEW et pour Java et C++, dans un fichier ZIP).

25.1.4 Identification et traitement des cibles

Une fois qu'une image est capturée, l'étape suivante consiste à identifier les cibles de vision dans l'image. Ce document présentera une approche pour identifier les cibles de la saison 2016. Notez que les images utilisées dans cette section ont été prises avec la caméra intentionnellement réglée pour sous-exposer les images, produisant des images très sombres à l'exception des cibles éclairées, voir la section sur les paramètres de caméra pour plus de détails.

Image d'origine

L'image ci-dessous est l'image de départ pour notre exemple. L'image a été prise à l'aide de l'anneau lumineux vert disponible dans *FIRST*® Choice, combiné avec un anneau supplémentaire d'une taille différente. Des exemples d'images sont fournis avec les exemples de code de vision.



Qu'est-ce que HSL/HSV ?

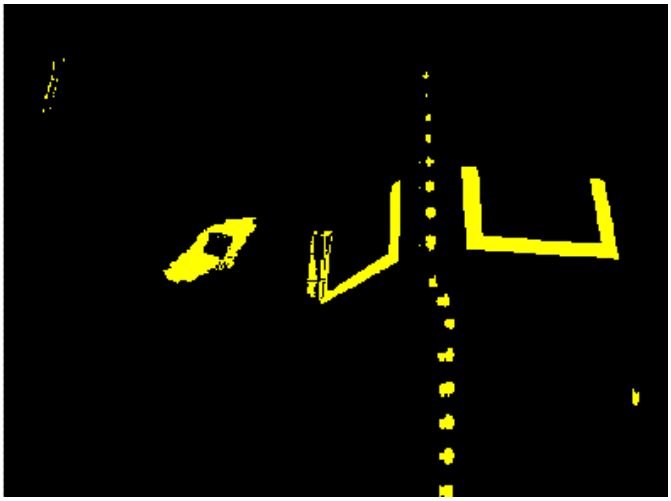
La teinte de la couleur est généralement visible sur la roue chromatique et contient les couleurs de l'arc-en-ciel, soit le rouge, orange, jaune, vert, bleu, indigo et violet. La teinte est spécifiée en utilisant un angle radial sur la roue, mais en imagerie, le cercle ne contient généralement que 256 unités, en commençant par le rouge à zéro, en passant par l'arc-en-ciel et en revenant au rouge à l'extrémité supérieure. La saturation d'une couleur spécifie la quantité de couleur ou le rapport de la couleur de teinte à une quantité donnée de gris. Un rapport plus élevé signifie plus de couleurs, moins de gris. La saturation zéro n'a pas de teinte et est complètement grise. La luminance ou la valeur indique la quantité de gris avec laquelle la teinte est mélangée. Le noir est 0 et le blanc 255.

L'exemple de code utilise l'espace colorimétrique HSV pour spécifier la couleur de la cible. La principale raison est qu'elle permet facilement d'utiliser la luminosité des cibles par rapport au reste de l'image comme critère de filtrage en utilisant le composant Valeur (HSV) ou Luminance (HSL). Une autre raison d'utiliser le système de couleurs HSV est que l'opération de seuillage utilisée dans l'exemple s'exécute plus efficacement sur le roboRIO lorsqu'elle est effectuée dans l'espace colorimétrique HSV.

Masquage

Dans cette étape initiale, les valeurs des pixels sont comparées aux valeurs de couleur ou de luminosité constantes pour créer un masque binaire illustré ci-dessous en jaune. Cette étape unique élimine la plupart des pixels qui ne font pas partie du ruban rétroréfléchissant d'une cible. Le masquage basé sur la couleur fonctionne bien à condition que la couleur soit relativement saturée, lumineuse et cohérente. Les inégalités de couleur sont généralement plus précises lorsqu'elles sont spécifiées à l'aide de l'espace colorimétrique HSL (teinte, saturation et luminance) ou HSV (teinte, saturation et valeur) que l'espace RVB (rouge, vert et bleu). Cela est particulièrement vrai lorsque la gamme de couleurs est assez large dans une ou plusieurs dimensions.

Notez qu'en plus de la cible, d'autres parties lumineuses de l'image (lumière aérienne et éclairage de la tour) sont également capturées par l'étape de masquage.



Analyse des particules

Après l'opération de masquage, une opération de rapport de particules est utilisée pour examiner la zone, le rectangle de délimitation et le rectangle équivalent pour les particules. Ceux-ci sont utilisés pour calculer plusieurs termes notés pour aider à choisir les formes les plus rectangulaires. Chaque test décrit ci-dessous génère un score (0-100) qui est ensuite comparé aux limites de score prédéfinies pour décider si la particule est une cible ou non.

Zone de couverture

Le score d'aire est calculé en comparant l'aire de la particule à l'aire de la boîte englobante dessinée autour de la particule. La surface des bandes rétroréfléchissantes est de 80 pouces carrés (~516 cm^2). La zone du rectangle qui contient la cible est de 240 pouces carrés (~0.15 m^2). Cela signifie que le rapport idéal entre la zone et la zone de la zone de délimitation est de 1/3. Des ratios de surface proches de 1/3 produiront un score proche de 100, car le ratio diverge de 1/3, le score approchera de 0.

Rapport d'aspect

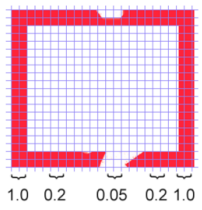
Le score du ratio d'aspect est basé sur le calcul (Largeur de particule / Hauteur de particule). La largeur et la hauteur de la particule sont déterminées en utilisant ce que l'on appelle le « rectangle équivalent ». Le rectangle équivalent est le rectangle avec des longueurs latérales x et y où $2x + 2y$ est égal au périmètre de particules et $x \cdot y$ est égal à la zone de particules. Le rectangle équivalent est utilisé pour le calcul du ratio d'aspect car il est moins affecté par l'inclinaison du rectangle que par l'utilisation du cadre de sélection. Lorsque vous utilisez le rectangle du cadre de sélection pour les proportions, à mesure que le rectangle est incliné, la hauteur augmente et la largeur diminue.

La cible mesure 20″ (508 mm) de largeur sur 12″ (304,8 mm) de hauteur, pour un rapport de 1.6. Le rapport hauteur / largeur détecté est comparé à ce rapport idéal. Le score du rapport hauteur / largeur est normalisé pour renvoyer 100 lorsque le rapport correspond au rapport cible et diminue linéairement lorsque le rapport varie en dessous ou au-dessus.

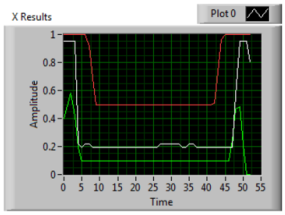
Moment d'inertie

The « moment » measurement calculates how spread out each pixel is from the center of the blob. This measurement provides a representation of the pixel distribution in the particle. It can be thought of as analogous to a physics *moment of inertia* calculation. The ideal score for this test is ~0.28.

Profils X/Y



Column averages for a particle rectangle.



White line is the average, red is upper limit, and green is lower limit..

Ce score décrit si la particule correspond au profil approprié dans les directions X et Y. Il est calculé en utilisant les moyennes des lignes et des colonnes à travers le cadre de délimitation extraites de l'image d'origine et en les comparant à un masque de profil. Le score varie de 0 à 100 en fonction du nombre de valeurs dans les moyennes de ligne ou de colonne qui se situent entre les valeurs limites supérieure et inférieure.

Mesures

Si une particule a un score suffisamment élevé pour être considérée comme une cible, il est logique de calculer certaines mesures du monde réel telles que la position et la distance. L'exemple de code comprend ces mesures de base, alors examinons les mathématiques impliquées pour mieux les comprendre.

Position

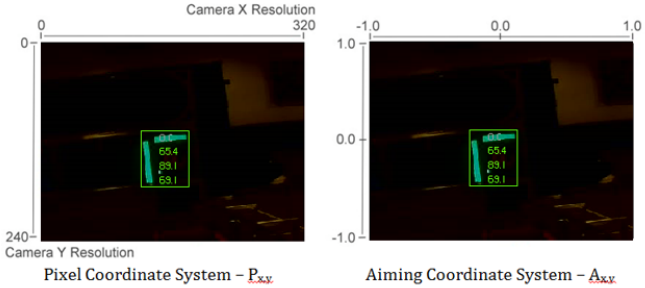
La position cible est identifiée à la fois par la particule et le cadre de sélection, mais toutes les coordonnées sont en pixels. 0.0 étant en haut et à gauche de l'écran et les bords droit et inférieur déterminés par la résolution de la caméra. Il s'agit d'un système utile pour les mathématiques des pixels, mais pas vraiment utile pour conduire un robot ; alors changeons-le en quelque chose qui pourrait être plus utile.

Pour convertir un point du système de pixels à un environnement plus compatible pour un robot, nous pouvons utiliser la formule ci-dessous.

Les coordonnées résultantes sont proches de ce que vous pouvez souhaiter, mais l'axe Y est inversé. Cela pourrait être corrigé en multipliant le point par [1, -1] (Remarque : cela n'est pas fait dans l'exemple de code). Ce système de coordonnées est utile car il a une origine centrée et l'échelle est similaire aux sorties du joystick et aux entrées de l'entraînement (Drive).

$$A_{x,y} = \left(P_{x,y} - \frac{resolution_{x,y}}{2}\right) / \frac{resolution_{x,y}}{2}$$

$$A_{x,y} = (P_{x,y} - \frac{resolution_{x,y}}{2}) / \frac{resolution_{x,y}}{2}$$



Pixel Coordinate System – $P_{x,y}$ Aiming Coordinate System – $A_{x,y}$

Champ de vision

Vous pouvez utiliser des constantes connues et la position de la cible sur le plan de coordonnées pour déterminer votre distance, lacet (yaw) et tangage (pitch) par rapport à la cible. Cependant, pour les calculer, vous devez déterminer votre FOV (champ de vision). Afin de déterminer empiriquement le champ de vision vertical, réglez votre caméra à une distance définie d'une surface plane et mesurez la distance entre la rangée de pixels la plus haute et la plus basse.

$$\frac{1}{2}FOV_{vertical} = tan\left(\frac{\frac{1}{2}distance_y}{distance_z}\right)$$

Vous pouvez trouver le champ de vision horizontal en utilisant la même méthode, mais en utilisant la distance entre la première et la dernière colonne de pixels.

Tangage (Pitch) et Lacet (Yaw)

Trouver le tangage et le lacet de la cible par rapport à votre robot est simple une fois que vous connaissez vos FOV et l'emplacement de votre cible dans le système de coordonnées de visée.

$$pitch = \frac{A_y}{2} FOV_{vertical}$$

$$yaw = \frac{A_x}{2} FOV_{horizontal}$$

Distance

Si votre cible est à une hauteur sensiblement différente de celle de votre robot, vous pouvez utiliser des constantes connues, telles que la hauteur physique de la cible et de votre caméra, ainsi que l'angle de montage de votre caméra, pour calculer la distance entre votre caméra et la cible.

$$distance = \frac{height_{target} - height_{camera}}{tan(angle_{camera} + pitch)}$$

Une autre option consiste à créer une table de correspondance de la zone à la distance ou à estimer la constante de variation inverse de la zone et de la distance. Cependant, cette méthode est moins précise.

Note : Pour de meilleurs résultats pour les méthodes d'estimation d'angle et de distance ci-dessus, vous pouvez étalonner votre caméra à l'aide d'OpenCV pour éliminer les distorsions susceptibles d'affecter la précision en reprojetant les pixels de la cible à l'aide de la matrice d'étalonnage.

25.1.5 Lire et traiter la vidéo : classe CameraServer

Concepts

Les caméras généralement utilisées en FRC® (les caméras USB et Ethernet de base telles que la caméra Axis) offrent des modes de fonctionnement relativement limités. En général, elles ne fournissent qu'une seule sortie d'image (généralement dans un format compressé RGB tel que JPG) à une seule résolution et taux d'image. Les caméras USB sont particulièrement limitées car une seule application peut accéder à la caméra à la fois.

CameraServer peut prendre en charge plusieurs caméras. Il gère des détails tels que la reconnexion automatique lorsqu'une caméra est déconnectée momentanément, et met également les images de la caméra à la disposition de plusieurs « clients » (par exemple, votre code de robot et le tableau de bord peuvent lire les images de la caméra même simultanément).

Nommer les caméras

Chaque caméra dans CameraServer doit porter un nom unique. C'est également le nom qui apparaît pour la caméra dans le tableau de bord. Certaines variantes des fonctions CameraServer `startAutomaticCapture()` et `addAxisCamera()` nommeront automatiquement la caméra (par exemple « USB Camera 0 » ou « Axis Camera »), ou vous pouvez donner à la caméra un nom plus descriptif (par exemple « Cam d'admission »). La seule exigence est que chaque caméra ait un nom unique.

Remarques sur la caméra USB

L'utilisation du processeur

Le CameraServer est conçu pour minimiser l'utilisation du processeur en effectuant uniquement des opérations de compression et de décompression lorsque cela est nécessaire et en désactivant automatiquement le streaming lorsqu'aucun client n'est connecté.

Pour minimiser l'utilisation du processeur, la résolution du tableau de bord doit être réglée sur la même résolution que la caméra ; cela permet au CameraServer de ne pas décompresser et recompresser l'image, au lieu de cela, il peut simplement transmettre l'image JPEG reçue de la caméra directement au tableau de bord. Il est important de noter que la modification de la résolution sur le tableau de bord **ne change pas** la résolution de la caméra ; changer la résolution de la caméra peut être fait en appelant `setResolution()` sur l'objet caméra.

Bande passante USB

Le roboRIO ne peut transmettre et recevoir autant de données à la fois via ses interfaces USB. Les images de la caméra peuvent nécessiter beaucoup de données, et il est donc relativement facile de dépasser cette limite. La cause plus fréquente d'une erreur de bande passante USB est le choix d'un mode vidéo non JPEG ou une résolution trop élevée, en particulier lorsque plusieurs caméras sont connectées.

Architecture

Le CameraServer se compose de deux couches, la classe haut-niveau WPILib **CameraServer class** et la librairie de bas-niveau **cscore library**.

Classe CameraServer

La classe CameraServer (qui fait partie de WPILib) fournit une interface de haut niveau pour ajouter des caméras à votre code de robot. Elle est également responsable de la publication d'informations sur les caméras et les serveurs de caméras sur NetworkTables afin que les tableaux de bord de Driver Station tels que LabVIEW Dashboard et Shuffleboard puissent répertorier les caméras et déterminer l'emplacement de leurs flux. La classe utilise un modèle de type « singleton » (possédant un seul objet à la fois) pour maintenir une base de données de toutes les caméras et tous les serveurs créés.

Certaines fonctions importantes de CameraServer :

— `startAutomaticCapture()` : ajoute une caméra USB (par exemple Microsoft LifeCam) et lance un serveur pour qu'elle puisse être visualisée à partir du tableau de bord.

— `addAxisCamera()` : Ajoute une caméra Axis. Même si vous ne traitez pas les images de la caméra Axis dans votre code de robot, vous pouvez utiliser cette fonction pour que la caméra Axis apparaisse dans la liste déroulante des caméras du tableau de bord. Ceci démarre également un serveur afin que le flux Axis puisse toujours être affiché lorsque votre poste de pilotage est connecté au roboRIO via USB (utile en compétition si vous avez à la fois la caméra Axis et le roboRIO connectés aux deux ports Ethernet radio du robot).
— `getVideo()` : Obtenez l'accès OpenCV à une caméra. Cela vous permet d'obtenir des images de la caméra pour le traitement d'images sur le roboRIO (dans votre code de robot).
— `putVideo()` : démarrez un serveur sur lequel vous pouvez fournir des OpenCV à ce serveur. Cela vous permet de passer des images traitées et/ou annotées personnalisées au tableau de bord.

Librairie cscore

La librairie cscore fournit l'implémentation de niveau bas pour :
— Obtenir des images de caméras USB et HTTP (par exemple Axis)
— Modifier les paramètres de l'appareil photo (par exemple le contraste et la luminosité)
— Changer les modes vidéo de la caméra (format pixel, résolution et fréquence d'images)
— Agir en tant que serveur Web et fournir des images en tant que flux MJPEG standard
— Convertir des images vers/depuis des objets `Mat` OpenCV pour le traitement d'images

Sources et Récepteurs

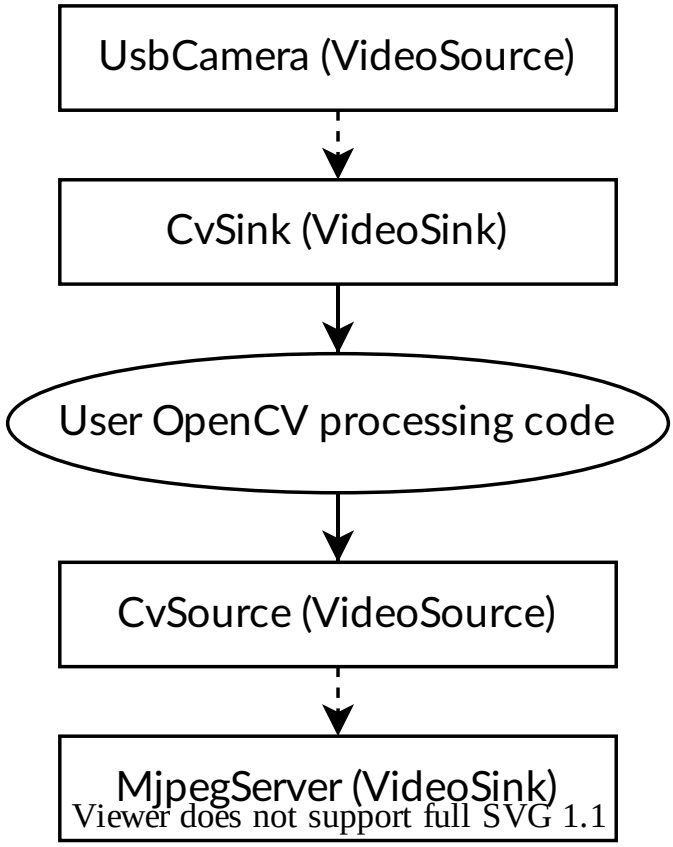
L'architecture de base de la bibliothèque cscore est similaire à celle de MJPGStreamer, avec des fonctionnalités réparties entre les sources et les récepteurs. Il peut y avoir plusieurs Sources et plusieurs Récepteurs créés et fonctionnant simultanément.

Un objet qui génère des images est une Source et un objet qui accepte/consomme des images est un Récepteur. La génération/consommation est du point de vue de la librairie. Les caméras sont donc des Sources (elles génèrent des images). Le serveur Web MJPEG est un Récepteur, car il accepte des images à partir du programme (même s'il peut transférer ces images vers un navigateur Web ou un tableau de bord). Les Sources peuvent être connectées à plusieurs Récepteurs, mais les Récepteurs peuvent être connectés à une seule et unique Source. Lorsqu'un Récepteur est connecté à une Source, la librairie cscore se charge de transmettre chaque image de la Source au Récepteur.

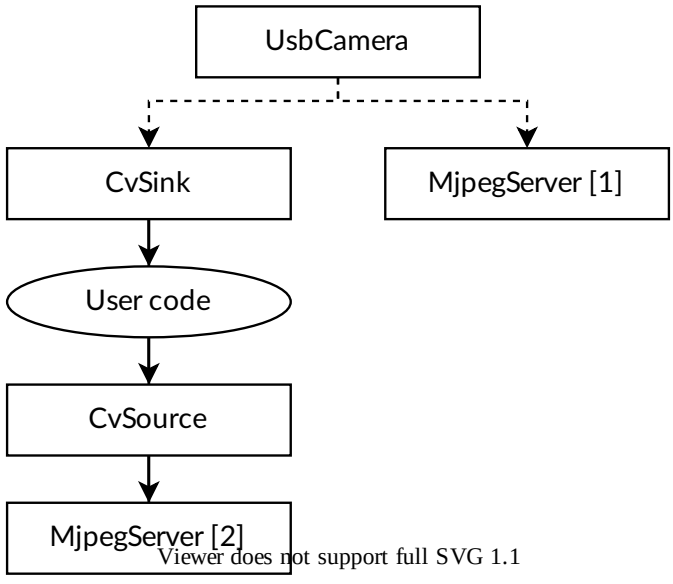
— **Les Sources** obtiennent des images individuelles (comme celles fournies par une caméra USB) et déclenchent un événement lorsqu'une nouvelle image est disponible. Si aucun Récepteur n'écoute une Source particulière, la librairie peut s'arrêter ou se déconnecter d'une Source pour économiser le processeur et les ressources d'Entrées/Sorties. La librairie gère de manière autonome les déconnexions/reconnexions de la caméra en interrompant simplement et en reprenant le déclenchement des événements (par exemple, une déconnexion arrête l'envoie de nouvelles trames, mais ne génère pas d'erreur).
— **Les Récepteurs** écoutent l'événement d'une Source particulière, récupérent la dernière image et la transfère à sa destination dans le format approprié. De même que pour les Sources, si un Récepteur particulier est inactif (par exemple, aucun client n'est connecté à un serveur MJPEG configuré sur HTTP), la librairie peut désactiver des parties de son traitement pour économiser les ressources du processeur.

Le code utilisateur (tel que celui utilisé dans un programme de robot FRC) peut agir soit comme Source (fournissant des images traitées comme s'il s'agissait d'une caméra), soit

comme Récepteur (recevant une image pour traitement) via les objets Source et Récepteur OpenCV. Ainsi, un pipeline de traitement d'image qui récupère les images d'une caméra et sert les images traitées ressemble au graphique ci-dessous :



Étant donné que les Sources peuvent avoir plusieurs Récepteurs connectés, le pipeline peut se ramifier. Par exemple, l'image d'origine de la caméra peut également être servie en connectant la Source UsbCamera à un deuxième Récepteur MjpegServer en plus du récepteur CvSink, ce qui donne le graphique ci-dessous :



Lorsqu'une nouvelle image est capturée par la caméra, le CvSink et le MjpegServer [1] la reçoivent.

Le graphique ci-dessus est crée par le bout de code CameraServer ci-dessous :

JAVA

```
import edu.wpi.first.cameraserver.CameraServer;
import edu.wpi.cscore.CvSink;
import edu.wpi.cscore.CvSource;

// Creates UsbCamera and MjpegServer [1] and connects them
CameraServer.startAutomaticCapture();

// Creates the CvSink and connects it to the UsbCamera
CvSink cvSink = CameraServer.getVideo();

// Creates the CvSource and MjpegServer [2] and connects them
CvSource outputStream = CameraServer.putVideo("Blur", 640, 480);
```

C++

```
#include "cameraserver/CameraServer.h"

// Creates UsbCamera and MjpegServer [1] and connects them
frc::CameraServer::StartAutomaticCapture();

// Creates the CvSink and connects it to the UsbCamera
cs::CvSink cvSink = frc::CameraServer::GetVideo();

// Creates the CvSource and MjpegServer [2] and connects them
cs::CvSource outputStream = frc::CameraServer::PutVideo("Blur", 640, 480);
```

L'implémentation de CameraServer effectue les opérations suivantes au niveau cscore : CameraServer prend en charge de nombreux détails tels que la création de noms uniques pour tous les objets cscore et la sélection automatique des numéros de port. CameraServer conserve également un registre de type « singleton » des objets créés afin qu'ils ne soient pas détruits s'ils sortent de leur cadre, ou champ.

JAVA

```
import edu.wpi.cscore.CvSink;
import edu.wpi.cscore.CvSource;
import edu.wpi.cscore.MjpegServer;
import edu.wpi.cscore.UsbCamera;

// Creates UsbCamera and MjpegServer [1] and connects them
UsbCamera usbCamera = new UsbCamera("USB Camera 0", 0);
MjpegServer mjpegServer1 = new MjpegServer("serve_USB Camera 0", 1181);
mjpegServer1.setSource(usbCamera);

// Creates the CvSink and connects it to the UsbCamera
```

(suite sur la page suivante)

(suite de la page précédente)

```
CvSink cvSink = new CvSink("opencv_USB Camera 0");
cvSink.setSource(usbCamera);

// Creates the CvSource and MjpegServer [2] and connects them
CvSource outputStream = new CvSource("Blur", PixelFormat.kMJPEG, 640, 480, 30);
MjpegServer mjpegServer2 = new MjpegServer("serve_Blur", 1182);
mjpegServer2.setSource(outputStream);
```

C++

```
#include "cscore_oo.h"

// Creates UsbCamera and MjpegServer [1] and connects them
cs::UsbCamera usbCamera("USB Camera 0", 0);
cs::MjpegServer mjpegServer1("serve_USB Camera 0", 1181);
mjpegServer1.SetSource(usbCamera);

// Creates the CvSink and connects it to the UsbCamera
cs::CvSink cvSink("opencv_USB Camera 0");
cvSink.SetSource(usbCamera);

// Creates the CvSource and MjpegServer [2] and connects them
cs::CvSource outputStream("Blur", cs::PixelFormat::kMJPEG, 640, 480, 30);
cs::MjpegServer mjpegServer2("serve_Blur", 1182);
mjpegServer2.SetSource(outputStream);
```

Comptage de références

Tous les objets cscore sont comptés en interne. La connexion d'un Récepteur à une Source augmente le compte de références de la Source, il est donc strictement nécessaire de garder le Récepteur dans sa portée. La classe CameraServer conserve un registre de tous les objets créés avec les fonctions CameraServer, de sorte que les Sources et les Récepteurs créés de cette manière ne sortent jamais de leur portée (sauf s'ils sont explicitement supprimés).

25.1.6 Exemples de vision pour la saison 2017

LabVIEW

L'exemple de vision en LabVIEW 2017 est inclus dans les autres exemples LabVIEW. Depuis la fenêtre d'accueil, cliquez sur Support->Find FRC® Examples ou à partir de toute autre fenêtre LabVIEW, cliquez sur Help->Find Examples et localisez le dossier Vision pour trouver 2017 Vision Example. Les images d'exemple sont groupées avec l'exemple.

25.2 Vision avec WPILibPi

25.2.1 Vidéo faisant un passage en revue du processus d'utilisation de WPILibPi avec le Raspberry Pi.

Note : La vidéo parle de FRCVision qui est l'ancien nom de WPILibPi.

Lors de la «RSN Spring Conference, Presented by WPI» en 2020, Peter Johnson de l'équipe WPILib a fait une présentation sur FRC® Vision avec un Raspberry Pi.

Le lien vers la présentation est disponible ici.

25.2.2 Utilisation d'un coprocesseur pour le traitement de la vision

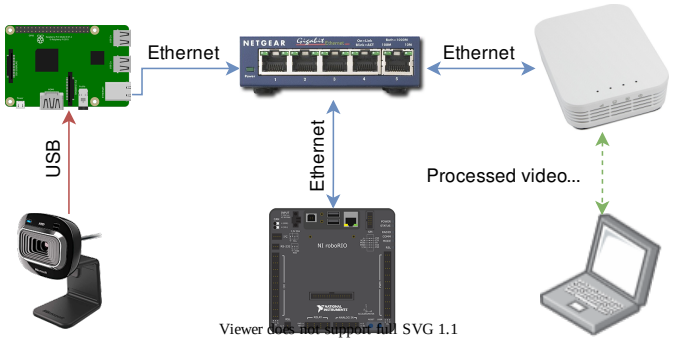
Le traitement de la vision à l'aide de bibliothèques comme OpenCV pour reconnaître des cibles de terrain ou des pièces de jeu peut souvent être un processus gourmand en ressources processeur. Souvent, la charge n'est pas trop importante et le traitement peut être facilement géré par le roboRIO. Dans les cas où il y a plus de flux de caméra ou où le traitement de l'image est complexe, il est souhaitable de décharger le roboRIO en plaçant le code et la connexion de la caméra sur un processeur différent. Il existe un certain nombre de choix de processeurs qui sont populaires dans FRC® comme le Raspberry PI, le Kangaroo basé sur Intel, le LimeLight pour le summum de la simplicité, ou pour le code de vision plus complexe un accélérateur graphique tel que l'un des modèles nVidia Jetson.

Stratégie

Généralement, l'idée est de configurer le coprocesseur avec le logiciel requis qui comprend généralement :

- OpenCV - la librairie de vision par ordinateur
- *NetworkTables* - pour commuter les résultats du traitement d'image vers le programme roboRIO
- Bibliothèque de serveurs de caméras - pour gérer les connexions de caméras et publier des flux pouvant être affichés sur un tableau de bord
- Les librairies associés au langage informatique utilisé pour le programme de vision
- Le programme spécifique de vision qui fait la détection d'objet

Le coprocesseur est connecté au réseau roboRIO en le branchant sur le port Ethernet supplémentaire du routeur réseau ou, pour plus de connexions, en ajoutant un petit commutateur réseau (Switch) au robot. Les caméras sont connectées au coprocesseur, il acquiert les images, les traite et publie les résultats, généralement des informations de localisation de la cible, sur NetworkTables afin qu'elles puissent être utilisées par le programme du robot pour la direction et la visée.



Diffuser des données de la caméra vers le tableau de bord

Il est souvent souhaitable de simplement diffuser les données de la caméra sur le tableau de bord via le réseau du robot. Dans ce cas, une ou plusieurs connexions de caméra peuvent être envoyées au réseau et affichées sur un tableau de bord tel que Shuffleboard ou un navigateur Web. L'utilisation de Shuffleboard donne l'avantage d'avoir accès à des commandes faciles pour régler la résolution de la caméra et le débit binaire ainsi que d'intégrer les flux de caméra avec d'autres données envoyées par le robot.

Il est également possible de traiter des images et d'ajouter des annotations à l'image, telles que des lignes ou des boîtes cibles montrant ce que le code de traitement d'image a détecté, puis de l'envoyer vers le tableau de bord pour permettre aux opérateurs de voir plus clairement ce qui se trouve autour du robot.

25.2.3 Utilisation du Raspberry Pi pour FRC

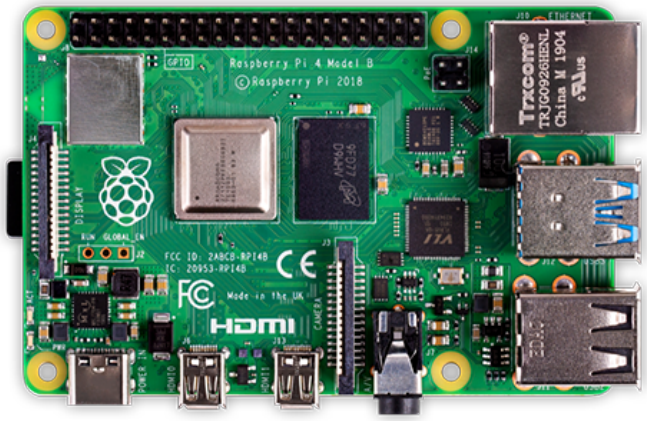
Un des coprocesseurs les plus populaires est le Raspberry Pi car :

- Il a un coût raisonnable - environ $35 US
- Sa disponibilité - il est facile de trouver le Raspberry Pi auprès d'un certain nombre de fournisseurs, y compris Amazon
- Il a de très bonnes performances - le Raspberry Pi 3b + a les spécifications suivantes :
- Spécifications techniques : - Ordinateur à carte unique Broadcom BCM2837BO 64 bits ARMv8 QUAD Core A53 64 bits alimenté par processeur à 1,4 GHz - 1 Go de RAM - BCM43143 WiFi à bord - Bluetooth Low Energy (BLE) à bord - GPIO étendu à 40 broches - 4 ports USB2 - Sortie stéréo 4 pôles et port vidéo composite - HDMI pleine taille - Port de caméra CSI pour connecter la caméra Raspberry Pi - Port d'affichage DSI pour connecter un écran tactile - Port MicroSD pour charger votre système d'exploitation et stocker des données - Connecteur pour la source d'alimentation capable de prendre en charge jusqu'à 2.5 ampères sur les ports USB.

Image Raspberry Pi pré-construite

Pour rendre l'utilisation du Raspberry Pi aussi simple que possible pour les équipes, une image Raspberry Pi est fournie. L'image peut être copiée sur une carte micro SD, insérée dans le Pi et démarrée. Par défaut, il prend en charge :

- Une interface web pour configurer le rPi pour les fonctions les plus courantes
- Prend en charge un nombre arbitraire de flux de caméras (par défaut un) qui sont publiés sur l'interface réseau
- OpenCV, *NetworkTables*, Camera Server et librairies de langage pour les programmes personnalisés C ++, Java et Python



Si la seule exigence est de diffuser une ou plusieurs caméras sur le réseau (et le tableau de bord), aucune programmation n'est requise et ces tâches peuvent être entièrement configurées via l'interface Web.

La section suivante explique comment installer l'image sur une carte flash et démarrer le Pi.

25.2.4 Ce dont vous avez besoin pour faire fonctionner l'image Pi

Pour commencer à utiliser le Raspberry Pi comme coprocesseur vidéo ou image, vous avez besoin des éléments suivants :

- Un Raspberry Pi 3 B, Raspberry Pi 3 B + ou un Raspberry Pi 4 B
- Une carte micro SD d'au moins 8 Go pour contenir tous les logiciels fournis, avec une classe de vitesse recommandée de 10 (10 Mo/s)
- Un câble Ethernet pour connecter le Pi à votre réseau roboRIO
- Un micro câble d'alimentation USB pour se connecter au module régulateur de tension (VRM) sur votre robot. Il est recommandé d'utiliser la connexion VRM pour l'alimentation plutôt que de l'alimenter à partir de l'un des ports USB roboRIO pour une plus grande fiabilité
- Un ordinateur portable qui peut écrire la carte MicroSD, à l'aide d'un dongle USB (préféré) ou d'un adaptateur SD vers MicroSD fourni avec la plupart des cartes MicroSD.

Un dongle USB bon marché qui installera l'image FRC® dans la carte MicroSD.

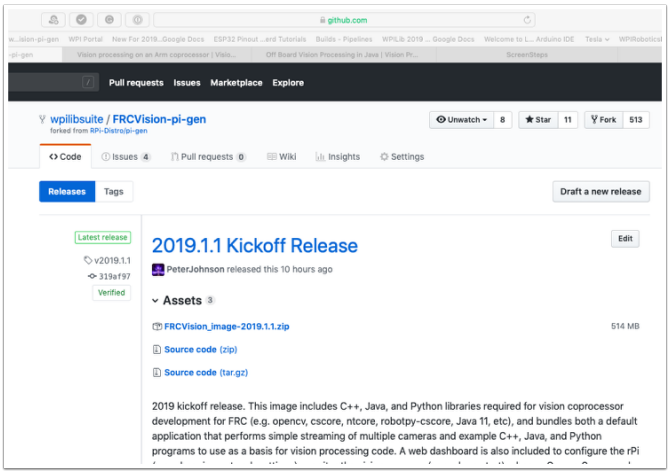
25.2.5 Installation de l'image RPi sur votre carte MicroSD

Obtenir l'image FRC Raspberry PI

L'image est stockée sur la page GitHub dédiée pour le WPILibPi dépôt.

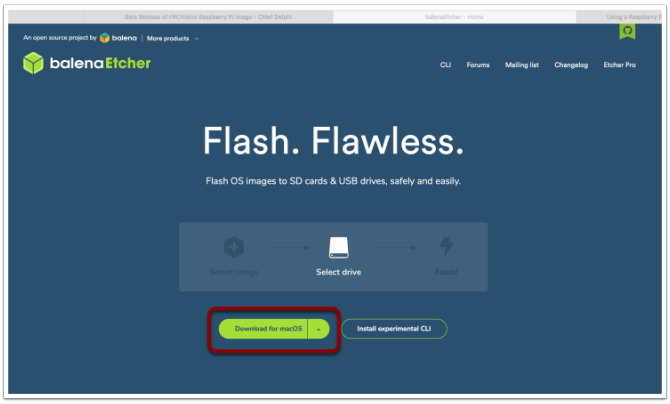
En plus des instructions sur cette page, consultez la documentation sur la page Web GitHub (ci-dessous).

L'image est assez grande, il faut donc une connexion Internet rapide lors du téléchargement. Utilisez toujours la version la plus récente en haut de la liste des versions.

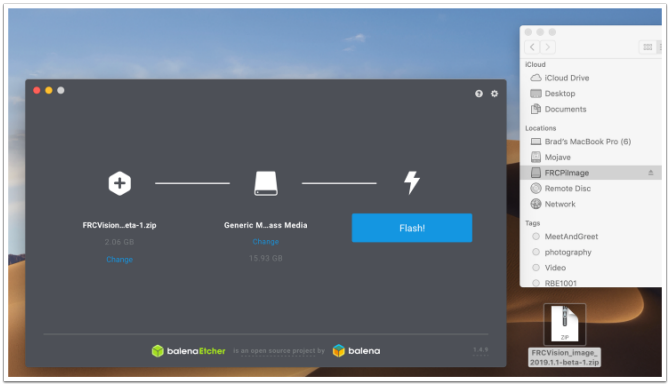


Copiez l'image sur votre carte MicroSD

Télécharger et installer Etcher pour créer une image de carte micro SD. La carte micro SD nécessite au moins 8 GB. Le produit micro SD to USB dongle fonctionne bien pour écrire sur des cartes micro SD.



Flashez la carte MicroSD avec l'image en utilisant Etcher en sélectionnant le fichier zip comme source, votre carte SD comme destination et cliquez sur « Flash ». Attendez-vous à ce que le processus prenne environ 3 minutes sur un ordinateur portable assez rapide.



Test du Raspberry PI

1. Mettez la carte micro SD dans un rPi 3 et mettez-la sous tension.
2. Connectez le rPi 3 ethernet à un LAN ou à un PC. Ouvrez un navigateur Web et connectez-vous à `http://wpilibpi.local/` pour ouvrir le dashboard Web. Au premier démarrage, le système de fichiers sera en écrtiture, mais les démarrages ultérieurs seront défaut pour lecture uniquement, il est donc nécessaire de cliquer sur le bouton « writable » pour faire des modifications.

Connexion au Raspberry PI

La plupart des tâches avec le rPi peuvent être effectuées à partir de l'interface de la console Web. Parfois, pour une utilisation avancée telle que le développement de programmes sur le rPi, il est nécessaire de se connecter. Pour se connecter, utilisez le mot de passe Raspberry PI par défaut :

```
Username: pi
 Password: raspberry
```

25.2.6 Le Raspberry PI

Console FRC

L'image FRC® pour le Raspberry PI comprend une console qui peut être affichée dans n'importe quel navigateur Web, ce qui permet de :

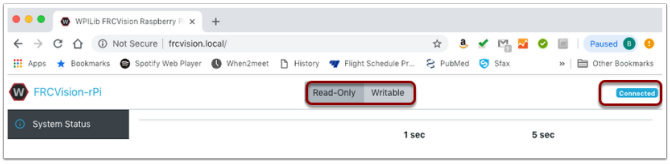
— Visualiser l'état du Raspberry PI
— Afficher l'état du processus d'arrière-plan exécutant la caméra
— Afficher ou modifier les paramètres réseau
— Regardez chaque caméra connectée au rPI et ajoutez des caméras supplémentaires
— Charger un nouveau programme de vision sur le rPI

Définition du rPI en lecture seule ou en écriture

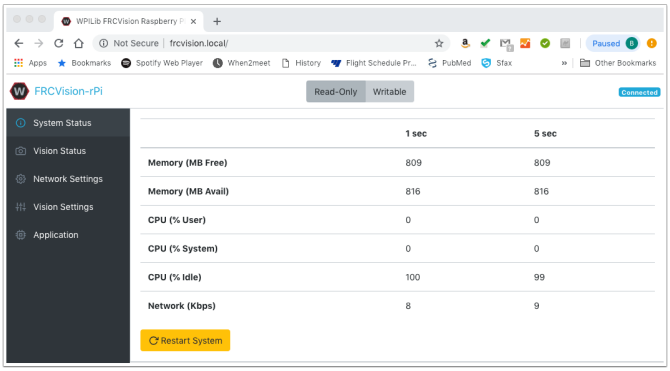
Le rPI est normalement défini en mode lecture-seulement (Read-Only), ce qui signifie que le système de fichiers ne peut pas être modifié. Cela garantit que si l'alimentation est interrompue sans d'abord arrêter le rPi, le système de fichiers n'est pas corrompu. Lorsque les paramètres sont modifiés (sections suivantes), les nouveaux paramètres ne peuvent pas être enregistrés lorsque le système de fichiers rPI est défini en lecture-seulement. Des boutons sont fournis pour permettre au système de fichiers de passer de lecture-seulement à écriture et inversement chaque fois que des modifications sont apportées. Si vous ne pouvez pas appuyer sur les autres boutons qui modifient les informations stockées sur le rPI, vérifiez l'état en lecture-seulement du système.

État de la connexion réseau au rPI

Il y a une icône dans le coin supérieur droit de la console qui indique si le rPi est actuellement connecté. Il passera de Connected à Disconnected s'il n'y a plus de connexion réseau au rPi.



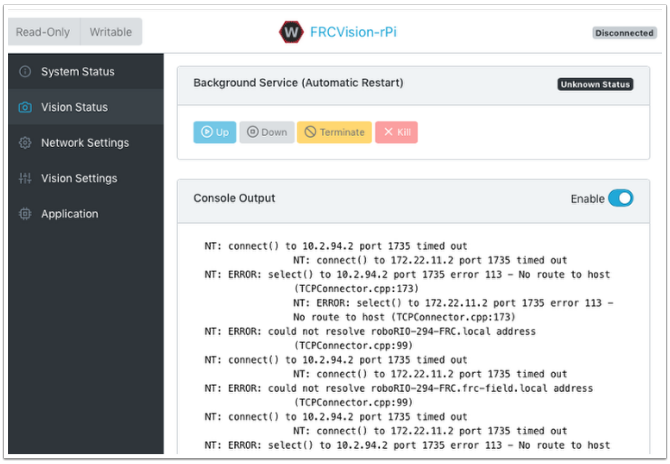
État du système



L'état du système indique à tout moment ce que fait le CPU sur le rPI. Il y a deux colonnes de valeurs d'état, soit une tabulée à chaque seconde et l'autre tabulée sur une durée de 5 secondes. Voici ce qui est affiché :

— Free and available RAM on the PI
— CPU usage for user processes and system processes as well as idle time
— Network bandwidth - which allows one to determine if the used camera bandwidth is exceeding the maximum bandwidth allowed in the robot rules for any year

État de la vision

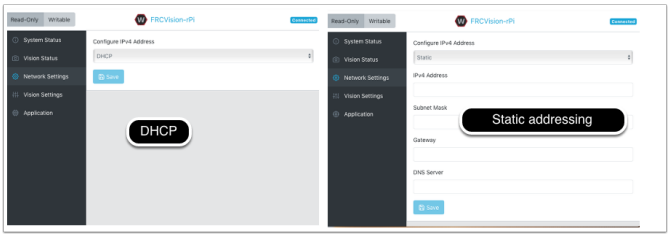


Permet de surveiller la tâche qui exécute le code de la caméra dans le rPI, soit l'un des programmes par défaut, soit votre propre programme en Java, C ++ ou Python. Vous pouvez également activer et afficher la sortie de la console pour voir les messages provenant du service de caméra d'arrière-plan. Dans ce cas, il y a un certain nombre de messages sur l'impossibilité de se connecter aux *NetworkTables* (NT : connect ()) car dans cet exemple, le rPI est simplement connecté à un ordinateur portable sans serveur NetworkTables en cours d'exécution (généralement le roboRIO.)

Paramètres réseau

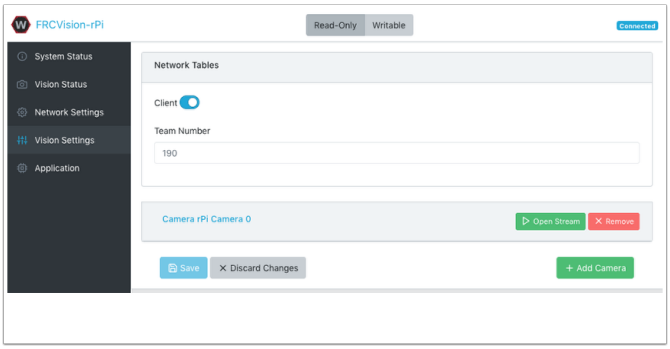
Les paramètres réseau rPI ont des options pour se connecter au PI :

- *DHCP* - the default name resolution usually used by the roboRIO. The default name is wpilibpi.local.
- Static - où une adresse IP fixe, un masque de réseau et des paramètres de routeur sont définis de façon explicite.
- DHCP with Static Fallback - le PI essaiera d'obtenir une adresse IP via DHCP, mais s'il ne trouve pas de serveur DHCP, il utilisera l'adresse IP statique et les paramètres fournis



L'image ci-dessus montre les paramètres pour DHCP et l'adressage IP statique. Le nom mDNS du rPi doit toujours fonctionner quelles que soient les options sélectionnées ci-dessus.

Paramètres pour la vision



The Vision Settings are to set the parameters for each camera and whether the rPI should be a NetworkTables client or server. There can only be one server on the network and the roboRIO is always a server. Therefore when connected to a roboRIO, the rPI should always be in client mode with the team number filled in. If testing on a desktop setup with no roboRIO or anything acting as a server then it should be set to Server (Client switch is off).

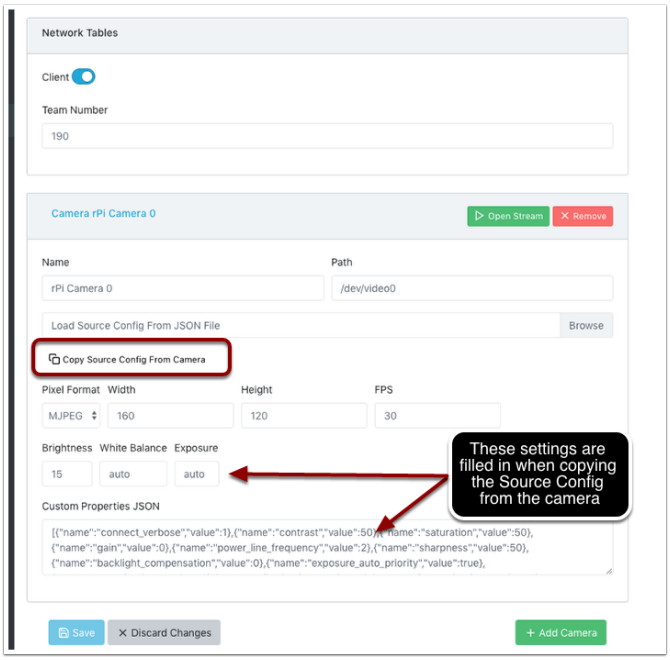
Pour afficher et manipuler tous les paramètres de la caméra, cliquez sur la caméra en question. Dans ce cas, la caméra est appelée « Camera rPi Camera 0 » et en cliquant sur le nom, le panorama vu par la caméra et ses paramètres associés sont affichés.

La prise de vue de la caméra réflète la manipulation de ses paramètres. Le bas de la page affiche tous les modes de caméra possibles (combinaisons de largeur, hauteur et le taux de rafraichissement des images (frame rate)) pris en charge par cette caméra.

Note : Si l'image de la caméra n'est pas visible sur l'écran *Open Stream*, vérifiez les modes vidéo pris en charge au bas de la page. Revenez ensuite aux paramètres "Vision Settings" et cliquez sur la caméra en question et vérifiez que les paramètres format pixel, la largeur (width), la hauteur (height) et la fréquence de rafraîchissement de l'image (FPS) sont répertoriés dans les modes vidéo pris en charge.

Obtention des paramètres actuels pour qu'ils persistent lors des redémarrages

Le rPi chargera tous les paramètres de la caméra au démarrage. La modification de la configuration de la caméra dans l'écran ci-dessus est temporaire. Pour que les valeurs persistent, cliquez sur le bouton « Load Source Config From Camera » et les paramètres actuels seront chargés dans les champs des paramètres de la caméra. Cliquez ensuite sur « Save » en bas de la page. Remarque : vous devez définir le système de fichiers accessible en écriture (Writeable) pour enregistrer les paramètres. *Le bouton Writeable est en haut de la page.*



Certaines valeurs de paramètres de caméra couramment utilisées sont indiquées dans les réglages (ci-dessus). Ces valeurs de luminosité, de balance des blancs et d'exposition sont chargées dans la caméra en premier, avant l'application du fichier utilisateur JSON. Donc, si un fichier utilisateur JSON contient aussi ces paramètres, ils viendront remplacer ceux définis auparavent.

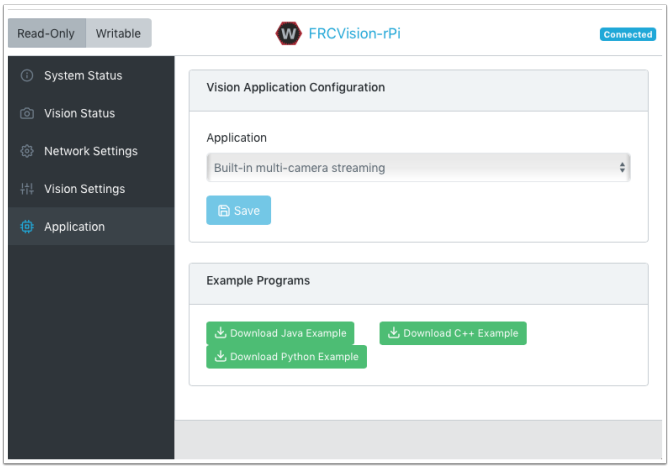
Application

L'onglet Application affiche l'application en cours d'exécution sur le rPi.

Implémentation des tâches pour la vision

Il existe un exemple de programme de vision utilisant OpenCV dans chacun des langages supportés, C++, Java ou Python. Chaque exemple de programme peut capturer et diffuser des vidéos à partir du rPi. De plus, les exemples ont quelques commandes OpenCV. Ils sont tous configurés pour être redéployé avec un code OpenCV amélioré par l'utilisateur et conforme à l'application du robot. L'onglet Application rPi prend en charge un certain nombre d'algorithmes de programmation :

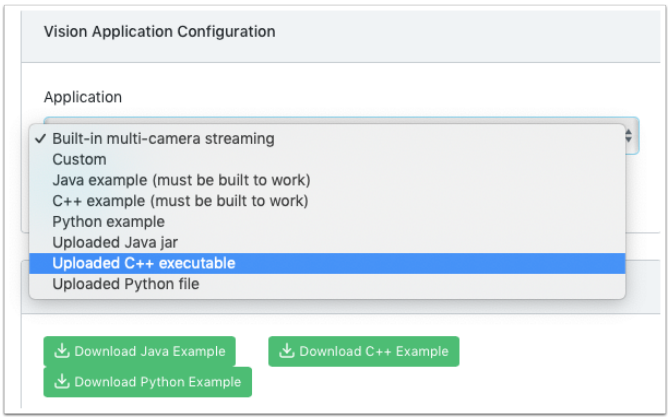
— Diffusez une ou plusieurs caméras à partir du rPi vers l'ordinateur du poste de conduite et les afficher à l'aide de ShuffleBoard
— Modifiez et créez l'un des exemples de programmes (un pour chaque langage : Java, C++ ou Python) sur le rPi en utilisant la chaîne d'outils fournie (éditeur, compilateur…)
— Téléchargez un exemple source de programme (Java, C++ ou Python) et modifiez-le. Ensuite, compilez-le en vous servant de votre ordinateur de développement. Ensuite, téléchargez le fichier objet de ce programme sur le rPi
— Concevez vos propres applications et scripts (en utilisant un ou plusieurs des exemples fournis comme référence)



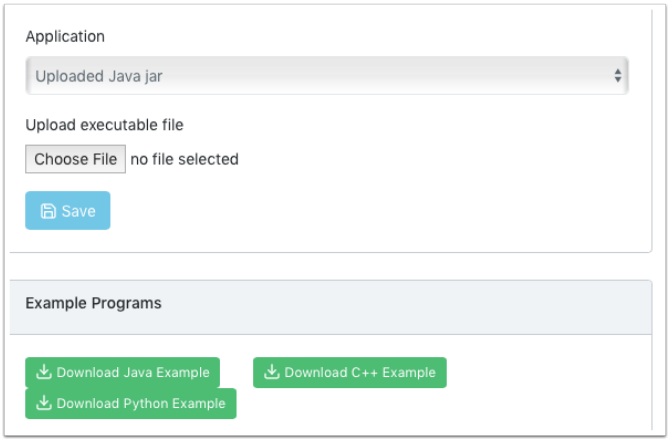
L'application en cours d'exécution peut être modifiée en sélectionnant l'un des choix dans le menu. Les choix sont :

— Built-in multi camera streaming which streams whatever cameras are plugged into the rPi. The camera configuration including number of cameras can be set on the « Vision Settings » tab.
— Custom Application qui ne télécharge rien sur le rPi et suppose que le programmeur souhaite avoir un code et un script personnalisés.
— Exemples de programmes préinstallés Java, C++ ou Python pouvant être modifiés dans votre propre application.
— Java, C++, or Python uploaded program. Java programs require a `.jar` file with the compiled program and C++ programs require an rPi executable to be uploaded to the rPI.

Lors du choix de l'une des options de téléchargement, un sélecteur de fichier est présenté dans lequel le fichier jar, exécutable ou Python peut être sélectionné et téléchargé sur le rPi. Dans l'image suivante, un fichier Java jar est choisi via le bouton « Choose File » et en cliquant sur le bouton « Save », le fichier sélectionné sera téléchargé.



Remarque : pour enregistrer un nouveau fichier sur le rPi, le système de fichiers doit être paramétrable en écriture à l'aide du bouton « Writable » en haut à gauche de la page Web. Après avoir enregistré le nouveau fichier, redéfinissez le système de fichiers sur « Read-Only » afin qu'il soit protégé contre les modifications accidentelles.



25.2.7 Utilisation de la classe CameraServer

Capture de trames à partir de CameraServer

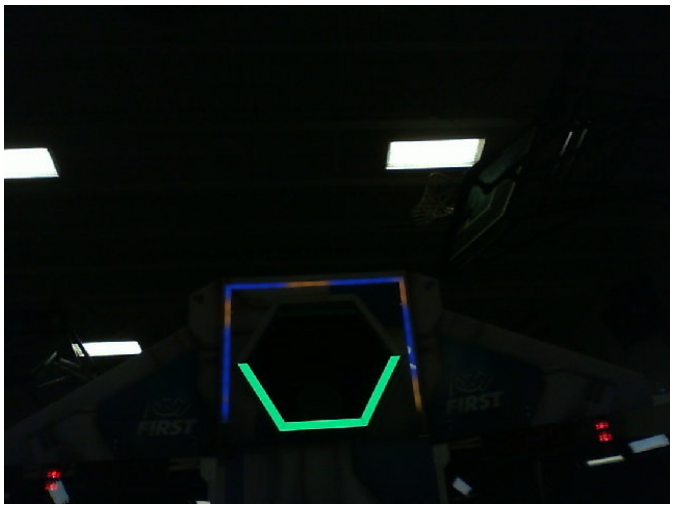
L'image WPILibPi est livrée avec toutes les librairies nécessaires pour faire votre propre système de traitement de la vision. Afin d'obtenir le cadre actuel de la caméra, vous pouvez utiliser la librairie CameraServer. Pour plus d'informations sur CameraServer, consultez *Lire et traiter la vidéo : classe CameraServer*.

PY

```
from cscore import CameraServer
import cv2
import numpy as np

CameraServer.enableLogging()

camera = CameraServer.startAutomaticCapture()
camera.setResolution(width, height)

sink = cs.getVideo()

while True:
   time, input_img = cvSink.grabFrame(input_img)

   if time == 0: # There is an error
      continue
```

Note : OpenCV comprends par défaut le format d'image comme **BGR** (Bleu,Vert, Rouge), pas **RGB** (Rouge, Vert, Bleu) pour des raisons historiques. Utilisez `cv2.cvtColor` si vous souhaitez le changer en RGB.

Vous trouverez ci-dessous un exemple d'image pouvant être récupérée à partir de CameraServer.



Envoi d'images à CameraServer

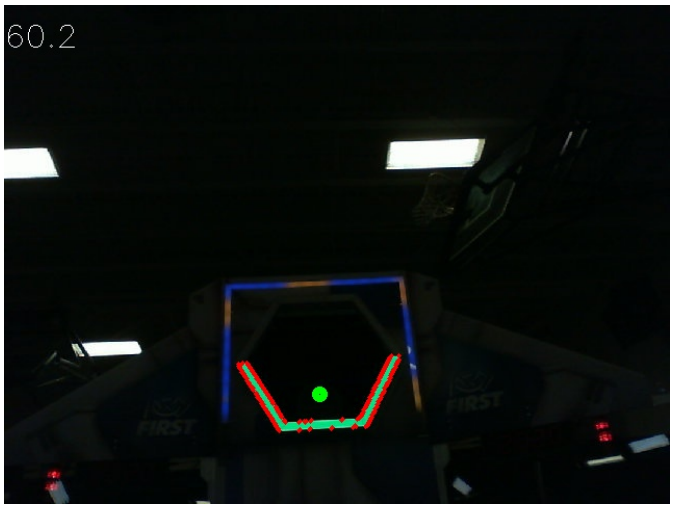
Dans certains cas, vous desirez envoyer des images vidéo traitées à CameraServer pour fins de débogage ou d'affichage dans une application de tableau de bord comme Shuffleboard.

PY

```
#
# CameraServer initialization code here
#

output = cs.putVideo("Name", width, height)

while True:
   time, input_img = cvSink.grabFrame(input_img)

   if time == 0: # There is an error
      output.notifyError(sink.getError())
      continue

   #
   # Insert processing code here
   #

   output.putFrame(processed_img)
```

Par exemple, le code de traitement peut décrire la cible en rouge et afficher les coins en jaune à des fins de débogage.

Vous trouverez ci-dessous un exemple d'image entièrement traitée qui serait renvoyée à CameraServer et affichée sur l'ordinateur du poste de pilotage.



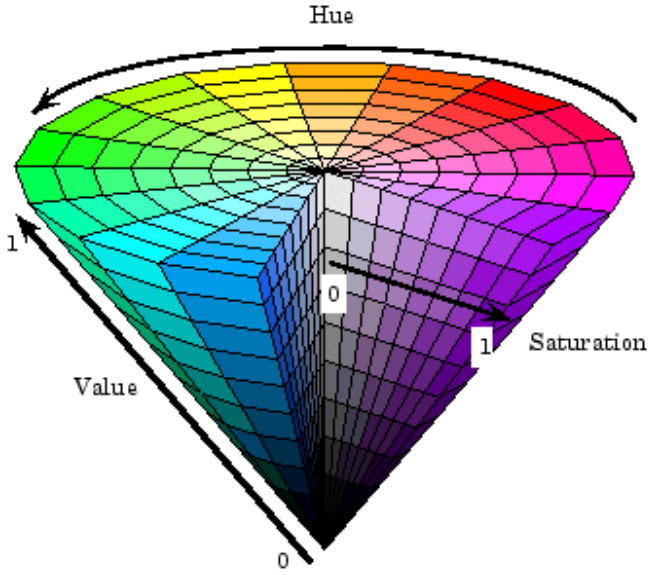
25.2.8 Trouver le seuil d'une image (Thresholding)

Afin de transformer une image colorée, telle que celle capturée par votre caméra, en une image binaire, avec la cible en « premier plan », nous devons seuiller l'image en utilisant la teinte, la saturation et la valeur de chaque pixel.

L'espace de couleur HSV (teinte, saturation et luminosité)

Il existe plusieurs façon de définir une même couleur : l'espace RGB (rouge, vert, bleu) et l'espace HSV. Ils ont chacun leur utilité selon le traitement désiré. Contrairement à l'espace RGB, l'espace HSV vous permet d'isoler le vert par exemple, peu importe son intensité sa luminosité.

— Teinte : mesure la tonalité d'une couleur dans le spectre lumineux.
— Saturation : mesure l'intensité de la couleur du pixel.
— Value : Measures the brightness of the pixel.



Vous pouvez utiliser OpenCV pour convertir une matrice d'images BGR (bleu, vert, rouge) en HSV. En OpenCV, l'ordre des couleurs primaires, communément appelée l'espace RGB, est différente (BGR).

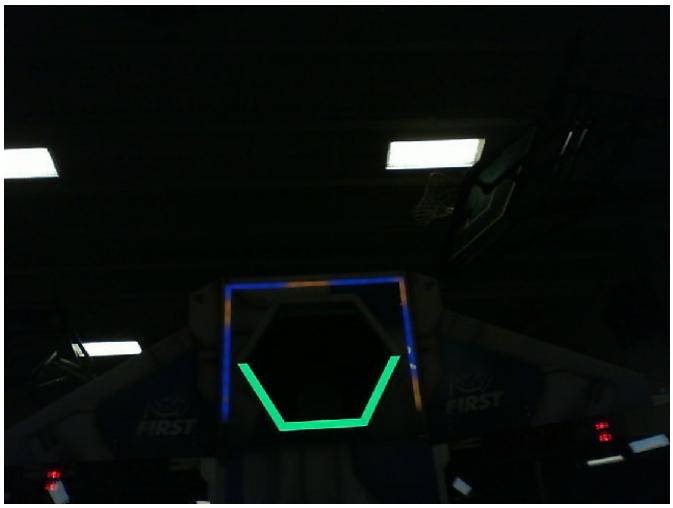
PY

```
hsv_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2HSV)
```

Note : La gamme de teintes d'OpenCV est de 1° à 180° au lieu des 1° à 360° habituels. Afin de convertir une valeur de teinte commune en OpenCV, divisez par 2.

Seuillage

Nous utiliserons cette image du terrain de jeu comme exemple pour l'ensemble du processus de traitement d'image.



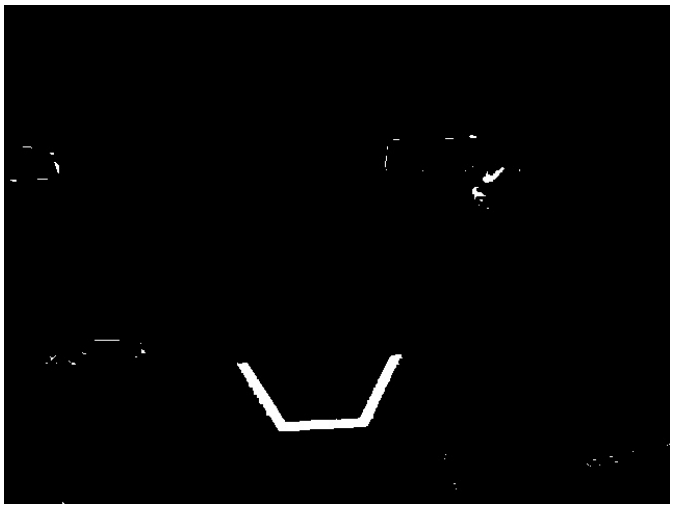
En limitant l'image à l'aide de HSV, vous pouvez séparer l'image en la cible de vision (premier plan) et les autres éléments que la caméra voit (arrière-plan). L'exemple de code suivant convertit une image HSV en image binaire par seuillage avec des valeurs HSV.

PY

```
binary_img = cv2.inRange(hsv_img, (min_hue, min_sat, min_val), (max_hue, max_sat, max_
↪val))
```

Note : Ces valeurs peuvent devoir être ajustées sur le terrain, car l'éclairage ambiant peut différer d'un site à l'autre. Il est recommandé d'autoriser la modification de ces valeurs via NetworkTables afin de faciliter les changements rapides de valeurs.

Après le seuillage, votre image devrait ressembler à ceci.



Comme vous pouvez le voir, le processus de seuillage peut ne pas être 100% efficace. Vous pouvez utiliser des opérations morphologiques pour amenuiser le bruit.

25.2.9 Opérations morphologiques

Parfois, après avoir traité votre image avec un seuil (Threshold), vous avez du bruit indésirable dans l'image binaire résultante. Les opérations morphologiques peuvent aider à supprimer ce bruit.

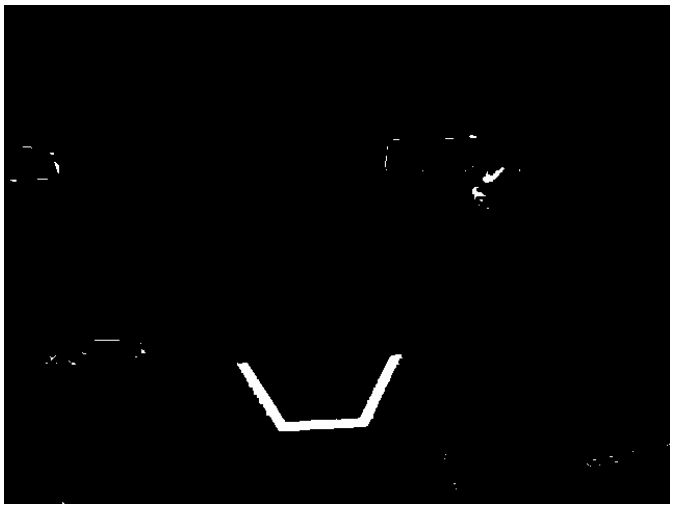
Noyau (Kernel)

Le noyau est une forme simple où l'origine est superposée à chaque pixel de valeur 1 de l'image binaire. OpenCV limite le noyau à une matrice NxN où N est un nombre impair. L'origine du noyau est le centre. Un noyau commun est :

$$kernel = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Différents noyaux peuvent affecter l'image différemment, tels que l'érosion ou la dilatation appliquée verticalement.

Pour référence, voici notre image binaire que nous avons créée :

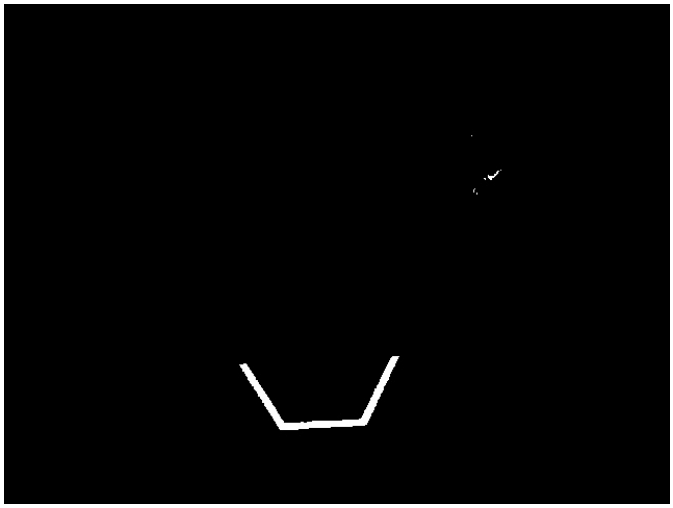


Érosion

L'érosion en vision par ordinateur est similaire à l'érosion sur le sol. Elle amenuise les rebords des objets de premier plan. Ce processus peut supprimer le bruit de l'arrière-plan.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.erode(binary_img, kernel, iterations = 1)
```



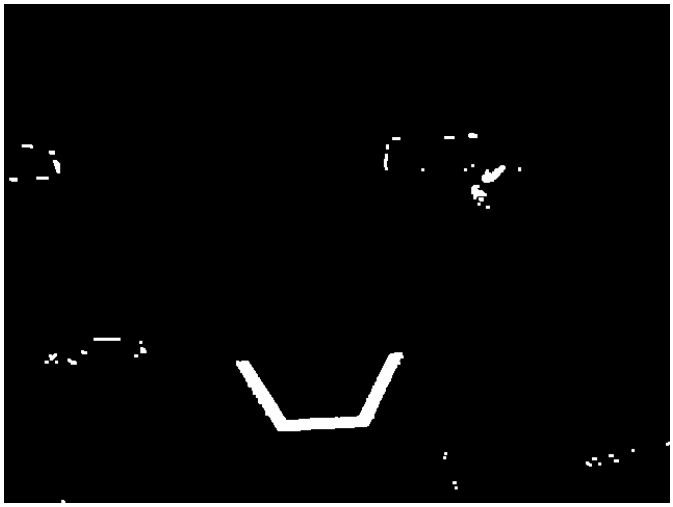
Pendant l'érosion, les pixels de l'image binaire qui ne sont pas superposés par les pixels du noyau sont supprimés.

Dilatation

La dilatation est le contraire de l'érosion. Au lieu d'amenuiser les rebords, on les renforcit. Ce processus peut contribuer à supprimer de petits trous à l'intérieur d'une plus grande région.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.dilate(binary_img, kernel, iterations = 1)
```



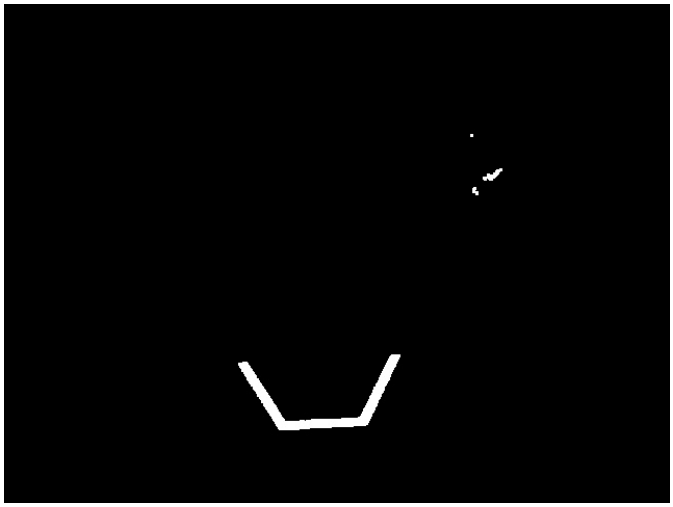
Pendant la dilatation, chaque pixel de chaque noyau superposé est inclus dans la dilatation.

Ouverture

L'ouverture est une érosion suivie d'une dilatation. Ce processus supprime le bruit sans affecter la forme des entités plus grandes.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel)
```



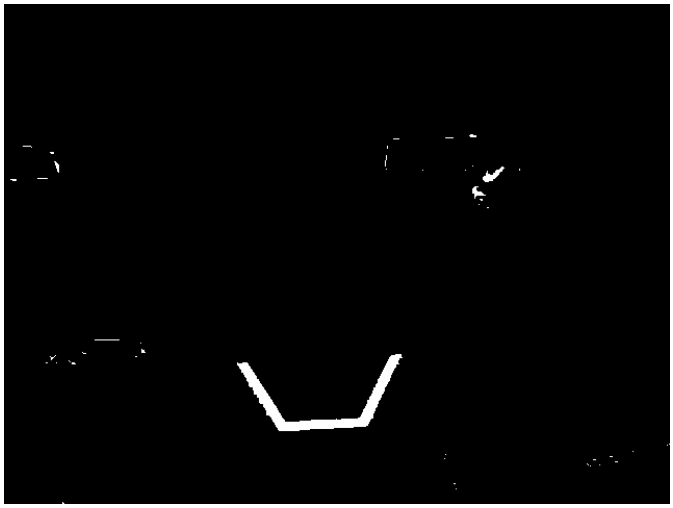
Note : Dans ce cas précis, il convient de faire quelques itérations d'ouverture afin de se débarrasser des pixels en haut à droite.

Fermeture

La fermeture est une dilatation suivie d'une érosion. Ce processus supprime les petits trous ou les ruptures sans affecter la forme des entités plus grandes.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, kernel)
```



25.2.10 Traitements des contours

Après avoir défini le seuil et éliminé le bruit avec des opérations morphologiques, vous êtes maintenant prêt à utiliser la méthode `findContours` d'OpenCV. Cette méthode vous permet de générer des contours en fonction de votre image binaire.

Recherche et filtrage des contours

PY

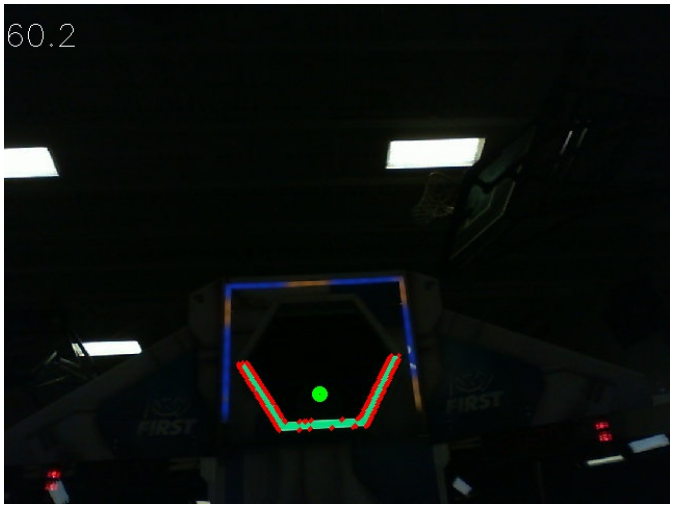
```
_, contours, _ = cv2.findContours(binary_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
↪SIMPLE)
```

In cases where there is only one vision target, you can just take the largest contour and assume that is the target you are looking for. When there is more than one vision target, you can use size, shape, fullness, and other properties to filter unwanted contours out.

PY

```
if len(contours) > 0:
   largest = contours[0]
   for contour in contours:
      if cv2.contourArea(contour) > cv2.contourArea(largest):
         largest = contour

   #
   # Contour processing code
   #
```

Si vous dessinez le contour que vous venez de trouver, il devrait ressembler à ceci :



Extraire des informations des contours

Maintenant que vous avez trouvé le (s) contour (s) que vous voulez, vous voulez maintenant obtenir des informations à ce sujet, telles que le centre, les coins et la rotation.

Centre

PY

```
rect = cv2.minAreaRect(contour)
center, _, _ = rect
center_x, center_y = center
```

Coins

PY

```
corners = cv2.convexHull(contour)
corners = cv2.approxPolyDP(corners, 0.1 * cv2.arcLength(contour), True)
```

Rotation

PY

```
_, _, rotation = cv2.fitEllipse(contour)
```

Pour plus d'informations sur la façon dont vous pouvez utiliser ces valeurs, voir : *Mesures*

Publication sur NetworkTables

Vous pouvez utiliser NetworkTables pour envoyer ces propriétés au Driver Station et au RoboRIO. Un traitement supplémentaire peut être effectué sur le Raspberry Pi ou le RoboRIO lui-même.

PY

```
import ntcore

nt = ntcore.NetworkTableInstance.getDefault().getTable('vision')

#
# Initialization code here
#

while True:

   #
   # Image processing code here
   #

   nt.putNumber('center_x', center_x)
   nt.putNumber('center_y', center_y)
```

25.2.11 Exemple basique de Vision

Ceci est un exemple basique d'une configuration de vision qui affiche l'emplacement de la cible dans le système de coordonnées de visée décrit *ici* dans les NetworkTables et utilise CameraServer pour afficher un rectangle englobant le contour détecté. Cet exemple affiche le taux de rafraîchissement d'images (Frame rate) envoyé à CameraServer.

PY

```
from cscore import CameraServer
import ntcore

import cv2
import json
import numpy as np
import time


def main():
   with open('/boot/frc.json') as f:
      config = json.load(f)
   camera = config['cameras'][0]

   width = camera['width']
   height = camera['height']
```

(suite sur la page suivante)

(suite de la page précédente)

```
    nt = ntcore.NetworkTableInstance.getDefault()

    CameraServer.startAutomaticCapture()

    input_stream = CameraServer.getVideo()
    output_stream = CameraServer.putVideo('Processed', width, height)

    # Table for vision output information
    vision_nt = nt.getTable('Vision')

    # Allocating new images is very expensive, always try to preallocate
    img = np.zeros(shape=(240, 320, 3), dtype=np.uint8)

    # Wait for NetworkTables to start
    time.sleep(0.5)

    while True:
        start_time = time.time()

        frame_time, input_img = input_stream.grabFrame(img)
        output_img = np.copy(input_img)

        # Notify output of error and skip iteration
        if frame_time == 0:
            output_stream.notifyError(input_stream.getError())
            continue

        # Convert to HSV and threshold image
        hsv_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2HSV)
        binary_img = cv2.inRange(hsv_img, (0, 0, 100), (85, 255, 255))

        _, contour_list = cv2.findContours(binary_img, mode=cv2.RETR_EXTERNAL,␣
↪method=cv2.CHAIN_APPROX_SIMPLE)

        x_list = []
        y_list = []

        for contour in contour_list:

            # Ignore small contours that could be because of noise/bad thresholding
            if cv2.contourArea(contour) < 15:
                continue

            cv2.drawContours(output_img, contour, -1, color=(255, 255, 255),␣
↪thickness=-1)

            rect = cv2.minAreaRect(contour)
            center, size, angle = rect
            center = tuple([int(dim) for dim in center])  # Convert to int so we can␣
↪draw

            # Draw rectangle and circle
            cv2.drawContours(output_img, [cv2.boxPoints(rect).astype(int)], -1,␣
↪color=(0, 0, 255), thickness=2)
            cv2.circle(output_img, center=center, radius=3, color=(0, 0, 255),␣
↪thickness=-1)
```

(suite sur la page suivante)

(suite de la page précédente)

```
            x_list.append((center[0] - width / 2) / (width / 2))
            x_list.append((center[1] - width / 2) / (width / 2))

      vision_nt.putNumberArray('target_x', x_list)
      vision_nt.putNumberArray('target_y', y_list)

      processing_time = time.time() - start_time
      fps = 1 / processing_time
      cv2.putText(output_img, str(round(fps, 1)), (0, 40), cv2.FONT_HERSHEY_SIMPLEX,␣
↪1, (255, 255, 255))
      output_stream.putFrame(output_img)


main()
```

25.3 AprilTag Introduction

25.3.1 Que sont les AprilTags ?

Les AprilTags sont un système d'étiquettes visuelles développé par des chercheurs de l'Université du Michigan visant à fournir une localisation d'emplacement à basse consommation de ressources et haute précision adaptée à plusieurs applications.

Des informations additionnelles à propos du système d'étiquettes et de ses créateurs se trouvent sur leur site Ce document essaie de résumer le contenu pour les besoins de la Compétition de robotique FIRST.

Application en FRC

Dans le contexte de la FRC, les AprilTags sont utiles pour aider votre robot à savoir où il se trouve sur le terrain, afin qu'Il puisse s'aligner avec les positions de buts.

Les AprilTags sont développées depuis 2011 et se sont raffinées avec les années, afin d'augmenter leur robustesse et la vitesse de détection.

Débutant en 2023, FIRST fournit plusieurs étiquettes <https://www.firstinspires.org/robotics/frc/blog/2022-2023-approved-devices-rules-preview-and-vision-target-update>`__ dispersées sur le terrain, connues sous le nom de *pose*.

En 2024, la famille d'étiquettes a été mise à jour vers la famille 36h11 <https://www.firstinspires.org/robotics/frc/blog/2023-technology-updates-past-present-future-and-beyond>.

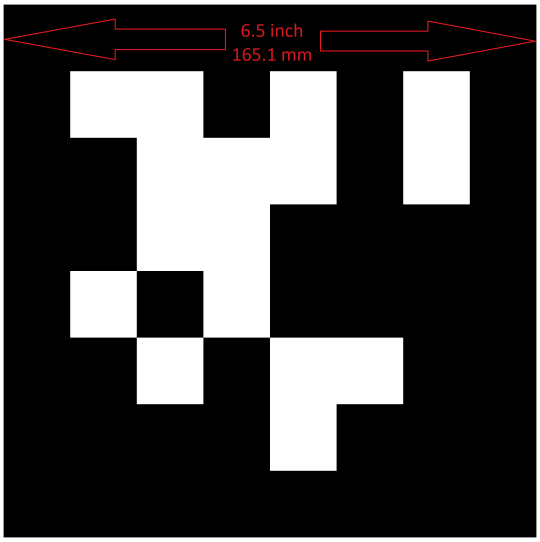
L'implémentation de la bibliothèque AprilTag définit les standards sur la manières que les ensembles d'étiquettes devraient être conçues. Certaines des familles possibles sont décrites ici.

FIRST a choisi la famille 36h11 pour 2024. Cette famille d'étiquettes est composée d'une grille de pixels 6x6, représentant chacun un bit d'information. Une bordure supplémentaire en noir et blanc doit être présente autour des bits.

Bien qu'il y ait $2^{36} = 68,719,476,736$ étiquettes théoriquement possibles, seules 587 sont réellement utilisées. Celles-ci sont choisies pour :

1. Être robustes contre le basculement de bit (ex : problèmes où le bit a sa couleur identifiée incorrectement).
2. Ne pas contenir de motifs géométriques « simples » qui pourraient se trouver sur des choses qui ne sont pas des cibles (ex : carrés, lignes, etc...).
3. S'assurer que le motif géométrique est assez asymétrique que vous pouvez toujours savoir quel côté pointe vers le haut.

Toutes les étiquettes seront imprimées de sorte que le « corps » principal de l'étiquette mesure 6,5 pouces de longueur.



Pour utilisation à la maison, les fichiers d'étiquette peuvent êtres imprimés et placés autour de votre aire de pratique. Fixez les sur un support rigide afin que les étiquettes restent plate, puisque l'algorithme de vision assume que l'étiquette est plate.

Support Logiciel

La principale bibliothèque pour le code source qui détecte et décode les AprilTags est située ici <https://github.com/wpilibsuite/apriltag/tree/main>.

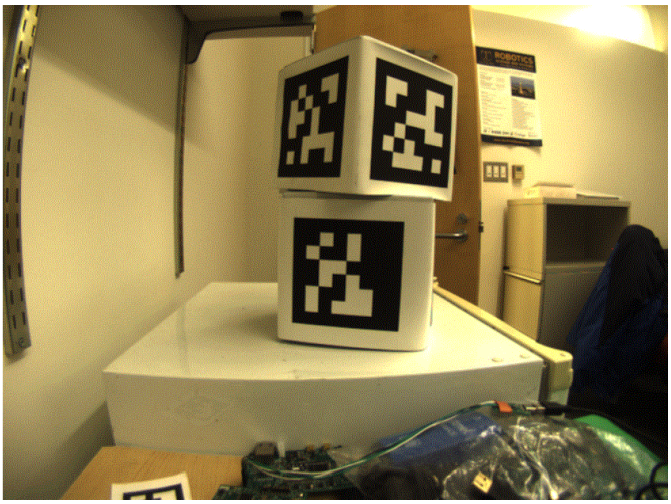
WPIlib a ramifié la bibliothèque afin d'ajouter des nouvelles fonctionnalités pour la FRC. Entre autres :

1. Compiler le code source pour des cibles FRC courantes, incluant le roboRIO et Raspberry Pi.
2. Ajouter un support Java Native Interface (JNI) pour permettre l'invocation de ses fonctionnalités depuis Java
3. Support de publication Gradle et Maven

Technique de calcul

Bien que la plupart des équipes FRC ne devraient pas avoir à implémenter leur propre code pour identifier les AprilTags dans une image de caméra, il est utile de connaître les bases du fonctionnement des bibliothèques sous-jacentes.

Image d'origine



Une image d'une caméra est simplement une liste de valeurs correspondant à la couleur et la luminosité de chaque pixel.

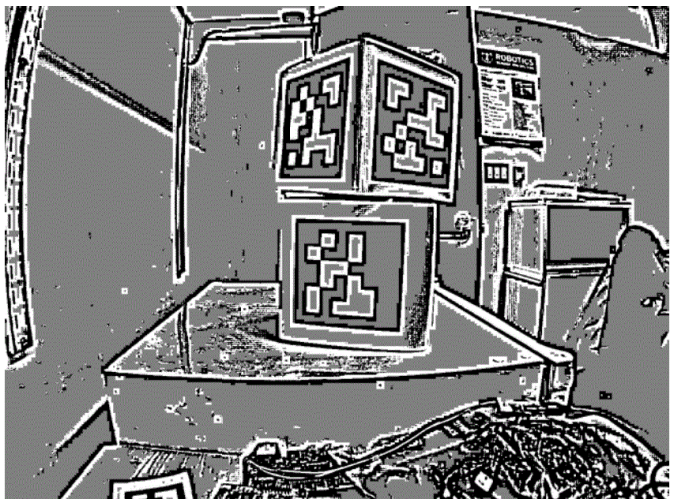
Retirer les couleurs

La première étape est de convertir l'image en tons de gris (luminosité seulement). L'information des couleurs n'est pas nécessaire pour détecter les étiquettes en noir et blanc.

Décimer

La prochaine étape est de convertir l'image vers une résolution plus basse. Travailler avec moins de pixels aide l'algorithme à travailler plus vite. L'image en pleine résolution sera utilisée plus tard pour raffiner les estimations préliminaires.

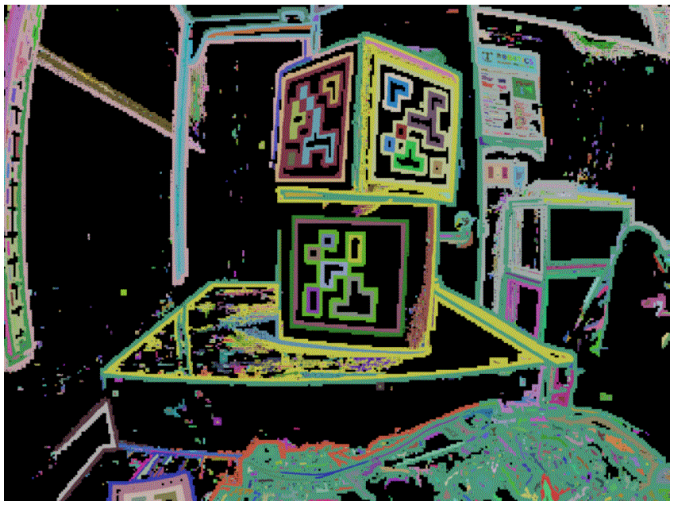
Seuil adaptatif



Un algorithme de seuil adaptatif est exécuté afin de classifier chacun des pixels en « définitivement clair », « définitivement foncé » ou « incertain ».

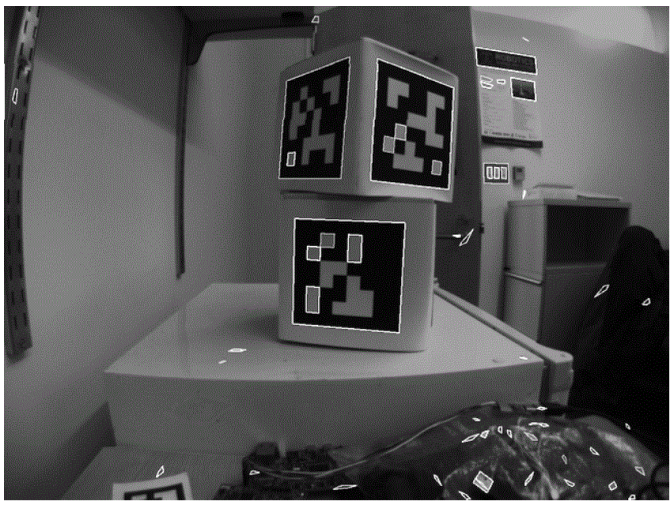
Le seuil est calculé en regardant la luminosité du pixel comparé à celle du voisinage des pixels.

Segmentation



Ensuite, les pixels connus sont regroupés ensemble. Tout les groupes trop petits pour être une partie significative d'une étiquette sont supprimés.

Détection des quadrilatères



Un algorithme pour associer un quadrilatère à chaque groupe est maintenant exécuté :

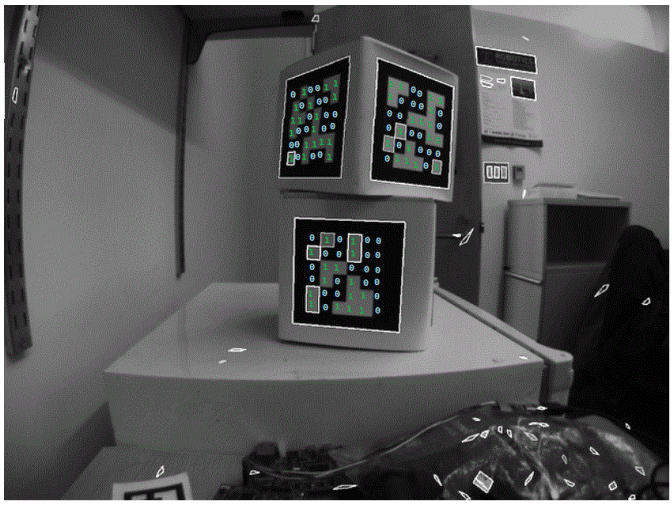
— Identifier les « coins » probables par les pixels qui sont des valeurs aberrantes dans les deux dimensions.
— Itérer au travers de toutes les combinaison possibles de coins, tout en évaluant l'appartenance à chaque fois.
— Choisir le quadrilatère avec le meilleur ajustement

Compte tenu de l'ensemble de tout les quadrilatères, on identifie un sous-ensemble de quadrilatères qui est probablement une étiquette.

Un seul grand quadrilatère extérieur avec plusieurs quadrilatères intérieurs est sûrement un bon candidat.

Si tout s'est bien passé jusqu'ici, il nous reste une région de pixels à quatre côtés qui est sûrement une étiquette valide.

Décoder l'identifiant



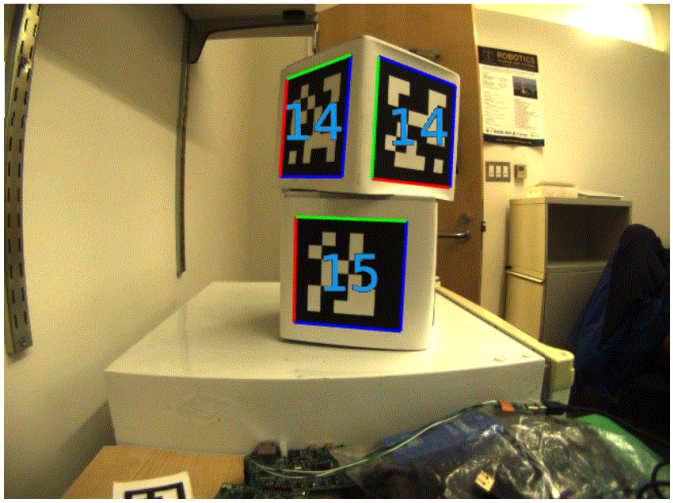
Maintenant que nous avons ai moins une région de pixels que nous croyons être un AprilTag valide, nous devons identifier l'étiquette que nous regardons. Cela est fait en « décodant » le

motif de carrés clairs et foncés à l'intérieur.

- Calculer les coordonnées attendues des pixels intérieurs où le centre de chaque bit devrait être
- Marquer chaque emplacement en tant que « 1 » ou « 0 » en comparant l'intensité des pixels à un seuil
- Trouver l'identifiant d'étiquette qui correspond de près avec ce qui a été vu dans l'image, en permettant une erreur d'un ou deux bits.

Il est possible qu'il n'y ait aucun identifiant d'étiquette valide qui correspond à l'étiquette suspectée. Dans ce cas, le processus de décodage s'arrête.

Adapter le quadrilatère externe



Maintenant que nous avons un identifiant d'étiquette pour la région de pixels, nous devons faire quelque chose d'utile avec.

Pour la plupart des applications en FRC, nous nous soucions de savoir la localisation précise des coins de l'étiquette, ou son centre. Dans les deux cas, nous nous attendons à ce que l'opération de diminution de la résolution que nous avons effectué au débit ait déformé l'image et nous voulons renverser ces effets.

L'algorithme pour faire cela est :

- Utiliser la position de l'étiquette détectée pour définir une région d'intérêt dans l'image de résolution originale
- Calculer la pente à des points prédéfinis dans la région d'intérêt pour détecter où l'image transitionne nettement entre le blanc et le noir
- Utiliser ces mesures de pente pour rapidement réajuster un quadrilatère extérieur à la pleine résolution
- Utiliser la géométrie pour calculer le centre exact du quadrilatère réajusté

Notez que cette étape est optionnelle et peut être ignorée pour un calcul plus rapide de l'image. Cependant, l'ignorer peut introduire des erreurs significative dans le comportement de votre robot, dépendamment de la manière que vous utilisez les sorties des étiquettes.

Utilisation

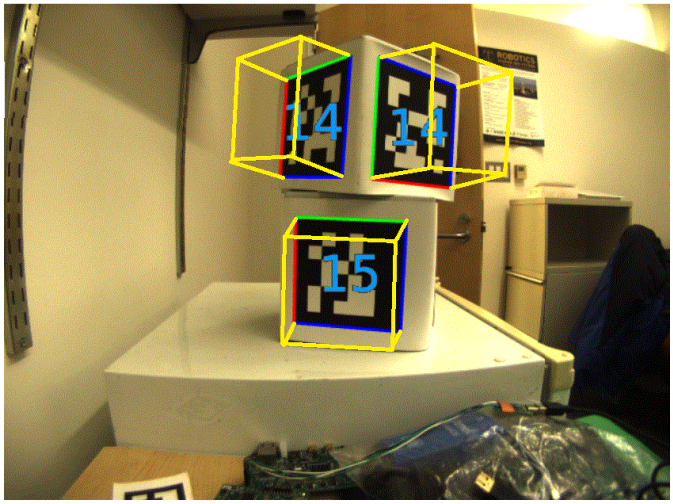
Alignement 2D

Une stratégie simple pour utiliser les cibles est de bouger le robot jusqu'à ce que la cible soit centrée dans l'image. Supposant que le terrain et le robot sont construits d'une manière que les pièces de jeu, la position du but, la cible de vision et la caméra sont alignés, cette méthode devrait être une méthode simple pour aligner le robot à la position de pointage.

En utilisant une caméra, identifiez le « centroïde » des AprilTags en vue. Si l'identifiant d'étiquette est correct, appliquez des commandes de conduite pour tourner le robot vers la gauche ou la droite jusqu'à ce que l'étiquette soit centrée dans l'image.

Cette méthode ne requiert pas de calibration ou de performer l'étape d'homographie.

Alignement 3D



A more advanced usage of AprilTags is to use their corner locations to help perform *pose estimation*.

Des AprilTags sont recherchés dans chaque image en utilisant l'algorithme décrit sur cette page. En utilisant des inférences sur la manière que la lentille de la caméra déforme le monde 3d en un tableau de pixels 2d dans la caméra, une estimation de la position de la caméra en relation avec l'étiquette est calculée. Une bonne calibration de la caméra est requise pour que les estimations sur le comportement de sa lentille soient précises.

L'identifiant de l'étiquette est aussi décodé depuis l'image. Compte tenu de chaque identifiant d'étiquette, la position de l'étiquette sur le terrain peut être recherchée.

Sachant la position de l'étiquette sur le terrain ainsi que la position de la caméra en relation avec l'étiquette, les classes de géométrie 3D peuvent être utilisées pour estimer la position de la caméra sur le terrain.

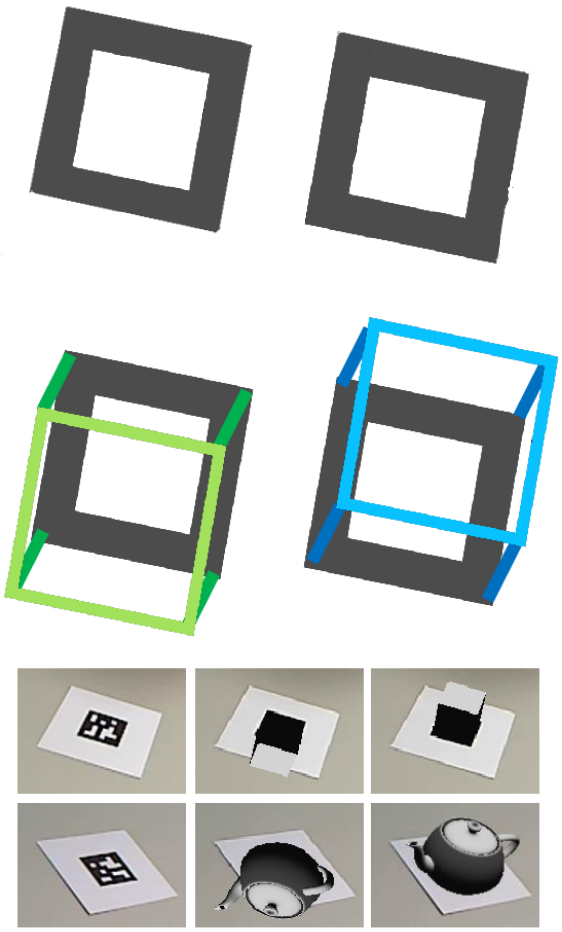
Si la positon de la caméra sur le robot est connue, la position du robot sur le terrain peut aussi être estimée.

Ces estimations peuvent être incorporées dans les classes d'estimation de pose de la WPILib.

Ambiguïté 2D vers 3D

Le processus de traduire les quatre coins connus de la cible dans l'image (en deux dimension) en une position réelle relative à la caméra (en trois dimensions) est intrinsèquement ambiguë. Cela est pour dire qu'il y a plusieurs positions réelles qui résultent en les coins cibles étant aux mêmes endroits dans l'image de la caméra.

Les humains peuvent souvent utiliser l'éclairage ou les indices en arrière-plan pour comprendre comment les objets sont orientés dans l'espace. Cependant, les ordinateurs n'ont pas ce bienfait. Ils peuvent être confus par des cibles similaires :



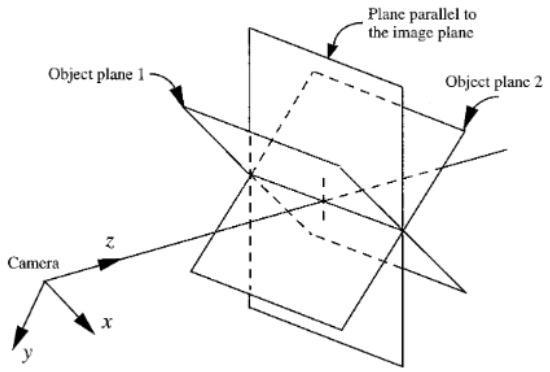


FIG. 4. Two object poses giving the same image under the SOP approximation.

Résoudre quelle position est « correcte » peut être fait de quelques manières différentes :

1. Use your *odometry* history from all sensors to pick the pose closest to where you expect the robot to be.
2. Rejeter les poses très improbables (ex : hors du périmètre du terrain, ou en l'air)
3. Ignorer les estimations de poses qui sont très proche l'une de l'autre (et difficile à différencier)
4. Utiliser plusieurs caméras pour regarder la même cible afin qu'au moins une caméra puisse générer une assez bonne estimation
5. Regarder plusieurs cibles à la fois, en utilisant chacune pour générer plusieurs estimations de pose. Rejeter les estimations externes, en utilisant celles qui sont regroupées fermement ensemble.

Paramètres ajustables

`Decimation Factor` modifie à quel point l'image est dé-échantillonnée avant le traitement. L'augmenter accroîtra la vitesse de détection, au coût de ne pas voir les étiquettes qui sont éloignées.

`Blur` applique un lissage à l'image en entrée afin de réduire le bruit, ce qui accroît la vitesse de correspondance entre les pixels et les quadrilatères, au coût de la précision. Pour la plupart des caméras, ceci peut être laissé à zéro.

`Threads` changent le nombre de processus parallèles que l'algorithme utilise afin de traiter l'image. Certaines étapes peuvent êtres accélérées en permettant le multitraitement. En général, vous voudrez ce paramètre à peu près égal au nombre de cœurs physiques dans votre processeur, moins le nombre de cœurs qui seront utilisés dans d'autres tâches de traitement.

De l'information détaillée à propos des paramètres modifiables se retrouve ici.

Pour en savoir plus

Les trois versions majeures d'AprilTags sont décrites dans trois papiers académiques. Il est recommandé de les lire en ordre, puisque chacun ajoute au précédent :

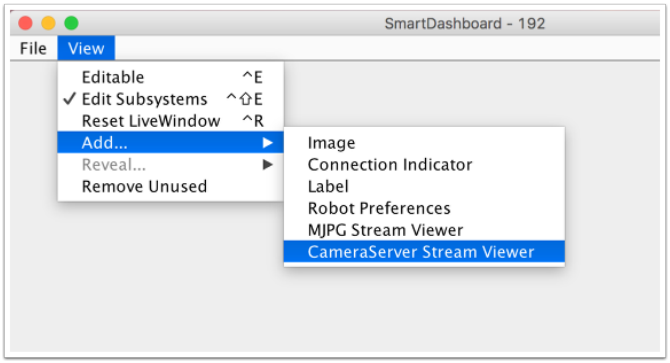
- AprilTags v1
- AprilTags v2
- AprilTags v3
- Pose Ambiguity

25.4 Vision avec le RoboRIO

25.4.1 Utilisation de CameraServer sur le roboRIO

Programme CameraServer élémentaire

Le programme suivant démarre la capture automatique d'une caméra USB comme la Microsoft LifeCam connectée au roboRIO. Dans ce mode, la caméra capturera des images et les enverra au tableau de bord. Pour visualiser les images, créez un gadget-widget « CameraServer Stream Viewer » à l'aide du menu « View », puis « Add » -ajouter- dans le tableau de bord. Les images ne sont pas traitées et sont simplement transmises de la caméra au tableau de bord.



JAVA

```
7  import edu.wpi.first.cameraserver.CameraServer;
8  import edu.wpi.first.wpilibj.TimedRobot;
9
10 /**
11  * Uses the CameraServer class to automatically capture video from a USB webcam and␣
   ↪send it to the
12  * FRC dashboard without doing any vision processing. This is the easiest way to get␣
   ↪camera images
13  * to the dashboard. Just add this to the robotInit() method in your program.
14  */
15 public class Robot extends TimedRobot {
16   @Override
17   public void robotInit() {
18     CameraServer.startAutomaticCapture();
```

(suite sur la page suivante)

(suite de la page précédente)

```
  }
}
```

C++

```
#include <cameraserver/CameraServer.h>
#include <frc/TimedRobot.h>
class Robot : public frc::TimedRobot {
 public:
  void RobotInit() override {
    frc::CameraServer::StartAutomaticCapture();
  }
};

#ifndef RUNNING_FRC_TESTS
int main() {
  return frc::StartRobot<Robot>();
}
```

PYTHON

```
import wpilib
from wpilib.cameraserver import CameraServer


class MyRobot(wpilib.TimedRobot):
    """
    Uses the CameraServer class to automatically capture video from a USB webcam and send it to the
    FRC dashboard without doing any vision processing. This is the easiest way to get camera images
    to the dashboard. Just add this to the robotInit() method in your program.
    """
```

Programme avancé de serveur de caméras

Dans l'exemple suivant, un thread créé dans robotInit() obtient l'instance de Camera Server. Chaque image de la vidéo est traitée individuellement, dans ce cas dessinant un rectangle sur l'image en utilisant la méthode OpenCV `rectangle()`. Les images résultantes sont ensuite transmises au flux de sortie et envoyées au tableau de bord. Vous pouvez remplacer l'opération `rectangle` par n'importe quel code de traitement d'image nécessaire à votre application. Vous pouvez même annoter l'image à l'aide des méthodes OpenCV pour écrire des informations de ciblage sur l'image envoyée au tableau de bord.

Java

```
import edu.wpi.first.cameraserver.CameraServer;
import edu.wpi.first.cscore.CvSink;
import edu.wpi.first.cscore.CvSource;
import edu.wpi.first.cscore.UsbCamera;
import edu.wpi.first.wpilibj.TimedRobot;
import org.opencv.core.Mat;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;

/**
 * This is a demo program showing the use of OpenCV to do vision processing. The image is acquired
 * from the USB camera, then a rectangle is put on the image and sent to the dashboard. OpenCV has
 * many methods for different types of processing.
 */
public class Robot extends TimedRobot {
  Thread m_visionThread;

  @Override
  public void robotInit() {
    m_visionThread =
        new Thread(
            () -> {
              // Get the UsbCamera from CameraServer
              UsbCamera camera = CameraServer.startAutomaticCapture();
              // Set the resolution
              camera.setResolution(640, 480);

              // Get a CvSink. This will capture Mats from the camera
              CvSink cvSink = CameraServer.getVideo();
              // Setup a CvSource. This will send images back to the Dashboard
              CvSource outputStream = CameraServer.putVideo("Rectangle", 640, 480);

              // Mats are very memory expensive. Lets reuse this Mat.
              Mat mat = new Mat();

              // This cannot be 'true'. The program will never exit if it is. This
              // lets the robot stop this thread when restarting robot code or
              // deploying.
              while (!Thread.interrupted()) {
                // Tell the CvSink to grab a frame from the camera and put it
                // in the source mat.  If there is an error notify the output.
                if (cvSink.grabFrame(mat) == 0) {
                  // Send the output the error.
                  outputStream.notifyError(cvSink.getError());
                  // skip the rest of the current iteration
                  continue;
                }
                // Put a rectangle on the image
                Imgproc.rectangle(
                    mat, new Point(100, 100), new Point(400, 400), new Scalar(255, 255, 255), 5);
                // Give the output stream a new image to display
```

(suite sur la page suivante)

(suite de la page précédente)

```
                outputStream.putFrame(mat);
              }
            });
    m_visionThread.setDaemon(true);
    m_visionThread.start();
  }
}
```

c++

```
#include <cstdio>
#include <thread>

#include <cameraserver/CameraServer.h>
#include <frc/TimedRobot.h>
#include <opencv2/core/core.hpp>
#include <opencv2/core/types.hpp>
#include <opencv2/imgproc/imgproc.hpp>

/**
 * This is a demo program showing the use of OpenCV to do vision processing. The
 * image is acquired from the USB camera, then a rectangle is put on the image
 * and sent to the dashboard. OpenCV has many methods for different types of
 * processing.
 */
class Robot : public frc::TimedRobot {
 private:
  static void VisionThread() {
    // Get the USB camera from CameraServer
    cs::UsbCamera camera = frc::CameraServer::StartAutomaticCapture();
    // Set the resolution
    camera.SetResolution(640, 480);

    // Get a CvSink. This will capture Mats from the Camera
    cs::CvSink cvSink = frc::CameraServer::GetVideo();
    // Setup a CvSource. This will send images back to the Dashboard
    cs::CvSource outputStream =
        frc::CameraServer::PutVideo("Rectangle", 640, 480);

    // Mats are very memory expensive. Lets reuse this Mat.
    cv::Mat mat;

    while (true) {
      // Tell the CvSink to grab a frame from the camera and
      // put it
      // in the source mat.  If there is an error notify the
      // output.
      if (cvSink.GrabFrame(mat) == 0) {
        // Send the output the error.
        outputStream.NotifyError(cvSink.GetError());
        // skip the rest of the current iteration
        continue;
      }
      // Put a rectangle on the image
```

(suite sur la page suivante)

(suite de la page précédente)

```
      rectangle(mat, cv::Point(100, 100), cv::Point(400, 400),
                cv::Scalar(255, 255, 255), 5);
      // Give the output stream a new image to display
      outputStream.PutFrame(mat);
    }
  }

  void RobotInit() override {
    // We need to run our vision program in a separate thread. If not, our robot
    // program will not run.
    std::thread visionThread(VisionThread);
    visionThread.detach();
  }
};

#ifndef RUNNING_FRC_TESTS
int main() {
  return frc::StartRobot<Robot>();
}
#endif
```

PYTHON

Image processing on the roboRIO when using Python is slightly different from C++/Java. Instead of using a separate thread, we need to launch the image processing code in a completely separate process.

Avertissement : Image processing is a CPU intensive task, and because of the Python Global Interpreter Lock (GIL) **we do NOT recommend using cscore directly in your robot process**. Don't do it. Really.

For more information on the GIL and its effects, you may wish to read the following resources :

- Python Wiki : Global Interpreter Lock
- Efficiently Exploiting Multiple Cores with Python

This introduces a number of rules that your image processing code must follow to efficiently and safely run on the RoboRIO :

- Your image processing code must be in its own file
- Never import the `cscore` package from your robot code, it will just waste memory
- Never import the `wpilib` or `hal` packages from your image processing file
- The camera code will be killed when the `robot.py` program exits. If you wish to perform cleanup, you should register an atexit handler.

Avertissement : `wpilib` may not be imported from two programs on the RoboRIO. If this happens, the second program will attempt to kill the first program.

Here's what your `robot.py` needs to contain to launch the image processing process :

```
import wpilib


class MyRobot(wpilib.TimedRobot):
    """
    This is a demo program showing the use of OpenCV to do vision processing. The image is acquired
    from the USB camera, then a rectangle is put on the image and sent to the dashboard. OpenCV has
    many methods for different types of processing.
    """

```

The `launch("vision.py")` function says to launch `vision.py` and call the `run` function in that file. Here's what is in `vision.py` :

```
# NOTE: This code runs in its own process, so we cannot access the robot here,
#       nor can we create/use/see wpilib objects
#
# To try this code out locally (if you have robotpy-cscore installed), you
# can execute `python3 -m cscore vision.py:main`
#

import cv2
import numpy as np

from cscore import CameraServer as CS


def main():
    CS.enableLogging()

    # Get the UsbCamera from CameraServer
    camera = CS.startAutomaticCapture()
    # Set the resolution
    camera.setResolution(640, 480)

    # Get a CvSink. This will capture images from the camera
    cvSink = CS.getVideo()
    # Setup a CvSource. This will send images back to the Dashboard
    outputStream = CS.putVideo("Rectangle", 640, 480)

    # Allocating new images is very expensive, always try to preallocate
    mat = np.zeros(shape=(480, 640, 3), dtype=np.uint8)

    while True:
        # Tell the CvSink to grab a frame from the camera and put it
        # in the source image.  If there is an error notify the output.
        time, mat = cvSink.grabFrame(mat)
        if time == 0:
            # Send the output the error.
            outputStream.notifyError(cvSink.getError())
            # skip the rest of the current iteration
            continue

        # Put a rectangle on the image
        cv2.rectangle(mat, (100, 100), (400, 400), (255, 255, 255), 5)
```

(suite sur la page suivante)

(suite de la page précédente)

```
42
43          # Give the output stream a new image to display
44          outputStream.putFrame(mat)
```

Notez que dans ces exemples, la méthode `PutVideo()` écrit la vidéo dans un flux nommé. Pour afficher ce flux sur SmartDashboard ou Shuffleboard, sélectionnez ce flux nommé. Dans ce cas, c'est « Rectangle ».

25.4.2 Utilisation de plusieurs caméras

Changer le panorama pour le pilote

Si vous souhaitez simplement changer ce que voit le pilote et que vous utilisez SmartDashboard, le lecteur de flux SmartDashboard CameraServer a une option (« Selected Camera Path ») qui lit la clé *NetworkTables* et modifie le « Camera Choice » à cette valeur (affichage de cette caméra). Le code du robot doit alors simplement définir la clé *NetworkTables* sur le nom correct de la caméra. En supposant que « Selected Camera Path » est réglé sur « CameraSelection », le code suivant utilise l'état du bouton de déclenchement du joystick 1 pour afficher la caméra1 et la caméra2.

Java

```
UsbCamera camera1;
UsbCamera camera2;
Joystick joy1 = new Joystick(0);
NetworkTableEntry cameraSelection;

@Override
public void robotInit() {
    camera1 = CameraServer.startAutomaticCapture(0);
    camera2 = CameraServer.startAutomaticCapture(1);

    cameraSelection = NetworkTableInstance.getDefault().getTable("").getEntry(
↪"CameraSelection");
}

@Override
public void teleopPeriodic() {
    if (joy1.getTriggerPressed()) {
        System.out.println("Setting camera 2");
        cameraSelection.setString(camera2.getName());
    } else if (joy1.getTriggerReleased()) {
        System.out.println("Setting camera 1");
        cameraSelection.setString(camera1.getName());
    }
}
```

C++

```
cs::UsbCamera camera1;
cs::UsbCamera camera2;
frc::Joystick joy1{0};

nt::NetworkTableEntry cameraSelection;

void RobotInit() override {
  camera1 = frc::CameraServer::StartAutomaticCapture(0);
  camera2 = frc::CameraServer::StartAutomaticCapture(1);

  cameraSelection = nt::NetworkTableInstance::GetDefault().GetTable("")->GetEntry(
↪"CameraSelection");
}

void TeleopPeriodic() override {
  if (joy1.GetTriggerPressed()) {
    std::cout << "Setting Camera 2" << std::endl;
    cameraSelection.SetString(camera2.GetName());
  } else if (joy1.GetTriggerReleased()) {
    std::cout << "Setting Camera 1" << std::endl;
    cameraSelection.SetString(camera1.GetName());
  }
}
```

PYTHON

`robot.py` contents :

```
import wpilib
from ntcore import NetworkTableInstance

class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        self.joy1 = wpilib.Joystick(0)
        self.cameraSelection = NetworkTableInstance.getDefault().getTable("").
↪getEntry("CameraSelection")
        wpilib.CameraServer.launch("vision.py:main")

    def teleopPeriodic(self):
        if self.joy1.getTriggerPressed():
            print("Setting camera 2")
            self.cameraSelection.setString("USB Camera 1")
        elif self.joy1.getTriggerReleased():
            print("Setting camera 1")
            self.cameraSelection.setString("USB Camera 0")
```

`vision.py` contents :

```
from cscore import CameraServer

def main():
    CameraServer.enableLogging()
```

(suite sur la page suivante)

(suite de la page précédente)

```
    camera1 = CameraServer.startAutomaticCapture(0)
    camera2 = CameraServer.startAutomaticCapture(1)

    CameraServer.waitForever()
```

Si vous utilisez un autre tableau de bord, vous pouvez changer dynamiquement la caméra utilisée par le serveur de caméra. Si vous ouvrez un visualiseur de flux relié à camera1, le code du robot changera le contenu du flux en camera1 ou camera2 en lorsque le déclencheur de la manette est pesé.

JAVA

```
UsbCamera camera1;
UsbCamera camera2;
VideoSink server;
Joystick joy1 = new Joystick(0);

@Override
public void robotInit() {
    camera1 = CameraServer.startAutomaticCapture(0);
    camera2 = CameraServer.startAutomaticCapture(1);
    server = CameraServer.getServer();
}

@Override
public void teleopPeriodic() {
    if (joy1.getTriggerPressed()) {
        System.out.println("Setting camera 2");
        server.setSource(camera2);
    } else if (joy1.getTriggerReleased()) {
        System.out.println("Setting camera 1");
        server.setSource(camera1);
    }
}
```

C++

```
cs::UsbCamera camera1;
cs::UsbCamera camera2;
cs::VideoSink server;
frc::Joystick joy1{0};
bool prevTrigger = false;

void RobotInit() override {
  camera1 = frc::CameraServer::StartAutomaticCapture(0);
  camera2 = frc::CameraServer::StartAutomaticCapture(1);
  server = frc::CameraServer::GetServer();
}

void TeleopPeriodic() override {
  if (joy1.GetTrigger() && !prevTrigger) {
    std::cout << "Setting Camera 2" << std::endl;
```

(suite sur la page suivante)

(suite de la page précédente)

```
    server.SetSource(camera2);
  } else if (!joy1.GetTrigger() && prevTrigger) {
    std::cout << "Setting Camera 1" << std::endl;
    server.SetSource(camera1);
  }
  prevTrigger = joy1.GetTrigger();
}
```

PYTHON

```
# Setting the source directly via joystick isn't possible in Python, you
# should use NetworkTables as shown above instead
```

Garder les flux ouverts

Par défaut, la librairie « cscore » est assez prompte pour éteindre les caméras non utilisées. Cela signifie que lorsque vous changez de caméra, la librairie peut détacher la caméra non utilisée, avec la conséquence d'un retard possible lors de la reconnexion à un autre caméra. Pour garder les deux connexions de caméra ouvertes, utilisez la méthode `SetConnectionStrategy()` pour indiquer à la librairie de garder les flux ouverts, même si vous ne les utilisez pas.

Java

```
UsbCamera camera1;
UsbCamera camera2;
VideoSink server;
Joystick joy1 = new Joystick(0);

@Override
public void robotInit() {
    camera1 = CameraServer.startAutomaticCapture(0);
    camera2 = CameraServer.startAutomaticCapture(1);
    server = CameraServer.getServer();

    camera1.setConnectionStrategy(ConnectionStrategy.kKeepOpen);
    camera2.setConnectionStrategy(ConnectionStrategy.kKeepOpen);
}

@Override
public void teleopPeriodic() {
    if (joy1.getTriggerPressed()) {
        System.out.println("Setting camera 2");
        server.setSource(camera2);
    } else if (joy1.getTriggerReleased()) {
        System.out.println("Setting camera 1");
        server.setSource(camera1);
    }
}
```

C++

```
cs::UsbCamera camera1;
cs::UsbCamera camera2;
cs::VideoSink server;
frc::Joystick joy1{0};
bool prevTrigger = false;
void RobotInit() override {
  camera1 = frc::CameraServer::StartAutomaticCapture(0);
  camera2 = frc::CameraServer::StartAutomaticCapture(1);
  server = frc::CameraServer::GetServer();
  camera1.
↪SetConnectionStrategy(cs::VideoSource::ConnectionStrategy::kConnectionKeepOpen);
  camera2.
↪SetConnectionStrategy(cs::VideoSource::ConnectionStrategy::kConnectionKeepOpen);
}

void TeleopPeriodic() override {
  if (joy1.GetTrigger() && !prevTrigger) {
    std::cout << "Setting Camera 2" << std::endl;
    server.SetSource(camera2);
  } else if (!joy1.GetTrigger() && prevTrigger) {
    std::cout << "Setting Camera 1" << std::endl;
    server.SetSource(camera1);
  }
  prevTrigger = joy1.GetTrigger();
}
```

PYTHON

`robot.py` contents :

```
import wpilib
from ntcore import NetworkTableInstance

class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        self.joy1 = wpilib.Joystick(0)
        self.cameraSelection = NetworkTableInstance.getDefault().getTable("").
↪getEntry("CameraSelection")
        wpilib.CameraServer.launch("vision.py:main")

    def teleopPeriodic(self):
        if self.joy1.getTriggerPressed():
            print("Setting camera 2")
            self.cameraSelection.setString("USB Camera 1")
        elif self.joy1.getTriggerReleased():
            print("Setting camera 1")
            self.cameraSelection.setString("USB Camera 0")
```

`vision.py` contents :

```
from cscore import CameraServer, VideoSource

def main():
```

(suite sur la page suivante)

(suite de la page précédente)

```
    CameraServer.enableLogging()

    camera1 = CameraServer.startAutomaticCapture(0)
    camera2 = CameraServer.startAutomaticCapture(1)

    camera1.setConnectionStrategy(VideoSource.ConnectionStrategy.kConnectionKeepOpen)
    camera2.setConnectionStrategy(VideoSource.ConnectionStrategy.kConnectionKeepOpen)

    CameraServer.waitForever()
```

Note : Si les deux caméras sont USB, vous pouvez rencontrer des limitations de bande passante USB avec des résolutions plus élevées. Dans tous ces cas, le roboRIO va diffuser simultanément les données des deux caméras vers le roboRIO (pendant une courte période avec les options 1 et 2, et en continu avec l'option 3). Il est théoriquement possible pour la librairie d'éviter cette simultanéité dans le cas de l'option 2 uniquement, mais ceci n'est pas encore implémenté dans cette version..

Différentes caméras signalent l'utilisation de la bande passante différemment. La librairie vous indiquera si vous atteignez la limite ; vous obtiendrez ce message d'erreur :

```
could not start streaming due to USB bandwidth limitations;
try a lower resolution or a different pixel format
(VIDIOC_STREAMON: No space left on device)
```

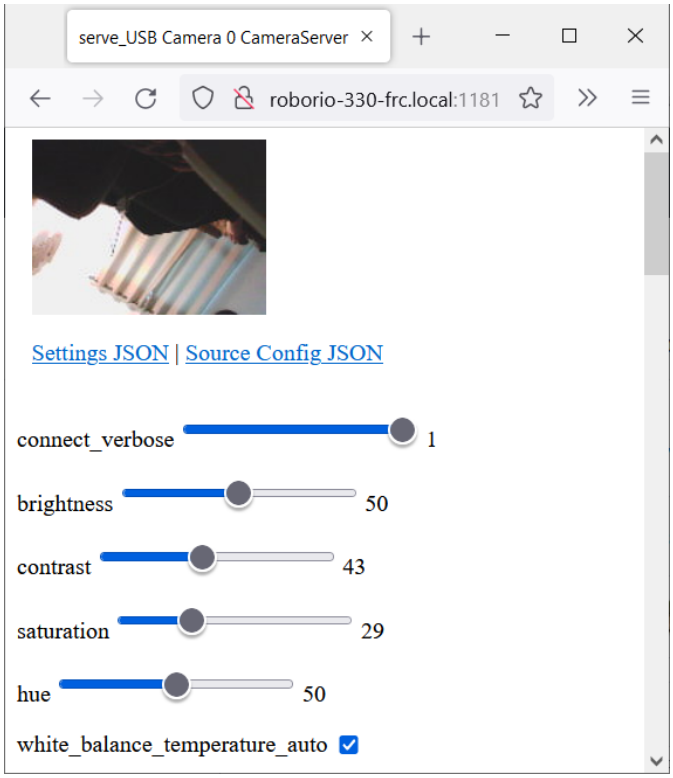
Si vous utilisez l'option 3, vous obtiendrez cette erreur durant l'exécution de `RobotInit()`. Pour palier à cet inconvénient, vous devez simplement essayer une résolution plus basse et l'ajuster si nécessaire jusqu'à ce que vous n'obteniez plus cette erreur en respectant les limites de la bande passante radio.

25.4.3 Interface Web du serveur de caméra

Lorsque CameraServer ouvre une caméra, il crée une page Web que vous pouvez utiliser pour afficher le flux de la caméra et afficher les effets des différents paramètres de la caméra. Pour vous connecter à l'interface Web, utilisez un navigateur Web pour accéder à `http://roboRIO-TEAM-frc.local:1181`. Aucun code supplémentaire n'est nécessaire autre que *Programme CameraServer élémentaire*.

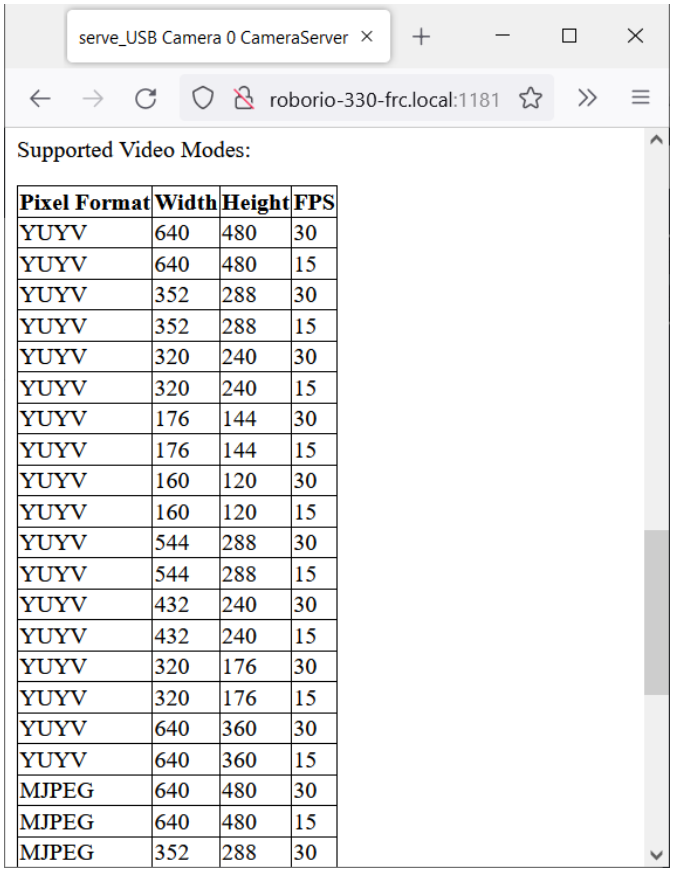
Note : The port 1181 is used for the first camera. The port increments for additional camera, so if you have two cameras, the replace 1181 above with 1182.

Paramètres de la caméra



Le serveur Web affichera une image de caméra en direct et dispose de curseurs pour régler divers paramètres de la caméra, tels que la luminosité, le contraste, la netteté et de nombreuses autres options. Vous pouvez ajuster les valeurs et voir les résultats en direct, puis utiliser la classe `VideoCamera` pour les définir dans votre code robot.

Modes vidéo de la caméra



Pixel Format	Width	Height	FPS
YUYV	640	480	30
YUYV	640	480	15
YUYV	352	288	30
YUYV	352	288	15
YUYV	320	240	30
YUYV	320	240	15
YUYV	176	144	30
YUYV	176	144	15
YUYV	160	120	30
YUYV	160	120	15
YUYV	544	288	30
YUYV	544	288	15
YUYV	432	240	30
YUYV	432	240	15
YUYV	320	176	30
YUYV	320	176	15
YUYV	640	360	30
YUYV	640	360	15
MJPEG	640	480	30
MJPEG	640	480	15
MJPEG	352	288	30

Une fonctionnalité utile est la liste des modes vidéo pris en charge au bas de la page Web. Cela montre tous les modes pris en charge par la caméra pour vous permettre de choisir celui qui est la meilleure combinaison de résolution et de fréquence d'images pour vos besoins.

26

Programmation orientée commande

Ces articles servent d'introduction et de référence pour le cadre d'application orienté commande de WPILib.

Pour un ensemble d'exemples de projets qui utilisent le cadre d'application orienté commande, consultez *Exemples de programmes basés sur l'architecture orientée Commande*.

26.1 Qu'est-ce que la programmation « orientée commande » ?

WPILib supporte une méthodologie de programmation robot appelée programmation « orientée commande. » L'expression « orientée commande » peut faire référence à la fois au paradigme général de programmation et à l'ensemble des fonctionnalités incluses dans WPILib afin de faciliter l'utilisation de ce paradigme.

« Command-based » programming is one possible *design pattern* for robot software. It is not the only way to write a robot program, but it is a very effective one. Command-based robot code tends to be clean, extensible, and (with some tricks) easy to re-use from year to year.

The command-based paradigm is also an example of *declarative programming*. The command-based library allow users to define desired robot behaviors while minimizing the amount of iteration-by-iteration robot logic that they must write. For example, in the command-based program, a user can specify that « the robot should perform an action when a condition is true » (note the use of a *lambda*) :

JAVA

```
new Trigger(condition::get).onTrue(Commands.runOnce(() -> piston.set(DoubleSolenoid.
↪Value.kForward)));
```

C++

```
Trigger([&condition] { return condition.Get()).OnTrue(frc2::cmd::RunOnce([&piston] {␣
↪piston.Set(frc::DoubleSolenoid::kForward)));
```

In contrast, without using command-based, the user would need to check the button state every iteration, and perform the appropriate action based on the state of the button.

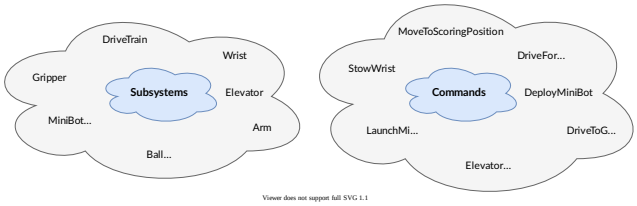
JAVA

```
if(condition.get()) {
  if(!pressed) {
    piston.set(DoubleSolenoid.Value.kForward);
    pressed = true;
  }
} else {
  pressed = false;
}
```

C++

```
if(condition.Get()) {
  if(!pressed) {
    piston.Set(frc::DoubleSolenoid::kForward);
    pressed = true;
  }
} else {
  pressed = false;
}
```

26.1.1 Sous-systèmes et commandes



Le patron orienté commande repose sur deux abstractions fondamentales : les **commandes** et les **sous-systèmes**.

Commands represent actions the robot can take. Commands run when scheduled, until they are interrupted or their end condition is met. Commands are very recursively composable : commands can be composed to accomplish more-complicated tasks. See *Commandes* for more info.

Subsystems represent independently-controlled collections of robot hardware (such as motor controllers, sensors, pneumatic actuators, etc.) that operate together. Subsystems back the resource-management system of command-based : only one command can use a given

subsystem at the same time. Subsystems allow users to « hide » the internal complexity of their actual hardware from the rest of their code - this both simplifies the rest of the robot code, and allows changes to the internal details of a subsystem's hardware without also changing the rest of the robot code.

26.1.2 L'exécution des commandes

Note : Pour une explication plus détaillée, consultez *Le planificateur de commandes*.

Commands are run by the `CommandScheduler` (Java, C++) singleton, which polls triggers (such as buttons) for commands to schedule, preventing resource conflicts, and executing scheduled commands. The scheduler's `run()` method must be called; it is generally recommended to call it from the `robotPeriodic()` method of the `Robot` class, which is run at a default frequency of 50Hz (once every 20ms).

Multiple commands can run concurrently, as long as they do not require the same resources on the robot. Resource management is handled on a per-subsystem basis : commands specify which subsystems they interact with, and the scheduler will ensure that no more more than one command requiring a given subsystem is scheduled at a time. This ensures that, for example, users will not end up with two different pieces of code attempting to set the same motor controller to different output values.

26.1.3 Command Compositions

It is often desirable to build complex commands from simple pieces. This is achievable by creating a *composition* of commands. The command-based library provides several types of *command compositions* for teams to use, and users may write their own. As command compositions are commands themselves, they may be used in a *recursive composition*. That is to say - one can create a command compositions from multiple command compositions. This provides an extremely powerful way of building complex robot actions from simple components.

26.2 Commandes

Commands represent actions the robot can take. Commands run when scheduled, until they are interrupted or their end condition is met. Commands are represented in the command-based library by the `Command` class (Java, C++).

26.2.1 La structure d'une commande

Commands specify what the command will do in each of its possible states. This is done by overriding the `initialize()`, `execute()`, and `end()` methods. Additionally, a command must be able to tell the scheduler when (if ever) it has finished execution - this is done by overriding the `isFinished()` method. All of these methods are defaulted to reduce clutter in user code : `initialize()`, `execute()`, and `end()` are defaulted to simply do nothing, while `isFinished()` is defaulted to return false (resulting in a command that never finishes naturally, and will run until interrupted).

Initialisation

The `initialize()` method (Java, C++) marks the command start, and is called exactly once per time a command is scheduled. The `initialize()` method should be used to place the command in a known starting state for execution. Command objects may be reused and scheduled multiple times, so any state or resources needed for the command's functionality should be initialized or opened in `initialize` (which will be called at the start of each use) rather than the constructor (which is invoked only once on object allocation). It is also useful for performing tasks that only need to be performed once per time scheduled, such as setting motors to run at a constant speed or setting the state of a solenoid actuator.

Exécution

The `execute()` method (Java, C++) is called repeatedly while the command is scheduled ; this is when the scheduler's `run()` method is called (this is generally done in the main robot periodic method, which runs every 20ms by default). The execute block should be used for any task that needs to be done continually while the command is scheduled, such as updating motor outputs to match joystick inputs, or using the output of a control loop.

Terminaison

The `end(bool interrupted)` method (Java, C++) is called once when the command ends, whether it finishes normally (i.e. `isFinished()` returned true) or it was interrupted (either by another command or by being explicitly canceled). The method argument specifies the manner in which the command ended ; users can use this to differentiate the behavior of their command end accordingly. The end block should be used to « wrap up » command state in a neat way, such as setting motors back to zero or reverting a solenoid actuator to a « default » state. Any state or resources initialized in `initialize()` should be closed in `end()`.

Spécifier les conditions de terminaison

The `isFinished()` method (Java, C++) is called repeatedly while the command is scheduled, whenever the scheduler's `run()` method is called. As soon as it returns true, the command's `end()` method is called and it ends. The `isFinished()` method is called after the `execute()` method, so the command will execute once on the same iteration that it ends.

26.2.2 Command Properties

In addition to the four lifecycle methods described above, each `Command` also has three properties, defined by getter methods that should always return the same value with no side affects.

getRequirements

Each command should declare any subsystems it controls as requirements. This backs the scheduler's resource management mechanism, ensuring that no more than one command requires a given subsystem at the same time. This prevents situations such as two different pieces of code attempting to set the same motor controller to different output values.

Declaring requirements is done by overriding the `getRequirements()` method in the relevant command class, by calling `addRequirements()`, or by using the `requirements` vararg (Java) / `Requirements` struct (C++) parameter at the end of the parameter list of most command constructors and factories in the library :

JAVA

```
Commands.run(intake::activate, intake);
```

C++

```
frc2::cmd::Run([&intake] { intake.Activate(); }, {&intake});
```

As a rule, command compositions require all subsystems their components require.

runsWhenDisabled

The `runsWhenDisabled()` method (Java, C++) returns a `boolean`/`bool` specifying whether the command may run when the robot is disabled. With the default of returning `false`, the command will be canceled when the robot is disabled and attempts to schedule it will do nothing. Returning `true` will allow the command to run and be scheduled when the robot is disabled.

Important : When the robot is disabled, *PWM* outputs are disabled and CAN motor controllers may not apply voltage, regardless of `runsWhenDisabled`!

This property can be set either by overriding the `runsWhenDisabled()` method in the relevant command class, or by using the `ignoringDisable` decorator (Java, C++) :

JAVA

```
Command mayRunDuringDisabled = Commands.run(() -> updateTelemetry()).
↪ignoringDisable(true);
```

C++

```
frc2::CommandPtr mayRunDuringDisabled = frc2::cmd::Run([] { UpdateTelemetry(); }).
↪IgnoringDisable(true);
```

As a rule, command compositions may run when disabled if all their component commands set `runsWhenDisabled` as `true`.

getInterruptionBehavior

The `getInterruptionBehavior()` method (Java, C++) defines what happens if another command sharing a requirement is scheduled while this one is running. In the default behavior, `kCancelSelf`, the current command will be canceled and the incoming command will be scheduled successfully. If `kCancelIncoming` is returned, the incoming command's scheduling will be aborted and this command will continue running. Note that `getInterruptionBehavior` only affects resolution of requirement conflicts : all commands can be canceled, regardless of `getInterruptionBehavior`.

Note : This was previously controlled by the `interruptible` parameter passed when scheduling a command, and is now a property of the command object.

This property can be set either by overriding the `getInterruptionBehavior` method in the relevant command class, or by using the *withInterruptBehavior()* decorator (Java, C++) :

JAVA

```
Command noninteruptible = Commands.run(intake::activate, intake).
↪withInterruptBehavior(Command.InterruptBehavior.kCancelIncoming);
```

C++

```
frc2::CommandPtr noninterruptible = frc2::cmd::Run([&intake] { intake.Activate(); },
↪{&intake}).WithInterruptBehavior(Command::InterruptBehavior::kCancelIncoming);
```

As a rule, command compositions are `kCancelIncoming` if all their components are `kCancelIncoming` as well.

26.2.3 Included Command Types

The command-based library includes many pre-written command types. Through the use of *lambdas*, these commands can cover almost all use cases and teams should rarely need to write custom command classes. Many of these commands are provided via static factory functions in the `Commands` utility class (Java) or in the `frc2::cmd` namespace defined in the `Commands.h` header (C++). Classes inheriting from `Subsystem` also have instance methods that implicitly require `this`.

Running Actions

The most basic commands are actions the robot takes : setting voltage to a motor, changing a solenoid's direction, etc. For these commands, which typically consist of a method call or two, the command-based library offers several factories to be construct commands inline with one or more lambdas to be executed.

The `runOnce` factory, backed by the `InstantCommand` (Java, C++) class, creates a command that calls a lambda once, and then finishes.

Java

```
  /** Grabs the hatch. */
  public Command grabHatchCommand() {
    // implicitly require `this`
    return this.runOnce(() -> m_hatchSolenoid.set(kForward));
  }

  /** Releases the hatch. */
  public Command releaseHatchCommand() {
    // implicitly require `this`
    return this.runOnce(() -> m_hatchSolenoid.set(kReverse));
  }
```

C++ (Header ou en-tête)

```
  /**
   * Grabs the hatch.
   */
  frc2::CommandPtr GrabHatchCommand();

  /**
   * Releases the hatch.
   */
  frc2::CommandPtr ReleaseHatchCommand();
```

C++ (Source)

```
frc2::CommandPtr HatchSubsystem::GrabHatchCommand() {
  // implicitly require `this`
  return this->RunOnce(
      [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kForward); });
}

frc2::CommandPtr HatchSubsystem::ReleaseHatchCommand() {
  // implicitly require `this`
  return this->RunOnce(
      [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kReverse); });
}
```

The `run` factory, backed by the `RunCommand` (Java, C++) class, creates a command that calls a lambda repeatedly, until interrupted.

JAVA

```
// A split-stick arcade command, with forward/backward controlled by the left
// hand, and turning controlled by the right.
new RunCommand(() -> m_robotDrive.arcadeDrive(
    -driverController.getLeftY(),
    driverController.getRightX()),
    m_robotDrive)
```

C++

```
// A split-stick arcade command, with forward/backward controlled by the left
// hand, and turning controlled by the right.
frc2::RunCommand(
  [this] {
    m_drive.ArcadeDrive(
        -m_driverController.GetLeftY(),
        m_driverController.GetRightX());
  },
  {&m_drive}))
```

The `startEnd` factory, backed by the `StartEndCommand` (Java, C++) class, calls one lambda when scheduled, and then a second lambda when interrupted.

JAVA

```
Commands.startEnd(
    // Start a flywheel spinning at 50% power
    () -> m_shooter.shooterSpeed(0.5),
    // Stop the flywheel at the end of the command
    () -> m_shooter.shooterSpeed(0.0),
    // Requires the shooter subsystem
    m_shooter
)
```

C++

```
frc2::cmd::StartEnd(
  // Start a flywheel spinning at 50% power
  [this] { m_shooter.shooterSpeed(0.5); },
  // Stop the flywheel at the end of the command
  [this] { m_shooter.shooterSpeed(0.0); },
  // Requires the shooter subsystem
  {&m_shooter}
)
```

`FunctionalCommand` (Java, C++) accepts four lambdas that constitute the four command lifecycle methods : a `Runnable`/`std::function<void()>` for each of `initialize()` and `execute()`, a `BooleanConsumer`/`std::function<void(bool)>` for `end()`, and a `BooleanSupplier`/`std::function<bool()>` for `isFinished()`.

JAVA

```
new FunctionalCommand(
    // Reset encoders on command start
    m_robotDrive::resetEncoders,
    // Start driving forward at the start of the command
    () -> m_robotDrive.arcadeDrive(kAutoDriveSpeed, 0),
    // Stop driving at the end of the command
    interrupted -> m_robotDrive.arcadeDrive(0, 0),
    // End the command when the robot's driven distance exceeds the desired value
    () -> m_robotDrive.getAverageEncoderDistance() >= kAutoDriveDistanceInches,
    // Require the drive subsystem
    m_robotDrive
)
```

C++

```
frc2::FunctionalCommand(
  // Reset encoders on command start
  [this] { m_drive.ResetEncoders(); },
  // Start driving forward at the start of the command
  [this] { m_drive.ArcadeDrive(ac::kAutoDriveSpeed, 0); },
  // Stop driving at the end of the command
  [this] (bool interrupted) { m_drive.ArcadeDrive(0, 0); },
  // End the command when the robot's driven distance exceeds the desired value
  [this] { return m_drive.GetAverageEncoderDistance() >= kAutoDriveDistanceInches; },
  // Requires the drive subsystem
  {&m_drive}
)
```

To print a string and ending immediately, the library offers the `Commands.print(String)`/`frc2::cmd::Print(std::string_view)` factory, backed by the `PrintCommand` (Java, C++) subclass of `InstantCommand`.

Waiting

Waiting for a certain condition to happen or adding a delay can be useful to synchronize between different commands in a command composition or between other robot actions.

To wait and end after a specified period of time elapses, the library offers the `Commands.waitSeconds(double)`/`frc2::cmd::Wait(units::second_t)` factory, backed by the `WaitCommand` (Java, C++) class.

JAVA

```
// Ends 5 seconds after being scheduled
new WaitCommand(5.0)
```

C++

```
// Ends 5 seconds after being scheduled
frc2::WaitCommand(5.0_s)
```

To wait until a certain condition becomes `true`, the library offers the `Commands.waitUntil(BooleanSupplier)`/`frc2::cmd::WaitUntil(std::function<bool()>)` factory, backed by the `WaitUntilCommand` class (Java, C++).

JAVA

```
// Ends after m_limitSwitch.get() returns true
new WaitUntilCommand(m_limitSwitch::get)
```

C++

```
// Ends after m_limitSwitch.Get() returns true
frc2::WaitUntilCommand([&m_limitSwitch] { return m_limitSwitch.Get(); })
```

Control Algorithm Commands

There are commands for various control setups :

- `PIDCommand` uses a PID controller. For more info, see *La classe PIDCommand*.
- `TrapezoidProfileCommand` tracks a trapezoid motion profile. For more info, see *TrapezoidProfileCommand*.
- `ProfiledPIDCommand` combines PID control with trapezoid motion profiles. For more info, see *ProfiledPIDCommand*.
- `MecanumControllerCommand` (Java, C++) is useful for controlling mecanum drivetrains. See API docs and the **MecanumControllerCommand** (Java, C++) example project for more info.
- `SwerveControllerCommand` (Java, C++) is useful for controlling swerve drivetrains. See API docs and the **SwerveControllerCommand** (Java, C++) example project for more info.
- `RamseteCommand` (Java, C++) is useful for path following with differential drivetrains (« tank drive »). See API docs and the *Trajectory Tutorial* for more info.

26.2.4 Custom Command Classes

Users may also write custom command classes. As this is significantly more verbose, it's recommended to use the more concise factories mentioned above.

Note : In the C++ API, a *CRTP* is used to allow certain Command methods to work with the object ownership model. Users should always extend the `CommandHelper` class when defining their own command classes, as is shown below.

To write a custom command class, subclass the abstract `Command` class (Java) or `CommandHelper` (C++), as seen in the command-based template (Java, C++) :

JAVA

```java
import edu.wpi.first.wpilibj.templates.commandbased.subsystems.ExampleSubsystem;
import edu.wpi.first.wpilibj2.command.Command;

/** An example command that uses an example subsystem. */
public class ExampleCommand extends Command {
  @SuppressWarnings({"PMD.UnusedPrivateField", "PMD.SingularField"})
  private final ExampleSubsystem m_subsystem;

  /**
   * Creates a new ExampleCommand.
   *
   * @param subsystem The subsystem used by this command.
   */
  public ExampleCommand(ExampleSubsystem subsystem) {
    m_subsystem = subsystem;
    // Use addRequirements() here to declare subsystem dependencies.
    addRequirements(subsystem);
  }
```

C++

```cpp
#pragma once

#include <frc2/command/Command.h>
#include <frc2/command/CommandHelper.h>

#include "subsystems/ExampleSubsystem.h"

/**
 * An example command that uses an example subsystem.
 *
 * <p>Note that this extends CommandHelper, rather extending Command
 * directly; this is crucially important, or else the decorator functions in
 * Command will *not* work!
 */
class ExampleCommand
    : public frc2::CommandHelper<frc2::Command, ExampleCommand> {
 public:
```

(suite sur la page suivante)

(suite de la page précédente)

```
  /**
   * Creates a new ExampleCommand.
   *
   * @param subsystem The subsystem used by this command.
   */
  explicit ExampleCommand(ExampleSubsystem* subsystem);

 private:
  ExampleSubsystem* m_subsystem;
};
```

26.2.5 Exemple de commande de base

What might a functional command look like in practice ? As before, below is a simple command from the HatchBot example project (Java, C++) that uses the `HatchSubsystem` :

Java

```
package edu.wpi.first.wpilibj.examples.hatchbottraditional.commands;

import edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems.
↪HatchSubsystem;
import edu.wpi.first.wpilibj2.command.Command;

/**
 * A simple command that grabs a hatch with the {@link HatchSubsystem}.␣
↪Written explicitly for
 * pedagogical purposes. Actual code should inline a command this simple␣
↪with {@link
 * edu.wpi.first.wpilibj2.command.InstantCommand}.
 */
public class GrabHatch extends Command {
  // The subsystem the command runs on
  private final HatchSubsystem m_hatchSubsystem;

  public GrabHatch(HatchSubsystem subsystem) {
    m_hatchSubsystem = subsystem;
    addRequirements(m_hatchSubsystem);
  }

  @Override
  public void initialize() {
    m_hatchSubsystem.grabHatch();
  }

  @Override
  public boolean isFinished() {
    return true;
  }
}
```

C++ (Header ou en-tête)

```
#pragma once

#include <frc2/command/Command.h>
#include <frc2/command/CommandHelper.h>

#include "subsystems/HatchSubsystem.h"

/**
 * A simple command that grabs a hatch with the HatchSubsystem.  Written
 * explicitly for pedagogical purposes.  Actual code should inline a command
 * this simple with InstantCommand.
 *
 * @see InstantCommand
 */
class GrabHatch : public frc2::CommandHelper<frc2::Command, GrabHatch> {
 public:
  explicit GrabHatch(HatchSubsystem* subsystem);

  void Initialize() override;

  bool IsFinished() override;

 private:
  HatchSubsystem* m_hatch;
};
```

C++ (Source)

```
#include "commands/GrabHatch.h"

GrabHatch::GrabHatch(HatchSubsystem* subsystem) : m_hatch(subsystem) {
  AddRequirements(subsystem);
}

void GrabHatch::Initialize() {
  m_hatch->GrabHatch();
}

bool GrabHatch::IsFinished() {
  return true;
}
```

Notice that the hatch subsystem used by the command is passed into the command through the command's constructor. This is a pattern called *dependency injection*, and allows users to avoid declaring their subsystems as global variables. This is widely accepted as a best-practice - the reasoning behind this is discussed in a *later section*.

Notice also that the above command calls the subsystem method once from initialize, and then immediately ends (as `isFinished()` simply returns true). This is typical for commands that toggle the states of subsystems, and as such it would be more succinct to write this command using the factories described above.

Mais que se passe-t-il dans un cas plus compliqué ? Ci-dessous se trouve une commande de pilotage, du même projet d'exemple :

Java

```java
package edu.wpi.first.wpilibj.examples.hatchbottraditional.commands;

import edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems.DriveSubsystem;
import edu.wpi.first.wpilibj2.command.Command;
import java.util.function.DoubleSupplier;

/**
 * A command to drive the robot with joystick input (passed in as {@link DoubleSupplier}s). Written
 * explicitly for pedagogical purposes - actual code should inline a command this simple with {@link
 * edu.wpi.first.wpilibj2.command.RunCommand}.
 */
public class DefaultDrive extends Command {
  private final DriveSubsystem m_drive;
  private final DoubleSupplier m_forward;
  private final DoubleSupplier m_rotation;

  /**
   * Creates a new DefaultDrive.
   *
   * @param subsystem The drive subsystem this command wil run on.
   * @param forward The control input for driving forwards/backwards
   * @param rotation The control input for turning
   */
  public DefaultDrive(DriveSubsystem subsystem, DoubleSupplier forward, DoubleSupplier rotation) {
    m_drive = subsystem;
    m_forward = forward;
    m_rotation = rotation;
    addRequirements(m_drive);
  }

  @Override
  public void execute() {
    m_drive.arcadeDrive(m_forward.getAsDouble(), m_rotation.getAsDouble());
  }
}
```

C++ (Header ou en-tête)

```cpp
#pragma once

#include <functional>

#include <frc2/command/Command.h>
#include <frc2/command/CommandHelper.h>

#include "subsystems/DriveSubsystem.h"

/**
 * A command to drive the robot with joystick input passed in through lambdas.
 * Written explicitly for pedagogical purposes - actual code should inline a
 * command this simple with RunCommand.
 *
 * @see RunCommand
 */
class DefaultDrive : public frc2::CommandHelper<frc2::Command, DefaultDrive> {
 public:
  /**
   * Creates a new DefaultDrive.
   *
   * @param subsystem The drive subsystem this command wil run on.
   * @param forward The control input for driving forwards/backwards
   * @param rotation The control input for turning
   */
  DefaultDrive(DriveSubsystem* subsystem, std::function<double()> forward,
               std::function<double()> rotation);

  void Execute() override;

 private:
  DriveSubsystem* m_drive;
  std::function<double()> m_forward;
  std::function<double()> m_rotation;
};
```

C++ (Source)

```cpp
#include "commands/DefaultDrive.h"

#include <utility>

DefaultDrive::DefaultDrive(DriveSubsystem* subsystem,
                           std::function<double()> forward,
                           std::function<double()> rotation)
    : m_drive{subsystem},
      m_forward{std::move(forward)},
      m_rotation{std::move(rotation)} {
  AddRequirements(subsystem);
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
17
18 void DefaultDrive::Execute() {
19   m_drive->ArcadeDrive(m_forward(), m_rotation());
20 }
```

And then usage :

JAVA

```
59     // Configure default commands
60     // Set the default drive command to split-stick arcade drive
61     m_robotDrive.setDefaultCommand(
62         // A split-stick arcade command, with forward/backward controlled by the left
63         // hand, and turning controlled by the right.
64         new DefaultDrive(
65             m_robotDrive,
66             () -> -m_driverController.getLeftY(),
67             () -> -m_driverController.getRightX()));
```

C++

```
57   // Set up default drive command
58   m_drive.SetDefaultCommand(DefaultDrive(
59       &m_drive, [this] { return -m_driverController.GetLeftY(); },
60       [this] { return -m_driverController.GetRightX(); }));
```

Notice that this command does not override `isFinished()`, and thus will never end; this is the norm for commands that are intended to be used as default commands. Once more, this command is rather simple and calls the subsystem method only from one place, and as such, could be more concisely written using factories :

JAVA

```
51     // Configure default commands
52     // Set the default drive command to split-stick arcade drive
53     m_robotDrive.setDefaultCommand(
54         // A split-stick arcade command, with forward/backward controlled by the left
55         // hand, and turning controlled by the right.
56         Commands.run(
57             () ->
58                 m_robotDrive.arcadeDrive(
59                     -m_driverController.getLeftY(), -m_driverController.getRightX()),
60             m_robotDrive));
```

C++

```
  // Set up default drive command
  m_drive.SetDefaultCommand(frc2::cmd::Run(
      [this] {
        m_drive.ArcadeDrive(-m_driverController.GetLeftY(),
                            -m_driverController.GetRightX());
      },
      {&m_drive}));
```

26.3 Command Compositions

Individual commands are capable of accomplishing a large variety of robot tasks, but the simple three-state format can quickly become cumbersome when more advanced functionality requiring extended sequences of robot tasks or coordination of multiple robot subsystems is required. In order to accomplish this, users are encouraged to use the powerful command composition functionality included in the command-based library.

As the name suggests, a command composition is a *composition* of one or more commands. This allows code to be kept much cleaner and simpler, as the individual component commands may be written independently of the code that combines them, greatly reducing the amount of complexity at any given step of the process.

Most importantly, however, command compositions are themselves commands - they extend the `Command` class. This allows command compositions to be further composed as a *recursive composition* - that is, a command composition may contain other command compositions as components. This allows very powerful and concise inline expressions :

JAVA

```
// Will run fooCommand, and then a race between barCommand and bazCommand
button.onTrue(fooCommand.andThen(barCommand.raceWith(bazCommand)));
```

C++

```
// Will run fooCommand, and then a race between barCommand and bazCommand
button.OnTrue(std::move(fooCommand).AndThen(std::move(barCommand).
↪RaceWith(std::move(bazCommand))));
```

As a rule, command compositions require all subsystems their components require, may run when disabled if all their component set `runsWhenDisabled` as `true`, and are `kCancelIncoming` if all their components are `kCancelIncoming` as well.

Command instances that have been passed to a command composition cannot be independently scheduled or passed to a second command composition. Attempting to do so will throw an exception and crash the user program. This is because composition members are run through their encapsulating command composition, and errors could occur if those same command instances were independently scheduled at the same time as the composition - the

command would be being run from multiple places at once, and thus could end up with inconsistent internal state, causing unexpected and hard-to-diagnose behavior. The C++ command-based library uses `CommandPtr`, a class with move-only semantics, so this type of mistake is easier to avoid.

26.3.1 Composition Types

The command-based library includes various composition types. All of them can be constructed using factories that accept the member commands, and some can also be constructed using decorators : methods that can be called on a command object, which is transformed into a new object that is returned.

Important : After calling a decorator or being passed to a composition, the command object cannot be reused ! Use only the command object returned from the decorator.

Repeating

The `repeatedly()` decorator (Java, C++), backed by the `RepeatCommand` class (Java, C++) restarts the command each time it ends, so that it runs until interrupted.

JAVA

```
// Will run forever unless externally interrupted, restarting every time command.
↪isFinished() returns true
Command repeats = command.repeatedly();
```

C++

```
// Will run forever unless externally interrupted, restarting every time command.
↪IsFinished() returns true
frc2::CommandPtr repeats = std::move(command).Repeatedly();
```

Sequence

The `Sequence` factory (Java, C++), backed by the `SequentialCommandGroup` class (Java, C++), runs a list of commands in sequence : the first command will be executed, then the second, then the third, and so on until the list finishes. The sequential group finishes after the last command in the sequence finishes. It is therefore usually important to ensure that each command in the sequence does actually finish (if a given command does not finish, the next command will never start !).

The `andThen()` (Java, C++) and `beforeStarting()` (Java, C++) decorators can be used to construct a sequence composition with infix syntax.

JAVA

```
fooCommand.andThen(barCommand)
```

C++

```
std::move(fooCommand).AndThen(std::move(barCommand))
```

Repeating Sequence

As it's a fairly common combination, the `RepeatingSequence` factory (Java, C++) creates a *Repeating Sequence* that runs until interrupted, restarting from the first command each time the last command finishes.

Parallel

There are three types of parallel compositions, differing based on when the composition finishes :

- The `Parallel` factory (Java, C++), backed by the `ParallelCommandGroup` class (Java, C++), constructs a parallel composition that finishes when all members finish. The `alongWith` decorator (Java, C++) does the same in infix notation.
- The `Race` factory (Java, C++), backed by the `ParallelRaceGroup` class (Java, C++), constructs a parallel composition that finishes as soon as any member finishes ; all other members are interrupted at that point. The `raceWith` decorator (Java, C++) does the same in infix notation.
- The `Deadline` factory (Java, C++), `ParallelDeadlineGroup` (Java, C++) finishes when a specific command (the « deadline ») ends ; all other members still running at that point are interrupted. The `deadlineWith` decorator (Java, C++) does the same in infix notation ; the comand the decorator was called on is the deadline.

JAVA

```
// Will be a parallel command composition that ends after three seconds with all␣
↪three commands running their full duration.
button.onTrue(Commands.parallel(twoSecCommand, oneSecCommand, threeSecCommand));

// Will be a parallel race composition that ends after one second with the two and␣
↪three second commands getting interrupted.
button.onTrue(Commands.race(twoSecCommand, oneSecCommand, threeSecCommand));

// Will be a parallel deadline composition that ends after two seconds (the deadline)␣
↪with the three second command getting interrupted (one second command already␣
↪finished).
button.onTrue(Commands.deadline(twoSecCommand, oneSecCommand, threeSecCommand));
```

C++

```
// Will be a parallel command composition that ends after three seconds with all␣
↪three commands running their full duration.
button.OnTrue(frc2::cmd::Parallel(std::move(twoSecCommand), std::move(oneSecCommand),␣
↪std::move(threeSecCommand)));

// Will be a parallel race composition that ends after one second with the two and␣
↪three second commands getting interrupted.
button.OnTrue(frc2::cmd::Race(std::move(twoSecCommand), std::move(oneSecCommand),␣
↪std::move(threeSecCommand)));

// Will be a parallel deadline composition that ends after two seconds (the deadline)␣
↪with the three second command getting interrupted (one second command already␣
↪finished).
button.OnTrue(frc2::cmd::Deadline(std::move(twoSecCommand), std::move(oneSecCommand),␣
↪std::move(threeSecCommand)));
```

Adding Command End Conditions

The `until()` (Java, C++) decorator composes the command with an additional end condition. Note that the command the decorator was called on will see this end condition as an interruption.

JAVA

```
// Will be interrupted if m_limitSwitch.get() returns true
button.onTrue(command.until(m_limitSwitch::get));
```

C++

```
// Will be interrupted if m_limitSwitch.get() returns true
button.OnTrue(command.Until([&m_limitSwitch] { return m_limitSwitch.Get(); }));
```

The `withTimeout()` decorator (Java, C++) is a specialization of `until` that uses a timeout as the additional end condition.

JAVA

```
// Will time out 5 seconds after being scheduled, and be interrupted
button.onTrue(command.withTimeout(5));
```

C++

```
// Will time out 5 seconds after being scheduled, and be interrupted
button.OnTrue(command.WithTimeout(5.0_s));
```

Adding End Behavior

The `finallyDo()` (Java, C++) decorator composes the command with an a lambda that will be called after the command's `end()` method, with the same boolean parameter indicating whether the command finished or was interrupted.

The `handleInterrupt()` (Java, C++) decorator composes the command with an a lambda that will be called only when the command is interrupted.

Selecting Compositions

Sometimes it's desired to run a command out of a few options based on sensor feedback or other data known only at runtime. This can be useful for determining an auto routine, or running a different command based on whether a game piece is present or not, and so on.

The `Select` factory (Java, C++), backed by the `SelectCommand` class (Java, C++), executes one command from a map, based on a selector function called when scheduled.

Java

```
public class RobotContainer {
  // The enum used as keys for selecting the command to run.
  private enum CommandSelector {
    ONE,
    TWO,
    THREE
  }

  // An example selector method for the selectcommand.  Returns the selector that will select
  // which command to run.  Can base this choice on logical conditions evaluated at runtime.
  private CommandSelector select() {
    return CommandSelector.ONE;
  }

  // An example selectcommand.  Will select from the three commands based on the value returned
  // by the selector method at runtime.  Note that selectcommand works on Object(), so the
  // selector does not have to be an enum; it could be any desired type (string, integer,
  // boolean, double...)
  private final Command m_exampleSelectCommand =
      new SelectCommand<>(
          // Maps selector values to commands
          Map.ofEntries(
```

(suite sur la page suivante)

(suite de la page précédente)

```
                Map.entry(CommandSelector.ONE, new PrintCommand("Command one was␣
↪selected!")),
                Map.entry(CommandSelector.TWO, new PrintCommand("Command two was␣
↪selected!")),
                Map.entry(CommandSelector.THREE, new PrintCommand("Command three was␣
↪selected!"))),
            this::select);
```

C++ (Header ou en-tête)

```
  // The enum used as keys for selecting the command to run.
  enum CommandSelector { ONE, TWO, THREE };

  // An example of how command selector may be used with SendableChooser
  frc::SendableChooser<CommandSelector> m_chooser;

  // The robot's subsystems and commands are defined here...

  // An example selectcommand.  Will select from the three commands based on the
  // value returned by the selector method at runtime.  Note that selectcommand
  // takes a generic type, so the selector does not have to be an enum; it could
  // be any desired type (string, integer, boolean, double...)
  frc2::CommandPtr m_exampleSelectCommand = frc2::cmd::Select<CommandSelector>(
      [this] { return m_chooser.GetSelected(); },
      // Maps selector values to commands
      std::pair{ONE, frc2::cmd::Print("Command one was selected!")},
      std::pair{TWO, frc2::cmd::Print("Command two was selected!")},
      std::pair{THREE, frc2::cmd::Print("Command three was selected!")});
```

The `Either` factory (Java, C++), backed by the `ConditionalCommand` class (Java, C++), is a specialization accepting two commands and a boolean selector function.

JAVA

```
// Runs either commandOnTrue or commandOnFalse depending on the value of m_
↪limitSwitch.get()
new ConditionalCommand(commandOnTrue, commandOnFalse, m_limitSwitch::get)
```

C++

```
// Runs either commandOnTrue or commandOnFalse depending on the value of m_
↪limitSwitch.get()
frc2::ConditionalCommand(commandOnTrue, commandOnFalse, [&m_limitSwitch] { return m_
↪limitSwitch.Get(); })
```

The `unless()` decorator (Java, C++) composes a command with a condition that will prevent it from running.

JAVA

```
// Command will only run if the intake is deployed. If the intake gets deployed while␣
↪the command is running, the command will not stop running
button.onTrue(command.unless(() -> !intake.isDeployed()));
```

C++

```
// Command will only run if the intake is deployed. If the intake gets deployed while␣
↪the command is running, the command will not stop running
button.OnTrue(command.Unless([&intake] { return !intake.IsDeployed(); }));
```

`ProxyCommand` described below also has a constructor overload (Java, C++) that calls a command-returning lambda at schedule-time and runs the returned command by proxy.

Scheduling Other Commands

By default, composition members are run through the command composition, and are never themselves seen by the scheduler. Accordingly, their requirements are added to the composition's requirements. While this is usually fine, sometimes it is undesirable for the entire command composition to gain the requirements of a single command. A good solution is to « fork off » from the command composition and schedule that command separately. However, this requires synchronization between the composition and the individually-scheduled command.

`ProxyCommand` (Java, C++), also creatable using the `.asProxy()` decorator (Java, C++), schedules a command « by proxy » : the command is scheduled when the proxy is scheduled, and the proxy finishes when the command finishes. In the case of « forking off » from a command composition, this allows the composition to track the command's progress without it being in the composition.

Command compositions inherit the union of their compoments" requirements and requirements are immutable. Therefore, a `SequentialCommandGroup` (Java, C++) that intakes a game piece, indexes it, aims a shooter, and shoots it would reserve all three subsystems (the intake, indexer, and shooter), precluding any of those subsystems from performing other operations in their « downtime ». If this is not desired, the subsystems that should only be reserved for the composition while they are actively being used by it should have their commands proxied.

Avertissement : Do not use `ProxyCommand` unless you are sure of what you are doing and there is no other way to accomplish your need ! Proxying is only intended for use as an escape hatch from command composition requirement unions.

Note : Because proxied commands still require their subsystem, despite not leaking that requirement to the composition, all of the commands that require a given subsystem must be proxied if one of them is. Otherwise, when the proxied command is scheduled its requirement will conflict with that of the composition, canceling the composition.

JAVA

```
// composition requirements are indexer and shooter, intake still reserved during its␣
↪command but not afterwards
Commands.sequence(
   intake.intakeGamePiece().asProxy(), // we want to let the intake intake another␣
↪game piece while we are processing this one
   indexer.processGamePiece(),
   shooter.aimAndShoot()
);
```

C++

```
// composition requirements are indexer and shooter, intake still reserved during its␣
↪command but not afterwards
frc2::cmd::Sequence(
   intake.IntakeGamePiece().AsProxy(), // we want to let the intake intake another␣
↪game piece while we are processing this one
   indexer.ProcessGamePiece(),
   shooter.AimAndShoot()
);
```

For cases that don't need to track the proxied command, `ScheduleCommand` (Java, C++) schedules a specified command and ends instantly.

JAVA

```
// ScheduleCommand ends immediately, so the sequence continues
new ScheduleCommand(Commands.waitSeconds(5.0))
   .andThen(Commands.print("This will be printed immediately!"))
```

C++

```
// ScheduleCommand ends immediately, so the sequence continues
frc2::ScheduleCommand(frc2::cmd::Wait(5.0_s))
   .AndThen(frc2::cmd::Print("This will be printed immediately!"))
```

26.3.2 Subclassing Compositions

Command compositions can also be written as a constructor-only subclass of the most exterior composition type, passing the composition members to the superclass constructor. Consider the following from the Hatch Bot example project (Java, C++) :

Java

```
package edu.wpi.first.wpilibj.examples.hatchbottraditional.commands;

import edu.wpi.first.wpilibj.examples.hatchbottraditional.Constants.AutoConstants;
import edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems.DriveSubsystem;
import edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems.HatchSubsystem;
import edu.wpi.first.wpilibj2.command.SequentialCommandGroup;

/** A complex auto command that drives forward, releases a hatch, and then drives␣
↪backward. */
public class ComplexAuto extends SequentialCommandGroup {
  /**
   * Creates a new ComplexAuto.
   *
   * @param drive The drive subsystem this command will run on
   * @param hatch The hatch subsystem this command will run on
   */
  public ComplexAuto(DriveSubsystem drive, HatchSubsystem hatch) {
    addCommands(
        // Drive forward the specified distance
        new DriveDistance(
            AutoConstants.kAutoDriveDistanceInches, AutoConstants.kAutoDriveSpeed,␣
↪drive),

        // Release the hatch
        new ReleaseHatch(hatch),

        // Drive backward the specified distance
        new DriveDistance(
            AutoConstants.kAutoBackupDistanceInches, -AutoConstants.kAutoDriveSpeed,␣
↪drive));
  }
}
```

C++ (Header ou en-tête)

```
#pragma once

#include <frc2/command/CommandHelper.h>
#include <frc2/command/SequentialCommandGroup.h>

#include "Constants.h"
#include "commands/DriveDistance.h"
#include "commands/ReleaseHatch.h"

/**
 * A complex auto command that drives forward, releases a hatch, and then drives
 * backward.
 */
class ComplexAuto
    : public frc2::CommandHelper<frc2::SequentialCommandGroup, ComplexAuto> {
 public:
  /**
   * Creates a new ComplexAuto.
```

(suite sur la page suivante)

(suite de la page précédente)

```
   *
   * @param drive The drive subsystem this command will run on
   * @param hatch The hatch subsystem this command will run on
   */
  ComplexAuto(DriveSubsystem* drive, HatchSubsystem* hatch);
};
```

C++ (Source)

```
#include "commands/ComplexAuto.h"

using namespace AutoConstants;

ComplexAuto::ComplexAuto(DriveSubsystem* drive, HatchSubsystem* hatch) {
  AddCommands(
      // Drive forward the specified distance
      DriveDistance(kAutoDriveDistanceInches, kAutoDriveSpeed, drive),
      // Release the hatch
      ReleaseHatch(hatch),
      // Drive backward the specified distance
      DriveDistance(kAutoBackupDistanceInches, -kAutoDriveSpeed, drive));
}
```

The advantages and disadvantages of this subclassing approach in comparison to others are discussed in *Subclassing Command Groups*.

26.4 Les sous-systèmes

Subsystems are the basic unit of robot organization in the command-based paradigm. A subsystem is an abstraction for a collection of robot hardware that *operates together as a unit*. Subsystems form an *encapsulation* for this hardware, « hiding » it from the rest of the robot code and restricting access to it except through the subsystem's public methods. Restricting the access in this way provides a single convenient place for code that might otherwise be duplicated in multiple places (such as scaling motor outputs or checking limit switches) if the subsystem internals were exposed. It also allows changes to the specific details of how the subsystem works (the « implementation ») to be isolated from the rest of robot code, making it far easier to make substantial changes if/when the design constraints change.

Subsystems also serve as the backbone of the `CommandScheduler`'s resource management system. Commands may declare resource requirements by specifying which subsystems they interact with ; the scheduler will never concurrently schedule more than one command that requires a given subsystem. An attempt to schedule a command that requires a subsystem that is already-in-use will either interrupt the currently-running command or be ignored, based on the running command's *Interruption Behavior*.

Subsystems can be associated with « default commands » that will be automatically scheduled when no other command is currently using the subsystem. This is useful for « background » actions such as controlling the robot drive, keeping an arm held at a setpoint, or stopping motors when the subsystem isn't used. Similar functionality can be achieved in the subsystem's `periodic()` method, which is run once per run of the scheduler ; teams should try to be consistent within their codebase about which functionality is achieved through either of

these methods. Subsystems are represented in the command-based library by the `Subsystem` interface (Java, C++).

26.4.1 Créer un sous-système

The recommended method to create a subsystem for most users is to subclass the abstract `SubsystemBase` class (Java, C++), as seen in the command-based template (Java, C++) :

Java

```
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.SubsystemBase;

public class ExampleSubsystem extends SubsystemBase {
  /** Creates a new ExampleSubsystem. */
  public ExampleSubsystem() {}

  /**
   * Example command factory method.
   *
   * @return a command
   */
  public Command exampleMethodCommand() {
    // Inline construction of command goes here.
    // Subsystem::RunOnce implicitly requires `this` subsystem.
    return runOnce(
        () -> {
          /* one-time action goes here */
        });
  }

  /**
   * An example method querying a boolean state of the subsystem (for example, a digital sensor).
   *
   * @return value of some boolean subsystem state, such as a digital sensor.
   */
  public boolean exampleCondition() {
    // Query some boolean state, such as a digital sensor.
    return false;
  }

  @Override
  public void periodic() {
    // This method will be called once per scheduler run
  }

  @Override
  public void simulationPeriodic() {
    // This method will be called once per scheduler run during simulation
  }
}
```

C++

```
#pragma once

#include <frc2/command/CommandPtr.h>
#include <frc2/command/SubsystemBase.h>

class ExampleSubsystem : public frc2::SubsystemBase {
 public:
  ExampleSubsystem();

  /**
   * Example command factory method.
   */
  frc2::CommandPtr ExampleMethodCommand();

  /**
   * An example method querying a boolean state of the subsystem (for example, a
   * digital sensor).
   *
   * @return value of some boolean subsystem state, such as a digital sensor.
   */
  bool ExampleCondition();

  /**
   * Will be called periodically whenever the CommandScheduler runs.
   */
  void Periodic() override;

  /**
   * Will be called periodically whenever the CommandScheduler runs during
   * simulation.
   */
  void SimulationPeriodic() override;

 private:
  // Components (e.g. motor controllers and sensors) should generally be
  // declared private and exposed only through public methods.
};
```

This class contains a few convenience features on top of the basic `Subsystem` interface : it automatically calls the `register()` method in its constructor to register the subsystem with the scheduler (this is necessary for the `periodic()` method to be called when the scheduler runs), and also implements the `Sendable` interface so that it can be sent to the dashboard to display/log relevant status information.

Advanced users seeking more flexibility may simply create a class that implements the `Subsystem` interface.

26.4.2 Exemple de sous-système simple

What might a functional subsystem look like in practice ? Below is a simple pneumatically-actuated hatch mechanism from the HatchBotTraditional example project (Java, C++) :

Java

```java
package edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems;

import static edu.wpi.first.wpilibj.DoubleSolenoid.Value.kForward;
import static edu.wpi.first.wpilibj.DoubleSolenoid.Value.kReverse;

import edu.wpi.first.util.sendable.SendableBuilder;
import edu.wpi.first.wpilibj.DoubleSolenoid;
import edu.wpi.first.wpilibj.PneumaticsModuleType;
import edu.wpi.first.wpilibj.examples.hatchbottraditional.Constants.HatchConstants;
import edu.wpi.first.wpilibj2.command.SubsystemBase;

/** A hatch mechanism actuated by a single {@link DoubleSolenoid}. */
public class HatchSubsystem extends SubsystemBase {
  private final DoubleSolenoid m_hatchSolenoid =
      new DoubleSolenoid(
          PneumaticsModuleType.CTREPCM,
          HatchConstants.kHatchSolenoidPorts[0],
          HatchConstants.kHatchSolenoidPorts[1]);

  /** Grabs the hatch. */
  public void grabHatch() {
    m_hatchSolenoid.set(kForward);
  }

  /** Releases the hatch. */
  public void releaseHatch() {
    m_hatchSolenoid.set(kReverse);
  }

  @Override
  public void initSendable(SendableBuilder builder) {
    super.initSendable(builder);
    // Publish the solenoid state to telemetry.
    builder.addBooleanProperty("extended", () -> m_hatchSolenoid.get() == kForward, null);
  }
}
```

C++ (Header)

```
#pragma once

#include <frc/DoubleSolenoid.h>
#include <frc/PneumaticsControlModule.h>
#include <frc2/command/SubsystemBase.h>

#include "Constants.h"

class HatchSubsystem : public frc2::SubsystemBase {
 public:
  HatchSubsystem();

  // Subsystem methods go here.

  /**
   * Grabs the hatch.
   */
  void GrabHatch();

  /**
   * Releases the hatch.
   */
  void ReleaseHatch();

  void InitSendable(wpi::SendableBuilder& builder) override;

 private:
  // Components (e.g. motor controllers and sensors) should generally be
  // declared private and exposed only through public methods.
  frc::DoubleSolenoid m_hatchSolenoid;
};
```

C++ (Source)

```
#include "subsystems/HatchSubsystem.h"

#include <wpi/sendable/SendableBuilder.h>

using namespace HatchConstants;

HatchSubsystem::HatchSubsystem()
    : m_hatchSolenoid{frc::PneumaticsModuleType::CTREPCM,
                      kHatchSolenoidPorts[0], kHatchSolenoidPorts[1]} {}

void HatchSubsystem::GrabHatch() {
  m_hatchSolenoid.Set(frc::DoubleSolenoid::kForward);
}

void HatchSubsystem::ReleaseHatch() {
  m_hatchSolenoid.Set(frc::DoubleSolenoid::kReverse);
}

void HatchSubsystem::InitSendable(wpi::SendableBuilder& builder) {
```

(suite sur la page suivante)

(suite de la page précédente)

```
  SubsystemBase::InitSendable(builder);

  // Publish the solenoid state to telemetry.
  builder.AddBooleanProperty(
      "extended",
      [this] { return m_hatchSolenoid.Get() == frc::DoubleSolenoid::kForward; },
      nullptr);
}
```

Remarquez que le sous-système cache au reste du code la présence du `DoubleSolenoid` (il est déclaré `private`), et expose plutôt publiquement deux actions descriptives de plus haut niveau : `grabHatch()` et `releaseHatch()`. C'est extrêmement important que les « détails d'implémentation » comme le solénoïde double soient « cachés » de cette manière : cela garantit que le code hors du sous-système ne mettra jamais le solénoïde dans un état inattendu. Cela permet également à l'utilisateur de changer l'implémentation (par exemple, un moteur pourrait être utilisé au lieu d'un système pneumatique) sans qu'aucun code à l'extérieur du sous-système ne soit obligé d'être modifié avec lui.

Alternatively, instead of writing `void` public methods that are called from commands, we can define the public methods as factories that return a command. Consider the following from the HatchBotInlined example project (Java, C++) :

Java

```
package edu.wpi.first.wpilibj.examples.hatchbotinlined.subsystems;

import static edu.wpi.first.wpilibj.DoubleSolenoid.Value.kForward;
import static edu.wpi.first.wpilibj.DoubleSolenoid.Value.kReverse;

import edu.wpi.first.util.sendable.SendableBuilder;
import edu.wpi.first.wpilibj.DoubleSolenoid;
import edu.wpi.first.wpilibj.PneumaticsModuleType;
import edu.wpi.first.wpilibj.examples.hatchbotinlined.Constants.HatchConstants;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.SubsystemBase;

/** A hatch mechanism actuated by a single {@link edu.wpi.first.wpilibj.
 ↪DoubleSolenoid}. */
public class HatchSubsystem extends SubsystemBase {
  private final DoubleSolenoid m_hatchSolenoid =
      new DoubleSolenoid(
          PneumaticsModuleType.CTREPCM,
          HatchConstants.kHatchSolenoidPorts[0],
          HatchConstants.kHatchSolenoidPorts[1]);

  /** Grabs the hatch. */
  public Command grabHatchCommand() {
    // implicitly require `this`
    return this.runOnce(() -> m_hatchSolenoid.set(kForward));
  }

  /** Releases the hatch. */
  public Command releaseHatchCommand() {
    // implicitly require `this`
```

(suite sur la page suivante)

(suite de la page précédente)

```
    return this.runOnce(() -> m_hatchSolenoid.set(kReverse));
  }

  @Override
  public void initSendable(SendableBuilder builder) {
    super.initSendable(builder);
    // Publish the solenoid state to telemetry.
    builder.addBooleanProperty("extended", () -> m_hatchSolenoid.get() == kForward,␣
↪null);
  }
}
```

C++ (Header)

```
#pragma once

#include <frc/DoubleSolenoid.h>
#include <frc/PneumaticsControlModule.h>
#include <frc2/command/CommandPtr.h>
#include <frc2/command/SubsystemBase.h>

#include "Constants.h"

class HatchSubsystem : public frc2::SubsystemBase {
 public:
  HatchSubsystem();

  // Subsystem methods go here.

  /**
   * Grabs the hatch.
   */
  frc2::CommandPtr GrabHatchCommand();

  /**
   * Releases the hatch.
   */
  frc2::CommandPtr ReleaseHatchCommand();

  void InitSendable(wpi::SendableBuilder& builder) override;

 private:
  // Components (e.g. motor controllers and sensors) should generally be
  // declared private and exposed only through public methods.
  frc::DoubleSolenoid m_hatchSolenoid;
};
```

C++ (Source)

```cpp
#include "subsystems/HatchSubsystem.h"

#include <wpi/sendable/SendableBuilder.h>

using namespace HatchConstants;

HatchSubsystem::HatchSubsystem()
    : m_hatchSolenoid{frc::PneumaticsModuleType::CTREPCM,
                      kHatchSolenoidPorts[0], kHatchSolenoidPorts[1]} {}

frc2::CommandPtr HatchSubsystem::GrabHatchCommand() {
  // implicitly require `this`
  return this->RunOnce(
      [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kForward); });
}

frc2::CommandPtr HatchSubsystem::ReleaseHatchCommand() {
  // implicitly require `this`
  return this->RunOnce(
      [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kReverse); });
}

void HatchSubsystem::InitSendable(wpi::SendableBuilder& builder) {
  SubsystemBase::InitSendable(builder);

  // Publish the solenoid state to telemetry.
  builder.AddBooleanProperty(
      "extended",
      [this] { return m_hatchSolenoid.Get() == frc::DoubleSolenoid::kForward; },
      nullptr);
}
```

Note the qualification of the `RunOnce` factory used here : this isn't the static factory in `Commands` ! Subsystems have similar instance factories that return commands requiring `this` subsystem. Here, the `Subsystem.runOnce(Runnable)` factory (Java, C++) is used.

For a comparison between these options, see *Instance Command Factory Methods*.

26.4.3 Periodic

Subsystems have a `periodic` method that is called once every scheduler iteration (usually, once every 20 ms). This method is typically used for telemetry and other periodic actions that do not interfere with whatever command is requiring the subsystem.

Java

```
  @Override
  public void periodic() {
    // Update the odometry in the periodic block
    m_odometry.update(
        Rotation2d.fromDegrees(getHeading()),
        m_leftEncoder.getDistance(),
        m_rightEncoder.getDistance());
    m_fieldSim.setRobotPose(getPose());
  }
```

C++ (Header)

```
  void Periodic() override;
```

C++ (Source)

```
void DriveSubsystem::Periodic() {
  // Implementation of subsystem periodic method goes here.
  m_odometry.Update(m_gyro.GetRotation2d(),
                    units::meter_t{m_leftEncoder.GetDistance()},
                    units::meter_t{m_rightEncoder.GetDistance()});
  m_fieldSim.SetRobotPose(m_odometry.GetPose());
}
```

There is also a `simulationPeriodic()` method that is similar to `periodic()` except that it is only run during *Simulation* and can be used to update the state of the robot.

26.4.4 Default Commands

Note : In the C++ command-based library, the CommandScheduler *owns* the default command object.

« Default commands » are commands that run automatically whenever a subsystem is not being used by another command. This can be useful for « background » actions such as controlling the robot drive, or keeping an arm held at a setpoint.

Setting a default command for a subsystem is very easy ; one simply calls `CommandScheduler.getInstance().setDefaultCommand()`, or, more simply, the `setDefaultCommand()` method of the `Subsystem` interface :

JAVA

```
CommandScheduler.getInstance().setDefaultCommand(exampleSubsystem, exampleCommand);
```

C++

```
CommandScheduler.GetInstance().SetDefaultCommand(exampleSubsystem,␣
↪std::move(exampleCommand));
```

JAVA

```
exampleSubsystem.setDefaultCommand(exampleCommand);
```

C++

```
exampleSubsystem.SetDefaultCommand(std::move(exampleCommand));
```

Note : Une commande affectée comme commande par défaut pour un sous-système doit requérir ce sous-système.

26.5 Liaison de commandes à des déclencheurs

Mis à part les commandes autonomes, qui sont planifiées au début de la période autonome, et les commandes par défaut, qui sont automatiquement planifiées chaque fois que leur sous-système n'est pas actuellement en cours d'utilisation, la façon la plus courante d'exécuter une commande est de la lier à un événement déclencheur, comme un bouton appuyé par un opérateur humain. Le paradigme orienté commande rend cela extrêmement facile à faire.

As mentioned earlier, command-based is a *declarative programming* paradigm. Accordingly, binding buttons to commands is done declaratively ; the association of a button and a command is « declared » once, during robot initialization. The library then does all the hard work of checking the button state and scheduling (or canceling) the command as needed, behind-the-scenes. Users only need to worry about designing their desired UI setup - not about implementing it !

Command binding is done through the `Trigger` class (Java, C++).

26.5.1 Getting a Trigger Instance

To bind commands to conditions, we need a `Trigger` object. There are three ways to get a Trigger object :

HID Factories

The command-based HID classes contain factory methods returning a `Trigger` for a given button. `CommandGenericHID` has an index-based `button(int)` factory (Java, C++), and its subclasses `CommandXboxController` (Java, C++), `CommandPS4Controller` (Java, C++), and `CommandJoystick` (Java, C++) have named factory methods for each button.

JAVA

```
CommandXboxController exampleCommandController = new CommandXboxController(1); //␣
↪Creates a CommandXboxController on port 1.
Trigger xButton = exampleCommandController.x(); // Creates a new Trigger object for␣
↪the `X` button on exampleCommandController
```

C++

```
frc2::CommandXboxController exampleCommandController{1} // Creates a␣
↪CommandXboxController on port 1
frc2::Trigger xButton = exampleCommandController.X() // Creates a new Trigger object␣
↪for the `X` button on exampleCommandController
```

JoystickButton

Alternatively, the *regular HID classes* can be used and passed to create an instance of `JoystickButton` (Java, C++), a constructor-only subclass of `Trigger` :

JAVA

```
XboxController exampleController = new XboxController(2); // Creates an␣
↪XboxController on port 2.
Trigger yButton = new JoystickButton(exampleController, XboxController.Button.kY.
↪value); // Creates a new JoystickButton object for the `Y` button on␣
↪exampleController
```

C++

```
frc::XboxController exampleController{2} // Creates an XboxController on port 2
frc2::JoystickButton yButton(&exampleStick, frc::XboxController::Button::kY); //␣
↪Creates a new JoystickButton object for the `Y` button on exampleController
```

Arbitrary Triggers

While binding to HID buttons is by far the most common use case, users may want to bind commands to arbitrary triggering events. This can be done inline by passing a lambda to the constructor of `Trigger` :

JAVA

```
DigitalInput limitSwitch = new DigitalInput(3); // Limit switch on DIO 3

Trigger exampleTrigger = new Trigger(limitSwitch::get);
```

C++

```
frc::DigitalInput limitSwitch{3}; // Limit switch on DIO 3

frc2::Trigger exampleTrigger([&limitSwitch] { return limitSwitch.Get(); });
```

26.5.2 Trigger Bindings

Note : The C++ command-based library offers two overloads of each button binding method - one that takes an rvalue reference (`CommandPtr&&`), and one that takes a raw pointer (`Command*`). The rvalue overload moves ownership to the scheduler, while the raw pointer overload leaves the user responsible for the lifespan of the command object. It is recommended that users preferentially use the rvalue reference overload unless there is a specific need to retain a handle to the command in the calling code.

There are a number of bindings available for the `Trigger` class. All of these bindings will automatically schedule a command when a certain trigger activation event occurs - however, each binding has different specific behavior.

`Trigger` objects *do not need to survive past the call to a binding method*, so the binding methods may be simply called on a temp. Remember that button binding is *declarative* : bindings only need to be declared once, ideally some time during robot initialization. The library handles everything else.

Note : The `Button` subclass is deprecated, and usage of its binding methods should be replaced according to the respective deprecation messages in the API docs.

---

onTrue

This binding schedules a command when a trigger changes from `false` to `true` (or, accordingly, when a button changes is initially pressed). The command will be scheduled on the iteration when the state changes, and will not be scheduled again unless the trigger becomes `false` and then `true` again (or the button is released and then re-pressed).

JAVA

```
52    // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
53    m_driverController.a().onTrue(m_robotArm.setArmGoalCommand(2));
```

C++

```
25  // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
26  m_driverController.A().OnTrue(m_arm.SetArmGoalCommand(2_rad));
```

The `onFalse` binding is identical, only that it schedules on `false` instead of on `true`.

whileTrue

This binding schedules a command when a trigger changes from `false` to `true` (or, accordingly, when a button is initially pressed) and cancels it when the trigger becomes `false` again (or the button is released). The command will *not* be re-scheduled if it finishes while the trigger is still `true`. For the command to restart if it finishes while the trigger is `true`, wrap the command in a `RepeatCommand`, or use a `RunCommand` instead of an `InstantCommand`.

JAVA

```
114   // While holding the shoulder button, drive at half speed
115   new JoystickButton(m_driverController, Button.kRightBumper.value)
116       .whileTrue(new HalveDriveSpeed(m_robotDrive));
```

C++

```
75  // While holding the shoulder button, drive at half speed
76  frc2::JoystickButton(&m_driverController,
77                       frc::XboxController::Button::kRightBumper)
78      .WhileTrue(HalveDriveSpeed(&m_drive).ToPtr());
```

The `whileFalse` binding is identical, only that it schedules on `false` and cancels on `true`.

toggleOnTrue

This binding toggles a command, scheduling it when a trigger changes from `false` to `true` (or a button is initially pressed), and canceling it under the same condition if the command is currently running. Note that while this functionality is supported, toggles are not a highly-recommended option for user control, as they require the driver to keep track of the robot state. The preferred method is to use two buttons ; one to turn on and another to turn off. Using a StartEndCommand or a ConditionalCommand is a good way to specify the commands that you want to be want to be toggled between.

JAVA

```
myButton.toggleOnTrue(Commands.startEnd(mySubsystem::onMethod,
    mySubsystem::offMethod,
    mySubsystem));
```

C++

```
myButton.ToggleOnTrue(frc2::cmd::StartEnd([&] { mySubsystem.OnMethod(); },
    [&] { mySubsystem.OffMethod(); },
    {&mySubsystem}));
```

The `toggleOnFalse` binding is identical, only that it toggles on `false` instead of on `true`.

26.5.3 Chaining Calls

It is useful to note that the command binding methods all return the trigger that they were called on, and thus can be chained to bind multiple commands to different states of the same trigger. For example :

JAVA

```
exampleButton
    // Binds a FooCommand to be scheduled when the button is pressed
    .onTrue(new FooCommand())
    // Binds a BarCommand to be scheduled when that same button is released
    .onFalse(new BarCommand());
```

C++

```
exampleButton
    // Binds a FooCommand to be scheduled when the button is pressed
    .OnTrue(FooCommand().ToPtr())
    // Binds a BarCommand to be scheduled when that same button is released
    .OnFalse(BarCommand().ToPtr());
```

26.5.4 Composition de déclencheurs

The `Trigger` class can be composed to create composite triggers through the `and()`, `or()`, and `negate()` methods (or, in C++, the &&, ||, and ! operators). For example :

JAVA

```
// Binds an ExampleCommand to be scheduled when both the 'X' and 'Y' buttons of the␣
↪driver gamepad are pressed
exampleCommandController.x()
    .and(exampleCommandController.y())
    .onTrue(new ExampleCommand());
```

C++

```
// Binds an ExampleCommand to be scheduled when both the 'X' and 'Y' buttons of the␣
↪driver gamepad are pressed
(exampleCommandController.X()
    && exampleCommandController.Y())
    .OnTrue(ExampleCommand().ToPtr());
```

26.5.5 Déclencheurs anti-rebonds

Pour éviter une activation répétée rapide, les déclencheurs (en particulier ceux provenant d'entrées numériques) peuvent être traités par des filtres anti-rebonds à l'aide de la *classe Debouncer de la WPILIb* par l'intérmédiaire de sa méthode *debounce* :

JAVA

```
// debounces exampleButton with a 0.1s debounce time, rising edges only
exampleButton.debounce(0.1).onTrue(new ExampleCommand());

// debounces exampleButton with a 0.1s debounce time, both rising and falling edges
exampleButton.debounce(0.1, Debouncer.DebounceType.kBoth).onTrue(new␣
↪ExampleCommand());
```

C++

```
// debounces exampleButton with a 100ms debounce time, rising edges only
exampleButton.Debounce(100_ms).OnTrue(ExampleCommand().ToPtr());

// debounces exampleButton with a 100ms debounce time, both rising and falling edges
exampleButton.Debounce(100_ms, Debouncer::DebounceType::Both).OnTrue(ExampleCommand().
↪ToPtr());
```

26.6 Structurer un projet de robot orienté commande

Alors que les utilisateurs sont libres d'utiliser les bibliothèques orientées commande comme ils le veulent (et les utilisateurs avancés sont encouragés à le faire), les nouveaux utilisateurs pourraient avoir besoin de conseils sur la manière de structurer un projet de robot orienté commande de base.

Un modèle standard pour un projet de robot orienté commande est inclus dans le dépôt d'exemples WPILib (Java, C++). Cette section guidera les utilisateurs à travers la structure de ce modèle.

Le package/répertoire racine contient généralement quatre classes :

`Main`, qui est l'application principale du robot (Java seulement). Les nouveaux utilisateurs *ne devraient pas* toucher à cette classe. `Robot`, qui est responsable du flux de contrôle principal du code robot. `RobotContainer`, qui contient des sous-systèmes et des commandes du robot, et c'est là que l'essentielle de la configuration déclarative du robot (par exemple la liaison des boutons avec les commandes) est effectuée. `Constants`, qui renferme des constantes accessibles globalement, donc à utiliser dans l'ensemble du programme du robot.

Le répertoire racine contiendra aussi deux sous-packages/sous-répertoires : `Subsystems` contiendra toutes les classes sous-systèmes définies par l'utilisateur. `Commands` contiendra tout les classes commandes définies par l'utilisateur.

26.6.1 Robot

Alors que `Robot` (Java, C++ (En-tête), C++ (Source)) est responsable du flux de contrôle du programme et la programmation orientée commande est un paradigme déclaratif conçu pour minimiser l'attention que l'utilisateur doit porter au flux de contrôle du programme explicite, la classe `Robot` d'un projet orienté commande devrait être presque vide. Cependant, il y a quelques éléments importants qui doivent être inclus.

Java

```
22  /**
23   * This function is run when the robot is first started up and should be used for␣
↪any
24   * initialization code.
25   */
26  @Override
27  public void robotInit() {
28    // Instantiate our RobotContainer.  This will perform all our button bindings,␣
↪and put our
29    // autonomous chooser on the dashboard.
30    m_robotContainer = new RobotContainer();
31  }
```

En Java, une instance de `RobotContainer` devrait être construite dans la méthode `robotInit()` - cela est important, car l'essentiel de la configuration déclarative du robot sera appelé depuis le constructeur `RobotContainer`.

En C++, ce n'est pas nécessaire, car RobotContainer est un membre valeur et va être construit durant la construction de `Robot`.

Java

```
  /**
   * This function is called every 20 ms, no matter the mode. Use this for items like diagnostics
   * that you want ran during disabled, autonomous, teleoperated and test.
   *
   * <p>This runs after the mode specific periodic functions, but before LiveWindow and
   * SmartDashboard integrated updating.
   */
  @Override
  public void robotPeriodic() {
    // Runs the Scheduler.  This is responsible for polling buttons, adding newly-scheduled
    // commands, running already-scheduled commands, removing finished or interrupted commands,
    // and running subsystem periodic() methods.  This must be called from the robot's periodic
    // block in order for anything in the Command-based framework to work.
    CommandScheduler.getInstance().run();
  }
```

C++ (Source)

```
/**
 * This function is called every 20 ms, no matter the mode. Use
 * this for items like diagnostics that you want to run during disabled,
 * autonomous, teleoperated and test.
 *
 * <p> This runs after the mode specific periodic functions, but before
 * LiveWindow and SmartDashboard integrated updating.
 */
void Robot::RobotPeriodic() {
  frc2::CommandScheduler::GetInstance().Run();
}
```

L'inclusion de l'appel `CommandScheduler.getInstance().run()` dans la méthode `robotPeriodic()` est essentielle ; sans cela, le planificateur n'exécutera pas les commandes programmées. Puisque `TimedRobot` s'exécute à la fréquence de la boucle principale (50Hz par défaut), c'est la fréquence à laquelle les commandes périodiques et les méthodes de sous-systèmes s'exécuteront. Il n'est pas recommandé pour les utilisateurs novices d'appeler cette méthode ailleurs dans leur code.

Java

```
  /** This autonomous runs the autonomous command selected by your {@link␣
↪RobotContainer} class. */
  @Override
  public void autonomousInit() {
    m_autonomousCommand = m_robotContainer.getAutonomousCommand();

    // schedule the autonomous command (example)
    if (m_autonomousCommand != null) {
      m_autonomousCommand.schedule();
    }
  }
```

C++ (Source)

```
/**
 * This autonomous runs the autonomous command selected by your {@link
 * RobotContainer} class.
 */
void Robot::AutonomousInit() {
  m_autonomousCommand = m_container.GetAutonomousCommand();

  if (m_autonomousCommand) {
    m_autonomousCommand->Schedule();
  }
}
```

La méthode `autonomousInit()` planifie une commande autonome retournée par l'instance de `RobotContainer`. La logique pour sélectionner quelle commande autonome sera exécutée peut être gérée à l'intérieur de `RobotContainer`.

Java

```
  @Override
  public void teleopInit() {
    // This makes sure that the autonomous stops running when
    // teleop starts running. If you want the autonomous to
    // continue until interrupted by another command, remove
    // this line or comment it out.
    if (m_autonomousCommand != null) {
      m_autonomousCommand.cancel();
    }
  }
```

C++ (Source)

```
46 void Robot::TeleopInit() {
47   // This makes sure that the autonomous stops running when
48   // teleop starts running. If you want the autonomous to
49   // continue until interrupted by another command, remove
50   // this line or comment it out.
51   if (m_autonomousCommand) {
52     m_autonomousCommand->Cancel();
53   }
54 }
```

La méthode `teleopInit()` annulera toutes les commandes du mode autonome qui sont en train d'être exécutées. C'est généralement une bonne pratique.

Les utilisateurs avancés peuvent ajouter du code additionnel aux méthodes `init` et `periodic` comme ils l'entendent; cependant, il faut noter que le fait d'inclure de grandes quantités de code robot impératif dans `Robot.java` va à l'encontre de la philosophie de conception déclarative du paradigme orienté commande, et peut entraîner un code structuré de manière confuse/désorganisée.

26.6.2 RobotContainer

Cette classe (Java, C++ (En-tête), C++ (Source)) est où la plupart de la configuration d'un robot orienté commande sera effectuée. Dans cette classe, on définit les sous-systèmes et commandes du robot, lie ces commandes aux événements déclencheurs (tels que les boutons), et spécifie quelle commande sera exécutée dans la routine autonome. Il y a quelques aspects de cette classe pour lesquels de nouveaux utilisateurs pourraient avoir besoin d'explications :

Java

```
23   private final ExampleSubsystem m_exampleSubsystem = new ExampleSubsystem();
```

C++ (Header)

```
32   ExampleSubsystem m_subsystem;
```

Notez que les sous-systèmes sont déclarés comme des champs privés dans `RobotContainer`. Ceci est en contraste notable avec l'incarnation précédente du cadre orienté commande, mais est beaucoup plus aligné sur les meilleures pratiques orientées objet convenues. Si les sous-systèmes sont déclarés comme des variables globales, cela permet à l'utilisateur d'y accéder n'importe où dans le code. Alors que cela peut faciliter certaines choses (par exemple, il n'y aurait pas besoin de passer les sous-systèmes aux commandes pour que ces commandes puissent y avoir accès), cela rend aussi le flux de contrôle du programme beaucoup plus difficile à suivre, car on ne peut pas voir immédiatement quelles parties du code pourraient changer ou pourraient être changées par d'autres parties du code. Cela peut également empêcher le système de gestion de ressources de faire son travail, car la facilité d'accès rend facile pour un utilisateur de faire un appel à une méthode de sous-systèmes hors des commandes gérées par les ressources.

Java

```
61      return Autos.exampleAuto(m_exampleSubsystem);
```

C++ (Source)

```
34    return autos::ExampleAuto(&m_subsystem);
```

Puisque les sous-systèmes sont déclarés comme des membres privés, ils doivent être passés explicitement aux commandes (un modèle nommé « dependency injection ») pour que les commandes puissent appeler leurs méthodes. Cela est fait ici avec `ExampleCommand`, qui se fait passer un pointeur vers un `ExampleSubsystem`.

Java

```
35    /**
36     * Use this method to define your trigger->command mappings. Triggers can be␣
↪created via the
37     * {@link Trigger#Trigger(java.util.function.BooleanSupplier)} constructor with an␣
↪arbitrary
38     * predicate, or via the named factories in {@link
39     * edu.wpi.first.wpilibj2.command.button.CommandGenericHID}'s subclasses for {@link
40     * CommandXboxController Xbox}/{@link edu.wpi.first.wpilibj2.command.button.
↪CommandPS4Controller
41     * PS4} controllers or {@link edu.wpi.first.wpilibj2.command.button.CommandJoystick␣
↪Flight
42     * joysticks}.
43     */
44    private void configureBindings() {
45      // Schedule `ExampleCommand` when `exampleCondition` changes to `true`
46      new Trigger(m_exampleSubsystem::exampleCondition)
47          .onTrue(new ExampleCommand(m_exampleSubsystem));
48
49      // Schedule `exampleMethodCommand` when the Xbox controller's B button is pressed,
50      // cancelling on release.
51      m_driverController.b().whileTrue(m_exampleSubsystem.exampleMethodCommand());
52    }
```

C++ (Source)

```
19  void RobotContainer::ConfigureBindings() {
20    // Configure your trigger bindings here
21
22    // Schedule `ExampleCommand` when `exampleCondition` changes to `true`
23    frc2::Trigger([this] {
24      return m_subsystem.ExampleCondition();
25    }).OnTrue(ExampleCommand(&m_subsystem).ToPtr());
26
27    // Schedule `ExampleMethodCommand` when the Xbox controller's B button is
28    // pressed, cancelling on release.
```

(suite sur la page suivante)

(suite de la page précédente)

```
  m_driverController.B().WhileTrue(m_subsystem.ExampleMethodCommand());
}
```

As mentioned before, the `RobotContainer()` constructor is where most of the declarative setup for the robot should take place, including button bindings, configuring autonomous selectors, etc. If the constructor gets too « busy, » users are encouraged to migrate code into separate subroutines (such as the `configureBindings()` method included by default) which are called from the constructor.

Java

```
  /**
   * Use this to pass the autonomous command to the main {@link Robot} class.
   *
   * @return the command to run in autonomous
   */
  public Command getAutonomousCommand() {
    // An example command will be run in autonomous
    return Autos.exampleAuto(m_exampleSubsystem);
  }
}
```

C++ (Source)

```
frc2::CommandPtr RobotContainer::GetAutonomousCommand() {
  // An example command will be run in autonomous
  return autos::ExampleAuto(&m_subsystem);
}
```

Finalement, la méthode `getAutonomousCommand()` fournit un moyen pratique à l'utilisateur pour envoyer sa commande autonome sélectionnée à la classe `Robot` principale (qui a besoin d'y accèder pour la planifier au début de la période autonome).

26.6.3 Constants

La classe `Constants` (Java, C++ (En-tête)) (en C++, ce n'est pas une classe, mais simplement un fichier en-tête dans lequel plusieurs espaces de noms sont définis) est où les constantes du robot accessibles globalement (comme les vitesses, les facteurs de conversion d'unités, les gains PID, les ports de capteurs/moteurs, etc.) peuvent être stockées. Il est recommandé que les utilisateurs séparent ces constantes en classes internes individuelles qui correspondent aux sous-systèmes ou aux modes du robot pour garder le nom des variables le plus court possible.

En Java, toutes les constantes devraient être déclarées `public static final` pour qu'elles soient accessibles globalement et ne puissent pas être changées. En C++, toutes les constantes devraient être `constexpr`.

Pour des exemples plus illustratifs de ce à quoi devrait ressembler une classe `Constants` en pratique, voir les classes des projets exemples orientés commande :

— FrisbeeBot (Java, C++)

— GyroDriveCommands (Java, C++)
— Hatchbot (Java, C++)
— RapidReactCommandBot (Java, C++)

En java, il est recommandé que les constantes soient utilisées à partir d'autres classes par l'importation statique de la classe intérieure nécessaire. Une déclaration `import static` importe l'espace de nom statique d'une classe dans la classe dans laquelle vous travaillez pour que les constantes `static` puissent être référencées directement comme si elles étaient définies dans cette classe. En C++, la même chose peut être faite avec `using namespace` :

JAVA

```
import static edu.wpi.first.wpilibj.templates.commandbased.Constants.OIConstants.*;
```

C++

```
using namespace OIConstants;
```

26.6.4 Subsystems

Les sous-systèmes définis par l'utilisateur devraient entrer dans ce package/répertoire.

26.6.5 Commands

Les commandes définies par l'utilisateur devraient entrer dans cet package/répertoire.

26.7 Organizing Command-Based Robot Projects

As robot code becomes more complicated, navigating, understanding, and maintaining the code takes up more and more time and energy. Making changes to the code often becomes more difficult, sometimes for reasons that have very little to do with the actual complexity of the underlying logic. For a simplified example : putting the logic for many unrelated robot functions into a single 1000-line file makes it difficult to find a specific piece of code within that file, particularly under stress at a competition. But spreading out closely related logic across dozens of tiny files is often just as difficult to navigate.

This is not a problem unique to FRC, and in fact, good organization only becomes more and more critical as software projects become bigger and bigger. The « best » organization system is a perennial topic of debate, much like the « best » programming language, but in the end, the choice (in both cases) comes down to the specific task at hand and the programmer (or programmers) implementing said task. Even in the relatively small space of FRC robot programming, there is no right answer. The best choice for a given team will depend on the nature of the specific robot code, team structure, and pure personal preference.

This article discusses various facets of command-based robot program design that advanced FRC programmers may want to be aware of when writing code. It is not a prescriptive tutorial, though it presents some recommended best practices. If this level of choice seems daunting, however, many teams have been highly successful while sticking closely to WPILib's

example code and guidelines. However, this discussion may be of interest to intermediate and advanced programmers who want to make their code not only effective, but flexible, easily changeable, and sometimes even beautiful.

26.7.1 Why Care About Organization ?

Good code organization will rarely make or break a team's competitive ability—but it does mean easier debugging, faster modifications, nicer-looking code, and happier programmers. While it's impossible to define « good » organization by way of what the code looks like from the inside, it's easier to define in terms of what the robot's software looks like from the outside.

What Good Organization Looks Like

When code is well-designed and well-organized, the code's internal structure is intuitive and easily comprehensible. Cumbersome boilerplate is minimized, meaning that new robot functionality can often be added with just a few lines of code. When a constant value (such as the speed of the robot's intake) needs to be changed, it only needs to change in one place. If multiple programmers are working together, they can easily understand each others" work. Bugs are rare, since it is difficult to accidentally introduce unintended behavior (such as creating a command that does not require necessary subsystems). Implementing more advanced functions like unit tests is easier, since the code is abstracted away from the physical hardware. Programmers are happy (most of the time).

What Bad Organization Looks Like

Poorly organized code often has internal structure that makes little to no sense, even to whoever wrote it. When functionality has to be added or changed, it often breaks unrelated parts of the robot : adding automatic shooter control might introduce a bug in the climbing sequence for unclear reasons. Alternatively, the organizational framework might be so strict that it's impossible to implement necessary behavior, requiring nasty hacks or workarounds. Many lines of boilerplate code are needed for simple robot logic. Constants are scattered across the codebase, and changing basic behavior often requires making the same change to many different files. Collaboration among multiple programmers is difficult or impossible.

26.7.2 Defining Commands

In larger robot codebases, multiple copies of the same command need to be used in many different places. For instance, a command that runs a robot's intake might be used in teleop, bound to a certain button ; as part of a complicated command group for an autonomous routine ; and as part of a self-test sequence.

As an example, let's look at some ways to define a simple command that simply runs the robot's intake forward at full power until canceled.

Inline Commands

The easiest and most expressive way to do this is with a `StartEndCommand` :

JAVA

```
Command runIntake = Commands.startEnd(() -> intake.set(1), () -> intake.set(0),␣
↪intake);
```

C++

```
frc2::CommandPtr runIntake = frc2::cmd::StartEnd([&intake] { intake.Set(1.0); }, [&
↪intake] { intake.Set(0); }, {&intake});
```

This is sufficient for commands that are only used once. However, for a command like this that might get used in many different autonomous routines and button bindings, inline commands everywhere means a lot of repetitive code :

JAVA

```
// RobotContainer.java
intakeButton.whileTrue(Commands.startEnd(() -> intake.set(1.0), () -> intake.set(0),␣
↪intake));

Command intakeAndShoot = Commands.startEnd(() -> intake.set(1.0), () -> intake.set(0),
↪ intake)
    .alongWith(new RunShooter(shooter));

Command autonomousCommand = Commands.sequence(
    Commands.startEnd(() -> intake.set(1.0), () -> intake.set(0.0), intake).
↪withTimeout(5.0),
    Commands.waitSeconds(3.0),
    Commands.startEnd(() -> intake.set(1.0), () -> intake.set(0.0), intake).
↪withTimeout(5.0)
);
```

C++

```
intakeButton.WhileTrue(frc2::cmd::StartEnd([&intake] { intake.Set(1.0); }, [&intake]
↪{ intake.Set(0); }, {&intake}));

frc2::CommandPtr intakeAndShoot = frc2::cmd::StartEnd([&intake] { intake.Set(1.0); },␣
↪[&intake] { intake.Set(0); }, {&intake})
    .AlongWith(RunShooter(&shooter).ToPtr());

frc2::CommandPtr autonomousCommand = frc2::cmd::Sequence(
  frc2::cmd::StartEnd([&intake] { intake.Set(1.0); }, [&intake] { intake.Set(0); }, {&
↪intake}).WithTimeout(5.0_s),
  frc2::cmd::Wait(3.0_s),
```

(suite sur la page suivante)

(suite de la page précédente)

```
  frc2::cmd::StartEnd([&intake] { intake.Set(1.0); }, [&intake] { intake.Set(0); }, {&
↪intake}).WithTimeout(5.0_s)
);
```

Creating one `StartEndCommand` instance and putting it in a variable won't work here, since once an instance of a command is added to a command group it is effectively « owned » by that command group and cannot be used in any other context.

Instance Command Factory Methods

One way to solve this quandary is using the « factory method » design pattern : a function that returns a new object every invocation, according to some specification. Using *command composition*, a factory method can construct a complex command object with merely a few lines of code.

For example, a command like the intake-running command is conceptually related to exactly one subsystem : the `Intake`. As such, it makes sense to put a `runIntakeCommand` method as an instance method of the `Intake` class :

Note : In this document we will name factory methods as `lowerCamelCaseCommand`, but teams may decide on other conventions. In general, it is recommended to end the method name with `Command` if it might otherwise be confused with an ordinary method (e.g. `intake.run` might be the name of a method that simply turns on the intake).

JAVA

```
public class Intake extends SubsystemBase {
    // [code for motor controllers, configuration, etc.]
    // ...

    public Command runIntakeCommand() {
      // implicitly requires `this`
      return this.startEnd(() -> this.set(1.0), () -> this.set(0.0));
    }
}
```

C++

```
frc2::CommandPtr Intake::RunIntakeCommand() {
  // implicitly requires `this`
  return this->StartEnd([this] { this->Set(1.0); }, [this] { this->Set(0); });
}
```

Notice how since we are in the `Intake` class, we no longer refer to `intake`; instead, we use the `this` keyword to refer to the current instance.

Since we are inside the `Intake` class, technically we can access `private` variables and methods directly from within the `runIntakeCommand` method, thus not needing intermediary methods. (For example, the `runIntakeCommand` method can directly interface with the motor

controller objects instead of calling `set()`.) On the other hand, these intermediary methods can reduce code duplication and increase encapsulation. Like many other choices outlined in this document, this tradeoff is a matter of personal preference on a case-by-case basis.

Using this new factory method in command groups and button bindings is highly expressive :

JAVA

```
intakeButton.whileTrue(intake.runIntakeCommand());

Command intakeAndShoot = intake.runIntakeCommand().alongWith(new RunShooter(shooter));

Command autonomousCommand = Commands.sequence(
    intake.runIntakeCommand().withTimeout(5.0),
    Commands.waitSeconds(3.0),
    intake.runIntakeCommand().withTimeout(5.0)
);
```

C++

```
intakeButton.WhileTrue(intake.RunIntakeCommand());

frc2::CommandPtr intakeAndShoot = intake.RunIntakeCommand().AlongWith(RunShooter(&
↪shooter).ToPtr());

frc2::CommandPtr autonomousCommand = frc2::cmd::Sequence(
  intake.RunIntakeCommand().WithTimeout(5.0_s),
  frc2::cmd::Wait(3.0_s),
  intake.RunIntakeCommand().WithTimeout(5.0_s)
);
```

Adding a parameter to the `runIntakeCommand` method to provide the exact percentage to run the intake is easy and allows for even more flexibility.

JAVA

```
public Command runIntakeCommand(double percent) {
    return new StartEndCommand(() -> this.set(percent), () -> this.set(0.0), this);
}
```

C++

```
frc2::CommandPtr Intake::RunIntakeCommand() {
  // implicitly requires `this`
  return this->StartEnd([this, percent] { this->Set(percent); }, [this] { this->
↪Set(0); });
}
```

For instance, this code creates a command group that runs the intake forwards for two seconds, waits for two seconds, and then runs the intake backwards for five seconds.

JAVA

```
Command intakeRunSequence = intake.runIntakeCommand(1.0).withTimeout(2.0)
    .andThen(Commands.waitSeconds(2.0))
    .andThen(intake.runIntakeCommand(-1.0).withTimeout(5.0));
```

C++

```
frc2::CommandPtr intakeRunSequence = intake.RunIntakeCommand(1.0).WithTimeout(2.0_s)
    .AndThen(frc2::cmd::Wait(2.0_s))
    .AndThen(intake.RunIntakeCommand(-1.0).WithTimeout(5.0_s));
```

This approach is recommended for commands that are conceptually related to only a single subsystem, and is very concise. However, it doesn't fare well with commands related to more than one subsystem : passing in other subsystem objects is unintuitive and can cause race conditions and circular dependencies, and thus should be avoided. Therefore, this approach is best suited for single-subsystem commands, and should be used only for those cases.

Static Command Factories

Instance factory methods work great for single-subsystem commands. However, complicated robot actions (like the ones often required during the autonomous period) typically need to coordinate multiple subsystems at once. When we want to define an inline command that uses multiple subsystems, it doesn't make sense for the command factory to live in any single one of those subsystems. Instead, it can be cleaner to define the command factory methods statically in some external class :

Note : The `sequence` and `parallel` static factories construct sequential and parallel command groups : this is equivalent to the `andThen` and `alongWith` decorators, but can be more readable. Their use is a matter of personal preference.

JAVA

```
public class AutoRoutines {

    public static Command driveAndIntake(Drivetrain drivetrain, Intake intake) {
        return Commands.sequence(
            Commands.parallel(
                drivetrain.driveCommand(0.5, 0.5),
                intake.runIntakeCommand(1.0)
            ).withTimeout(5.0),
            Commands.parallel(
              drivetrain.stopCommand();
              intake.stopCommand();
            )
        );
    }
}
```

C++

```
// TODO
```

Non-Static Command Factories

If we want to avoid the verbosity of adding required subsystems as parameters to our factory methods, we can instead construct an instance of our `AutoRoutines` class and inject our subsystems through the constructor :

JAVA

```
public class AutoRoutines {

    private Drivetrain drivetrain;

    private Intake intake;

    public AutoRoutines(Drivetrain drivetrain, Intake intake) {
      this.drivetrain = drivetrain;
      this.intake = intake;
    }

    public Command driveAndIntake() {
        return Commands.sequence(
            Commands.parallel(
                drivetrain.driveCommand(0.5, 0.5),
                intake.runIntakeCommand(1.0)
            ).withTimeout(5.0),
            Commands.parallel(
              drivetrain.stopCommand();
              intake.stopCommand();
            )
        );
    }

    public Command driveThenIntake() {
        return Commands.sequence(
            drivetrain.driveCommand(0.5, 0.5).withTimeout(5.0),
            drivetrain.stopCommand(),
            intake.runIntakeCommand(1.0).withTimeout(5.0),
            intake.stopCommand()
        );
    }
}
```

C++

```
// TODO
```

Then, elsewhere in our code, we can instantiate an single instance of this class and use it to produce several commands :

JAVA

```
AutoRoutines autoRoutines = new AutoRoutines(this.drivetrain, this.intake);

Command driveAndIntake = autoRoutines.driveAndIntake();
Command driveThenIntake = autoRoutines.driveThenIntake();

Command drivingAndIntakingSequence = Commands.sequence(
  autoRoutines.driveAndIntake(),
  autoRoutines.driveThenIntake()
);
```

C++

```
// TODO
```

Capturing State in Inline Commands

Inline commands are extremely concise and expressive, but do not offer explicit support for commands that have their own internal state (such as a drivetrain trajectory following command, which may encapsulate an entire controller). This is often accomplished by instead writing a Command class, which will be covered later in this article.

However, it is still possible to ergonomically write a stateful command composition using inline syntax, so long as we are working within a factory method. To do so, we declare the state as a method local and « capture » it in our inline definition. For example, consider the following instance command factory to turn a drivetrain to a specific angle with a PID controller :

Note : The `Subsystem.run` and `Subsystem.runOnce` factory methods sugar the creation of a `RunCommand` and an `InstantCommand` requiring `this` subsystem.

JAVA

```
public Command turnToAngle(double targetDegrees) {
    // Create a controller for the inline command to capture
    PIDController controller = new PIDController(Constants.kTurnToAngleP, 0, 0);
    // We can do whatever configuration we want on the created state before returning␣
↪from the factory
    controller.setPositionTolerance(Constants.kTurnToAngleTolerance);

    // Try to turn at a rate proportional to the heading error until we're at the␣
↪setpoint, then stop
    return run(() -> arcadeDrive(0,-controller.calculate(gyro.getHeading(),␣
↪targetDegrees)))
        .until(controller::atSetpoint)
        .andThen(runOnce(() -> arcadeDrive(0, 0)));
}
```

C++

```
// TODO
```

This pattern works very well in Java so long as the captured state is « effectively final » - i.e., it is never reassigned. This means that we cannot directly define and capture primitive types (e.g. *int*, *double*, *boolean*) - to circumvent this, we need to wrap any state primitives in a mutable container type (the same way *PIDController* wraps its internal *kP*, *kI*, and *kD* values).

Writing Command Classes

Another possible way to define reusable commands is to write a class that represents the command. This is typically done by subclassing either `Command` or one of the `CommandGroup` classes.

Subclassing Command

Returning to our simple intake command from earlier, we could do this by creating a new subclass of `Command` that implements the necessary `initialize` and `end` methods.

JAVA

```
public class RunIntakeCommand extends Command {
    private Intake m_intake;

    public RunIntakeCommand(Intake intake) {
        this.m_intake = intake;
        addRequirements(intake);
    }

    @Override
```

(suite sur la page suivante)

(suite de la page précédente)

```
    public void initialize() {
        m_intake.set(1.0);
    }

    @Override
    public void end(boolean interrupted) {
        m_intake.set(0.0);
    }

    // execute() defaults to do nothing
    // isFinished() defaults to return false
}
```

C++

```
// TODO
```

This, however, is just as cumbersome as the original repetitive code, if not more verbose. The only two lines that really matter in this entire file are the two calls to `intake.set()`, yet there are over 20 lines of boilerplate code ! Not to mention, doing this for a lot of robot actions quickly clutters up a robot project with dozens of small files. Nevertheless, this might feel more « natural, » particularly for programmers who prefer to stick closely to an object-oriented model.

This approach should be used for commands with internal state (not subsystem state !), as the class can have fields to manage said state. It may also be more intuitive to write commands with complex logic as classes, especially for those less experienced with command composition. As the command is detached from any specific subsystem class and the required subsystem objects are injected through the constructor, this approach deals well with commands involving multiple subsystems.

Subclassing Command Groups

If we wish to write composite commands as their own classes, we may write a constructor-only subclass of the most exterior group type. For example, an intake-then-outtake sequence (with single-subsystem commands defined as instance factory methods) can look like this :

JAVA

```
public class IntakeThenOuttake extends SequentialCommandGroup {
    public IntakeThenOuttake(Intake intake) {
        super(
            intake.runIntakeCommand(1.0).withTimeout(2.0),
            new WaitCommand(2.0),
            intake.runIntakeCommand(-1).withTimeout(5.0)
        );
    }
}
```

C++

```
// TODO
```

This is relatively short and minimizes boilerplate. It is also comfortable to use in a purely object-oriented paradigm and may be more acceptable to novice programmers. However, it has some downsides. For one, it is not immediately clear exactly what type of command group this is from the constructor definition : it is better to define this in a more inline and expressive way, particularly when nested command groups start showing up. Additionally, it requires a new file for every single command group, even when the groups are conceptually related.

As with factory methods, state can be defined and captured within the command group subclass constructor, if necessary.

Summary

Approach	Primary Use Case	Single-subsystem Commands	Multi-subsystem Commands	Stateful Commands	Complex Logic Commands
Instance Factory Methods	Single-subsystem commands	Excels at them	No	Yes, but must obey capture rules	Yes
Subclassing Command	Stateful commands	Very verbose	Relatively verbose	Excels at them	Yes ; may be more natural than other approaches
Static and Instance Command Factories	Multi-subsystem commands	Yes	Yes	Yes, but must obey capture rules	Yes
Subclassing Command Groups	Multi-subsystem command groups	Yes	Yes	Yes, but must obey capture rules	Yes

26.8 Le planificateur de commandes

The `CommandScheduler` (Java, C++) is the class responsible for actually running commands. Each iteration (ordinarily once per 20ms), the scheduler polls all registered buttons, schedules commands for execution accordingly, runs the command bodies of all scheduled commands, and ends those commands that have finished or are interrupted.

Le `CommandScheduler` exécute aussi la méthode `periodic()` de chaque `Subsystem` enrégistré.

26.8.1 Utiliser le planificateur de commandes

Le `CommandScheduler` est un *singleton*, ce qui veux dire qu'il est une classe accessible globalement avec une seule instance. Donc, pour accéder au planificateur, l'utilisateur devra appeler la commande `CommandScheduler.getInstance()`.

For the most part, users do not have to call scheduler methods directly - almost all important scheduler methods have convenience wrappers elsewhere (e.g. in the `Command` and `Subsystem` classes).

Cependant, il y a une exception : l'utilisateur *doit* appeler `CommandScheduler.getInstance().run()` de la méthode `robotPeriodic()` de la classe `Robot`. Si ce n'est pas effectué, le planificateur ne s'exécutera jamais et le cadre d'application des commandes ne fonctionnera pas. Le modèle de projet orienté commande fourni inclus déjà cet appel.

26.8.2 La méthode `schedule()`

To schedule a command, users call the `schedule()` method (Java, C++). This method takes a command, and attempts to add it to list of currently-running commands, pending whether it is already running or whether its requirements are available. If it is added, its `initialize()` method is called.

This method walks through the following steps :

1. Verifies that the command isn't in a composition.
2. *No-op* if scheduler is disabled, command is already scheduled, or robot is disabled and command doesn't <commands :runsWhenDisabled>.
3. If requirements are in use : * If all conflicting commands are interruptible, cancel them. * If not, don't schedule the new command.
4. Call `initialize()`.

Java

```
202   private void schedule(Command command) {
203     if (command == null) {
204       DriverStation.reportWarning("Tried to schedule a null command", true);
205       return;
206     }
207     if (m_inRunLoop) {
208       m_toSchedule.add(command);
209       return;
210     }
211
212     requireNotComposed(command);
213
214     // Do nothing if the scheduler is disabled, the robot is disabled and the command␣
    ↪doesn't
215     // run when disabled, or the command is already scheduled.
216     if (m_disabled
217         || isScheduled(command)
218         || RobotState.isDisabled() && !command.runsWhenDisabled()) {
219       return;
220     }
```

(suite sur la page suivante)

(suite de la page précédente)

```
221
222     Set<Subsystem> requirements = command.getRequirements();
223
224     // Schedule the command if the requirements are not currently in-use.
225     if (Collections.disjoint(m_requirements.keySet(), requirements)) {
226       initCommand(command, requirements);
227     } else {
228       // Else check if the requirements that are in use have all have interruptible␣
    ↪commands,
229       // and if so, interrupt those commands and schedule the new command.
230       for (Subsystem requirement : requirements) {
231         Command requiring = requiring(requirement);
232         if (requiring != null
233             && requiring.getInterruptionBehavior() == InterruptionBehavior.
    ↪kCancelIncoming) {
234           return;
235         }
236       }
237       for (Subsystem requirement : requirements) {
238         Command requiring = requiring(requirement);
239         if (requiring != null) {
240           cancel(requiring);
241         }
242       }
243       initCommand(command, requirements);
244     }
245   }
```

```
181   private void initCommand(Command command, Set<Subsystem> requirements) {
182     m_scheduledCommands.add(command);
183     for (Subsystem requirement : requirements) {
184       m_requirements.put(requirement, command);
185     }
186     command.initialize();
187     for (Consumer<Command> action : m_initActions) {
188       action.accept(command);
189     }
190
191     m_watchdog.addEpoch(command.getName() + ".initialize()");
```

C++ (Source)

```
114 void CommandScheduler::Schedule(Command* command) {
115   if (m_impl->inRunLoop) {
116     m_impl->toSchedule.emplace_back(command);
117     return;
118   }
119
120   RequireUngrouped(command);
121
122   if (m_impl->disabled || m_impl->scheduledCommands.contains(command) ||
123       (frc::RobotState::IsDisabled() && !command->RunsWhenDisabled())) {
124     return;
125   }
```

(suite sur la page suivante)

(suite de la page précédente)

```

  const auto& requirements = command->GetRequirements();

  wpi::SmallVector<Command*, 8> intersection;

  bool isDisjoint = true;
  bool allInterruptible = true;
  for (auto&& i1 : m_impl->requirements) {
    if (requirements.find(i1.first) != requirements.end()) {
      isDisjoint = false;
      allInterruptible &= (i1.second->GetInterruptionBehavior() ==
                           Command::InterruptionBehavior::kCancelSelf);
      intersection.emplace_back(i1.second);
    }
  }

  if (isDisjoint || allInterruptible) {
    if (allInterruptible) {
      for (auto&& cmdToCancel : intersection) {
        Cancel(cmdToCancel);
      }
    }
    m_impl->scheduledCommands.insert(command);
    for (auto&& requirement : requirements) {
      m_impl->requirements[requirement] = command;
    }
    command->Initialize();
    for (auto&& action : m_impl->initActions) {
      action(*command);
    }
    m_watchdog.AddEpoch(command->GetName() + ".Initialize()");
  }
}
```

26.8.3 La séquence d'exécution du planificateur

Note : La méthode `initialize()` de chaque `Command` est appelée lorsque la commande est programmée, ce qui n'est pas forcément quand le planificateur s'exécute (sauf si cette commande est liée à un bouton).

What does a single iteration of the scheduler's `run()` method (Java, C++) actually do? The following section walks through the logic of a scheduler iteration. For the full implementation, see the source code (Java, C++).

Étape 1 : exécuter les méthodes périodique des sous-systèmes

First, the scheduler runs the `periodic()` method of each registered `Subsystem`. In simulation, each subsystem's `simulationPeriodic()` method is called as well.

Java

```
278     // Run the periodic method of all registered subsystems.
279     for (Subsystem subsystem : m_subsystems.keySet()) {
280       subsystem.periodic();
281       if (RobotBase.isSimulation()) {
282         subsystem.simulationPeriodic();
283       }
284       m_watchdog.addEpoch(subsystem.getClass().getSimpleName() + ".periodic()");
285     }
```

C++ (Source)

```
183   // Run the periodic method of all registered subsystems.
184   for (auto&& subsystem : m_impl->subsystems) {
185     subsystem.getFirst()->Periodic();
186     if constexpr (frc::RobotBase::IsSimulation()) {
187       subsystem.getFirst()->SimulationPeriodic();
188     }
189     m_watchdog.AddEpoch("Subsystem Periodic()");
190   }
```

Étape 2 : sonder les déclencheurs de planification de commande

---

Note : Pour plus d'informations sur le fonctionnement des liens de déclencheurs, voir *Liaison de commandes à des déclencheurs*

Deuxièmement, le planificateur sonde l'état de tous les déclencheurs enregistres pour voir si des nouvelles commandes qui ont été liées à ces déclencheurs devraient être planifiées. Si les conditions pour planifier une commande liée sont remplies, la commande est planifié et sa méthode `initialize()` est exécutée.

Java

```
290     // Poll buttons for new commands to add.
291     loopCache.poll();
292     m_watchdog.addEpoch("buttons.run()");
```

C++ (Source)

```
195   // Poll buttons for new commands to add.
196   loopCache->Poll();
197   m_watchdog.AddEpoch("buttons.Run()");
```

Étape 3 : exécuter/terminer les commandes planifiées

Troisièmement, le planificateur appelle la méthode `execute()` de chaque commande en cours d'exécution après avoir vérifié si la commande est terminé en appelant la méthode `isFinished()`. Si la commande est terminé, la méthode `end()` est appelée aussi, et la commande est dé-planifiée et ses sous-systèmes requis sont libérés.

Notez qui cette séquence d'appels est faite en ordre pour chaque commande - ainsi, une commande peut avoir sa méthode `end()` appelée avant qu'une autre ait sa méthode `execute()` appelée. Les commandes sont manipulées dans l'ordre dont elles ont été planifiées.

Java

```
295     // Run scheduled commands, remove finished commands.
296     for (Iterator<Command> iterator = m_scheduledCommands.iterator(); iterator.
    ↪hasNext(); ) {
297       Command command = iterator.next();
298
299       if (!command.runsWhenDisabled() && RobotState.isDisabled()) {
300         command.end(true);
301         for (Consumer<Command> action : m_interruptActions) {
302           action.accept(command);
303         }
304         m_requirements.keySet().removeAll(command.getRequirements());
305         iterator.remove();
306         m_watchdog.addEpoch(command.getName() + ".end(true)");
307         continue;
308       }
309
310       command.execute();
311       for (Consumer<Command> action : m_executeActions) {
312         action.accept(command);
313       }
314       m_watchdog.addEpoch(command.getName() + ".execute()");
315       if (command.isFinished()) {
316         command.end(false);
317         for (Consumer<Command> action : m_finishActions) {
318           action.accept(command);
319         }
320         iterator.remove();
321
322         m_requirements.keySet().removeAll(command.getRequirements());
323         m_watchdog.addEpoch(command.getName() + ".end(false)");
324       }
325     }
```

C++ (Source)

```
  for (Command* command : m_impl->scheduledCommands) {
    if (!command->RunsWhenDisabled() && frc::RobotState::IsDisabled()) {
      Cancel(command);
      continue;
    }

    command->Execute();
    for (auto&& action : m_impl->executeActions) {
      action(*command);
    }
    m_watchdog.AddEpoch(command->GetName() + ".Execute()");

    if (command->IsFinished()) {
      command->End(false);
      for (auto&& action : m_impl->finishActions) {
        action(*command);
      }

      for (auto&& requirement : command->GetRequirements()) {
        m_impl->requirements.erase(requirement);
      }

      m_impl->scheduledCommands.erase(command);
      m_watchdog.AddEpoch(command->GetName() + ".End(false)");
    }
  }
```

Étape 4 : planifier les commandes par défaut

Finalement, chaque `Subsystem` enrégistré a sa commande par défaut planifiée (si elle existe). Notez que la méthode `initialize()` de la commande de défaut sera appelée à ce moment.

Java

```
    // Add default commands for un-required registered subsystems.
    for (Map.Entry<Subsystem, Command> subsystemCommand : m_subsystems.entrySet()) {
      if (!m_requirements.containsKey(subsystemCommand.getKey())
          && subsystemCommand.getValue() != null) {
        schedule(subsystemCommand.getValue());
      }
    }
```

C++ (Source)

```
240   // Add default commands for un-required registered subsystems.
241   for (auto&& subsystem : m_impl->subsystems) {
242     auto s = m_impl->requirements.find(subsystem.getFirst());
243     if (s == m_impl->requirements.end() && subsystem.getSecond()) {
244       Schedule({subsystem.getSecond().get()});
245     }
246   }
```

26.8.4 Désactiver le Planificateur

Le planificateur peut être désactiver en appelant `CommandScheduler.getInstance().disable()`. Quand il est désactivé, les commandes `schedule()` et `run()` du planificateur ne font rien.

Le planificateur peut être ré-activé en appelant `CommandScheduler.getInstance().enable()`.

26.8.5 Méthodes d'évènement de commande

Parfois, il est souhaitable que le planificateur exécute une action personnalisée chaque fois qu'un certain événement de commande (initialisation, exécution ou fin) se produit. Cela peut être fait avec les méthodes suivantes :

- `onCommandInitialize` (Java, C++) runs a specified action whenever a command is initialized.
- `onCommandExecute` (Java, C++) runs a specified action whenever a command is executed.
- `onCommandFinish` (Java, C++) runs a specified action whenever a command finishes normally (i.e. the `isFinished()` method returned true).
- `onCommandInterrupt` (Java, C++) runs a specified action whenever a command is interrupted (i.e. by being explicitly canceled or by another command that shares one of its requirements).

A typical use-case for these methods is adding markers in an event log whenever a command scheduling event takes place, as demonstrated in the following code from the HatchbotInlined example project (Java, C++) :

Java

```
73     // Set the scheduler to log Shuffleboard events for command initialize, interrupt,
   ↪ finish
74     CommandScheduler.getInstance()
75         .onCommandInitialize(
76             command ->
77                 Shuffleboard.addEventMarker(
78                     "Command initialized", command.getName(), EventImportance.
   ↪kNormal));
79     CommandScheduler.getInstance()
80         .onCommandInterrupt(
81             command ->
```

(suite sur la page suivante)

(suite de la page précédente)

```
                Shuffleboard.addEventMarker(
                    "Command interrupted", command.getName(), EventImportance.
↪kNormal));
    CommandScheduler.getInstance()
        .onCommandFinish(
            command ->
                Shuffleboard.addEventMarker(
                    "Command finished", command.getName(), EventImportance.kNormal));
```

C++ (Source)

```
// Log Shuffleboard events for command initialize, execute, finish, interrupt
frc2::CommandScheduler::GetInstance().OnCommandInitialize(
    [](const frc2::Command& command) {
      frc::Shuffleboard::AddEventMarker(
          "Command initialized", command.GetName(),
          frc::ShuffleboardEventImportance::kNormal);
    });
frc2::CommandScheduler::GetInstance().OnCommandExecute(
    [](const frc2::Command& command) {
      frc::Shuffleboard::AddEventMarker(
          "Command executed", command.GetName(),
          frc::ShuffleboardEventImportance::kNormal);
    });
frc2::CommandScheduler::GetInstance().OnCommandFinish(
    [](const frc2::Command& command) {
      frc::Shuffleboard::AddEventMarker(
          "Command finished", command.GetName(),
          frc::ShuffleboardEventImportance::kNormal);
    });
frc2::CommandScheduler::GetInstance().OnCommandInterrupt(
    [](const frc2::Command& command) {
      frc::Shuffleboard::AddEventMarker(
          "Command interrupted", command.GetName(),
          frc::ShuffleboardEventImportance::kNormal);
    });
```

26.9 Une discussion technique sur les commandes C++

Note : This article assumes that you have a fair understanding of advanced C++ concepts, including templates, smart pointers, inheritance, rvalue references, copy semantics, move semantics, and CRTP. You do not need to understand the information within this article to use the command-based framework in your robot code.

This article will help you understand the reasoning behind some of the decisions made in the 2020 command-based framework (such as the use of `std::unique_ptr`, CRTP in the form of `CommandHelper<Base, Derived>`, etc.). You do not need to understand the information within this article to use the command-based framework in your robot code.

Note : The model was further changed in 2023, as described *below*.

26.9.1 Modèle de propriété

L'ancien cadre basé sur les commandes utilisait des pointeurs bruts, ce qui signifie que les usagers devaient employer la déclaration `new` (résultant en des allocations manuelles de mémoire de type « heap ») dans leur code robot. Comme il n'y avait aucune indication claire sur quel partie du programme qui était responsable des commandes (le planificateur, les groupes de commandes ou l'utilisateur lui-même), il n'était pas évident de savoir qui devait s'occuper de libérer cette mémoire allouée.

Plusieurs exemples dans l'ancien cadre basé sur des commandes impliquaient du code comme celui-ci :

```
#include "PlaceSoda.h"
#include "Elevator.h"
#include "Wrist.h"

PlaceSoda::PlaceSoda() {
  AddSequential(new SetElevatorSetpoint(Elevator::TABLE_HEIGHT));
  AddSequential(new SetWristSetpoint(Wrist::PICKUP));
  AddSequential(new OpenClaw());
}
```

In the command-group above, the component commands of the command group were being heap allocated and passed into `AddSequential` all in the same line. This meant that user had no reference to that object in memory and therefore had no means of freeing the allocated memory once the command group ended. The command group itself never freed the memory and neither did the command scheduler. This led to memory leaks in robot programs (i.e. memory was allocated on the heap but never freed).

Ce problème flagrant a été l'une des raisons de la réécriture du cadre. Un nouveau modèle de propriété complet a été généré avec cette réécriture, qui comprend l'utilisation de pointeurs intelligents pour libérer automatiquement la mémoire lorsqu'ils ne seront plus actifs.

Default commands are owned by the command scheduler whereas component commands of command compositions are owned by the command composition. Other commands are owned by whatever the user decides they should be owned by (e.g. a subsystem instance or a `RobotContainer` instance). This means that the ownership of the memory allocated by any commands or command compositions is clearly defined.

`std::unique_ptr` vs. `std::shared_ptr`

L'utilisation de `std::unique_ptr` nous permet de déterminer clairement à qui appartient l'objet. Comme un `std::unique_ptr` ne peut pas être copié, il n'y aura jamais plus d'une instance de `std::unique_ptr` qui pointe vers le même bloc de mémoire allouée. Par exemple, un constructeur pour `SequentialCommandGroup` prend un `std::vector<std::unique_ptr<Command>>&&`. Cela signifie qu'il nécessite une référence rvalue à un vecteur de ``std ::unique_ptr``<Command>. Passons en revue un exemple de code étape par étape pour mieux comprendre cela :

```
// Let's create a vector to store our commands that we want to run sequentially.
std::vector<std::unique_ptr<Command>> commands;

// Add an instant command that prints to the console.
commands.emplace_back(std::make_unique<InstantCommand>([]{ std::cout << "Hello"; },␣
↪requirements));

// Add some other command: this can be something that a user has created.
commands.emplace_back(std::make_unique<MyCommand>(args, needed, for, this, command));

// Now the vector "owns" all of these commands. In its current state, when the vector␣
↪is destroyed (i.e.
// it goes out of scope), it will destroy all of the commands we just added.

// Let's create a SequentialCommandGroup that will run these two commands␣
↪sequentially.
auto group = SequentialCommandGroup(std::move(commands));

// Note that we MOVED the vector of commands into the sequential command group,␣
↪meaning that the
// command group now has ownership of our commands. When we call std::move on the␣
↪vector, all of its
// contents (i.e. the unique_ptr instances) are moved into the command group.

// Even if the vector were to be destroyed while the command group was running,␣
↪everything would be OK
// since the vector does not own our commands anymore.
```

Avec `std::shared_ptr`, il n'y a pas de modèle de propriété clair car il peut y avoir plusieurs instances d'un `std::shared_ptr` qui pointent vers le même bloc de mémoire. Si les commandes étaient dans des instances `std::shared_ptr`, un groupe de commandes ou le planificateur de commandes ne peuvent pas prendre possession et libérer la mémoire une fois que la commande a fini de s'exécuter car l'utilisateur peut encore, sans le savoir, avoir une instance `std::shared_ptr` pointant vers ce bloc de mémoire quelque part.

26.9.2 Utilisation de CRTP

You may have noticed that in order to create a new command, you must extend `CommandHelper`, providing the base class (usually `frc2::Command`) and the class that you just created. Let's take a look at the reasoning behind this :

Décorateurs de commandes

Le nouveau cadre basé sur les commandes comprend une fonctionnalité connue sous le nom de « décorateurs de commandes », qui permet à l'utilisateur de faire quelque chose comme ceci :

```
auto task = MyCommand().AndThen([] { std::cout << "This printed after my command␣
↪ended."; },
  requirements);
```

Lorsque `task` est planifiée, elle exécutera d'abord `MyCommand()` et une fois que cette commande aura fini de s'exécuter, elle imprimera le message sur la console. La façon dont cela est réalisé en interne consiste à utiliser un groupe de commandes séquentielles.

Rappelez-vous de la section précédente que pour construire un groupe de commandes séquentielles, nous avons besoin d'un vecteur de pointeurs uniques vers chaque commande. La création du pointeur unique pour la fonction d'impression est assez simple :

```
temp.emplace_back(
   std::make_unique<InstantCommand>(std::move(toRun), requirements));
```

Ici, `temp` stocke le vecteur de commandes que nous devons passer dans le constructeur `SequentialCommandGroup`. Mais avant d'ajouter `InstantCommand`, nous devons ajouter `` MyCommand () "" au `SequentialCommandGroup`. Comment fait-on cela ?

```
temp.emplace_back(std::make_unique<MyCommand>(std::move(*this));
```

You might think it would be this straightforward, but that is not the case. Because this decorator code is in the `Command` class, `*this` refers to the `Command` in the subclass that you are calling the decorator from and has the type of `Command`. Effectively, you will be trying to move a `Command` instead of `MyCommand`. We could cast the `this` pointer to a `MyCommand*` and then dereference it but we have no information about the subclass to cast to at compile-time.

Solutions au problème

Notre solution initiale à cela était de créer une méthode virtuelle dans `Command` appelée `TransferOwnership()` que chaque sous-classe de `Command` devait remplacer. Un tel remplacement aurait ressemblé à ceci :

```
std::unique_ptr<Command> TransferOwnership() && override {
  return std::make_unique<MyCommand>(std::move(*this));
}
```

Parce que le code serait dans la sous-classe dérivée, `*this` pointerait alors vers l'instance souhaitée de la sous-classe, et par conséquant, l'utilisateur dispose alors des informations de type de la classe dérivée pour créer le pointeur unique.

Après quelques jours de délibération, une méthode CRTP a été proposée. Ici, une classe dérivée intermédiaire de `Command` appelée `CommandHelper` serait créé. `CommandHelper` aurait deux arguments de modèle, la classe de base d'origine et la sous-classe dérivée souhaitée. Jetons un coup d'œil à une implémentation de base de `CommandHelper` pour comprendre ceci :

```
// In the real implementation, we use SFINAE to check that Base is actually a
// Command or a subclass of Command.
template<typename Base, typename Derived>
class CommandHelper : public Base {
  // Here, we are just inheriting all of the superclass (base class) constructors.
  using Base::Base;

  // Here, we will override the TransferOwnership() method mentioned above.
  std::unique_ptr<Command> TransferOwnership() && override {
    // Previously, we mentioned that we had no information about the derived class
    // to cast to at compile-time, but because of CRTP we do! It's one of our template
    // arguments!
    return std::make_unique<Derived>(std::move(*static_cast<Derived*>(this)));
  }
};
```

Ainsi, en faisant étendre vos commandes personnalisées via `CommandHelper` au lieu de `Command` fera implémenter automatiquement ce processus pour vous et c'est le raisonnement

derrière le fait de demander aux équipes d'utiliser ce qui peut sembler être une façon assez obscure de faire les choses.

Pour revenir à notre exemple `AndThen()`, nous pouvons maintenant faire ce qui suit :

```
// Because of how inheritance works, we will call the TransferOwnership()
// of the subclass. We are moving *this because TransferOwnership() can only
// be called on rvalue references.
temp.emplace_back(std::move(*this).TransferOwnership());
```

26.9.3 Manque de décorateurs avancés

La plupart des décorateurs C ++ utilisent `std::function<void()>` au lieu des commandes proprement dites. L'idée de prendre des commandes réelles dans les décorateurs telles que `AndThen()`, `BeforeStarting()`, etc. a été envisagée mais abandonnée pour diverses raisons.

Décorateurs de modèles

Parce que nous avons besoin de connaître les types de commandes que nous ajoutons à un groupe de commandes au moment de la compilation, nous devrons utiliser des modèles (variadic pour plusieurs commandes). Cependant, cela peut ne pas sembler un gros problème. Les constructeurs des groupes de commandes le font par défaut :

```
template <class... Types,
          typename = std::enable_if_t<std::conjunction_v<
              std::is_base_of<Command, std::remove_reference_t<Types>>...>>>
explicit SequentialCommandGroup(Types&&... commands) {
  AddCommands(std::forward<Types>(commands)...);
}

template <class... Types,
          typename = std::enable_if_t<std::conjunction_v<
              std::is_base_of<Command, std::remove_reference_t<Types>>...>>>
void AddCommands(Types&&... commands) {
  std::vector<std::unique_ptr<Command>> foo;
  ((void)foo.emplace_back(std::make_unique<std::remove_reference_t<Types>>(
       std::forward<Types>(commands))),
   ...);
  AddCommands(std::move(foo));
}
```

Note : Il s'agit d'un constructeur secondaire pour `SequentialCommandGroup` en plus du constructeur vectoriel que nous avons décrit ci-dessus.

Cependant, lorsque nous créons une fonction basée sur un modèle, sa définition doit être déclarée en ligne. Cela signifie que nous devrons instancier le `SequentialCommandGroup` dans l'en-tête `Command.h`, ce qui pose un problème. `SequentialCommandGroup.h` inclut `Command.h`. Si nous incluons `SequentialCommandGroup.h` à l'intérieur de `Command.h`, nous avons une dépendance circulaire. Comment procéder alors ?

Nous utilisons une déclaration en haut de `Command.h` :

```
class SequentialCommandGroup;

class Command { ... };
```

Et puis nous incluons `SequentialCommandGroup.h` dans `Command.cpp`. Cependant, si ces fonctions décoratrices ont été modélisées, nous ne pouvons pas écrire de définitions dans les fichiers `.cpp`, ce qui entraîne une dépendance circulaire.

Syntaxe Java vs C ++

Ces décorateurs réduisent la verbosité en Java (car Java nécessite des appels bruts de type `new`) par rapport à C ++, donc en général, cela ne fait pas beaucoup de différence au niveau de la syntaxe en C ++ si vous créez le groupe de commandes manuellement dans le code utilisateur.

26.9.4 2023 Updates

After a few years in the new command-based framework, the recommended way to create commands increasingly shifted towards inline commands, decorators, and factory methods. With this paradigm shift, it became evident that the C++ commands model introduced in 2020 and described above has some pain points when used according to the new recommendations.

A significant root cause of most pain points was commands being passed by value in a non-polymorphic way. This made object slicing mistakes rather easy, and changes in composition structure could propagate type changes throughout the codebase : for example, if a `ParallelRaceGroup` were changed to a `ParallelDeadlineGroup`, those type changes would propagate through the codebase. Passing around the object as a `Command` (as done in Java) would result in object slicing.

Additionally, various decorators weren't supported in C++ due to reasons described *above*. As long as decorators were rarely used and were mainly to reduce verbosity (where Java was more verbose than C++), this was less of a problem. Once heavy usage of decorators was recommended, this became more of an issue.

`CommandPtr`

Let's recall the mention of `std::unique_ptr` far above : a value type with only move semantics. This is the ownership model we want !

However, plainly using `std::unique_ptr<Command>` had some drawbacks. Primarily, implementing decorators would be impossible : `unique_ptr` is defined in the standard library so we can't define methods on it, and any methods defined on `Command` wouldn't have access to the owning `unique_ptr`.

The solution is `CommandPtr` : a move-only value class wrapping `unique_ptr`, that we can define methods on.

Commands should be passed around as `CommandPtr`, using `std::move`. All decorators, including those not supported in C++ before, are defined on `CommandPtr` with rvalue-this. The use of rvalues, move-only semantics, and clear ownership makes it very easy to avoid mistakes such as adding the same command instance to more than one *command composition*.

In addition to decorators, `CommandPtr` instances also define utility methods such as `Schedule()`, `IsScheduled()`. `CommandPtr` instances can be used in nearly almost every way command objects can be used in Java : they can be moved into trigger bindings, default commands, and so on. For the few things that require a `Command*` (such as non-owning trigger bindings), a raw pointer to the owned command can be retrieved using `get()`.

There are multiple ways to get a `CommandPtr` instance :

- `CommandPtr`-returning factories are present in the `frc2::cmd` namespace in the `Commands.h` header for almost all command types. For multi-command compositions, there is a vector-taking overload as well as a variadic-templated overload for multiple `CommandPtr` instances.
- All decorators, including those defined on `Command`, return `CommandPtr`. This has allowed defining almost all decorators on `Command`, so a decorator chain can start from a `Command`.
- A `ToPtr()` method has been added to the CRTP, akin to `TransferOwnership`. This is useful especially for user-defined command classes, as well as other command classes that don't have factories.

For instance, consider the following from the HatchbotInlined example project :

```
frc2::CommandPtr autos::ComplexAuto(DriveSubsystem* drive,
                                    HatchSubsystem* hatch) {
  return frc2::cmd::Sequence(
      // Drive forward the specified distance
      frc2::FunctionalCommand(
          // Reset encoders on command start
          [drive] { drive->ResetEncoders(); },
          // Drive forward while the command is executing
          [drive] { drive->ArcadeDrive(kAutoDriveSpeed, 0); },
          // Stop driving at the end of the command
          [drive](bool interrupted) { drive->ArcadeDrive(0, 0); },
          // End the command when the robot's driven distance exceeds the
          // desired value
          [drive] {
            return drive->GetAverageEncoderDistance() >=
                   kAutoDriveDistanceInches;
          },
          // Requires the drive subsystem
          {drive})
          .ToPtr(),
      // Release the hatch
      hatch->ReleaseHatchCommand(),
      // Drive backward the specified distance
      // Drive forward the specified distance
      frc2::FunctionalCommand(
          // Reset encoders on command start
          [drive] { drive->ResetEncoders(); },
          // Drive backward while the command is executing
          [drive] { drive->ArcadeDrive(-kAutoDriveSpeed, 0); },
          // Stop driving at the end of the command
          [drive](bool interrupted) { drive->ArcadeDrive(0, 0); },
          // End the command when the robot's driven distance exceeds the
          // desired value
          [drive] {
            return drive->GetAverageEncoderDistance() <=
                   kAutoBackupDistanceInches;
          },
          // Requires the drive subsystem
```

(suite sur la page suivante)

(suite de la page précédente)

```
71          {drive})
72          .ToPtr());
73  }
```

To avoid breakage, command compositions still use `unique_ptr<Command>`, so `CommandPtr` instances can be destructured into a `unique_ptr<Command>` using the `Unwrap()` rvalue-this method. For vectors, the static `CommandPtr::UnwrapVector(vector<CommandPtr>)` function exists.

26.10 Contrôle PID via PIDSubsystems et PIDCommands

Note : Pour une description des fonctionnalités de contrôle PID WPILib utilisées par ces wrappers basés sur des commandes, voir *Contrôle PID dans WPILib*.

One of the most common control algorithms used in FRC® is the *PID* controller. WPILib offers its own *PIDController* class to help teams implement this functionality on their robots. To further help teams integrate PID control into a command-based robot project, the command-based library includes two convenience wrappers for the `PIDController` class : `PIDSubsystem`, which integrates the PID controller into a subsystem, and `PIDCommand`, which integrates the PID controller into a command.

26.10.1 Sous-systèmes PIDS

The `PIDSubsystem` class (Java, C++) allows users to conveniently create a subsystem with a built-in `PIDController`. In order to use the `PIDSubsystem` class, users must create a subclass of it.

Création d'un sous-système PIDS

Note : If `periodic` is overridden when inheriting from `PIDSubsystem`, make sure to call `super.periodic()`! Otherwise, PID functionality will not work properly.

Lors de la sous-classification de `PIDSubsystem`, les utilisateurs doivent remplacer deux méthodes abstraites pour fournir des fonctionnalités que la classe utilisera dans son fonctionnement ordinaire :

getMeasurement()

Java

```
protected abstract double getMeasurement();
```

C++

```
virtual double GetMeasurement() = 0;
```

La méthode `getMeasurement` renvoie la mesure actuelle de la variable de processus. Le `PIDSubsystem` appellera automatiquement cette méthode depuis son bloc `periodic()`, et passera sa valeur à la boucle de contrôle.

Les utilisateurs doivent remplacer cette méthode pour renvoyer plutôt la lecture du capteur qu'ils souhaitent utiliser comme mesure de variable de processus.

useOutput()

Java

```
protected abstract void useOutput(double output, double setpoint);
```

C++

```
virtual void UseOutput(double output, double setpoint) = 0;
```

La méthode `useOutput()` utilise la sortie du contrôleur PID et le point de consigne actuel (qui est souvent utile pour calculer un feedforward). `PIDSubsystem` appellera automatiquement cette méthode depuis son segment `periodic()`, et lui passera la sortie calculée de la boucle de contrôle.

Les utilisateurs doivent remplacer cette méthode pour transmettre plutôt la sortie de contrôle calculée et finale aux moteurs de leur sous-système.

La transmission d'un contrôleur

Les utilisateurs doivent également transmettre un `PIDController` à la classe de base `PIDSubsystem` via l'appel de constructeur de superclasse de leur sous-classe. Cela permet de spécifier les gains PID, ainsi que la période (si l'utilisateur utilise une période de boucle différente de celle utilisée dans le code du robot standard).

Des modifications supplémentaires (par exemple l'activation d'une entrée continue) peuvent être apportées au contrôleur dans le corps du constructeur en appelant `getController()`.

Utilisation d'un sous-système PIDS

Une fois qu'une instance d'une sous-classe `PIDSubsystem` a été créée, elle peut être utilisée par des commandes via les méthodes suivantes :

setSetpoint()

La méthode `setSetpoint()` peut être utilisée pour définir le point de consigne du `PIDSubsystem`. Le sous-système suivra automatiquement le point de consigne en utilisant la sortie définie :

JAVA

```
// The subsystem will track to a setpoint of 5.
examplePIDSubsystem.setSetpoint(5);
```

C++

```
// The subsystem will track to a setpoint of 5.
examplePIDSubsystem.SetSetpoint(5);
```

enable() and disable()

Les méthodes `enable()` et `disable()` activent et désactivent respectivement le contrôle PID du `PIDSubsystem`. Lorsque le sous-système est activé, il exécute automatiquement la boucle de contrôle et suit le point de consigne. Lorsqu'il est désactivé, aucun contrôle n'est effectué.

De plus, la méthode `enable()` réinitialise le `PIDController` interne, et la méthode `disable()` appelle la méthode définie par l'utilisateur *useOutput ()* avec à la fois la sortie et le point de consigne définis sur la valeur `0`.

Exemple complet avec PIDSubsystem

À quoi ressemble un `PIDSubsystem` lorsqu'il est utilisé dans la pratique ? Les exemples suivants sont pris à partir du projet d'exemple FrisbeeBot(Java, C++) :

Java

```
5  package edu.wpi.first.wpilibj.examples.frisbeebot.subsystems;
6
7  import edu.wpi.first.math.controller.PIDController;
8  import edu.wpi.first.math.controller.SimpleMotorFeedforward;
9  import edu.wpi.first.wpilibj.Encoder;
10 import edu.wpi.first.wpilibj.examples.frisbeebot.Constants.ShooterConstants;
11 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
12 import edu.wpi.first.wpilibj2.command.PIDSubsystem;
```

(suite sur la page suivante)

(suite de la page précédente)

```
13
14  public class ShooterSubsystem extends PIDSubsystem {
15    private final PWMSparkMax m_shooterMotor = new PWMSparkMax(ShooterConstants.
    ↪kShooterMotorPort);
16    private final PWMSparkMax m_feederMotor = new PWMSparkMax(ShooterConstants.
    ↪kFeederMotorPort);
17    private final Encoder m_shooterEncoder =
18        new Encoder(
19            ShooterConstants.kEncoderPorts[0],
20            ShooterConstants.kEncoderPorts[1],
21            ShooterConstants.kEncoderReversed);
22    private final SimpleMotorFeedforward m_shooterFeedforward =
23        new SimpleMotorFeedforward(
24            ShooterConstants.kSVolts, ShooterConstants.kVVoltSecondsPerRotation);
25
26    /** The shooter subsystem for the robot. */
27    public ShooterSubsystem() {
28      super(new PIDController(ShooterConstants.kP, ShooterConstants.kI,␣
    ↪ShooterConstants.kD));
29      getController().setTolerance(ShooterConstants.kShooterToleranceRPS);
30      m_shooterEncoder.setDistancePerPulse(ShooterConstants.kEncoderDistancePerPulse);
31      setSetpoint(ShooterConstants.kShooterTargetRPS);
32    }
33
34    @Override
35    public void useOutput(double output, double setpoint) {
36      m_shooterMotor.setVoltage(output + m_shooterFeedforward.calculate(setpoint));
37    }
38
39    @Override
40    public double getMeasurement() {
41      return m_shooterEncoder.getRate();
42    }
43
44    public boolean atSetpoint() {
45      return m_controller.atSetpoint();
46    }
47
48    public void runFeeder() {
49      m_feederMotor.set(ShooterConstants.kFeederSpeed);
50    }
51
52    public void stopFeeder() {
53      m_feederMotor.set(0);
54    }
55  }
```

C++

```
#pragma once

#include <frc/Encoder.h>
#include <frc/controller/SimpleMotorFeedforward.h>
#include <frc/motorcontrol/PWMSparkMax.h>
#include <frc2/command/PIDSubsystem.h>
#include <units/angle.h>

class ShooterSubsystem : public frc2::PIDSubsystem {
 public:
  ShooterSubsystem();

  void UseOutput(double output, double setpoint) override;

  double GetMeasurement() override;

  bool AtSetpoint();

  void RunFeeder();

  void StopFeeder();

 private:
  frc::PWMSparkMax m_shooterMotor;
  frc::PWMSparkMax m_feederMotor;
  frc::Encoder m_shooterEncoder;
  frc::SimpleMotorFeedforward<units::turns> m_shooterFeedforward;
};
```

C++ (Source)

```
#include "subsystems/ShooterSubsystem.h"

#include <frc/controller/PIDController.h>

#include "Constants.h"

using namespace ShooterConstants;

ShooterSubsystem::ShooterSubsystem()
    : PIDSubsystem{frc::PIDController{kP, kI, kD}},
      m_shooterMotor(kShooterMotorPort),
      m_feederMotor(kFeederMotorPort),
      m_shooterEncoder(kEncoderPorts[0], kEncoderPorts[1]),
      m_shooterFeedforward(kS, kV) {
  m_controller.SetTolerance(kShooterToleranceRPS.value());
  m_shooterEncoder.SetDistancePerPulse(kEncoderDistancePerPulse);
  SetSetpoint(kShooterTargetRPS.value());
}

void ShooterSubsystem::UseOutput(double output, double setpoint) {
  m_shooterMotor.SetVoltage(units::volt_t{output} +
                            m_shooterFeedforward.Calculate(kShooterTargetRPS));
```

(suite sur la page suivante)

(suite de la page précédente)

```
}

bool ShooterSubsystem::AtSetpoint() {
  return m_controller.AtSetpoint();
}

double ShooterSubsystem::GetMeasurement() {
  return m_shooterEncoder.GetRate();
}

void ShooterSubsystem::RunFeeder() {
  m_feederMotor.Set(kFeederSpeed);
}

void ShooterSubsystem::StopFeeder() {
  m_feederMotor.Set(0);
}
```

L'utilisation d'un `PIDSubsystem` avec des commandes peut être très simple :

Java

```
  private final Command m_spinUpShooter = Commands.runOnce(m_shooter::enable, m_
↪shooter);
  private final Command m_stopShooter = Commands.runOnce(m_shooter::disable, m_
↪shooter);

    // We can bind commands while retaining references to them in RobotContainer

    // Spin up the shooter when the 'A' button is pressed
    m_driverController.a().onTrue(m_spinUpShooter);

    // Turn off the shooter when the 'B' button is pressed
    m_driverController.b().onTrue(m_stopShooter);
```

C++

```
  frc2::CommandPtr m_spinUpShooter =
      frc2::cmd::RunOnce([this] { m_shooter.Enable(); }, {&m_shooter});

  frc2::CommandPtr m_stopShooter =
      frc2::cmd::RunOnce([this] { m_shooter.Disable(); }, {&m_shooter});
```

C++ (Source)

```
  // We can bind commands while keeping their ownership in RobotContainer

  // Spin up the shooter when the 'A' button is pressed
  m_driverController.A().OnTrue(m_spinUpShooter.get());

  // Turn off the shooter when the 'B' button is pressed
  m_driverController.B().OnTrue(m_stopShooter.get());
```

26.10.2 La classe PIDCommand

The `PIDCommand` class allows users to easily create commands with a built-in PIDController.

Création d'une commande PID

A `PIDCommand` can be created two ways - by subclassing the `PIDCommand` class, or by defining the command *inline*. Both methods ultimately extremely similar, and ultimately the choice of which to use comes down to where the user desires that the relevant code be located.

Note : If subclassing `PIDCommand` and overriding any methods, make sure to call the `super` version of those methods ! Otherwise, PID functionality will not work properly.

Dans les deux cas, un `PIDCommand` est créé en passant les paramètres nécessaires à son constructeur (lors de la définition d'une sous-classe, cela peut être fait avec un appel *super()*) :

Java

```
  /**
   * Creates a new PIDCommand, which controls the given output with a PIDController.
   *
   * @param controller the controller that controls the output.
   * @param measurementSource the measurement of the process variable
   * @param setpointSource the controller's setpoint
   * @param useOutput the controller's output
   * @param requirements the subsystems required by this command
   */
  public PIDCommand(
      PIDController controller,
      DoubleSupplier measurementSource,
      DoubleSupplier setpointSource,
      DoubleConsumer useOutput,
      Subsystem... requirements) {
```

C++

```
28  /**
29   * Creates a new PIDCommand, which controls the given output with a
30   * PIDController.
31   *
32   * @param controller        the controller that controls the output.
33   * @param measurementSource the measurement of the process variable
34   * @param setpointSource   the controller's reference (aka setpoint)
35   * @param useOutput         the controller's output
36   * @param requirements      the subsystems required by this command
37   */
38  PIDCommand(frc::PIDController controller,
39             std::function<double()> measurementSource,
40             std::function<double()> setpointSource,
41             std::function<void(double)> useOutput,
42             Requirements requirements = {});
```

manette

Le paramètre `controller` est l'objet `PIDController` qui sera utilisé par la commande. En transmettant cela, les utilisateurs peuvent spécifier les gains PID et la période pour le contrôleur (dans ce dernier cas, si l'utilisateur utilise une période de boucle de robot principal non standard).

Lors de la sous-classification de `PIDCommand`, des modifications supplémentaires (par exemple, l'activation d'une entrée continue) peuvent être apportées au contrôleur dans le corps du constructeur en appelant `getController()`.

Le paramètre measurementSource

The `measurementSource` parameter is a function (usually passed as a *lambda*) that returns the measurement of the process variable. Passing in the `measurementSource` function in `PIDCommand` is functionally analogous to overriding the *getMeasurement()* function in `PIDSubsystem`.

Lors de la sous-classification de `PIDCommand`, les utilisateurs plus expérimentés peuvent modifier le système qui fournit les mesures en modifiant le champ `m_measurement` de la classe.

Le paramètre setpointSource

The `setpointSource` parameter is a function (usually passed as a *lambda*) that returns the current setpoint for the control loop. If only a constant setpoint is needed, an overload exists that takes a constant setpoint rather than a supplier.

Lors de la sous-classification de `PIDCommand`, les utilisateurs plus expérimentés peuvent modifier le système qui fournit les mesures en modifiant le champ `m_measurement` de la classe.

Le paramètre useOutput

The `useOutput` parameter is a function (usually passed as a *lambda*) that consumes the output and setpoint of the control loop. Passing in the `useOutput` function in `PIDCommand` is functionally analogous to overriding the *useOutput()* function in `PIDSubsystem`.

Lors de la sous-classification de `PIDCommand`, les utilisateurs plus expérimentés peuvent modifier le système qui fournit les mesures en modifiant le champ `m_measurement` de la classe.

exigences

Comme toutes les commandes en ligne, `PIDCommand` permet à l'utilisateur de spécifier des exigences de sous-système via ses paramètres de constructeur.

Exemple complet de PIDCommand

À quoi ressemble un `PIDCommand` lorsqu'il est utilisé dans la pratique ? Les exemples suivants proviennent du projet d'exemple GyroDriveCommands (Java, C++) :

Java

```
5  package edu.wpi.first.wpilibj.examples.gyrodrivecommands.commands;
6
7  import edu.wpi.first.math.controller.PIDController;
8  import edu.wpi.first.wpilibj.examples.gyrodrivecommands.Constants.DriveConstants;
9  import edu.wpi.first.wpilibj.examples.gyrodrivecommands.subsystems.DriveSubsystem;
10 import edu.wpi.first.wpilibj2.command.PIDCommand;
11
12 /** A command that will turn the robot to the specified angle. */
13 public class TurnToAngle extends PIDCommand {
14   /**
15    * Turns to robot to the specified angle.
16    *
17    * @param targetAngleDegrees The angle to turn to
18    * @param drive The drive subsystem to use
19    */
20   public TurnToAngle(double targetAngleDegrees, DriveSubsystem drive) {
21     super(
22         new PIDController(DriveConstants.kTurnP, DriveConstants.kTurnI,␣
   ↪DriveConstants.kTurnD),
23         // Close loop on heading
24         drive::getHeading,
25         // Set reference to target
26         targetAngleDegrees,
27         // Pipe output to turn robot
28         output -> drive.arcadeDrive(0, output),
29         // Require the drive
30         drive);
31
32     // Set the controller to be continuous (because it is an angle controller)
33     getController().enableContinuousInput(-180, 180);
34     // Set the controller tolerance - the delta tolerance ensures the robot is␣
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪stationary at the
    // setpoint before it is considered as having reached the reference
    getController()
        .setTolerance(DriveConstants.kTurnToleranceDeg, DriveConstants.
↪kTurnRateToleranceDegPerS);
  }

  @Override
  public boolean isFinished() {
    // End when the controller is at the reference.
    return getController().atSetpoint();
  }
}
```

C++

```
#pragma once

#include <frc2/command/CommandHelper.h>
#include <frc2/command/PIDCommand.h>

#include "subsystems/DriveSubsystem.h"

/**
 * A command that will turn the robot to the specified angle.
 */
class TurnToAngle : public frc2::CommandHelper<frc2::PIDCommand, TurnToAngle> {
 public:
  /**
   * Turns to robot to the specified angle.
   *
   * @param targetAngleDegrees The angle to turn to
   * @param drive              The drive subsystem to use
   */
  TurnToAngle(units::degree_t target, DriveSubsystem* drive);

  bool IsFinished() override;
};
```

C++ (Source)

```
#include "commands/TurnToAngle.h"

#include <frc/controller/PIDController.h>

using namespace DriveConstants;

TurnToAngle::TurnToAngle(units::degree_t target, DriveSubsystem* drive)
    : CommandHelper{frc::PIDController{kTurnP, kTurnI, kTurnD},
                    // Close loop on heading
                    [drive] { return drive->GetHeading().value(); },
                    // Set reference to target
```

(suite sur la page suivante)

(suite de la page précédente)

```
                    target.value(),
                    // Pipe output to turn robot
                    [drive](double output) { drive->ArcadeDrive(0, output); },
                    // Require the drive
                    {drive}} {
  // Set the controller to be continuous (because it is an angle controller)
  m_controller.EnableContinuousInput(-180, 180);
  // Set the controller tolerance - the delta tolerance ensures the robot is
  // stationary at the setpoint before it is considered as having reached the
  // reference
  m_controller.SetTolerance(kTurnTolerance.value(), kTurnRateTolerance.value());

  AddRequirements(drive);
}

bool TurnToAngle::IsFinished() {
  return GetController().AtSetpoint();
}
```

And, for an *inlined* example :

Java

```
    // Stabilize robot to drive straight with gyro when left bumper is held
    new JoystickButton(m_driverController, Button.kL1.value)
        .whileTrue(
            new PIDCommand(
                new PIDController(
                    DriveConstants.kStabilizationP,
                    DriveConstants.kStabilizationI,
                    DriveConstants.kStabilizationD),
                // Close the loop on the turn rate
                m_robotDrive::getTurnRate,
                // Setpoint is 0
                0,
                // Pipe the output to the turning controls
                output -> m_robotDrive.arcadeDrive(-m_driverController.getLeftY(),␣
↪output),
                // Require the robot drive
                m_robotDrive));
```

C++

```
  // Stabilize robot to drive straight with gyro when L1 is held
  frc2::JoystickButton(&m_driverController, frc::PS4Controller::Button::kL1)
      .WhileTrue(
          frc2::PIDCommand(
              frc::PIDController{dc::kStabilizationP, dc::kStabilizationI,
                                 dc::kStabilizationD},
              // Close the loop on the turn rate
              [this] { return m_drive.GetTurnRate(); },
              // Setpoint is 0
```

(suite sur la page suivante)

(suite de la page précédente)

```
                0,
                // Pipe the output to the turning controls
                [this](double output) {
                  m_drive.ArcadeDrive(m_driverController.GetLeftY(), output);
                },
                // Require the robot drive
                {&m_drive})
```

26.11 Profilage de mouvement via TrapezoidProfileSubsystems et TrapezoidProfileCommands

Note : Pour une description des fonctionnalités de profilage de mouvement WPILib utilisées par ces wrappers basés sur des commandes, voir *Profils de mouvement trapézoïdal dans WPILib*.

Note : Les wrappers de commande `TrapezoidProfile` sont généralement destinés pour usage avec des contrôleurs personnalisés ou externes. Pour combiner le profilage de mouvement trapézoïdal avec le `PIDController` de WPILib, voir *Combinaison du profilage de mouvement et du PID dans les commandes*.

When controlling a mechanism, is often desirable to move it smoothly between two positions, rather than to abruptly change its setpoint. This is called « motion-profiling, » and is supported in WPILib through the `TrapezoidProfile` class (Java, C++).

Pour aider davantage les équipes à intégrer le profilage de mouvement dans leurs projets de robot basés sur des commandes, WPILib comprend deux wrappers pratiques pour la classe `TrapezoidProfile` : `TrapezoidProfileSubsystem`, qui génère et exécute automatiquement des profils de mouvement dans la méthode `periodic()`, et `TrapezoidProfileCommand`, qui exécute un seul `TrapezoidProfile` fourni par l'utilisateur.

26.11.1 TrapezoidProfileSubsystem

Note : En C ++, la classe `TrapezoidProfileSubsystem` est basée sur le type d'unité utilisé pour les mesures de distance, qui peut être angulaire ou linéaire. Les valeurs transmises *doivent* avoir des unités cohérentes avec les unités de distance, sinon une erreur de compilation sera indiquée. Pour plus d'informations sur les unités C ++, voir *La librairie d'unités C++*.

The `TrapezoidProfileSubsystem` class (Java, C++) will automatically create and execute trapezoidal motion profiles to reach the user-provided goal state. To use the `TrapezoidProfileSubsystem` class, users must create a subclass of it.

Création d'un sous-système TrapezoidProfile

Note : If `periodic` is overridden when inheriting from `TrapezoidProfileSubsystem`, make sure to call `super.periodic()` ! Otherwise, motion profiling functionality will not work properly.

Lors de la sous-classification de `TrapezoidProfileSubsystem`, les utilisateurs doivent remplacer une seule méthode abstraite pour fournir des fonctionnalités que la classe utilisera dans son fonctionnement ordinaire :

useState()

Java

```
protected abstract void useState(TrapezoidProfile.State state);
```

C++

```
virtual void UseState(State state) = 0;
```

La méthode `useState()` utilise l'état actuel du profil de mouvement pour les calculs. Le `TrapezoidProfileSubsystem` appellera automatiquement cette méthode à partir de son segment `periodic()`, et lui passera l'état du profil de mouvement correspondant à la progression actuelle du sous-système à travers le profil de mouvement.

Les utilisateurs peuvent faire ce qu'ils veulent avec cet état ; un cas d'utilisation typique (comme indiqué dans *Full TrapezoidProfileSubsystem Example*) consiste à utiliser l'état pour obtenir un point de consigne et une action directe pour un contrôleur de moteur externe « intelligent ».

Paramètres du constructeur

Les utilisateurs doivent transmettre un ensemble de valeurs-contraintes, soit `TrapezoidProfile.Constraints` à la classe de base `TrapezoidProfileSubsystem` via l'appel de constructeur de superclasse de leur sous-classe. Cela permet de contraindre les profils générés automatiquement à une vitesse et une accélération maximales données.

Les utilisateurs doivent également définir et transmettre une position initiale pour le mécanisme.

Les utilisateurs avancés peuvent transmettre une autre valeur pour la période de boucle, si la période de boucle principale est différente de la valeur standard.

Utilisation d'un sous-système TrapezoidProfile

Une fois qu'une instance d'une sous-classe `TrapezoidProfileSubsystem` a été créée, elle peut être utilisée par les commandes via les méthodes suivantes :

setGoal()

Note : Si vous souhaitez définir l'objectif à atteindre comme étant une distance fixe, avec une vitesse de zéro, il existe une version simplifiée de `setGoal()` qui prend une seule valeur de distance, plutôt qu'un état de profil de mouvement complet.

La méthode `setGoal()` peut être utilisée pour définir l'état de l'objectif du `TrapezoidProfileSubsystem`. Le sous-système exécutera automatiquement un profil vers l'objectif, en passant l'état actuel à chaque itération à la méthode *useState()* fournie.

JAVA

```
// The subsystem will execute a profile to a position of 5 and a velocity of 3.
examplePIDSubsystem.setGoal(new TrapezoidProfile.State(5, 3);
```

C++

```
// The subsystem will execute a profile to a position of 5 meters and a velocity of 3␣
↪mps.
examplePIDSubsyste.SetGoal({5_m, 3_mps});
```

enable() and disable()

The `enable()` and `disable()` methods enable and disable the motion profiling control of the `TrapezoidProfileSubsystem`. When the subsystem is enabled, it will automatically run the control loop and call `useState()` periodically. When it is disabled, no control is performed.

Un exemple complet de sous-système TrapezoidProfile

À quoi ressemble un `TrapezoidProfileSubsystem` lorsqu'il est utilisé dans la pratique ? Les exemples suivants proviennent du projet d'exemple ArmbotOffobard (Java, C++) :

Java

```java
package edu.wpi.first.wpilibj.examples.armbotoffboard.subsystems;

import edu.wpi.first.math.controller.ArmFeedforward;
import edu.wpi.first.math.trajectory.TrapezoidProfile;
import edu.wpi.first.wpilibj.examples.armbotoffboard.Constants.ArmConstants;
import edu.wpi.first.wpilibj.examples.armbotoffboard.ExampleSmartMotorController;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.Commands;
import edu.wpi.first.wpilibj2.command.TrapezoidProfileSubsystem;

/** A robot arm subsystem that moves with a motion profile. */
public class ArmSubsystem extends TrapezoidProfileSubsystem {
  private final ExampleSmartMotorController m_motor =
      new ExampleSmartMotorController(ArmConstants.kMotorPort);
  private final ArmFeedforward m_feedforward =
      new ArmFeedforward(
          ArmConstants.kSVolts, ArmConstants.kGVolts,
          ArmConstants.kVVoltSecondPerRad, ArmConstants.kAVoltSecondSquaredPerRad);

  /** Create a new ArmSubsystem. */
  public ArmSubsystem() {
    super(
        new TrapezoidProfile.Constraints(
            ArmConstants.kMaxVelocityRadPerSecond, ArmConstants.
↪kMaxAccelerationRadPerSecSquared),
        ArmConstants.kArmOffsetRads);
    m_motor.setPID(ArmConstants.kP, 0, 0);
  }

  @Override
  public void useState(TrapezoidProfile.State setpoint) {
    // Calculate the feedforward from the sepoint
    double feedforward = m_feedforward.calculate(setpoint.position, setpoint.
↪velocity);
    // Add the feedforward to the PID output to get the motor output
    m_motor.setSetpoint(
        ExampleSmartMotorController.PIDMode.kPosition, setpoint.position, feedforward␣
↪/ 12.0);
  }

  public Command setArmGoalCommand(double kArmOffsetRads) {
    return Commands.runOnce(() -> setGoal(kArmOffsetRads), this);
  }
}
```

C++ (Header)

```
#pragma once

#include <frc/controller/ArmFeedforward.h>
#include <frc2/command/Commands.h>
#include <frc2/command/TrapezoidProfileSubsystem.h>
#include <units/angle.h>

#include "ExampleSmartMotorController.h"

/**
 * A robot arm subsystem that moves with a motion profile.
 */
class ArmSubsystem : public frc2::TrapezoidProfileSubsystem<units::radians> {
  using State = frc::TrapezoidProfile<units::radians>::State;

 public:
  ArmSubsystem();

  void UseState(State setpoint) override;

  frc2::CommandPtr SetArmGoalCommand(units::radian_t goal);

 private:
  ExampleSmartMotorController m_motor;
  frc::ArmFeedforward m_feedforward;
};
```

C++ (Source)

```
#include "subsystems/ArmSubsystem.h"

#include "Constants.h"

using namespace ArmConstants;
using State = frc::TrapezoidProfile<units::radians>::State;

ArmSubsystem::ArmSubsystem()
    : frc2::TrapezoidProfileSubsystem<units::radians>(
          {kMaxVelocity, kMaxAcceleration}, kArmOffset),
      m_motor(kMotorPort),
      m_feedforward(kS, kG, kV, kA) {
  m_motor.SetPID(kP, 0, 0);
}

void ArmSubsystem::UseState(State setpoint) {
  // Calculate the feedforward from the sepoint
  units::volt_t feedforward =
      m_feedforward.Calculate(setpoint.position, setpoint.velocity);
  // Add the feedforward to the PID output to get the motor output
  m_motor.SetSetpoint(ExampleSmartMotorController::PIDMode::kPosition,
                      setpoint.position.value(), feedforward / 12_V);
}

```

(suite sur la page suivante)

(suite de la page précédente)

```
frc2::CommandPtr ArmSubsystem::SetArmGoalCommand(units::radian_t goal) {
  return frc2::cmd::RunOnce([this, goal] { this->SetGoal(goal); }, {this});
}
```

L'utilisation d'un `TrapezoidProfileSubsystem` avec des commandes peut être assez simple :

Java

```
    // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
    m_driverController.a().onTrue(m_robotArm.setArmGoalCommand(2));

    // Move the arm to neutral position when the 'B' button is pressed.
    m_driverController
        .b()
        .onTrue(m_robotArm.setArmGoalCommand(Constants.ArmConstants.kArmOffsetRads));
```

C++

```
  // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
  m_driverController.A().OnTrue(m_arm.SetArmGoalCommand(2_rad));

  // Move the arm to neutral position when the 'B' button is pressed.
  m_driverController.B().OnTrue(
      m_arm.SetArmGoalCommand(ArmConstants::kArmOffset));
```

26.11.2 TrapezoidProfileCommand

Note : En C ++, la classe `TrapezoidProfileCommand` est basée sur le type d'unité utilisé pour les mesures de distance, qui peut être angulaire ou linéaire. Les valeurs transmises *doivent* avoir des unités cohérentes avec les unités de distance, sinon une erreur de compilation sera indiquée. Pour plus d'informations sur les unités C++, voir *La librairie d'unités C++*.

The `TrapezoidProfileCommand` class (Java, C++) allows users to create a command that will execute a single `TrapezoidProfile`, passing its current state at each iteration to a user-defined function.

Création d'un TrapezoidProfileCommand

A `TrapezoidProfileCommand` can be created two ways - by subclassing the `TrapezoidProfileCommand` class, or by defining the command *inline*. Both methods are ultimately extremely similar, and ultimately the choice of which to use comes down to where the user desires that the relevant code be located.

Note : If subclassing `TrapezoidProfileCommand` and overriding any methods, make sure to call the `super` version of those methods ! Otherwise, motion profiling functionality will not

work properly.

Dans les deux cas, un `TrapezoidProfileCommand` est créé en passant les paramètres nécessaires à son constructeur (si vous définissez une sous-classe, cela peut être fait avec un appel *super()*) :

Java

```
  /**
   * Creates a new TrapezoidProfileCommand that will execute the given {@link␣
↪TrapezoidProfile}.
   * Output will be piped to the provided consumer function.
   *
   * @param profile The motion profile to execute.
   * @param output The consumer for the profile output.
   * @param goal The supplier for the desired state
   * @param currentState The current state
   * @param requirements The subsystems required by this command.
   */
  @SuppressWarnings("this-escape")
  public TrapezoidProfileCommand(
      TrapezoidProfile profile,
      Consumer<State> output,
      Supplier<State> goal,
      Supplier<State> currentState,
```

C++

```
  /**
   * Creates a new TrapezoidProfileCommand that will execute the given
   * TrapezoidalProfile. Output will be piped to the provided consumer function.
   *
   * @param profile      The motion profile to execute.
   * @param output       The consumer for the profile output.
   * @param goal The supplier for the desired state
   * @param currentState The current state
   * @param requirements The list of requirements.
   */
  TrapezoidProfileCommand(frc::TrapezoidProfile<Distance> profile,
                          std::function<void(State)> output,
                          std::function<State()> goal,
                          std::function<State()> currentState,
                          Requirements requirements = {})
```

Le paramètre Profile (Profil)

The `profile` parameter is the `TrapezoidProfile` object that will be executed by the command. By passing this in, users specify the motion constraints of the profile that the command will use.

Le paramètre Output (Sortie)

The `output` parameter is a function (usually passed as a *lambda*) that consumes the output and setpoint of the control loop. Passing in the `useOutput` function in `PIDCommand` is functionally analogous to overriding the *useState()* function in `PIDSubsystem`.

goal

The `goal` parameter is a function that supplies the desired state of the motion profile. This can be used to change the goal at runtime if desired.

currentState

The `currentState` parameter is a function that supplies the starting state of the motion profile. Combined with `goal`, this can be used to dynamically generate and follow any motion profile at runtime.

Les exigences

Comme toutes les commandes en ligne, `TrapezoidProfileCommand` permet à l'utilisateur de spécifier ses exigences de sous-système en tant que paramètre constructeur.

Exemple de commande TrapezoidProfileCommand complète

What does a `TrapezoidProfileSubsystem` look like when used in practice ? The following examples are taking from the DriveDistanceOffboard example project (Java, C++) :

Java

```
5  package edu.wpi.first.wpilibj.examples.drivedistanceoffboard.commands;
6
7  import edu.wpi.first.math.trajectory.TrapezoidProfile;
8  import edu.wpi.first.wpilibj.examples.drivedistanceoffboard.Constants.DriveConstants;
9  import edu.wpi.first.wpilibj.examples.drivedistanceoffboard.subsystems.DriveSubsystem;
10 import edu.wpi.first.wpilibj2.command.TrapezoidProfileCommand;
11
12 /** Drives a set distance using a motion profile. */
13 public class DriveDistanceProfiled extends TrapezoidProfileCommand {
14   /**
15    * Creates a new DriveDistanceProfiled command.
```

(suite sur la page suivante)

(suite de la page précédente)

```
   *
   * @param meters The distance to drive.
   * @param drive The drive subsystem to use.
   */
  public DriveDistanceProfiled(double meters, DriveSubsystem drive) {
    super(
        new TrapezoidProfile(
            // Limit the max acceleration and velocity
            new TrapezoidProfile.Constraints(
                DriveConstants.kMaxSpeedMetersPerSecond,
                DriveConstants.kMaxAccelerationMetersPerSecondSquared)),
        // Pipe the profile state to the drive
        setpointState -> drive.setDriveStates(setpointState, setpointState),
        // End at desired position in meters; implicitly starts at 0
        () -> new TrapezoidProfile.State(meters, 0),
        // Current position
        TrapezoidProfile.State::new,
        // Require the drive
        drive);
    // Reset drive encoders since we're starting at 0
    drive.resetEncoders();
  }
}
```

C++ (Header)

```
#pragma once

#include <frc2/command/CommandHelper.h>
#include <frc2/command/TrapezoidProfileCommand.h>

#include "subsystems/DriveSubsystem.h"

class DriveDistanceProfiled
    : public frc2::CommandHelper<frc2::TrapezoidProfileCommand<units::meters>,
                                 DriveDistanceProfiled> {
 public:
  DriveDistanceProfiled(units::meter_t distance, DriveSubsystem* drive);
};
```

C++ (Source)

```
#include "commands/DriveDistanceProfiled.h"

#include "Constants.h"

using namespace DriveConstants;

DriveDistanceProfiled::DriveDistanceProfiled(units::meter_t distance,
                                             DriveSubsystem* drive)
    : CommandHelper{
          frc::TrapezoidProfile<units::meters>{
```

(suite sur la page suivante)

(suite de la page précédente)

```
              // Limit the max acceleration and velocity
              {kMaxSpeed, kMaxAcceleration}},
          // Pipe the profile state to the drive
          [drive](auto setpointState) {
            drive->SetDriveStates(setpointState, setpointState);
          },
          // End at desired position in meters; implicitly starts at 0
          [distance] {
            return frc::TrapezoidProfile<units::meters>::State{distance, 0_mps};
          },
          [] { return frc::TrapezoidProfile<units::meters>::State{}; },
          // Require the drive
          {drive}} {
  // Reset drive encoders since we're starting at 0
  drive->ResetEncoders();
}
```

And, for an *inlined* example :

Java

```
    // Do the same thing as above when the 'B' button is pressed, but defined inline
    m_driverController
        .b()
        .onTrue(
            new TrapezoidProfileCommand(
                    new TrapezoidProfile(
                        // Limit the max acceleration and velocity
                        new TrapezoidProfile.Constraints(
                            DriveConstants.kMaxSpeedMetersPerSecond,
                            DriveConstants.kMaxAccelerationMetersPerSecondSquared)),
                    // Pipe the profile state to the drive
                    setpointState -> m_robotDrive.setDriveStates(setpointState,␣
↪setpointState),
                    // End at desired position in meters; implicitly starts at 0
                    () -> new TrapezoidProfile.State(3, 0),
                    // Current position
                    TrapezoidProfile.State::new,
                    // Require the drive
                    m_robotDrive)
                .beforeStarting(m_robotDrive::resetEncoders)
                .withTimeout(10));
```

C++

```
      DriveDistanceProfiled(3_m, &m_drive).WithTimeout(10_s));

  // Do the same thing as above when the 'B' button is pressed, but defined
  // inline
  m_driverController.B().OnTrue(
      frc2::TrapezoidProfileCommand<units::meters>(
          frc::TrapezoidProfile<units::meters>(
```

(suite sur la page suivante)

(suite de la page précédente)

```
            // Limit the max acceleration and velocity
            {DriveConstants::kMaxSpeed, DriveConstants::kMaxAcceleration}),
        // Pipe the profile state to the drive
        [this](auto setpointState) {
          m_drive.SetDriveStates(setpointState, setpointState);
        },
        // End at desired position in meters; implicitly starts at 0
        [] {
          return frc::TrapezoidProfile<units::meters>::State{3_m, 0_mps};
        },
        // Current position
        [] { return frc::TrapezoidProfile<units::meters>::State{}; },
        // Require the drive
        {&m_drive})
        // Convert to CommandPtr
        .ToPtr()
        .BeforeStarting(
```

26.12 Combinaison du profilage de mouvement et du PID dans les commandes

Note : Pour une description des fonctionnalités de contrôle PID WPILib utilisées par ces wrappers basés sur des commandes, voir : *Contrôle PID dans WPILib*.

A common FRC® controls solution is to pair a trapezoidal motion profile for setpoint generation with a PID controller for setpoint tracking. To facilitate this, WPILib includes its own *ProfiledPIDController* class. To further aid teams in integrating this functionality into their robots, the command-based framework contains two convenience wrappers for the `ProfiledPIDController` class : `ProfiledPIDSubsystem`, which integrates the controller into a subsystem, and `ProfiledPIDCommand`, which integrates the controller into a command.

26.12.1 ProfiledPIDSubsystem

Note : En C ++, la classe `ProfiledPIDSubsystem` est basée sur le type d'unité utilisé pour les mesures de distance, qui peut être angulaire ou linéaire. Les valeurs transmises *doivent* avoir des unités cohérentes avec les unités de distance, sinon une erreur de compilation sera levée. Pour plus d'informations sur les unités C++, voir *La librairie d'unités C++*.

The `ProfiledPIDSubsystem` class (Java, C++) allows users to conveniently create a subsystem with a built-in PIDController. In order to use the `ProfiledPIDSubsystem` class, users must create a subclass of it.

Création d'un sous-système ProfiledPIDS

Note : If `periodic` is overridden when inheriting from `ProfiledPIDSubsystem`, make sure to call `super.periodic()` ! Otherwise, control functionality will not work properly.

Dans la sous-classe `ProfiledPIDSubsystem`, les utilisateurs doivent remplacer deux méthodes abstraites pour fournir des fonctionnalités que la classe utilisera dans son fonctionnement normal :

getMeasurement ()

Java

```
protected abstract double getMeasurement();
```

C++

```
virtual Distance_t GetMeasurement() = 0;
```

La méthode `getMeasurement()` renvoie la mesure actuelle de la variable du processus en cours. `PIDSubsystem` appellera automatiquement cette méthode depuis son bloc `periodic()`, et passera sa valeur à la boucle de contrôle.

Les utilisateurs doivent remplacer cette méthode pour renvoyer la lecture du capteur qu'ils souhaitent utiliser comme mesure de variable de processus.

useOutput ()

Java

```
protected abstract void useOutput(double output, State setpoint);
```

C++

```
virtual void UseOutput(double output, State setpoint) = 0;
```

The `useOutput()` method consumes the output of the Profiled PID controller, and the current setpoint state (which is often useful for computing a feedforward). The `PIDSubsystem` will automatically call this method from its `periodic()` block, and pass it the computed output of the control loop.

Les utilisateurs doivent remplacer cette méthode pour transmettre la sortie de contrôle calculée finale aux moteurs de leur sous-système.

Passer le contrôleur

Les utilisateurs doivent également transmettre un `ProfiledPIDController` à la classe de base `ProfiledPIDSubsystem` via l'appel du constructeur de superclasse de leur sous-classe. Cela permet de spécifier les gains PID, les contraintes du profil de mouvement et la période (si l'utilisateur utilise une période de boucle de robot principal non standard).

Des modifications supplémentaires (par exemple l'activation d'une entrée continue) peuvent être apportées au contrôleur dans le corps du constructeur en appelant `getController()`.

Utilisation d'un sous-système ProfiledPIDS

Une fois qu'une instance d'une sous-classe `PIDSubsystem` a été créée, elle peut être utilisée par des commandes via les méthodes suivantes :

setGoal()

Note : Si vous souhaitez définir l'objectif à une distance simple avec une vitesse cible implicite de zéro, il existe une version de `setGoal()` qui prend une seule valeur de distance, plutôt qu'un état de profil de mouvement complet.

La méthode `setGoal()` peut être utilisée pour définir le point de consigne du `PIDSubsystem`. Le sous-système suivra automatiquement le point de consigne à l'aide de la sortie définie :

JAVA

```
// The subsystem will track to a goal of 5 meters and velocity of 3 meters per second.
examplePIDSubsystem.setGoal(5, 3);
```

C++

```
// The subsystem will track to a goal of 5 meters and velocity of 3 meters per second.
examplePIDSubsystem.SetGoal({5_m, 3_mps});
```

enable() et disable()

Les méthodes `enable()` et `disable()` activent et désactivent le contrôle automatique du `ProfiledPIDSubsystem`. Lorsque le sous-système est activé, il exécutera automatiquement le profil de mouvement et la boucle de contrôle et suivra jusqu'au but. Lorsqu'il est désactivé, aucun contrôle n'est effectué.

De plus, la méthode `enable()` réinitialise le `ProfiledPIDController` interne, et la méthode `disable()` appelle la méthode définie par l'utilisateur *useOutput()* avec à la fois la sortie et le point de consigne définis sur `0`.

Exemple de sous-système ProfiledPIDS complet

À quoi ressemble un PIDSubsystem lorsqu'il est utilisé dans la pratique ? Les exemples suivants sont pris à partir du projet d'exemple ArmBot (Java, C++) :

Java

```
package edu.wpi.first.wpilibj.examples.armbot.subsystems;

import edu.wpi.first.math.controller.ArmFeedforward;
import edu.wpi.first.math.controller.ProfiledPIDController;
import edu.wpi.first.math.trajectory.TrapezoidProfile;
import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.examples.armbot.Constants.ArmConstants;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
import edu.wpi.first.wpilibj2.command.ProfiledPIDSubsystem;

/** A robot arm subsystem that moves with a motion profile. */
public class ArmSubsystem extends ProfiledPIDSubsystem {
  private final PWMSparkMax m_motor = new PWMSparkMax(ArmConstants.kMotorPort);
  private final Encoder m_encoder =
      new Encoder(ArmConstants.kEncoderPorts[0], ArmConstants.kEncoderPorts[1]);
  private final ArmFeedforward m_feedforward =
      new ArmFeedforward(
          ArmConstants.kSVolts, ArmConstants.kGVolts,
          ArmConstants.kVVoltSecondPerRad, ArmConstants.kAVoltSecondSquaredPerRad);

  /** Create a new ArmSubsystem. */
  public ArmSubsystem() {
    super(
        new ProfiledPIDController(
            ArmConstants.kP,
            0,
            0,
            new TrapezoidProfile.Constraints(
                ArmConstants.kMaxVelocityRadPerSecond,
                ArmConstants.kMaxAccelerationRadPerSecSquared)),
        0);
    m_encoder.setDistancePerPulse(ArmConstants.kEncoderDistancePerPulse);
    // Start arm at rest in neutral position
    setGoal(ArmConstants.kArmOffsetRads);
  }

  @Override
  public void useOutput(double output, TrapezoidProfile.State setpoint) {
    // Calculate the feedforward from the sepoint
    double feedforward = m_feedforward.calculate(setpoint.position, setpoint.
↪velocity);
    // Add the feedforward to the PID output to get the motor output
    m_motor.setVoltage(output + feedforward);
  }

  @Override
  public double getMeasurement() {
    return m_encoder.getDistance() + ArmConstants.kArmOffsetRads;
```

(suite sur la page suivante)

(suite de la page précédente)

```
  }
}
```

C++ (Header)

```
#pragma once

#include <frc/Encoder.h>
#include <frc/controller/ArmFeedforward.h>
#include <frc/motorcontrol/PWMSparkMax.h>
#include <frc2/command/ProfiledPIDSubsystem.h>
#include <units/angle.h>

/**
 * A robot arm subsystem that moves with a motion profile.
 */
class ArmSubsystem : public frc2::ProfiledPIDSubsystem<units::radians> {
  using State = frc::TrapezoidProfile<units::radians>::State;

 public:
  ArmSubsystem();

  void UseOutput(double output, State setpoint) override;

  units::radian_t GetMeasurement() override;

 private:
  frc::PWMSparkMax m_motor;
  frc::Encoder m_encoder;
  frc::ArmFeedforward m_feedforward;
};
```

C++ (Source)

```
#include "subsystems/ArmSubsystem.h"

#include "Constants.h"

using namespace ArmConstants;
using State = frc::TrapezoidProfile<units::radians>::State;

ArmSubsystem::ArmSubsystem()
    : frc2::ProfiledPIDSubsystem<units::radians>(
          frc::ProfiledPIDController<units::radians>(
              kP, 0, 0, {kMaxVelocity, kMaxAcceleration})),
      m_motor(kMotorPort),
      m_encoder(kEncoderPorts[0], kEncoderPorts[1]),
      m_feedforward(kS, kG, kV, kA) {
  m_encoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
  // Start arm in neutral position
  SetGoal(State{kArmOffset, 0_rad_per_s});
}
```

(suite sur la page suivante)

(suite de la page précédente)

```

void ArmSubsystem::UseOutput(double output, State setpoint) {
  // Calculate the feedforward from the sepoint
  units::volt_t feedforward =
      m_feedforward.Calculate(setpoint.position, setpoint.velocity);
  // Add the feedforward to the PID output to get the motor output
  m_motor.SetVoltage(units::volt_t{output} + feedforward);
}

units::radian_t ArmSubsystem::GetMeasurement() {
  return units::radian_t{m_encoder.GetDistance()} + kArmOffset;
}
```

L'utilisation d'un `ProfiledPIDSubsystem` avec des commandes peut être très simple :

Java

```
    // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
    m_driverController
        .a()
        .onTrue(
            Commands.runOnce(
                () -> {
                  m_robotArm.setGoal(2);
                  m_robotArm.enable();
                },
                m_robotArm));
```

C++

```
  // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
  m_driverController.A().OnTrue(frc2::cmd::RunOnce(
      [this] {
        m_arm.SetGoal(2_rad);
        m_arm.Enable();
      },
      {&m_arm}));
```

26.12.2 ProfiledPIDCommand

Note : En C++, la classe `ProfiledPIDCommand` est basée sur le type d'unité utilisé pour les mesures de distance, qui peut être angulaire ou linéaire. Les valeurs transmises *doivent* avoir des unités cohérentes avec les unités de distance, sinon une erreur de compilation sera levée. Pour plus d'informations sur les unités C++, voir *La librairie d'unités C++*.

The `ProfiledPIDCommand` class (Java, C++) allows users to easily create commands with a built-in ProfiledPIDController.

Création d'une commande PID

A `ProfiledPIDCommand` can be created two ways - by subclassing the `ProfiledPIDCommand` class, or by defining the command *inline*. Both methods ultimately extremely similar, and ultimately the choice of which to use comes down to where the user desires that the relevant code be located.

Note : If subclassing `ProfiledPIDCommand` and overriding any methods, make sure to call the `super` version of those methods ! Otherwise, control functionality will not work properly.

Dans les deux cas, un `ProfiledPIDCommand` est créé en passant les paramètres nécessaires à son constructeur (si vous définissez une sous-classe, cela peut être fait avec un appel *super()*) :

Java

```
29  /**
30   * Creates a new PIDCommand, which controls the given output with a␣
    ↪ProfiledPIDController. Goal
31   * velocity is specified.
32   *
33   * @param controller the controller that controls the output.
34   * @param measurementSource the measurement of the process variable
35   * @param goalSource the controller's goal
36   * @param useOutput the controller's output
37   * @param requirements the subsystems required by this command
38   */
39  public ProfiledPIDCommand(
40      ProfiledPIDController controller,
41      DoubleSupplier measurementSource,
42      Supplier<State> goalSource,
43      BiConsumer<Double, State> useOutput,
44      Subsystem... requirements) {
```

C++

```
38  /**
39   * Creates a new PIDCommand, which controls the given output with a
40   * ProfiledPIDController.
41   *
42   * @param controller        the controller that controls the output.
43   * @param measurementSource the measurement of the process variable
44   * @param goalSource   the controller's goal
45   * @param useOutput         the controller's output
46   * @param requirements      the subsystems required by this command
47   */
48  ProfiledPIDCommand(frc::ProfiledPIDController<Distance> controller,
49                     std::function<Distance_t()> measurementSource,
50                     std::function<State()> goalSource,
51                     std::function<void(double, State)> useOutput,
52                     Requirements requirements = {})
```

Le paramètre controller

Le paramètre `controller` est l'objet `ProfiledPIDController` qui sera utilisé par la commande. En transmettant cela, les utilisateurs peuvent spécifier les gains PID, les contraintes de profil de mouvement et la période pour le contrôleur (si l'utilisateur utilise une période de boucle de robot principal non standard).

Lors de la sous-classification de `ProfiledPIDCommand`, des modifications supplémentaires (par exemple, l'activation d'une entrée continue) peuvent être apportées au contrôleur dans le corps du constructeur en appelant `getController()`.

Le paramètre measurementSource

The `measurementSource` parameter is a function (usually passed as a *lambda*) that returns the measurement of the process variable. Passing in the `measurementSource` function in `ProfiledPIDCommand` is functionally analogous to overriding the *getMeasurement()* function in `ProfiledPIDSubsystem`.

Lors de la sous-classe de `ProfiledPIDCommand`, les utilisateurs avancés peuvent modifier davantage les sources de mesures en modifiant le champ `m_measurement` de la classe.

Le paramètre goalSource

The `goalSource` parameter is a function (usually passed as a *lambda*) that returns the current goal state for the mechanism. If only a constant goal is needed, an overload exists that takes a constant goal rather than a supplier. Additionally, if goal velocities are desired to be zero, overloads exist that take a constant distance rather than a full profile state.

Lors de la sous-classe de `ProfiledPIDCommand`, les utilisateurs avancés peuvent modifier davantage le fournisseur de point de consigne en modifiant le champ `m_goal` de la classe.

Le paramètre useOutput

The `useOutput` parameter is a function (usually passed as a *lambda*) that consumes the output and setpoint state of the control loop. Passing in the `useOutput` function in `ProfiledPIDCommand` is functionally analogous to overriding the *useOutput()* function in `ProfiledPIDSubsystem`.

Lors de la sous-classe de `ProfiledPIDCommand`, les utilisateurs avancés peuvent modifier davantage l'état de sortie en modifiant le champ `m_useOutput` de la classe.

Les exigences

Comme toutes les commandes en ligne, `ProfiledPIDCommand` permet à l'utilisateur de spécifier ses exigences de sous-système en tant que paramètre constructeur.

Exemple complet de ProfiledPIDCommand

What does a `ProfiledPIDCommand` look like when used in practice ? The following examples are from the GyroDriveCommands example project (Java, C++) :

Java

```java
package edu.wpi.first.wpilibj.examples.gyrodrivecommands.commands;

import edu.wpi.first.math.controller.ProfiledPIDController;
import edu.wpi.first.math.trajectory.TrapezoidProfile;
import edu.wpi.first.wpilibj.examples.gyrodrivecommands.Constants.DriveConstants;
import edu.wpi.first.wpilibj.examples.gyrodrivecommands.subsystems.DriveSubsystem;
import edu.wpi.first.wpilibj2.command.ProfiledPIDCommand;

/** A command that will turn the robot to the specified angle using a motion profile. */
public class TurnToAngleProfiled extends ProfiledPIDCommand {
  /**
   * Turns to robot to the specified angle using a motion profile.
   *
   * @param targetAngleDegrees The angle to turn to
   * @param drive The drive subsystem to use
   */
  public TurnToAngleProfiled(double targetAngleDegrees, DriveSubsystem drive) {
    super(
        new ProfiledPIDController(
            DriveConstants.kTurnP,
            DriveConstants.kTurnI,
            DriveConstants.kTurnD,
            new TrapezoidProfile.Constraints(
                DriveConstants.kMaxTurnRateDegPerS,
                DriveConstants.kMaxTurnAccelerationDegPerSSquared)),
        // Close loop on heading
        drive::getHeading,
        // Set reference to target
        targetAngleDegrees,
        // Pipe output to turn robot
        (output, setpoint) -> drive.arcadeDrive(0, output),
        // Require the drive
        drive);

    // Set the controller to be continuous (because it is an angle controller)
    getController().enableContinuousInput(-180, 180);
    // Set the controller tolerance - the delta tolerance ensures the robot is stationary at the
    // setpoint before it is considered as having reached the reference
    getController()
        .setTolerance(DriveConstants.kTurnToleranceDeg, DriveConstants.kTurnRateToleranceDegPerS);
  }

  @Override
  public boolean isFinished() {
    // End when the controller is at the reference.
    return getController().atGoal();
```

(suite sur la page suivante)

(suite de la page précédente)

```
  }
}
```

C++ (Header)

```
#pragma once

#include <frc2/command/CommandHelper.h>
#include <frc2/command/ProfiledPIDCommand.h>

#include "subsystems/DriveSubsystem.h"

/**
 * A command that will turn the robot to the specified angle using a motion
 * profile.
 */
class TurnToAngleProfiled
    : public frc2::CommandHelper<frc2::ProfiledPIDCommand<units::radians>,
                                 TurnToAngleProfiled> {
 public:
  /**
   * Turns to robot to the specified angle using a motion profile.
   *
   * @param targetAngleDegrees The angle to turn to
   * @param drive              The drive subsystem to use
   */
  TurnToAngleProfiled(units::degree_t targetAngleDegrees,
                      DriveSubsystem* drive);

  bool IsFinished() override;
};
```

C++ (Source)

```
#include "commands/TurnToAngleProfiled.h"

#include <frc/controller/ProfiledPIDController.h>

using namespace DriveConstants;

TurnToAngleProfiled::TurnToAngleProfiled(units::degree_t target,
                                         DriveSubsystem* drive)
    : CommandHelper{
          frc::ProfiledPIDController<units::radians>{
              kTurnP, kTurnI, kTurnD, {kMaxTurnRate, kMaxTurnAcceleration}},
          // Close loop on heading
          [drive] { return drive->GetHeading(); },
          // Set reference to target
          target,
          // Pipe output to turn robot
          [drive](double output, auto setpointState) {
            drive->ArcadeDrive(0, output);
```

(suite sur la page suivante)

(suite de la page précédente)

```
          },
          // Require the drive
          {drive}} {
  // Set the controller to be continuous (because it is an angle controller)
  GetController().EnableContinuousInput(-180_deg, 180_deg);
  // Set the controller tolerance - the delta tolerance ensures the robot is
  // stationary at the setpoint before it is considered as having reached the
  // reference
  GetController().SetTolerance(kTurnTolerance, kTurnRateTolerance);

  AddRequirements(drive);
}

bool TurnToAngleProfiled::IsFinished() {
  return GetController().AtGoal();
}
```

26.13 Passing Functions As Parameters

In order to provide a concise inline syntax, the command-based library often accepts functions as parameters of constructors, factories, and decorators. Fortunately, both Java and C++ offer users the ability to *pass functions as objects* :

26.13.1 Method References (Java)

In Java, a reference to a function that can be passed as a parameter is called a method reference. The general syntax for a method reference is `object::method`. Note that no method parameters are included, since the method *itself* is passed. The method is not being called - it is being passed to another piece of code (in this case, a command) so that *that* code can call it when needed. For further information on method references, see *Références de méthodes*.

26.13.2 Lambda Expressions (Java)

While method references work well for passing a function that has already been written, often it is inconvenient/wasteful to write a function solely for the purpose of sending as a method reference, if that function will never be used elsewhere. To avoid this, Java also supports a feature called « lambda expressions. » A lambda expression is an inline method definition - it allows a function to be defined *inside of a parameter list*. For specifics on how to write Java lambda expressions, see *Expressions Lambda en Java*.

26.13.3 Lambda Expressions (C++)

Avertissement : Due to complications in C++ semantics, capturing `this` in a C++ lambda can cause a null pointer exception if done from a component command of a command composition. Whenever possible, C++ users should capture relevant command members explicitly and by value. For more details, see here.

C++ lacks a close equivalent to Java method references - pointers to member functions are generally not directly usable as parameters due to the presence of the implicit `this` parameter. However, C++ does offer lambda expressions - in addition, the lambda expressions offered by C++ are in many ways more powerful than those in Java. For specifics on how to write C++ lambda expressions, see *Expressions lambda en C++*.

27

Cinématique et Odométrie

27.1 Introduction to Kinematics and The ChassisSpeeds Class

Note : Kinematics and odometry uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.1.1 Qu'est-ce que la cinématique ?

The kinematics suite contains classes for differential drive, swerve drive, and mecanum drive kinematics and odometry. The kinematics classes help convert between a universal `ChassisSpeeds` (Java, C++, `Python`)object, containing linear and angular velocities for a robot to usable speeds for each individual type of drivetrain i.e. left and right wheel speeds for a differential drive, four wheel speeds for a mecanum drive, or individual module states (speed and angle) for a swerve drive.

27.1.2 Qu'est-ce que l'odométrie ?

L'odométrie consiste à utiliser des capteurs sur le robot pour créer une estimation de la position du robot sur le terrain de jeu. En FRC, ces capteurs sont généralement composés de plusieurs encodeurs (le nombre exact dépend du type d'entraînement) et d'un gyroscope pour mesurer l'angle du robot. Les classes d'odométrie utilisent les classes cinématiques ainsi que les entrées périodiques de l'utilisateur sur les vitesses (et les angles en cas de déviation) pour créer une estimation de l'emplacement du robot sur le terrain.

27.1.3 The ChassisSpeeds Class

L'objet `ChassisSpeeds` est essentiel à la nouvelle suite cinématique et odométrique WPILib. L'objet `ChassisSpeeds` représente les vitesses d'un châssis de robot. Cette structure a trois composants :

- `vx` : La vitesse du robot dans la direction x (avant).
- `vy` : La vitesse du robot dans la direction y (latérale). (Les valeurs positives signifient que le robot se déplace vers la gauche).
- `omega` : La vitesse angulaire du robot en radians par seconde.

Note : Une transmission non holonomique (c'est-à-dire une transmission qui ne peut pas se déplacer latéralement, ex : un entraînement différentiel) aura une composante `vy` de zéro en raison de son incapacité à se déplacer latéralement.

27.1.4 Construction d'un objet ChassisSpeeds

The constructor for the `ChassisSpeeds` object is very straightforward, accepting three arguments for `vx`, `vy`, and `omega`. In Java and Python, `vx` and `vy` must be in meters per second. In C++, the units library may be used to provide a linear velocity using any linear velocity unit.

JAVA

```java
// The robot is moving at 3 meters per second forward, 2 meters
// per second to the right, and rotating at half a rotation per
// second counterclockwise.
var speeds = new ChassisSpeeds(3.0, -2.0, Math.PI);
```

C++

```cpp
// The robot is moving at 3 meters per second forward, 2 meters
// per second to the right, and rotating at half a rotation per
// second counterclockwise.
frc::ChassisSpeeds speeds{3.0_mps, -2.0_mps,
  units::radians_per_second_t(std::numbers::pi)};
```

PYTHON

```python
import math
from wpimath.kinematics import ChassisSpeeds

# The robot is moving at 3 meters per second forward, 2 meters
# per second to the right, and rotating at half a rotation per
# second counterclockwise.
speeds = ChassisSpeeds(3.0, -2.0, math.pi)
```

27.1.5 Création d'un objet ChassisSpeeds à partir de vitesses relatives au terrain de jeu

Un objet `ChassisSpeeds` peut également être créé à partir d'un ensemble de vitesses relatives au terrain lorsque l'angle du robot est connu. L'objet utilise un ensemble de vitesses souhaitées par rapport au terrain (par exemple, vers la station d'alliance opposée, à droite sur le terrain) et calcule et génère un objet `ChassisSpeeds` qui représente des vitesses relatives au châssis du robot. Ceci est utile pour mettre en œuvre des contrôles orientés sur le terrain pour un robot avec un mécanisme de type « à embardée » ou Mécanum.

The static `ChassisSpeeds.fromFieldRelativeSpeeds` (Java / Python) / `ChassisSpeeds::FromFieldRelativeSpeeds` (C++) method can be used to generate the `ChassisSpeeds` object from field-relative speeds. This method accepts the `vx` (relative to the field), `vy` (relative to the field), `omega`, and the robot angle.

JAVA

```
// The desired field relative speed here is 2 meters per second
// toward the opponent's alliance station wall, and 2 meters per
// second toward the left field boundary. The desired rotation
// is a quarter of a rotation per second counterclockwise. The current
// robot angle is 45 degrees.
ChassisSpeeds speeds = ChassisSpeeds.fromFieldRelativeSpeeds(
  2.0, 2.0, Math.PI / 2.0, Rotation2d.fromDegrees(45.0));
```

C++

```
// The desired field relative speed here is 2 meters per second
// toward the opponent's alliance station wall, and 2 meters per
// second toward the left field boundary. The desired rotation
// is a quarter of a rotation per second counterclockwise. The current
// robot angle is 45 degrees.
frc::ChassisSpeeds speeds = frc::ChassisSpeeds::FromFieldRelativeSpeeds(
  2_mps, 2_mps, units::radians_per_second_t(std::numbers::pi / 2.0), Rotation2d(45_
↪deg));
```

PYTHON

```
import math
from wpimath.kinematics import ChassisSpeeds
from wpimath.geometry  import Rotation2d

# The desired field relative speed here is 2 meters per second
# toward the opponent's alliance station wall, and 2 meters per
# second toward the left field boundary. The desired rotation
# is a quarter of a rotation per second counterclockwise. The current
# robot angle is 45 degrees.
speeds = ChassisSpeeds.fromFieldRelativeSpeeds(
  2.0, 2.0, math.pi / 2.0, Rotation2d.fromDegrees(45.0))
```

Note : La vitesse angulaire n'est pas explicitement indiquée comme étant « relative au terrain » car la vitesse angulaire est la même que celle mesurée du point de vue du terrain ou du robot.

27.2 Cinématique d'entraînement différentiel

La classe `DifferentialDriveKinematics` est un outil utile qui convertit entre un objet `ChassisSpeeds` et un objet `DifferentialDriveWheelSpeeds`, qui contient des vitesses pour les côtés gauche et droit d'un robot à entraînement différentiel.

27.2.1 Construction de l'objet cinématique

L'objet `DifferentialDriveKinematics` accepte un argument constructeur, qui est la largeur de piste du robot. Cela représente la distance entre les deux trains de roues sur un entraînement différentiel.

Note : In Java and Python, the track width must be in meters. In C++, the units library can be used to pass in the track width using any length unit.

27.2.2 Conversion de Chassis Speeds vers Wheel Speeds

The `toWheelSpeeds(ChassisSpeeds speeds)` (Java / Python) / `ToWheelSpeeds(ChassisSpeeds speeds)` (C++) method should be used to convert a `ChassisSpeeds` object to a `DifferentialDriveWheelSpeeds` object. This is useful in situations where you have to convert a linear velocity (`vx`) and an angular velocity (`omega`) to left and right wheel velocities.

JAVA

```
// Creating my kinematics object: track width of 27 inches
DifferentialDriveKinematics kinematics =
  new DifferentialDriveKinematics(Units.inchesToMeters(27.0));

// Example chassis speeds: 2 meters per second linear velocity,
// 1 radian per second angular velocity.
var chassisSpeeds = new ChassisSpeeds(2.0, 0, 1.0);

// Convert to wheel speeds
DifferentialDriveWheelSpeeds wheelSpeeds = kinematics.toWheelSpeeds(chassisSpeeds);

// Left velocity
double leftVelocity = wheelSpeeds.leftMetersPerSecond;

// Right velocity
double rightVelocity = wheelSpeeds.rightMetersPerSecond;
```

C++

```cpp
// Creating my kinematics object: track width of 27 inches
frc::DifferentialDriveKinematics kinematics{27_in};

// Example chassis speeds: 2 meters per second linear velocity,
// 1 radian per second angular velocity.
frc::ChassisSpeeds chassisSpeeds{2_mps, 0_mps, 1_rad_per_s};

// Convert to wheel speeds. Here, we can use C++17's structured bindings
// feature to automatically split the DifferentialDriveWheelSpeeds
// struct into left and right velocities.
auto [left, right] = kinematics.ToWheelSpeeds(chassisSpeeds);
```

PYTHON

```python
from wpimath.kinematics import DifferentialDriveKinematics
from wpimath.kinematics import ChassisSpeeds
from wpimath.units import inchesToMeters

# Creating my kinematics object: track width of 27 inches
kinematics = DifferentialDriveKinematics(Units.inchesToMeters(27.0))

# Example chassis speeds: 2 meters per second linear velocity,
# 1 radian per second angular velocity.
chassisSpeeds = ChassisSpeeds(2.0, 0, 1.0)

# Convert to wheel speeds
wheelSpeeds = kinematics.toWheelSpeeds(chassisSpeeds)

# Left velocity
leftVelocity = wheelSpeeds.left
# Right velocity
rightVelocity = wheelSpeeds.right
```

27.2.3 Conversion de Wheel Speeds vers Chassis Speeds

One can also use the kinematics object to convert individual wheel speeds (left and right) to a singular `ChassisSpeeds` object. The `toChassisSpeeds(DifferentialDriveWheelSpeeds speeds)` (Java / Python) / `ToChassisSpeeds(DifferentialDriveWheelSpeeds speeds)` (C++) method should be used to achieve this.

JAVA

```java
// Creating my kinematics object: track width of 27 inches
DifferentialDriveKinematics kinematics =
  new DifferentialDriveKinematics(Units.inchesToMeters(27.0));

// Example differential drive wheel speeds: 2 meters per second
// for the left side, 3 meters per second for the right side.
var wheelSpeeds = new DifferentialDriveWheelSpeeds(2.0, 3.0);

// Convert to chassis speeds.
ChassisSpeeds chassisSpeeds = kinematics.toChassisSpeeds(wheelSpeeds);

// Linear velocity
double linearVelocity = chassisSpeeds.vxMetersPerSecond;

// Angular velocity
double angularVelocity = chassisSpeeds.omegaRadiansPerSecond;
```

C++

```cpp
// Creating my kinematics object: track width of 27 inches
frc::DifferentialDriveKinematics kinematics{27_in};

// Example differential drive wheel speeds: 2 meters per second
// for the left side, 3 meters per second for the right side.
frc::DifferentialDriveWheelSpeeds wheelSpeeds{2_mps, 3_mps};

// Convert to chassis speeds. Here we can use C++17's structured bindings
// feature to automatically split the ChassisSpeeds struct into its 3 components.
// Note that because a differential drive is non-holonomic, the vy variable
// will be equal to zero.
auto [linearVelocity, vy, angularVelocity] = kinematics.ToChassisSpeeds(wheelSpeeds);
```

PYTHON

```python
from wpimath.kinematics import DifferentialDriveKinematics
from wpimath.kinematics import DifferentialDriveWheelSpeeds
from wpimath.units import inchesToMeters

# Creating my kinematics object: track width of 27 inches
kinematics = DifferentialDriveKinematics(inchesToMeters(27.0))

# Example differential drive wheel speeds: 2 meters per second
# for the left side, 3 meters per second for the right side.
wheelSpeeds = DifferentialDriveWheelSpeeds(2.0, 3.0)

# Convert to chassis speeds.
chassisSpeeds = kinematics.toChassisSpeeds(wheelSpeeds)

# Linear velocity
linearVelocity = chassisSpeeds.vx
```

(suite sur la page suivante)

(suite de la page précédente)

```
# Angular velocity
angularVelocity = chassisSpeeds.omega
```

27.3 Odométrie à entraînement différentiel

Un utilisateur peut utiliser les classes cinématiques d'entraînement différentiel afin d'effectuer l'" *Odométrie*. WPILib contient une classe `DifferentialDriveOdometry` qui peut être utilisée pour détecter la position d'un robot avec entraînement différentiel sur le terrain.

Note : Étant donné que cette méthode utilise seulement que des encodeurs et un gyroscope, l'estimation de la position du robot sur le terrain va dévier avec le temps, surtout quand votre robot va entrer en contact avec des objets ou autres robots pendant le jeu. Cependant, l'odométrie est généralement très précise pendant la période autonome.

27.3.1 Création de l'objet Odometry

The `DifferentialDriveOdometry` class constructor requires three mandatory arguments and one optional argument. The mandatory arguments are :

- The angle reported by your gyroscope (as a `Rotation2d`)
- The initial left and right encoder readings. In Java / Python, these are a number that represents the distance traveled by each side in meters. In C++, the *units library* must be used to represent your wheel positions.

The optional argument is the starting pose of your robot on the field (as a `Pose2d`). By default, the robot will start at `x = 0, y = 0, theta = 0`.

Note : 0 degrees / radians represents the robot angle when the robot is facing directly toward your opponent's alliance station. As your robot turns to the left, your gyroscope angle should increase. The `Gyro` interface supplies `getRotation2d`/`GetRotation2d` that you can use for this purpose. See *Coordinate System* for more information about the coordinate system.

JAVA

```
// Creating my odometry object. Here,
// our starting pose is 5 meters along the long end of the field and in the
// center of the field along the short end, facing forward.
DifferentialDriveOdometry m_odometry = new DifferentialDriveOdometry(
  m_gyro.getRotation2d(),
  m_leftEncoder.getDistance(), m_rightEncoder.getDistance(),
  new Pose2d(5.0, 13.5, new Rotation2d()));
```

C++

```
// Creating my odometry object. Here,
// our starting pose is 5 meters along the long end of the field and in the
// center of the field along the short end, facing forward.
frc::DifferentialDriveOdometry m_odometry{
  m_gyro.GetRotation2d(),
  units::meter_t{m_leftEncoder.GetDistance()},
  units::meter_t{m_rightEncoder.GetDistance()},
  frc::Pose2d{5_m, 13.5_m, 0_rad}};
```

PYTHON

```
from wpimath.kinematics import DifferentialDriveOdometry
from wpimath.geometry import Pose2d
from wpimath.geometry import Rotation2d

# Creating my odometry object. Here,
# our starting pose is 5 meters along the long end of the field and in the
# center of the field along the short end, facing forward.
m_odometry = DifferentialDriveOdometry(
  m_gyro.getRotation2d(),
  m_leftEncoder.getDistance(), m_rightEncoder.getDistance(),
  Pose2d(5.0, 13.5, Rotation2d()))
```

27.3.2 Mise à jour de Robot Pose

La méthode Update peut être utilisée pour mettre à jour la position du robot sur le terrain. Cette méthode doit être appelée périodiquement, de préférence dans la méthode `periodic()` du *Sous-système*. La méthode Update renvoie la toute dernière pose du robot. Cette méthode prend en compte l'angle gyroscopique du robot, ainsi que le compte (distance) de l'encodeur gauche et celui de l'encodeur droit.

Note : If the robot is moving forward in a straight line, **both** distances (left and right) must be increasing positively – the rate of change must be positive.

JAVA

```
@Override
public void periodic() {
  // Get the rotation of the robot from the gyro.
  var gyroAngle = m_gyro.getRotation2d();

  // Update the pose
  m_pose = m_odometry.update(gyroAngle,
    m_leftEncoder.getDistance(),
    m_rightEncoder.getDistance());
}
```

C++

```
void Periodic() override {
  // Get the rotation of the robot from the gyro.
  frc::Rotation2d gyroAngle = m_gyro.GetRotation2d();

  // Update the pose
  m_pose = m_odometry.Update(gyroAngle,
    units::meter_t{m_leftEncoder.GetDistance()},
    units::meter_t{m_rightEncoder.GetDistance()});
}
```

PYTHON

```
def periodic(self):
  # Get the rotation of the robot from the gyro.
  gyroAngle = m_gyro.getRotation2d()

  # Update the pose
  m_pose = m_odometry.update(gyroAngle,
    m_leftEncoder.getDistance(),
    m_rightEncoder.getDistance())
```

27.3.3 Réinitialisation de la pose de robot

The robot pose can be reset via the `resetPosition` method. This method accepts four arguments : the current gyro angle, the left and right wheel positions, and the new field-relative pose.

Important : If at any time, you decide to reset your gyroscope or encoders, the `resetPosition` method MUST be called with the new gyro angle and wheel distances.

Note : A full example of a differential drive robot with odometry is available here : C++ / Java / Python

---

In addition, the `GetPose` (C++) / `getPoseMeters` (Java / Python) methods can be used to retrieve the current robot pose without an update.

27.4 La cinématique de l'entraînement de type « à embardée » (Swerve)

N.D.T. (le mot « Swerve » sera utilisé tout le long de cette page, afin d'alléger le texte). La classe `SwerveDriveKinematics` est un outil utile qui convertit entre un objet `ChassisSpeeds` et plusieurs objets `SwerveModuleState`, qui contient les vitesses et les angles pour chaque module Swerve d'un robot avec entraînement de ce type.

Note : Swerve drive kinematics uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.4.1 La classe d'état du module swerve

La classe `SwerveModuleState` contient des informations sur la vitesse et l'angle d'un seul module Swerve. Le constructeur d'un `SwerveModuleState` prend deux arguments, la vitesse de la roue sur le module et l'angle du module.

Note : In Java / Python, the velocity of the wheel must be in meters per second. In C++, the units library can be used to provide the velocity using any linear velocity unit.

Note : Un angle de 0 correspond aux modules tournés vers l'avant.

27.4.2 Construction de l'objet cinématique

La classe `SwerveDriveKinematics` accepte un nombre variable d'arguments constructeur, chaque argument étant l'emplacement d'un module swerve par rapport au centre du robot (sous la forme d'un `Translation2d`. Le nombre d'arguments constructeur correspond au nombre de modules Swerve.

Note : Un robot doit avoir au moins 2 modules Swerve.

Note : En C ++, la classe est basée sur le nombre de modules. Par conséquent, lors de la construction d'un objet `SwerveDriveKinematics` en tant que variable membre d'une classe, le nombre de modules doit être transmis en tant qu'argument. Par exemple, pour un entraînement Swerve typique avec quatre modules, l'objet cinématique doit être construit comme suit : `frc::SwerveDriveKinematics<4> m_kinematics{...}`.

Les emplacements des modules doivent être relatifs au centre du robot. Les valeurs x positives représentent le déplacement vers l'avant du robot tandis que les valeurs y positives représentent le déplacement vers la gauche du robot.

JAVA

```
// Locations for the swerve drive modules relative to the robot center.
Translation2d m_frontLeftLocation = new Translation2d(0.381, 0.381);
Translation2d m_frontRightLocation = new Translation2d(0.381, -0.381);
Translation2d m_backLeftLocation = new Translation2d(-0.381, 0.381);
Translation2d m_backRightLocation = new Translation2d(-0.381, -0.381);

// Creating my kinematics object using the module locations
```

(suite sur la page suivante)

(suite de la page précédente)

```
SwerveDriveKinematics m_kinematics = new SwerveDriveKinematics(
  m_frontLeftLocation, m_frontRightLocation, m_backLeftLocation, m_backRightLocation
);
```

C++

```
// Locations for the swerve drive modules relative to the robot center.
frc::Translation2d m_frontLeftLocation{0.381_m, 0.381_m};
frc::Translation2d m_frontRightLocation{0.381_m, -0.381_m};
frc::Translation2d m_backLeftLocation{-0.381_m, 0.381_m};
frc::Translation2d m_backRightLocation{-0.381_m, -0.381_m};

// Creating my kinematics object using the module locations.
frc::SwerveDriveKinematics<4> m_kinematics{
  m_frontLeftLocation, m_frontRightLocation, m_backLeftLocation,
  m_backRightLocation};
```

PYTHON

```
# Python requires using the right class for the number of modules you have

from wpimath.geometry import Translation2d
from wpimath.kinematics import SwerveDrive4Kinematics

# Locations for the swerve drive modules relative to the robot center.
frontLeftLocation = Translation2d(0.381, 0.381)
frontRightLocation = Translation2d(0.381, -0.381)
backLeftLocation = Translation2d(-0.381, 0.381)
backRightLocation = Translation2d(-0.381, -0.381)

# Creating my kinematics object using the module locations
self.kinematics = SwerveDrive4Kinematics(
  frontLeftLocation, frontRightLocation, backLeftLocation, backRightLocation
)
```

27.4.3 Conversion des vitesses de châssis en états de module

The `toSwerveModuleStates(ChassisSpeeds speeds)` (Java / Python) / `ToSwerveModuleStates(ChassisSpeeds speeds)` (C++) method should be used to convert a `ChassisSpeeds` object to a an array of `SwerveModuleState` objects. This is useful in situations where you have to convert a forward velocity, sideways velocity, and an angular velocity into individual module states.

Les éléments du tableau renvoyé par cette méthode sont du même ordre dans lequel l'objet cinématique a été construit. Par exemple, si l'objet cinématique a été construit avec l'emplacement du module avant-gauche, l'emplacement du module avant-droit, l'emplacement du module arrière-gauche et l'emplacement du module arrière-droit dans cet ordre, les éléments du tableau seront : état du module avant-gauche, état du module avant-droit, état du module arrière-gauche et état du module arrière-droit (en respectant cet ordre).

JAVA

```
// Example chassis speeds: 1 meter per second forward, 3 meters
// per second to the left, and rotation at 1.5 radians per second
// counterclockwise.
ChassisSpeeds speeds = new ChassisSpeeds(1.0, 3.0, 1.5);

// Convert to module states
SwerveModuleState[] moduleStates = kinematics.toSwerveModuleStates(speeds);

// Front left module state
SwerveModuleState frontLeft = moduleStates[0];

// Front right module state
SwerveModuleState frontRight = moduleStates[1];

// Back left module state
SwerveModuleState backLeft = moduleStates[2];

// Back right module state
SwerveModuleState backRight = moduleStates[3];
```

C++

```
// Example chassis speeds: 1 meter per second forward, 3 meters
// per second to the left, and rotation at 1.5 radians per second
// counterclockwise.
frc::ChassisSpeeds speeds{1_mps, 3_mps, 1.5_rad_per_s};

// Convert to module states. Here, we can use C++17's structured
// bindings feature to automatically split up the array into its
// individual SwerveModuleState components.
auto [fl, fr, bl, br] = kinematics.ToSwerveModuleStates(speeds);
```

PYTHON

```
from wpimath.kinematics import ChassisSpeeds

# Example chassis speeds: 1 meter per second forward, 3 meters
# per second to the left, and rotation at 1.5 radians per second
# counterclockwise.
speeds = ChassisSpeeds(1.0, 3.0, 1.5)

# Convert to module states
frontLeft, frontRight, backLeft, backRight = self.kinematics.
↪toSwerveModuleStates(speeds)
```

Module d'optimisation de l'angle

La classe `SwerveModuleState` contient une méthode statique `optimize()` (Java) / `Optimize()` (C++) qui est utilisée pour « optimiser » les points de consigne de vitesse et d'angle pour un `SwerveModuleState` donné afin de minimiser le changement d'orientation angulaire. Par exemple, si le point de consigne angulaire d'un certain module de cinématique inverse est de 90 degrés, mais que votre angle actuel est de -89 degrés, cette méthode annulera automatiquement la vitesse du point de consigne du module et fournira le point de consigne angulaire -90 degrés pour réduire la distance parcourue par le module.

Cette méthode prend deux paramètres : l'état désiré (généralement par l'intermédiaire de la méthode `toSwerveModuleStates`) et l'angle actuel. Il retournera le nouvel état optimisé que vous pouvez utiliser comme point de consigne dans votre boucle de contrôle de rétroaction.

JAVA

```
var frontLeftOptimized = SwerveModuleState.optimize(frontLeft,
   new Rotation2d(m_turningEncoder.getDistance()));
```

C++

```
auto flOptimized = frc::SwerveModuleState::Optimize(fl,
   units::radian_t(m_turningEncoder.GetDistance()));
```

PYTHON

```
from wpimath.kinematics import SwerveModuleState
from wpimath.geometry import Rotation2d

frontLeftOptimized = SwerveModuleState.optimize(frontLeft,
   Rotation2d(self.m_turningEncoder.getDistance()))
```

Entraînement orienté terrain

Recall peut être créé à partir d'un ensemble de vitesses orientées en fonction du terrain de jeu. Cette fonction peut être utilisée également pour obtenir les vitesses de roue à partir du même ensemble (de vitesses orientées en fonction du terrain de jeu).

JAVA

```
// The desired field relative speed here is 2 meters per second
// toward the opponent's alliance station wall, and 2 meters per
// second toward the left field boundary. The desired rotation
// is a quarter of a rotation per second counterclockwise. The current
// robot angle is 45 degrees.
ChassisSpeeds speeds = ChassisSpeeds.fromFieldRelativeSpeeds(
  2.0, 2.0, Math.PI / 2.0, Rotation2d.fromDegrees(45.0));
```

(suite sur la page suivante)

(suite de la page précédente)

```
// Now use this in our kinematics
SwerveModuleState[] moduleStates = kinematics.toSwerveModuleStates(speeds);
```

C++

```
// The desired field relative speed here is 2 meters per second
// toward the opponent's alliance station wall, and 2 meters per
// second toward the left field boundary. The desired rotation
// is a quarter of a rotation per second counterclockwise. The current
// robot angle is 45 degrees.
frc::ChassisSpeeds speeds = frc::ChassisSpeeds::FromFieldRelativeSpeeds(
  2_mps, 2_mps, units::radians_per_second_t(std::numbers::pi / 2.0), Rotation2d(45_
↪deg));

// Now use this in our kinematics
auto [fl, fr, bl, br] = kinematics.ToSwerveModuleStates(speeds);
```

PYTHON

```
from wpimath.kinematics import ChassisSpeeds
import math
from wpimath.geometry import Rotation2d

# The desired field relative speed here is 2 meters per second
# toward the opponent's alliance station wall, and 2 meters per
# second toward the left field boundary. The desired rotation
# is a quarter of a rotation per second counterclockwise. The current
# robot angle is 45 degrees.
speeds = ChassisSpeeds.fromFieldRelativeSpeeds(
  2.0, 2.0, math.pi / 2.0, Rotation2d.fromDegrees(45.0))

# Now use this in our kinematics
self.moduleStates = self.kinematics.toSwerveModuleStates(speeds)
```

Utilisation de centres de rotation personnalisés

Parfois, la rotation autour d'un point spécifique peut être souhaitable pour certaines manœuvres d'évitement. Ce type de comportement est également pris en charge par les classes WPILib. La même méthode `ToSwerveModuleStates()` accepte un deuxième paramètre pour le centre de rotation (comme un `Translation2d`). Tout comme les emplacements des roues, le `Translation2d` représentant le centre de rotation doit être relatif au centre du robot.

Note : Étant donné que tous les robots ont un châssis rigide, les vitesses `vx` et `vy` fournies par l'objet `ChassisSpeeds` s'appliqueront toujours à l'intégralité du robot. Cependant, `omega` de l'objet `ChassisSpeeds` sera mesuré à partir du centre de rotation.

Par exemple, on peut définir le centre de rotation sur un certain module et si l'objet `ChassisSpeeds fourni a un vx et un vy de zéro et un omega différent de zéro, le robot tournerar autour de ce module Swerve spécifique.

27.4.4 Conversion des états des modules en vitesses de châssis

One can also use the kinematics object to convert an array of `SwerveModuleState` objects to a singular `ChassisSpeeds` object. The `toChassisSpeeds(SwerveModuleState... states)` (Java / Python) / `ToChassisSpeeds(SwerveModuleState... states)` (C++) method can be used to achieve this.

JAVA

```
// Example module states
var frontLeftState = new SwerveModuleState(23.43, Rotation2d.fromDegrees(-140.19));
var frontRightState = new SwerveModuleState(23.43, Rotation2d.fromDegrees(-39.81));
var backLeftState = new SwerveModuleState(54.08, Rotation2d.fromDegrees(-109.44));
var backRightState = new SwerveModuleState(54.08, Rotation2d.fromDegrees(-70.56));

// Convert to chassis speeds
ChassisSpeeds chassisSpeeds = kinematics.toChassisSpeeds(
  frontLeftState, frontRightState, backLeftState, backRightState);

// Getting individual speeds
double forward = chassisSpeeds.vxMetersPerSecond;
double sideways = chassisSpeeds.vyMetersPerSecond;
double angular = chassisSpeeds.omegaRadiansPerSecond;
```

C++

```
// Example module States
frc::SwerveModuleState frontLeftState{23.43_mps, Rotation2d(-140.19_deg)};
frc::SwerveModuleState frontRightState{23.43_mps, Rotation2d(-39.81_deg)};
frc::SwerveModuleState backLeftState{54.08_mps, Rotation2d(-109.44_deg)};
frc::SwerveModuleState backRightState{54.08_mps, Rotation2d(-70.56_deg)};

// Convert to chassis speeds. Here, we can use C++17's structured bindings
// feature to automatically break up the ChassisSpeeds struct into its
// three components.
auto [forward, sideways, angular] = kinematics.ToChassisSpeeds(
  frontLeftState, frontRightState, backLeftState, backRightState);
```

PYTHON

```python
from wpimath.kinematics import SwerveModuleState
from wpimath.geometry import Rotation2d

# Example module states
frontLeftState = SwerveModuleState(23.43, Rotation2d.fromDegrees(-140.19))
frontRightState = SwerveModuleState(23.43, Rotation2d.fromDegrees(-39.81))
backLeftState = SwerveModuleState(54.08, Rotation2d.fromDegrees(-109.44))
backRightState = SwerveModuleState(54.08, Rotation2d.fromDegrees(-70.56))

# Convert to chassis speeds
chassisSpeeds = self.kinematics.toChassisSpeeds(
  frontLeftState, frontRightState, backLeftState, backRightState)

# Getting individual speeds
forward = chassisSpeeds.vx
sideways = chassisSpeeds.vy
angular = chassisSpeeds.omega
```

27.4.5 Module state visualization with AdvantageScope

By recording a set of swerve module states using *NetworkTables* or *WPILib data logs*, *AdvantageScope* can be used to visualize the state of a swerve drive. The code below shows how a set of `SwerveModuleState` objects can be published to NetworkTables.

JAVA

```java
public class Example {
  private final StructArrayPublisher<SwerveModuleState> publisher;

  public Example() {
    // Start publishing an array of module states with the "/SwerveStates" key
    publisher = NetworkTableInstance.getDefault()
      .getStructArrayTopic("/SwerveStates", SwerveModuleState.struct).publish();
  }

  public void periodic() {
    // Periodically send a set of module states
    publisher.set(new SwerveModuleState[] {
      frontLeftState,
      frontRightState,
      backLeftState,
      backRightState
    });
  }
}
```

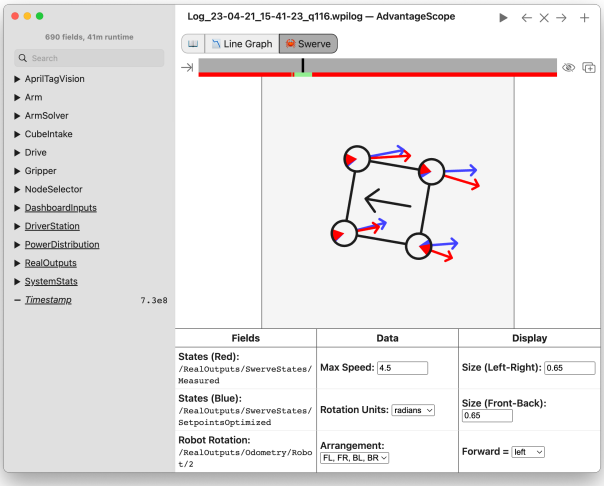
C++

```
class Example {
  nt::StructArrayPublisher<frc::SwerveModuleState> publisher

 public:
  Example() {
    // Start publishing an array of module states with the "/SwerveStates" key
    publisher = nt::NetworkTableInstance::GetDefault()
      .GetStructArrayTopic<frc::SwerveModuleState>("/SwerveStates").Publish();
  }

  void Periodic() {
    // Periodically send a set of module states
    swervePublisher.Set(
      std::vector{
        frontLeftState,
        frontRightState,
        backLeftState,
        backRightState
      }
    );
  }
};
```

PYTHON

```
import ntcore
from wpimath.kinematics import SwerveModuleState

# get the default instance of NetworkTables
nt = ntcore.NetworkTableInstance.getDefault()

# Start publishing an array of module states with the "/SwerveStates" key
topic = nt.getStructArrayTopic("/SwerveStates", SwerveModuleState)
self.pub = topic.publish()

def periodic(self):
  # Periodically send a set of module states
  self.pub.set([frontLeftState,frontRightState,backLeftState,backRightState])
```

See the documentation for the swerve tab for more details on visualizing this data using AdvantageScope.



27.5 Odométrie pour l'entaînement de type « à embardée » (Swerve)

Un utilisateur peut utiliser les classes cinématiques « swerve drive » pour effectuer l'"*odométrie*. WPILib contient une classe `SwerveDriveOdometry` qui peut être utilisée pour localiser la position d'un robot avec entraînement par embardée sur le terrain.

Note : Étant donné que cette méthode n'utilise que des encodeurs et un gyroscope, l'estimation de la position du robot sur le terrain dérivera avec le temps, d'autant plus que votre robot entre en contact avec d'autres robots pendant le jeu. Cependant, l'odométrie est généralement très précise pendant la période autonome.

27.5.1 Création de l'objet odométrie

The `SwerveDriveOdometry<int NumModules>` class constructor requires one template argument (only C++), three mandatory arguments, and one optional argument. The template argument (only C++) is an integer representing the number of swerve modules.

The mandatory arguments are :

- The kinematics object that represents your swerve drive (as a `SwerveDriveKinematics` instance)
- The angle reported by your gyroscope (as a `Rotation2d`)
- The initial positions of the swerve modules (as an array of `SwerveModulePosition`). In Java, this must be constructed with each wheel position in meters. In C++, the *units library* must be used to represent your wheel positions. It is important that the order in which you pass the `SwerveModulePosition` objects is the same as the order in which you created the kinematics object.

The fourth optional argument is the starting pose of your robot on the field (as a `Pose2d`). By default, the robot will start at `x = 0, y = 0, theta = 0`.

Note : 0 degrees / radians represents the robot angle when the robot is facing directly toward your opponent's alliance station. As your robot turns to the left, your gyroscope angle should increase. The `Gyro` interface supplies `getRotation2d`/`GetRotation2d` that you can use for this purpose. See *Coordinate System* for more information about the coordinate system.

JAVA

```
// Locations for the swerve drive modules relative to the robot center.
Translation2d m_frontLeftLocation = new Translation2d(0.381, 0.381);
Translation2d m_frontRightLocation = new Translation2d(0.381, -0.381);
Translation2d m_backLeftLocation = new Translation2d(-0.381, 0.381);
Translation2d m_backRightLocation = new Translation2d(-0.381, -0.381);

// Creating my kinematics object using the module locations
SwerveDriveKinematics m_kinematics = new SwerveDriveKinematics(
  m_frontLeftLocation, m_frontRightLocation, m_backLeftLocation, m_backRightLocation
);

// Creating my odometry object from the kinematics object and the initial wheel␣
↪positions.
// Here, our starting pose is 5 meters along the long end of the field and in the
// center of the field along the short end, facing the opposing alliance wall.
SwerveDriveOdometry m_odometry = new SwerveDriveOdometry(
  m_kinematics, m_gyro.getRotation2d(),
  new SwerveModulePosition[] {
    m_frontLeftModule.getPosition(),
    m_frontRightModule.getPosition(),
    m_backLeftModule.getPosition(),
    m_backRightModule.getPosition()
  }, new Pose2d(5.0, 13.5, new Rotation2d()));
```

C++

```
// Locations for the swerve drive modules relative to the robot center.
frc::Translation2d m_frontLeftLocation{0.381_m, 0.381_m};
frc::Translation2d m_frontRightLocation{0.381_m, -0.381_m};
frc::Translation2d m_backLeftLocation{-0.381_m, 0.381_m};
frc::Translation2d m_backRightLocation{-0.381_m, -0.381_m};

// Creating my kinematics object using the module locations.
frc::SwerveDriveKinematics<4> m_kinematics{
  m_frontLeftLocation, m_frontRightLocation,
  m_backLeftLocation, m_backRightLocation
};

// Creating my odometry object from the kinematics object. Here,
// our starting pose is 5 meters along the long end of the field and in the
// center of the field along the short end, facing forward.
frc::SwerveDriveOdometry<4> m_odometry{m_kinematics, m_gyro.GetRotation2d(),
  {m_frontLeft.GetPosition(), m_frontRight.GetPosition(),
  m_backLeft.GetPosition(), m_backRight.GetPosition()},
  frc::Pose2d{5_m, 13.5_m, 0_rad}};
```

PYTHON

```
# Python requires using the right class for the number of modules you have
# For both the Kinematics and Odometry classes

from wpimath.geometry import Translation2d
from wpimath.kinematics import SwerveDrive4Kinematics
from wpimath.kinematics import SwerveDrive4Odometry
from wpimath.geometry import Pose2d
from wpimath.geometry import Rotation2d

class MyRobot:
  def robotInit(self):
    # Locations for the swerve drive modules relative to the robot center.
    frontLeftLocation = Translation2d(0.381, 0.381)
    frontRightLocation = Translation2d(0.381, -0.381)
    backLeftLocation = Translation2d(-0.381, 0.381)
    backRightLocation = Translation2d(-0.381, -0.381)

    # Creating my kinematics object using the module locations
    self.kinematics = SwerveDrive4Kinematics(
      frontLeftLocation, frontRightLocation, backLeftLocation, backRightLocation
    )

    # Creating my odometry object from the kinematics object and the initial wheel␣
↪positions.
    # Here, our starting pose is 5 meters along the long end of the field and in the
    # center of the field along the short end, facing the opposing alliance wall.
    self.odometry = SwerveDrive4Odometry(
      self.kinematics, self.gyro.getRotation2d(),
      (
        self.frontLeftModule.getPosition(),
        self.frontRightModule.getPosition(),
        self.backLeftModule.getPosition(),
        self.backRightModule.getPosition()
      ),
      Pose2d(5.0, 13.5, Rotation2d()))
```

27.5.2 Mise à jour de la pose du robot

The `update` method of the odometry class updates the robot position on the field. The update method takes in the gyro angle of the robot, along with an array of `SwerveModulePosition` objects. It is important that the order in which you pass the `SwerveModulePosition` objects is the same as the order in which you created the kinematics object.

Cette méthode `Update` doit être appelée périodiquement, de préférence dans la méthode `periodic()` du *Sous-Système*. La méthode `update` envoie la nouvelle pose du robot.

JAVA

```
@Override
public void periodic() {
  // Get the rotation of the robot from the gyro.
  var gyroAngle = m_gyro.getRotation2d();

  // Update the pose
  m_pose = m_odometry.update(gyroAngle,
    new SwerveModulePosition[] {
      m_frontLeftModule.getPosition(), m_frontRightModule.getPosition(),
      m_backLeftModule.getPosition(), m_backRightModule.getPosition()
    });
}
```

C++

```
void Periodic() override {
  // Get the rotation of the robot from the gyro.
  frc::Rotation2d gyroAngle = m_gyro.GetRotation2d();

  // Update the pose
  m_pose = m_odometry.Update(gyroAngle,
    {
      m_frontLeftModule.GetPosition(), m_frontRightModule.GetPosition(),
      m_backLeftModule.GetPosition(), m_backRightModule.GetPosition()
    };
  )
}
```

PYTHON

```
def periodic(self):
  # Get the rotation of the robot from the gyro.
  self.gyroAngle = self.gyro.getRotation2d()

  # Update the pose
  self.pose = self.odometry.update(self.gyroAngle,
      self.frontLeftModule.getPosition(), self.frontRightModule.getPosition(),
      self.backLeftModule.getPosition(), self.backRightModule.getPosition()
  )
```

27.5.3 Réinitialisation de la pose de robot

The robot pose can be reset via the `resetPosition` method. This method accepts three arguments : the current gyro angle, an array of the current module positions (as in the constructor and update method), and the new field-relative pose.

Important : If at any time, you decide to reset your gyroscope or wheel encoders, the `resetPosition` method MUST be called with the new gyro angle and wheel encoder positions.

Note : The implementation of `getPosition() / GetPosition()` above is left to the user. The idea is to get the module position (distance and angle) from each module. For a full example, see here : C++ / Java / Python

---

In addition, the `GetPose` (C++) / `getPoseMeters` (Java / Python) methods can be used to retrieve the current robot pose without an update.

27.6 La cinématique de l'entraînement Mécanum

La classe `MecanumDriveKinematics` est un outil utile qui convertit entre un objet `ChassisSpeeds` et un objet `MecanumDriveWheelSpeeds` , qui contient les vitesses pour chacune des quatre roues sur un entraînement mécanique.

Note : Mecanum kinematics uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.6.1 Construction de l'objet cinématique

La classe `MecanumDriveKinematics` accepte quatre arguments de constructeur, chaque argument étant l'emplacement d'une roue par rapport au centre du robot (en tant que `Translation2d`). L'ordre des arguments est avant gauche, avant droit, arrière gauche et arrière droit. L'emplacement des roues doit être relatif au centre du robot. Les valeurs x positives représentent le déplacement vers l'avant du robot tandis que les valeurs y positives représentent le déplacement vers la gauche du robot.

JAVA

```
// Locations of the wheels relative to the robot center.
Translation2d m_frontLeftLocation = new Translation2d(0.381, 0.381);
Translation2d m_frontRightLocation = new Translation2d(0.381, -0.381);
Translation2d m_backLeftLocation = new Translation2d(-0.381, 0.381);
Translation2d m_backRightLocation = new Translation2d(-0.381, -0.381);

// Creating my kinematics object using the wheel locations.
MecanumDriveKinematics m_kinematics = new MecanumDriveKinematics(
  m_frontLeftLocation, m_frontRightLocation, m_backLeftLocation, m_backRightLocation
);
```

C++

```
// Locations of the wheels relative to the robot center.
frc::Translation2d m_frontLeftLocation{0.381_m, 0.381_m};
frc::Translation2d m_frontRightLocation{0.381_m, -0.381_m};
frc::Translation2d m_backLeftLocation{-0.381_m, 0.381_m};
frc::Translation2d m_backRightLocation{-0.381_m, -0.381_m};

// Creating my kinematics object using the wheel locations.
frc::MecanumDriveKinematics m_kinematics{
  m_frontLeftLocation, m_frontRightLocation, m_backLeftLocation,
  m_backRightLocation};
```

PYTHON

```
from wpimath.geometry import Translation2d
from wpimath.kinematics import MecanumDriveKinematics

# Locations of the wheels relative to the robot center.
frontLeftLocation = Translation2d(0.381, 0.381)
frontRightLocation = Translation2d(0.381, -0.381)
backLeftLocation = Translation2d(-0.381, 0.381)
backRightLocation = Translation2d(-0.381, -0.381)

# Creating my kinematics object using the wheel locations.
self.kinematics = MecanumDriveKinematics(
  frontLeftLocation, frontRightLocation, backLeftLocation, backRightLocation
)
```

27.6.2 Conversion des vitesses du châssis en vitesses de roue

The `toWheelSpeeds(ChassisSpeeds speeds)` (Java / Python) / `ToWheelSpeeds(ChassisSpeeds speeds)` (C++) method should be used to convert a `ChassisSpeeds` object to a `MecanumDriveWheelSpeeds` object. This is useful in situations where you have to convert a forward velocity, sideways velocity, and an angular velocity into individual wheel speeds.

JAVA

```
// Example chassis speeds: 1 meter per second forward, 3 meters
// per second to the left, and rotation at 1.5 radians per second
// counterclockwise.
ChassisSpeeds speeds = new ChassisSpeeds(1.0, 3.0, 1.5);

// Convert to wheel speeds
MecanumDriveWheelSpeeds wheelSpeeds = kinematics.toWheelSpeeds(speeds);

// Get the individual wheel speeds
double frontLeft = wheelSpeeds.frontLeftMetersPerSecond
double frontRight = wheelSpeeds.frontRightMetersPerSecond
```

(suite sur la page suivante)

(suite de la page précédente)

```
double backLeft = wheelSpeeds.rearLeftMetersPerSecond
double backRight = wheelSpeeds.rearRightMetersPerSecond
```

C++

```
// Example chassis speeds: 1 meter per second forward, 3 meters
// per second to the left, and rotation at 1.5 radians per second
// counterclockwise.
frc::ChassisSpeeds speeds{1_mps, 3_mps, 1.5_rad_per_s};

// Convert to wheel speeds. Here, we can use C++17's structured
// bindings feature to automatically split up the MecanumDriveWheelSpeeds
// struct into it's individual components
auto [fl, fr, bl, br] = kinematics.ToWheelSpeeds(speeds);
```

PYTHON

```
from wpimath.kinematics import ChassisSpeeds

# Example chassis speeds: 1 meter per second forward, 3 meters
# per second to the left, and rotation at 1.5 radians per second
# counterclockwise.
speeds = ChassisSpeeds(1.0, 3.0, 1.5)

# Convert to wheel speeds
frontLeft, frontRight, backLeft, backRight = self.kinematics.toWheelSpeeds(speeds)
```

Entraînement orienté terrain

Recall qu'un objet `ChassisSpeeds` peut être créé à partir d'un ensemble de vitesses orientées en fonction du terrain de jeu. Cette fonction peut être utilisée également pour obtenir les vitesses de roue à partir du même ensemble (de vitesses orientées en fonction du terrain de jeu).

JAVA

```
// The desired field relative speed here is 2 meters per second
// toward the opponent's alliance station wall, and 2 meters per
// second toward the left field boundary. The desired rotation
// is a quarter of a rotation per second counterclockwise. The current
// robot angle is 45 degrees.
ChassisSpeeds speeds = ChassisSpeeds.fromFieldRelativeSpeeds(
  2.0, 2.0, Math.PI / 2.0, Rotation2d.fromDegrees(45.0));

// Now use this in our kinematics
MecanumDriveWheelSpeeds wheelSpeeds = kinematics.toWheelSpeeds(speeds);
```

C++

```
// The desired field relative speed here is 2 meters per second
// toward the opponent's alliance station wall, and 2 meters per
// second toward the left field boundary. The desired rotation
// is a quarter of a rotation per second counterclockwise. The current
// robot angle is 45 degrees.
frc::ChassisSpeeds speeds = frc::ChassisSpeeds::FromFieldRelativeSpeeds(
  2_mps, 2_mps, units::radians_per_second_t(std::numbers::pi / 2.0), Rotation2d(45_
↪deg));

// Now use this in our kinematics
auto [fl, fr, bl, br] = kinematics.ToWheelSpeeds(speeds);
```

PYTHON

```
from wpimath.kinematics import ChassisSpeeds
import math
from wpimath.geometry import Rotation2d

# The desired field relative speed here is 2 meters per second
# toward the opponent's alliance station wall, and 2 meters per
# second toward the left field boundary. The desired rotation
# is a quarter of a rotation per second counterclockwise. The current
# robot angle is 45 degrees.
speeds = ChassisSpeeds.fromFieldRelativeSpeeds(
  2.0, 2.0, math.pi / 2.0, Rotation2d.fromDegrees(45.0))

# Now use this in our kinematics
wheelSpeeds = self.kinematics.toWheelSpeeds(speeds)
```

Utilisation de centres de rotation personnalisés

Parfois, la rotation autour d'un point spécifique peut être souhaitable pour que le robot puisse éviter un obstacle. Cette manoeuvre est également prise en charge par les classes WPILib. La même méthode `ToWheelSpeeds()` accepte un deuxième paramètre pour le centre de rotation (comme un `Translation2d`). Tout comme les emplacements des roues, le `Translation2d` qui correspond au centre de rotation doit être défini par rapport au centre du robot.

Note : Étant donné que tous les robots ont un châssis rigide, les vitesses `vx` et `vy` fournies par l'objet `ChassisSpeeds` s'appliqueront toujours à l'intégralité du robot. Cependant, `omega``de l'objet ``ChassisSpeeds` sera mesuré à partir du centre de rotation.

Par exemple, on peut définir le centre de rotation sur une certaine roue et si l'objet `ChassisSpeeds` fourni a un `vx` et un `vy` de zéro et un `omega` différent de zéro, le robot tournera autour de cette roue spécifique.

27.6.3 Conversion des vitesses de roue en vitesses de châssis

One can also use the kinematics object to convert a `MecanumDriveWheelSpeeds` object to a singular `ChassisSpeeds` object. The `toChassisSpeeds(MecanumDriveWheelSpeeds speeds)` (Java / Python) / `ToChassisSpeeds(MecanumDriveWheelSpeeds speeds)` (C++) method can be used to achieve this.

JAVA

```
// Example wheel speeds
var wheelSpeeds = new MecanumDriveWheelSpeeds(-17.67, 20.51, -13.44, 16.26);

// Convert to chassis speeds
ChassisSpeeds chassisSpeeds = kinematics.toChassisSpeeds(wheelSpeeds);

// Getting individual speeds
double forward = chassisSpeeds.vxMetersPerSecond;
double sideways = chassisSpeeds.vyMetersPerSecond;
double angular = chassisSpeeds.omegaRadiansPerSecond;
```

C++

```
// Example wheel speeds
frc::MecanumDriveWheelSpeeds wheelSpeeds{-17.67_mps, 20.51_mps, -13.44_mps, 16.26_mps}
↪;

// Convert to chassis speeds. Here, we can use C++17's structured bindings
// feature to automatically break up the ChassisSpeeds struct into its
// three components.
auto [forward, sideways, angular] = kinematics.ToChassisSpeeds(wheelSpeeds);
```

PYTHON

```
from wpimath.kinematics import MecanumDriveWheelSpeeds

# Example wheel speeds
wheelSpeeds = MecanumDriveWheelSpeeds(-17.67, 20.51, -13.44, 16.26)

# Convert to chassis speeds
chassisSpeeds = self.kinematics.toChassisSpeeds(wheelSpeeds)

# Getting individual speeds
forward = chassisSpeeds.vx
sideways = chassisSpeeds.vy
angular = chassisSpeeds.omega
```

27.7 Odométrie avec l'entraînement de type Mécanum

Un utilisateur peut utiliser les classes de cinématique d'entraînement de Mecanum afin d'effectuer l'" *Odométrie*. WPILib contient une classe `MecanumDriveOdometry` qui peut être utilisée pour suivre la position d'un robot avec entraînement Mécanum sur le terrain de jeu.

Note : Étant donné que cette méthode n'utilise que des encodeurs et un gyroscope, l'estimation de la position du robot sur le terrain de jeu dérivera avec le temps, d'autant plus que votre robot va entrer en contact avec des objets ou d'autres robots pendant le jeu. Cependant, l'odométrie est généralement très précise pendant la période autonome.

27.7.1 Création de l'objet odométrie

The `MecanumDriveOdometry` class constructor requires three mandatory arguments and one optional argument.

The mandatory arguments are :

- The kinematics object that represents your mecanum drive (as a `MecanumDriveKinematics` instance)
- The angle reported by your gyroscope (as a `Rotation2d`)
- The initial positions of the wheels (as `MecanumDriveWheelPositions`). In Java / Python, this must be constructed with each wheel position in meters. In C++, the *units library* must be used to represent your wheel positions.

The fourth optional argument is the starting pose of your robot on the field (as a `Pose2d`). By default, the robot will start at `x = 0, y = 0, theta = 0`.

Note : 0 degrees / radians represents the robot angle when the robot is facing directly toward your opponent's alliance station. As your robot turns to the left, your gyroscope angle should increase. The `Gyro` interface supplies `getRotation2d/GetRotation2d` that you can use for this purpose. See *Coordinate System* for more information about the coordinate system.

JAVA

```
// Locations of the wheels relative to the robot center.
Translation2d m_frontLeftLocation = new Translation2d(0.381, 0.381);
Translation2d m_frontRightLocation = new Translation2d(0.381, -0.381);
Translation2d m_backLeftLocation = new Translation2d(-0.381, 0.381);
Translation2d m_backRightLocation = new Translation2d(-0.381, -0.381);

// Creating my kinematics object using the wheel locations.
MecanumDriveKinematics m_kinematics = new MecanumDriveKinematics(
  m_frontLeftLocation, m_frontRightLocation, m_backLeftLocation, m_backRightLocation
);

// Creating my odometry object from the kinematics object and the initial wheel␣
↪positions.
// Here, our starting pose is 5 meters along the long end of the field and in the
// center of the field along the short end, facing the opposing alliance wall.
```

(suite sur la page suivante)

(suite de la page précédente)

```
MecanumDriveOdometry m_odometry = new MecanumDriveOdometry(
  m_kinematics,
  m_gyro.getRotation2d(),
  new MecanumDriveWheelPositions(
    m_frontLeftEncoder.getDistance(), m_frontRightEncoder.getDistance(),
    m_backLeftEncoder.getDistance(), m_backRightEncoder.getDistance()
  ),
  new Pose2d(5.0, 13.5, new Rotation2d())
);
```

C++

```
// Locations of the wheels relative to the robot center.
frc::Translation2d m_frontLeftLocation{0.381_m, 0.381_m};
frc::Translation2d m_frontRightLocation{0.381_m, -0.381_m};
frc::Translation2d m_backLeftLocation{-0.381_m, 0.381_m};
frc::Translation2d m_backRightLocation{-0.381_m, -0.381_m};

// Creating my kinematics object using the wheel locations.
frc::MecanumDriveKinematics m_kinematics{
  m_frontLeftLocation, m_frontRightLocation,
  m_backLeftLocation, m_backRightLocation
};

// Creating my odometry object from the kinematics object. Here,
// our starting pose is 5 meters along the long end of the field and in the
// center of the field along the short end, facing forward.
frc::MecanumDriveOdometry m_odometry{
  m_kinematics,
  m_gyro.GetRotation2d(),
  frc::MecanumDriveWheelPositions{
    units::meter_t{m_frontLeftEncoder.GetDistance()},
    units::meter_t{m_frontRightEncoder.GetDistance()},
    units::meter_t{m_backLeftEncoder.GetDistance()},
    units::meter_t{m_backRightEncoder.GetDistance()}
  },
  frc::Pose2d{5_m, 13.5_m, 0_rad}};
```

PYTHON

```
from wpimath.geometry import Translation2d
from wpimath.kinematics import MecanumDriveKinematics
from wpimath.kinematics import MecanumDriveOdometry
from wpimath.kinematics import MecanumDriveWheelPositions
from wpimath.geometry import Pose2d
from wpimath.geometry import Rotation2d

# Locations of the wheels relative to the robot center.
frontLeftLocation = Translation2d(0.381, 0.381)
frontRightLocation = Translation2d(0.381, -0.381)
backLeftLocation = Translation2d(-0.381, 0.381)
backRightLocation = Translation2d(-0.381, -0.381)
```

(suite sur la page suivante)

(suite de la page précédente)

```
# Creating my kinematics object using the wheel locations.
self.kinematics = MecanumDriveKinematics(
  frontLeftLocation, frontRightLocation, backLeftLocation, backRightLocation
)

# Creating my odometry object from the kinematics object and the initial wheel␣
↪positions.
# Here, our starting pose is 5 meters along the long end of the field and in the
# center of the field along the short end, facing the opposing alliance wall.
self.odometry = MecanumDriveOdometry(
  self.kinematics,
  self.gyro.getRotation2d(),
  MecanumDriveWheelPositions(
    self.frontLeftEncoder.getDistance(), self.frontRightEncoder.getDistance(),
    self.backLeftEncoder.getDistance(), self.backRightEncoder.getDistance()
  ),
  Pose2d(5.0, 13.5, Rotation2d())
)
```

27.7.2 Mise à jour de la pose du robot

The `update` method of the odometry class updates the robot position on the field. The update method takes in the gyro angle of the robot, along with a `MecanumDriveWheelPositions` object representing the position of each of the 4 wheels on the robot. This `update` method must be called periodically, preferably in the `periodic()` method of a *Subsystem*. The `update` method returns the new updated pose of the robot.

JAVA

```
@Override
public void periodic() {
  // Get my wheel positions
  var wheelPositions = new MecanumDriveWheelPositions(
    m_frontLeftEncoder.getDistance(), m_frontRightEncoder.getDistance(),
    m_backLeftEncoder.getDistance(), m_backRightEncoder.getDistance());

  // Get the rotation of the robot from the gyro.
  var gyroAngle = m_gyro.getRotation2d();

  // Update the pose
  m_pose = m_odometry.update(gyroAngle, wheelPositions);
}
```

C++

```
void Periodic() override {
  // Get my wheel positions
  frc::MecanumDriveWheelPositions wheelPositions{
    units::meter_t{m_frontLeftEncoder.GetDistance()},
    units::meter_t{m_frontRightEncoder.GetDistance()},
    units::meter_t{m_backLeftEncoder.GetDistance()},
    units::meter_t{m_backRightEncoder.GetDistance()}};

  // Get the rotation of the robot from the gyro.
  frc::Rotation2d gyroAngle = m_gyro.GetRotation2d();

  // Update the pose
  m_pose = m_odometry.Update(gyroAngle, wheelPositions);
}
```

PYTHON

```
from wpimath.kinematics import MecanumDriveWheelPositions

def periodic(self):
  # Get my wheel positions
  wheelPositions = MecanumDriveWheelPositions(
    self.frontLeftEncoder.getDistance(), self.frontRightEncoder.getDistance(),
    self.backLeftEncoder.getDistance(), self.backRightEncoder.getDistance())

  # Get the rotation of the robot from the gyro.
  gyroAngle = gyro.getRotation2d()

  # Update the pose
  self.pose = odometry.update(gyroAngle, wheelPositions)
```

27.7.3 Réinitialisation de la pose de robot

The robot pose can be reset via the `resetPosition` method. This method accepts three arguments : the current gyro angle, the current wheel positions, and the new field-relative pose.

Important : If at any time, you decide to reset your gyroscope or encoders, the `resetPosition` method MUST be called with the new gyro angle and wheel positions.

Note : A full example of a mecanum drive robot with odometry is available here : C++ / Java / Python

---

In addition, the `GetPose` (C++) / `getPoseMeters` (Java / Python) methods can be used to retrieve the current robot pose without an update.

28 NetworkTables

This section outlines the details of using the NetworkTables (v4) API to communicate information across the robot network.

Important : The code examples in this section are not intended for the user to copy-paste. Ensure that the following documentation is thoroughly read and the API (Java, C++, Python) is consulted when necessary.

28.1 Que sont les NetworkTables ?

NetworkTables is an implementation of a publish-subscribe messaging system. Values are published to named « topics » either on the robot, driver station, or potentially an attached coprocessor, and the values are automatically distributed to all subscribers to the topic. For example, a driver station laptop might receive camera images over the network, perform some vision processing algorithm, and come up with some values to sent back to the robot. The values might be an X, Y, and Distance. By writing these results to NetworkTables topics called « X », « Y », and « Distance » they can be read by the robot shortly after being written. Then the robot can act upon them. Similarly, the robot program can write sensor values to topics and those can be read and plotted in real time on a dashboard application.

NetworkTables can be used by programs on the robot in Java, C++, or LabVIEW, and is built into each version of WPILib.

Note : NetworkTables has changed substantially in 2023. For more information on migrating pre-2023 code to use the new features, see *Migrating from NetworkTables 3.0 to NetworkTables 4.0*.

28.1.1 NetworkTables Concepts

First, let's define some terms :

- **Topic** : a named data channel. Topics have a fixed data type (for the lifetime of the topic) and *mutable* properties.
- **Publisher** : defines the topic and creates and sends timestamped data values.
- **Subscriber** : receives timestamped data value updates to one or more topics.
- **Entry** : a combined publisher and subscriber. The subscriber is always active, but the publisher is not created until a publish operation is performed (e.g. a value is « set », aka published, on the entry). This may be more convenient than maintaining a separate publisher and subscriber.
- **Property** : named information (metadata) about a topic stored and updated separately from the topic's data. A topic may have any number of properties. A property's value can be any data type that can be represented in JSON.

NetworkTables supports a range of data types, including *boolean*, numeric, string, and arrays of those types. Supported numeric data types are single or double precision *floating point*, or 64-bit integer. There is also the option of storing raw data (an array of bytes), which can be used for representing binary encoded structured data. Types are represented as strings for efficiency reasons. There is also an *enumeration* for the most common types in the NetworkTables API.

Topics are created when the first publisher announces the topic and are removed when the last publisher stops publishing. It's possible to subscribe to a topic that has not yet been created/published.

Topics have properties. Properties are initially set by the first publisher, but may be changed at any time. Similarly to values, property changes to a topic are propagated to all subscribers to that topic. Properties are structured data (JSON), but at the top level are simply a key/value store (a JSON map). Some properties have defined behavior, but arbitrary ones can be set by the application.

Publishers specify the topic's data type ; while there can be multiple publishers to a single topic, they must all be publishing the same data type. This is enforced by the NetworkTables server (the first publisher « wins »). Typically single-topic subscribers also specify what data type they're expecting to receive on a topic and thus won't receive value updates of other types.

28.1.2 Retained and Persistent Topics

While by default topics are *transitory* and disappear after the last publisher stops publishing, topics can be marked as *retained* (via setting the « retained » property to true) to prevent them from disappearing. For retained topics, the server acts as an implicit publisher of the last value, and will keep doing so as long as the server is running. This is primarily useful for configuration values ; e.g. an autonomous mode selection published by a dashboard should set the topic as retained so its value is preserved in case the dashboard disconnects.

Additionally, topics can be marked as *persistent* via setting the « persistent » property to true. These operate similarly to retained topics, but in addition, persistent topic values are automatically saved to a file on the server and when the server starts up again, the topic is created and its last value is published by the server.

28.1.3 Value Propagation

The server keeps a copy of the last published value for every topic. When a subscriber initially subscribes to a topic, the server sends the last published value. After that initial value, new value updates are communicated to subscribers each time the publisher sends a new value.

NetworkTables is a client/server system ; clients do not talk directly to each other, but rather communicate via the server. Typically, the robot program is the server, and other pieces of software on other computers (e.g. the driver station or a coprocessor) are clients that connect to it. Thus, when a coprocessor (client) publishes a value, the value is sent first from the coprocessor (client) to the robot program (server), and then the robot program distributes that value to any subscribers (e.g. the robot program local program, or other clients such as dashboards).

The server does not send topic changes or value updates to clients that have not subscribed to the topic.

By default, NetworkTables sends value updates periodically, batching the data to help limit the number of small packets being sent over the network. Also, by default, only the most recent value is transmitted ; any intermediate value changes made between network transmissions are discarded. This behavior can be changed via publish/subscribe options–publishers and subscribers can indicate that all value updates should be preserved and communicated via the « send all » option. In addition, it is possible to force NetworkTables to « flush » all current updates to the network ; this is useful for minimizing latency.

28.1.4 Timestamps

All NetworkTable value updates are timestamped at the time they are published. Timestamps in NetworkTables are measured in integer microseconds.

NetworkTables automatically synchronizes time between the server and clients. Each client maintains an offset between the client local time and the server time, so when a client publishes a value, it stores a timestamp in local time and calculates the equivalent server timestamp. The server timestamp is what is communicated over the network to any subscribers. This makes it possible e.g. for a robot program to get a reasonable estimation of the time when a value was published on a coprocessor relative to the current time.

Because of this, two timestamps are visible through the API : a server timestamp indicating the time (estimated) on the server, and a local timestamp indicating the time on the client. When the RoboRIO is the NetworkTables server, the server timestamp is the same as the FPGA timestamp returned by `Timer.getFPGATimestamp()` (except the units are different : NetworkTables uses microseconds, while `getFPGATimestamp()` returns seconds).

28.1.5 NetworkTables Organization

Data is organized in NetworkTables in a hierarchy much like a filesystem's folders and files. There can be multiple subtables (folders) and topics (files) that may be nested in whatever way fits the data organization desired. At the top level (`NetworkTableInstance` : Java, C++, `Python`), topic names are handled similar to absolute paths in a filesystem : subtables are represented as a long topic name with slashes (« / ») separating the nested subtable and value names. A `NetworkTable` (Java, C++, `Python`) object represents a single subtable (folder), so topic names are relative to the NetworkTable's base path : e.g. for a root table called « SmartDashboard » with a topic named « xValue », the same topic can be accessed via `NetworkTableInstance` as a topic named « /SmartDashboard/xValue ». However, unlike a

filesystem, subtables don't really exist in the same way folders do, as there is no way to represent an empty subtable on the network–a subtable « appears » only as long as there are topics published within it.

OutlineViewer is a utility for exploring the values stored in NetworkTables, and can show either a flat view (topics with absolute paths) or a nested view (subtables and topics).

There are some default tables that are created automatically when a robot program starts up :

Nom de table	Utilité
/SmartDashboard	Utilisé pour stocker les valeurs écrites sur le SmartDashboard ou le Shuffleboard à l'aide de l'ensemble de méthodes `SmartDashboard.put()`.
/LiveWindow	Utilisé pour stocker les valeurs du mode Test (Test sur le Driver Station). Il s'agit généralement de sous-systèmes et des capteurs et actionneurs associés.
/FMSInfo	Informations sur le match en cours d'exécution provenant de Driver Station et du Field Management System

28.1.6 NetworkTables API Variants

There are two major variants of the NetworkTables API. The object-oriented API (C++ and Java) is recommended for robot code and general team use, and provides classes that help ensure correct use of the API. For advanced use cases such as writing object-oriented wrappers for other programming languages, there's also a C/C++ handle-based API.

28.1.7 Lifetime Management

Publishers, subscribers, and entries only exist as long as the objects exist.

In Java, a common bug is to create a subscriber or publisher and not properly release it by calling `close()`, as this will result in the object lingering around for an unknown period of time and not releasing resources properly. This is less common of an issue in robot programs, as long as the publisher or subscriber object is stored in an instance variable that persists for the life of the program.

In C++, publishers, subscribers, and entries are *RAII*, which means they are automatically destroyed when they go out of scope. `NetworkTableInstance` is an exception to this; it is designed to be explicitly destroyed, so it's not necessary to maintain a global instance of it.

Python is similar to Java, except that subscribers or publishers are released when they are garbage collected.

28.2 NetworkTables Tables and Topics

28.2.1 Using the NetworkTable Class

The `NetworkTable` (Java, C++, `Python`) class is an API abstraction that represents a single « folder » (or « table ») of topics as described in *NetworkTables Organization*. The NetworkTable class stores the base path to the table and provides functions to get topics within the table, automatically prepending the table path.

28.2.2 Getting a Topic

A `Topic` (Java, C++, `Python`) object (or `NT_Topic` handle) represents a *topic*. This has a 1 :1 correspondence with the topic's name, and will not change as long as the instance exists. Unlike publishers and subscribers, it is not necessary to store a Topic object.

Having a Topic object or handle does not mean the topic exists or is of the correct type. For convenience when creating publishers and subscribers, there are type-specific Topic classes (e.g. `BooleanTopic` : Java, C++, `Python`), but there is no check at the Topic level to ensure that the topic's type actually matches. The preferred method to get a type-specific topic to call the appropriate type-specific getter, but it's also possible to directly convert a generic Topic into a type-specific Topic class. Note : the handle-based API does not have a concept of type-specific classes.

Java

```
NetworkTableInstance inst = NetworkTableInstance.getDefault();
NetworkTable table = inst.getTable("datatable");

// get a topic from a NetworkTableInstance
// the topic name in this case is the full name
DoubleTopic dblTopic = inst.getDoubleTopic("/datatable/X");

// get a topic from a NetworkTable
// the topic name in this case is the name within the table;
// this line and the one above reference the same topic
DoubleTopic dblTopic = table.getDoubleTopic("X");

// get a type-specific topic from a generic Topic
Topic genericTopic = inst.getTopic("/datatable/X");
DoubleTopic dblTopic = new DoubleTopic(genericTopic);
```

C++

```
nt::NetworkTableInstance inst = nt::NetworkTableInstance::GetDefault();
std::shared_ptr<nt::NetworkTable> table = inst.GetTable("datatable");

// get a topic from a NetworkTableInstance
// the topic name in this case is the full name
nt::DoubleTopic dblTopic = inst.GetDoubleTopic("/datatable/X");

// get a topic from a NetworkTable
// the topic name in this case is the name within the table;
// this line and the one above reference the same topic
nt::DoubleTopic dblTopic = table->GetDoubleTopic("X");

// get a type-specific topic from a generic Topic
nt::Topic genericTopic = inst.GetTopic("/datatable/X");
nt::DoubleTopic dblTopic{genericTopic};
```

C++ (Handle-based)

```
NT_Instance inst = nt::GetDefaultInstance();

// get a topic from a NetworkTableInstance
NT_Topic topic = nt::GetTopic(inst, "/datatable/X");
```

C

```
NT_Instance inst = NT_GetDefaultInstance();

// get a topic from a NetworkTableInstance
NT_Topic topic = NT_GetTopic(inst, "/datatable/X");
```

Python

```
import ntcore

inst = ntcore.NetworkTableInstance.getDefault()
table = inst.getTable("datatable")

# get a topic from a NetworkTableInstance
# the topic name in this case is the full name
dblTopic = inst.getDoubleTopic("/datatable/X")

# get a topic from a NetworkTable
# the topic name in this case is the name within the table;
# this line and the one above reference the same topic
dblTopic = table.getDoubleTopic("X")

# get a type-specific topic from a generic Topic
```

(suite sur la page suivante)

(suite de la page précédente)

```
genericTopic = inst.getTopic("/datatable/X")
dblTopic = ntcore.DoubleTopic(genericTopic)
```

28.3 Publishing and Subscribing to a Topic

28.3.1 Publishing to a Topic

In order to create a *topic* and publish values to it, it's necessary to create a *publisher*.

NetworkTable publishers are represented as type-specific Publisher objects (e.g. `BooleanPublisher` : Java, C++, Python). Publishers are only active as long as the Publisher object exists. Typically you want to keep publishing longer than the local scope of a function, so it's necessary to store the Publisher object somewhere longer term, e.g. in an instance variable. In Java, the `close()` method needs be called to stop publishing ; in C++ this is handled by the destructor. C++ publishers are moveable and non-copyable. In Python the `close()` method should be called to stop publishing, but it will also be closed when the object is garbage collected.

In the handle-based APIs, there is only the non-type-specific `NT_Publisher` handle ; the user is responsible for keeping track of the type of the publisher and using the correct type-specific set methods.

Publishing values is done via a `set()` operation. By default, this operation uses the current time, but a timestamp may optionally be specified. Specifying a timestamp can be useful when multiple values should have the same update timestamp. The timestamp units are integer microseconds (see example code for how to get a current timestamp that is consistent with the library).

Java

```
public class Example {
  // the publisher is an instance variable so its lifetime matches that of␣
↪the class
  final DoublePublisher dblPub;

  public Example(DoubleTopic dblTopic) {
    // start publishing; the return value must be retained (in this case, via
    // an instance variable)
    dblPub = dblTopic.publish();

    // publish options may be specified using PubSubOption
    dblPub = dblTopic.publish(PubSubOption.keepDuplicates(true));

    // publishEx provides additional options such as setting initial
    // properties and using a custom type string. Using a custom type string␣
↪for
    // types other than raw and string is not recommended. The properties␣
↪string
    // must be a JSON map.
    dblPub = dblTopic.publishEx("double", "{\"myprop\": 5}");
```

(suite sur la page suivante)

(suite de la page précédente)

```
  }

  public void periodic() {
    // publish a default value
    dblPub.setDefault(0.0);

    // publish a value with current timestamp
    dblPub.set(1.0);
    dblPub.set(2.0, 0);  // 0 = use current time

    // publish a value with a specific timestamp; NetworkTablesJNI.now() can
    // be used to get the current time. On the roboRIO, this is the same as
    // the FPGA timestamp (e.g. RobotController.getFPGATime())
    long time = NetworkTablesJNI.now();
    dblPub.set(3.0, time);

    // publishers also implement the appropriate Consumer functional␣
↪interface;
    // this example assumes void myFunc(DoubleConsumer func) exists
    myFunc(dblPub);
  }

  // often not required in robot code, unless this class doesn't exist for
  // the lifetime of the entire robot program, in which case close() needs␣
↪to be
  // called to stop publishing
  public void close() {
    // stop publishing
    dblPub.close();
  }
}
```

C++

```
class Example {
  // the publisher is an instance variable so its lifetime matches that of␣
↪the class
  // publishing is automatically stopped when dblPub is destroyed by the␣
↪class destructor
  nt::DoublePublisher dblPub;

 public:
  explicit Example(nt::DoubleTopic dblTopic) {
    // start publishing; the return value must be retained (in this case, via
    // an instance variable)
    dblPub = dblTopic.Publish();

    // publish options may be specified using PubSubOptions
    dblPub = dblTopic.Publish({.keepDuplicates = true});

    // PublishEx provides additional options such as setting initial
    // properties and using a custom type string. Using a custom type string␣
↪for
    // types other than raw and string is not recommended. The properties␣
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪must
    // be a JSON map.
    dblPub = dblTopic.PublishEx("double", {{"myprop", 5}});
  }

  void Periodic() {
    // publish a default value
    dblPub.SetDefault(0.0);

    // publish a value with current timestamp
    dblPub.Set(1.0);
    dblPub.Set(2.0, 0);  // 0 = use current time

    // publish a value with a specific timestamp; nt::Now() can
    // be used to get the current time.
    int64_t time = nt::Now();
    dblPub.Set(3.0, time);
  }
};
```

C++ (Handle-based)

```
class Example {
  // the publisher is an instance variable, but since it's a handle, it's
  // not automatically released, so we need a destructor
  NT_Publisher dblPub;

 public:
  explicit Example(NT_Topic dblTopic) {
    // start publishing. It's recommended that the type string be standard
    // for all types except string and raw.
    dblPub = nt::Publish(dblTopic, NT_DOUBLE, "double");

    // publish options may be specified using PubSubOptions
    dblPub = nt::Publish(dblTopic, NT_DOUBLE, "double",
        {.keepDuplicates = true});

    // PublishEx allows setting initial properties. The
    // properties must be a JSON map.
    dblPub = nt::PublishEx(dblTopic, NT_DOUBLE, "double", {{"myprop", 5}});
  }

  void Periodic() {
    // publish a default value
    nt::SetDefaultDouble(dblPub, 0.0);

    // publish a value with current timestamp
    nt::SetDouble(dblPub, 1.0);
    nt::SetDouble(dblPub, 2.0, 0);  // 0 = use current time

    // publish a value with a specific timestamp; nt::Now() can
    // be used to get the current time.
    int64_t time = nt::Now();
    nt::SetDouble(dblPub, 3.0, time);
```

(suite sur la page suivante)

(suite de la page précédente)

```
  }

  ~Example() {
    // stop publishing
    nt::Unpublish(dblPub);
  }
};
```

C

```
// This code assumes that a NT_Topic dblTopic variable already exists

// start publishing. It's recommended that the type string be standard
// for all types except string and raw.
NT_Publisher dblPub = NT_Publish(dblTopic, NT_DOUBLE, "double", NULL, 0);

// publish options may be specified
struct NT_PubSubOptions options;
memset(&options, 0, sizeof(options));
options.structSize = sizeof(options);
options.keepDuplicates = 1;  // true
NT_Publisher dblPub = NT_Publish(dblTopic, NT_DOUBLE, "double", &options);

// PublishEx allows setting initial properties. The properties string must
// be a JSON map.
NT_Publisher dblPub =
    NT_PublishEx(dblTopic, NT_DOUBLE, "double", "{\"myprop\", 5}", NULL, 0);

// publish a default value
NT_SetDefaultDouble(dblPub, 0.0);

// publish a value with current timestamp
NT_SetDouble(dblPub, 1.0);
NT_SetDouble(dblPub, 2.0, 0);  // 0 = use current time

// publish a value with a specific timestamp; NT_Now() can
// be used to get the current time.
int64_t time = NT_Now();
NT_SetDouble(dblPub, 3.0, time);

// stop publishing
NT_Unpublish(dblPub);
```

Python

```
class Example:
    def __init__(self, dblTopic: ntcore.DoubleTopic):

        # start publishing; the return value must be retained (in this case,␣
↪via
        # an instance variable)
        self.dblPub = dblTopic.publish()

        # publish options may be specified using PubSubOption
        self.dblPub = dblTopic.publish(ntcore.
↪PubSubOptions(keepDuplicates=True))

        # publishEx provides additional options such as setting initial
        # properties and using a custom type string. Using a custom type␣
↪string for
        # types other than raw and string is not recommended. The properties␣
↪string
        # must be a JSON map.
        self.dblPub = dblTopic.publishEx("double", '{"myprop": 5}')

    def periodic(self):
        # publish a default value
        self.dblPub.setDefault(0.0)

        # publish a value with current timestamp
        self.dblPub.set(1.0)
        self.dblPub.set(2.0, 0)  # 0 = use current time

        # publish a value with a specific timestamp with microsecond␣
↪resolution.
        # On the roboRIO, this is the same as the FPGA timestamp (e.g.
        # RobotController.getFPGATime())
        self.dblPub.set(3.0, ntcore._now())

    # often not required in robot code, unless this class doesn't exist for
    # the lifetime of the entire robot program, in which case close() needs␣
↪to be
    # called to stop publishing
    def close(self):
        # stop publishing
        self.dblPub.close()
```

28.3.2 Subscribing to a Topic

A *subscriber* receives value updates made to a topic. Similar to publishers, NetworkTable subscribers are represented as type-specific Subscriber classes (e.g. `BooleanSubscriber` : Java, C++, Python) that must be stored somewhere to continue subscribing.

Subscribers have a range of different ways to read received values. It's possible to just read the most recent value using `get()`, read the most recent value, along with its timestamp, using `getAtomic()`, or get an array of all value changes since the last call using `readQueue()` or `readQueueValues()`.

Java

```
public class Example {
  // the subscriber is an instance variable so its lifetime matches that of␣
↪the class
  final DoubleSubscriber dblSub;

  public Example(DoubleTopic dblTopic) {
    // start subscribing; the return value must be retained.
    // the parameter is the default value if no value is available when␣
↪get() is called
    dblSub = dblTopic.subscribe(0.0);

    // subscribe options may be specified using PubSubOption
    dblSub =
        dblTopic.subscribe(0.0, PubSubOption.keepDuplicates(true),␣
↪PubSubOption.pollStorage(10));

    // subscribeEx provides the options of using a custom type string.
    // Using a custom type string for types other than raw and string is not␣
↪recommended.
    dblSub = dblTopic.subscribeEx("double", 0.0);
  }

  public void periodic() {
    // simple get of most recent value; if no value has been published,
    // returns the default value passed to the subscribe() function
    double val = dblSub.get();

    // get the most recent value; if no value has been published, returns
    // the passed-in default value
    double val = dblSub.get(-1.0);

    // subscribers also implement the appropriate Supplier interface, e.g.␣
↪DoubleSupplier
    double val = dblSub.getAsDouble();

    // get the most recent value, along with its timestamp
    TimestampedDouble tsVal = dblSub.getAtomic();

    // read all value changes since the last call to readQueue/
↪readQueueValues
    // readQueue() returns timestamps; readQueueValues() does not.
    TimestampedDouble[] tsUpdates = dblSub.readQueue();
    double[] valUpdates = dblSub.readQueueValues();
  }

  // often not required in robot code, unless this class doesn't exist for
  // the lifetime of the entire robot program, in which case close() needs␣
↪to be
  // called to stop subscribing
  public void close() {
    // stop subscribing
    dblSub.close();
  }
}
```

C++

```
class Example {
  // the subscriber is an instance variable so its lifetime matches that of the class
  // subscribing is automatically stopped when dblSub is destroyed by the class destructor
  nt::DoubleSubscriber dblSub;

 public:
  explicit Example(nt::DoubleTopic dblTopic) {
    // start subscribing; the return value must be retained.
    // the parameter is the default value if no value is available when get() is called
    dblSub = dblTopic.Subscribe(0.0);

    // subscribe options may be specified using PubSubOptions
    dblSub =
        dblTopic.subscribe(0.0,
        {.pollStorage = 10, .keepDuplicates = true});

    // SubscribeEx provides the options of using a custom type string.
    // Using a custom type string for types other than raw and string is not recommended.
    dblSub = dblTopic.SubscribeEx("double", 0.0);
  }

  void Periodic() {
    // simple get of most recent value; if no value has been published,
    // returns the default value passed to the Subscribe() function
    double val = dblSub.Get();

    // get the most recent value; if no value has been published, returns
    // the passed-in default value
    double val = dblSub.Get(-1.0);

    // get the most recent value, along with its timestamp
    nt::TimestampedDouble tsVal = dblSub.GetAtomic();

    // read all value changes since the last call to ReadQueue/ReadQueueValues
    // ReadQueue() returns timestamps; ReadQueueValues() does not.
    std::vector<nt::TimestampedDouble> tsUpdates = dblSub.ReadQueue();
    std::vector<double> valUpdates = dblSub.ReadQueueValues();
  }
};
```

C++ (Handle-based)

```
class Example {
  // the subscriber is an instance variable, but since it's a handle, it's
  // not automatically released, so we need a destructor
  NT_Subscriber dblSub;

 public:
  explicit Example(NT_Topic dblTopic) {
    // start subscribing
    // Using a custom type string for types other than raw and string is not␣
↪recommended.
    dblSub = nt::Subscribe(dblTopic, NT_DOUBLE, "double");

    // subscribe options may be specified using PubSubOptions
    dblSub =
        nt::Subscribe(dblTopic, NT_DOUBLE, "double",
        {.pollStorage = 10, .keepDuplicates = true});
  }

  void Periodic() {
    // get the most recent value; if no value has been published, returns
    // the passed-in default value
    double val = nt::GetDouble(dblSub, 0.0);

    // get the most recent value, along with its timestamp
    nt::TimestampedDouble tsVal = nt::GetAtomic(dblSub, 0.0);

    // read all value changes since the last call to ReadQueue/
↪ReadQueueValues
    // ReadQueue() returns timestamps; ReadQueueValues() does not.
    std::vector<nt::TimestampedDouble> tsUpdates =␣
↪nt::ReadQueueDouble(dblSub);
    std::vector<double> valUpdates = nt::ReadQueueValuesDouble(dblSub);
  }

  ~Example() {
    // stop subscribing
    nt::Unsubscribe(dblSub);
  }
```

C

```
// This code assumes that a NT_Topic dblTopic variable already exists

// start subscribing
// Using a custom type string for types other than raw and string is not␣
↪recommended.
NT_Subscriber dblSub = NT_Subscribe(dblTopic, NT_DOUBLE, "double", NULL, 0);

// subscribe options may be specified using NT_PubSubOptions
struct NT_PubSubOptions options;
memset(&options, 0, sizeof(options));
options.structSize = sizeof(options);
```

(suite sur la page suivante)

(suite de la page précédente)

```
options.keepDuplicates = 1;  // true
options.pollStorage = 10;
NT_Subscriber dblSub = NT_Subscribe(dblTopic, NT_DOUBLE, "double", &options);

// get the most recent value; if no value has been published, returns
// the passed-in default value
double val = NT_GetDouble(dblSub, 0.0);

// get the most recent value, along with its timestamp
struct NT_TimestampedDouble tsVal;
NT_GetAtomic(dblSub, 0.0, &tsVal);
NT_DisposeTimestamped(&tsVal);

// read all value changes since the last call to ReadQueue/ReadQueueValues
// ReadQueue() returns timestamps; ReadQueueValues() does not.
size_t tsUpdatesLen;
struct NT_TimestampedDouble* tsUpdates = NT_ReadQueueDouble(dblSub, &
↪tsUpdatesLen);
NT_FreeQueueDouble(tsUpdates, tsUpdatesLen);

size_t valUpdatesLen;
double* valUpdates = NT_ReadQueueValuesDouble(dblSub, &valUpdatesLen);
NT_FreeDoubleArray(valUpdates, valUpdatesLen);

// stop subscribing
NT_Unsubscribe(dblSub);
```

Python

```
class Example:
    def __init__(self, dblTopic: ntcore.DoubleTopic):

        # start subscribing; the return value must be retained.
        # the parameter is the default value if no value is available when␣
↪get() is called
        self.dblSub = dblTopic.subscribe(0.0)

        # subscribe options may be specified using PubSubOption
        self.dblSub = dblTopic.subscribe(
            0.0, ntcore.PubSubOptions(keepDuplicates=True, pollStorage=10)
        )

        # subscribeEx provides the options of using a custom type string.
        # Using a custom type string for types other than raw and string is␣
↪not recommended.
        dblSub = dblTopic.subscribeEx("double", 0.0)

    def periodic(self):
        # simple get of most recent value; if no value has been published,
        # returns the default value passed to the subscribe() function
        val = self.dblSub.get()

        # get the most recent value; if no value has been published, returns
        # the passed-in default value
```

(suite sur la page suivante)

(suite de la page précédente)

```
        val = self.dblSub.get(-1.0)

        # get the most recent value, along with its timestamp
        tsVal = self.dblSub.getAtomic()

        # read all value changes since the last call to readQueue
        # readQueue() returns timestamps
        tsUpdates = self.dblSub.readQueue()

    # often not required in robot code, unless this class doesn't exist for
    # the lifetime of the entire robot program, in which case close() needs␣
↪to be
    # called to stop subscribing
    def close(self):
        # stop subscribing
        self.dblSub.close()
```

28.3.3 Using Entry to Both Subscribe and Publish

An *entry* is a combined publisher and subscriber. The subscriber is always active, but the publisher is not created until a publish operation is performed (e.g. a value is « set », aka published, on the entry). This may be more convenient than maintaining a separate publisher and subscriber. Similar to publishers and subscribers, NetworkTable entries are represented as type-specific Entry classes (e.g. `BooleanEntry` : Java, C++, Python) that must be retained to continue subscribing (and publishing).

Java

```
public class Example {
  // the entry is an instance variable so its lifetime matches that of the␣
↪class
  final DoubleEntry dblEntry;

  public Example(DoubleTopic dblTopic) {
    // start subscribing; the return value must be retained.
    // the parameter is the default value if no value is available when␣
↪get() is called
    dblEntry = dblTopic.getEntry(0.0);

    // publish and subscribe options may be specified using PubSubOption
    dblEntry =
        dblTopic.getEntry(0.0, PubSubOption.keepDuplicates(true),␣
↪PubSubOption.pollStorage(10));

    // getEntryEx provides the options of using a custom type string.
    // Using a custom type string for types other than raw and string is not␣
↪recommended.
    dblEntry = dblTopic.getEntryEx("double", 0.0);
  }

  public void periodic() {
    // entries support all the same methods as subscribers:
```

(suite sur la page suivante)

(suite de la page précédente)

```
    double val = dblEntry.get();
    double val = dblEntry.get(-1.0);
    double val = dblEntry.getAsDouble();
    TimestampedDouble tsVal = dblEntry.getAtomic();
    TimestampedDouble[] tsUpdates = dblEntry.readQueue();
    double[] valUpdates = dblEntry.readQueueValues();

    // entries also support all the same methods as publishers; the first␣
↪time
    // one of these is called, an internal publisher is automatically created
    dblEntry.setDefault(0.0);
    dblEntry.set(1.0);
    dblEntry.set(2.0, 0);  // 0 = use current time
    long time = NetworkTablesJNI.now();
    dblEntry.set(3.0, time);
    myFunc(dblEntry);
  }

  public void unpublish() {
    // you can stop publishing while keeping the subscriber alive
    dblEntry.unpublish();
  }

  // often not required in robot code, unless this class doesn't exist for
  // the lifetime of the entire robot program, in which case close() needs␣
↪to be
  // called to stop subscribing
  public void close() {
    // stop subscribing/publishing
    dblEntry.close();
  }
}
```

C++

```
class Example {
  // the entry is an instance variable so its lifetime matches that of the␣
↪class
  // subscribing/publishing is automatically stopped when dblEntry is␣
↪destroyed by
  // the class destructor
  nt::DoubleEntry dblEntry;

 public:
  explicit Example(nt::DoubleTopic dblTopic) {
    // start subscribing; the return value must be retained.
    // the parameter is the default value if no value is available when␣
↪get() is called
    dblEntry = dblTopic.GetEntry(0.0);

    // publish and subscribe options may be specified using PubSubOptions
    dblEntry =
        dblTopic.GetEntry(0.0,
        {.pollStorage = 10, .keepDuplicates = true});
```

(suite sur la page suivante)

(suite de la page précédente)

```
    // GetEntryEx provides the options of using a custom type string.
    // Using a custom type string for types other than raw and string is not␣
↪recommended.
    dblEntry = dblTopic.GetEntryEx("double", 0.0);
  }

  void Periodic() {
    // entries support all the same methods as subscribers:
    double val = dblEntry.Get();
    double val = dblEntry.Get(-1.0);
    nt::TimestampedDouble tsVal = dblEntry.GetAtomic();
    std::vector<nt::TimestampedDouble> tsUpdates = dblEntry.ReadQueue();
    std::vector<double> valUpdates = dblEntry.ReadQueueValues();

    // entries also support all the same methods as publishers; the first␣
↪time
    // one of these is called, an internal publisher is automatically created
    dblEntry.SetDefault(0.0);
    dblEntry.Set(1.0);
    dblEntry.Set(2.0, 0);  // 0 = use current time
    int64_t time = nt::Now();
    dblEntry.Set(3.0, time);
  }

  void Unpublish() {
    // you can stop publishing while keeping the subscriber alive
    dblEntry.Unpublish();
  }
};
```

C++ (Handle-based)

```
class Example {
  // the entry is an instance variable, but since it's a handle, it's
  // not automatically released, so we need a destructor
  NT_Entry dblEntry;

 public:
  explicit Example(NT_Topic dblTopic) {
    // start subscribing
    // Using a custom type string for types other than raw and string is not␣
↪recommended.
    dblEntry = nt::GetEntry(dblTopic, NT_DOUBLE, "double");

    // publish and subscribe options may be specified using PubSubOptions
    dblEntry =
        nt::GetEntry(dblTopic, NT_DOUBLE, "double",
        {.pollStorage = 10, .keepDuplicates = true});
  }

  void Periodic() {
    // entries support all the same methods as subscribers:
    double val = nt::GetDouble(dblEntry, 0.0);
```

(suite sur la page suivante)

(suite de la page précédente)

```
    nt::TimestampedDouble tsVal = nt::GetAtomic(dblEntry, 0.0);
    std::vector<nt::TimestampedDouble> tsUpdates =␣
↪nt::ReadQueueDouble(dblEntry);
    std::vector<double> valUpdates = nt::ReadQueueValuesDouble(dblEntry);

    // entries also support all the same methods as publishers; the first␣
↪time
    // one of these is called, an internal publisher is automatically created
    nt::SetDefaultDouble(dblPub, 0.0);
    nt::SetDouble(dblPub, 1.0);
    nt::SetDouble(dblPub, 2.0, 0);  // 0 = use current time
    int64_t time = nt::Now();
    nt::SetDouble(dblPub, 3.0, time);
  }

  void Unpublish() {
    // you can stop publishing while keeping the subscriber alive
    nt::Unpublish(dblEntry);
  }

  ~Example() {
    // stop publishing and subscribing
    nt::ReleaseEntry(dblEntry);
  }
```

C

```
// This code assumes that a NT_Topic dblTopic variable already exists

// start subscribing
// Using a custom type string for types other than raw and string is not␣
↪recommended.
NT_Entry dblEntry = NT_GetEntryEx(dblTopic, NT_DOUBLE, "double", NULL, 0);

// publish and subscribe options may be specified using NT_PubSubOptions
struct NT_PubSubOptions options;
memset(&options, 0, sizeof(options));
options.structSize = sizeof(options);
options.keepDuplicates = 1;  // true
options.pollStorage = 10;
NT_Entry dblEntry = NT_GetEntryEx(dblTopic, NT_DOUBLE, "double", &options);

// entries support all the same methods as subscribers:
double val = NT_GetDouble(dblEntry, 0.0);

struct NT_TimestampedDouble tsVal;
NT_GetAtomic(dblEntry, 0.0, &tsVal);
NT_DisposeTimestamped(&tsVal);

size_t tsUpdatesLen;
struct NT_TimestampedDouble* tsUpdates = NT_ReadQueueDouble(dblEntry, &
↪tsUpdatesLen);
NT_FreeQueueDouble(tsUpdates, tsUpdatesLen);
```

(suite sur la page suivante)

(suite de la page précédente)

```
size_t valUpdatesLen;
double* valUpdates = NT_ReadQueueValuesDouble(dblEntry, &valUpdatesLen);
NT_FreeDoubleArray(valUpdates, valUpdatesLen);

// entries also support all the same methods as publishers; the first time
// one of these is called, an internal publisher is automatically created
NT_SetDefaultDouble(dblPub, 0.0);
NT_SetDouble(dblPub, 1.0);
NT_SetDouble(dblPub, 2.0, 0);  // 0 = use current time
int64_t time = NT_Now();
NT_SetDouble(dblPub, 3.0, time);

// you can stop publishing while keeping the subscriber alive
// it's not necessary to call this before NT_ReleaseEntry()
NT_Unpublish(dblEntry);

// stop subscribing
NT_ReleaseEntry(dblEntry);
```

Python

```
class Example:
    def __init__(self, dblTopic: ntcore.DoubleTopic):

        # start subscribing; the return value must be retained.
        # the parameter is the default value if no value is available when␣
↪get() is called
        self.dblEntry = dblTopic.getEntry(0.0)

        # publish and subscribe options may be specified using PubSubOption
        self.dblEntry = dblTopic.getEntry(
            0.0, ntcore.PubSubOptions(keepDuplicates=True, pollStorage=10)
        )

        # getEntryEx provides the options of using a custom type string.
        # Using a custom type string for types other than raw and string is␣
↪not recommended.
        self.dblEntry = dblTopic.getEntryEx("double", 0.0)

    def periodic(self):
        # entries support all the same methods as subscribers:
        val = self.dblEntry.get()
        val = self.dblEntry.get(-1.0)
        val = self.dblEntry.getAsDouble()
        tsVal = self.dblEntry.getAtomic()
        tsUpdates = self.dblEntry.readQueue()

        # entries also support all the same methods as publishers; the first␣
↪time
        # one of these is called, an internal publisher is automatically␣
↪created
        self.dblEntry.setDefault(0.0)
        self.dblEntry.set(1.0)
        self.dblEntry.set(2.0, 0)  # 0 = use current time
```

(suite sur la page suivante)

(suite de la page précédente)

```
        time = ntcore._now()
        self.dblEntry.set(3.0, time)

    def unpublish(self):
        # you can stop publishing while keeping the subscriber alive
        self.dblEntry.unpublish()

    # often not required in robot code, unless this class doesn't exist for
    # the lifetime of the entire robot program, in which case close() needs␣
↪to be
    # called to stop subscribing
    def close(self):
        # stop subscribing/publishing
        self.dblEntry.close()
```

28.3.4 Using GenericEntry, GenericPublisher, and GenericSubscriber

For the most robust code, using the type-specific Publisher, Subscriber, and Entry classes is recommended, but in some cases it may be easier to write code that uses type-specific get and set function calls instead of having the NetworkTables type be exposed via the class (object) type. The `GenericPublisher` (Java, C++, Python), `GenericSubscriber` (Java, C++, Python), and `GenericEntry` (Java, C++, Python) classes enable this approach.

Java

```
public class Example {
  // the entry is an instance variable so its lifetime matches that of the␣
↪class
  final GenericPublisher pub;
  final GenericSubscriber sub;
  final GenericEntry entry;

  public Example(Topic topic) {
    // start subscribing; the return value must be retained.
    // when publishing, a type string must be provided
    pub = topic.genericPublish("double");

    // subscribing can optionally include a type string
    // unlike type-specific subscribers, no default value is provided
    sub = topic.genericSubscribe();
    sub = topic.genericSubscribe("double");

    // when getting an entry, the type string is also optional; if not␣
↪provided
    // the publisher data type will be determined by the first publisher-
↪creating call
    entry = topic.getGenericEntry();
    entry = topic.getGenericEntry("double");

    // publish and subscribe options may be specified using PubSubOption
    pub = topic.genericPublish("double",
        PubSubOption.keepDuplicates(true), PubSubOption.pollStorage(10));
```

(suite sur la page suivante)

(suite de la page précédente)

```
    sub =
        topic.genericSubscribe(PubSubOption.keepDuplicates(true),␣
↪PubSubOption.pollStorage(10));
    entry =
        topic.getGenericEntry(PubSubOption.keepDuplicates(true),␣
↪PubSubOption.pollStorage(10));

    // genericPublishEx provides the option of setting initial properties.
    pub = topic.genericPublishEx("double", "{\"retained\": true}",
        PubSubOption.keepDuplicates(true), PubSubOption.pollStorage(10));
  }

  public void periodic() {
    // generic subscribers and entries have typed get operations; a default␣
↪must be provided
    double val = sub.getDouble(-1.0);
    double val = entry.getDouble(-1.0);

    // they also support an untyped get (also meets Supplier
↪<NetworkTableValue> interface)
    NetworkTableValue val = sub.get();
    NetworkTableValue val = entry.get();

    // they also support readQueue
    NetworkTableValue[] updates = sub.readQueue();
    NetworkTableValue[] updates = entry.readQueue();

    // publishers and entries have typed set operations; these return false␣
↪if the
    // topic already exists with a mismatched type
    boolean success = pub.setDefaultDouble(1.0);
    boolean success = pub.setBoolean(true);

    // they also implement a generic set and Consumer<NetworkTableValue>␣
↪interface
    boolean success = entry.set(NetworkTableValue.makeDouble(...));
    boolean success = entry.accept(NetworkTableValue.makeDouble(...));
  }

  public void unpublish() {
    // you can stop publishing an entry while keeping the subscriber alive
    entry.unpublish();
  }

  // often not required in robot code, unless this class doesn't exist for
  // the lifetime of the entire robot program, in which case close() needs␣
↪to be
  // called to stop subscribing/publishing
  public void close() {
    pub.close();
    sub.close();
    entry.close();
  }
}
```

C++

```
class Example {
  // the entry is an instance variable so its lifetime matches that of the␣
↪class
  // subscribing/publishing is automatically stopped when dblEntry is␣
↪destroyed by
  // the class destructor
  nt::GenericPublisher pub;
  nt::GenericSubscriber sub;
  nt::GenericEntry entry;

 public:
  Example(nt::Topic topic) {
    // start subscribing; the return value must be retained.
    // when publishing, a type string must be provided
    pub = topic.GenericPublish("double");

    // subscribing can optionally include a type string
    // unlike type-specific subscribers, no default value is provided
    sub = topic.GenericSubscribe();
    sub = topic.GenericSubscribe("double");

    // when getting an entry, the type string is also optional; if not␣
↪provided
    // the publisher data type will be determined by the first publisher-
↪creating call
    entry = topic.GetEntry();
    entry = topic.GetEntry("double");

    // publish and subscribe options may be specified using PubSubOptions
    pub = topic.GenericPublish("double",
        {.pollStorage = 10, .keepDuplicates = true});
    sub = topic.GenericSubscribe(
        {.pollStorage = 10, .keepDuplicates = true});
    entry = topic.GetGenericEntry(
        {.pollStorage = 10, .keepDuplicates = true});

    // genericPublishEx provides the option of setting initial properties.
    pub = topic.genericPublishEx("double", {{"myprop", 5}},
        {.pollStorage = 10, .keepDuplicates = true});
  }

  void Periodic() {
    // generic subscribers and entries have typed get operations; a default␣
↪must be provided
    double val = sub.GetDouble(-1.0);
    double val = entry.GetDouble(-1.0);

    // they also support an untyped get
    nt::NetworkTableValue val = sub.Get();
    nt::NetworkTableValue val = entry.Get();

    // they also support readQueue
    std::vector<nt::NetworkTableValue> updates = sub.ReadQueue();
    std::vector<nt::NetworkTableValue> updates = entry.ReadQueue();
```

(suite sur la page suivante)

(suite de la page précédente)

```
    // publishers and entries have typed set operations; these return false␣
↪if the
    // topic already exists with a mismatched type
    bool success = pub.SetDefaultDouble(1.0);
    bool success = pub.SetBoolean(true);

    // they also implement a generic set and Consumer<NetworkTableValue>␣
↪interface
    bool success = entry.Set(nt::NetworkTableValue::MakeDouble(...));
  }

  void Unpublish() {
    // you can stop publishing an entry while keeping the subscriber alive
    entry.Unpublish();
  }
};
```

Python

```
class Example:
    def __init__(self, topic: ntcore.Topic):

        # start subscribing; the return value must be retained.
        # when publishing, a type string must be provided
        self.pub = topic.genericPublish("double")

        # subscribing can optionally include a type string
        # unlike type-specific subscribers, no default value is provided
        self.sub = topic.genericSubscribe()
        self.sub = topic.genericSubscribe("double")

        # when getting an entry, the type string is also optional; if not␣
↪provided
        # the publisher data type will be determined by the first publisher-
↪creating call
        self.entry = topic.getGenericEntry()
        self.entry = topic.getGenericEntry("double")

        # publish and subscribe options may be specified using PubSubOption
        self.pub = topic.genericPublish(
            "double", ntcore.PubSubOptions(keepDuplicates=True,␣
↪pollStorage=10)
        )
        self.sub = topic.genericSubscribe(
            ntcore.PubSubOptions(keepDuplicates=True, pollStorage=10)
        )
        self.entry = topic.getGenericEntry(
            ntcore.PubSubOptions(keepDuplicates=True, pollStorage=10)
        )

        # genericPublishEx provides the option of setting initial properties.
        self.pub = topic.genericPublishEx(
            "double",
            '{"retained": true}',
```

(suite sur la page suivante)

(suite de la page précédente)

```
            ntcore.PubSubOptions(keepDuplicates=True, pollStorage=10),
        )

    def periodic(self):
        # generic subscribers and entries have typed get operations; a␣
↪default must be provided
        val = self.sub.getDouble(-1.0)
        val = self.entry.getDouble(-1.0)

        # they also support an untyped get (also meets Supplier
↪<NetworkTableValue> interface)
        val = self.sub.get()
        val = self.entry.get()

        # they also support readQueue
        updates = self.sub.readQueue()
        updates = self.entry.readQueue()

        # publishers and entries have typed set operations; these return␣
↪false if the
        # topic already exists with a mismatched type
        success = self.pub.setDefaultDouble(1.0)
        success = self.pub.setBoolean(True)

        # they also implement a generic set
        success = self.entry.set(ntcore.Value.makeDouble(...))

    def unpublish(self):
        # you can stop publishing an entry while keeping the subscriber alive
        self.entry.unpublish()

    # often not required in robot code, unless this class doesn't exist for
    # the lifetime of the entire robot program, in which case close() needs␣
↪to be
    # called to stop subscribing/publishing
    def close(self):
        self.pub.close()
        self.sub.close()
        self.entry.close()
```

28.3.5 Subscribing to Multiple Topics

While in most cases it's only necessary to subscribe to individual topics, it is sometimes useful (e.g. in dashboard applications) to subscribe and get value updates for changes to multiple topics. Listeners (see *Listening for Changes*) can be used directly, but creating a `MultiSubscriber` (Java, C++) allows specifying subscription options and reusing the same subscriber for multiple listeners.

Java

```
public class Example {
  // the subscriber is an instance variable so its lifetime matches that of the class
  final MultiSubscriber multiSub;
  final NetworkTableListenerPoller poller;

  public Example(NetworkTableInstance inst) {
    // start subscribing; the return value must be retained.
    // provide an array of topic name prefixes
    multiSub = new MultiSubscriber(inst, new String[] {"/table1/", "/table2/"});

    // subscribe options may be specified using PubSubOption
    multiSub = new MultiSubscriber(inst, new String[] {"/table1/", "/table2/"},
        PubSubOption.keepDuplicates(true));

    // to get value updates from a MultiSubscriber, it's necessary to create a listener
    // (see the listener documentation for more details)
    poller = new NetworkTableListenerPoller(inst);
    poller.addListener(multiSub, EnumSet.of(NetworkTableEvent.Kind.kValueAll));
  }

  public void periodic() {
    // read value events
    NetworkTableEvent[] events = poller.readQueue();

    for (NetworkTableEvent event : events) {
      NetworkTableValue value = event.valueData.value;
    }
  }

  // often not required in robot code, unless this class doesn't exist for
  // the lifetime of the entire robot program, in which case close() needs to be
  // called to stop subscribing
  public void close() {
    // close listener
    poller.close();
    // stop subscribing
    multiSub.close();
  }
}
```

C++

```
class Example {
  // the subscriber is an instance variable so its lifetime matches that of the class
  // subscribing is automatically stopped when multiSub is destroyed by the class destructor
  nt::MultiSubscriber multiSub;
  nt::NetworkTableListenerPoller poller;

 public:
  explicit Example(nt::NetworkTableInstance inst) {
    // start subscribing; the return value must be retained.
    // provide an array of topic name prefixes
    multiSub = nt::MultiSubscriber{inst, {{"/table1/", "/table2/"}}};

    // subscribe options may be specified using PubSubOption
    multiSub = nt::MultiSubscriber{inst, {{"/table1/", "/table2/"}},
        {.keepDuplicates = true}};

    // to get value updates from a MultiSubscriber, it's necessary to create a listener
    // (see the listener documentation for more details)
    poller = nt::NetworkTableListenerPoller{inst};
    poller.AddListener(multiSub, nt::EventFlags::kValueAll);
  }

  void Periodic() {
    // read value events
    std::vector<nt::Event> events = poller.ReadQueue();

    for (auto&& event : events) {
      nt::NetworkTableValue value = event.GetValueEventData()->value;
    }
  }
};
```

C++ (Handle-based)

```
class Example {
  // the subscriber is an instance variable, but since it's a handle, it's
  // not automatically released, so we need a destructor
  NT_MultiSubscriber multiSub;
  NT_ListenerPoller poller;

 public:
  explicit Example(NT_Inst inst) {
    // start subscribing; the return value must be retained.
    // provide an array of topic name prefixes
    multiSub = nt::SubscribeMultiple(inst, {{"/table1/", "/table2/"}});

    // subscribe options may be specified using PubSubOption
    multiSub = nt::SubscribeMultiple(inst, {{"/table1/", "/table2/"}},
        {.keepDuplicates = true});
```

(suite sur la page suivante)

(suite de la page précédente)

```
    // to get value updates from a MultiSubscriber, it's necessary to create␣
↪a listener
    // (see the listener documentation for more details)
    poller = nt::CreateListenerPoller(inst);
    nt::AddPolledListener(poller, multiSub, nt::EventFlags::kValueAll);
  }

  void Periodic() {
    // read value events
    std::vector<nt::Event> events = nt::ReadListenerQueue(poller);

    for (auto&& event : events) {
      nt::NetworkTableValue value = event.GetValueEventData()->value;
    }
  }

  ~Example() {
    // close listener
    nt::DestroyListenerPoller(poller);
    // stop subscribing
    nt::UnsubscribeMultiple(multiSub);
  }
```

C

```
// This code assumes that a NT_Inst inst variable already exists

// start subscribing
// provide an array of topic name prefixes
struct NT_String prefixes[2];
prefixes[0].str = "/table1/";
prefixes[0].len = 8;
prefixes[1].str = "/table2/";
prefixes[1].len = 8;
NT_MultiSubscriber multiSub = NT_SubscribeMultiple(inst, prefixes, 2, NULL,␣
↪0);

// subscribe options may be specified using NT_PubSubOptions
struct NT_PubSubOptions options;
memset(&options, 0, sizeof(options));
options.structSize = sizeof(options);
options.keepDuplicates = 1;  // true
NT_MultiSubscriber multiSub = NT_SubscribeMultiple(inst, prefixes, 2, &
↪options);

// to get value updates from a MultiSubscriber, it's necessary to create a␣
↪listener
// (see the listener documentation for more details)
NT_ListenerPoller poller = NT_CreateListenerPoller(inst);
NT_AddPolledListener(poller, multiSub, NT_EVENT_VALUE_ALL);

// read value events
size_t eventsLen;
```

(suite sur la page suivante)

(suite de la page précédente)

```
struct NT_Event* events = NT_ReadListenerQueue(poller, &eventsLen);

for (size_t i = 0; i < eventsLen; i++) {
  NT_Value* value = &events[i].data.valueData.value;
}

NT_DisposeEventArray(events, eventsLen);

// close listener
NT_DestroyListenerPoller(poller);
// stop subscribing
NT_UnsubscribeMultiple(multiSub);
```

Python

```
class Example:
    def __init__(self, inst: ntcore.NetworkTableInstance):

        # start subscribing; the return value must be retained.
        # provide an array of topic name prefixes
        self.multiSub = ntcore.MultiSubscriber(inst, ["/table1/", "/table2/
↪"])

        # subscribe options may be specified using PubSubOption
        self.multiSub = ntcore.MultiSubscriber(
            inst, ["/table1/", "/table2/"], ntcore.
↪PubSubOptions(keepDuplicates=True)
        )

        # to get value updates from a MultiSubscriber, it's necessary to␣
↪create a listener
        # (see the listener documentation for more details)
        self.poller = ntcore.NetworkTableListenerPoller(inst)
        self.poller.addListener(self.multiSub, ntcore.EventFlags.kValueAlls)

    def periodic(self):
        # read value events
        events = self.poller.readQueue()

        for event in events:
            value: ntcore.Value = event.data.value

    # often not required in robot code, unless this class doesn't exist for
    # the lifetime of the entire robot program, in which case close() needs␣
↪to be
    # called to stop subscribing
    def close(self):
        # close listener
        self.poller.close()
        # stop subscribing
        self.multiSub.close()
```

28.3.6 Publish/Subscribe Options

Publishers and subscribers have various options that affect their behavior. Options can only be set at the creation of the publisher, subscriber, or entry. Options set on an entry affect both the publisher and subscriber portions of the entry. The above examples show how options can be set when creating a publisher or subscriber.

Subscriber options :

- `pollStorage` : Polling storage size for a subscription. Specifies the maximum number of updates NetworkTables should store between calls to the subscriber's `readQueue()` function. If zero, defaults to 1 if sendAll is false, 20 if sendAll is true.
- `topicsOnly` : Don't send value changes, only topic announcements. Defaults to false. As a client doesn't get topic announcements for topics it is not subscribed to, this option may be used with `MultiSubscriber` to get topic announcements for a particular topic name prefix, without also getting all value changes.
- `excludePublisher` : Used to exclude a single publisher's updates from being queued to the subscriber's `readQueue()` function. This is primarily useful in scenarios where you don't want local value updates to be « echoed back » to a local subscriber. Regardless of this setting, the topic value is updated–this only affects `readQueue()` on this subscriber.
- `disableRemote` : If true, remote value updates are not queued for `readQueue()`. Defaults to false. Regardless of this setting, the topic value is updated–this only affects `readQueue()` on this subscriber.
- `disableLocal` : If true, local value updates are not queued for `readQueue()`. Defaults to false. Regardless of this setting, the topic value is updated–this only affects `readQueue()` on this subscriber.

Subscriber and publisher options :

- `periodic` : How frequently changes will be sent over the network, in seconds. NetworkTables may send more frequently than this (e.g. use a combined minimum period for all values) or apply a restricted range to this value. The default is 0.1 seconds. For publishers, it specifies how frequently local changes should be sent over the network ; for subscribers, it is a request to the server to send server changes at the requested rate. Note that regardless of the setting of this option, only value changes are sent, unless the `keepDuplicates` option is set.
- `sendAll` : If true, send all value changes over the network. Defaults to false. As with `periodic`, this is a request to the server for subscribers and a behavior change for publishers.
- `keepDuplicates` : If true, preserves duplicate value changes (rather than ignoring them). Defaults to false. As with `periodic`, this is a request to the server for subscribers and a behavior change for publishers.

Entry options :

- `excludeSelf` : Provides the same behavior as `excludePublisher` for the entry's internal publisher. Defaults to false.

28.3.7 NetworkTableEntry

`NetworkTableEntry` (Java, C++, Python) is a class that exists for backwards compatibility. New code should prefer using type-specific Publisher and Subscriber classes, or GenericEntry if non-type-specific access is needed.

It is similar to `GenericEntry` in that it supports both publishing and subscribing in a single object. However, unlike `GenericEntry`, `NetworkTableEntry` is not released (e.g. unsubscribes/unpublishes) if `close()` is called (in Java) or the object is destroyed (in C++) ; instead, it operates similar to `Topic`, in that only a single `NetworkTableEntry` exists for each topic and it lasts for the lifetime of the instance.

28.4 NetworkTables Instances

The NetworkTables implementation supports simultaneous operation of multiple « instances. » Each instance has a completely independent set of topics, publishers, subscribers, and client/server state. This feature is mainly useful for unit testing. It allows a single program to be a member of two *NetworkTables* « networks » that contain different (and unrelated) sets of topics, or running both client and server instances in a single program.

For most general usage, you should use the « default » instance, as all current dashboard programs can only connect to a single NetworkTables server at a time. Normally the default instance is set up on the robot as a server, and used for communication with the dashboard program running on your driver station computer. This is what the SmartDashboard and LiveWindow classes use.

However, if you wanted to do unit testing of your robot program's NetworkTables communications, you could set up your unit tests such that they create a separate client instance (still within the same program) and have it connect to the server instance that the main robot code is running.

The `NetworkTableInstance` (Java, C++, Python) class provides the API abstraction for instances. The number of instances that can be simultaneously created is limited to 16 (including the default instance), so when using multiple instances in cases such as unit testing code, it's important to destroy instances that are no longer needed.

Destroying a NetworkTableInstance frees all resources related to the instance. All classes or handles that reference the instance (e.g. Topics, Publishers, and Subscribers) are invalidated and may result in unexpected behavior if used after the instance is destroyed–in particular, instance handles are reused so it's possible for a handle « left over » from a previously destroyed instance to refer to an unexpected resource in a newly created instance.

Java

```
// get the default NetworkTable instance
NetworkTableInstance defaultInst = NetworkTableInstance.getDefault();

// create a NetworkTable instance
NetworkTableInstance inst = NetworkTableInstance.create();

// destroy a NetworkTable instance
inst.close();
```

C++

```
// get the default NetworkTable instance
nt::NetworkTableInstance defaultInst =␣
↪nt::NetworkTableInstance::GetDefault();

// create a NetworkTable instance
nt::NetworkTableInstance inst = nt::NetworkTableInstance::Create();

// destroy a NetworkTable instance; NetworkTableInstance objects are not RAII
nt::NetworkTableInstance::Destroy(inst);
```

C++ (Handle-based)

```
// get the default NetworkTable instance
NT_Instance defaultInst = nt::GetDefaultInstance();

// create a NetworkTable instance
NT_Instance inst = nt::CreateInstance();

// destroy a NetworkTable instance
nt::DestroyInstance(inst);
```

C

```
// get the default NetworkTable instance
NT_Instance defaultInst = NT_GetDefaultInstance();

// create a NetworkTable instance
NT_Instance inst = NT_CreateInstance();

// destroy a NetworkTable instance
NT_DestroyInstance(inst);
```

Python

```
import ntcore

# get the default NetworkTable instance
defaultInst = ntcore.NetworkTableInstance.getDefault()

# create a NetworkTable instance
inst = ntcore.NetworkTableInstance.create()

# destroy a NetworkTable instance
ntcore.NetworkTableInstance.destroy(inst)
```

28.5 Réseautique NetworkTables

L'avantage que le programme du robot soit le serveur est qu'il s'agit d'un nom de réseau connu (et typiquement à une adresse connue) qui est basé sur le numéro de l'équipe. Cela est pourquoi il est possible dans le client NetworkTables, l'API client et plusieurs tableaux de bord de simplement fournir le numéro de l'équipe au lieu d'une adresse de serveur. Puisque le programme du robot est le serveur, notez que cela signifie que NetworkTables s'exécute sur l'ordinateur local lorsqu'en mode simulation.

28.5.1 Démarrer un serveur NetworkTables

Java

```
NetworkTableInstance inst = NetworkTableInstance.getDefault();
inst.startServer();
```

C++

```
nt::NetworkTableInstance inst = nt::NetworkTableInstance::GetDefault();
inst.StartServer();
```

C++ (Handle-based)

```
NT_Inst inst = nt::GetDefaultInstance();
nt::StartServer(inst, "networktables.json", "", NT_DEFAULT_PORT3, NT_DEFAULT_
↪PORT4);
```

C

```
NT_Inst inst = NT_GetDefaultInstance();
NT_StartServer(inst, "networktables.json", "", NT_DEFAULT_PORT3, NT_DEFAULT_
↪PORT4);
```

Python

```
import ntcore

inst = ntcore.NetworkTableInstance.getDefault()
inst.startServer()
```

28.5.2 Démarrer un client NetworkTables

Java

```
NetworkTableInstance inst = NetworkTableInstance.getDefault();

// start a NT4 client
inst.startClient4("example client");

// connect to a roboRIO with team number TEAM
inst.setServerTeam(TEAM);
```

(suite sur la page suivante)

(suite de la page précédente)

```
// starting a DS client will try to get the roboRIO address from the DS␣
↪application
inst.startDSClient();

// connect to a specific host/port
inst.setServer("host", NetworkTableInstance.kDefaultPort4)
```

C++

```
nt::NetworkTableInstance inst = nt::NetworkTableInstance::GetDefault();

// start a NT4 client
inst.StartClient4("example client");

// connect to a roboRIO with team number TEAM
inst.SetServerTeam(TEAM);

// starting a DS client will try to get the roboRIO address from the DS␣
↪application
inst.StartDSClient();

// connect to a specific host/port
inst.SetServer("host", NT_DEFAULT_PORT4)
```

C++ (Handle-based)

```
NT_Inst inst = nt::GetDefaultInstance();

// start a NT4 client
nt::StartClient4(inst, "example client");

// connect to a roboRIO with team number TEAM
nt::SetServerTeam(inst, TEAM);

// starting a DS client will try to get the roboRIO address from the DS␣
↪application
nt::StartDSClient(inst);

// connect to a specific host/port
nt::SetServer(inst, "host", NT_DEFAULT_PORT4)
```

C

```
NT_Inst inst = NT_GetDefaultInstance();

// start a NT4 client
NT_StartClient4(inst, "example client");

// connect to a roboRIO with team number TEAM
NT_SetServerTeam(inst, TEAM);

// starting a DS client will try to get the roboRIO address from the DS␣
↪application
NT_StartDSClient(inst);

// connect to a specific host/port
NT_SetServer(inst, "host", NT_DEFAULT_PORT4)
```

Python

```
import ntcore

inst = ntcore.NetworkTableInstance.getDefault()

# start a NT4 client
inst.startClient4("example client")

# connect to a roboRIO with team number TEAM
inst.setServerTeam(TEAM)

# starting a DS client will try to get the roboRIO address from the DS␣
↪application
inst.startDSClient()

# connect to a specific host/port
inst.setServer("host", ntcore.NetworkTableInstance.kDefaultPort4)
```

28.6 Listening for Changes

A common use case for *NetworkTables* is where a coprocessor generates values that need to be sent to the robot. For example, imagine that some image processing code running on a coprocessor computes the heading and distance to a goal and sends those values to the robot. In this case it might be desirable for the robot program to be notified when new values arrive.

There are a few different ways to detect that a topic's value has changed; the easiest way is to periodically call a subscriber's `get()`, `readQueue()`, or `readQueueValues()` function from the robot's periodic loop, as shown below :

Java

```
public class Example {
  final DoubleSubscriber ySub;
  double prev;

  public Example() {
    // get the default instance of NetworkTables
    NetworkTableInstance inst = NetworkTableInstance.getDefault();

    // get the subtable called "datatable"
    NetworkTable datatable = inst.getTable("datatable");

    // subscribe to the topic in "datatable" called "Y"
    ySub = datatable.getDoubleTopic("Y").subscribe(0.0);
  }

  public void periodic() {
    // get() can be used with simple change detection to the previous value
    double value = ySub.get();
    if (value != prev) {
      prev = value;  // save previous value
      System.out.println("X changed value: " + value);
    }

    // readQueueValues() provides all value changes since the last call;
    // this way it's not possible to miss a change by polling too slowly
    for (double iterVal : ySub.readQueueValues()) {
      System.out.println("X changed value: " + iterVal);
    }

    // readQueue() is similar to readQueueValues(), but provides timestamps
    // for each change as well
    for (TimestampedDouble tsValue : ySub.readQueue()) {
      System.out.println("X changed value: " + tsValue.value + " at local␣
↪time " + tsValue.timestamp);
    }
  }

  // may not be necessary for robot programs if this class lives for
  // the length of the program
  public void close() {
    ySub.close();
  }
}
```

C++

```
class Example {
  nt::DoubleSubscriber ySub;
  double prev = 0;

 public:
  Example() {
    // get the default instance of NetworkTables
    nt::NetworkTableInstance inst = nt::NetworkTableInstance::GetDefault();

    // get the subtable called "datatable"
    auto datatable = inst.GetTable("datatable");

    // subscribe to the topic in "datatable" called "Y"
    ySub = datatable->GetDoubleTopic("Y").Subscribe(0.0);
  }

  void Periodic() {
    // Get() can be used with simple change detection to the previous value
    double value = ySub.Get();
    if (value != prev) {
      prev = value;  // save previous value
      fmt::print("X changed value: {}\n", value);
    }

    // ReadQueueValues() provides all value changes since the last call;
    // this way it's not possible to miss a change by polling too slowly
    for (double iterVal : ySub.ReadQueueValues()) {
      fmt::print("X changed value: {}\n", iterVal);
    }

    // ReadQueue() is similar to ReadQueueValues(), but provides timestamps
    // for each change as well
    for (nt::TimestampedDouble tsValue : ySub.ReadQueue()) {
      fmt::print("X changed value: {} at local time {}\n", tsValue.value,␣
↪tsValue.timestamp);
    }
  }
};
```

C++ (Handle-based)

```
class Example {
  NT_Subscriber ySub;
  double prev = 0;

 public:
  Example() {
    // get the default instance of NetworkTables
    NT_Inst inst = nt::GetDefaultInstance();

    // subscribe to the topic in "datatable" called "Y"
    ySub = nt::Subscribe(nt::GetTopic(inst, "/datatable/Y"), NT_DOUBLE,
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪"double");
  }

  void Periodic() {
    // Get() can be used with simple change detection to the previous value
    double value = nt::GetDouble(ySub, 0.0);
    if (value != prev) {
      prev = value;  // save previous value
      fmt::print("X changed value: {}\n", value);
    }

    // ReadQueue() provides all value changes since the last call;
    // this way it's not possible to miss a change by polling too slowly
    for (nt::TimestampedDouble value : nt::ReadQueueDouble(ySub)) {
      fmt::print("X changed value: {} at local time {}\n", tsValue.value,␣
↪tsValue.timestamp);
    }
  }
};
```

Python

```
class Example:
    def __init__(self) -> None:

        # get the default instance of NetworkTables
        inst = ntcore.NetworkTableInstance.getDefault()

        # get the subtable called "datatable"
        datatable = inst.getTable("datatable")

        # subscribe to the topic in "datatable" called "Y"
        self.ySub = datatable.getDoubleTopic("Y").subscribe(0.0)

        self.prev = 0

    def periodic(self):
        # get() can be used with simple change detection to the previous␣
↪value
        value = self.ySub.get()
        if value != self.prev:
            self.prev = value
            # save previous value
            print("X changed value: " + value)

        # readQueue() provides all value changes since the last call;
        # this way it's not possible to miss a change by polling too slowly
        for tsValue in self.ySub.readQueue():
            print(f"X changed value: {tsValue.value} at local time {tsValue.
↪time}")

    # may not be necessary for robot programs if this class lives for
    # the length of the program
    def close(self):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        self.ySub.close()
```

With a command-based robot, it's also possible to use `NetworkBooleanEvent` to link boolean topic changes to callback actions (e.g. running commands).

While these functions suffice for value changes on a single topic, they do not provide insight into changes to topics (when a topic is published or unpublished, or when a topic's properties change) or network connection changes (e.g. when a client connects or disconnects). They also don't provide a way to get in-order updates for value changes across multiple topics. For these needs, NetworkTables provides an event listener facility.

The easiest way to use listeners is via `NetworkTableInstance`. For more automatic control over listener lifetime (particularly in C++), and to operate without a background thread, NetworkTables also provides separate classes for both polled listeners (`NetworkTableListenerPoller`), which store events into an internal queue that must be periodically read to get the queued events, and threaded listeners (`NetworkTableListener`), which call a callback function from a background thread.

28.6.1 NetworkTableEvent

All listener callbacks take a single `NetworkTableEvent` parameter, and similarly, reading a listener poller returns an array of `NetworkTableEvent`. The event contains information including what kind of event it is (e.g. a value update, a new topic, a network disconnect), the handle of the listener that caused the event to be generated, and more detailed information that depends on the type of the event (connection information for connection events, topic information for topic-related events, value data for value updates, and the log message for log message events).

28.6.2 Using NetworkTableInstance to Listen for Changes

The below example listens to various kinds of events using `NetworkTableInstance`. The listener callback provided to any of the addListener functions will be called asynchronously from a background thread when a matching event occurs.

> **Avertissement :** Because the listener callback is called from a separate background thread, it's important to use thread-safe synchronization approaches such as mutexes or atomics to pass data to/from the main code and the listener callback function.

The `addListener` functions in NetworkTableInstance return a listener handle. This can be used to remove the listener later.

Java

```
public class Example {
  final DoubleSubscriber ySub;
  // use an AtomicReference to make updating the value thread-safe
  final AtomicReference<Double> yValue = new AtomicReference<Double>();
  // retain listener handles for later removal
  int connListenerHandle;
  int valueListenerHandle;
  int topicListenerHandle;

  public Example() {
    // get the default instance of NetworkTables
    NetworkTableInstance inst = NetworkTableInstance.getDefault();

    // add a connection listener; the first parameter will cause the
    // callback to be called immediately for any current connections
    connListenerHandle = inst.addConnectionListener(true, event -> {
      if (event.is(NetworkTableEvent.Kind.kConnected)) {
        System.out.println("Connected to " + event.connInfo.remote_id);
      } else if (event.is(NetworkTableEvent.Kind.kDisconnected)) {
        System.out.println("Disconnected from " + event.connInfo.remote_id);
      }
    });

    // get the subtable called "datatable"
    NetworkTable datatable = inst.getTable("datatable");

    // subscribe to the topic in "datatable" called "Y"
    ySub = datatable.getDoubleTopic("Y").subscribe(0.0);

    // add a listener to only value changes on the Y subscriber
    valueListenerHandle = inst.addListener(
        ySub,
        EnumSet.of(NetworkTableEvent.Kind.kValueAll),
        event -> {
          // can only get doubles because it's a DoubleSubscriber, but
          // could check value.isDouble() here too
          yValue.set(event.valueData.value.getDouble());
        });

    // add a listener to see when new topics are published within datatable
    // the string array is an array of topic name prefixes.
    topicListenerHandle = inst.addListener(
        new String[] { datatable.getPath() + "/" },
        EnumSet.of(NetworkTableEvent.Kind.kTopic),
        event -> {
          if (event.is(NetworkTableEvent.Kind.kPublish)) {
            // topicInfo.name is the full topic name, e.g. "/datatable/X"
            System.out.println("newly published " + event.topicInfo.name);
          }
        });
  }

  public void periodic() {
    // get the latest value by reading the AtomicReference; set it to null
    // when we read to ensure we only get value changes
```

(suite sur la page suivante)

(suite de la page précédente)

```
    Double value = yValue.getAndSet(null);
    if (value != null) {
      System.out.println("got new value " + value);
    }
  }

  // may not be needed for robot programs if this class exists for the
  // lifetime of the program
  public void close() {
    NetworkTableInstance inst = NetworkTableInstance.getDefault();
    inst.removeListener(topicListenerHandle);
    inst.removeListener(valueListenerHandle);
    inst.removeListener(connListenerHandle);
    ySub.close();
  }
}
```

C++

```
class Example {
  nt::DoubleSubscriber ySub;
  // use a mutex to make updating the value and flag thread-safe
  wpi::mutex mutex;
  double yValue;
  bool yValueUpdated = false;
  // retain listener handles for later removal
  NT_Listener connListenerHandle;
  NT_Listener valueListenerHandle;
  NT_Listener topicListenerHandle;

 public:
  Example() {
    // get the default instance of NetworkTables
    nt::NetworkTableInstance inst = nt::NetworkTableInstance::GetDefault();

    // add a connection listener; the first parameter will cause the
    // callback to be called immediately for any current connections
    connListenerHandle = inst.AddConnectionListener(true, [] (const␣
↪nt::Event& event) {
      if (event.Is(nt::EventFlags::kConnected)) {
        fmt::print("Connected to {}\n", event.GetConnectionInfo()->remote_
↪id);
      } else if (event.Is(nt::EventFlags::kDisconnected)) {
        fmt::print("Disconnected from {}\n", event.GetConnectionInfo()->
↪remote_id);
      }
    });

    // get the subtable called "datatable"
    auto datatable = inst.GetTable("datatable");

    // subscribe to the topic in "datatable" called "Y"
    ySub = datatable.GetDoubleTopic("Y").Subscribe(0.0);
```

(suite sur la page suivante)

(suite de la page précédente)

```
    // add a listener to only value changes on the Y subscriber
    valueListenerHandle = inst.AddListener(
        ySub,
        nt::EventFlags::kValueAll,
        [this] (const nt::Event& event) {
          // can only get doubles because it's a DoubleSubscriber, but
          // could check value.IsDouble() here too
          std::scoped_lock lock{mutex};
          yValue = event.GetValueData()->value.GetDouble();
          yValueUpdated = true;
        });

    // add a listener to see when new topics are published within datatable
    // the string array is an array of topic name prefixes.
    topicListenerHandle = inst.AddListener(
        {{fmt::format("{}/", datatable->GetPath())}},
        nt::EventFlags::kTopic,
        [] (const nt::Event& event) {
          if (event.Is(nt::EventFlags::kPublish)) {
            // name is the full topic name, e.g. "/datatable/X"
            fmt::print("newly published {}\n", event.GetTopicInfo()->name);
          }
        });
  }

  void Periodic() {
    // get the latest value by reading the value; set it to false
    // when we read to ensure we only get value changes
    wpi::scoped_lock lock{mutex};
    if (yValueUpdated) {
      yValueUpdated = false;
      fmt::print("got new value {}\n", yValue);
    }
  }

  ~Example() {
    nt::NetworkTableInstance inst = nt::NetworkTableInstance::GetDefault();
    inst.RemoveListener(connListenerHandle);
    inst.RemoveListener(valueListenerHandle);
    inst.RemoveListener(topicListenerHandle);
  }
};
```

Python

```
import ntcore
import threading

class Example:
    def __init__(self) -> None:

        # get the default instance of NetworkTables
        inst = ntcore.NetworkTableInstance.getDefault()
```

(suite sur la page suivante)

(suite de la page précédente)

```
        # Use a mutex to ensure thread safety
        self.lock = threading.Lock()
        self.yValue = None

        # add a connection listener; the first parameter will cause the
        # callback to be called immediately for any current connections
        def _connect_cb(event: ntcore.Event):
            if event.is_(ntcore.EventFlags.kConnected):
                print("Connected to", event.data.remote_id)
            elif event.is_(ntcore.EventFlags.kDisconnected):
                print("Disconnected from", event.data.remote_id)

        self.connListenerHandle = inst.addConnectionListener(True, _connect_
↪cb)

        # get the subtable called "datatable"
        datatable = inst.getTable("datatable")

        # subscribe to the topic in "datatable" called "Y"
        self.ySub = datatable.getDoubleTopic("Y").subscribe(0.0)

        # add a listener to only value changes on the Y subscriber
        def _on_ysub(event: ntcore.Event):
            # can only get doubles because it's a DoubleSubscriber, but
            # could check value.isDouble() here too
            with self.lock:
                self.yValue = event.data.value.getDouble()

        self.valueListenerHandle = inst.addListener(
            self.ySub, ntcore.EventFlags.kValueAll, _on_ysub
        )

        # add a listener to see when new topics are published within␣
↪datatable
        # the string array is an array of topic name prefixes.
        def _on_pub(event: ntcore.Event):
            if event.is_(ntcore.EventFlags.kPublish):
                # topicInfo.name is the full topic name, e.g. "/datatable/X"
                print("newly published", event.data.name)

        self.topicListenerHandle = inst.addListener(
            [datatable.getPath() + "/"], ntcore.EventFlags.kTopic, _on_pub
        )

    def periodic(self):
        # get the latest value by reading the value; set it to null
        # when we read to ensure we only get value changes
        with self.lock:
            value, self.yValue = self.yValue, None

        if value is not None:
            print("got new value", value)

    # may not be needed for robot programs if this class exists for the
    # lifetime of the program
    def close(self):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        inst = ntcore.NetworkTableInstance.getDefault()
        inst.removeListener(self.topicListenerHandle)
        inst.removeListener(self.valueListenerHandle)
        inst.removeListener(self.connListenerHandle)
        self.ySub.close()
```

28.7 Writing a Simple NetworkTables Robot Program

In a robot program, a NetworkTables server is automatically started on the default instance. So it's only necessary to get the default instance to start publishing or subscribing and have it visible over the network.

The example robot program below publishes incrementing X and Y values to a table named `datatable`. The values for X and Y can be easily viewed using the OutlineViewer program that shows the NetworkTables hierarchy and all the values associated with each topic.

JAVA

```
package edu.wpi.first.wpilibj.templates;

import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.networktables.DoublePublisher;
import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableInstance;

public class EasyNetworkTableExample extends TimedRobot {
  DoublePublisher xPub;
  DoublePublisher yPub;

  public void robotInit() {
    // Get the default instance of NetworkTables that was created automatically
    // when the robot program starts
    NetworkTableInstance inst = NetworkTableInstance.getDefault();

    // Get the table within that instance that contains the data. There can
    // be as many tables as you like and exist to make it easier to organize
    // your data. In this case, it's a table called datatable.
    NetworkTable table = inst.getTable("datatable");

    // Start publishing topics within that table that correspond to the X and Y values
    // for some operation in your program.
    // The topic names are actually "/datatable/x" and "/datatable/y".
    xPub = table.getDoubleTopic("x").publish();
    yPub = table.getDoubleTopic("y").publish();
  }

  double x = 0;
  double y = 0;

  public void teleopPeriodic() {
    // Publish values that are constantly increasing.
    xPub.set(x);
```

(suite sur la page suivante)

(suite de la page précédente)

```
    yPub.set(y);
    x += 0.05;
    y += 1.0;
  }
}
```

C++

```
#include <frc/TimedRobot.h>
#include <networktables/DoubleTopic.h>
#include <networktables/NetworkTable.h>
#include <networktables/NetworkTableInstance.h>

class EasyNetworkExample : public frc::TimedRobot {
 public:
  nt::DoublePublisher xPub;
  nt::DoublePublisher yPub;

  void RobotInit() {
    // Get the default instance of NetworkTables that was created automatically
    // when the robot program starts
    auto inst = nt::NetworkTableInstance::GetDefault();

    // Get the table within that instance that contains the data. There can
    // be as many tables as you like and exist to make it easier to organize
    // your data. In this case, it's a table called datatable.
    auto table = inst.GetTable("datatable");

    // Start publishing topics within that table that correspond to the X and Y values
    // for some operation in your program.
    // The topic names are actually "/datatable/x" and "/datatable/y".
    xPub = table->GetDoubleTopic("x").Publish();
    yPub = table->GetDoubleTopic("y").Publish();
  }

  double x = 0;
  double y = 0;

  void TeleopPeriodic() {
    // Publish values that are constantly increasing.
    xPub.Set(x);
    yPub.Set(y);
    x += 0.05;
    y += 0.05;
  }
}

START_ROBOT_CLASS(EasyNetworkExample)
```

PYTHON

```
import ntcore
import wpilib


class EasyNetworkTableExample(wpilib.TimedRobot):
    def robotInit(self) -> None:
        # Get the default instance of NetworkTables that was created automatically
        # when the robot program starts
        inst = ntcore.NetworkTableInstance.getDefault()

        # Get the table within that instance that contains the data. There can
        # be as many tables as you like and exist to make it easier to organize
        # your data. In this case, it's a table called datatable.
        table = inst.getTable("datatable")

        # Start publishing topics within that table that correspond to the X and Y␣
↪values
        # for some operation in your program.
        # The topic names are actually "/datatable/x" and "/datatable/y".
        self.xPub = table.getDoubleTopic("x").publish()
        self.yPub = table.getDoubleTopic("y").publish()

        self.x = 0
        self.y = 0

    def teleopPeriodic(self) -> None:
        # Publish values that are constantly increasing.
        self.xPub.set(self.x)
        self.yPub.set(self.y)
        self.x += 0.05
        self.y += 1.0
```

28.8 Creating a Client-side Program

If all you need to do is have your robot program communicate with a *COTS* coprocessor or a dashboard running on the Driver Station laptop, then the previous examples of writing robot programs are sufficient. But if you would like to write some custom client code that would run on the drivers station or on a coprocessor then you need to know how to build *NetworkTables* programs for those (non-roboRIO) platforms.

Un programme client de base ressemble à l'exemple suivant.

Java

```
import edu.wpi.first.networktables.DoubleSubscriber;
import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableInstance;
import edu.wpi.first.networktables.NetworkTablesJNI;
import edu.wpi.first.util.CombinedRuntimeLoader;

import java.io.IOException;

import edu.wpi.first.cscore.CameraServerJNI;
import edu.wpi.first.math.WPIMathJNI;
import edu.wpi.first.util.WPIUtilJNI;


public class Program {
    public static void main(String[] args) throws IOException {
        NetworkTablesJNI.Helper.setExtractOnStaticLoad(false);
        WPIUtilJNI.Helper.setExtractOnStaticLoad(false);
        WPIMathJNI.Helper.setExtractOnStaticLoad(false);
        CameraServerJNI.Helper.setExtractOnStaticLoad(false);

        CombinedRuntimeLoader.loadLibraries(Program.class, "wpiutiljni",
↪"wpimathjni", "ntcorejni",
                "cscorejnicvstatic");
        new Program().run();
    }

    public void run() {
        NetworkTableInstance inst = NetworkTableInstance.getDefault();
        NetworkTable table = inst.getTable("datatable");
        DoubleSubscriber xSub = table.getDoubleTopic("x").subscribe(0.0);
        DoubleSubscriber ySub = table.getDoubleTopic("y").subscribe(0.0);
        inst.startClient4("example client");
        inst.setServer("localhost"); // where TEAM=190, 294, etc, or use␣
↪inst.setServer("hostname") or similar
        inst.startDSClient(); // recommended if running on DS computer; this␣
↪gets the robot IP from the DS
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                System.out.println("interrupted");
                return;
            }
            double x = xSub.get();
            double y = ySub.get();
            System.out.println("X: " + x + " Y: " + y);
        }
    }
}
```

C++

```
#include <chrono>
#include <thread>
#include <fmt/format.h>
#include <networktables/NetworkTableInstance.h>
#include <networktables/NetworkTable.h>
#include <networktables/DoubleTopic.h>

int main() {
  auto inst = nt::NetworkTableInstance::GetDefault();
  auto table = inst.GetTable("datatable");
  auto xSub = table->GetDoubleTopic("x").Subscribe(0.0);
  auto ySub = table->GetDoubleTopic("y").Subscribe(0.0);
  inst.StartClient4("example client");
  inst.SetServerTeam(TEAM);  // where TEAM=190, 294, etc, or use inst.
↪setServer("hostname") or similar
  inst.StartDSClient();  // recommended if running on DS computer; this gets␣
↪the robot IP from the DS
  while (true) {
    using namespace std::chrono_literals;
    std::this_thread::sleep_for(1s);
    double x = xSub.Get();
    double y = ySub.Get();
    fmt::print("X: {} Y: {}\n", x, y);
  }
}
```

C++ (Handle-based)

```
#include <chrono>
#include <thread>
#include <fmt/format.h>
#include <ntcore_cpp.h>

int main() {
  NT_Inst inst = nt::GetDefaultInstance();
  NT_Subscriber xSub =
      nt::Subscribe(nt::GetTopic(inst, "/datatable/x"), NT_DOUBLE, "double");
  NT_Subscriber ySub =
      nt::Subscribe(nt::GetTopic(inst, "/datatable/y"), NT_DOUBLE, "double");
  nt::StartClient4(inst, "example client");
  nt::SetServerTeam(inst, TEAM, 0);  // where TEAM=190, 294, etc, or use␣
↪inst.setServer("hostname") or similar
  nt::StartDSClient(inst, 0);  // recommended if running on DS computer;␣
↪this gets the robot IP from the DS
  while (true) {
    using namespace std::chrono_literals;
    std::this_thread::sleep_for(1s);
    double x = nt::GetDouble(xSub, 0.0);
    double y = nt::GetDouble(ySub, 0.0);
    fmt::print("X: {} Y: {}\n", x, y);
  }
}
```

C

```c
#include <stdio.h>
#include <threads.h>
#include <time.h>
#include <networktables/ntcore.h>

int main() {
  NT_Instance inst = NT_GetDefaultInstance();
  NT_Subscriber xSub =
      NT_Subscribe(NT_GetTopic(inst, "/datatable/x"), NT_DOUBLE, "double",␣
↪NULL, 0);
  NT_Subscriber ySub =
      NT_Subscribe(NT_GetTopic(inst, "/datatable/y"), NT_DOUBLE, "double",␣
↪NULL, 0);
  NT_StartClient4(inst, "example client");
  NT_SetServerTeam(inst, TEAM);  // where TEAM=190, 294, etc, or use inst.
↪setServer("hostname") or similar
  NT_StartDSClient(inst);  // recommended if running on DS computer; this␣
↪gets the robot IP from the DS
  while (true) {
    thrd_sleep(&(struct timespec){.tv_sec=1}, NULL);
    double x = NT_GetDouble(xSub, 0.0);
    double y = NT_GetDouble(ySub, 0.0);
    printf("X: %f Y: %f\n", x, y);
  }
}
```

Python

```python
#!/usr/bin/env python3

import ntcore
import time

if __name__ == "__main__":
    inst = ntcore.NetworkTableInstance.getDefault()
    table = inst.getTable("datatable")
    xSub = table.getDoubleTopic("x").subscribe(0)
    ySub = table.getDoubleTopic("y").subscribe(0)
    inst.startClient4("example client")
    inst.setServerTeam(TEAM) # where TEAM=190, 294, etc, or use inst.
↪setServer("hostname") or similar
    inst.startDSClient() # recommended if running on DS computer; this gets␣
↪the robot IP from the DS

    while True:
        time.sleep(1)

        x = xSub.get()
        y = ySub.get()
        print(f"X: {x} Y: {y}")
```

In this example an instance of NetworkTables is created and subscribers are created to reference the values of « x » and « y » from a table called « datatable ».

Ensuite, cette instance est démarrée en tant que client NetworkTables avec le numéro d'équipe (le roboRIO est toujours le serveur). De plus, si le programme est en cours d'exécution sur l'ordinateur de Driver Station, en utilisant la méthode startDSClient (), NetworkTables obtiendra l'adresse IP du robot à partir de Driver Station.

Ensuite, ce programme boucle simplement une fois par seconde et obtient les valeurs de x et y et les imprime sur la console. Dans un programme plus réaliste, le client pourrait traiter ou générer des valeurs calculés de ces lectures et les retourner au robot.

28.8.1 Construire avec Gradle

Example build.gradle files are provided in the StandaloneAppSamples Repository Update the GradleRIO version to correspond to the desired WPILib version.

Java

```
plugins {
    id "java"
    id 'application'
    id 'com.github.johnrengelman.shadow' version '8.1.1'
    id "edu.wpi.first.GradleRIO" version "2024.2.1"
    id 'edu.wpi.first.WpilibTools' version '1.3.0'
}

application {
    mainClass = 'Program'
}

wpilibTools.deps.wpilibVersion = wpi.versions.wpilibVersion.get()

def nativeConfigName = 'wpilibNatives'
def nativeConfig = configurations.create(nativeConfigName)

def nativeTasks = wpilibTools.createExtractionTasks {
    configurationName = nativeConfigName
}

nativeTasks.addToSourceSetResources(sourceSets.main)
nativeConfig.dependencies.add wpilibTools.deps.wpilib("wpimath")
nativeConfig.dependencies.add wpilibTools.deps.wpilib("wpinet")
nativeConfig.dependencies.add wpilibTools.deps.wpilib("wpiutil")
nativeConfig.dependencies.add wpilibTools.deps.wpilib("ntcore")
nativeConfig.dependencies.add wpilibTools.deps.wpilib("cscore")
nativeConfig.dependencies.add wpilibTools.deps.wpilibOpenCv("frc" + wpi.
↪frcYear.get(), wpi.versions.opencvVersion.get())

dependencies {
    implementation wpilibTools.deps.wpilibJava("wpiutil")
    implementation wpilibTools.deps.wpilibJava("wpimath")
    implementation wpilibTools.deps.wpilibJava("wpinet")
    implementation wpilibTools.deps.wpilibJava("ntcore")
    implementation wpilibTools.deps.wpilibJava("cscore")
    implementation wpilibTools.deps.wpilibJava("cameraserver")
    implementation wpilibTools.deps.wpilibOpenCvJava("frc" + wpi.frcYear.
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪get(), wpi.versions.opencvVersion.get())

    implementation group: "com.fasterxml.jackson.core", name: "jackson-
↪annotations", version: wpi.versions.jacksonVersion.get()
    implementation group: "com.fasterxml.jackson.core", name: "jackson-core",
↪ version: wpi.versions.jacksonVersion.get()
    implementation group: "com.fasterxml.jackson.core", name: "jackson-
↪databind", version: wpi.versions.jacksonVersion.get()

    implementation group: "org.ejml", name: "ejml-simple", version: wpi.
↪versions.ejmlVersion.get()
    implementation group: "us.hebi.quickbuf", name: "quickbuf-runtime",␣
↪version: wpi.versions.quickbufVersion.get();
}

shadowJar {
    archiveBaseName = "TestApplication"
    archiveVersion = ""
    exclude("module-info.class")
    archiveClassifier.set(wpilibTools.currentPlatform.platformName)
}

wrapper {
    gradleVersion = '8.5'
}
```

C++

Uncomment the appropriate platform as highlighted.

```
plugins {
    id "cpp"
    id "edu.wpi.first.GradleRIO" version "2024.2.1"
}

// Disable local cache, as it won't have the cross artifact necessary
wpi.maven.useLocal = false

// Set to true to run simulation in debug mode
wpi.cpp.debugSimulation = false

def appName = "TestApplication"

nativeUtils.withCrossLinuxArm64()
//nativeUtils.withCrossLinuxArm32() // Uncomment to build for arm32.␣
↪targetPlatform below also needs to be fixed

model {
    components {
        "${appName}"(NativeExecutableSpec) {
            //targetPlatform wpi.platforms.desktop // Uncomment to build on␣
↪whatever the native platform currently is
            targetPlatform wpi.platforms.linuxarm64
            //targetPlatform wpi.platforms.linuxarm32 // Uncomment to build␣
↪for arm32
```

(suite sur la page suivante)

(suite de la page précédente)

```
23
24              sources.cpp {
25                  source {
26                      srcDir 'src/main/cpp'
27                      include '**/*.cpp', '**/*.cc'
28                  }
29                  exportedHeaders {
30                      srcDir 'src/main/include'
31                  }
32              }
33
34              // Enable run tasks for this component
35              wpi.cpp.enableExternalTasks(it)
36
37              wpi.cpp.deps.wpilibStatic(it)
38          }
39      }
40  }
41
42  wrapper {
43      gradleVersion = '8.5'
44  }
```

28.8.2 Building Python

For Python, refer to the *RobotPy install documentation*.

28.9 Migrating from NetworkTables 3.0 to NetworkTables 4.0

NetworkTables 4.0 (new for 2023) has a number of significant API breaking changes from NetworkTables 3.0, the version of NetworkTables used from 2016-2022.

28.9.1 NetworkTableEntry

While `NetworkTableEntry` can still be used (for backwards compatibility), users are encouraged to migrate to use of type-specific Publisher/Subscriber/Entry classes as appropriate, or if necessary, `GenericEntry` (see *Publishing and Subscribing to a Topic*. It's important to note that unlike `NetworkTableEntry`, these classes need to have appropriate lifetime management. Some functionality (e.g. persistent settings) has also moved to `Topic` properties (see *NetworkTables Tables and Topics*).

NT3 code (was) :

JAVA

```
public class Example {
  final NetworkTableEntry yEntry;
  final NetworkTableEntry outEntry;

  public Example() {
    NetworkTableInstance inst = NetworkTableInstance.getDefault();

    // get the subtable called "datatable"
    NetworkTable datatable = inst.getTable("datatable");

    // get the entry in "datatable" called "Y"
    yEntry = datatable.getEntry("Y");

    // get the entry in "datatable" called "Out"
    outEntry = datatable.getEntry("Out");
  }

  public void periodic() {
    // read a double value from Y, and set Out to that value multiplied by 2
    double value = yEntry.getDouble(0.0);  // default to 0
    outEntry.setDouble(value * 2);
  }
}
```

C++

```
class Example {
  nt::NetworkTableEntry yEntry;
  nt::NetworkTableEntry outEntry;

  public:
  Example() {
    nt::NetworkTableInstance inst = nt::NetworkTableInstance::GetDefault();

    // get the subtable called "datatable"
    auto datatable = inst.GetTable("datatable");

    // get the entry in "datatable" called "Y"
    yEntry = datatable->GetEntry("Y");

    // get the entry in "datatable" called "Out"
    outEntry = datatable->GetEntry("Out");
  }

  void Periodic() {
    // read a double value from Y, and set Out to that value multiplied by 2
    double value = yEntry.GetDouble(0.0);  // default to 0
    outEntry.SetDouble(value * 2);
  }
};
```

PYTHON

```
class Example:
    def __init__(self):
        inst = ntcore.NetworkTableInstance.getDefault()

        # get the subtable called "datatable"
        datatable = inst.getTable("datatable")

        # get the entry in "datatable" called "Y"
        self.yEntry = datatable.getEntry("Y")

        # get the entry in "datatable" called "Out"
        self.outEntry = datatable.getEntry("Out")

    def periodic(self):
        # read a double value from Y, and set Out to that value multiplied by 2
        value = self.yEntry.getDouble(0.0)  # default to 0
        self.outEntry.setDouble(value * 2)
```

Recommended NT4 equivalent (should be) :

JAVA

```
public class Example {
  final DoubleSubscriber ySub;
  final DoublePublisher outPub;

  public Example() {
    NetworkTableInstance inst = NetworkTableInstance.getDefault();

    // get the subtable called "datatable"
    NetworkTable datatable = inst.getTable("datatable");

    // subscribe to the topic in "datatable" called "Y"
    // default value is 0
    ySub = datatable.getDoubleTopic("Y").subscribe(0.0);

    // publish to the topic in "datatable" called "Out"
    outPub = datatable.getDoubleTopic("Out").publish();
  }

  public void periodic() {
    // read a double value from Y, and set Out to that value multiplied by 2
    double value = ySub.get();
    outPub.set(value * 2);
  }

  // often not required in robot code, unless this class doesn't exist for
  // the lifetime of the entire robot program, in which case close() needs to be
  // called to stop subscribing
  public void close() {
    ySub.close();
    outPub.close();
  }
}
```

C++

```
class Example {
  nt::DoubleSubscriber ySub;
  nt::DoublePublisher outPub;

  public:
  Example() {
    nt::NetworkTableInstance inst = nt::NetworkTableInstance::GetDefault();

    // get the subtable called "datatable"
    auto datatable = inst.GetTable("datatable");

    // subscribe to the topic in "datatable" called "Y"
    // default value is 0
    ySub = datatable->GetDoubleTopic("Y").Subscribe(0.0);

    // publish to the topic in "datatable" called "Out"
    outPub = datatable->GetDoubleTopic("Out").Publish();
  }

  void Periodic() {
    // read a double value from Y, and set Out to that value multiplied by 2
    double value = ySub.Get();
    outPub.Set(value * 2);
  }
};
```

PYTHON

```
class Example:
    def __init__(self) -> None:
        inst = ntcore.NetworkTableInstance.getDefault()

        # get the subtable called "datatable"
        datatable = inst.getTable("datatable")

        # subscribe to the topic in "datatable" called "Y"
        # default value is 0
        self.ySub = datatable.getDoubleTopic("Y").subscribe(0.0)

        # publish to the topic in "datatable" called "Out"
        self.outPub = datatable.getDoubleTopic("Out").publish()

    def periodic(self):
        # read a double value from Y, and set Out to that value multiplied by 2
        value = self.ySub.get()
        self.outPub.set(value * 2)

    # often not required in robot code, unless this class doesn't exist for
    # the lifetime of the entire robot program, in which case close() needs to be
    # called to stop subscribing
    def close(self):
        self.ySub.close()
        self.outPub.close()
```

28.9.2 Shuffleboard

In WPILib's Shuffleboard classes, usage of `NetworkTableEntry` has been replaced with use of `GenericEntry`. In C++, since `GenericEntry` is non-copyable, return values now return a reference rather than a value.

28.9.3 Force Set Operations

Force set operations have been removed, as it's no longer possible to change a topic's type once it's been published. In most cases calls to `forceSet` can simply be replaced with `set`, but more complex scenarios may require a different design approach (e.g. splitting into different topics).

28.9.4 Listeners

The separate connection, value, and log listeners/events have been unified into a single listener/event. The NetworkTable-level listeners have also been removed. Listeners in many cases can be replaced with subscriber `readQueue()` calls, but if listeners are still required, they can be used via `NetworkTableInstance` (see *Listening for Changes* for more information).

28.9.5 Client/Server Operations

Starting a NetworkTable server now requires specifying both the NT3 port and the NT4 port. For a NT4-only server, the NT3 port can be specified as 0.

A NetworkTable client can only operate in NT3 mode or NT4 mode, not both (there is no provision for automatic fallback). As such, the `startClient()` call has been replaced by `startClient3()` and `startClient4()`. The client must also specify a unique name for itself–the server will reject connection attempts with duplicate names.

28.9.6 C++ Changes

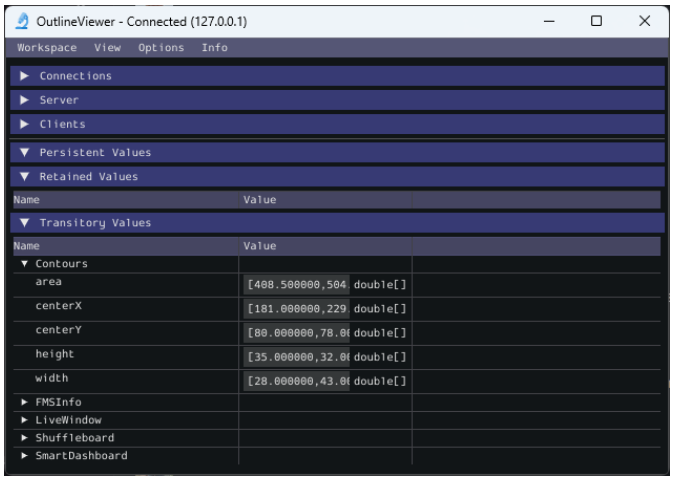
C++ values are now returned/used as value objects (plain `nt::Value`) instead of shared pointers to them (`std::shared_ptr<nt::Value>`).

28.10 Reading Array Values Published by NetworkTables

This article describes how to read values published by *NetworkTables* using a program running on the robot. This is useful when using computer vision where the images are processed on your driver station laptop and the results stored into NetworkTables possibly using a separate vision processor like a raspberry pi, or a tool on the robot like a python program to do the image processing.

Très souvent, les valeurs concernent un ou plusieurs domaines d'intérêt tels que les objectifs ou les pièces de jeu et plusieurs instances sont renvoyées. Dans l'exemple ci-dessous, plusieurs valeurs de x, y, largeur, hauteur et zones sont envoyés par le processeur d'image et le programme du robot peut trier les valeurs qui sont intéressantes grâce à un traitement plus poussé.

28.10.1 Verify the NetworkTables Topics Being Published



You can verify the names of the NetworkTables topics used for publishing the values by using the Outline Viewer application. It is a C++ program in your user directory in the wpilib/<YEAR>/tools folder. The application is started by selecting the « WPILib » menu in Visual Studio Code then Start Tool then « OutlineViewer ». In this example, with the image processing program running (GRIP) you can see the values being put into NetworkTables.

In this case the values are stored in a table called GRIP and a sub-table called myContoursReport. You can see that the values are in brackets and there are 2 values in this case for each topic. The NetworkTables topic names are centerX, centerY, area, height and width.

Les deux exemples suivants sont des programmes simples qui illustrent l'utilisation de NetworkTables. Tout le code est dans la méthode `robotInit()`, il n'est donc exécuté qu'au démarrage du programme. Dans vos programmes, vous obtiendrez plus probablement les valeurs dans le code qui gère vers quelle direction le robot doit viser pour atteindre une cible, ou une boucle de contrôle pendant les périodes autonome ou téléopérée.

28.10.2 Writing a Program to Access the Topics

JAVA

```
DoubleArraySubscriber areasSub;

@Override
public void robotInit() {
  NetworkTable table = NetworkTableInstance.getDefault().getTable("GRIP/
↪mycontoursReport");
  areasSub = table.getDoubleArrayTopic("area").subscribe(new double[] {});
}

@Override
public void teleopPeriodic() {
    double[] areas = areasSub.get();

    System.out.print("areas: " );

    for (double area : areas) {
```

(suite sur la page suivante)

(suite de la page précédente)

```
      System.out.print(area + " ");
    }

    System.out.println();
}
```

C++

```
nt::DoubleArraySubscriber areasSub;

void Robot::RobotInit() override {
  auto table = nt::NetworkTableInstance::GetDefault().GetTable("GRIP/myContoursReport
↪");
  areasSub = table->GetDoubleArrayTopic("area").Subscribe({});
}

void Robot::TeleopPeriodic() override {
  std::cout << "Areas: ";

  std::vector<double> arr = areasSub.Get();

  for (double val : arr) {
    std::cout << val << " ";
  }

  std::cout << std::endl;
}
```

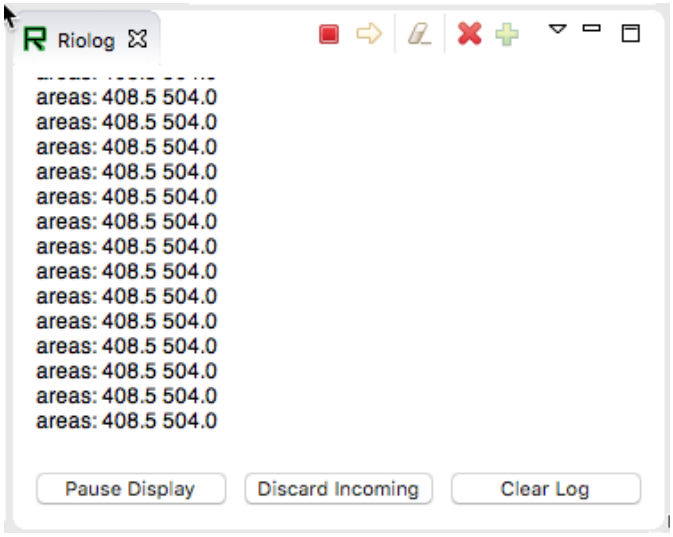
PYTHON

```
def robotInit(self):
    table = ntcore.NetworkTableInstance.getDefault().getTable("GRIP/mycontoursReport")
    self.areasSub = table.getDoubleArrayTopic("area").subscribe([])

def teleopPeriodic(self):
    areas = self.areasSub.get()
    print("Areas:", areas)
```

Les étapes pour obtenir les valeurs et aussi les imprimer sont les suivantes :

1. Déclarer la variable de table qui contiendra l'instance de la sous-table contenant les valeurs.
2. Initialiser l'instance de sous-table afin qu'elle puisse être utilisée plus tard pour récupérer les valeurs.
3. Read the array of values from NetworkTables. In the case of a communicating programs, it's possible that the program producing the output being read here might not yet be available when the robot program starts up. To avoid issues of the data not being ready, a default array of values is supplied. This default value will be returned if the NetworkTables topic hasn't yet been published. This code will loop over the value of areas every 20ms.

28.10.3 Program Output



Dans ce cas, le programme ne regarde seulement que le tableau des zones, mais dans un exemple réel, toutes les valeurs seraient plus susceptibles d'être utilisées. En utilisant le Riolog dans VS Code ou le journal (log) du Driver Station, vous pouvez voir les valeurs au fur et à mesure qu'elles sont récupérées. Ce programme utilise un exemple d'image statique afin que les zones ne changent pas, mais vous pouvez imaginer qu'avec une caméra sur votre robot, les valeurs changeraient constamment.

29

Planification de trajectoire

La planification de trajectoires est le processus de création et de suivi des trajectoires. Ces trajectoires utilisent les API de trajectoire de la librairie WPILib pour la génération et un :ref :` contrôleur Ramsete <docs/software/advanced-controls/trajectories/ramsete :Ramsete Controller>` pour le suivi. Cette section met l'emphase sur le processus de caractérisation de votre robot pour l'identification du système, le suivi des trajectoires et l'utilisation de PathWeaver. Les utilisateurs peuvent également lire les documents suivants des *documents génériques sur le suivi de trajectoires* pour plus d'informations sur l'API et une utilisation non orientée commande.

29.1 Note sur le support relatif au Swerve

Swerve support in path following has a couple of limitations that teams need to be aware of :

- WPILib currently does not support swerve in simulation, please see this pull request.
- SysId only supports tuning the swerve heading using a General Mechanism project and does not regularly support module velocity data. A workaround is to lock the module's heading into place. This can be done via blocking module rotation using something like a block of wood.
- Pathweaver and Trajectory following currently do not incorporate independent heading. Path following using the WPILib trajectory framework on swerve will be the same as a DifferentialDrive robot.

Nous sommes désolés pour l'inconvénient causé.

29.1.1 Didacticiel sur la pratique des trajectoires

Il s'agit d'un tutoriel complet portant sur la mise en application de la génération et du suivi de trajectoires sur un robot à entraînement différentiel. Le code complet utilisé dans ce didacticiel peut être trouvé dans l'exempe de projet RamseteCommand (Java, C++).

Aperçu du didacticiel de trajectoire

Note : Avant de suivre ce tutoriel, il est utile (mais pas strictement nécessaire) d'avoir une familiarité de base avec les fonctionnalités de la WPILib *PID control*, *feedforward*, et *trajectory*.

Note : Le code robot de ce didacticiel utilise l'environnement de développement orienté Commandes *command-based*. Cet environnement de développement est fortement recommandé pour les équipes de recrues et intermédiaires.

L'objectif de ce didacticiel est de fournir des instructions de bout en bout sur la mise en œuvre d'une routine autonome de suivi de trajectoire pour un robot à entraînement différentiel. En suivant ce didacticiel, les lecteurs apprendront à :

1. Caractériser avec précision la base pilotable de leur robot pour obtenir des calculs précis pour la commande prédictive et des gains approximatifs pour les boucles de rétroaction .
2. Configurer un sous-système de déplacement de manière à suivre la pose du robot à l'aide de la librairie odometry de WPILib.
3. Générer une trajectoire simple à travers un ensemble de points de passage à l'aide de la classe `TrajectoryGenerator` de WPILib.
4. Suivez la trajectoire générée dans une routine autonome à l'aide de la classe `RamseteCommand` de WPILib avec les gains de commande prédictive/rétroaction calculés et la pose.

Ce didacticiel est destiné à être accessible aux équipes n'ayant pas beaucoup d'expertise en programmation. Bien que la logithèque WPILib offre une flexibilité significative dans la manière dont ses fonctionnalités de suivi de trajectoire sont implémentées, suivre de près l'implémentation décrite dans ce didacticiel devrait fournir aux équipes une solution relativement simple, propre et reproductible pour les mouvements autonomes.

Le code robot complet pour ce tutoriel peut être trouvé dans le Projet d'exemple RamseteCommand (Java, C++).

Pourquoi le suvi de trajectoire ?

Les jeux FRC® comportent souvent des tâches autonomes qui exigent qu'un robot se déplace efficacement et avec précision d'un emplacement de départ connu à une position de tir connue. Historiquement, la solution la plus courante pour ce genre de tâche dans FRC a été une approche « avance-tourne-avance » - c'est-à-dire, avancer d'une distance connue, tourner d'un angle connu, et avancer d'un autre distance connue.

Bien que l'approche « avance-tourne-avance » soit certainement fonctionnelle, ces dernières années, les équipes ont commencé à suivre les trajectoires lisses qui exigent que le robot avance et tourne en même temps. Bien qu'il s'agisse, sur le plan technique, d'une tâche fondamentalement plus compliquée, elle offre néamoins des avantages significatifs : en particulier, puisque le robot n'a plus à s'arrêter pour changer de direction, les tronçons peuvent être conduits beaucoup plus rapidement, permettant ainsi à un robot de marquer plus de pièces de jeu pendant la période autonome.

À partir de 2020, WPILib fournit désormais aux équipes des solutions sous forme de codes fonctionnels et avancés pour la génération et le suivi des trajectoires, abaissant considérablement le « barrière à l'entrée » pour ce type de déplacement autonome à la fois avancé et efficace.

Équipement requis

Pour suivre ce didacticiel, vous aurez besoin d'un accès facile aux documents suivants :

1. A differential-drive robot (such as the AndyMark AM14U5), equipped with :

— Encodeurs en quadrature pour mesurer la rotation des roues de chaque côté de l'entraînement.
— Gyroscope pour mesurer l'orientation du robot.

2. Un ordinateur de pilotage dans lequel sont installés outils logiciels suivants :

— *FRC Driver Station*.
— *WPILib*.
— *The System Identification Toolsuite*.

Étape 1 : Caractérisation de l'entraînement de votre robot

Note : For detailed instructions on using the System Identification tool, see its *dedicated documentation*.

Note : Le processus d'identification du lecteur nécessite suffisamment d'espace pour que le robot puisse conduire. Assurez-vous d'avoir *au moins* un espace de 10" (idéalement plus proche de 20") dans lequel le robot peut conduire pendant la routine d'identification.

Note : Les données d'identification de ce didacticiel ont été généreusement fournies par l'équipe 5190, qui les a générées dans le cadre d'une démonstration de cette fonctionnalité lors de l'atelier P2P 2019 de la North Carolina State University.

Avant de suivre avec précision une trajectoire avec un robot, il est important d'avoir un modèle précis de la façon dont le robot se déplace en réponse à ses entrées de commande. La détermination d'un tel modèle est un processus appelé « identification du système ». L'outil d'identification du système de WPILib peut déterminer avec précision un tel modèle.

Collecte des données

Nous commençons par rassembler nos données d'identification de disque.

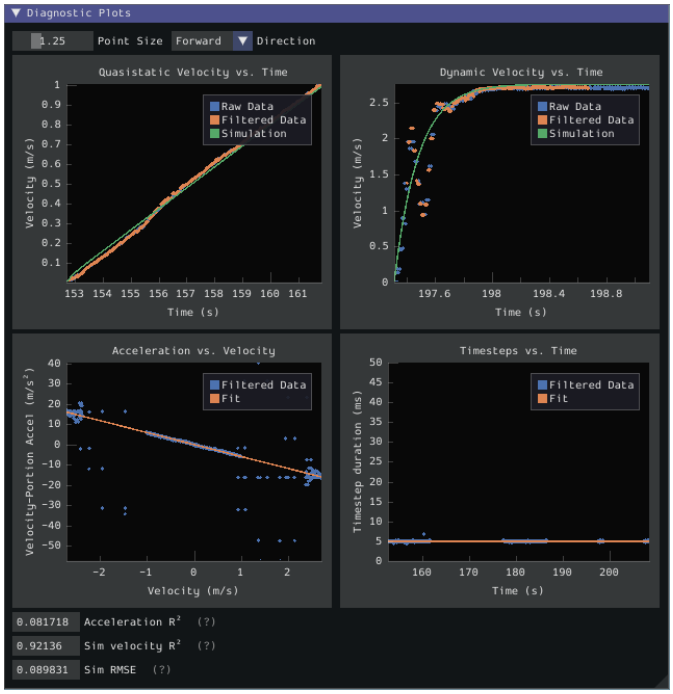
1. *Configure and Deploy your robot project*.
2. *Run the identification Routine*.

Analyse des données

Once the identification routine has been run and the data file has been saved, it is time to *open it in the analysis pane*.

Vérification des diagnostics

Per the *system identification guide*, we first view the diagnostics to ensure that our data look reasonable :

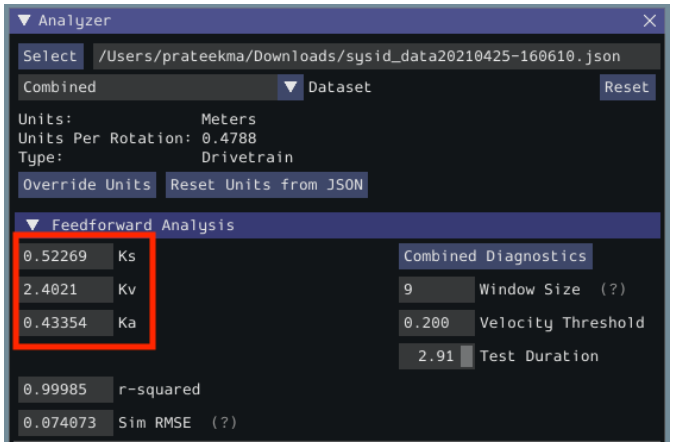


Comme nos données semblent raisonnablement linéaires et que les mesures d'ajustement se situent dans des paramètres acceptables, nous passons à l'étape suivante.

Enregistrer les gains de commande Feedforward

Note : Les gains de commande prédictive ou Feedforward ne se transfèrent *pas*, en général, entre les robots. Aussi, n'utilisez *pas* les gains de ce tutoriel pour votre propre robot.

Nous enregistrons maintenant les gains de commande prédictive calculés par l'outil :



Puisque notre diamètre de roue a été spécifié en mètres, nos gains de prédiction sont dans les unités suivantes :

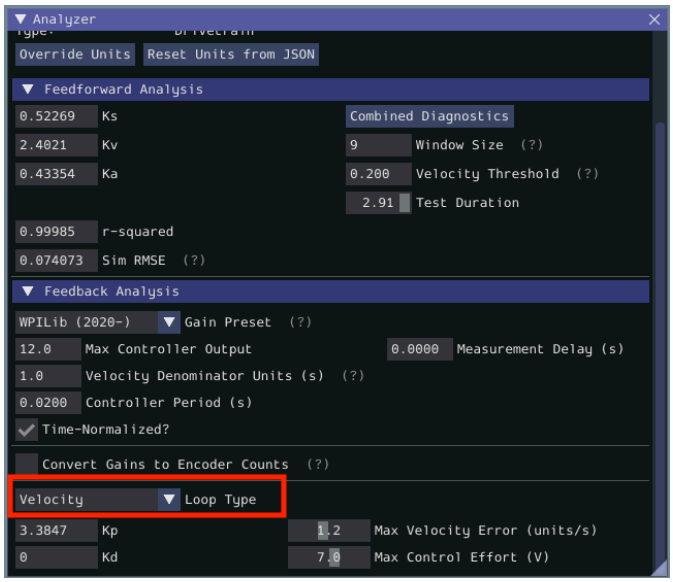
- kS : Volts
- kV : Volts * Seconds / Meters
- kA : Volts * Seconds^2 / Meters

Si vous avez correctement spécifié vos unités, vos gains de commande prédictive seront probablement dans un ordre de grandeur de ceux du présent tutoriel (une exception possible existe pour kA, qui peut être beaucoup plus petite si votre robot est léger). Si ce n'est pas le cas, il est possible que vous ayiez incorrectement spécifié un de vos paramètres du système d'entraînement lors de la génération de votre projet de robot. Pour cela, un bon test est de calculer la valeur « théorique » de kV, qui est de 12 volts divisé par la vitesse libre théorique de votre base pilotable (qui est, à son tour, la vitesse libre du moteur multipliée par la circonférence de la roue divisée par le rapport d'engrenage). Cette valeur devrait être très proche du kV mesuré par l'outil - si ce n'est pas le cas, vous avez probablement fait une erreur quelque part.

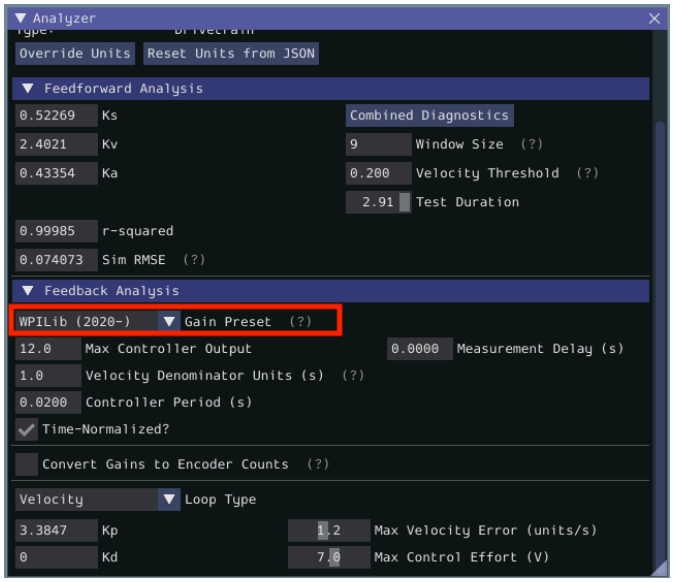
Calculer les gains de rétroaction

Note : Les gains de rétroaction ne se transfèrent *pas*, en général, entre les robots. Aussi, n'utilisez pas les gains de ce tutoriel pour votre propre robot.

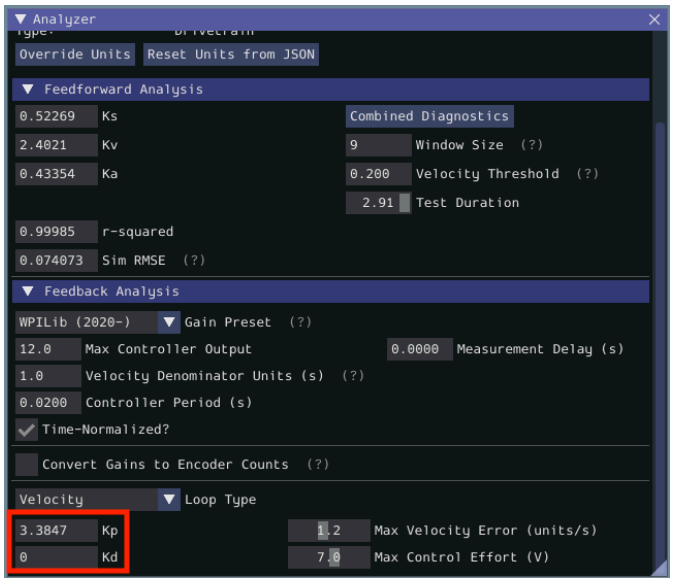
We now *calculate the feedback gains* for the PID control that we will use to follow the path. Trajectory following with WPILib's RAMSETE controller uses velocity closed-loop control, so we first select `Velocity` mode in the identification tool :



Puisque nous utiliserons lePIDController WPILib pour notre boucle de vitesse, nous sélectionnons en outre l'option `WPILib (2020-)` dans le menu déroulant « presets ». Ceci est *très* important, car les gains de rétroaction ne seront pas dans les unités concordantes si nous ne sélectionnons pas le bon préréglage :



Enfin, nous calculons et enregistrons les gains de rétroaction pour notre boucle de contrôle. Puisqu'il s'agit d'un contrôleur de vitesse, seul un gain P est requis :



En supposant que nous avons tout fait correctement, notre gain proportionnel sera en unités de Volts * Secondes / Mètres. Ainsi, notre gain calculé signifie que, pour chaque mètre par seconde d'erreur de vitesse, le contrôleur produira 3.38 volts supplémentaires.

Étape 2 : Saisie des constantes calculées

Note : En C++, il est important que les constantes de la commande prédictive soient saisies dans le type d'unité approprié. Pour plus d'informations sur les unités C++, voir *La librairie d'unités C++*.

Maintenant que nous avons nos constantes système, il est temps de les placer dans notre code. L'emplacement recommandé pour cela est le fichier `Constants` de la *structure de projet basée sur des commandes standard*.

Les parties pertinentes du fichier des constantes de l'exemple de projet RamseteCommand (Java, C++) peuvent être vue ci-dessous.

Gains Prédicitif/Rétroactif

Tout d'abord, nous devons entrer les gains d'anticipation et de rétroaction que nous avons obtenus à partir de l'outil d'identification.

Note : Les gains prédictif et rétroactif ne se transfèrent *pas*, en général, entre les robots. Aussi, n'utilisez pas les gains de ce tutoriel sur votre propre robot.

JAVA

```
    // These are example values only - DO NOT USE THESE FOR YOUR OWN ROBOT!
    // These characterization values MUST be determined either experimentally or theoretically
    // for *your* robot's drive.
    // The Robot Characterization Toolsuite provides a convenient tool for obtaining these
    // values for your robot.
    public static final double ksVolts = 0.22;
    public static final double kvVoltSecondsPerMeter = 1.98;
    public static final double kaVoltSecondsSquaredPerMeter = 0.2;

    // Example value only - as above, this must be tuned for your drive!
    public static final double kPDriveVel = 8.5;
```

C++

```
// These are example values only - DO NOT USE THESE FOR YOUR OWN ROBOT!
// These characterization values MUST be determined either experimentally or
// theoretically for *your* robot's drive. The Robot Characterization
// Toolsuite provides a convenient tool for obtaining these values for your
// robot.
inline constexpr auto ks = 0.22_V;
inline constexpr auto kv = 1.98 * 1_V * 1_s / 1_m;
inline constexpr auto ka = 0.2 * 1_V * 1_s * 1_s / 1_m;

// Example value only - as above, this must be tuned for your drive!
inline constexpr double kPDriveVel = 8.5;
```

DifferentialDriveKinematics

En outre, nous devons créer une instance de la classe `DifferentialDriveKinematics` , qui nous permet d'utiliser la *trackwidth* (c'est-à-dire la distance horizontale entre les roues) du robot pour la conversion des vitesses du châssis aux vitesses des roues. Comme partout ailleurs, nous gardons nos unités en mètres.

JAVA

```
    public static final double kTrackwidthMeters = 0.69;
    public static final DifferentialDriveKinematics kDriveKinematics =
        new DifferentialDriveKinematics(kTrackwidthMeters);
```

C++

```
inline constexpr auto kTrackwidth = 0.69_m;
extern const frc::DifferentialDriveKinematics kDriveKinematics;
```

Vitesse/accélération max sur la trajectoire

Nous devons également décider d'une accélération nominale max et d'une vitesse maximale du robot pendant le suivi de la trajectoire. La valeur maximale de la vitesse doit être choisie légèrement en dessous de la vitesse libre nominale du robot. En raison de l'utilisation ultérieure du `DifferentialDriveVoltageConstraint`, la valeur maximale de l'accélération n'est pas extrêmement cruciale.

Avertissement : Max velocity and acceleration, as defined here, are applied only during trajectory generation. They do not limit the `RamseteCommand` itself, which may give values to the `DriveSubsystem` that can cause the robot to greatly exceed these velocities and accelerations.

JAVA

```
    public static final double kMaxSpeedMetersPerSecond = 3;
    public static final double kMaxAccelerationMetersPerSecondSquared = 1;
```

C++

```
inline constexpr auto kMaxSpeed = 3_mps;
inline constexpr auto kMaxAcceleration = 1_mps_sq;
```

Paramètres Ramsete

Enfin, nous devons inclure une paire de paramètres pour le contrôleur RAMSETE. Les valeurs indiquées ci-dessous devraient bien fonctionner pour la plupart des robots, à condition que les distances aient été correctement mesurées en mètres - pour plus d'informations sur le réglage de ces valeurs (si nécessaire), voir *Construire l'objet contrôleur Ramsete*.

JAVA

```
    // Reasonable baseline values for a RAMSETE follower in units of meters and␣
↪seconds
    public static final double kRamseteB = 2;
    public static final double kRamseteZeta = 0.7;
```

C++

```
// Reasonable baseline values for a RAMSETE follower in units of meters and
// seconds
inline constexpr auto kRamseteB = 2.0 * 1_rad * 1_rad / (1_m * 1_m);
inline constexpr auto kRamseteZeta = 0.7 / 1_rad;
```

Étape 3 : Création d'un sous-système de déplacement

Maintenant que la base pilotable est caractérisée, c'est le moment de commencer à écrire notre code robot *proprement dit*. Comme mentionné précédemment, nous utiliserons le cadre de développement *command-based*, pour ce faire. Aussi, notre première étape consiste à créer une classe sous-système de déplacement appropriée *subsystem*.

La classe de déplacement complète du projet d'exemple RamseteCommand (Java, C++) peut être vu ci-dessous. Le reste de l'article décrira les étapes nécessaires dans l'écriture de cette classe.

Java

```
package edu.wpi.first.wpilibj.examples.ramsetecommand.subsystems;

import edu.wpi.first.math.geometry.Pose2d;
import edu.wpi.first.math.kinematics.DifferentialDriveOdometry;
import edu.wpi.first.math.kinematics.DifferentialDriveWheelSpeeds;
import edu.wpi.first.util.sendable.SendableRegistry;
import edu.wpi.first.wpilibj.ADXRS450_Gyro;
import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import edu.wpi.first.wpilibj.examples.ramsetecommand.Constants.DriveConstants;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
import edu.wpi.first.wpilibj2.command.SubsystemBase;

public class DriveSubsystem extends SubsystemBase {
  // The motors on the left side of the drive.
  private final PWMSparkMax m_leftLeader = new PWMSparkMax(DriveConstants.
↪kLeftMotor1Port);
  private final PWMSparkMax m_leftFollower = new PWMSparkMax(DriveConstants.
↪kLeftMotor2Port);

  // The motors on the right side of the drive.
  private final PWMSparkMax m_rightLeader = new PWMSparkMax(DriveConstants.
↪kRightMotor1Port);
  private final PWMSparkMax m_rightFollower = new PWMSparkMax(DriveConstants.
↪kRightMotor2Port);

  // The robot's drive
  private final DifferentialDrive m_drive =
      new DifferentialDrive(m_leftLeader::set, m_rightLeader::set);

  // The left-side drive encoder
  private final Encoder m_leftEncoder =
      new Encoder(
```

(suite sur la page suivante)

(suite de la page précédente)

```
          DriveConstants.kLeftEncoderPorts[0],
          DriveConstants.kLeftEncoderPorts[1],
          DriveConstants.kLeftEncoderReversed);

  // The right-side drive encoder
  private final Encoder m_rightEncoder =
      new Encoder(
          DriveConstants.kRightEncoderPorts[0],
          DriveConstants.kRightEncoderPorts[1],
          DriveConstants.kRightEncoderReversed);

  // The gyro sensor
  private final ADXRS450_Gyro m_gyro = new ADXRS450_Gyro();

  // Odometry class for tracking robot pose
  private final DifferentialDriveOdometry m_odometry;

  /** Creates a new DriveSubsystem. */
  public DriveSubsystem() {
    SendableRegistry.addChild(m_drive, m_leftLeader);
    SendableRegistry.addChild(m_drive, m_rightLeader);

    m_leftLeader.addFollower(m_leftFollower);
    m_rightLeader.addFollower(m_rightFollower);

    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side instead.
    m_rightLeader.setInverted(true);

    // Sets the distance per pulse for the encoders
    m_leftEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
    m_rightEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);

    resetEncoders();
    m_odometry =
        new DifferentialDriveOdometry(
            m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
↪getDistance());
  }

  @Override
  public void periodic() {
    // Update the odometry in the periodic block
    m_odometry.update(
        m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
↪getDistance());
  }

  /**
   * Returns the currently-estimated pose of the robot.
   *
   * @return The pose.
   */
  public Pose2d getPose() {
    return m_odometry.getPoseMeters();
```

(suite sur la page suivante)

(suite de la page précédente)

```
  }

  /**
   * Returns the current wheel speeds of the robot.
   *
   * @return The current wheel speeds.
   */
  public DifferentialDriveWheelSpeeds getWheelSpeeds() {
    return new DifferentialDriveWheelSpeeds(m_leftEncoder.getRate(), m_rightEncoder.
↪getRate());
  }

  /**
   * Resets the odometry to the specified pose.
   *
   * @param pose The pose to which to set the odometry.
   */
  public void resetOdometry(Pose2d pose) {
    m_odometry.resetPosition(
        m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
↪getDistance(), pose);
  }

  /**
   * Drives the robot using arcade controls.
   *
   * @param fwd the commanded forward movement
   * @param rot the commanded rotation
   */
  public void arcadeDrive(double fwd, double rot) {
    m_drive.arcadeDrive(fwd, rot);
  }

  /**
   * Controls the left and right sides of the drive directly with voltages.
   *
   * @param leftVolts the commanded left output
   * @param rightVolts the commanded right output
   */
  public void tankDriveVolts(double leftVolts, double rightVolts) {
    m_leftLeader.setVoltage(leftVolts);
    m_rightLeader.setVoltage(rightVolts);
    m_drive.feed();
  }

  /** Resets the drive encoders to currently read a position of 0. */
  public void resetEncoders() {
    m_leftEncoder.reset();
    m_rightEncoder.reset();
  }

  /**
   * Gets the average distance of the two encoders.
   *
   * @return the average of the two encoder readings
   */
```

(suite sur la page suivante)

(suite de la page précédente)

```
  public double getAverageEncoderDistance() {
    return (m_leftEncoder.getDistance() + m_rightEncoder.getDistance()) / 2.0;
  }

  /**
   * Gets the left drive encoder.
   *
   * @return the left drive encoder
   */
  public Encoder getLeftEncoder() {
    return m_leftEncoder;
  }

  /**
   * Gets the right drive encoder.
   *
   * @return the right drive encoder
   */
  public Encoder getRightEncoder() {
    return m_rightEncoder;
  }

  /**
   * Sets the max output of the drive. Useful for scaling the drive to drive more
  ↪slowly.
   *
   * @param maxOutput the maximum output to which the drive will be constrained
   */
  public void setMaxOutput(double maxOutput) {
    m_drive.setMaxOutput(maxOutput);
  }

  /** Zeroes the heading of the robot. */
  public void zeroHeading() {
    m_gyro.reset();
  }

  /**
   * Returns the heading of the robot.
   *
   * @return the robot's heading in degrees, from -180 to 180
   */
  public double getHeading() {
    return m_gyro.getRotation2d().getDegrees();
  }

  /**
   * Returns the turn rate of the robot.
   *
   * @return The turn rate of the robot, in degrees per second
   */
  public double getTurnRate() {
    return -m_gyro.getRate();
  }
}
```

C++ (Header)

```
#pragma once

#include <frc/ADXRS450_Gyro.h>
#include <frc/Encoder.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/geometry/Pose2d.h>
#include <frc/kinematics/DifferentialDriveOdometry.h>
#include <frc/motorcontrol/PWMSparkMax.h>
#include <frc2/command/SubsystemBase.h>
#include <units/voltage.h>

#include "Constants.h"

class DriveSubsystem : public frc2::SubsystemBase {
 public:
  DriveSubsystem();

  /**
   * Will be called periodically whenever the CommandScheduler runs.
   */
  void Periodic() override;

  // Subsystem methods go here.

  /**
   * Drives the robot using arcade controls.
   *
   * @param fwd the commanded forward movement
   * @param rot the commanded rotation
   */
  void ArcadeDrive(double fwd, double rot);

  /**
   * Controls each side of the drive directly with a voltage.
   *
   * @param left the commanded left output
   * @param right the commanded right output
   */
  void TankDriveVolts(units::volt_t left, units::volt_t right);

  /**
   * Resets the drive encoders to currently read a position of 0.
   */
  void ResetEncoders();

  /**
   * Gets the average distance of the TWO encoders.
   *
   * @return the average of the TWO encoder readings
   */
  double GetAverageEncoderDistance();

  /**
   * Gets the left drive encoder.
   *
```

(suite sur la page suivante)

(suite de la page précédente)

```
   * @return the left drive encoder
   */
  frc::Encoder& GetLeftEncoder();

  /**
   * Gets the right drive encoder.
   *
   * @return the right drive encoder
   */
  frc::Encoder& GetRightEncoder();

  /**
   * Sets the max output of the drive.  Useful for scaling the drive to drive
   * more slowly.
   *
   * @param maxOutput the maximum output to which the drive will be constrained
   */
  void SetMaxOutput(double maxOutput);

  /**
   * Returns the heading of the robot.
   *
   * @return the robot's heading in degrees, from -180 to 180
   */
  units::degree_t GetHeading() const;

  /**
   * Returns the turn rate of the robot.
   *
   * @return The turn rate of the robot, in degrees per second
   */
  double GetTurnRate();

  /**
   * Returns the currently-estimated pose of the robot.
   *
   * @return The pose.
   */
  frc::Pose2d GetPose();

  /**
   * Returns the current wheel speeds of the robot.
   *
   * @return The current wheel speeds.
   */
  frc::DifferentialDriveWheelSpeeds GetWheelSpeeds();

  /**
   * Resets the odometry to the specified pose.
   *
   * @param pose The pose to which to set the odometry.
   */
  void ResetOdometry(frc::Pose2d pose);

 private:
  // Components (e.g. motor controllers and sensors) should generally be
```

(suite sur la page suivante)

(suite de la page précédente)

```
  // declared private and exposed only through public methods.

  // The motor controllers
  frc::PWMSparkMax m_left1;
  frc::PWMSparkMax m_left2;
  frc::PWMSparkMax m_right1;
  frc::PWMSparkMax m_right2;

  // The robot's drive
  frc::DifferentialDrive m_drive{[&](double output) { m_left1.Set(output); },
                                 [&](double output) { m_right1.Set(output); }};

  // The left-side drive encoder
  frc::Encoder m_leftEncoder;

  // The right-side drive encoder
  frc::Encoder m_rightEncoder;

  // The gyro sensor
  frc::ADXRS450_Gyro m_gyro;

  // Odometry class for tracking robot pose
  frc::DifferentialDriveOdometry m_odometry;
};
```

C++ (Source)

```
#include "subsystems/DriveSubsystem.h"

#include <frc/geometry/Rotation2d.h>
#include <frc/kinematics/DifferentialDriveWheelSpeeds.h>

using namespace DriveConstants;

DriveSubsystem::DriveSubsystem()
    : m_left1{kLeftMotor1Port},
      m_left2{kLeftMotor2Port},
      m_right1{kRightMotor1Port},
      m_right2{kRightMotor2Port},
      m_leftEncoder{kLeftEncoderPorts[0], kLeftEncoderPorts[1]},
      m_rightEncoder{kRightEncoderPorts[0], kRightEncoderPorts[1]},
      m_odometry{m_gyro.GetRotation2d(), units::meter_t{0}, units::meter_t{0}} {
  wpi::SendableRegistry::AddChild(&m_drive, &m_left1);
  wpi::SendableRegistry::AddChild(&m_drive, &m_right1);

  m_left1.AddFollower(m_left2);
  m_right1.AddFollower(m_right2);

  // We need to invert one side of the drivetrain so that positive voltages
  // result in both sides moving forward. Depending on how your robot's
  // gearbox is constructed, you might have to invert the left side instead.
  m_right1.SetInverted(true);

  // Set the distance per pulse for the encoders
```

(suite sur la page suivante)

(suite de la page précédente)

```
  m_leftEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
  m_rightEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());

  ResetEncoders();
}

void DriveSubsystem::Periodic() {
  // Implementation of subsystem periodic method goes here.
  m_odometry.Update(m_gyro.GetRotation2d(),
                    units::meter_t{m_leftEncoder.GetDistance()},
                    units::meter_t{m_rightEncoder.GetDistance()});
}

void DriveSubsystem::ArcadeDrive(double fwd, double rot) {
  m_drive.ArcadeDrive(fwd, rot);
}

void DriveSubsystem::TankDriveVolts(units::volt_t left, units::volt_t right) {
  m_left1.SetVoltage(left);
  m_right1.SetVoltage(right);
  m_drive.Feed();
}

void DriveSubsystem::ResetEncoders() {
  m_leftEncoder.Reset();
  m_rightEncoder.Reset();
}

double DriveSubsystem::GetAverageEncoderDistance() {
  return (m_leftEncoder.GetDistance() + m_rightEncoder.GetDistance()) / 2.0;
}

frc::Encoder& DriveSubsystem::GetLeftEncoder() {
  return m_leftEncoder;
}

frc::Encoder& DriveSubsystem::GetRightEncoder() {
  return m_rightEncoder;
}

void DriveSubsystem::SetMaxOutput(double maxOutput) {
  m_drive.SetMaxOutput(maxOutput);
}

units::degree_t DriveSubsystem::GetHeading() const {
  return m_gyro.GetRotation2d().Degrees();
}

double DriveSubsystem::GetTurnRate() {
  return -m_gyro.GetRate();
}

frc::Pose2d DriveSubsystem::GetPose() {
  return m_odometry.GetPose();
}

```

(suite sur la page suivante)

(suite de la page précédente)

```
frc::DifferentialDriveWheelSpeeds DriveSubsystem::GetWheelSpeeds() {
  return {units::meters_per_second_t{m_leftEncoder.GetRate()},
          units::meters_per_second_t{m_rightEncoder.GetRate()}};
}

void DriveSubsystem::ResetOdometry(frc::Pose2d pose) {
  m_odometry.ResetPosition(m_gyro.GetRotation2d(),
                           units::meter_t{m_leftEncoder.GetDistance()},
                           units::meter_t{m_rightEncoder.GetDistance()}, pose);
}
```

Configuration des encodeurs du sous-système de déplacement

Les encodeurs du sous-système de déplacement mesurent la rotation des roues de chaque côté de la base pilotable. Pour configurer correctement ces encodeurs, nous devons spécifier deux choses : les ports dans lesquels les encodeurs sont branchés et la distance par impulsion d'encodeur. Ensuite, nous devons écrire des méthodes permettant d'obtention de valeurs d'encodeur à partir du code qui utilise le sous-système.

Ports des encodeurs

Les ports d'encodeur sont spécifiés dans le constructeur de l'encodeur de la manière suivante :

Java

```
  // The left-side drive encoder
  private final Encoder m_leftEncoder =
      new Encoder(
          DriveConstants.kLeftEncoderPorts[0],
          DriveConstants.kLeftEncoderPorts[1],
          DriveConstants.kLeftEncoderReversed);

  // The right-side drive encoder
  private final Encoder m_rightEncoder =
      new Encoder(
          DriveConstants.kRightEncoderPorts[0],
          DriveConstants.kRightEncoderPorts[1],
          DriveConstants.kRightEncoderReversed);
```

C++ (Source)

```
      m_leftEncoder{kLeftEncoderPorts[0], kLeftEncoderPorts[1]},
      m_rightEncoder{kRightEncoderPorts[0], kRightEncoderPorts[1]},
```

Distance par impulsion d'encodeur

La distance par impulsion est spécifiée en appelant la méthode `setDistancePerPulse` de l'encodeur. Notez que pour la classe WPILib Encoder, « impulsion » fait référence à un cycle d'encodeur complet (c'est-à-dire quatre fronts), et sera donc 1/4 de la valeur qui a été spécifiée dans la configuration SysId. N'oubliez pas non plus que la distance doit être mesurée en mètres !

Java

```
    m_leftEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
    m_rightEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
```

C++ (Source)

```
  m_leftEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
  m_rightEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
```

Méthode d'obtention d'encodeur

Pour accéder aux valeurs mesurées par les encodeurs, nous incluons la méthode suivante :

Important : Les vitesses renvoyées ** doivent ** être en mètres ! Parce que nous avons configuré la distance par impulsion sur les encodeurs ci-dessus, l'appel de `getRate()` appliquera automatiquement le facteur de conversion des unités d'encodeur aux mètres. Si vous n'utilisez pas la classe `Encoder` de WPILib, vous devez effectuer cette conversion soit via l'API du fournisseur respectif, soit en multipliant manuellement par un facteur de conversion.

Java

```
  /**
   * Returns the current wheel speeds of the robot.
   *
   * @return The current wheel speeds.
   */
  public DifferentialDriveWheelSpeeds getWheelSpeeds() {
    return new DifferentialDriveWheelSpeeds(m_leftEncoder.getRate(), m_rightEncoder.
↪getRate());
  }
```

C++ (Source)

```
frc::DifferentialDriveWheelSpeeds DriveSubsystem::GetWheelSpeeds() {
  return {units::meters_per_second_t{m_leftEncoder.GetRate()},
          units::meters_per_second_t{m_rightEncoder.GetRate()}};
}
```

Nous enveloppons les valeurs d'encodeur mesurées dans un objet `DifferentialDriveWheelSpeeds` pour faciliter l'intégration, plus tard, avec la classe `RamseteCommand`.

Configuration du Gyroscope

The gyroscope measures the rate of change of the robot's heading (which can then be integrated to provide a measurement of the robot's heading relative to when it first turned on). In our example, we use the Analog Devices ADXRS450 FRC Gyro Board, which was included in the kit of parts for several years :

Java

```
  // The gyro sensor
  private final ADXRS450_Gyro m_gyro = new ADXRS450_Gyro();
```

C++ (Header)

```
  // The gyro sensor
  frc::ADXRS450_Gyro m_gyro;
```

Méthode d'obtention du gyroscope

Pour accéder à l'orientation actuelle mesurée par le gyroscope, nous incluons la méthode suivante :

Java

```
  /**
   * Returns the heading of the robot.
   *
   * @return the robot's heading in degrees, from -180 to 180
   */
  public double getHeading() {
    return m_gyro.getRotation2d().getDegrees();
  }
```

C++ (Source)

```
76 units::degree_t DriveSubsystem::GetHeading() const {
77   return m_gyro.GetRotation2d().Degrees();
78 }
```

Configuration de l'odométrie

Maintenant que nous avons nos encodeurs et gyroscope configurés, nous pouvons mettre en place notre sous-système de déplacement pour calculer automatiquement sa position à partir des lectures obtenues des encodeurs et de gyroscope.

Tout d'abord, nous créons une instance membre de la classe `DifferentialDriveOdometry` :

Java

```
48   // Odometry class for tracking robot pose
49   private final DifferentialDriveOdometry m_odometry;
```

C++ (Header)

```
137   // Odometry class for tracking robot pose
138   frc::DifferentialDriveOdometry m_odometry;
```

Then we initialize the `DifferentialDriveOdometry`.

Java

```
69     m_odometry =
70         new DifferentialDriveOdometry(
71             m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
↪getDistance());
```

C++ (Source)

```
19       m_odometry{m_gyro.GetRotation2d(), units::meter_t{0}, units::meter_t{0}} {
```

Mise à jour de l'odométrie

La classe d'odométrie doit être régulièrement mise à jour pour incorporer de nouvelles lectures de l'encodeur et du gyroscope. Nous accomplissons cela à l'intérieur de la méthode `periodic` du sous-système, qui est automatiquement appelée une fois à chaque itération de boucle principale :

Java

```
  @Override
  public void periodic() {
    // Update the odometry in the periodic block
    m_odometry.update(
        m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
↪getDistance());
  }
```

C++ (Source)

```
void DriveSubsystem::Periodic() {
  // Implementation of subsystem periodic method goes here.
  m_odometry.Update(m_gyro.GetRotation2d(),
                    units::meter_t{m_leftEncoder.GetDistance()},
                    units::meter_t{m_rightEncoder.GetDistance()});
}
```

Méthode d'obtention de l'odométrie

Pour accéder à la pose actuelle calculée du robot, nous incluons la méthode suivante :

Java

```
  /**
   * Returns the currently-estimated pose of the robot.
   *
   * @return The pose.
   */
  public Pose2d getPose() {
    return m_odometry.getPoseMeters();
  }
```

C++ (Source)

```
frc::Pose2d DriveSubsystem::GetPose() {
  return m_odometry.GetPose();
}
```

Important : Before running a `RamseteCommand`, teams are strongly encouraged to deploy and test the odometry code alone, with values sent to the SmartDashboard or Shuffleboard during the `DriveSubsystem`'s `periodic()`. This odometry must be correct for a RamseteCommand to successfully work, as sign or unit errors can cause a robot to move at high speeds in unpredictable directions.

Méthode du sous-système de déplacement en charge de la tension

Enfin, nous devons inclure une méthode supplémentaire - une méthode qui nous permet de régler la tension de chaque côté de la base pilotable en utilisant la méthode `setVoltage()` de l'interface `MotorController`. La classe du système d'entraînement WPILib par défaut n'inclut pas cette fonctionnalité, nous devons donc l'écrire nous-mêmes :

Java

```
  /**
   * Controls the left and right sides of the drive directly with voltages.
   *
   * @param leftVolts the commanded left output
   * @param rightVolts the commanded right output
   */
  public void tankDriveVolts(double leftVolts, double rightVolts) {
    m_leftLeader.setVoltage(leftVolts);
    m_rightLeader.setVoltage(rightVolts);
    m_drive.feed();
  }
```

C++ (Source)

```
void DriveSubsystem::TankDriveVolts(units::volt_t left, units::volt_t right) {
  m_left1.SetVoltage(left);
  m_right1.SetVoltage(right);
  m_drive.Feed();
}
```

Il est très important d'utiliser la méthode `setVoltage()` plutôt que la méthode ordinaire `set()`, car cela compensera automatiquement la « baisse de tension » de la batterie pendant le fonctionnement. Étant donné que nos tensions d'anticipation ont une signification physique (car elles sont basées sur des données d'identification mesurées), cela est essentiel pour garantir leur exactitude.

Avertissement : `RamseteCommand` itself does not internally enforce any speed or acceleration limits before providing motor voltage parameters to this method. During initial code development, teams are strongly encouraged to apply both maximum and minimum bounds on the input variables before passing these values to `setVoltage()` while ensuring the trajectory velocity and acceleration are achievable. For example, generate a trajectory with a little less than half of the Robot's maximum velocity and limit voltage to 6 volts.

Étape 4 : Création et suivi d'une trajectoire

Avec notre sous-système de déplacement écrit, il est maintenant temps de générer une trajectoire et d'écrire une commande autonome pour la suivre.

Selon le projet *structure standard d'un projet orienté commande*, nous le ferons dans la méthode `getAutonomousCommand` de la classe `RobotContainer`. La méthode complète du projet d'exemple RamseteCommand (Java, C++) peut être vue ci-dessous. Le reste de l'article décomposera plus en détail les différentes parties de la méthode.

Java

```
74  /**
75   * Use this to pass the autonomous command to the main {@link Robot} class.
76   *
77   * @return the command to run in autonomous
78   */
79  public Command getAutonomousCommand() {
80    // Create a voltage constraint to ensure we don't accelerate too fast
81    var autoVoltageConstraint =
82        new DifferentialDriveVoltageConstraint(
83            new SimpleMotorFeedforward(
84                DriveConstants.ksVolts,
85                DriveConstants.kvVoltSecondsPerMeter,
86                DriveConstants.kaVoltSecondsSquaredPerMeter),
87            DriveConstants.kDriveKinematics,
88            10);
89
90    // Create config for trajectory
91    TrajectoryConfig config =
92        new TrajectoryConfig(
93                AutoConstants.kMaxSpeedMetersPerSecond,
94                AutoConstants.kMaxAccelerationMetersPerSecondSquared)
95            // Add kinematics to ensure max speed is actually obeyed
96            .setKinematics(DriveConstants.kDriveKinematics)
97            // Apply the voltage constraint
98            .addConstraint(autoVoltageConstraint);
99
100   // An example trajectory to follow. All units in meters.
101   Trajectory exampleTrajectory =
102       TrajectoryGenerator.generateTrajectory(
103           // Start at the origin facing the +X direction
104           new Pose2d(0, 0, new Rotation2d(0)),
105           // Pass through these two interior waypoints, making an 's' curve path
106           List.of(new Translation2d(1, 1), new Translation2d(2, -1)),
107           // End 3 meters straight ahead of where we started, facing forward
```

(suite sur la page suivante)

(suite de la page précédente)

```
            new Pose2d(3, 0, new Rotation2d(0)),
            // Pass config
            config);

    RamseteCommand ramseteCommand =
        new RamseteCommand(
            exampleTrajectory,
            m_robotDrive::getPose,
            new RamseteController(AutoConstants.kRamseteB, AutoConstants.
↪kRamseteZeta),
            new SimpleMotorFeedforward(
                DriveConstants.ksVolts,
                DriveConstants.kvVoltSecondsPerMeter,
                DriveConstants.kaVoltSecondsSquaredPerMeter),
            DriveConstants.kDriveKinematics,
            m_robotDrive::getWheelSpeeds,
            new PIDController(DriveConstants.kPDriveVel, 0, 0),
            new PIDController(DriveConstants.kPDriveVel, 0, 0),
            // RamseteCommand passes volts to the callback
            m_robotDrive::tankDriveVolts,
            m_robotDrive);

    // Reset odometry to the initial pose of the trajectory, run path following
    // command, then stop at the end.
    return Commands.runOnce(() -> m_robotDrive.resetOdometry(exampleTrajectory.
↪getInitialPose()))
        .andThen(ramseteCommand)
        .andThen(Commands.runOnce(() -> m_robotDrive.tankDriveVolts(0, 0)));
  }
}
```

C++ (Source)

```
frc2::CommandPtr RobotContainer::GetAutonomousCommand() {
  // Create a voltage constraint to ensure we don't accelerate too fast
  frc::DifferentialDriveVoltageConstraint autoVoltageConstraint{
      frc::SimpleMotorFeedforward<units::meters>{
          DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
      DriveConstants::kDriveKinematics, 10_V};

  // Set up config for trajectory
  frc::TrajectoryConfig config{AutoConstants::kMaxSpeed,
                               AutoConstants::kMaxAcceleration};
  // Add kinematics to ensure max speed is actually obeyed
  config.SetKinematics(DriveConstants::kDriveKinematics);
  // Apply the voltage constraint
  config.AddConstraint(autoVoltageConstraint);

  // An example trajectory to follow.  All units in meters.
  auto exampleTrajectory = frc::TrajectoryGenerator::GenerateTrajectory(
      // Start at the origin facing the +X direction
      frc::Pose2d{0_m, 0_m, 0_deg},
      // Pass through these two interior waypoints, making an 's' curve path
      {frc::Translation2d{1_m, 1_m}, frc::Translation2d{2_m, -1_m}},
```

(suite sur la page suivante)

(suite de la page précédente)

```
      // End 3 meters straight ahead of where we started, facing forward
      frc::Pose2d{3_m, 0_m, 0_deg},
      // Pass the config
      config);

  frc2::CommandPtr ramseteCommand{frc2::RamseteCommand(
      exampleTrajectory, [this] { return m_drive.GetPose(); },
      frc::RamseteController{AutoConstants::kRamseteB,
                             AutoConstants::kRamseteZeta},
      frc::SimpleMotorFeedforward<units::meters>{
          DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
      DriveConstants::kDriveKinematics,
      [this] { return m_drive.GetWheelSpeeds(); },
      frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
      frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
      [this](auto left, auto right) { m_drive.TankDriveVolts(left, right); },
      {&m_drive})};

  // Reset odometry to the initial pose of the trajectory, run path following
  // command, then stop at the end.
  return frc2::cmd::RunOnce(
             [this, initialPose = exampleTrajectory.InitialPose()] {
               m_drive.ResetOdometry(initialPose);
             },
             {})
      .AndThen(std::move(ramseteCommand))
      .AndThen(
          frc2::cmd::RunOnce([this] { m_drive.TankDriveVolts(0_V, 0_V); }, {}));
}
```

Configuration des contraintes de la trajectoire

Tout d'abord, nous devons définir certains paramètres de configuration pour la trajectoire qui garantiront que la trajectoire ainsi générée peut effectivement être suivie.

Création d'une contrainte de tension

La première pièce de configuration dont nous aurons besoin est une contrainte de tension. Cela permettra de s'assurer que la trajectoire générée ne contraigne jamais le robot d'aller plus vite qu'il n'est capable d'atteindre avec l'alimentation de tension disponible :

Java

```
    // Create a voltage constraint to ensure we don't accelerate too fast
    var autoVoltageConstraint =
        new DifferentialDriveVoltageConstraint(
            new SimpleMotorFeedforward(
                DriveConstants.ksVolts,
                DriveConstants.kvVoltSecondsPerMeter,
                DriveConstants.kaVoltSecondsSquaredPerMeter),
```

(suite sur la page suivante)

(suite de la page précédente)

```
            DriveConstants.kDriveKinematics,
            10);
```

C++ (Source)

```
// Create a voltage constraint to ensure we don't accelerate too fast
frc::DifferentialDriveVoltageConstraint autoVoltageConstraint{
    frc::SimpleMotorFeedforward<units::meters>{
        DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
    DriveConstants::kDriveKinematics, 10_V};
```

Notez que nous avons fixé la tension maximale à 10V, plutôt qu'à la tension nominale de la batterie de 12V. Cela nous laisse une certaine « marge » pour faire face à un éventuel « creux de tension » pendant le suivi de la trajectoire.

Création de la configuration

Maintenant que nous avons notre contrainte de tension, nous pouvons créer notre instance `TrajectoryConfig`, qui englobe toutes nos contraintes de trajectoire :

Java

```
  // Create config for trajectory
  TrajectoryConfig config =
      new TrajectoryConfig(
              AutoConstants.kMaxSpeedMetersPerSecond,
              AutoConstants.kMaxAccelerationMetersPerSecondSquared)
          // Add kinematics to ensure max speed is actually obeyed
          .setKinematics(DriveConstants.kDriveKinematics)
          // Apply the voltage constraint
          .addConstraint(autoVoltageConstraint);
```

C++ (Source)

```
// Set up config for trajectory
frc::TrajectoryConfig config{AutoConstants::kMaxSpeed,
                             AutoConstants::kMaxAcceleration};
// Add kinematics to ensure max speed is actually obeyed
config.SetKinematics(DriveConstants::kDriveKinematics);
// Apply the voltage constraint
config.AddConstraint(autoVoltageConstraint);
```

Génération de la trajectoire

Avec notre configuration de trajectoire en main, nous sommes maintenant prêts à générer notre trajectoire. Pour cet exemple, nous allons générer une trajectoire « clamped cubic » - cela signifie que nous allons spécifier au complet les poses de robots aux extrémités, et spécifier des positions uniquement pour les points de passage intérieurs (également connu sous le nom « points noeuds »). Comme ailleurs, toutes les distances sont en mètres.

Java

```
100     // An example trajectory to follow. All units in meters.
101     Trajectory exampleTrajectory =
102         TrajectoryGenerator.generateTrajectory(
103             // Start at the origin facing the +X direction
104             new Pose2d(0, 0, new Rotation2d(0)),
105             // Pass through these two interior waypoints, making an 's' curve path
106             List.of(new Translation2d(1, 1), new Translation2d(2, -1)),
107             // End 3 meters straight ahead of where we started, facing forward
108             new Pose2d(3, 0, new Rotation2d(0)),
109             // Pass config
110             config);
```

C++ (Source)

```
60  // An example trajectory to follow.  All units in meters.
61  auto exampleTrajectory = frc::TrajectoryGenerator::GenerateTrajectory(
62      // Start at the origin facing the +X direction
63      frc::Pose2d{0_m, 0_m, 0_deg},
64      // Pass through these two interior waypoints, making an 's' curve path
65      {frc::Translation2d{1_m, 1_m}, frc::Translation2d{2_m, -1_m}},
66      // End 3 meters straight ahead of where we started, facing forward
67      frc::Pose2d{3_m, 0_m, 0_deg},
68      // Pass the config
69      config);
```

Note : Au lieu de générer la trajectoire sur le roboRIO comme indiqué ci-dessus, on peut également *importer un PathWeaver JSON*.

Création de la commande Ramsete ou RamseteCommand

Nous allons d'abord réinitialiser la pose de notre robot à la pose de départ de la trajectoire. Cela garantit que l'emplacement du robot sur le système de coordonnées et la position de départ de la trajectoire sont identiques.

Java

```
    // Reset odometry to the initial pose of the trajectory, run path following
    // command, then stop at the end.
    return Commands.runOnce(() -> m_robotDrive.resetOdometry(exampleTrajectory.
↪getInitialPose()))
```

C++ (Source)

```
  // Reset odometry to the initial pose of the trajectory, run path following
  // command, then stop at the end.
  return frc2::cmd::RunOnce(
```

It is very important that the initial robot pose match the first pose in the trajectory. For the purposes of our example, the robot will be reliably starting at a position of `(0,0)` with a heading of `0`. In actual use, however, it is probably not desirable to base your coordinate system on the robot position, and so the starting position for both the robot and the trajectory should be set to some other value. If you wish to use a trajectory that has been defined in robot-centric coordinates in such a situation, you can transform it to be relative to the robot's current pose using the `transformBy` method (Java, C++). For more information about transforming trajectories, see *La transformation des trajectoires*.

Now that we have a trajectory, we can create a command that, when executed, will follow that trajectory. To do this, we use the `RamseteCommand` class (Java, C++)

Java

```
    RamseteCommand ramseteCommand =
        new RamseteCommand(
            exampleTrajectory,
            m_robotDrive::getPose,
            new RamseteController(AutoConstants.kRamseteB, AutoConstants.
↪kRamseteZeta),
            new SimpleMotorFeedforward(
                DriveConstants.ksVolts,
                DriveConstants.kvVoltSecondsPerMeter,
                DriveConstants.kaVoltSecondsSquaredPerMeter),
            DriveConstants.kDriveKinematics,
            m_robotDrive::getWheelSpeeds,
            new PIDController(DriveConstants.kPDriveVel, 0, 0),
            new PIDController(DriveConstants.kPDriveVel, 0, 0),
            // RamseteCommand passes volts to the callback
            m_robotDrive::tankDriveVolts,
            m_robotDrive);
```

C++ (Source)

```
frc2::CommandPtr ramseteCommand{frc2::RamseteCommand(
    exampleTrajectory, [this] { return m_drive.GetPose(); },
    frc::RamseteController{AutoConstants::kRamseteB,
                           AutoConstants::kRamseteZeta},
    frc::SimpleMotorFeedforward<units::meters>{
        DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
    DriveConstants::kDriveKinematics,
    [this] { return m_drive.GetWheelSpeeds(); },
    frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
    frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
    [this](auto left, auto right) { m_drive.TankDriveVolts(left, right); },
    {&m_drive})};
```

Cette déclaration est assez substantielle, nous allons donc la scruter paramètre par paramètre :

1. La trajectoire : C'est la trajectoire à suivre ; en conséquence, nous passons à la commande la trajectoire que nous venons de construire dans nos étapes antérieures.
2. Le fournisseur de pose : il s'agit d'une référence de méthode (ou lambda) à la méthode du sous-système *drive qui renvoie la pose*. Le contrôleur RAMSETE a besoin de la mesure de pose actuelle pour déterminer les sorties de roue requises.
3. The RAMSETE controller : This is the `RamseteController` object (Java, C++) that will perform the path-following computation that translates the current measured pose and trajectory state into a chassis speed setpoint.
4. The drive feedforward : This is a `SimpleMotorFeedforward` object (Java, C++) that will automatically perform the correct feedforward calculation with the feedforward gains (`kS`, `kV`, and `kA`) that we obtained from the drive identification tool.
5. The drive kinematics : This is the `DifferentialDriveKinematics` object (Java, C++) that we constructed earlier in our constants file, and will be used to convert chassis speeds to wheel speeds.
6. Le fournisseur de vitesse de roue : il s'agit d'une référence de méthode (ou lambda) à la méthode du sous-système *drive qui renvoie les vitesses de roue*
7. The left-side PIDController : This is the `PIDController` object (Java, C++) that will track the left-side wheel speed setpoint, using the P gain that we obtained from the drive identification tool.
8. The right-side PIDController : This is the `PIDController` object (Java, C++) that will track the right-side wheel speed setpoint, using the P gain that we obtained from the drive identification tool.
9. Le consommateur de sortie : il s'agit d'une référence de méthode (ou lambda) à la méthode du sous-système drive qui transmet les sorties de tension aux moteurs d'entraînement.
10. Le sous-système d'entraînement du robot : Il s'agit du sous-système de déplacement lui-même, inclus ici pour s'assurer que la commande ne fonctionne pas sur la base pilotable en même temps que toute autre commande qui utilise cette même base pilotable.

Enfin, notez que nous ajoutons une commande finale « stop » dans l'ordre après la commande de suivi de la trajectoire, pour s'assurer que le robot s'immobilise à la fin de la trajectoire.

Vidéo

Si tout s'est bien passé, la routine autonome de votre robot devrait ressembler à ceci :

29.1.2 PathWeaver

Note : Users may find a community driven project PathPlanner as potentially more useful. PathPlanner improves upon traditional pathplanning applications with an intuitive user interface and swerve path following support. Note that WPILib offers no support for community projects.

Introduction à PathWeaver

Note : Users may find a community driven project PathPlanner as potentially more useful. PathPlanner improves upon traditional pathplanning applications with an intuitive user interface and swerve path following support. Note that WPILib offers no support for community projects.

Le Mode Autonome est une partie importante dans un match ; c'est excitant de voir les robots faire des choses impressionnantes en autonome. Pour marquer, le robot doit généralement se déplacer quelque part. Plus vite le robot peut arriver à cet endroit, plus tôt il peut marquer de points ! La méthode traditionnelle de se déplacer en mode Autonome consiste à avancer en ligne droite, tourner d'un certain angle, et avancer en ligne droite à nouveau. Cette approche fonctionne très bien, mais le robot passe beaucoup de temps à s'arrêter et à recommencer après chaque ligne droite et virage.

Une approche plus avancée en Mode Autonome s'appelle la « planification des trajectoires ». Au lieu d'avancer en ligne droite et de tourner une fois la ligne terminée, le robot se déplace continuellement, avançant dans un mouvement courbé. Cela peut réduire le temps d'arrêt de rotation.

la WPILib contient une suite pour la génération de trajectoires qui peut être utilisée par les équipes afin de générer et suivre des trajectoires. Cette série d'articles examinera comment générer et visualiser des trajectoires à l'aide de PathWeaver. Pour un tutoriel complet sur le suivi des trajectoires, veuillez consulter le didacticiel de trajectoires dans *Didacticiel sur le suivi de trajectoires*.

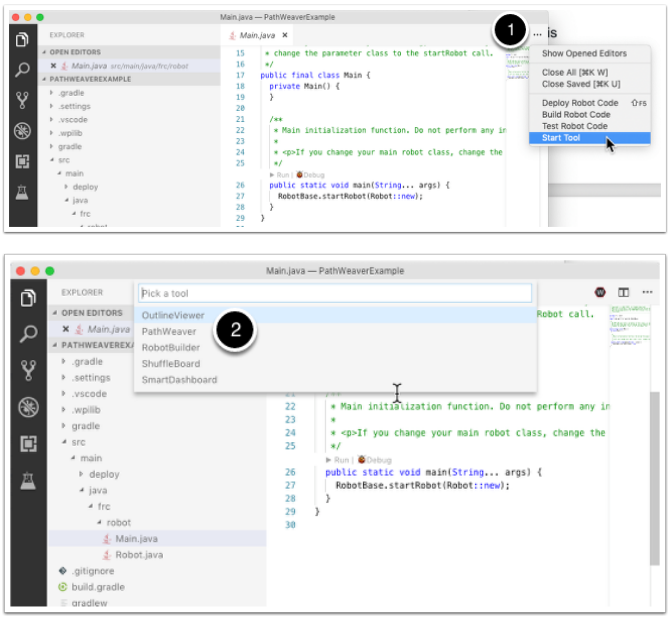
Note : *Le code de suivi des trajectoires* est requis pour utiliser PathWeaver. Nous vous recommandons de commencer avec le suivi de trajectoire et de le faire fonctionner avec des chemins simples. À partir de là, vous pouvez continuer à tester des chemins plus complexes générés par PathWeaver.

Création d'un projet Pathweaver

Le PathWeaver est l'outil utilisé pour tracer les trajectoires qu'un robot va suivre. Un projet PathWeaver ne stocke que les trajectoires définies pour un seul programme.

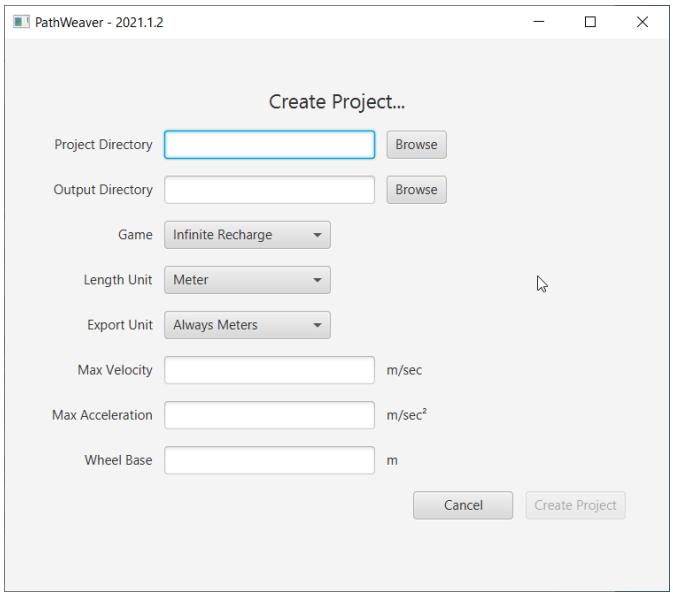
Démarrage de PathWeaver

On lance le PathWeaver en cliquant sur l'icône en forme d'ellipse située en haut à droite du coin de l'interface Visual Studio Code. Vous devez sélectionner un fichier source dans le projet WPILib pour afficher l'icône. Cliquez ensuite sur « Start tool » puis cliquez sur « PathWeaver » tel qu'indiqué ci-dessous.



Création du projet

Pour créer un projet PathWeaver, cliquez sur « Create project », puis remplissez le formulaire de création de projet. Notez qu'en plaçant votre souris sur l'un des champs du formulaire vous afficherez plus d'informations sur ce qui est requis.



Project Directory : Il s'agit normalement du répertoire de projet de niveau supérieur qui contient les fichiers build.gradle et src pour votre programme de robot. Le choix de ce répertoire est la façon prévue d'utiliser PathWeaver et qui fera en sorte de localiser tous les fichiers de sortie dans les répertoires concordants pour le déploiement automatique du chemin d'accès à votre robot.

Output directory : Répertoire où les trajectoires sont stockées pour le déploiement sur votre robot. Si vous avez spécifié le dossier de projet de niveau supérieur de votre projet de robot à l'étape précédente (comme recommandé), le remplissage du répertoire de sortie est facultatif.

Game : Le jeu (le jeu FRC® en vigueur) provoquera l'utilisation de la superposition correcte d'image de terrain . Vous pouvez également créer vos propres images de terrain et la procédure sera décrite plus tard dans cette série.

Length Unit : Les unités à utiliser pour décrire votre robot et pour les mesures sur le terrain lors de la visualisation des trajectoires à l'aide de PathWeaver.

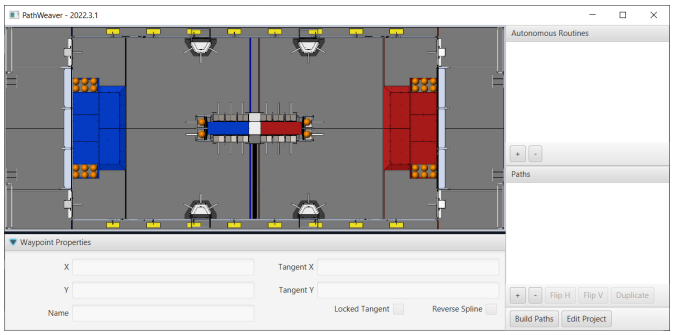
Export Unit : Les unités à utiliser lors de l'exportation des trajectoires et des points de passage. Si vous prévoyez d'utiliser les trajectoires WPILib, vous devez alors choisir l'option *Always Meters*. Le choix de l'option *Same as Project* exportera dans les mêmes unités que *Length Unit* ci-dessus.

Max Velocity : The max speed of the robot for trajectory tracking. The kitbot runs at ~10 ft/sec which is ~3 m/sec.

Max Acceleration : The max acceleration of the robot for trajectory tracking. Using a conservative 1 m/sec^2 is a good place to start if you don't know your drivetain's characteristics.

Wheel Base : La distance entre les roues gauche et droite de votre robot. Ce paramètre est utilisé pour s'assurer qu'aucune roue sur une commande différentielle ne dépassera la vitesse maximale spécifiée dans des virages.

Interface utilisateur PathWeaver

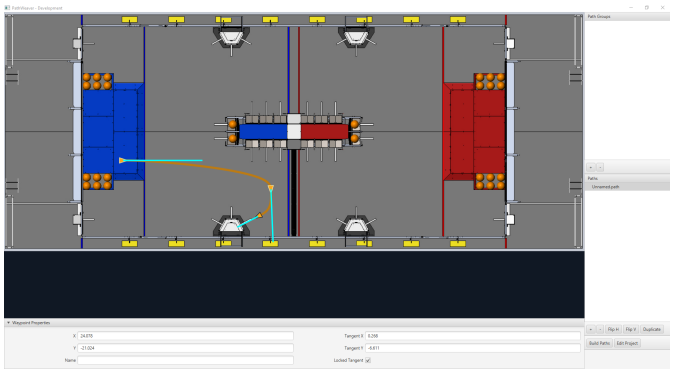


L'interface utilisateur PathWeaver se compose des éléments suivants :

1. Zone de champs localisée dans le coin supérieur gauche, qui occupe la majeure partie de la fenêtre PathWeaver. Les trajectoires sont tracées dans cette partie de l'interface.
2. Les propriétés du point de repère actuellement sélectionné s'affichent dans le volet inférieur. Ces propriétés comprennent l'abscisse X et l'ordonnée Y, ainsi que les tangentes à chaque point.
3. Un regroupement de trajectoires (ou un Mode « autonome ») s'affiche sur le côté supérieur droit de la fenêtre. C'est une façon pratique de voir toutes les trajectoires pour un Mode autonome donné.
4. Les trajectoires individuelles qu'un robot peut suivre sont affichées dans le côté inférieur droit de la fenêtre.
5. Le bouton « Build Paths » sert à exporter les trajectoires sous le format JSON. Ces fichiers JSON peuvent, par la suite, être utilisés à partir du code robot pour suivre la trajectoire.
6. Le bouton « Edit Project » vous permet de modifier les propriétés du projet.

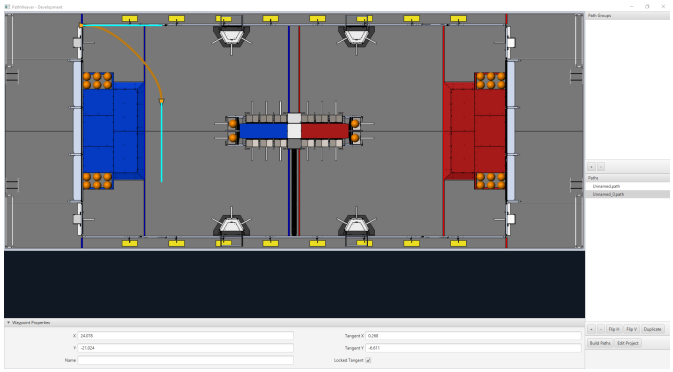
Visualisation des trajectoires PathWeaver

La principale caractéristique de PathWeaver est de visualiser les trajectoires. Les images suivantes représentent une trajectoire lisse qui correspond à une trajectoire qu'un robot pourrait prendre pendant la période Autonome. Les trajectoires peuvent avoir n'importe quel nombre de points de passage ou waypoints qui permettent de définir des trajectoies beaucoup plus complexes. Dans ce cas, il ya 3 points de passage (incluant le début et l'arrêt) représentés avec les icônes triangulaires. Chaque point de passage se compose d'une position X, Y sur le terrain ainsi que de l'orientation du robot décrite comme des lignes tangentielles en X et Y.



Création de la trajectoire initiale

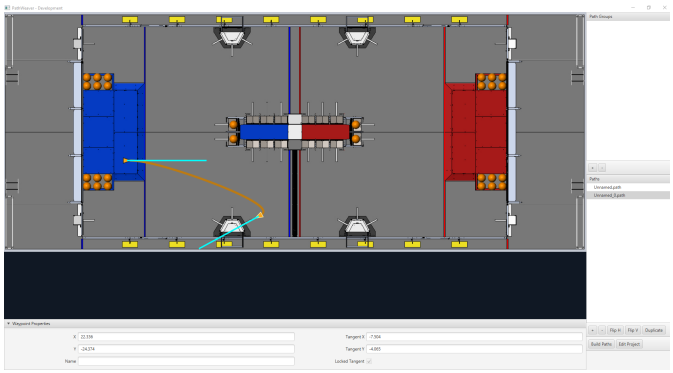
Pour débuter la création d'une trajectoire, cliquez sur le bouton + (plus) dans la fenêtre **Paths** en bas à droite. Une trajectoire par défaut sera créée qui n'a probablement pas les points de début et de fin que vous désirez. Cette trajectoire affiche également les vecteurs tangents (lignes de couleur sarcelle) aux points de début et de fin de trajectoire. En changeant l'angle de ces vecteurs tangents, on modifie la forme de la trajectoire.



Faites glisser les points de début et de fin de la trajectoire vers les emplacements souhaités. Notez que dans ce cas, la trajectoire par défaut ne commence pas dans un point légal pour le jeu 2019. Nous pouvons faire glisser le point de passage initial pour faire démarrer le robot sur la Plateforme d'Habitat.

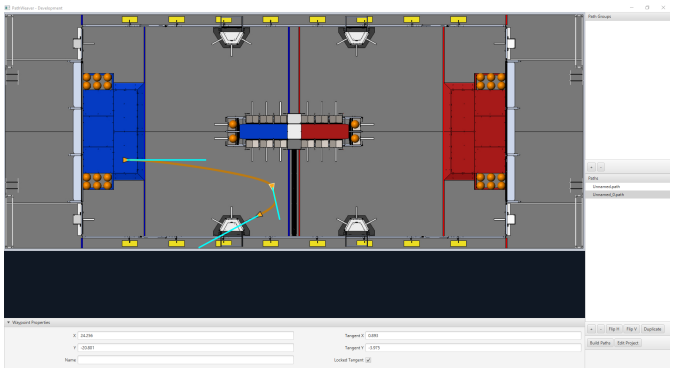
Modification de l'orientation d'un point de passage

L'orientation du robot peut être modifiée en faisant glisser le support du vecteur tangent (sarcelle). Ici, le point de passage final a été déplacée à la pose finale désirée et a été tourné pour faire face à la fusée.



Ajout de points de passage supplémentaires pour contrôler la trajectoire du robot

L'ajout de points de passage supplémentaires et la modification de leurs vecteurs tangents peuvent affecter la trajectoire suivie. Des points de passage supplémentaires peuvent être ajoutés en faisant glisser la souris à l'endroit voulu de la trajectoire. Dans ce cas, nous avons ajouté un autre point de passage au milieu du trajet.



Verrouillage des lignes tangentielles

Le verrouillage des lignes tangentielles les empêche de changer lorsque la trajectoire est manipulée. Les lignes tangentielles seront également verrouillées lorsque le point est déplacé.

Contrôle plus précis des points de passage

While PathWeaver makes it simple to draw trajectories that the robot should follow, it is sometimes hard to precisely set where the waypoints should be placed. In this case, setting the waypoint locations can be done by entering the X and Y value which might come from an accurate *CAD* model of the field. The points can be entered in the X and Y fields when a waypoint is selected.

Créer des routines autonomes

Les routines autonomes (également connues sous le nom de groupes de chemins) sont un moyen de visualiser où un chemin se termine et le suivant commence. Un exemple est lorsque le programme du robot parcourt un chemin, fait quelque chose une fois le chemin terminé, se rend à un autre endroit pour obtenir une pièce de jeu, puis revient pour la marquer. Il est important que les points de départ et d'arrivée de chaque chemin de la routine aient des points d'arrivée et de départ communs. En ajoutant tous les chemins à une seule routine autonome et en sélectionnant la routine, tous les chemins de cette routine seront affichés. Ensuite, chaque chemin peut être modifié tout en visualisant tous les chemins.

Créer une routine autonome

Appuyez sur le bouton + sous Routines autonomes. Faites ensuite glisser les chemins de la section Chemins vers votre routine autonome.

Chaque chemin ajouté à une routine autonome sera dessiné dans une couleur différente, ce qui facilitera la détermination du nom de chaque chemin.

S'il existe plusieurs chemins dans une routine, la sélection fonctionne comme suit :

1. La sélection de la routine affiche tous les chemins de la routine, ce qui permet de voir facilement la relation entre eux. Tout point de cheminement sur l'un des chemins peut être modifié pendant que la routine est sélectionnée et cela ne changera que le chemin contenant le point de cheminement.
2. La sélection d'un seul chemin dans la routine n'affichera que ce chemin, ce qui permettra de voir précisément ce que font tous les points de cheminement et d'éviter l'encombrement dans l'interface si plusieurs chemins se croisent ou sont proches les uns des autres.

Importation d'un fichier JSON PathWeaver

La classe `TrajectoryUtil` peut être utilisée pour importer un fichier JSON PathWeaver dans le code robot et permettre à ce dernier de suivre la trajectoire générée. Dans cet article, nous allons voir comment cette importation s'effectue. Veuillez consulter la section *Didacticiel sur la pratique des trajectoires* pour plus d'informations sur le suivi de trajectoires.

Les méthodes statiques `fromPathweaverJson` (Java) / `FromPathweaverJson` (C++) de la classe `TrajectoryUtil` peuvent être utilisées pour créer une trajectoire à partir d'un fichier JSON importé de PathWeaver et intégré dans le code du robot.

Important : Pour être compatible avec la vue `Field2d` dans l'interface graphique du simulateur, les coordonnées du JSON exporté ont changé. Auparavant (avant 2021), la plage de la coordonnée y était de -27 pieds à 0 pieds alors que maintenant, la plage de la coordonnée y est de 0 pieds à 27 pieds (0 étant au bas de l'écran et 27 pieds étant en haut). Cela ne devrait pas affecter les équipes qui ont correctement réinitialiser leur odométrie à la pose de départ de la trajectoire avant de suivre la trajectoire.

Note : PathWeaver place les fichiers JSON dans `src/main/deploy/paths` qui seront automatiquement placés sur le système de fichiers du roboRIO dans `/home/lvuser/deploy/paths` et accessibles à l'aide de getDeployDirectory comme indiqué ci-dessous.

JAVA

```
String trajectoryJSON = "paths/YourPath.wpilib.json";
Trajectory trajectory = new Trajectory();

@Override
public void robotInit() {
   try {
      Path trajectoryPath = Filesystem.getDeployDirectory().toPath().
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪resolve(trajectoryJSON);
      trajectory = TrajectoryUtil.fromPathweaverJson(trajectoryPath);
   } catch (IOException ex) {
      DriverStation.reportError("Unable to open trajectory: " + trajectoryJSON, ex.
↪getStackTrace());
   }
}
```

C++

```
#include <frc/Filesystem.h>
#include <frc/trajectory/TrajectoryUtil.h>
#include <wpi/fs.h>

frc::Trajectory trajectory;

void Robot::RobotInit() {
   fs::path deployDirectory = frc::filesystem::GetDeployDirectory();
   deployDirectory = deployDirectory / "paths" / "YourPath.wpilib.json";
   trajectory = frc::TrajectoryUtil::FromPathweaverJson(deployDirectory.string());
}
```

Dans les exemples ci-dessus, `YourPath` doit être remplacé par le nom de votre chemin.

Avertissement : Chargement d'une trajectoire PathWeaver JSON à partir d'un fichier en Java peut prendre plus d'une boucle d'itération, il est donc fortement recommandé que le robot charge ces trajectoires au démarrage.

Ajout d'images de Terrain à PathWeaver

Vous trouverez ici des instructions pour ajouter votre propre image de terrain de compétition en utilisant celle du jeu 2019 à titre d'exemple.

Les images des jeux sont chargées à partir du répertoire `~/PathWeaver/Games` sous Linux et macOS ou `%USERPROFILE%/PathWeaver/Games` sous Windows. Les fichiers peuvent être dans un sous-répertoire spécifique au jeu ou dans un fichier zip à l'intérieur du répertoire Games. Le fichier ZIP doit suivre la même disposition qu'un répertoire de jeux ; Le fichier JSON doit être dans la racine du fichier ZIP (ne peut pas être dans un sous-répertoire).

Download the example *FIRST* Destination Deep Space field definition `here`. Other field definitions are available in the allwpilib GitHub repository.

Format de fichier

```
~/PathWeaver
  /Games
   /Custom Game
      custom-game.json
      field-image.png
   OtherGame.zip
```

Format JSON

```
{
   "game": "game name",
   "field-image": "relative/path/to/img.png",
   "field-corners": {
      "top-left": [x, y],
      "bottom-right": [x, y]
   },
   "field-size": [width, length],
   "field-unit": "unit name"
}
```

Le chemin d'accès à l'image du terrain est relatif au fichier JSON. Pour faire simple, le fichier image doit être placé dans le même répertoire que le fichier JSON.

En exprimant les coordonnées X et Y en pixels, les coins du terrain correspondant au point situé en haut à gauche et à celui situé en bas à droite définissent les limites de l'aire rectangulaire de la zone jouable dans l'image du terrain. Les aires de jeu non rectangulaires ne sont pas prises en charge.

La taille du terrain est la largeur et la longueur de l'aire de jeu du terrain dans les unités fournies.

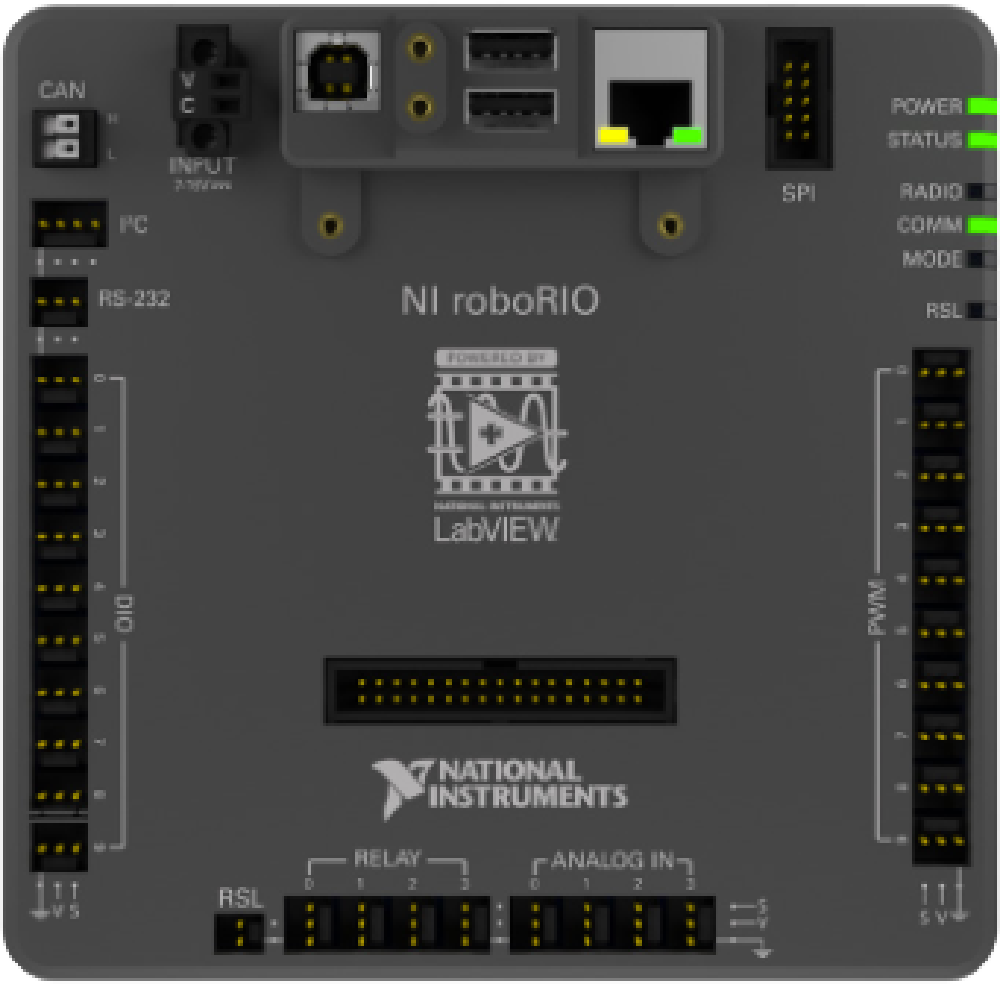
Les unités du terrain sont insensibles à la casse et peuvent être en mètres, cm, mm, pouces, pieds, yards, ou miles. Le singulier, le pluriel et les abréviations sont pris en charge (p. ex. « mètre », « mètres », et « m » sont toutes valables pour définir les mètres)

Note : Lors de la création d'une nouvelle image de champ, une bordure (un minimum de 20 pixels est recommandé) doit être laissée à l'extérieur afin que les points de cheminement sur le bord du champ soient accessibles.

30

roboRIO

30.1 Introduction au roboRIO



Le roboRIO est conçu en ayant spécifiquement FIRST à l'esprit. Le roboRIO a une architecture de base d'un processeurs en temps réel + FPGA (réseau de portes programmables) mais est

plus puissant, plus léger et plus petit que certains systèmes similaires utilisés dans l'industrie.

Le roboRIO est un contrôleur reconfigurable de robots qui comprend des ports intégrés pour les circuits inter-intégrés (I2C), des interfaces périphériques en série (SPI), RS232, USB, Ethernet, la modulation par largeur d'impulsions (PWM), et des relais pour connecter rapidement les capteurs et actionneurs communs utilisés en robotique. Le contrôleur est doté de LED, de boutons, d'un accéléromètre à bord et d'un port électronique personnalisé. Il dispose d'un processeur embarqué CORTEX-A9 à double coeur ARM temps réel et d'un FpGA Xilinx personnalisable.

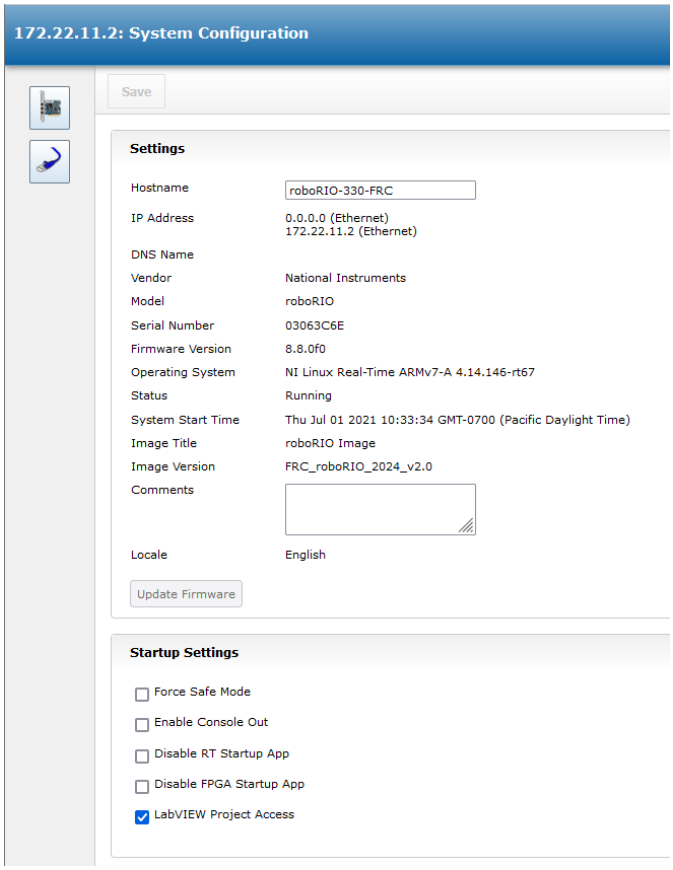
Detailed information on the roboRIO can be found in the roboRIO User Manual and in the roboRIO technical specifications.

Before deploying programs to your roboRIO, you must first image the roboRIO : *roboRIO 1 roboRIO 2*.

30.2 Tableau de bord Web du roboRIO

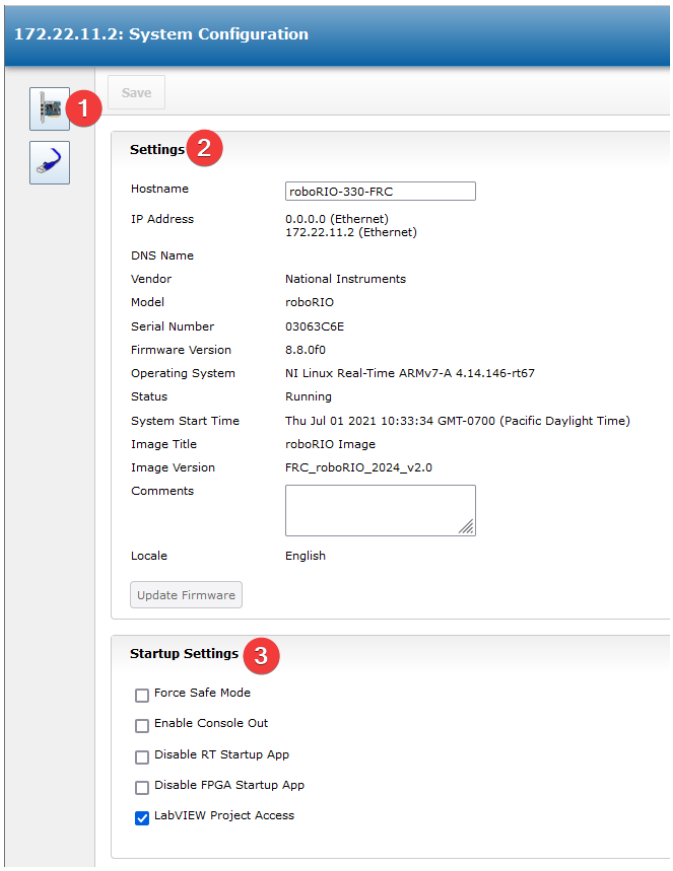
Le tableau de bord web roboRIO est une page Web intégrée dans le roboRIO qui peut être utilisée pour vérifier l'état et mettre à jour les paramètres du roboRIO.

30.2.1 Ouverture de l'interface Web



Pour ouvrir le tableau de bord Web, ouvrez un navigateur Web et entrez l'adresse du roboRIO dans la barre d'adresses (172.22.11.2 pour USB, ou « roboRIO-#####-FRC.local où #### est votre numéro d'équipe, sans zéros en tête, pour l'une ou l'autre interface). Pour plus de détails sur les configurations mDNS et roboRIO, consultez le document suivant : *IP Configurations*

30.2.2 Onglet System Configuration



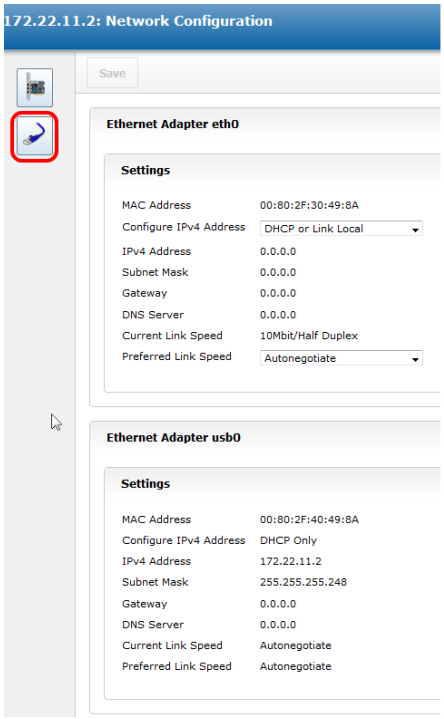
La fenêtre d'accueil du tableau de bord Web est l'onglet *System Configuration* qui consiste en 5 sections principales :

1. Barre de navigation - Cette section vous permet de naviguer vers différentes sections du tableau de bord Web. Les différentes pages accessibles par cette barre de navigation sont discutées ci-dessous.
2. System Settings - Cette section contient des informations sur les paramètres système. Le champ *Hostname* ne doit pas être modifié manuellement, utilisez plutôt l'outil *roboRIO Imaging tool* pour définir le nom d'hôte en fonction de votre numéro d'équipe. Cette section contient des informations telles que l'adresse IP du périphérique, la version du firmware et la version de l'image.
3. Startup Settings - Cette section contient les paramètres de démarrage du roboRIO. Celles-ci sont décrites dans la sous-étape ci-dessous
4. System Resources (pas montré) - Cette section fournit un instantané des ressources système telles que la mémoire et la charge du processeur.

Startup Settings

- Force Safe Mode - Force le contrôleur en mode sans échec. Cela peut être utilisé dans le dépannage des problèmes en lien avec l'image du roboRIO, mais il est plutôt recommandé d'utiliser le bouton Reset sur le roboRIO pour mettre l'appareil en mode sans échec (avec l'alimentation déjà branchée et établie, maintenez le bouton de repos pendant 5 secondes). **La valeur par défaut n'est pas cochée.**
- Enable Console Out - This enables the on-board RS232 port to be used as a Console output. This port uses RS232 levels and should not be connected to many microcontrollers which use TTL levels). **Default is unchecked.**
- Disable RT Startup App - Cocher cette case désactive l'exécution du code au démarrage. Ce paramètre peut être utilisé pour le dépannage si vous trouvez que le roboRIO ne réagit pas après le déploiement d'un nouveau programme. Par défaut, cette case n'est pas cochée
- Disable FPGA Startup App - **Cette case ne doit pas être cochée.**
- LabVIEW Project Access - **It is recommended to leave this box checked.** This setting allows LabVIEW projects to access the roboRIO.

30.2.3 Configuration du réseau



Cette page montre la configuration des cartes réseau du roboRIO. **Il n'est pas recommandé**

de modifier les paramètres de cette page. Pour plus d'informations sur le réseau roboRIO, consulter l'article suivant : *IP Configurations*

30.3 roboRIO FTP

Note : Le roboRIO a à la fois les protocoles SFTP et FTP anonyme activés. Cet article décrit comment utiliser chacun pour accéder au système de fichiers du roboRIO.

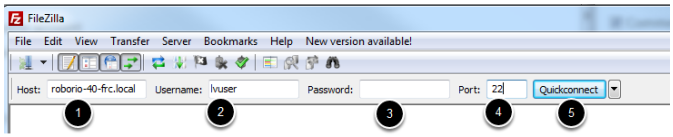
30.3.1 SFTP

SFTP est le moyen recommandé d'accéder au système de fichiers roboRIO. Étant donné que vous utiliserez le même compte sous lequel votre programme sera exécuté, les fichiers copiés doivent toujours avoir des autorisations compatibles avec votre code.

Logiciel

Il existe un certain nombre de programmes disponibles gratuitement pour le service SFTP. Cet article portera sur l'utilisation de FileZilla. Vous pouvez télécharger et installer FileZilla avant de procéder ou d'extrapoler les instructions ci-dessous à votre client de choix SFTP.

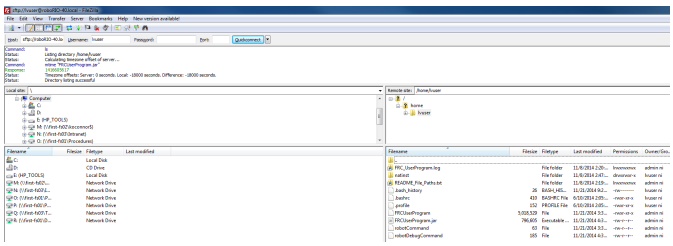
Connexion au roboRIO



Pour vous connecter à votre roboRIO :

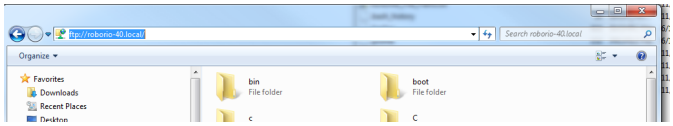
1. Entrez le nom mDNS (roboRIO-TEAM-frc.local) dans la zone de texte « Host »
2. Entrez « lvuser » dans la zone de texte Username (c'est le compte sous lequel votre programme s'exécute)
3. Laissez vide la zone de texte Password
4. Saisissez « 22 » dans la zone de texte port (le port SFTP par défaut)
5. Cliquez sur Quickconnect

Navigation sur le système de fichiers du roboRIO



Après s'être connecté au roboRIO, Filezilla s'ouvre sur le répertoire \home\lvuser. Le volet droit est le système distant (le roboRIO), le volet gauche est le système local (votre ordinateur). La section supérieure de chaque volet vous montre la hiérarchie du répertoire actuel que vous parcourez, le volet inférieur affiche le contenu du répertoire. Pour transférer des fichiers, il suffit de cliquer et de faire glisser d'un côté à l'autre. Pour créer des répertoires sur le roboRIO, cliquez avec le bouton droit et sélectionnez « Create Directory ».

30.3.2 FTP



The roboRIO also has anonymous FTP enabled. It is recommended to use SFTP as described above, but depending on what you need FTP may work in a pinch with no additional software required. To FTP to the roboRIO, open a Windows Explorer window. In the address bar, type ftp://roboRIO-TEAM-frc.local and press enter. You can now browse the roboRIO file system just like you would browse files on your computer.

30.4 Les comptes d'utilisateurs roboRIO et SSH

Note : Ce document contient des sujets avancés non requis pour une programmation typique en FRC®

L'image du roboRIO contient un certain nombre de comptes, cet article mettra en évidence les deux qui sont utilisés en FRC et fournir quelques détails sur leur but. Il décrira également comment se connecter au roboRIO via SSH.

30.4.1 Les comptes d'utilisateurs roboRIO

L'image du roboRIO contient un certain nombre de comptes d'utilisateurs, mais il en y a deux d'intérêt primordial pour la FRC.

admin

Le compte « admin » a un accès root au système et peut être utilisé pour manipuler les fichiers ou paramètres du système d'exploitation. Les équipes doivent faire preuve de prudence lors de l'utilisation de ce compte car il permet la modification des paramètres et des fichiers qui peuvent corrompre le système d'exploitation du roboRIO. Les informations d'identification de ce compte sont les suivantes :

```
Username: admin
```

```
Password:
```

Note : Le mot de passe est sciemment laissé vide.

---

lvuser

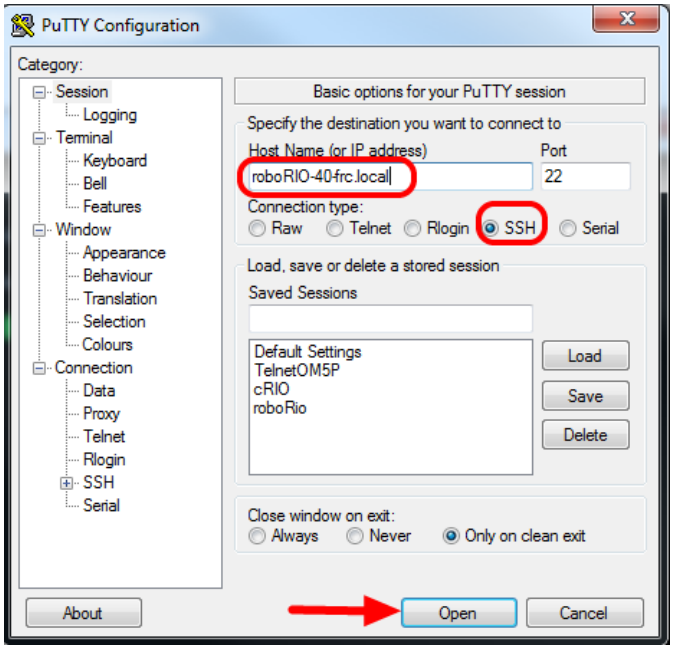
Le compte « lvuser » est le compte utilisé pour exécuter le code utilisateur pour les trois langages de programmation. Les informations d'identification de ce compte ne doivent pas être modifiées. Les équipes peuvent vouloir utiliser ce compte (via ssh ou sftp) lorsqu'elles travaillent avec le roboRIO pour s'assurer que tous les fichiers ou modifications de paramètres sont effectuées sur le même compte sous lequel leur code s'exécutera.

Danger : Changing the default ssh passwords for either « lvuser » or « admin » will prevent C++, Java, and Python teams from uploading code.

30.4.2 SSH

SSH (Secure SHell) est un protocole utilisé pour la communication de données sécurisée. Lorsqu'il est largement mentionné en ce qui concerne un système Linux (comme celui qui s'exécute sur le roboRIO), c'est généralement pour se référer à l'accès à la console de ligne de commande à l'aide du protocole SSH. Cela peut être utilisé pour exécuter des commandes sur le système distant. Un client gratuit pouvant être utilisé pour SSH est PuTTY : https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Ouvrir Putty



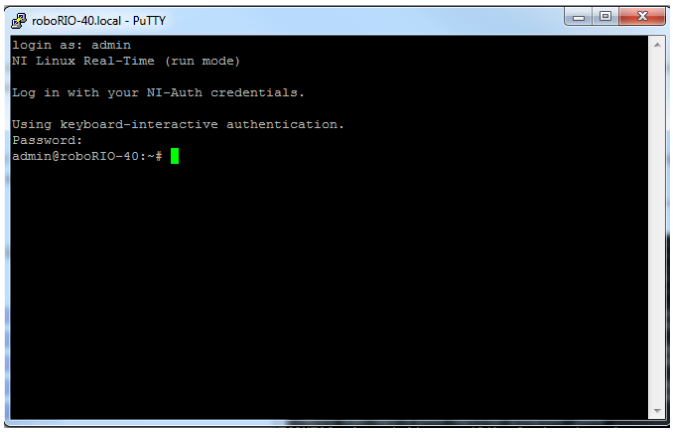
Ouvrez Putty (en cliquant sur OK à n'importe quelle invite de sécurité). Définissez ensuite les paramètres suivants :

1. Host Name : roboRIO-TEAM-frc.local (où TEAM est votre numéro d'équipe, l'exemple montre l'équipe 40)
2. Connection Type : SSH

Les autres paramètres peuvent être laissés à leurs valeurs par défaut. Cliquez sur Open pour ouvrir la connexion. Si vous voyez une invite sur les touches SSH, cliquez sur OK.

Si vous êtes connecté via USB, vous pouvez utiliser le 172.22.11.2 comme nom d'hôte. Si votre roboRIO est défini sur une adresse IP statique, vous pouvez utiliser cette adresse IP comme nom d'hôte si elle est connectée via Ethernet/sans fil.

Connectez-vous



Lorsque vous voyez l'invite, entrez le nom d'utilisateur souhaité (voir ci-dessus pour la description), puis appuyez sur Entrée. À l'invite de mot de passe, appuyez sur entrée (le mot de passe pour les deux comptes est vide).

30.5 Baisse de tension du roboRIO et compréhension de la consommation de courant

Afin d'aider à maintenir la tension de la batterie pour la préserver ainsi que d'autres composants du système de contrôle comme la radio dans des situations d'appel élevé de courant, le roboRIO est pourvu d'un système organisé de protection contre la baisse de tension. Cet article décrit ce système, fournit des informations sur la planification proactive de l'appel de courant du système et décrit comment utiliser la nouvelle fonctionnalité du PDP ainsi que le Visionneur de fichiers journaux DS pour comprendre les manifestations de baisses de tensions si elles se produisent sur votre robot.

30.5.1 Protection contre la baisse de tension du roboRIO

Le roboRIO utilise un système organisé de protection contre la baisse de tension pour tenter de préserver la tension d'entrée, pour lui-même et d'autres composants du système de contrôle afin d'éviter la réinitialisation des dispositifs en cas d'appels de courant élevés tirant la tension de la batterie dangereusement vers le bas.

Étape 1 - baisse de sortie de 6v

Déclencheur à tension - 6.8V

When the voltage drops below 6.8V, the 6V output on the *PWM* pins will start to drop.

Étape 2 - Désactiver la sortie

Déclencheur à tension - 6.3V

Lorsque la tension descend en dessous de 6,3 V, le contrôleur entrera dans l'état de protection contre la baisse de tension. Les indicateurs suivants montrent que cette condition s'est produite :

- Le voyant Power LED du roboRIO deviendra ambré
- L'arrière-plan de l'affichage de tension sur la Driver Station deviendra rouge
- Le mode d'affichage sur la Driver Station changera à Voltage Brownout
- L'onglet CAN/Power du DS incrémente le compteur 12V fault 1.
- La DS enregistrera un événement de baisse de tension dans le journal DS.

Le contrôleur prendra les mesures suivantes pour tenter de préserver la tension de la batterie :

- Les sorties PWM seront désactivées. Pour les sorties PWM dont la valeur neutre est configurée (tous les contrôleurs de moteurs de WPILib), une impulsion neutre unique sera envoyée avant que la sortie ne soit désactivée.
- Les rails d'utilisateur 6V, 5V, 3.3V sont désactivés (Cela inclut les sorties 6V sur les broches PWM, les broches 5V dans la reglette de raccordement DIO, les broches 5V dans le module analogique, les broches 3.3V dans le module SPI et I2C et les broches 5V et 3.3V dans le module MXP)

- Les ports GPIO configurés en sorties passent à un état haute-impédence (High-Z)
- Les sorties de relais sont désactivées (mises au niveau bas)
- Les contrôleurs moteurs de type CAN reçoivent une commande explicite de désactivation
- Les dispositifs pneumatiques tels que le module de contrôle pneumatique CTRE et le concentrateur pneumatique REV sont désactivés

Le contrôleur restera dans cet état jusqu'à ce que la tension monte à plus de 7,5 V ou tombe en dessous du seuil de déclanchement pour l'étape suivante de la chute de tension

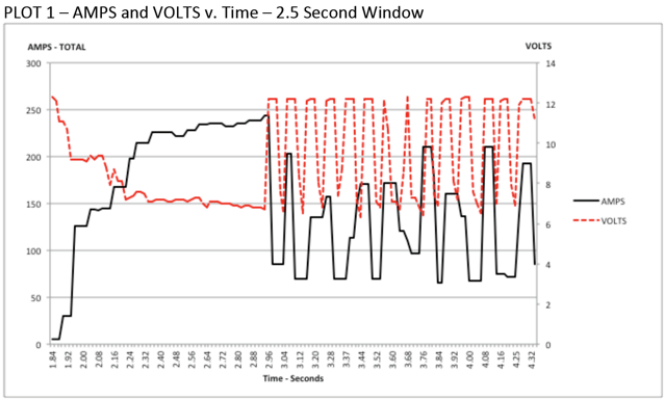
Étape 3 - Arrêt de l'appareil

Déclencheur à tension - 4.5V

En dessous de 4.5V l'appareil peut s'éteindre. La tension exacte peut être inférieure à cette valeur et dépend de la charge sur l'appareil.

Le contrôleur restera dans cet état jusqu'à ce que la tension s'élève au-dessus de 4,65 V lorsque l'appareil commencera la séquence de démarrage normale.

30.5.2 Éviter la baisse de tension - Planification proactive de la consommation de courant



La façon pour éviter une situation de baisse de tension est de planifier de manière proactive l'appel de courant de votre robot. La meilleure façon de le faire est de créer une certaine forme de budget de puissance. Il peut s'agir d'un document complexe qui tente de quantifier à la fois le courant consommé estimé et le temps dans un effort de mieux comprendre l'utilisation de la puissance et donc l'état de la batterie à la fin d'un match, ou il peut être un simple inventaire de l'utilisation du courant. Pour ce faire :

1. Établissez le courant maximum « soutenu » (soutenu étant défini ici comme non momentané). C'est probablement la partie la plus difficile de la création du budget de puissance. La consommation de courant exacte qu'une batterie peut supporter tout en maintenant une tension de plus de 7 volts dépend de divers facteurs tels que la santé de la batterie (voir *ceci*, un article de mesure de la santé de la batterie) et de l'état de charge. Comme le montre la fiche technique NP18-12, le graphique de tension aux bornes devient très raide à mesure que l'état de charge diminue, en particulier lorsque la consommation de courant augmente. Cette fiche technique montre qu'à une charge continue de 3CA (54A), une batterie neuve peut fonctionner en continu

pendant plus de 6 minutes tout en maintenant une tension aux bornes supérieure à 7V. Comme indiqué dans l'image ci-dessus (utilisé avec l'autorisation du *document Team 234s Drive System Testing <https ://www.chiefdelphi.com/t/paper-new-control-functions-drive-system-testing/139165>* __) , même avec une batterie neuve, consommer 240A pendant plus d'une seconde ou deux est susceptible de causer un problème. Cela nous donne quelques limites sur l'établissement de notre consommation de courant soutenu. Pour les besoins de cet exercice, nous fixerons notre limite à 180 A.

2. Énumérez les différentes fonctions de votre robot telles que la transmission, le manipulateur, le mécanisme de jeu principal, etc.
3. Start assigning your available current to these functions. You will likely find that you run out pretty quickly. Many teams gear their drivetrain to have enough *torque* to slip their wheels at 40-50A of current draw per motor. If we have 4 motors on the drivetrain, that eats up most, or even exceeds, our power budget ! This means that we may need to put together a few scenarios and understand what functions can (and need to be) be used at the same time. In many cases, this will mean that you really need to limit the current draw of the other functions if/while your robot is maxing out the drivetrain (such as trying to push something). Benchmarking the « driving » current requirements of a drivetrain for some of these alternative scenarios is a little more complex, as it depends on many factors such as number of motors, robot weight, gearing, and efficiency. Current numbers for other functions can be done by calculating the power required to complete the function and estimating efficiency (if the mechanism has not been designed) or by determining the *torque* load on the motor and using the torque-current curve to determine the current draw of the motors.
4. Si vous avez déterminé des fonctions mutuellement exclusives dans votre analyse, envisagez d'appliquer l'exclusion dans le logiciel. Vous pouvez également utiliser la surveillance du courant du PDP (couvert plus en détail ci-dessous) dans votre programme robot pour obtenir des limites de sortie ou des exclusions dynamiquement (comme ne pas actionner un moteur de mécanisme lorsque le courant de transmission est supérieur à X ou seulement laisser le moteur fonctionner jusqu'à la moitié de sortie maximale lorsque le courant de transmission est au-dessus de Y).

30.5.3 Brownout réglable

Le NI roboRIO 1.0 ne prend pas en charge les tensions de brownout personnalisées. Il est fixé à 6.3 V comme mentionné à l'étape 2 ci-dessus.

Le NI roboRIO 2.0 ajoute l'option pour un niveau de brownout réglable par logiciel. Le niveau de brownout par défaut (étape 2) du roboRIO 2.0 est de 6.75 V.

JAVA

```
RobotController.setBrownoutVoltage(7.0);
```

C++

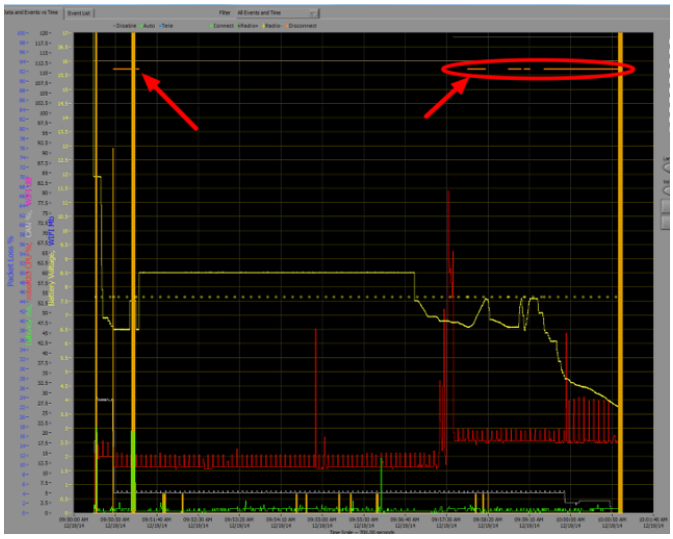
```
frc::RobotController::SetBrownoutVoltage(7_V);
```

30.5.4 Mesure de la consommation de courant à l'aide du PDP/PDH

L'application Driver Station FRC® travaille en conjonction avec le roboRIO et le PDP/PDH pour extraire les données enregistrées du PDP/PDH et les enregistrer sur votre PC DS. Un visionneur pour ces données est encore en développement.

En attendant, les équipes peuvent utiliser leur code robot et l'enrégistrement manuel, un panneau avant LabVIEW ou le SmartDashboard pour visualiser la consommation actuelle de courant de leur robot au fur et à mesure que les mécanismes entrent en action. Sous LabVIEW, vous pouvez lire le courant sur un canal PDP/PDH à l'aide l'instrument virtuel Get PD Currents qui se trouve dans la palette Power. Pour les équipes C++ et Java, utilisez la classe PowerDistribution comme décrit dans l'article *Power Distribution*. Tracer ces informations au fil du temps (plus facile avec un panneau avant LV ou avec le SmartDashboard à l'aide d'un indicateur graphique) peut fournir des informations pour comparer et mettre à jour votre budget de puissance ou peut localiser des mécanismes qui ne semblent pas fonctionner comme prévu (en raison d'un calcul de charge incorrect, d'hypothèses d'efficacité incorrectes ou de problèmes de mécanisme tels que la liaison).

30.5.5 Identification des baisses de tension



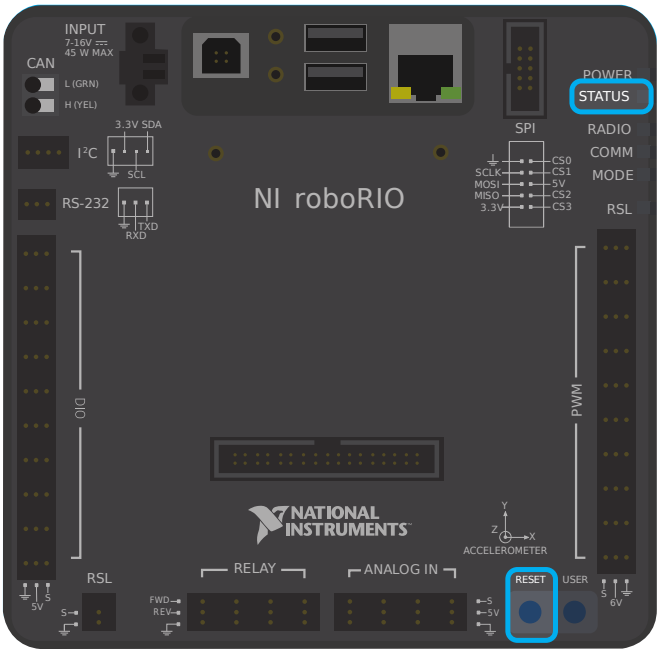
La façon la plus simple d'identifier une baisse de tension est de cliquer sur l'onglet CANPower de la Driver Station et de vérifier le nombre de défauts 12V. Alternativement, vous pouvez consulter le journal de la Driver Station après coup à l'aide du visionneur de journal. Le journal indiquera les baisses de tension avec une ligne orange vif, comme dans l'image ci-dessus (notez que ces baisses de tension ont été induites avec une alimentation sur un banc d'essai et peuvent ne pas refléter la durée et le comportement des baisses de tension sur un robot FRC typique).

30.6 Récupération d'un roboRIO en mode sans échec

De temps en temps, un roboRIO peut être corrompu au point qu'il ne peut pas être récupéré à l'aide du démarrage normal et du processus de création d'image. Le démarrage du roboRIO en mode sans échec peut permettre de le re-imager avec succès.

Important : These steps are not applicable to the roboRIO 2. Reimaging the SD card following *roboRIO 2.0 microSD card imaging process* will fully reformat the device.

30.6.1 Démarrage en mode sans échec



Pour démarrer le roboRIO en mode sans échec :

1. Mettez le roboRIO sous tension
2. Appuyez et maintenez le bouton Reset jusqu'à ce que le voyant DEL d'état s'allume (~5 secondes), puis relâchez le bouton Reset
3. Le roboRIO démarre en mode sans échec (indiqué par le voyant DEL d'état clignotant dans les groupes de 3)

30.6.2 Récupération du roboRIO

The roboRIO can now be imaged by using the roboRIO Imaging Tool as described in *Imaging your roboRIO*.

30.6.3 À propos du mode sans échec

In Safe Mode, the roboRIO boots a separate copy of the operating system into a RAM Disk (the firmware). This allows you to recover the roboRIO even if the normal copy of the OS is corrupted. While in Safe Mode, any changes made to the OS (such as changes made by accessing the device via SSH or Serial) will not persist to the normal copy of the OS stored on disk.

31

GradleRIO avancé

GradleRIO est le mécanisme qui est au coeur du déploiement du code robot sur le roboRIO. GradleRIO est construit sur la populaire dépendance Gradle et le système de gestion de compilation. Cette section met l'accent sur les configurations **advancées** que les équipes peuvent utiliser pour améliorer leur flux de travail.

31.1 Utilisation de librairies externes avec le code du robot

Avertissement : L'utilisation de librairies externes peut avoir un comportement inattendue avec votre code robot ! Il n'est pas recommandé à moins que vous ne sachiez ce que vous faites !

Souvent, une équipe peut vouloir ajouter des librairies externes Java ou C++ pour une utilisation avec son code robot. Cet article met l'emphase sur l'ajout de librairies Java à vos dépendances Gradle, ou les options disponibles pour les équipes C++.

31.1.1 Java

Note : Toutes les dépendances externes qui dépendent des librairies natives (JNI) ne fonctionneront probablement pas.

Il est assez simple d'ajouter des dépendances externes avec Java. Il vous suffit d'ajouter les `repositories` (référentiels) et les `dependencies` (dépendances) requises.

Les projets de robot par défaut n'ont pas de bloc `repositories {}` dans le fichier `build.gradle`. Vous devrez les ajouter vous-même. Au-dessus du bloc `dependencies {}`, veuillez ajouter ce qui suit :

```
repositories {
   mavenCentral()
```

(suite sur la page suivante)

(suite de la page précédente)

```
    ...
}
```

`mavenCentral()` peut être remplacé par n'importe quel référentiel que la librairie que vous souhaitez importer utilise. Maintenant, vous devez ajouter la dépendance à la librairie elle-même. Ceci est fait en ajoutant la ligne nécessaire à votre bloc `dependencies {}`. L'exemple ci-dessous présente l'ajout d'Apache Commons à votre projet Gradle.

```
dependencies {
    implementation 'org.apache.commons:commons-lang3:3.6'
    ...
}
```

Maintenant, vous lancez une compilation et assurez-vous que les dépendances sont téléchargées. Intellisense peut ne pas fonctionner correctement jusqu'à ce qu'une compilation soit en cours d'exécution !

31.1.2 C++

L'ajout de dépendances C++ à votre projet de robot n'est pas une tâche triviale en raison de la nécessité de compiler pour le roboRIO. Vous avez deux options.

1. Copiez le code source de la librairie souhaitée dans votre projet de robot.
2. Utilisez le modèle vendordep comme exemple et créez un vendordep.

Copie du code source

Il vous suffit de copier la source et/ou les en-têtes nécessaires dans votre projet de robot. Vous pouvez alors configurer toutes les args plate-forme nécessaires comme ci-dessous :

```
nativeUtils.platformConfigs.named("linuxx86-64").configure {
    it.linker.args.add('-lstdc++fs') // links in C++ filesystem library
}
```

Création d'un Vendordep

Veuillez suivre les instructions dans le référentiel vendordep.

31.2 Configurer l'Intégration Continue pour votre code de robot en utilisant GitHub Actions

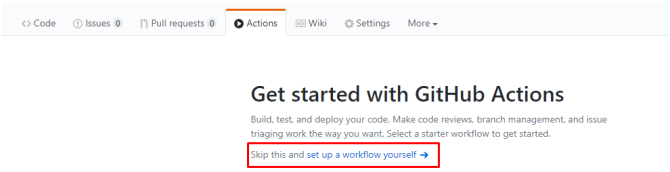
Un aspect important du travail dans un environnement d'équipe est de pouvoir tester le code qui est poussé vers un dépôt central tel que GitHub. Par exemple, un gestionnaire de projet ou un développeur principal peut vouloir exécuter un ensemble de tests unitaires avant de fusionner sa contribution ou peut vouloir s'assurer que tout le code de la branche principale d'un dépôt est en bon état de fonctionnement.

GitHub Actions est un service qui permet aux équipes et aux individus de créer et d'exécuter des tests unitaires sur le code sur divers branches et sur les demandes de type pull Ces types de services sont plus communément appelés services «d'intégration continue», ou CI. Ce tutoriel vous montrera comment configurer des actions GitHub sur des projets de code de robot.

Note : Ce tutoriel suppose que le code robot de votre équipe est hébergé sur GitHub. Pour une introduction à Git et GitHub, s'il vous plaît consulter *introduction guide*.

31.2.1 Création de l'action

Les instructions pour exécuter le processus CI sont stockées dans un fichier YAML. Pour le créer, cliquez sur l'onglet « Actions » en haut de votre référentiel de données. Cliquez ensuite sur l'hyperlien « set up a workflow yourself ».



Vous allez maintenant être accueilli par un éditeur de texte. Remplacez tout le texte par défaut par ce qui suit :

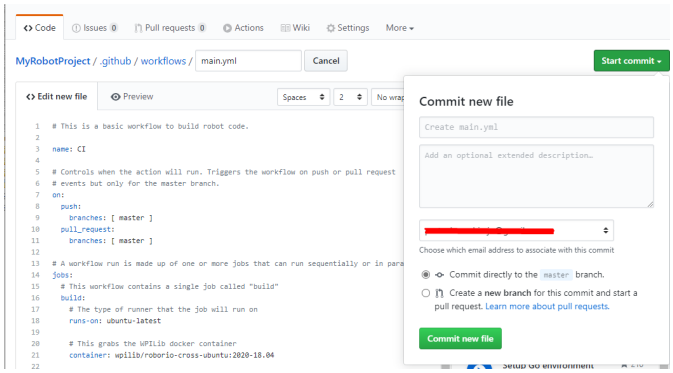
```yaml
# This is a basic workflow to build robot code.

name: CI

# Controls when the action will run. Triggers the workflow on push or pull request
# events but only for the main branch.
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

# A workflow run is made up of one or more jobs that can run sequentially or in␣
↪parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # This grabs the WPILib docker container
    container: wpilib/roborio-cross-ubuntu:2024-22.04

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
    # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
    - uses: actions/checkout@v4

    # Declares the repository safe and not under dubious ownership.
```

(suite sur la page suivante)

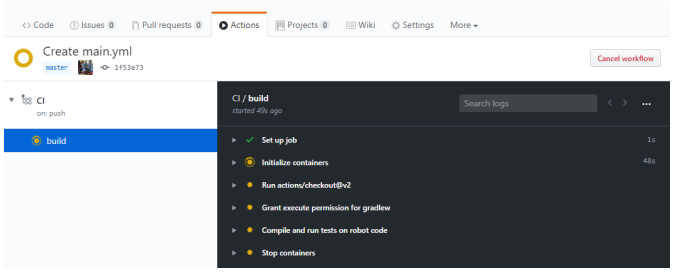
(suite de la page précédente)

```
    - name: Add repository to git safe directories
      run: git config --global --add safe.directory $GITHUB_WORKSPACE

    # Grant execute permission for gradlew
    - name: Grant execute permission for gradlew
      run: chmod +x gradlew

    # Runs a single command using the runners shell
    - name: Compile and run tests on robot code
      run: ./gradlew build
```

Enregistrez ensuite les modifications en cliquant sur le bouton « Start commit » dans le coin supérieur droit de l'écran. Vous pouvez modifier le message de validation par défaut si vous le souhaitez. Cliquez ensuite sur le bouton vert « Commit new file ».



GitHub exécute désormais automatiquement une compilation à chaque fois qu'un commit est envoyé vers la branche principale ou qu'un pull request est ouvert. Pour surveiller l'état de toute compilation, vous pouvez cliquer sur l'onglet « Actions » en haut de l'écran.



31.2.2 Une analyse du fichier Actions YAML

Voici une analyse du fichier YAML ci-dessus. Bien qu'une compréhension stricte de chaque ligne ne soit pas requise, un certain niveau de compréhension vous aidera à ajouter plus de fonctionnalités et à déboguer les problèmes potentiels qui pourraient survenir.

```
# Controls when the action will run. Triggers the workflow on push or pull request
# events but only for the main branch.
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
```

Ce bloc de code dicte quand l'Action s'exécutera. Actuellement, l'action s'exécute lorsque les commits sont poussés vers la branche principale ou lorsque les pull requests sont ouvertes par rapport à la branche principale.

```
# A workflow run is made up of one or more jobs that can run sequentially or in␣
↪parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # This grabs the WPILib docker container
    container: wpilib/roborio-cross-ubuntu:2024-22.04
```

Chaque Action est composée d'une ou de plusieurs tâches qui s'exécutent séquentiellement (l'une après l'autre) ou en parallèle (en même temps). Dans notre exemple, il n'y a qu'une seule tâche « build ».

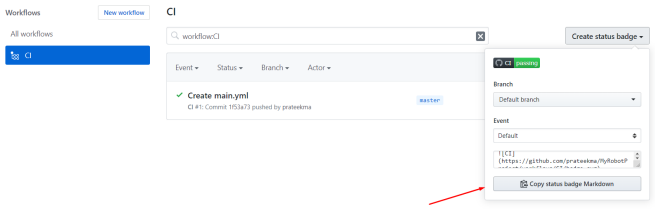
Nous spécifions que nous voulons que la tâche s'exécute sur une machine virtuelle Ubuntu et dans un Docker container qui contient le JDK, le compilateur C++ et les chaînes de compilation roboRIO .

```
# Steps represent a sequence of tasks that will be executed as part of the job
steps:
# Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
- uses: actions/checkout@v4

# Declares the repository safe and not under dubious ownership.
- name: Add repository to git safe directories
  run: git config --global --add safe.directory $GITHUB_WORKSPACE

# Grant execute permission for gradlew
- name: Grant execute permission for gradlew
  run: chmod +x gradlew

# Runs a single command using the runners shell
- name: Compile and run tests on robot code
  run: ./gradlew build
```

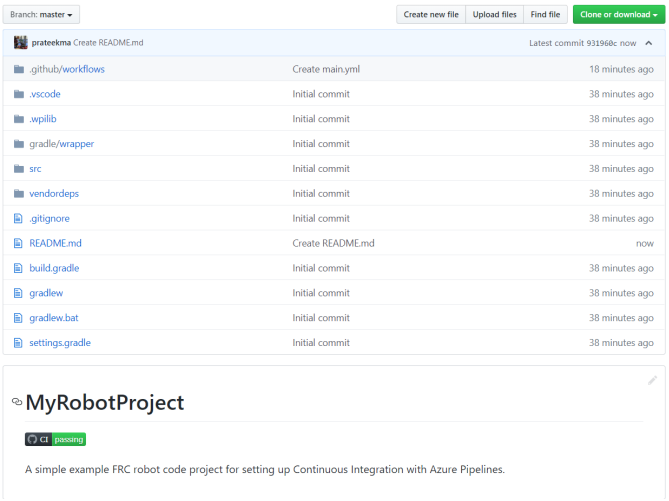
Each job has certain steps that will be executed. This job has four steps. The first step involves checking out the repository to access the robot code. The second step is a workaround for a GitHub Actions issue. The third step involves giving the virtual machine permission to execute gradle tasks using `./gradlew`. The final step runs `./gradlew build` to compile robot code and run any unit tests.

31.2.3 Ajout d'un Build Status Badge à un fichier README.md

Il est utile d'ajouter un badge d'état CI en haut du fichier README de votre dépôt pour vérifier rapidement l'état de la dernière version sur la principale. Pour ce faire, cliquez sur l'onglet « Actions » en haut de l'écran et sélectionnez l'onglet « CI » sur le côté gauche de l'écran. Cliquez ensuite sur le bouton « Create status badge » en haut à droite et copier le code Markdown du badge d'état.



Enfin, collez le code Markdown que vous avez copié en haut de votre fichier README, validez et envoyez vos modifications. Maintenant, vous devriez voir le badge d'état Actions GitHub sur votre page de référentiel de données principale.



31.3 Utilisation d'un formateur de code

Les formateurs de code existent pour s'assurer que le style de code écrit est cohérent dans l'ensemble du code de base. Ceci est utilisé dans de nombreux grands projets ; d'Android à OpenCV. Les équipes peuvent souhaiter ajouter un formateur dans tout leur code robot pour s'assurer que le code de base maintient la lisibilité et la cohérence tout le long.

Pour cet article, nous mettons l'emphase sur l'utilisation de Spotless pour les équipes Java et wpiformat pour les équipes C++.

31.3.1 Spotless

Configuration

Des modifications `build.gradle` nécessaires sont requises pour que Spotless soit fonctionnel. Dans le bloc `plugins {}` de votre `build.gradle`, ajoutez le plugin Spotless de sorte qu'il semble similaire à celui apparaissant ci-dessous.

```
plugins {
   id "java"
   id "edu.wpi.first.GradleRIO" version "2024.1.1"
   id 'com.diffplug.spotless' version '6.20.0'
}
```

Ensuite, assurez-vous d'ajouter un bloc `spotless {}` nécessaire pour configurer correctement Spotless. Celui-ci peut simplement être placé à la fin de votre fichier `build.gradle`.

```
spotless {
   java {
      target fileTree('.') {
            include '**/*.java'
            exclude '**/build/**', '**/build-*/**'
      }
      toggleOffOn()
      googleJavaFormat()
      removeUnusedImports()
      trimTrailingWhitespace()
      endWithNewline()
   }
   groovyGradle {
      target fileTree('.') {
            include '**/*.gradle'
            exclude '**/build/**', '**/build-*/**'
      }
      greclipse()
      indentWithSpaces(4)
      trimTrailingWhitespace()
      endWithNewline()
   }
   format 'xml', {
      target fileTree('.') {
            include '**/*.xml'
            exclude '**/build/**', '**/build-*/**'
      }
      eclipseWtp('xml')
      trimTrailingWhitespace()
      indentWithSpaces(2)
      endWithNewline()
   }
   format 'misc', {
      target fileTree('.') {
            include '**/*.md', '**/.gitignore'
            exclude '**/build/**', '**/build-*/**'
      }
      trimTrailingWhitespace()
      indentWithSpaces(2)
      endWithNewline()
   }
}
```

Exécution de Spotless

Spotless peut être exécuté en utilisant la commande `./gradlew spotlessApply` qui appliquera toutes les options de mise en forme. Vous pouvez également spécifier une tâche spécifique en ajoutant simplement le nom du formateur. Un exemple est `./gradlew spotlessMiscApply`.

In addition to formatting code, Spotless can also ensure the code is correctly formatted ; this can be used by running `./gradlew spotlessCheck`. Thus, Spotless can be used as a *CI check*, as shown in the following GitHub Actions workflow :

```yaml
on: [push]
# A workflow run is made up of one or more jobs that can run sequentially or in␣
↪parallel
jobs:
  spotless:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest
    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - uses: actions/setup-java@v4
        with:
          distribution: 'zulu'
          java-version: 17
      - run: ./gradlew spotlessCheck
```

Explication des options

Chaque section `format` met en évidence le formatage des fichiers personnalisés dans le projet. Les `java` et `groovyGradle` sont nativement soutenus par Spotless, de sorte qu'ils sont définis différemment.

En décomposant ceci, on peut le diviser en plusieurs parties.

— Mise en forme de Java
— Mise en forme des fichiers Gradle
— Mise en forme des fichiers XML
— Mise en forme de fichiers divers

Ils sont tous similaires, à l'exception de quelques petites différences qui seront expliquées. L'exemple ci-dessous mettra l'emphase sur le bloc `java {}`

```
java {
   target fileTree('.') {
      include '**/*.java'
      exclude '**/build/**', '**/build-*/**'
   }
   toggleOffOn()
   googleJavaFormat()
   removeUnusedImports()
   trimTrailingWhitespace()
   endWithNewline()
}
```

Expliquons la signification de chacune des options.

```
target fileTree('.') {
   include '**/*.java'
   exclude '**/build/**', '**/build-*/**'
}
```

L'exemple ci-dessus indique à Spotless où se trouvent nos classes Java et d'exclure le dossier `build`. Les options restantes sont assez explicites.

— `toggleOffOn()` ajoute à Spotless la possibilité d'ignorer des parties spécifiques d'un projet. L'utilisation ressemble à ce qui suit

```
// format:off

public void myWeirdFunction() {

}

// format:on
```

- `googleJavaFormat()` indique à Spotless de mettre en forme selon le Guide de Style de Google
- `removeUnusedImports()` will remove any unused imports from any of your Java classes
- `trimTrailingWhitespace()` supprimera tout espace blanc supplémentaire à la fin de vos lignes
- `endWithNewline()` ajoutera un caractère newline à la fin de vos classes

Dans le bloc `groovyGradle`, il existe une option `greclipse`. C'est le formateur que Spotless utilise pour mettre en forme les fichiers gradle.

Additionnellement, il existe une option `eclipseWtp` dans le bloc `xml`. Il s'agit de « Gradle Web Tools Platform » et est le formateur servant à mettre en forme les fichiers `xml`. Les équipes qui n'utilisent pas de fichiers XML peuvent choisir de ne pas inclure cette configuration.

Note : Une liste complète des configurations est disponible dans le fichier Spotless README

Problèmes avec les fins de ligne

Spotless tentera d'appliquer des fins de ligne par OS, ce qui signifie que les commandes Git diffs changeront constamment si deux utilisateurs sont sur des OS différents (Unix vs Windows). Il est recommandé aux équipes qui contribuent au même dépôt à partir de plusieurs OS d'utiliser un fichier `.gitattributes`. Ce qui suit devrait suffire pour la manipulation des fins de ligne.

```
*.gradle text eol=lf
*.java text eol=lf
*.md text eol=lf
*.xml text eol=lf
```

31.3.2 wpiformat

Exigences

- Python 3.6 ou ultérieur

Vous pouvez installer wpiformat en tapant `pip3 install wpiformat` dans un terminal ou une invite de commande.

Utilisation

wpiformat peut être couru en tapant `wpiformat` dans une console. La mise en forme sera effectuée à l'aide de `clang-format`. Trois fichiers de configuration sont nécessaires (`.clang-format`, `.styleguide`, `.styleguide-license`). Ceux-ci doivent être présents dans la racine du projet.

- `.clang-format` : Télécharger
- `.styleguide-license` : Télécharger

Un exemple de guide de style est montré ci-dessous :

```
cppHeaderFileInclude {
   \.h$
   \.hpp$
   \.inc$
   \.inl$
}

cppSrcFileInclude {
   \.cpp$
}

modifiableFileExclude {
   gradle/
}
```

Note : Les équipes peuvent adapter `.styleguide` et `.styleguide-license` comme elles le souhaitent. Il est important que ceux-ci ne sont pas supprimés, car ils sont requis pour exécuter wpiformat !

You can turn this into a *CI check* by running `git --no-pager diff --exit-code HEAD`, as shown in the example GitHub Actions workflow below :

```yaml
name: Lint and Format

on:
  pull_request:
  push:
jobs:
  wpiformat:
    name: "wpiformat"
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: Fetch all history and metadata
        run: |
          git checkout -b pr
          git branch -f main origin/main
      - name: Set up Python 3.8
        uses: actions/setup-python@v5
        with:
          python-version: 3.8
      - name: Install wpiformat
        run: pip3 install wpiformat==2024.32
```

(suite sur la page suivante)

(suite de la page précédente)

```
      - name: Run
        run: wpiformat
      - name: Check output
        run: git --no-pager diff --exit-code HEAD
```

31.4 Tâches Gradlew

Cet article vise à mettre l'emphase sur les commandes gradle prises en charge par l'équipe WPILib pour une utilisation par l'utilisateur. Ces commandes peuvent être visualisées en tapant `./gradlew tasks` à la racine de votre projet de robot. Toutes les commandes affichées dans `./gradlew tasks` ne seront pas documentées ici, de même que les commandes qui ne sont pas prises en charge.

31.4.1 Tâches Build (Compilation)

`./gradlew build` - Assembles and tests this project. Useful for prebuilding your project without deploying to the roboRIO.

`./gradlew clean` - Deletes the build directory.

31.4.2 Tâches CompileCommands

`./gradlew generateCompileCommands` - Generate compile_commands.json for C++ programs. This is a configuration file that is supported by many Integrated Development Environments and build tools.

31.4.3 DeployUtils tasks

`./gradlew deploy` - Déployez tous les artefacts sur toutes les cibles. Cela permettra de déployer votre projet de robot sur les cibles disponibles (IE, roboRIO).

`./gradlew discoverRoborio` - Determine the address(es) of target roboRIO. This will print out the IP address of a connected roboRIO.

31.4.4 Tâches GradleRIO

`./gradlew $TOOL$` - Exécute l'outil `$TOOL$` (Remplace `$TOOL$` par le nom de l'outil. IE, Glass, Shuffleboard, etc)

`./gradlew $TOOL$Install` - Installs the java tool `$TOOL$` (Replace `$TOOL$` with the name of the tool. IE, Shuffleboard, SmartDashboard, etc)

`./gradlew InstallAllTools` - Installs all available tools. This excludes the development environment such as Visual Studio Code. It's the users requirement to ensure the required dependencies (Java) is installed. Only recommended for advanced users !

`./gradlew simulateExternalNativeRelease` - Simulate External Task for native executable. Exports a JSON file for use by editors / tools

`./gradlew simulateExternalJavaRelease` - Simulate External Task for Java/Kotlin/JVM. Exports a JSON file for use by editors / tools

`./gradlew simulateJava` - Lancement la simulation pour les projets Java

`./gradlew simulateNative` - Lance la simulation pour les projets C++

`./gradlew vendordep` - Installe le fichier JSON vendordep à partir d'une URL ou d'une installation locale. Consulter *Librairies tierces*

31.5 Inclure les données Git dans Deploy

This article will explain how to include metadata from Git in robot code using the gversion Gradle plugin. This generates a file which can be used for accessing Git metadata in robot code. This can be used to track what revision of code is on the robot, such as by printing or logging it.

Note : For Python teams, Git metadata is always copied to your robot during the deploy process. You can use `wpilib.deployinfo.getDeployData()` to retrieve the stored information.

31.5.1 Installing gversion

To install gversion add the following line to the plugins block of `build.gradle`. This tells Gradle to use gversion in the project.

```
plugins {
    // ...
    id "com.peterabeles.gversion" version "1.10"
}
```

In order to generate the file when building the project, add the following block to the bottom of `build.gradle`.

```
project.compileJava.dependsOn(createVersionFile)
gversion {
  srcDir       = "src/main/java/"
  classPackage = "frc.robot"
  className    = "BuildConstants"
  dateFormat   = "yyyy-MM-dd HH:mm:ss z"
  timeZone     = "America/New_York" // Use preferred time zone
  indent       = "  "
}
```

The `srcDir`, `classPackage`, and `className` parameters tell the plugin where to put the manifest file, and what to name it. The `timeZone` field can be set to your team's time zone based on this list of timezone IDs. The `dateFormat` parameter is based on this Java class.

To test this, run a build of your project through the WPILib menu in the top right. Once the code has finished building, there should be a file called `BuildConstants.java` in your `src/main/java` folder. The following is an example of what this file should look like :

```java
package frc.robot;

/**
* Automatically generated file containing build version information.
*/
public final class BuildConstants {
  public static final String MAVEN_GROUP = "";
  public static final String MAVEN_NAME = "GitVersionTest";
  public static final String VERSION = "unspecified";
  public static final int GIT_REVISION = 1;
  public static final String GIT_SHA = "fad108a4b1c1dcdfc8859c6295ea64e06d43f557";
  public static final String GIT_DATE = "2023-10-26 17:38:59 EDT";
  public static final String GIT_BRANCH = "main";
  public static final String BUILD_DATE = "2023-10-27 12:29:57 EDT";
  public static final long BUILD_UNIX_TIME = 1698424197122L;
  public static final int DIRTY = 0;

  private BuildConstants(){}
}
```

DIRTY indicates whether there are uncommitted changes in the project. A value of 0 indicates a clean repository, a value of 1 indicates that there are uncommitted changes, and a value -1 indicates an error.

Ignorer les fichiers générés avec Git

These files are regenerated every time code is built or deployed, so it isn't necessary to track them with Git. The aptly named .gitignore file tells Git not to track any listed files and should exist by default in any project generated by the WPILib VS Code extension. Below is the line you should add to `.gitignore` to ignore the generated file :

```
src/main/java/frc/robot/BuildConstants.java
```

31.6 Utiliser des arguments de compilation

Compiler arguments allow us to change the behavior of our compiler. This includes making warnings into errors, ignoring certain warnings and choosing optimization level. When compiling code a variety of flags are already included by default which can be found here. Normally it could be proposed that the solution is to pass them in as flags when compiling our code but this doesn't work in GradleRIO. Instead modify the build.gradle.

Avertissement : Modifier les arguments est dangereux et peut causer un comportement inattendu.

31.6.1 C++

Plateformes

Different compilers and different platforms use a variety of different flags. Therefore to avoid breaking different platforms with compiler flags configure all flags per platform. The platforms that are supported are listed here

Configuration pour une plateforme

```
nativeUtils.platformConfigs.named('windowsx86-64').configure {
  it.cppCompiler.args.add("/utf-8")
}
```

native-utils est utilisé pour configurer la plateforme, dans ce cas, *windowsx86-64*. Cela peut être remplacé par n'importe quelle plateforme répertoriée dans la section précédente. Ensuite, des arguments, tels que */utf-8*, sont ajoutés au compilateur C++.

31.6.2 Java

Les arguments peuvent également être configurés pour Java. Cela peut être accompli en modifiant *build.gradle* et en ajoutant des arguments à *FRCJavaArtifact*. Un exemple est présenté ci-dessous.

```
frcJava(getArtifactClass('FRCJavaArtifact')) {
  jvmArgs.add("-XX:+DisableExplicitGC")
}
```

31.7 Profiling with VisualVM

This document is intended to familiarize the reader with the diagnostic tool that is VisualVM for debugging Java robot programs. VisualVM is a tool for profiling JVM based applications, such as viewing why an application is using a large amount of memory. This document assumes the reader is familiar with the *risks* associated with modifying their robot `build.gradle`. This tutorial also assumes that the user knows basic terminal/commandline knowledge.

31.7.1 Unpacking VisualVM

To begin, download VisualVM and unpack it to the WPILib installation folder. The folder is located at `~/wpilib/` where ~ indicates the users home directory. On Windows, this is `C:\Users\Public\wpilib`.

31.7.2 Setting up Gradle

GradleRIO supports passing JVM launch arguments, and this is what is necessary to enable remote debugging. Remote debugging is a feature that allows a local machine (such as the user's desktop) to view important information about a remote target (in our case, a roboRIO). To begin, locate the `frcJava` code block located in the projects `build.gradle`. Below is what is looks like.

```
deploy {
    targets {
        roborio(getTargetTypeClass('RoboRIO')) {
            // Team number is loaded either from the .wpilib/wpilib_preferences.json
            // or from command line. If not found an exception will be thrown.
            // You can use getTeamOrDefault(team) instead of getTeamNumber if you
            // want to store a team number in this file.
            team = project.frc.getTeamNumber()
            debug = project.frc.getDebugOrDefault(false)

            artifacts {
                // First part is artifact name, 2nd is artifact type
                // getTargetTypeClass is a shortcut to get the class type using a string

                frcJava(getArtifactTypeClass('FRCJavaArtifact')) {
                }

                // Static files artifact
                frcStaticFileDeploy(getArtifactTypeClass('FileTreeArtifact')) {
                    files = project.fileTree('src/main/deploy')
                    directory = '/home/lvuser/deploy'
                }
            }
        }
    }
}
```

We will be replacing the highlighted lines with :

```
frcJava(getArtifactTypeClass('FRCJavaArtifact')) {
   // Enable VisualVM connection
   jvmArgs.add("-Dcom.sun.management.jmxremote=true")
   jvmArgs.add("-Dcom.sun.management.jmxremote.port=1198")
   jvmArgs.add("-Dcom.sun.management.jmxremote.local.only=false")
   jvmArgs.add("-Dcom.sun.management.jmxremote.ssl=false")
   jvmArgs.add("-Dcom.sun.management.jmxremote.authenticate=false")
   jvmArgs.add("-Djava.rmi.server.hostname=10.XX.XX.2") // Replace XX.XX with team number
}
```

We are adding a few arguments here. In order :

— Enable remote debugging
— Set the remote debugging port to 1198
— Allow listening from remote targets
— Disable SSL authentication being required
— Set the hostname to the roboRIOs team number. Be sure to replace this.

Important : The hostname when connected via USB-B should be `172.22.11.2`.

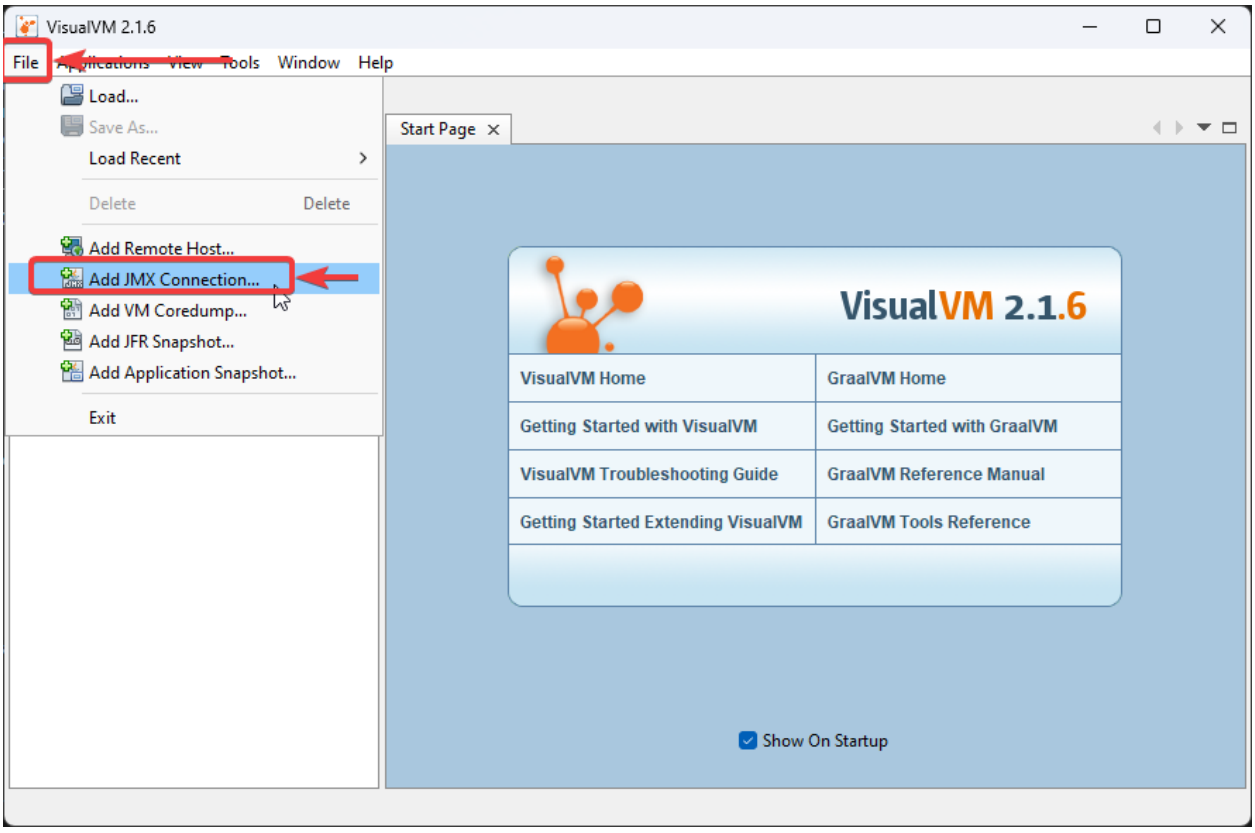
31.7.3 Running VisualVM

Launching VisualVM is done via the commandline with a few parameters. First, we navigate to the directory containing VisualVM. Then, launch it with parameters, passing it the WPILib JDK path. On a Windows machine, it looks like the following :

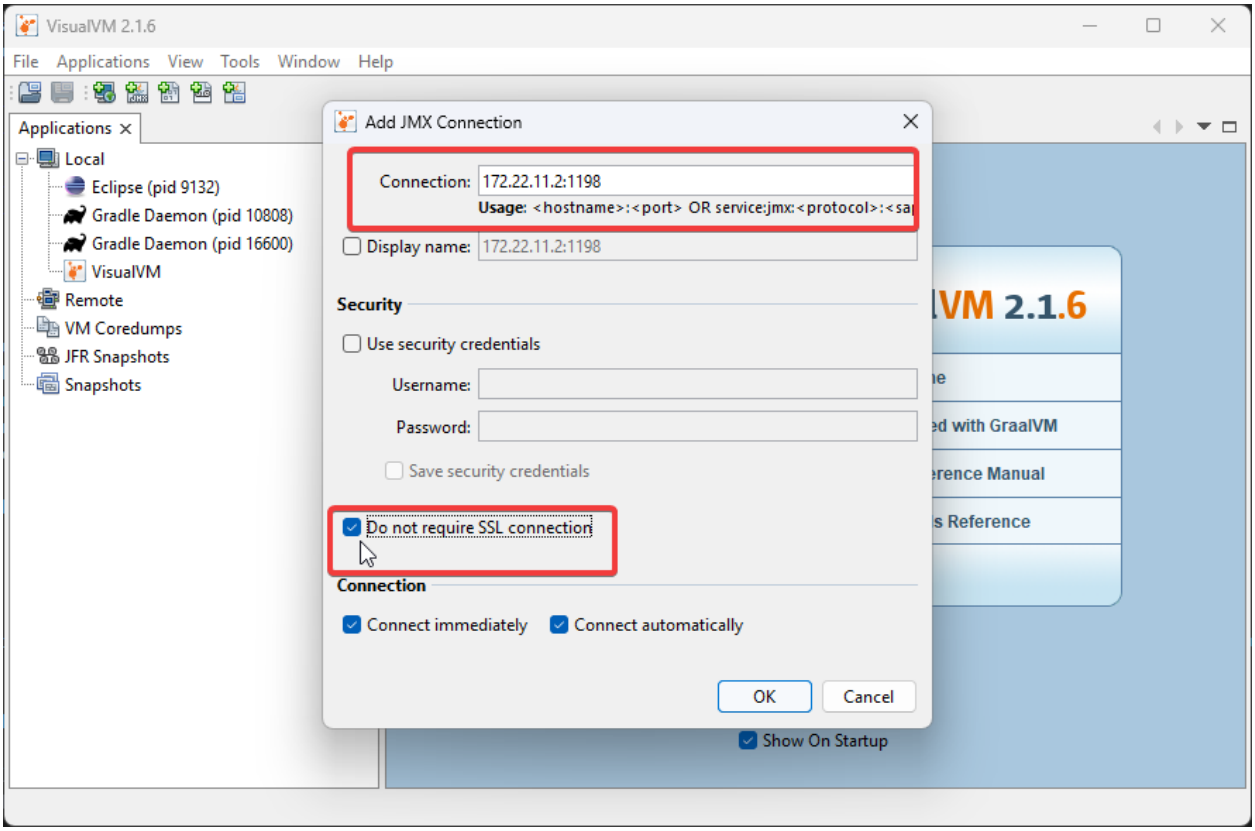
```
cd "C:\Users\Public\wpilib\visualvm_217\bin"
./visualvm --jdkhome "C:\Users\Public\wpilib\2024\jdk"
```

Important : The exact path `visualvm_217` may vary and depends on the version of VisualVM downloaded.

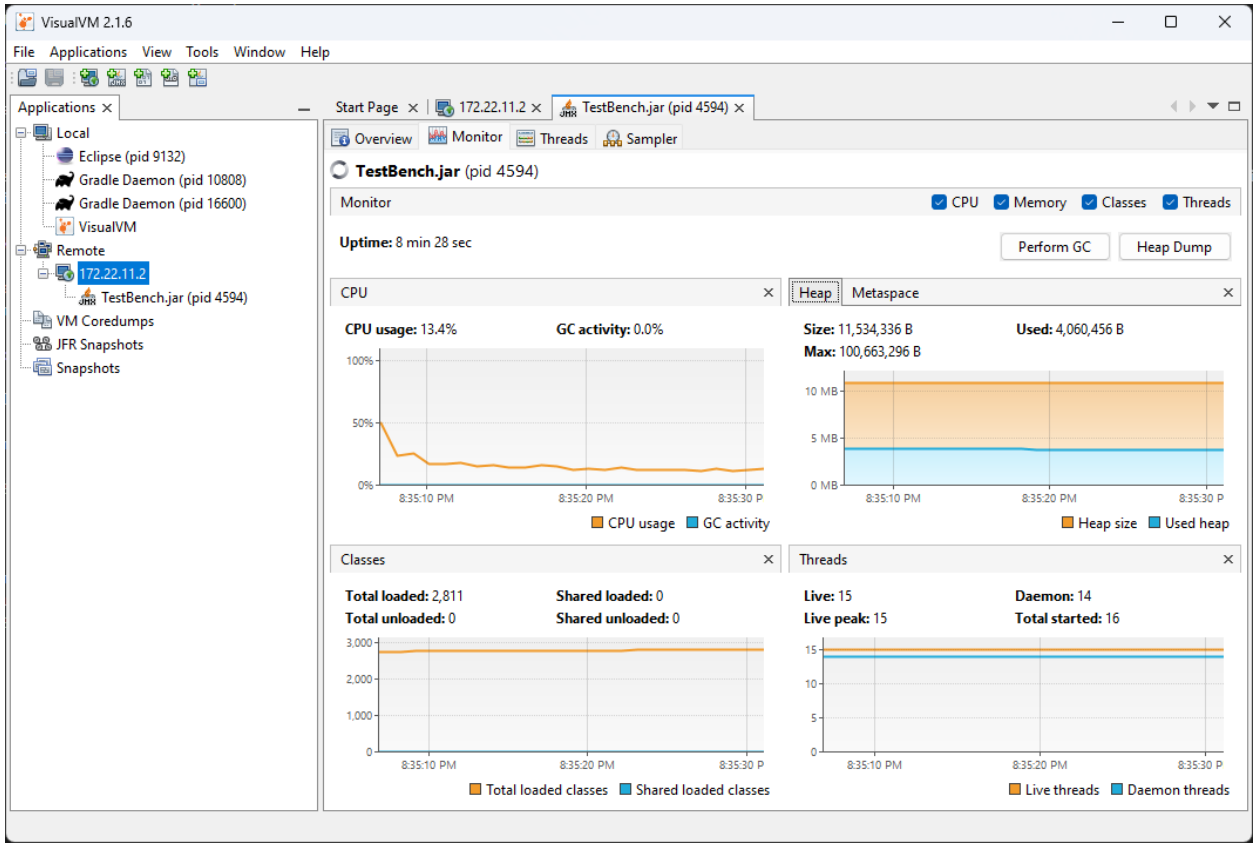
This should launch VisualVM. Once launched, open the *Add JMX Connection* dialog.



Once opened, configure the connection details and hostname. Ensure that *Do not require SSL connection* is ticked.

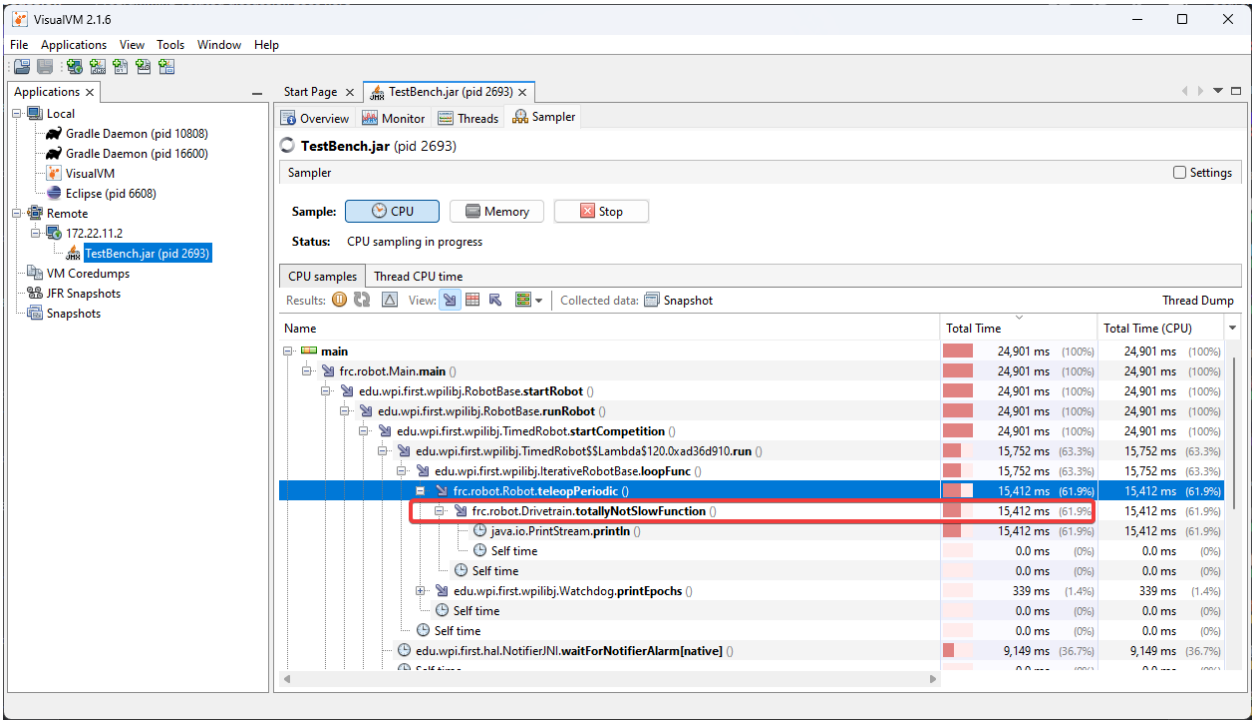


If correctly done, a new menu option in the left-hand sidebar will appear. Clicking on it will show you a detailed dashboard of the running JVM application.



31.7.4 Analyzing Function Timings

An important feature of VisualVM is the ability to view how much time a specific function is taking up. This is *without* having a code debugger attached. To begin, click on the *Sampler* tab and then click on *CPU*. This will immediately give a breakdown of what functions are taking CPU time.



The above screenshot shows a breakdown of the total time a specific function takes. You can see that `totallyNotSlowFunction()` accounts for `61.9%` of the robot program CPU time. We can then correlate this to our robot program. In `totallyNotSlowFunction()`, we have the following code.

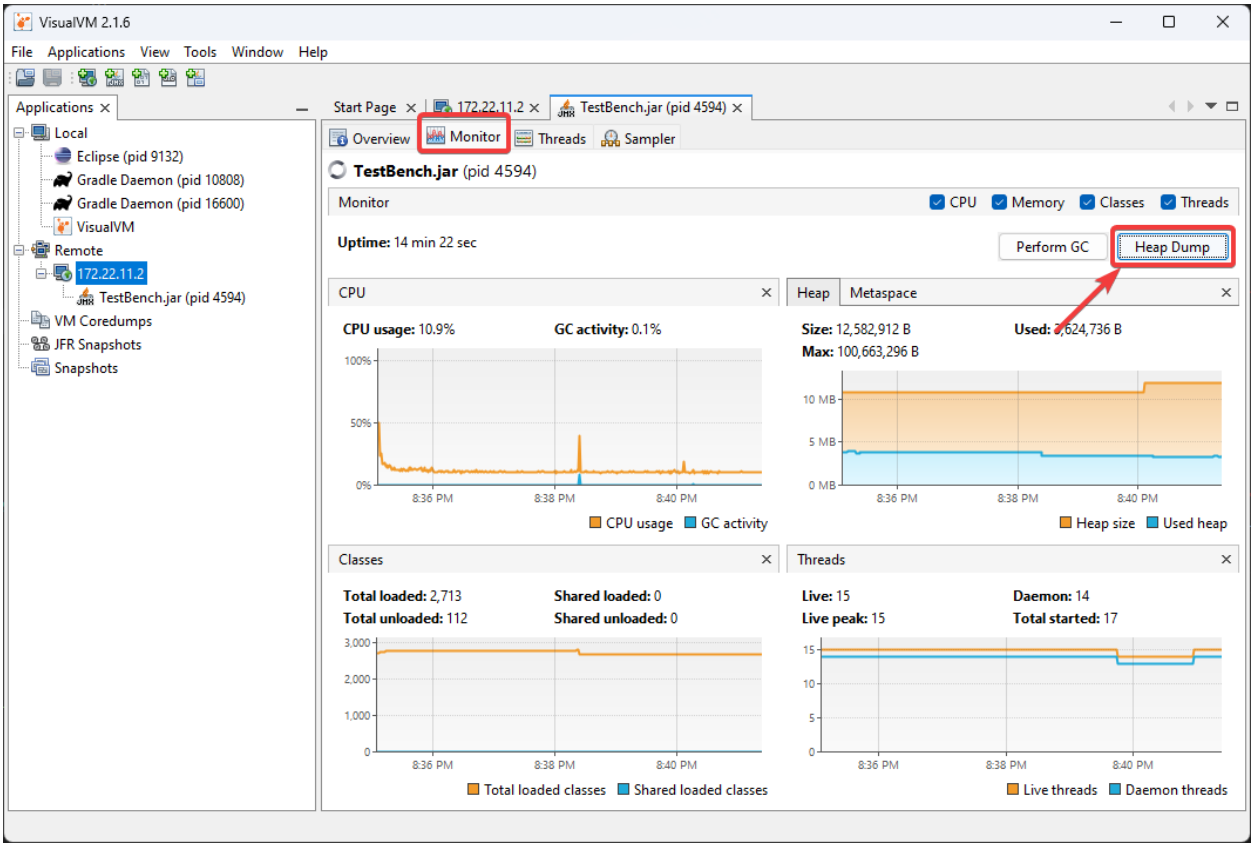
```
public static void totallyNotSlowFunction() {
   for (int i = 0; i < 2000; i++) {
      System.out.println("HAHAHAHA");
   }
}
```

In this code snippet, we can identify 2 major causes of concern. A long running `for` loop blocks the rest of the robot program from running. Additionally, `System.out.println()` calls on the roboRIO are typically quite expensive. We found this information by profiling the Java application on the roboRIO!

31.7.5 Creating a Heap Dump

Besides viewing the remote systems CPU and memory usage, VisualVM is most useful by creating a **Heap Dump**. When a Java object is created, it resides in an area of memory called the heap. When the heap is full, a process called garbage collection begins. Garbage collection can be a common cause of loop overruns in a traditional Java robot program.

To begin, ensure you are on the *Monitor* tab and click *Heap Dump*.



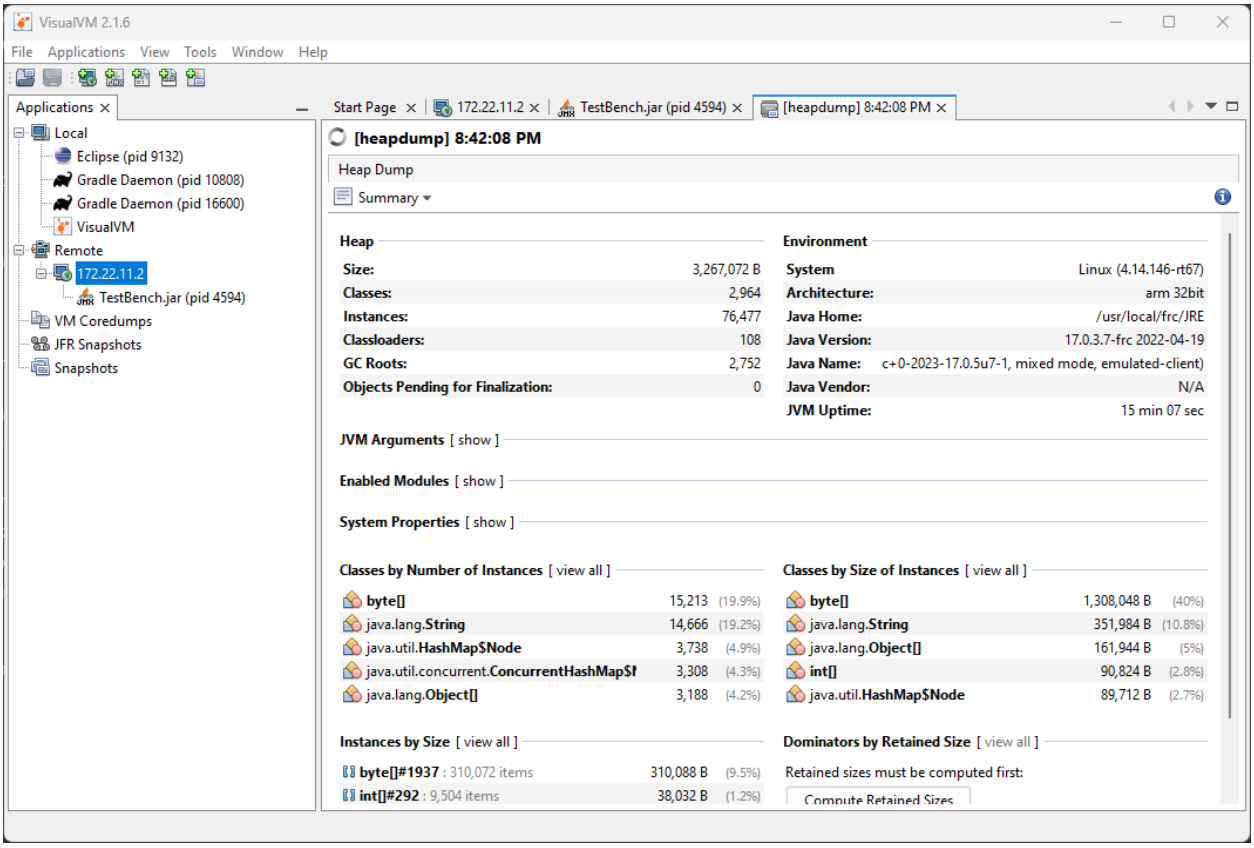
This heap dump will be stored on the target system (roboRIO) and must be retrieved using SFTP. See *this article* for information on retrieving the dump from the roboRIO.

Once downloaded, the dump can be analyzed with VisualVM.

Astuce : You can also *configure the JVM to take a heap dump automatically when your robot code runs out of memory*.

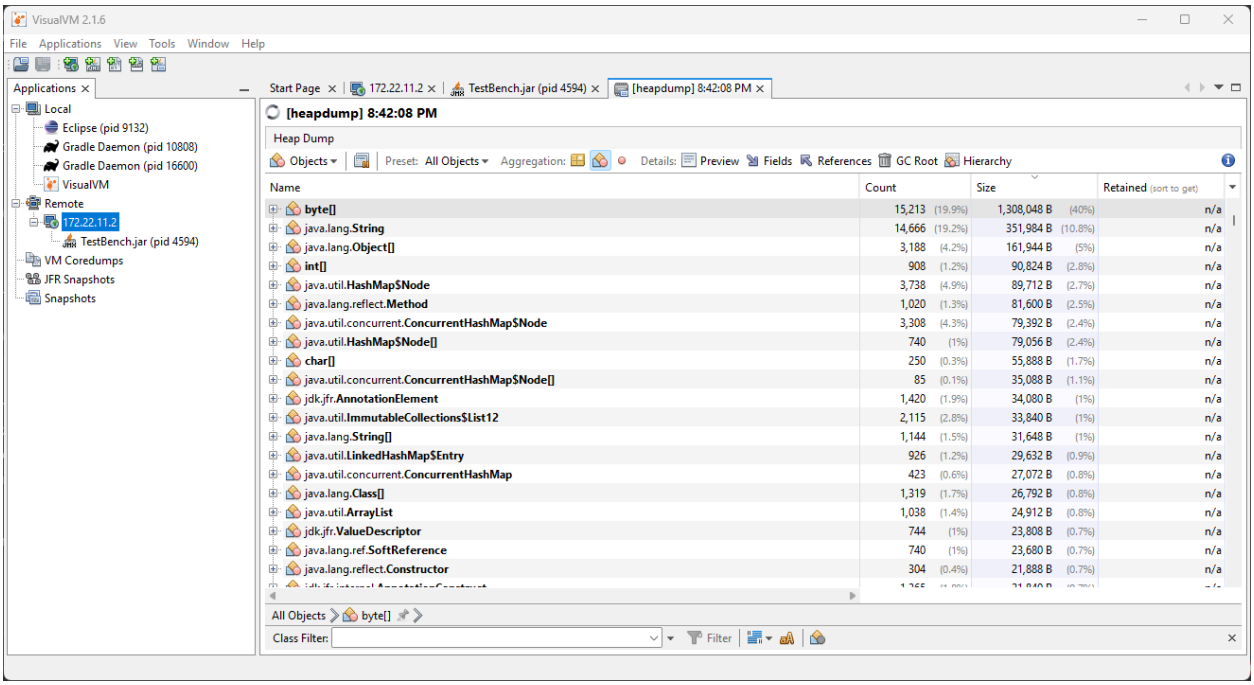
31.7.6 Analyzing a Heap Dump

Reopen VisualVM if closed using the previous instructions. Then click on *File* and *Load*. Navigate to the retrieved dump file and load it.

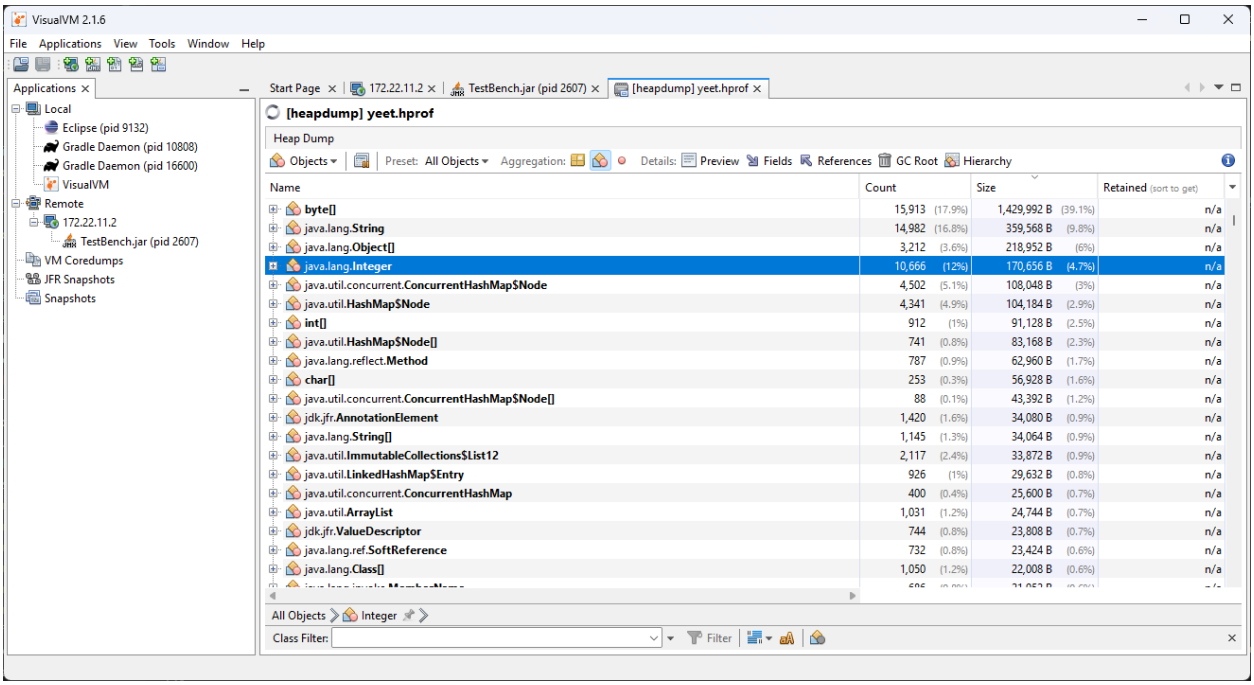


Clicking on *Summary* and selecting *Objects* instead will show a breakdown of objects by quantity. The below screenshot showcases a completely empty robot program, and then one that creates an million large `ArrayList` of integers.

Blank robot program :



with an `ArrayList` of ~10000 integers.



31.7.7 Additional Info

For more information on VisualVM, check out the VisualVM documentation pages.

32 Commandes Avancées

Cette section couvre les fonctionnalités de commandes avancées dans la WPILib, comme les algorithmes de commande rétroactive, commande prédictive et suivi de trajectoire.

32.1 Présentation vidéo du mode Autonome FRC basée sur un modèle de validation.

Pendant la « Conférence de printemps RSN, présentée par WPI » en 2020, Tyler Veness de l'équipe WPILib a fait une présentation sur la validation basée sur le modèle du mode autonome en FRC®.

The link to the presentation is available here.

32.2 Introduction aux commandes avancées

32.2.1 Les notions fondamentales d'un système de contrôle

Note : This article includes sections of Controls Engineering in FRC by Tyler Veness with permission.

The Need for Control Systems

Les systèmes de contrôle sont présents tout autour de nous et nous interagissons quotidiennement avec eux. Une petite liste de ceux que vous avez peut-être vus comprend : des radiateurs et des climatiseurs avec thermostats, un régulateur de vitesse pour une voiture, ou son système de freinage antiblocage (ABS), et la modulation de la vitesse du ventilateur sur les ordinateurs portables modernes. Les systèmes de contrôle surveillent ou contrôlent le comportement de systèmes comme ceux-ci et peuvent être activées par des opérateurs humains (contrôle manuel), ou uniquement par des machines (contrôle automatique).

All of these examples have a mechanism which does useful work, but cannot be *directly* commanded to the state that is desired.

For example, an air conditioner's fans and compressor have no mechanical or electrical input where the user specifies a temperature. Rather, some additional mechanism must compare the current air temperature to some setpoint, and choose how to cycle the compressor and fans on and off to achieve that temperature.

Similarly, an automobile's engine and transmission have no mechanical lever which directly sets a particular speed. Rather, some additional mechanism must measure the current speed of the vehicle, and adjust the transmission gear and fuel injected into the cylinders to achieve the desired vehicle speed.

Controls Engineering is the study of how to design those additional mechanisms to bridge the gap from what the user wants a mechanism to do, to how the mechanism is actually manipulated.

Comment pouvons-nous prouver que les contrôleurs en boucle fermée sur une voiture autonome, par exemple, se comporteront en toute sécurité et répondront aux spécifications de performance souhaitées en présence d'incertitude ? La théorie du contrôle est une application de l'algèbre et de la géométrie utilisée pour analyser et prédire le comportement des systèmes, les faire réagir comme nous le voulons et les rendre robustes aux perturbations et à l'incertitude.

L'ingénierie des contrôles est, en termes simples, le processus d'ingénierie appliqué à la théorie du contrôle. En tant que tel, c'est plus que des mathématiques appliquées. Alors que la théorie du contrôle a de belles mathématiques derrière elle, l'ingénierie des contrôles est une discipline d'ingénierie comme les autres qui est pleine de compromis. Les solutions que la théorie du contrôle donne doivent toujours être vérifiées et conformes à nos spécifications de performance. Nous n'avons pas besoin d'être parfaits ; nous devons juste être assez bons pour répondre à nos spécifications.

Nomenclature

La plupart des ressources pour les sujets d'ingénierie avancée supposent un niveau de connaissances bien supérieur à ce qui est nécessaire. Une partie du problème est l'utilisation du jargon (langage technique). Bien que ce langage communique efficacement les idées à ceux qui travaillent dans ce domaine, les initiés qui ne le connaissent pas sont perdues.

Le système ou la série de senseurs et actuateurs contrôlés par un système de commande est appelé *Usine*, (Plant, en anglais). Un contrôleur est utilisé pour amener l'état actuel de l'Usine à un état souhaité (la référence). Les contrôleurs qui n'incluent pas d'informations mesurées à partir de la sortie de l'Usine sont appelés contrôleurs en boucle ouverte.

Les contrôleurs qui intègrent les informations renvoyées par la sortie de l'Usine sont appelés contrôleurs en boucle fermée ou contrôleurs de rétroaction.

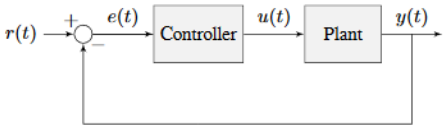


Figure 1.1: Control system nomenclature for a closed-loop system

$r(t)$	reference	$u(t)$	control input
$e(t)$	error	$y(t)$	output

Note : L'entrée et la sortie d'un système sont définies du point de vue de l'installation. Le contrôleur de rétroaction négative illustré conduit à zéro la différence entre la référence et la sortie, également connue sous le nom d'erreur.

Qu'est-ce que le gain ?

Gain est une valeur proportionnelle qui montre la relation entre l'amplitude d'un signal d'entrée et l'amplitude d'un signal de sortie à l'état stationnaire (Steady-state). De nombreux systèmes contiennent une méthode par laquelle le gain peut être modifié, fournissant plus ou moins de «puissance» au système.

La figure ci-dessous montre un système avec une entrée et une sortie hypothétiques. Étant donné que la sortie est deux fois l'amplitude de l'entrée, le système a un gain de deux.

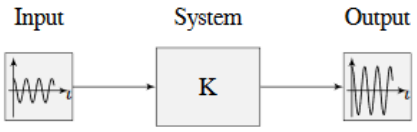


Figure 1.2: Demonstration of system with a gain of $K = 2$

What is a Model ?

A *model* of your mechanism is a mathematical description of its behavior. Specifically, this mathematical description must define the mechanism's inputs and outputs, and how the output values change over time as a function of its input values.

The mathematical description is often just simple algebra equations. It can also include some linear algebra, matrices, and differential equations. WPILib provides a number of classes to help simplify the more complex math.

Classical Mechanics defines many of the equations used to build up models of system behavior. Many of the values inside those equations can be determined by doing experiments on the mechanism.

Schémas fonctionnels

Lors de la conception ou de l'analyse d'un système de contrôle, il est utile de le modéliser graphiquement. Des schémas fonctionnels sont utilisés à cet effet. Ils peuvent être manipulés et simplifiés systématiquement.

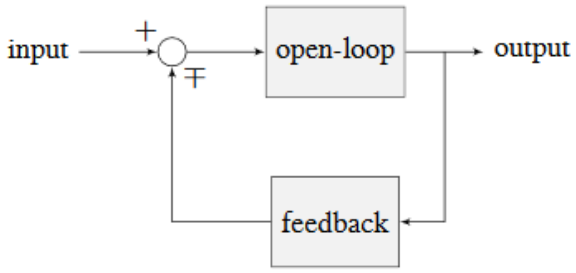


Figure 1.3: Block diagram with nomenclature

Le gain en boucle ouverte est le gain total du nœud de somme à l'entrée (le cercle) vers la branche de sortie. ce serait le gain du système si la boucle de rétroaction était déconnectée. Le gain de rétroaction est le gain total de la sortie vers le nœud de somme d'entrée. La sortie d'un nœud de somme est calculée en additionnant toutes ses entrées.

La figure ci-dessous est un schéma illustrant une configuration de rétroaction, avec la bonne nomenclature.

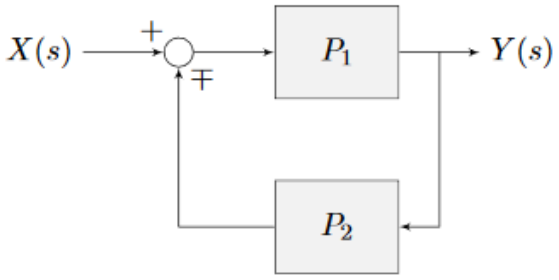


Figure 1.4: Feedback block diagram

∓ signifie « moins ou plus » où un signe moins représente une rétroaction négative.

A Note on Dimensionality

For the purposes of the introductory section, all systems and controllers (except feedforward controllers) are assumed to be « single-in, single-out » (SISO) - this means they only map single values to single values. For example, a DC motor is considered to take an *input* of a single scalar value (voltage) and yield an *output* of only a single scalar value in return (either position or velocity). This forces us to consider *position controllers* and *velocity controllers* as separate entities - this is sometimes source of confusion in situations when we want to control both (such as when following a motion profiles). Limiting ourselves to SISO systems also means that we are unable to analyze more-complex « multiple-in, multiple-out » (MIMO) systems like drivetrains that cannot be represented with a single state (there are at least two independent sets of wheels in a drive).

Nonetheless, we restrict ourselves to SISO systems here to be able to present the following tutorials in terms of the PID Controller formalism, which is commonly featured in introductory course material and has extensive documentation and many available implementations.

The *state-space* formalism is an alternate way to conceptualize these systems which allows us to easily capture interactions between different quantities (as well as simultaneously represent multiple aspects of the same quantity, such as position and velocity of a motor). It

does this, roughly, by replacing the single-dimensional scalars (e.g. the *gain*, *input*, and *output*) with multi-dimensional vectors. In the state-space formalism, the equivalent of a « PID » controller is a vector-proportional controller on a single vector-valued mechanism state, with a single *gain* vector (instead of three different *gain* scalars).

If you remember that a state-space controller is really just a PID controller written with dense notation, many of the principles covered in this set of introductory articles will transfer seamlessly to the case of state-space control.

32.2.2 Picking a Control Strategy

Note : This article includes sections of Controls Engineering in FRC by Tyler Veness with permission.

When designing a control algorithm for a robot mechanism, there are a number of different approaches to take. These range from very simple approaches, to advanced and complex ones. Each has *tradeoffs*. Some will work better than others in different situations, some require more mathematical analysis than others.

Teams should prioritize picking the easiest strategy which enables success on the field. However, as you do experiments, keep in mind there is almost always a « next-step » to take to improve your field performance.

There are two fundamental types of mechanism controller that we will cover here :

Note : These are not strict definitions - some control strategies are not easily classifiable and incorporate elements of both feedforward and feedback controllers. However, it is still a useful distinction in most FRC applications.

Feedforward control (or « open-loop control ») refers to the class of algorithms which incorporate knowledge of how the mechanism under control is *expected* to operate. Using this « model » of operation, the control input is chosen to make the mechanism get close to where it should be.

Feedback control (or « closed-loop control ») refers to the class of algorithms which use sensors to *measure* what a mechanism is doing, and issue corrective commands to move a mechanism from where it actually is, to where you want it to be.

These are not mutually exclusive, and in fact it is usually best to use both. The tutorial pages that follow will cover three types of mechanism (turret, flywheel, and vertical arm), and allow you to experiment with how each type of system responds to each type of control strategy, both individually and combined.

Feedforward Control : Making a Best Guess

« Feedforward control » means providing the mechanism with the control signal you think it needs to make the mechanism do what you want, without any knowledge of where the mechanism currently is. A feedforward controller feeds information we already know about the system *forward* into an estimate of the required *control effort*. The feedforward controller does *not* adjust this in response to the measured behavior of the system to try to correct for errors from the guess.

Feedforward control is also sometimes referred to as « open-loop control », because if you draw out a block diagram of the controlled system it consists of only a line from the controller to the plant, with no connection from the measured plant output back into the controller (hence an « open » loop, which really isn't a loop at all).

This is the type of control you are implicitly using whenever you use a joystick to « directly » control the speed of a motor through the applied voltage. It is the simplest and most straightforward type of control, and is probably the one you encountered first when programming a FRC motor, though it may not have been referred to by name.

When Do We Need Feedforward Control ?

In general, feedforward control is *required* whenever the system requires some constant control signal to remain at the desired setpoint (such as position control of a vertical arm where gravity will cause the arm to fall, or velocity control where internal motor dynamics and friction will cause the motor to slow down over time). Feedback controllers naturally fall to zero output when they achieve their setpoint, and so a feedforward controller is needed to provide the signal to *keep* the mechanism where we want it.

Some control strategies instead account for this in the feedback controller with integral gain - however, this is slow and prone to oscillation. It is almost always better to use a feedforward controller to account for the output needed to maintain the setpoint.

Feedforward and Position Control

The WPILib feedforward classes require velocity and acceleration setpoints to generate an estimated control voltage. This is because the equations-of-motion of a permanent-magnet DC motor relate the applied voltage to velocity and acceleration ; it is a fact of physics that we cannot change.

But what if we want to control position ? When controlling a DC motor, there's no immediate relation between position and control signal. In order to use feedforward effectively for position control, we need to come up with a sequence of velocities that will take the robot mechanism to the desired position. This is called a *motion profile*.

Many teams do not wish to incur the extra technical cost of using a motion profile when doing position control, and instead omit the feedforward controller entirely and opt to use only feedback control. As we will discuss later, this may work in *some* situations, but has some important caveats.

Most FRC mechanisms are well-described by WPILib's feedforward classes, though pure feedforward control typically only yields acceptable results for velocity control of mechanisms with little external load. In other cases, errors from the system model will be unavoidable and a feedback controller will be necessary to correct for them.

Feedback Control : Correcting for Errors and Disturbances

Even with unlimited study, it is impossible to know every force that will be exerted on a robot's mechanism in perfect detail. For example, in a flywheel shooter, the timing and exact forces associated with a ball being put through the mechanism are extremely difficult to measure accurately. For another example, consider the fact that gearboxes gradually throw off grease as they operate, increasing their internal friction over time. This is a *very* complex process to model well.

In practice, this means that the « guess » made by our feedforward controller will never be perfect. There will always be some error - that is, some lingering difference between the state we want our mechanism to be in, and the state the feedforward controller leaves it in. In many situations, this error is large enough that we need to adjust our output to correct it ; this is the job of the feedback controller. Feedback controllers are also called « closed-loop » controllers, because the flow of information about the current state *back* through the system « closes » the loop in the system's block diagram.

The simplest feedback controller possible is a « proportional controller », which responds proportionally to the current error (i.e. difference between the desired state and measured state). More advanced controllers (such as the PID controller) add response to the rate-of-change of the error and to the total accumulated error. All of these operate on the principle that the system response is roughly linear, in order to « nudge » the system towards the setpoint based on local measurements of the error.

When Do We Need Feedback Control ?

In general, there are two scenarios in which we *need* feedback control :

1. We are controlling the position of the system, so errors accumulate over time
2. There are a lot of difficult-to-dynamic external forces interacting with the mechanism that the feedforward loop cannot account for (e.g. a flywheel that is launching game pieces).

In each of these situations, the *best* solution is to combine a feedforward controller and a feedback controller by adding their outputs together. However, in the case of a simple position controller with no external loading, a pure feedback controller can work acceptably.

Feedback-Only Control

Feedforward controllers are extremely helpful and quite simple, but they require *explicit* knowledge of the system behavior in order to generate a guess at the required control signal. In many controls textbooks, you may see a set of techniques which rely on feedback control only. These are very common in industry, and works well in many cases, especially when the underlying system behavior is not easy to explicitly model, or when you want to quickly reach a « good enough » solution without spending the time to thoroughly investigate your system behavior.

Feedback-only control typically only works well in situations where :

1. The motors are fairly overpowered relative to loading.
2. The mechanism's position (not velocity) is being controlled.
3. There are no substantial or varying external forces on the mechanism.

When these criteria are met (such as in the turret tuning tutorial), feedback-only control can yield acceptable results. In other situations, it is necessary to use a feedforward model to reduce the amount of work done by the feedback controller. In FRC, our systems are almost all modeled by well-understood equations with working code support, so it is almost always a good idea to include a feedforward controller.

Modeling : How do you expect your system to behave ?

It's easiest to control a system if we have some prior knowledge of how the system responds to inputs. Even the « pure feedback » strategy described above implicitly assumes things about the system response (e.g. that it is approximately linear), and consequently won't work in cases where the system does not respond in the expected way. To control our system *optimally*, we need some way to reliably predict how it will respond to inputs.

This can be done by combining several concepts you may be familiar with from physics : drawing free body diagrams of the forces that act on the mechanism, taking measurements of mass and moment of inertia from your *CAD* models, applying standard equations of how DC motors or pneumatic cylinders convert energy into mechanical force and motion, etc.

The act of creating a consistent mathematical description of your system is called *modeling* your system's behavior. The resulting set of equations are called a *model* of how you expect the system to behave. Not every system requires an explicit model to be controlled (we will see in the turret tutorial that a pure, manually-tuned feedback controller is satisfactory *in some cases*), but an explicit model is *always* helpful.

Note that models do not have to be perfectly accurate to be useful. As we will see in later tuning exercises, even using a simple model of a mechanism can make the tuning effort much simpler.

Obtaining Models for Your Mechanisms

If modeling your mechanism seems daunting, don't worry ! Most mechanisms in FRC are modeled by well-studied equations and code for interacting with those models is included in WPILib. Usually, all that is needed is to determine a set of physical parameters (sometimes called « tuning constants » or « gains ») that depend on the specific details of your mechanism/robot. These can be estimated theoretically from other known parameters of your system (such as mass, length, and choice of motor/gearbox), or measured from your mechanism's actual behavior through a system identification routine.

When in doubt, ask a mentor or *support resource* !

Theoretical Modeling

ReCalc is an online calculator which estimates physical parameters for a number of common FRC mechanisms. Importantly, it can generate estimate the `kV`, `kA`, and `kG` gains for the WPILib feedforward classes.

The *WPILib system identification tool* supports a « theoretical mode » that can be used to determine PID gains for feedback control from the `kV` and `kA` gains from ReCalc, enabling (in theory) full tuning of a control loop without running any test routines.

Remember, however, that theory is not reality and purely theoretical gains are not guaranteed to work well. There is *never* a substitute for testing.

System Identification

A good way to improve the accuracy of a simple physics model is to perform experiments on the real mechanism, record data, and use the data to *derive* the constants associated with different parts of the model. This is very useful for physical quantities which are difficult or impossible to predict, but easy to measure (ex : friction in a gearbox).

WPILib's system identification tool supports some common FRC mechanisms, including drivetrain. It deploys its own code to the robot to exercise the mechanism, record data, and derive gains for both feedforward and feedback control schemes.

Manual Tuning : What to Do with No Explicit Model

Sometimes, you have to tune a system without at an explicit model. Maybe the system is uniquely complicated, or maybe you're under time constraints and need something that works quickly, even if it doesn't work optimally. Model-based control requires a correct mathematical model of the system, and for better or for worse, we do not always have one.

In such cases, the physical parameters of the control algorithm can be tuned *manually*. This is generally done by systematically « sweeping » the controller gains by hand until the mechanism behaves as expected. Manual tuning can work quickly in cases where only one or two parameters (such as kV and kP) need to be adjusted - however, in more-complicated scenarios it can become a very involved and difficult process.

One common problem with manual tuning is that it can be hard to distinguish a well-founded controller architecture that is not yet tuned properly, from an inappropriate controller architecture that cannot work (for example, it is generally not possible to tune a velocity controller or vertical arm position controller that functions well without a feedforward). In such a case, we can waste a lot of time searching for correct gains, when no such correct gains exist. There is no substitute for understanding the mechanics of the systems being controlled well enough to determine a correct controller architecture for the mechanism, *even if* we do not explicitly use any model-based control methodologies.

The tutorials that follow include simulations that will allow you to perform the manual tuning process on several typical FRC mechanisms. The fundamental concepts that govern which control strategies are valid for each mechanism are covered on the individual mechanism pages ; pay close attention to this as you work through the tutorials !

32.2.3 Introduction to DC Motor Feedforward

Note : For a guide on implementing PID control in code with WPILib, see *Commande prédictive ou par anticipation en WPILib*.

This page explains the conceptual and mathematical workings of WPILib's SimpleMotorFeedforward (and the other related classes).

The Permanent-Magnet DC Motor Feedforward Equation

Recall from earlier that the point of a feedforward controller is to use the known dynamics of a mechanism to make a best guess at the *control effort* required to put the mechanism in the state you want. In order to do this, we need to have some idea of what kind of mechanism we are controlling - that will determine the relationship between *control effort* and *output*, and let us guess at what value of the former will give us the desired value of the latter.

In FRC, the most common system that we're interested in controlling is the *permanent-magnet DC motor*.

These motors have a number of convenient properties that make them particularly easy to control, and ideal for FRC tasks. In particular, they obey a particular relationship between applied voltage, rotor velocity, and rotor acceleration known as a « voltage balance equation ».

$$V = K_s \cdot sgn(\dot{d}) + K_v \cdot \dot{d} + K_a \cdot \ddot{d}$$

where V is the applied voltage, d is the displacement (position) of the motor, $\dot{d}$ is its velocity, and $\ddot{d}$ is its acceleration (the « overdot » notation traditionally denotes the *derivative* with respect to time).

We can interpret the coefficients in the above equation as follows :

K_s is the voltage needed to overcome the motor's static friction, or in other words to just barely get it moving ; it turns out that this static friction (because it's, well, static) has the same effect regardless of velocity or acceleration. That is, no matter what speed you're going or how fast you're accelerating, some constant portion of the voltage you've applied to your motor (depending on the specific mechanism assembly) will be going towards overcoming the static friction in your gears, bearings, etc ; this value is your kS. Note the presence of the *signum function* because friction force always opposes the direction-of-motion.

K_v describes how much voltage is needed to hold (or « cruise ») at a given constant velocity while overcoming the *counter-electromotive force* and any additional friction that increases with speed (including *viscous drag* and some *churning losses*). The relationship between speed and voltage (at constant acceleration) is almost entirely linear (for FRC-legal components) because of how permanent-magnet DC motors work.

K_a describes the voltage needed to induce a given acceleration in the motor shaft. As with kV, the relationship between voltage and acceleration (at constant velocity) is almost perfectly linear for FRC components.

For more information, see this paper.

Variants of the Feedforward Equation

Some of WPILib's other feedforward classes introduce additional terms into the above equation to account for known differences from the simple case described above - details for each tool can be found below :

Elevator Feedforward

An elevator consists of a permanent-magnet DC motor attached to a mass under the force of gravity. Compared to the feedforward equation for an unloaded motor, it differs only in the inclusion of a constant K_g term that accounts for the action of gravity :

$$V = K_g + K_s \cdot sgn(\dot{d}) + K_v \cdot \dot{d} + K_a \cdot \ddot{d}$$

where V is the applied voltage, d is the displacement (position) of the drive, $\dot{d}$ is its velocity, and $\ddot{d}$ is its acceleration.

Arm Feedforward

An arm consists of a permanent-magnet DC motor attached to a mass on a stick held under the force of gravity. Like the elevator feedforward, it includes a K_g term to account for the effect of gravity - unlike the elevator feedforward, however, this term is multiplied by the cosine of the arm angle (since the gravitational force does not act directly on the motor) :

$$V = K_g \cdot cos(\theta) + K_s \cdot sgn(\dot{\theta}) + K_v \cdot \dot{\theta} + K_a \cdot \ddot{\theta}$$

where V is the applied voltage, θ is the angular displacement (position) of the arm, $\dot{\theta}$ is its angular velocity, and $\ddot{\theta}$ is its angular acceleration.

Using the Feedforward

In order to use the feedforward, we need to plug in values for each unknown in the above voltage-balance equation *other than the voltage*. As mentioned *earlier*, the values of the gains K_g, K_v, K_a can be obtained through theoretical modeling with ReCalc. Explicit measurement with *SysId* will yield the aforementioned gains in addition to K_s. That leaves us needing values for velocity, acceleration, and (in the case of the arm feedforward) position.

Typically, these come from our setpoints - remember that with feedforward we are making a « guess » as to the output we need based on where we want the system to be.

For velocity control, this does not pose a problem - we can take the velocity value from our setpoint directly, and if necessary (it can often be omitted in practice) we can infer the acceleration from the difference between the current and previous velocity setpoints.

For position control, however, this can be difficult - except for the arm controller, there's no direct term in the feedforward equation for position. We often have no choice but to calculate our velocity from the difference between the current and previous setpoint positions, and to ignore acceleration entirely. In order to do better, we need to ensure that our setpoints vary *smoothly* according to some set of constraints - this is usually accomplished with a *motion profile*.

32.2.4 Une introduction aux PIDs

Note : For a guide on implementing PID control with WPILib, see *Contrôle PID dans WPILib*.

This page explains the conceptual and mathematical workings of a PID controller. *A video explanation from WPI is also available*.

What is a PID Controller ?

The PID controller is a common *feedback controller* consisting of proportional, integral, and derivative terms, hence the name. This article will build up the definition of a PID controller term by term while trying to provide some intuition for how each term behaves.

First, we'll get some nomenclature for PID controllers out of the way. In a PID context, we use the term *reference* or *setpoint* to mean the desired state of the mechanism, and the term *output* or *process variable* to refer to the measured state of the mechanism. Below are some common variable naming conventions for relevant quantities.

$r(t)$	*setpoint*, *reference*	$u(t)$	*control effort*
$e(t)$	*erreur*	$y(t)$	*output*, *process variable*

The *error* $e(t)$ is the difference between the *reference* and the *output*, $r(t) - y(t)$.

For those already familiar with PID control, this interpretation may not be consistent with the classical explanation of the P, I, and D terms corresponding to response to « past », « present », and « future » errors. While that model has merit, we will instead be approaching PID control from the viewpoint of modern control theory, as proportional controllers applied to different physical quantities we care about. This will provide a more complete explanation of the derivative term's behavior for constant and moving *setpoints*.

Roughly speaking : the proportional term drives the position error to zero, the derivative term drives the velocity error to zero, and the integral term drives the total accumulated error-over-time to zero. All three terms are added together to produce the *control signal*. We'll go into more detail on each of these below.

Note : Throughout the WPILib documentation, you'll see two ways of writing the tunable constants of the PID controller.

For example, for the proportional gain :
- K_p is the standard math-equation-focused way to notate the constant.
- kP is a common way to see it written as a variable in software.

Despite the differences in capitalization, the two formats refer to the same concept.

Paramètre P, ou Proportionnel

The *Proportional* term attempts to drive the position error to zero by contributing to the control signal proportionally to the current position error. Intuitively, this tries to move the *output* towards the *reference*.

$$u(t) = K_p e(t)$$

où K_p est le gain proportionnel et $e(t)$ est l'erreur au moment actuel t.

La figure ci-dessous montre un schéma-bloc pour un *Système* contrôlé seulement par le paramètre P.

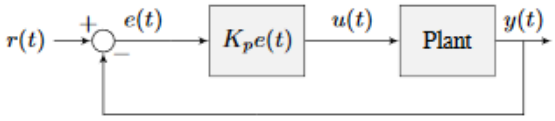


Figure 2.1: P controller block diagram

Les gains proportionnels agissent comme des « ressorts définis par logiciel » qui tirent le *Système* vers la position souhaitée. Les cours de Physique nous ont appris que nous modélisons les ressorts comme $F = -kx$ où F est la force appliquée, k est une constante proportionnelle, et x est le déplacement à partir du point d'équilibre. Cela peut s'écrire d'une autre manière $F = k(0 - x)$ où 0 est le point d'équilibre. Si nous redéfinissons le point d'équilibre comme le *Point de consigne* de notre contrôleur à rétroaction, les équations correspondent terme pour terme.

$$F = k(r - x)$$
$$u(t) = K_p e(t) = K_p(r(t) - y(t))$$

de sorte que la « force » avec laquelle le contrôleur proportionnel tire la Sortie du *Système* vers le *Point de consigne* est proportionnelle à l" *Erreur*, tout comme un ressort.

Paramètre D, ou dérivée

The *Derivative* term attempts to drive the derivative of the error to zero by contributing to the control signal proportionally to the derivative of the error. Intuitively, this tries to make the *output* move at the same rate as the *reference*.

$$u(t) = K_p e(t) + K_d \frac{de}{dt}$$

où K_p est le gain proportionnel, K_d est le gain dérivé, et $e(t)$ est l'erreur au moment actuelle t.

La figure ci-dessous montre un schéma-bloc pour un *Système* contrôlé par un contrôleur PD.

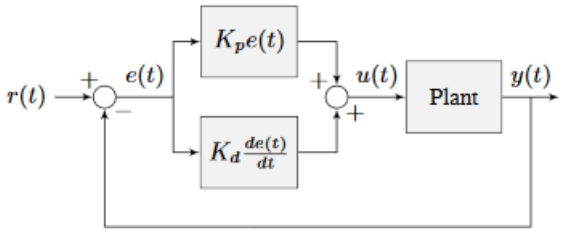


Figure 2.2: PD controller block diagram

Un contrôleur PD a un contrôleur proportionnel pour la position (K_p) ET un contrôleur proportionnel pour la vitesse (K_d). Le *Point de consigne* de la vitesse est implicitement défini par la façon dont le *Point de consigne* de la position change avec le temps. Pour le prouver, nous allons reformuler l'équation pour un contrôleur PD.

$$u_k = K_p e_k + K_d \frac{e_k - e_{k-1}}{dt}$$

where u_k is the *control effort* at timestep k and e_k is the *error* at timestep k. e_k is defined as $e_k = r_k - x_k$ where r_k is the *setpoint* and x_k is the current *state* at timestep k.

$$\begin{aligned}
u_k &= K_p(r_k - x_k) + K_d \frac{(r_k - x_k) - (r_{k-1} - x_{k-1})}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{r_k - x_k - r_{k-1} + x_{k-1}}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{r_k - r_{k-1} - x_k + x_{k-1}}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{(r_k - r_{k-1}) - (x_k - x_{k-1})}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \left(\frac{r_k - r_{k-1}}{dt} - \frac{x_k - x_{k-1}}{dt} \right)
\end{aligned}$$

Remarquez comment $\frac{r_k - r_{k-1}}{dt}$ est la vitesse du *Point de consigne*. Pour la même raison, $\frac{x_k - x_{k-1}}{dt}$ est la vitesse *Système* à un moment donné. Cela signifie que le terme K_d du contrôleur PD force la vitesse estimée vers la vitesse du *Point de consigne*.

Si le *Point de consigne* est constant, le *Point de consigne* vitesse est nécessairement zéro. Par conséquent, le terme K_d ralentit le *Système*, s'il est en déplacement. Ceci agit comme un « amortisseur défini par logiciel ». On retrouve ces amortisseurs sur certaines portes industrielles, et ils sont utilisés pour qu'elles se ferment doucement. Leur force augmente linéairement avec la vitesse.

Paramètre I, ou Intégrale

Important : Integral gain is generally not recommended for FRC® use. It is almost always better to use a feedforward controller to eliminate steady-state error. If you do employ integral gain, it is crucial to provide some protection against *integral windup*.

The *Integral* term attempts to drive the total accumulated error to zero by contributing to the control signal proportionally to the sum of all past errors. Intuitively, this tries to drive the

average of all past *output* values towards the *average* of all past *reference* values.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau$$

où K_p est le gain proportionnel, K_i est le gain intégral, $e(t)$ est l'erreur au moment donné t, et τ est la variable d'intégration.

L'intégrale intègre de l'instant 0 à l'instant actuel t. Nous utilisons τ pour l'intégration parce que nous avons besoin d'une variable pour prendre plusieurs valeurs dans l'intégrale, mais nous ne pouvons pas utiliser t parce que nous l'avons déjà défini comme l'instant actuel.

La figure ci-dessous montre un schéma-bloc pour un *Système* contrôlé par un contrôleur PI.

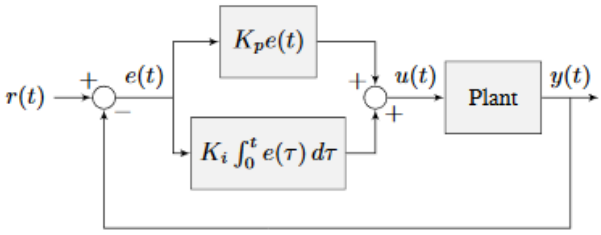


Figure 2.3: PI controller block diagram

Lorsque le *Système* s'approche du *Point de consigne* en régime permanent, le terme proportionnel peut être trop petit pour faire converger la Sortie jusqu'au *Point de consigne*, et le terme dérivé est alors à zéro. Cela peut entraîner une erreur de régime permanent comme le montre la figure 2.4

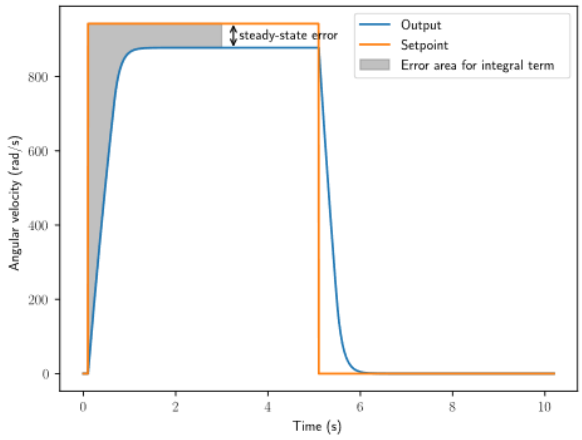


Figure 2.4: P controller with steady-state error

A common way of eliminating *steady-state error* is to integrate the *error* and add it to the *control effort*. This increases the *control effort* until the *system* converges. Figure 2.4 shows an example of *steady-state error* for a flywheel, and figure 2.5 shows how an integrator added to the flywheel controller eliminates it. However, too high of an integral gain can lead to overshoot, as shown in figure 2.6.

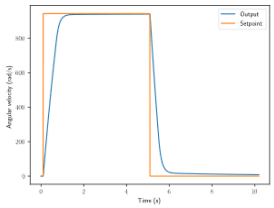


Figure 2.5: PI controller without steady-state error

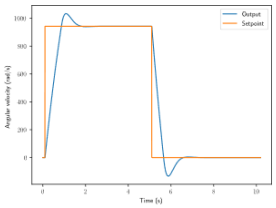


Figure 2.6: PI controller with overshoot from large K_i gain

Putting It All Together

Note : Pour plus d'informations sur l'utilisation du PIDController fourni par WPILib, voir *relevant article*.

When these terms are combined by summing them all together, one gets the typical definition for a PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de}{dt}$$

où K_p est le gain proportionnel, K_i est le gain intégral, K_d est le gain dérivé, $e(t)$ est l'erreur au moment actuel t, et τ est la variable d'intégration.

La figure ci-dessous montre un schéma-bloc d'un contrôleur PID.

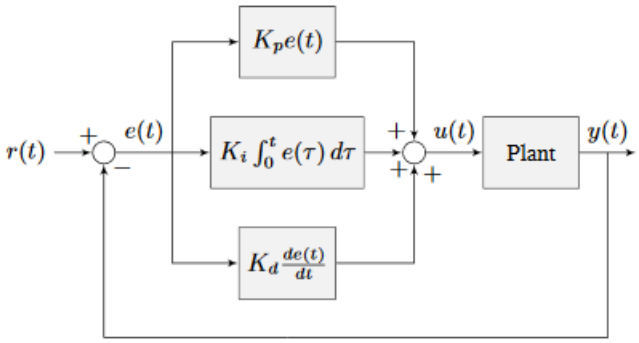


Figure 2.7: PID controller block diagram

Types de réponse

Un *Système* piloté par un contrôleur PID a généralement trois types de réponses : sous-amorti, sur-amorti et critique. Ces réponses sont illustrées à la figure 2.8.

Pour la *Réponse à une transition* de la figure 2.7, *Temps de montée* est le temps que le *Système* prend pour atteindre initialement la référence après avoir appliqué une *impulsion d'entrée*. Le *Temps de destabilisation* est le temps que prend le *Système* pour atteindre la *Référence* après l'application d'une *impulsion d'entrée*.

Une réponse *sous-amortie* oscille autour de la *Référence* avant de se stabiliser. Une réponse *sur-amortie*

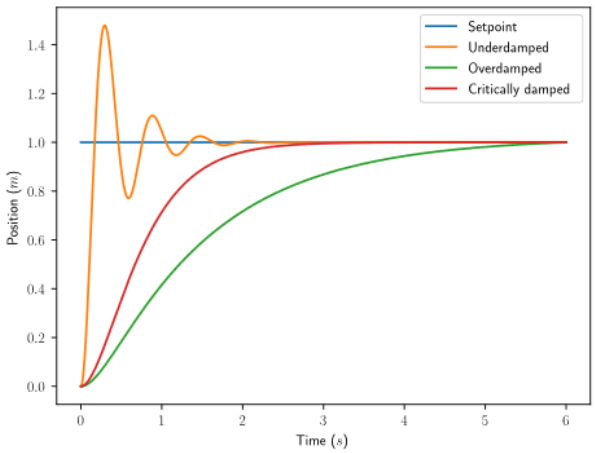


Figure 2.8: PID controller response types

est lente à augmenter et ne dépasse pas le *Référence*. Une réponse *amortie de manière critique* a le *temps de montée* le plus rapide, sans toutefois dépasser la *Référence*.

32.2.5 Vidéo d'introduction au PID, par WPI

Avez-vous déjà rencontré des difficultés à concevoir un système robotique pouvant bouger rapidement et s'arrêter exactement à la position désirée ? Ce type de défi survient lorsqu'on veut conduire sur une distance ou à une vitesse fixe, contrôler un bras, un élévateur ou un système motorisé nécessitant un mouvement spécifique. Dans le vidéo suivante, Dmitry Berenson, professeur à WPI, nous informe sur le sujet de l'asservissement en robotique et des boucles de rétroaction PID.

32.2.6 Introduction To Controls Tuning Tutorials

The WPILib docs include three interactive tuning simulations. Their goal is to allow students to learn how tuning parameters impact system behavior, without having to deal with software bugs or other real-world behavior.

Even though WPILib tooling can provide you with optimal gains, it is worth going through the manual tuning process to see how the different control strategies interact with the mechanism.

Ultimately, students should use the examples to build intuition and make their time on the robot more productive.

This page details a few tips while working with the tutorials.

Parameter Exponential Search

While interacting with the simulations, you will get instructions to « increase » or « decrease » different parameters.

When « increasing » a value, multiply it by two until the expected effect is observed. After the first time the value becomes too large (i.e. the behavior is unstable or the mechanism overshoots), reduce the value to halfway between the first too-large value encountered and the previous value tested before that. Continue iterating this « split-half » procedure to zero in on the optimal value (if the response undershoots, pick the halfway point between the new value and the last value immediately above it - if it overshoots, pick the halfway point between the new value and the last value immediately below it). This is called an *exponential search*, and is a very efficient way to find positive values of unknown scale.

System Noise

The « system noise » option introduces random, gaussian error into the plant to provide a more realistic situation of system behavior.

Leave the setting turned off at first to learn the system's ideal behavior. Later, turn it on to see how your tuning works in the presence of real-world effects.

Be Systematic

As seen in *the introduction to PID*, a PID controller has *three* tuned constants.Feedforward components will add even more. This means searching for the « correct » constants manually can be quite difficult - it is therefore necessary to approach the tuning procedure systematically.

Follow the order of tuning presented in the tutorials - it will maximize your chances of success.

Resist checking the tuning solutions until you believe your solution is close to correct. Then check your answer, and try the provided one to compare against your own results.

Furthermore, work from easy to difficult. *Flywheel mechanisms* are the easiest to tune. After that, look into the *turret tuning*. Then, finish off with the *vertical arm example*.

32.2.7 Tuning a Flywheel Velocity Controller

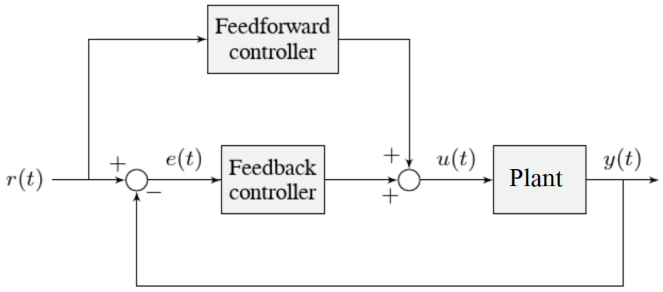
In this section, we will tune a simple velocity controller for a flywheel. The tuning principles explained here will also work for almost any velocity control scenario.

Flywheel Model Description

Our « Flywheel » consists of :

- A rotating inertial mass which launches the game piece (the flywheel)
- A motor (and possibly a gearbox) driving the mass.

For the purposes of this tutorial, this plant is modeled with the same equation used by WPILib's *La classe « SimpleMotorFeedforward »*, with additional adjustment for sensor delay and gearbox inefficiency. The simulation assumes the plant is controlled by feedforward and feedback controllers, composed in this fashion :



Where :

— The plant's *output* $y(t)$ is the flywheel rotational velocity
— The controller's *setpoint* $r(t)$ is the desired velocity of the flywheel
— The controller's *control effort*, $u(t)$ is the voltage applied to the motor driving the flywheel's motion

---

Note : A more detailed description of the mathematics of the system *can be found here*.

Picking the Control Strategy for a Flywheel Velocity Controller

In general : the more voltage that is applied to the motor, the faster the flywheel will spin. Once voltage is removed, friction and *back-EMF* oppose the motion and bring the flywheel to a stop.

Flywheels are commonly used to propel game pieces through the air, toward a target. In this simulation, a gamepiece is injected into the flywheel about halfway through the simulation. [1]

To consistently launch a gamepiece, a good first step is to make sure it is spinning at a particular speed before putting a gamepiece into it. Thus, we want to accurately control the velocity of our flywheel.

Note : This is fundamentally different from the *vertical arm* and *turret* controllers, which both control *position*.

The tutorials below will demonstrate the behavior of the system under bang-bang, pure feedforward, pure feedback (PID), and combined feedforward-feedback control strategies. Follow the instructions to learn how to manually tune these controllers, and expand the « tuning solution » to view an optimal model-based set of tuning parameters.

1. For this simulation, we model a ball being injected to the flywheel as a velocity-dependant (frictional) torque fighting the spinning of the wheel for one quarter of a wheel rotation, right around the 5 second mark. This is a very simplistic way to model the ball, but is sufficient to illustrate the controller's behavior under a sudden load. It would not be sufficient to predict the ball's trajectory, or the actual « pulldown » in *output* for the system.

Bang-Bang Control

Interact with the simulation below to see how the flywheel system responds when controlled by a bang-bang controller.

The « Bang-Bang » controller is a simple controller which applies a binary (present/not-present) force to a mechanism to try to get it closer to a setpoint. A more detailed description (and documentation for the corresponding WPILib implementation) can be found *here.*

There are no tuneable controller parameters for a bang-bang controller - you can only adjust the setpoint. This simplicity is a strength, and also a weakness.

Try adjusting the setpoint up and down. You should see that for almost all values, the output converges to be somewhat near the setpoint.

Common Issues with Bang-Bang Controllers

Note that the system behavior is not perfect, because of delays in the control loop. These can result from the nature of the sensors, measurement filters, loop iteration timers, or even delays in the control hardware itself. Collectively, these cause a cycle of « overshoot » and « undershoot », as the output repeatedly goes above and below the setpoint. This oscillation is unavoidable with a bang-bang controller.

Typically, the steady-state oscillation of a bang-bang controller is small enough that it performs quite well in practice. However, rapid on/off cycling of the control effort can cause mechanical issues - the cycles of rapidly applying and removing forces can loosen bolts and joints, and put a lot of stress on gearboxes.

The abrupt changes in control effort can cause abrupt changes in current draw if the system's inductance is too low. This may stress motor control hardware, and cause eventual damage or failure.

Finally, this technique only works for mechanisms that accelerate relatively slowly. A more in-depth discussion of the details *can be found here*.

Bang-bang control sacrifices a lot for simplicity and high performance (in the sense of fast convergence to the setpoint). To achieve « smoother » control, we need to consider a different control strategy.

Pure Feedforward Control

Interact with the simulation below to see how the flywheel system responds when controlled only by a feedforward controller.

To tune the feedforward controller, increase the velocity feedforward gain K_v until the flywheel approaches the correct setpoint over time. If the flywheel overshoots, reduce K_v.

The exact gain used by the simulation is $K_v = 0.0075$.

We can see that a pure feedforward control strategy works reasonably well for flywheel velocity control. As we mentioned earlier, this is why it's possible to control most motors « directly » with joysticks, without any explicit « control loop » at all. However, we can still do better - the pure feedforward strategy cannot reject disturbances, and so takes a while to recover after the ball is introduced. Additionally, the motor may not perfectly obey the feedforward equation (even after accounting for vibration/noise). To account for these, we need a feedback controller.

Pure Feedback Control

Interact with the simulation below to see how the flywheel system responds when controlled by only a feedback (PID) controller.

Perform the following :

1. Set K_p, K_i, K_d, and K_v to zero.
2. Increase K_p until the *output* starts to oscillate around the *setpoint*, then decrease it until the oscillations stop.
3. *In some cases*, increase K_i if *output* gets « stuck » before converging to the *setpoint*.

Note : PID-only control is not a very good control scheme for flywheel velocity ! Do not be surprised if/when the simulation below does not behave well, even when the « optimal » constants are used.

In this particular example, for a setpoint of 300, values of $K_p = 0.1$, $K_i = 0.0$, and $K_d = 0.0$ will produce somewhat reasonable results. Since this control strategy is not very good, it will not work well for all setpoints. You can attempt to improve this behavior by incorporating some K_i, but it is very difficult to achieve good behavior across a wide range of setpoints.

Issues with Feedback Control Alone

Because a non-zero amount of *control effort* is required to keep the flywheel spinning, even when the *output* and *setpoint* are equal, this feedback-only strategy is flawed. In order to optimally control a flywheel, a combined feedforward-feedback strategy is needed.

Combined Feedforward and Feedback Control

Interact with the simulation below to see how the flywheel system responds under simultaneous feedforward and feedback (PID) control.

Tuning the combined flywheel controller is simple - we first tune the feedforward controller following the same procedure as in the feedforward-only section, and then we tune the PID controller following the same procedure as in the feedback-only section. Notice that PID portion of the controller is *much* easier to tune « on top of » an accurate feedforward.

In this particular example, for a setpoint of 300, values of $K_v = 0.0075$ and $K_p = 0.1$ will produce very good results across all setpoints. Small changes to K_p will change the controller behavior to be more or less aggressive - the optimal choice depends on your problem constraints.

Note that the combined feedforward-feedback controller works well across all setpoints, and recovers very quickly after the external disturbance of the ball contacting the flywheel.

Tuning Conclusions

Applicability of Velocity Control

A gamepiece-launching flywheel is one of the most visible applications of velocity control. It is also applicable to drivetrain control - following a pre-defined path in autonomous involves controlling the velocity of the wheels with precision, under a variety of different loads.

Choice of Control Strategies

Because we are controlling velocity, we can achieve fairly good performance with a *pure feedforward controller*. This is because a permanent-magnet DC motor's steady-state velocity is roughly proportional to the voltage applied, and is the reason that you can drive your robot around with joysticks without appearing to use any control loop at all - in that case, you are implicitly using a proportional feedforward model.

Because we must apply a constant control voltage to the motor to maintain a velocity at the setpoint, we cannot successfully use a *pure feedback (PID) controller* (whose output typically disappears when you reach the setpoint) - in order to effectively control velocity, a feedback controller must be *combined with a feedforward controller*.

Bang-bang control can be combined with feedforward control much in the way PID control can - for the sake of brevity we do not include a combined feedforward-bang-bang simulation.

Tuning with only feedback can produce reasonable results in cases where no *control effort* is required to keep the *output* at the *setpoint*. This may work for mechanisms like turrets, or swerve drive steering. However, as seen above, it does not work well for a flywheel, where the back-EMF and friction both act to slow the motor even when it is sustaining motion at the setpoint. To control this system, we need to combine the PID controller with a feedforward controller.

K_d is not useful for velocity control with a constant setpoint - it is only necessary when the setpoint is changing.

Adding an integral gain to the *controller* is often a sub-optimal way to eliminate *steady-state error* - you can see how sloppy and « laggy » it is in the simulation above ! As we will see soon, a better approach is to combine the PID controller with a feedforward controller.

Velocity and Position Control

Velocity control also differs from position control in the effect of inertia - in a position controller, inertia tends to cause the mechanism to swing past the setpoint even if the control voltage drops to zero near the setpoint. This makes aggressive control strategies infeasible, as they end up wasting lots of energy fighting self-induced oscillations. In a velocity controller, however, the effect is different - the rotor shaft stops accelerating as soon as you stop applying a control voltage (in fact, it will slow down due to friction and back-EMF), so such overshoots are rare (in fact, overshoot typically occurs in velocity controllers only as a result of loop delay). This enables the use of an extremely simple, extremely aggressive control strategy called *bang-bang control*.

Feedforward Simplifications

For the sake of simplicity, the simulations above omit the K_s term from the WPILib SimpleMotorFeedforward equation. On actual mechanisms, however, this can be important - especially if there's a lot of friction in the mechanism gearing. A flywheel with a lot of static friction will not have a linear control voltage-velocity relationship unless the feedforward controller includes a K_s term to cancel it out.

To measure K_s manually, slowly increase the voltage to the mechanism until it starts to move. The value of K_s is the largest voltage applied before the mechanism begins to move.

Additionally, there is no need for a K_a term in the feedforward for velocity control unless the setpoint is changing - for a flywheel, this is not a concern, and so the gain is omitted here.

Footnotes

32.2.8 Tuning a Turret Position Controller

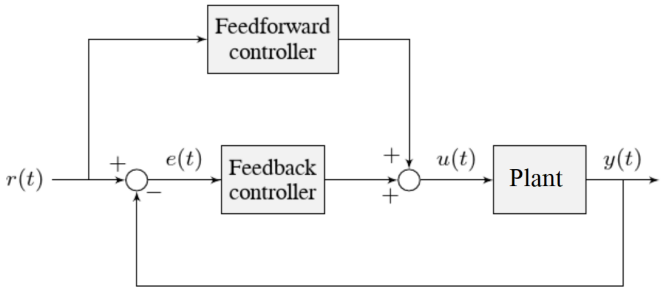
In this section, we will tune a simple position controller for a turret. The tuning principles explained here will also work for almost any position-control scenarios under no external loading.

Turret Model Description

A turret rotates some mechanism side-to-side to position it for scoring gamepieces.

Our « turret » consists of :
- A rotating inertial mass (the turret)
- A motor and gearbox driving the mass

For the purposes of this tutorial, this plant is modeled with the same equation used by WPILib's *La classe « SimpleMotorFeedforward »*, with additional adjustment for sensor delay and gearbox inefficiency. The simulation assumes the plant is controlled by feedforward and feedback controllers, composed in this fashion :



Where :
- The plant's *output* $y(t)$ is the turret's position
- The controller's *setpoint* $r(t)$ is the desired position of the turret
- The controller's *control effort*, $u(t)$ is the voltage applied to the motor driving the turret

Picking the Control Strategy for a Turret Position Controller

In general : the more voltage that is applied to the motor, the faster the motor (and turret) will spin. Once voltage is removed, friction and back-EMF slowly decrease the spinning until the turret stops. We want to make the turret rotate to a given position.

The tutorials below will demonstrate the behavior of the system under pure feedforward, pure feedback (PID), and combined feedforward-feedback control strategies. Follow the instructions to learn how to manually tune these controllers, and expand the « tuning solution » to view an optimal model-based set of tuning parameters. Even though WPILib tooling can provide you with optimal gains, it is worth going through the manual tuning process to see how the different control strategies interact with the mechanism.

This simulation does not include any motion profile generation, so acceleration setpoints are not very well-defined. Accordingly, the *kA* term of the feedforward equation is not used by the controller. This means there will be some amount of delay/lag inherent to the feedforward-only response.

Pure Feedforward Control

Interact with the simulation below to examine how the turret system responds when controlled only by a feedforward controller.

Note : To change the turret setpoint, click on the desired angle along the perimeter of the turret. To command smooth motion, click and drag the setpoint indicator.

To tune the feedforward controller, perform the following :

1. Set K_v to zero.
2. Increase the velocity feedforward gain K_v until the turret tracks the setpoint during smooth, slow motion. If the turret overshoots, reduce the gain.

Note that the turret may « lag » the commanded motion - this is normal, and is fine so long as it moves the correct amount in total.

Note : Feedforward-only control is not a viable control scheme for turrets ! Do not be surprised if/when the simulation below does not behave well, even when the « correct » constants are used.

The exact gain used by the plant is $K_v = 0.2$. Note that due to timing inaccuracy in browser simulations, the K_v that works best in the simulation may be somewhat smaller than this.

Issues with Feed-Forward Control Alone

As mentioned above, our simulated mechanism perfectly obeys the WPILib *La classe « SimpleMotorFeedforward »* equation (as long as the « system noise » option is disabled). We might then expect, like in the case of the *flywheel velocity controller*, that we should be able to achieve perfect convergence-to-setpoint with a feedforward loop alone.

However, our feedforward equation relates *velocity* and *acceleration* to voltage - it allows us to control the *instantaneous motion* of our mechanism with high accuracy, but it does not allow us direct control over the *position*. This is a problem even in our simulation (in which the

feedforward equation is the *actual* equation of motion), because unless we employ a *motion profile* to generate a sequence of velocity setpoints we can ask the turret to jump immediately from one position to another. This is impossible, even for our simulated turret.

The resulting behavior from the feedforward controller is to output a single « voltage spike » when the position setpoint changes (corresponding to a single loop iteration of very high velocity), and then zero voltage (because it is assumed that the system has already reached the setpoint). In practice, we can see in the simulation that this results in an initial « impulse » movement towards the target position, that stops at some indeterminate position in-between. This kind of response is called a « kick, » and is generally seen as undesirable.

You may notice that *smooth* motion below the turret's maximum achievable speed can be followed accurately in the simulation with feedforward alone. This is misleading, however, because no real mechanism perfectly obeys its feedforward equation. With the « system noise » option enabled, we can see that even smooth, slow motion eventually results in compounding position errors when only feedforward control is used. To accurately converge to the setpoint, we need to use a feedback (PID) controller.

Pure Feedback Control

Interact with the simulation below to examine how the turret system responds when controlled only by a feedback (PID) controller.

Perform the following :

1. Set K_p, K_i, K_d, and K_v to zero.
2. Increase K_p until the mechanism responds to a sudden change in setpoint by moving sharply to the new position. If the controller oscillates too much around the setpoint, reduce *K_p* until it stops.
3. Increase K_d to reduce the amount of « lag » when the controller tries to track a smoothly moving setpoint (reminder : click and drag the turret's directional indicator to move it smoothly). If the controller starts to oscillate, reduce *K_d* until it stops.

Gains of $K_p = 0.3$ and $K_d = 0.05$ yield rapid and stable convergence to the setpoint. Other, similar gains will work nearly as well.

Issues with Feedback Control Alone

Note that even with system noise enabled, the feedback controller is able to drive the turret to the setpoint in a stable manner over time. However, it may not be possible to smoothly track a moving setpoint without lag using feedback alone, as the feedback controller can only respond to errors once they have built up. To get the best of both worlds, we need to combine our feedback controller with a feedforward controller.

Combined Feedforward and Feedback Control

Interact with the simulation below to examine how the turret system responds under simultaneous feedforward and feedback control.

Tuning the combined turret controller is simple - we first tune the feedforward controller following the same procedure as in the feedforward-only section, and then we tune the PID controller following the same procedure as in the feedback-only section. Notice that PID portion of the controller is *much* easier to tune « on top of » an accurate feedforward.

The optimal gains for the combined controller are just the optimal gains for the individual controllers : gains of $K_v = 0.15$, $K_p = 0.3$, and $K_d = 0.05$ yield rapid and stable convergence to the setpoint and relatively accurate tracking of smooth motion. Other, similar gains will work nearly as well.

Once tuned properly, the combined controller should accurately track a smoothly moving setpoint, and also accurately converge to the setpoint over time after a « jump » command.

Tuning Conclusions

Choice of Control Strategies

Like in the case of the *vertical arm*, and unlike the case of the *flywheel*, we are trying to control the *position* rather than the *velocity* of our mechanism.

In the case of the flywheel *velocity* controller we could achieve good control performance with feedforward alone. However, it is very hard to predict how much voltage will cause a certain total change in *position* (time can turn even small errors in velocity into very big errors in position). In this case, we cannot rely on feedforward control alone - as with the vertical arm, we will need a feedback controller.

Unlike in the case of the vertical arm, though, there is no voltage required to keep the mechanism at the setpoint once it’s there. As a consequence, it is often possible to effectively control a turret without any feedforward controller at all, relying only on the output of the feedback controller (if the mechanism has a lot of friction, this may not work well and both a feedforward and feedback controller may be needed). Simple position control in the absence of external forces is one of the only cases in which pure feedback control works well.

Controlling a mechanism with only feedback can produce reasonable results in cases where no *control effort* is required to keep the *output* at the *setpoint*. On a turret, this can work acceptably - however, it may still run into problems when trying to follow a moving setpoint, as it relies entirely on the controller transients to control the mechanism’s intermediate motion between position setpoints.

We saw in the feedforward-only example above that an accurate feedforward can track slow, smooth velocity setpoints quite well. Combining a feedforward controller with the feedback controller gives the smooth velocity-following of a feedforward controller with the stable long-term error elimination of a feedback controller.

Reasons for Non-Ideal Performance

This simulation does not include any motion profile generation, so acceleration setpoints are not very well-defined. Accordingly, the *kA* term of the feedforward equation is not used by the controller. This means there will be some amount of delay/lag inherent to the feedforward-only response.

A Note on Feedforward and Static Friction

For the sake of simplicity, the simulations above omit the K_s term from the WPILib SimpleMotorFeedforward equation. On actual mechanisms, however, this can be important - especially if there's a lot of friction in the mechanism gearing. A turret with a lot of static friction will be very hard to control accurately with feedback alone - it will get « stuck » near (but not at) the setpoint when the loop output falls below K_s.

To measure K_s manually, slowly increase the voltage to the mechanism until it starts to move. The value of K_s is the largest voltage applied before the mechanism begins to move.

It can be mildly difficult to *apply* the measured K_s to a position controller without motion profiling, as the WPILib SimpleMotorFeedforward class uses the velocity setpoint to determine the direction in which the K_s term should point. To overcome this, either use a motion profile, or else add K_s manually to the output of the controller depending on which direction the mechanism needs to move to get to the setpoint.

32.2.9 Tuning a Vertical Arm Position Controller

In this section, we will tune a simple position controller for a vertical arm. The same tuning principles explained below will work also for almost all position-control scenarios under the load of gravity.

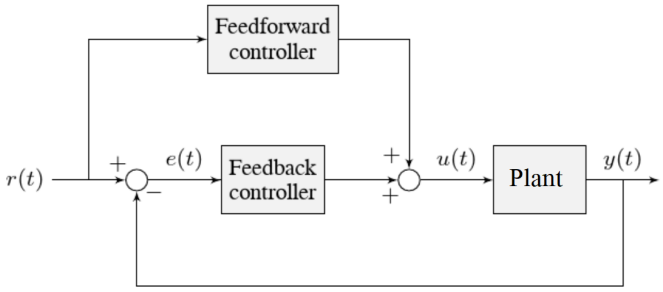
Arm Model Description

Vertical arms are commonly used to lift gamepieces from the ground up to a scoring position. Other similar examples include shooter hoods and elevators.

Our « vertical arm » consists of :

— A mass on a stick, under the force of gravity, pivoting around an axle.
— A motor and gearbox driving the axle to which the mass-on-a-stick is attached

For the purposes of this tutorial, this plant is modeled with the same equation used by WPILib's *La classe ArmFeedforward*, with additional adjustment for sensor delay and gearbox inefficiency. The simulation assumes the plant is controlled by feedforward and feedback controllers, composed in this fashion :



Where :
- The plant's *output* $y(t)$ is the arm's rotational position
- The controller's *setpoint* $r(t)$ is the desired angle of the arm
- The controller's *control effort*, $u(t)$ is the voltage applied to the motor driving the arm

Picking the Control Strategy for a Vertical Arm

Applying voltage to the motor causes a force on the mechanism that drives the arm up or down. If there is no voltage, gravity still acts on the arm to pull it downward. Generally, it is desirable to fight this effect, and keep the arm at a specific angle.

The tutorials below will demonstrate the behavior of the system under pure feedforward, pure feedback (PID), and combined feedforward-feedback control strategies. Follow the instructions to learn how to manually tune these controllers, and expand the « tuning solution » to view an optimal model-based set of tuning parameters. Even though WPILib tooling can provide you with optimal gains, it is worth going through the manual tuning process to see how the different control strategies interact with the mechanism.

Pure Feedforward Control

Interact with the simulation below to examine how the turret system responds when controlled only by a feedforward controller.

Note : To change the arm setpoint, click on the desired angle along the perimeter of the turret. To command smooth motion, click and drag the setpoint indicator.

To tune the feedforward controller, perform the following :

1. Set K_g and K_v to zero.
2. Increase K_g until the arm can hold its position with as little movement as possible. If the arm moves in the opposite direction, decrease K_g until it remains stationary. You will have to zero in on K_g fairly precisely (at least four decimal places).
3. Increase the velocity feedforward gain K_v until the arm tracks the setpoint during smooth, slow motion. If the arm overshoots, reduce the gain. Note that the arm may « lag » the commanded motion - this is normal, and is fine so long as it moves the correct amount in total.

Note : Feedforward-only control is not a viable control scheme for vertical arms ! Do not be surprised if/when the simulation below does not behave well, even when the « correct » constants are used.

The exact gains used by the simulation are $K_g = 1.75$ and $K_v = 1.95$.

Issues with Feed-Forward Control Alone

As mentioned above, our simulated mechanism almost-perfectly obeys the WPILib *La classe ArmFeedforward* equation (as long as the « system noise » option is disabled). We might then expect, like in the case of the *flywheel velocity controller*, that we should be able to achieve perfect convergence-to-setpoint with a feedforward loop alone.

However, our feedforward equation relates *velocity* and *acceleration* to voltage - it allows us to control the *instantaneous motion* of our mechanism with high accuracy, but it does not allow us direct control over the *position*. This is a problem even in our simulation (in which the feedforward equation is the *actual* equation of motion), because unless we employ a *motion profile* to generate a sequence of velocity setpoints we can ask the arm to jump immediately from one position to another. This is impossible, even for our simulated arm.

The resulting behavior from the feedforward controller is to output a single « voltage spike » when the position setpoint changes (corresponding to a single loop iteration of very high velocity), and then zero voltage (because it is assumed that the system has already reached the setpoint). In practice, we can see in the simulation that this results in an initial « impulse » movement towards the target position, that stops at some indeterminate position in-between. This kind of response is called a « kick, » and is generally seen as undesirable.

You will notice that, once properly tuned, the mechanism can track slow/smooth movement with a surprising amount of accuracy - however, there are some obvious problems with this approach. Our feedforward equation corrects for the force of gravity *at the setpoint* - this results in poor behavior if our arm is far from the setpoint. With the « system noise » option enabled, we can also see that even smooth, slow motion eventually results in compounding position errors when only feedforward control is used. To accurately converge to and remain at the setpoint, we need to use a feedback (PID) controller.

Pure Feedback Control

Interact with the simulation below to examine how the vertical arm system responds when controlled only by a feedback (PID) controller.

Perform the following :

1. Set K_p, K_i, K_d, and K_g to zero.
2. Increase K_p until the mechanism responds to a sudden change in setpoint by moving sharply to the new position. If the controller oscillates too much around the setpoint, reduce *K_p* until it stops.
3. Increase K_i when the *output* gets « stuck » before converging to the *setpoint*.
4. Increase K_d to help the system track smoothly-moving setpoints and further reduce oscillation.

Note : Feedback-only control is not a viable control scheme for vertical arms ! Do not be surprised if/when the simulation below does not behave well, even when the « correct » constants are used.

There is no good tuning solution for this control strategy. Values of $K_p = 5$ and $K_d = 1$ yield a reasonable approach to a stable equilibrium, but that equilibrium is not actually at the setpoint !

Issues with Feedback Control Alone

A set of gains that works well for one setpoint will act poorly for a different setpoint.

Adding some integral gain can push us to the setpoint over time, but it's unstable and laggy.

Because a non-zero amount of *control effort* is required to keep the arm at a constant height, even when the *output* and *setpoint* are equal, this feedback-only strategy is flawed. In order to optimally control a vertical arm, a combined feedforward-feedback strategy is needed.

Combined Feedforward and Feedback Control

Interact with the simulation below to examine how the vertical arm system responds under simultaneous feedforward and feedback control.

Tuning the combined arm controller is simple - we first tune the feedforward controller following the same procedure as in the feedforward-only section, and then we tune the PID controller following the same procedure as in the feedback-only section. Notice that PID portion of the controller is *much* easier to tune « on top of » an accurate feedforward.

Combining the feedforward coefficients from our first simulation ($K_g = 1.75$ and $K_v = 1.95$) and the feedback coefficients from our second simulation ($K_p = 5$ and $K_d = 1$) yields a good controller behavior.

Once tuned properly, the combined controller accurately tracks a smoothly moving setpoint, and also accurately converge to the setpoint over time after a « jump » command.

Tuning Conclusions

Choice of Control Strategies

Like in the case of the *turret*, and unlike the case of the *flywheel*, we are trying to control the *position* rather than the *velocity* of our mechanism.

In the case of the flywheel *velocity* controller we could achieve good control performance with feedforward alone. However, it is very hard to predict how much voltage will cause a certain total change in *position* (time can turn even small errors in velocity into very big errors in position). In this case, we cannot rely on feedforward control alone - as with the vertical arm, we will need a feedback controller.

Unlike in the case of the turret, though, there is a voltage required to keep the mechanism steady at the setpoint (because the arm is affected by the force of gravity). As a consequence, a pure feedback controller will not work acceptably for this system, and a combined feedforward-feedback strategy is needed.

The core reason the feedback-only control strategy fails for the vertical arm is gravity. The external force of gravity requires a constant *control effort* to counteract even when at rest at the setpoint, but a feedback controller does not typically output any control effort when at rest at the setpoint (unless integral gain is used, which we can see clearly in the simulation is laggy and introduces oscillations).

We saw in the feedforward-only example above that an accurate feedforward can track slow, smooth velocity setpoints quite well. Combining a feedforward controller with the feedback controller gives the smooth velocity-following of a feedforward controller with the stable long-term error elimination of a feedback controller.

Reasons for Non-Ideal Performance

This simulation does not include any motion profile generation, so acceleration setpoints are not very well-defined. Accordingly, the *kA* term of the feedforward equation is not used by the controller. This means there will be some amount of delay/lag inherent to the feedforward-only response.

The control law is good, but not perfect. There is usually some overshoot even for smoothly-moving setpoints - this is combination of the lack of K_a in the feedforward (see the note above for why it is omitted here), and some discretization error in the simulation. Attempting to move the setpoint too quickly can also cause the setpoint and mechanism to diverge, which (as mentioned earlier) will result in poor behavior due to the :math :"K_g" term correcting for the wrong force, as it is calculated from the setpoint, not the measurement. Using the measurement to correct for gravity is called « feedback linearization » (as opposed to « feedforward linearization » when the setpoint is used), and can be a better control strategy if your measurements are sufficiently fast and accurate.

A Note on Feedforward and Static Friction

For the sake of simplicity, the simulations above omit the K_s term from the WPILib SimpleMotorFeedforward equation. On actual mechanisms, however, this can be important - especially if there's a lot of friction in the mechanism gearing.

In the case of a vertical arm or elevator, K_s can be somewhat tedious to estimate separately from K_g. If your arm or elevator has enough friction for K_s to be important, it is recommended that you use the *WPILib system identification tool* to determine your system gains.

32.2.10 Problèmes de réglage de boucles de contrôle communs

Plusieurs problèmes courants peuvent survenir lors du réglage de contrôleurs prédictifs et à rétroaction

Emballement du terme Intégral

Sachez que si K_i est trop grand, un emballement intégral peut se produire. Suite à un changement important du *Point de consigne*, le terme intégral peut accumuler une erreur plus grande que l' Entrée contrôlée maximale. Par conséquent, le système dépasse les bornes et continue d'augmenter jusqu'à ce que cette erreur accumulée soit supprimée.

Il y a quelques manières de limiter cela :

1. Diminuer la valeur de K_i, à zéro, si possible
2. Ajouter un contrôle logique pour retourner le terme intégrateur à zéro, si le *output* est trop éloigné du *setpoint*. Certains contrôleurs de moteurs intelligents et la classe PIDController de WPILib implémentent ceci avec une méthode `setIZone()`.
3. Limiter l'intégrateur à une valeur maximale. Le `PIDController` de la WPILib implémente ceci avec la méthode `setIntegratorRange()`.

Important : La plupart des mécanismes en FRC ne nécessitent aucun contrôle intégral et les systèmes qui semblent nécessiter un contrôle intégral pour répondre de manière adéquate utilisent probablement un modèle prédictif imprécis.

Affaissement du voltage

Lorsque nous opérons des mécanismes sur notre robot, nous débitons du courant de sa batterie. Ceci cause une baisse dans le voltage disponible sur le bus d'alimentation commun du robot. Cela signifie que la performance de nos mécanismes va varier selon la charge et l'action du robot - cela n'est pas idéal.

Pour remédier à ceci, la plupart des contrôleurs de tension offrent un réglage «compensateur de tension» pour leur boucles de contrôle interne qui maintient la tension de sortie constante, malgré des changements de tension dans le bus commun. La classe `MotorController` de la WPILib offre une méthode `setVoltage` qui peut effectuer la même chose si les boucles de contrôle sont exécutées sur le RIO (supposant que vous l'appelez à chaque itération de la boucle robot).

Gardez en tête que la compensation de tension ne peut pas augmenter la tension appliquée au moteur plus haut que ce qui est disponible sur le bus - si votre actuateur est en saturation (décrit ci-bas), vous devrez prendre cela en compte séparément.

Saturation de l'actuateur

Un contrôleur calcule sa sortie en fonction de l'erreur entre la *Référence* et l' *état* actuel de *l'Usine*. Dans une application réelle, nous n'avons pas un contrôle illimité disponible au contrôleur. Lorsque les limites de l'actionneur sont atteintes, le contrôleur agit comme si le gain avait été temporairement réduit.

Si nos gains de contrôle sont trop agressifs, notre algorithme de contrôle pourrait tenter de bouger le mécanisme plus loin que son étendue de mouvement. Dans ce cas, le mécanisme va se « saturer » et se comporter comme si les gains de contrôle étaient plus petits qu'ils ne le sont. Cela peut négativement impacter la réponse adverse de contrôle (ex : résulter en des erreurs et une instabilité).

Si vous rencontrez des problèmes de saturation d'actuateur, pensez à modifier votre mécanisme, en changeant son rapport d'engrenage ou en l'alimentant par un moteur plus puissant.

32.3 Filtres

Note : Les données utilisées pour générer les différents tracés de démonstration dans cette section peuvent être trouvées `ici`.

Cette section décrit un certain nombre de filtres inclus avec WPILib qui sont utiles pour la réduction du bruit et / ou le lissage d'entrée.

32.3.1 Introduction aux filtres

Les filtres sont parmi les outils les plus couramment utilisés dans la technologie moderne et trouvent de nombreuses applications en robotique à la fois dans le traitement du signal et dans les commandes. La compréhension de la notion de filtre est cruciale pour comprendre l'utilité des différents types de filtres fournis par WPILib.

Qu'est-ce qu'un filtre ?

Note : Pour les besoins de cette section, nous supposerons que toutes les données sont des données chronologiques unidimensionnelles. De toute évidence, les concepts impliqués sont plus généraux que cela - mais une discussion complète et rigoureuse des signaux et du filtrage sort du cadre de cette documentation.

Alors, c'est quoi un filtre ? D'une façon simple, un filtre est une modélisation d'un flux d'entrées à un flux de sorties. C'est-à-dire que la valeur de sortie émise par un filtre peut dépendre (en principe) non seulement de la valeur *actuelle* de l'entrée, mais de *l'ensemble des valeurs passées et futures* (en pratique, les filtres fournis par WPILib utilisent des données en temps réel, par conséquant ils ne peuvent traiter que des valeurs *passées* de l'entrée, et non des valeurs futures). C'est un concept important, car généralement nous utilisons des filtres pour supprimer / atténuer la *dynamique* indésirable d'un signal. Lorsque nous filtrons un signal, nous souhaitons modifier *la façon dont le signal change au fil du temps*.

Effets de l'utilisation d'un filtre

Réduction du bruit

L'une des utilisations les plus courantes d'un filtre est la réduction du bruit. Un filtre qui réduit le bruit est appelé filtre *passe-bas* (car il permet aux basses fréquences de «passer» tout en bloquant les hautes fréquences). La plupart des filtres actuellement inclus dans WPILib sont effectivement des filtres passe-bas.

Limitation du débit

Les filtres sont également utilisés pour réduire la vitesse à laquelle un signal peut changer. Ceci est étroitement lié à la réduction du bruit, et les filtres qui réduisent le bruit ont également une tendance à limiter le taux de variation de leur sortie.

Détection des transitions (Edge)

Le pendant du filtre passe-bas est le filtre passe-haut, qui ne laisse passer que les hautes fréquences vers la sortie. Les filtres passe-haut peuvent être quelque peu difficiles à saisir à prime abord, mais une utilisation courante pour un filtre passe-haut est la détection de transitions - puisque les filtres passe-haut refléteront les changements soudains dans l'entrée tout en ignorant les changements plus lents, ils sont utiles pour déterminer l'emplacement des discontinuités nettes dans le signal.

Décalage de phase

Un effet négatif inévitable d'un filtre passe-bas en temps réel est l'introduction du «retard de phase». Comme mentionné précédemment, un filtre en temps réel ne peut dépendre que des valeurs passées du signal (nous ne pouvons pas voyager dans le temps pour obtenir les valeurs futures), la valeur filtrée prend un certain retard sur la valeur de l'entrée, lorsque cette dernière commence à changer. Plus la réduction du bruit est importante, plus le retard introduit est important. C'est, à bien des égards, *le* compromis fondamental du filtrage en temps réel, et cela devrait être le principal facteur déterminant de la conception de votre filtre.

Fait intéressant, les filtres passe-haut introduisent une *avance* de phase, plutôt qu'un retard de phase, car ils aggravent les changements locaux de la valeur de l'entrée.

32.3.2 Filtres linéaires

Le premier type de filtre (et le plus couramment utilisé) supporté par WPILib est un *filtre linéaire* - ou, plus précisément, un filtre linéaire invariable dans le temps (LTI).

An LTI filter is, put simply, a weighted moving average - the value of the output stream at any given time is a localized, weighted average of the inputs near that time. The difference between different types of LTI filters is thus reducible to the difference in the choice of the weighting function (also known as a « window function » or an « impulse response ») used. The mathematical term for this operation is *convolution*.

Il existe deux grandes « catégories » de réponses impulsionnelles : les réponses impulsionnelles infinies (IIR) et les réponses impulsionnelles finies (FIR).

Les réponses impulsionnelles infinies ont un « support » infini - c'est-à-dire qu'elles sont toutes non-égales à zéro sur une région infiniment grande. Cela signifie, en gros, qu'elles ont également une « mémoire » infinie - une fois qu'une valeur apparaît dans le flux d'entrée, elle influencera *toutes les sorties suivantes, pour toujours*. Ceci est généralement indésirable du point de vue strict du traitement du signal, cependant les filtres avec des réponses impulsionnelles infinies ont tendance à être très faciles à réaliser car ils peuvent être exprimés par de simples relations de récursivité.

Les réponses impulsionnelles finies ont un « support » fini - c'est-à-dire qu'elles sont non-égales à zéro sur une région limitée. Le filtre FIR « typique » est une moyenne mobile plate - c'est-à-dire qu'il suffit de régler la sortie égale à la moyenne des n dernières entrées. Les filtres FIR ont tendance à avoir des propriétés plus souhaitables que les filtres IIR, mais leur calcul est plus long.

Linear filters are supported in WPILib through the `LinearFilter` class (Java, C++, , `Python`).

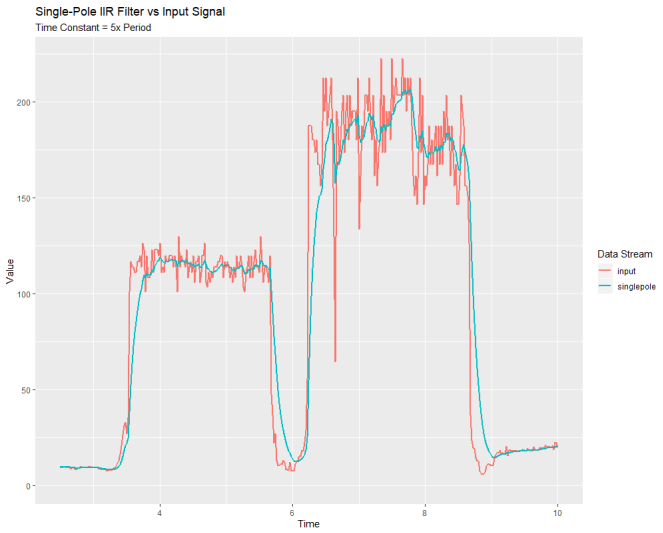
Création d'un LinearFilter

Note : La classe C ++ `LinearFilter` est basée sur le type de données utilisé pour l'entrée.

Note : Étant donné que les filtres ont une «mémoire», chaque flux d'entrée nécessite son propre objet filtre. N'essayez *pas* d'utiliser le même objet de filtre pour plusieurs flux d'entrée.

Bien qu'il soit possible d'instancier directement la classe `LinearFilter` pour créer un filtre personnalisé, il est beaucoup plus pratique (et courant) d'utiliser l'une des méthodes fournies dans WPIlib, à la place :

Filtre singlePoleIIR



The `singlePoleIIR()` factory method creates a single-pole infinite impulse response filter which performs *exponential smoothing*. This is the « go-to, » « first-try » low-pass filter in most applications ; it is computationally trivial and works in most cases.

JAVA

```
// Creates a new Single-Pole IIR filter
// Time constant is 0.1 seconds
// Period is 0.02 seconds - this is the standard FRC main loop period
LinearFilter filter = LinearFilter.singlePoleIIR(0.1, 0.02);
```

C++

```
// Creates a new Single-Pole IIR filter
// Time constant is 0.1 seconds
// Period is 0.02 seconds - this is the standard FRC main loop period
frc::LinearFilter<double> filter = frc::LinearFilter<double>::SinglePoleIIR(0.1_s, 0.
↪02_s);
```

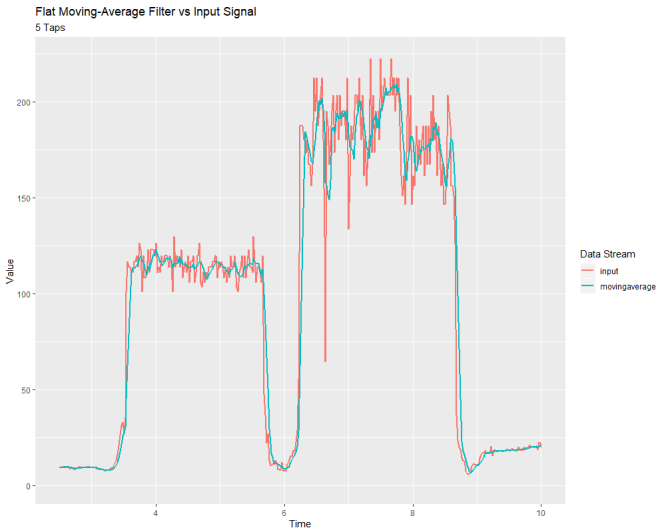
PYTHON

```python
from wpimath.filter import LinearFilter

# Creates a new Single-Pole IIR filter
# Time constant is 0.1 seconds
# Period is 0.02 seconds - this is the standard FRC main loop period
filter = LinearFilter.singlePoleIIR(0.1, 0.02)
```

Le paramètre « time constant » détermine « l'échelle de temps caractéristique » de la réponse impulsionnelle du filtre ; le filtre annulera toute dynamique de signal se produisant sur des échelles de temps significativement plus courtes que définie par ce paramètre. De même, c'est aussi l'échelle de temps approximative du *retard de phase* généré. L'inverse de cette échelle de temps, multiplié par 2 pi, est la « fréquence de coupure » du filtre.

Le paramètre « période » est la période de temps entre les appels successifs de la méthode `calculate()` propre au filtre. Pour la grande majorité des implémentations, ce sera la période standard de boucle du robot principal, soit 20 mSec.

Filtre à moyenne mobile (movingAverage)



La méthode `movingAverage` crée un simple filtre à moyenne mobile. Il s'agit du filtre FIR passe-bas le plus simple possible et il est utile dans bon nombre des mêmes contextes que le filtre IIR unipolaire. Il est un peu plus long à calculer, mais se comporte généralement de manière un peu plus agréable.

JAVA

```
// Creates a new flat moving average filter
// Average will be taken over the last 5 samples
LinearFilter filter = LinearFilter.movingAverage(5);
```

C++

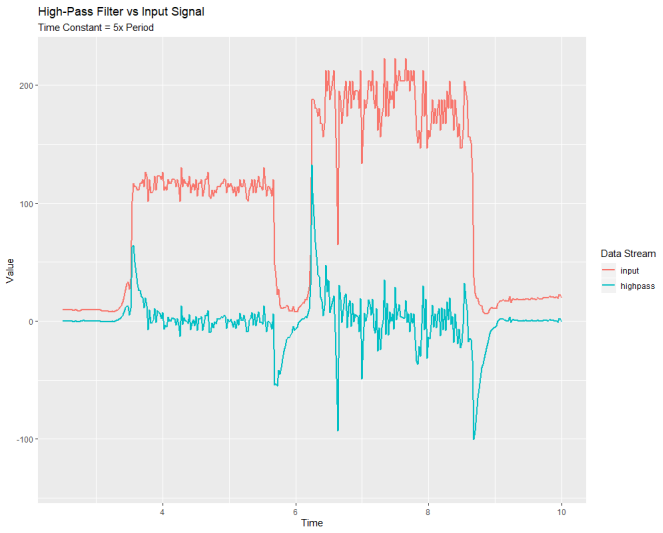
```
// Creates a new flat moving average filter
// Average will be taken over the last 5 samples
frc::LinearFilter<double> filter = frc::LinearFilter<double>::MovingAverage(5);
```

PYTHON

```
from wpimath.filter import LinearFilter

# Creates a new flat moving average filter
# Average will be taken over the last 5 samples
filter = LinearFilter.movingAverage(5)
```

Le paramètre « taps » est le nombre d'échantillons qui seront inclus dans la moyenne mobile. Cela se comporte de manière similaire au paramètre « time constant » ci-dessus - la constante de temps effective est le nombre d'échantillons multiplié par la période à laquelle `calculate()` est appelé (typiquement 20mSec).

Filter passe-haut (highPass)



La méthode `highPass` crée un simple filtre passe-haut à réponse impulsionnelle infinie de premier ordre. C'est le « pendant » du *singlePoleIIR*.

JAVA

```
// Creates a new high-pass IIR filter
// Time constant is 0.1 seconds
// Period is 0.02 seconds - this is the standard FRC main loop period
LinearFilter filter = LinearFilter.highPass(0.1, 0.02);
```

C++

```
// Creates a new high-pass IIR filter
// Time constant is 0.1 seconds
// Period is 0.02 seconds - this is the standard FRC main loop period
frc::LinearFilter<double> filter = frc::LinearFilter<double>::HighPass(0.1_s, 0.02_s);
```

PYTHON

```
from wpimath.filter import LinearFilter

# Creates a new high-pass IIR filter
# Time constant is 0.1 seconds
# Period is 0.02 seconds - this is the standard FRC main loop period
filter = LinearFilter.highPass(0.1, 0.02)
```

Le paramètre « time constant » détermine « l'échelle de temps caractéristique » de la réponse impulsionnelle du filtre ; le filtre annulera toute dynamique de signal qui se produit sur des échelles de temps beaucoup plus longues que définie par ce paramètre. De même, il s'agit également de la durée approximative de *l'avance de phase* généré. L'inverse de cette échelle de temps, multiplié par 2 pi, est la « fréquence de coupure » du filtre.

Le paramètre « période » est la période de temps entre les appels successifs de la méthode `calculate()` propre au filtre. Pour la grande majorité des implémentations, ce sera la période standard de boucle du robot principal, soit 20 mSec.

Utilisation d'un LinearFilter

Note : Pour que le filtre créé obéisse au paramètre d'échelle de temps spécifié, et fonctionne correctement, sa fonction `calculate()` *doit* être appelée régulièrement à la période spécifiée. Si, pour une raison quelconque, une interruption significative des appels `calculate()` doit se produire, la méthode `reset()` du filtre doit être appelée avant de réutiliser le filtre.

Une fois votre filtre créé, son utilisation est simple - il suffit d'appeler la méthode `calculate()` avec l'entrée la plus récente pour obtenir la sortie filtrée :

JAVA

```
// Calculates the next value of the output
filter.calculate(input);
```

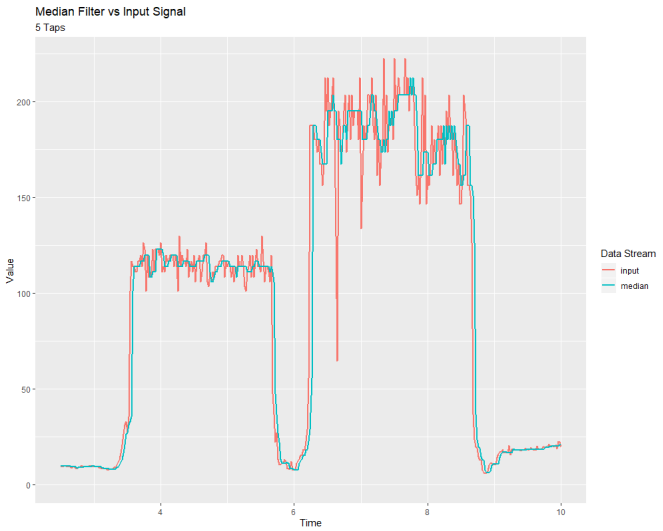
C++

```
// Calculates the next value of the output
filter.Calculate(input);
```

PYTHON

```
# Calculates the next value of the output
filter.calculate(input)
```

32.3.3 Filtre médian



A *statistically robust* alternative to the *moving-average filter* is the *median filter*. Where a moving average filter takes the arithmetic *mean* of the input over a moving sample window, a median filter (per the name) takes a median instead.

Le filtre médian est très utile pour supprimer les valeurs aberrantes quelquefois occasionnelles d'un flux d'entrée. Cela le rend particulièrement bien adapté au filtrage des entrées des capteurs de distance, qui sont sujets à des interférences de temps à autre. Contrairement à une moyenne mobile, le filtre médian ne sera pas affecté par un petit nombre de valeurs aberrantes, aussi extrêmes soient-elles.

The median filter is supported in WPILib through the `MedianFilter` class (Java, C++, , Python).

La création de MedianFilter

Note : La classe C++ `MedianFilter` est basée sur le type de données utilisé comme entrée.

Note : Étant donné que les filtres ont une «mémoire», chaque flux d'entrée nécessite son propre objet filtre. N'essayez *pas* d'utiliser le même objet filtre pour plusieurs flux d'entrée.

La création de `MedianFilter` est simple :

JAVA

```
// Creates a MedianFilter with a window size of 5 samples
MedianFilter filter = new MedianFilter(5);
```

C++

```
// Creates a MedianFilter with a window size of 5 samples
frc::MedianFilter<double> filter(5);
```

PYTHON

```
from wpimath.filter import MedianFilter

# Creates a MedianFilter with a window size of 5 samples
filter = MedianFilter(5)
```

L'utilisation de MedianFilter

Une fois votre filtre créé, son utilisation est simple - il suffit d'appeler la méthode `calculate()` avec l'entrée la plus récente pour obtenir la sortie filtrée :

JAVA

```
// Calculates the next value of the output
filter.calculate(input);
```

C++

```
// Calculates the next value of the output
filter.Calculate(input);
```

PYTHON

```
# Calculates the next value of the output
filter.calculate(input)
```

32.3.4 Limiteur de la vitesse de balayage (Slew Rate)

Une utilisation courante pour les filtres en FRC® est d'adoucir le comportement des entrées de contrôle (par exemple, les entrées joystick de vos commandes de pilote). Malheureusement, un simple filtre passe-bas est mal adapté à ce travail ; tandis qu'un filtre passe-bas adoucira la réponse d'un flux d'entrée aux changements soudains, il éliminera également les détails pour un contrôle fin et introduira un décalage de phase. Une meilleure solution consiste à limiter directement le taux de changement de l'entrée de contrôle. Ceci est effectué avec un *limiteur de la vitesse de balayage* - un filtre qui plafonne le taux maximum de changement du signal.

Un limiteur de vitesse de balayage peut être considéré comme une forme primitive de profil de mouvement. En effet, le limiteur de vitesse de balayage est l'équivalent de premier ordre du *Profil de mouvement trapézoïdal* supporté par WPILib - il est précisément le cas limite du mouvement trapézoïdal lorsque la contrainte d'accélération peut tendre vers l'infini. Par conséquent, le limiteur de vitesse de balayage est un bon choix pour appliquer un profil de mouvement de facto à un flux de données de vitesse (ou voltages, qui s'apparente à la vitesse). Pour les flux d'entrée qui contrôlent les positions, il est généralement préférable d'utiliser un profil trapézoïdal approprié.

Slew rate limiting is supported in WPILib through the `SlewRateLimiter` class (Java, C++, `Python`).

Création de SlewRateLimiter

Note : La classe C++ `SlewRateLimiter` est basée sur le type d'unité de l'entrée. Pour plus d'informations sur les unités C++, voir *La librairie d'unités C++*.

Note : Étant donné que les filtres ont une «mémoire», chaque flux d'entrée nécessite son propre objet filtre. N'essayez *pas* d'utiliser le même objet de filtre pour plusieurs flux d'entrée.

La création de SlewRateLimiter est simple :

JAVA

```
// Creates a SlewRateLimiter that limits the rate of change of the signal to 0.5␣
↪units per second
SlewRateLimiter filter = new SlewRateLimiter(0.5);
```

C++

```
// Creates a SlewRateLimiter that limits the rate of change of the signal to 0.5␣
↪volts per second
frc::SlewRateLimiter<units::volts> filter{0.5_V / 1_s};
```

PYTHON

```
from wpimath.filter import SlewRateLimiter

# Creates a SlewRateLimiter that limits the rate of change of the signal to 0.5 units␣
↪per second
filter = SlewRateLimiter(0.5)
```

Utilisation de SlewRateLimiter

Une fois votre filtre créé, son utilisation est simple - il suffit d'appeler la méthode `calculate()` avec l'entrée la plus récente pour obtenir la sortie filtrée :

JAVA

```
// Calculates the next value of the output
filter.calculate(input);
```

C++

```
// Calculates the next value of the output
filter.Calculate(input);
```

PYTHON

```
# Calculates the next value of the output
filter.calculate(input)
```

Utilisation de SlewRateLimiter avec la classe DifferentialDrive

Note : The C++ example below templates the filter on `units::scalar` for use with doubles, since joystick values are typically dimensionless.

Une utilisation typique d'un SlewRateLimiter est de limiter l'accélération de l'entraînement d'un robot. Cela peut être particulièrement pratique pour les robots qui sont très lourds ou qui ont des entraînements très puissants. Pour ce faire, appliquez un SlewRateLimiter à une valeur passée dans votre fonction d'entraînement de robot :

JAVA

```
// Ordinary call with no ramping applied
drivetrain.arcadeDrive(forward, turn);

// Slew-rate limits the forward/backward input, limiting forward/backward acceleration
drivetrain.arcadeDrive(filter.calculate(forward), turn);
```

C++

```
// Ordinary call with no ramping applied
drivetrain.ArcadeDrive(forward, turn);

// Slew-rate limits the forward/backward input, limiting forward/backward acceleration
drivetrain.ArcadeDrive(filter.Calculate(forward), turn);
```

PYTHON

```
# Ordinary call with no ramping applied
drivetrain.arcadeDrive(forward, turn)

# Slew-rate limits the forward/backward input, limiting forward/backward acceleration
drivetrain.arcadeDrive(filter.calculate(forward), turn)
```

32.3.5 Rebondisseur

Un filtre anti-rebond est un filtre utilisé pour éliminer les cycles de marche/arrêt rapides indésirables (appelés « rebonds », à l'origine des vibrations physiques d'un interrupteur lorsqu'il est lancé). Ces cycles sont généralement dus à une erreur du capteur comme le bruit ou les réflexions et non au phénomène physique réel que le capteur essaie d'enregistrer.

Debouncing is implemented in WPILib by the `Debouncer` class (Java, C++, Python), which filters a boolean stream so that the output only changes if the input sustains a change for some nominal time period.

Modes

L'objet `Debouncer` WPILib peut être configuré dans trois modes différents :
- Rising ou en montant (état par défaut) : active l'anti-rebonds pendant les fronts montants (transitions de l'état logique *false* à l'état logique *true*) uniquement.
- Falling ou en descendant : active l'anti-rebonds pendant les fronts descendants (transitions de l'état logique *true* à l'état logique *false*) uniquement.
- Both ou les deux : active l'anti-rebonds pendant toutes les transitions.

Utilisation

JAVA

```
// Initializes a DigitalInput on DIO 0
DigitalInput input = new DigitalInput(0);

// Creates a Debouncer in "both" mode.
Debouncer m_debouncer = new Debouncer(0.1, Debouncer.DebounceType.kBoth);

// So if currently false the signal must go true for at least .1 seconds before being␣
↪read as a True signal.
if (m_debouncer.calculate(input.get())) {
    // Do something now that the DI is True.
}
```

C++

```
// Initializes a DigitalInput on DIO 0
frc::DigitalInput input{0};

// Creates a Debouncer in "both" mode.
frc::Debouncer m_debouncer{100_ms, frc::Debouncer::DebounceType::kBoth};

// So if currently false the signal must go true for at least .1 seconds before being␣
↪read as a True signal.
if (m_debouncer.calculate(input.Get())) {
    // Do something now that the DI is True.
}
```

PYTHON

```
from wpilib import DigitalInput
from wpimath.filter import Debouncer

# Initializes a DigitalInput on DIO 0
self.input = DigitalInput(0)

# Creates a Debouncer in "both" mode with a debounce time of 0.1 seconds
self.debouncer = Debouncer(0.1, Debouncer.DebounceType.kBoth)
```

(suite sur la page suivante)

(suite de la page précédente)

```
# If currently false, the signal must go true for at least 0.1 seconds before being␣
↪read as a True signal.
if self.debouncer.calculate(self.input.get()):
    # Do something now that the DI is True.
    pass
```

32.4 Classes Géométriques

Cette section couvre les Classes Géométriques incluses dans WPILib.

32.4.1 Transfert, Rotation et Pose

Transfert

Translation in 2 dimensions is represented by WPILib's `Translation2d` class (Java, C++, Python). This class has an x and y component, representing the point (x, y) or the vector $\begin{bmatrix} x \\ y \end{bmatrix}$ on a 2-dimensional coordinate system.

Vous pouvez obtenir la distance à l'autre objet `Translation2d` en utilisant le `getDistance(Translation2d other)`, qui calcule la distance vectorielle entre les deux Translation2d en utilisant le théorème de Pythagore

---

Note : `Translation2d` uses the C++ Units library. If you're planning on using other WPILib classes that use `Translation2d` in Java/Python, such as the trajectory generator, make sure to use meters.

Rotation

Rotation in 2 dimensions is represented by WPILib's `Rotation2d` class (Java, C++, Python). This class has an angle component, which represents the robot's rotation relative to an axis on a 2-dimensional coordinate system. Positive rotations are counterclockwise.

Note : `Rotation2d` uses the C++ Units library. The constructor in Java/Python accepts either the angle in radians, or the sine and cosine of the angle, but the `fromDegrees` method will construct a `Rotation2d` object from degrees.

Note : `Rotation2d` n'enveloppe (wrap) pas la valeur de l'angle, donc si une valeur de 400 degrés est passée au constructeur, alors 400 degrés seront renvoyés dans les appels de valeur suivants.

Pose

Pose is a combination of both translation and rotation and is represented by the `Pose2d` class (Java, C++, Python). It can be used to describe the pose of your robot in the field coordinate system, or the pose of objects, such as vision targets, relative to your robot in the robot coordinate system. `Pose2d` can also represent the vector $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$.

32.4.2 Transformations

Translation2d

Les opérations sur un `Translation2d` exécutent des opérations sur le vecteur representé par le `Translation2d`.

- Addition : Addition between two `Translation2d` a and b can be performed using `plus` in Java, or the + operator in C++/Python. Addition adds the two vectors.
- Subtraction : Subtraction between two `Translation2d` can be performed using `minus` in Java, or the binary - operator in C++/Python. Subtraction subtracts the two vectors.
- Multiplication : Multiplication of a `Translation2d` and a scalar can be performed using `times` in Java, or the * operator in C++/Python. This multiplies the vector by the scalar.
- Division : Division of a `Translation2d` and a scalar can be performed using `div` in Java, or the `/` operator in C++/Python. This divides the vector by the scalar.
- Rotation : Rotation d'un `Translation2d` de façon anti-horaire en utilisant un angle de rotation θ autour de l'origine. Cela peut être exécuté en utilisant `rotateBy`, et équivaut à multiplier le vecteur par la matrice $\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$
- Additionally, you can rotate a `Translation2d` by 180 degrees by using `unaryMinus` in Java, or the unary - operator in C++/Python.

Rotation2d

Transformations sur un `Translation2d` sont juste des opérations arithmétiques sur la mesure d'angle representé par le `Rotation2d`.

- `plus` (Java) or + (C++/Python) : Adds the rotation component of `other` to this `Rotation2d`'s rotation component
- `minus` (Java) or binary - (C++/Python) : Subtracts the rotation component of `other` to this `Rotation2d`'s rotation component
- `unaryMinus` (Java) or unary - (C++/Python) : Multiplies the rotation component by a scalar of -1.
- `times` (Java) or * (C++/Python) : Multiplies the rotation component by a scalar.

Transform2d et Twist2d

WPILib provides 2 classes, `Transform2d` (Java, C++, Python), which represents a transformation to a pose, and `Twist2d` (Java, C++, Python) which represents a movement along an arc. `Transform2d` and `Twist2d` all have x, y and θ components.

`Transform2d` représente une transformation **relative**. Elle a deux composantes : une de transfert et l'autre de rotation. La transformation d'un `Pose2d` par un `Transform2d` s'effectue en faisant pivoter le composant « translation » du transfert selon le paramètre de rotation de la pose, et ensuite on ajoute le composant transfert tourné et le composant de rotation de pose. En d'autre termes, `Pose2d.plus(Transform2d)` revient $\begin{bmatrix} x_p \\ y_p \\ \theta_p \end{bmatrix} + \begin{bmatrix} cos\theta_p & -sin\theta_p & 0 \\ sin\theta_p & cos\theta_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$

`Twist2d` représente un changement en distance le long d'un arc. Généralement, cette classe est utilisé pour représenter le mouvement de du chassis du robot, où le composant x est la distance parcourue vers l'avant, le composant y est la distance parcourue vers le côté, (le côté gauche donne des valeurs positives), et le composant θ est le changement de cap (pivot sur l'axe central en Z du robot). Les mathématiques sous-jacentes qui expliquent les calculs de l'exponentielle de pose (nouvelle position qui suit la courbure de la fonction « Twist2d ») peuvent être trouvé ici , au chapitre 10.

Note : Pour les entraînements non holonomiques (comme le mode Tank), le composant y d'un `Twist2d` devrait toujours être 0.

Both classes can be used to estimate robot location. Twist2d is used in WPILib's *odometry* classes to update the robot's *pose* based on movement, while Transform2d can be used to estimate the robot's global position from vision data.

32.5 System Identification

32.5.1 Introduction to System Identification

What is « System Identification ? »

In Control Theory, *system identification* is the process of determining a mathematical model for the behavior of a system through statistical analysis of its inputs and outputs.

This model is a rule describing how input voltage affects the way our measurements (typically encoder data) evolve in time. A « system identification » routine takes such a model and a dataset and attempts to fit parameters which would make your model most closely-match the dataset. Generally, the model is not perfect - the real-world data are polluted by both measurement noise (e.g. timing errors, encoder resolution limitations) and system noise (unmodeled forces acting on the system, like vibrations). However, even an imperfect model is usually « good enough » to give us accurate *feedforward control* of the mechanism, and even to estimate optimal gains for *feedback control*.

Assumed Behavioral Model

If you haven't yet, read the full explanation of the feedforward equations used by the WPILib toolsuite in *The Permanent-Magnet DC Motor Feedforward Equation*.

The process of System Identification is to determine concrete values for the coefficients in the model that best-reflect the behavior of *your particular* real-world system.

To determine numeric values for each coefficient in our model, a curve-fitting technique (such as *least-squares regression*) is applied to measurements taken from the real mechanism. Careful selection of the data-producing experiments helps improve the accuracy of the curve-fitting.

Once these coefficients have been determined, we can then take a given desired velocity and acceleration for the motor and calculate the voltage that should be applied to achieve it. This is very useful - not only for, say, following motion profiles, but also for making mechanisms more controllable in open-loop control, because your joystick inputs will more closely match the actual mechanism motion.

Some of the tools in this toolsuite introduce additional terms into the above equation to account for known differences from the simple case described above - details for each tool can be found below :

The WPILib System Identification Tool (SysId)

The WPILib system identification tool consists of the SysId application that runs on the user's PC and a routine that lives in the code running on the user's robot. The routine will generate control signals which user-defined callbacks will send to the motors being characterized, while the robot records data into a log file. After the routine completes, the user will retrieve this file from the roboRIO and load it into SysId. SysId then processes the data and determines model parameters for the user's robot mechanism, as well as producing diagnostic plots.

Included Tools

Note : With a bit of ingenuity, these tools can be used to accurately characterize a surprisingly large variety of robot mechanisms. Even if your mechanism does not seem to obviously match any of the tools, an understanding of the system equations often reveals that one of the included routines will do.

The System Identification toolsuite currently supports :
- Simple Motor Setups
- Elevators
- Arms

Several of these options use nearly identical robot-side code, and differ only in the analysis used by SysId to interpret the data.

Simple Motor Identification

The simple motor identification tool determines the best-fit parameters for the equation :

$$V = kS \cdot sgn(\dot{d}) + kV \cdot \dot{d} + kA \cdot \ddot{d}$$

where V is the applied voltage, d is the displacement (position) of the drive, $\dot{d}$ is its velocity, and $\ddot{d}$ is its acceleration. This is the model for a permanent-magnet dc motor with no loading other than friction and inertia, as mentioned above, and is an accurate model for flywheels, turrets, and horizontal linear sliders.

Elevator Identification

The elevator identification tool determines the best-fit parameters for the equation :

$$V = kG + kS \cdot sgn(\dot{d}) + kV \cdot \dot{d} + kA \cdot \ddot{d}$$

where V is the applied voltage, d is the displacement (position) of the elevator, $\dot{d}$ is its velocity, and $\ddot{d}$ is its acceleration. The constant term (kG) is added to correctly account for the effect of gravity.

Arm Identification

The arm identification tool determines the best-fit parameters for the equation :

$$V = kG \cdot cos(\theta) + kS \cdot sgn(\dot{\theta}) + kV \cdot \dot{\theta} + kA \cdot \ddot{\theta}$$

where V is the applied voltage, θ is the angular displacement (position) of the arm, $\dot{\theta}$ is its angular velocity, and $\ddot{\theta}$ is its angular acceleration. The cosine term (kG) is added to correctly account for the effect of gravity.

Installing SysId

SysId is included with the WPILib Installer.

Note : The old Python characterization tool from previous years is no longer supported.

Launching the SysId Tool

The system identification tool can be opened from the `Start Tool` option in VS Code or by using the shortcut inside the WPILib Tools desktop folder (Windows).

32.5.2 Creating an Identification Routine

Types of Tests

A standard motor identification routine consists of two types of tests :

- **Quasistatic :** In this test, the mechanism is gradually sped-up such that the voltage corresponding to acceleration is negligible (hence, « as if static »).
- **Dynamic :** In this test, a constant “step voltage” is given to the mechanism, so that the behavior while accelerating can be determined.

Each test type is run both forwards and backwards, for four tests in total. The tests can be run in any order, but running a « backwards » test directly after a « forwards » test is generally advisable (as it will more or less reset the mechanism to its original position). `SysIdRoutine` provides command factories that may be used to run the tests, for example as part of an autonomous routine. Previous versions of SysId used a project generator to create and deploy robot code to run these tests, but it proved to be very fragile and difficult to maintain. The user code-based workflow enables teams to use mechanism code they already know works, including soft and hard limits.

User Code Setup

Note : Some familiarity with your language’s units library is recommended and knowing how to use Consumers is required. Ths page assumes you are using the Commands framework.

To assist in creating SysId-compatible identification routines, WPILib provides the `SysIdRoutine` class. Users should create a `SysIdRoutine` object, which take both a `Config` object describing the test settings and a `Mechanism` object describing how the routine will control the relevant motors and log the measurements needed to perform the fit.

Routine Config

The `Config` object takes in a a voltage ramp rate for use in Quasistatic tests, a steady state step voltage for use in Dynamic tests, a time to use as the maximum test duration for safety reasons, and a callback method that accepts the current test state (such as « dynamic-forward ») for use by a 3rd party logging solution. The constructor may be left blank to default the ramp rate to 1 volt per second and the step voltage to 7 volts.

Note : Not all 3rd party loggers will interact with SysIdRoutine directly. CTRE users who do not wish to use SysIdRoutine directly for logging should use the SignalLogger API and use Tuner X to convert to wpilog. REV users may use Team 6328’s Unofficial REV-Compatible Logger (URCL). In both cases the log callback should be set to `null`. Once the log file is in hand, it may be used with SysId just like any other.

The timeout and state callback are optional and defaulted to 10 seconds and null (which will log the data to a normal WPILog file) respectively.

Declaring the Mechanism

The `Mechanism` object takes a voltage consumer, a log consumer, the subsystem being characterized, and the name of the mechanism (to record in the log). The drive callback takes in the routine-generated voltage command and passes it to the relevant motors. The log callback reads the motor voltage, position, and velocity for each relevant motor and adds it to the running log. The subsystem is required so that it may be added to the requirements of the routine commands. The name is optional and will be defaulted to the string returned by getName().

The callbacks can either be created in-place via Lambda expressions or can be their own standalone functions and be passed in via method references. Best practice is to create the routine and callbacks inside the subsystem, to prevent leakage.

JAVA

```
// Creates a SysIdRoutine
SysIdRoutine routine = new SysIdRoutine(
    new SysIdRoutine.Config(),
    new SysIdRoutine.Mechanism(this::voltageDrive, this::logMotors, this)
);
```

Mechanism Callbacks

The `Mechanism` callbacks are essentially just plumbing between the routine and your motors and sensors.

The `drive` callback exists so that you can pass the requested voltage directly to your motor controller(s).

The `log` callback reads sensors so that the routine can log the voltage, position, and velocity at each timestep.

See the SysIdRoutine (Java, C++) example project for example callbacks.

Test Factories

To be able to run the tests, SysIdRoutine exposes test « factories », or functions that each return a command that will execute a given test.

JAVA

```
public Command sysIdQuasistatic(SysIdRoutine.Direction direction) {
  return routine.quasistatic(direction);
}

public Command sysIdDynamic(SysIdRoutine.Direction direction) {
  return routine.dynamic(direction);
}
```

Either bind the factory methods to either controller buttons or create an autonomous routine with them. It is recommended to bind them to buttons that the user must hold down for the duration of the test so that the user can stop the routine quickly if it exceeds safe limits.

32.5.3 Running the Identification Routine

Once the code has been deployed, we can now run the system identification routine, and record the resulting data for analysis.

Note : Ensure you have sufficient space around the robot before running any identification routine ! The drive identification requires at least 10” of space, ideally closer to 20”. The robot drive can not be accurately characterized while on blocks.

Avertissement : Only log files with a single routine in them are usable for analysis. Multiple motors can be run in one routine, but they must be run at the same time. If you run a routine on one motor and then run a routine on another motor without extracting the log or power-cycling the roboRIO in between, analysis will fail.

Running Tests

Perform the tests using the bindings you created in the previous section.

Avertissement : Watch out for your mechanism and stop the test early if it exceeds safe limits ! The routine only creates voltage commands for you to connect to your motors, it is up to you to set up hard or soft limits to prevent injury or damage.

The entire routine should look something like this :

Note : A drivetrain routine is shown below, but the same motions will occur on any mechanism.

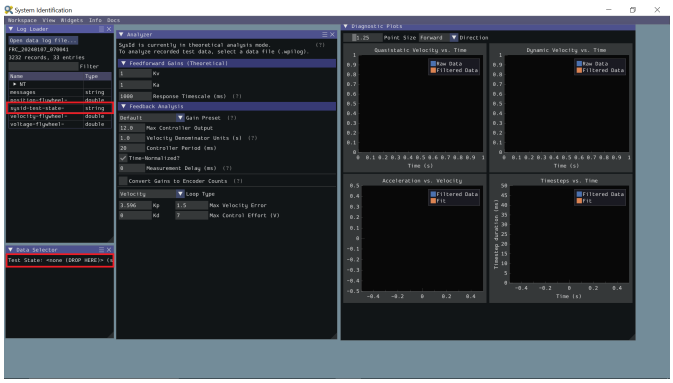
After all four tests have been completed, use the `DataLogTool` to retrieve the log file from the roboRIO.

32.5.4 Loading Data

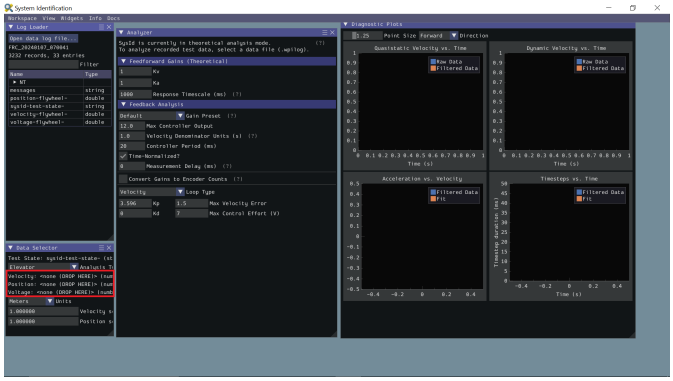
After downloading the WPILog containing the tests from the roboRIO, go to the `Log Loader` pane in SysId and click `Open data log file...`.

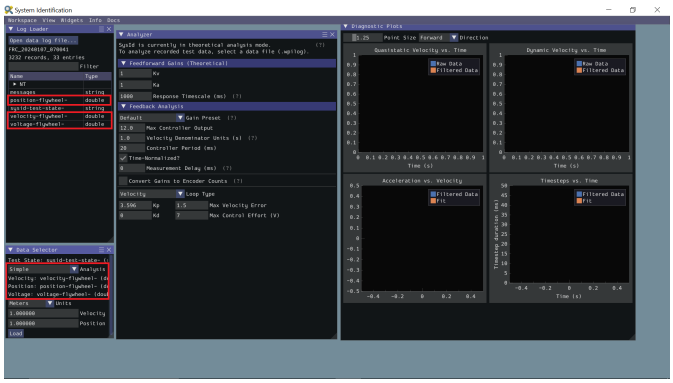
After the file loads, look for a `string` type entry with a name containing « state ». Drag this entry into the Data Selector pane’s Test State slot.

Note : SysIdRoutine will name the entry « sysid-test-state-mechanism », where « mechanism » is the name passed to the `Mechanism` contructor or the subsystem name.

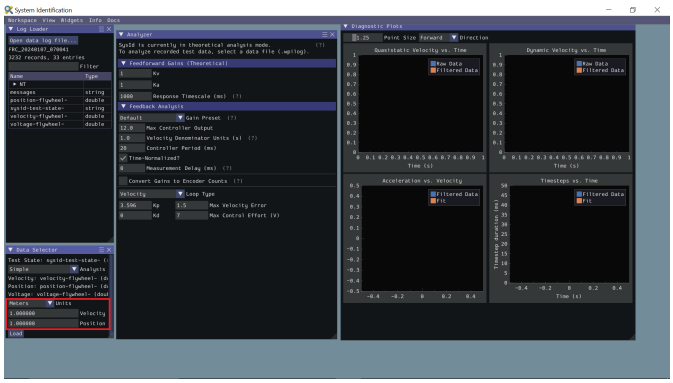


Now the `Data Selector` pane will present `Position`, `Velocity`, and `Voltage` slots. In the `Log Loader` pane, find entries starting with each of those terms and containing the motor name you set in the `log` callback, and drag those into the `Data Selector` slots.





Ideally, the correct units for the position and velocity entries would have been set in the code before running the tests. If this was not the case, use the Units dropdown in the `Data Selector` pane to correct it. Additionally, if you did not account for a gear ratio or some other factor that scales the recorded values up or down uniformly, you can compensate for that by setting position and velocity scaling factors in the provided boxes.

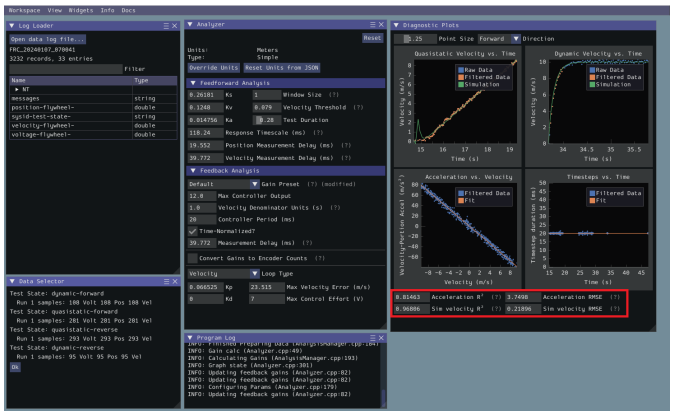


Ensure the correct analysis type has been selected, then click the `Load` button and move on to checking the fit diagnostics in the `Diagnostics` pane.

32.5.5 Viewing Diagnostics

Goodness-of-Fit Metrics

There are three numerical accuracy metrics that are computed with this tool : acceleration *r-squared*, simulated velocity r-squared, and the simulated velocity *RMSE*.



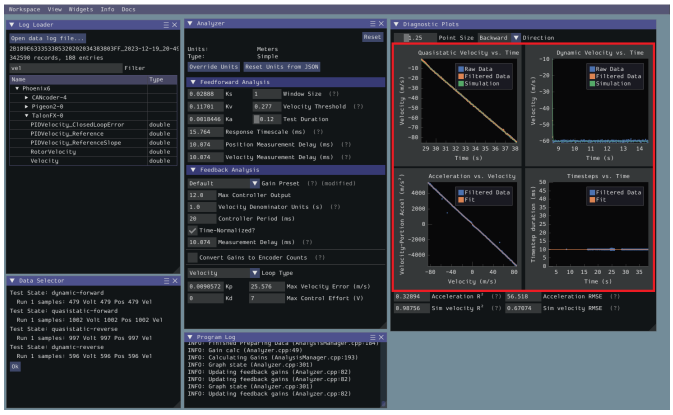
The acceleration r-squared is the fraction of the variance in measured acceleration (used as the independent variable in the SysId regression) explained by the linear model. This can be quite variable, because acceleration is very susceptible to system noise. Mechanisms tend to vibrate quite a bit, so this value rarely goes above `0.5`, even on very good datasets. If the acceleration r-squared goes below around `0.2`, the kA gain will be of dubious quality and the mechanism vibration should be reduced if possible. Even if your r-squared is outside this range it may still be valid, but it is recommended to improve the data if practical.

The simulated velocity r-squared is the fraction of the variance in measured velocity explained by a noiseless simulation of the motor movement stepped forward with the constants determined from the regression. A value north of `.9` indicates a good fit.

The simulated velocity RMSE is the standard deviation of the velocity error from the simulated model. This is a good estimation of the amount of process noise present during the test routine, and can be used as a low-end estimate for the model noise term in *state-space control*.

Diagnostic Plots

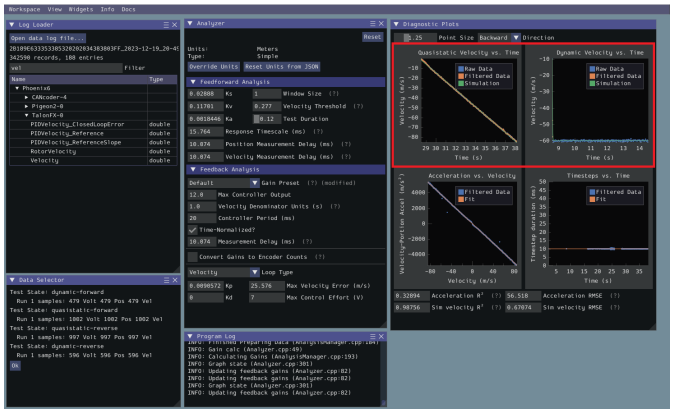
SysId also produces several diagnostic plots to help users evaluate the quality of their model fit.



Time-Domain Plots

Note : To improve plot quality, the diagnostic plots are separated by direction. Be sure to view both the forward *and* backward plots when troubleshooting !

The Time-Domain Diagnostics plots display velocity versus time over the course of the analyzed tests. These should look something like this :



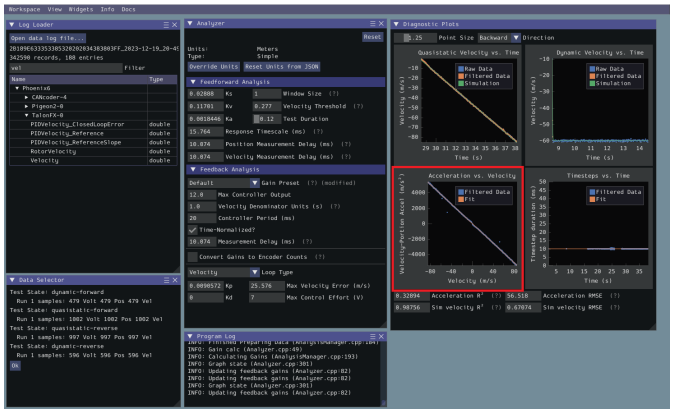
The velocity time domain plots contain three sets of data : Raw Data, Filtered Data, and Simulation. The Raw Data is the recorded data from your robot, the Filtered Data is the data after a median filter has been applied to the data, and the Simulation represents the velocity predictions of a model based off of the feedforward gains from the tool (these are used to calculate the « sim » error metrics mentioned above).

A successful quasistatic graph will be very nearly linear, while a successful dynamic graph will be an approximately exponential approach of the steady-speed.

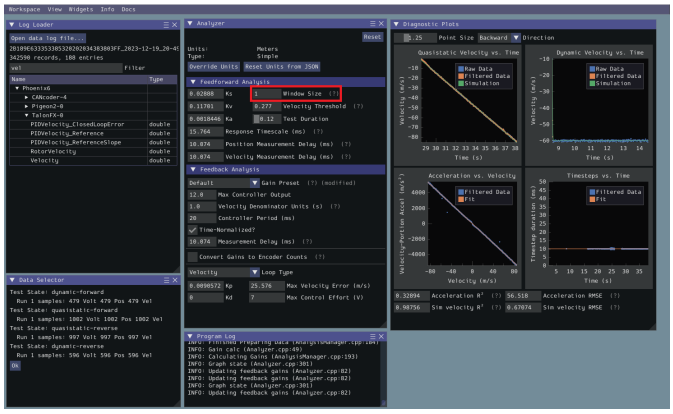
Deviation from this behavior is a sign of an *error*, either in your robot setup, analysis settings, or your test procedure.

Acceleration-Velocity Plot

The acceleration-versus-velocity plot displays the mechanism velocity versus the portion of acceleration corresponding to factors other than friction (ideally, this would leave only back-EMF) and applied voltage across *all* of the tests.



This plot should be quite linear, with patches of relatively noiseless quasistatic data intermixed with quite-noisy dynamic data. The noise on the dynamic sections of the plot may be reduced by increasing the *Window Size* setting.



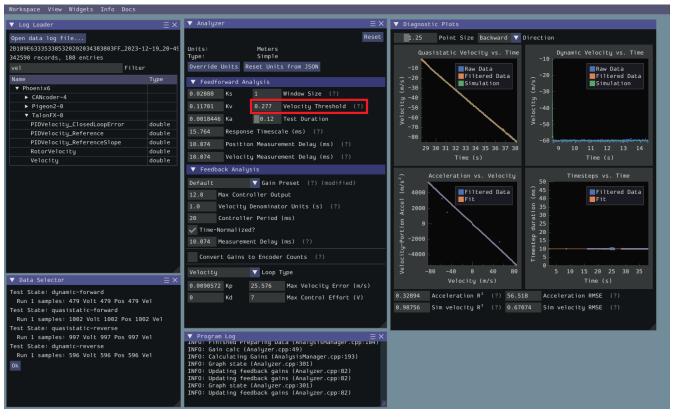
However, if your robot or mechanism has low mass compared to the motor power, this may « eat » what little meaningful acceleration data you have. In these cases kA will tend towards zero and can be ignored for feedforward purposes. However, if kA cannot be accurately measured, the calculated feedback gains are likely to be inaccurate, and manual tuning may be required.

Common Failure Modes

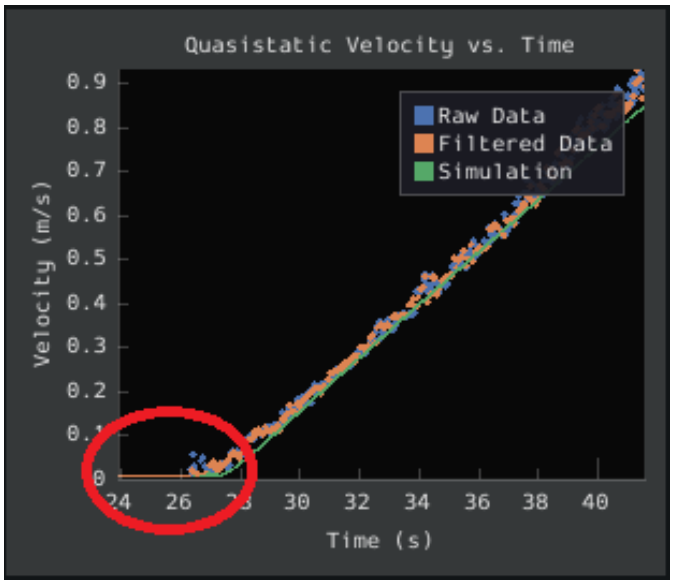
When something has gone wrong with the identification, diagnostic plots and console output provide crucial clues as to *what* has gone wrong. This section describes some common failures encountered while running the system identification tool, the identifying features of their diagnostic plots, and the steps that can be taken to fix them.

Improperly Set Motion Threshold

One of the most-common errors is an inappropriate value for the motion threshold.



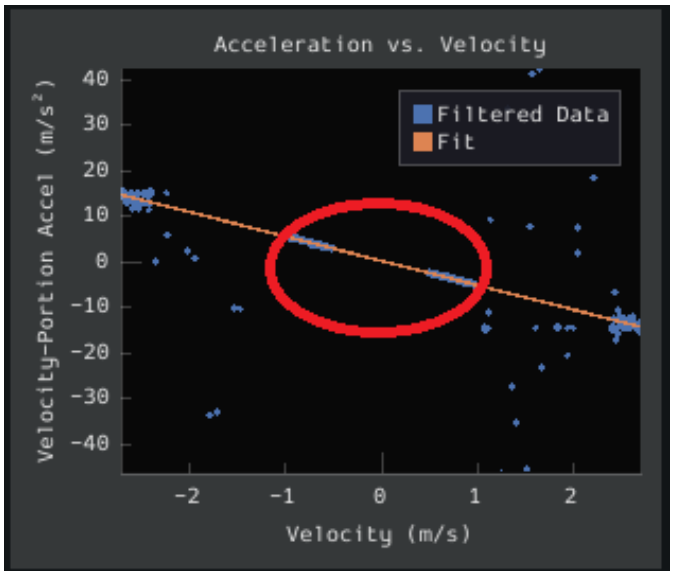
Velocity Threshold Too Low



The presence of a « leading tail » (emphasized by added red circle) in the quasistatic time-domain plot indicates that the *Velocity Threshold* setting is too low, and thus data points from before the robot begins to move are being included.

To solve this, increase the velocity threshold and re-analyze the data.

Motion Threshold Too High

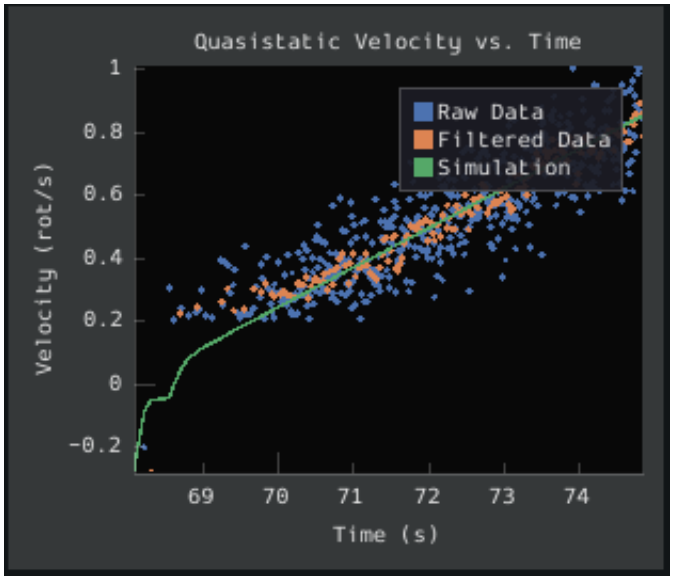


While not nearly as problematic as a too-low threshold, a velocity threshold that is too high will result in a large « gap » in the acceleration-versus-velocity plot.

To solve this, decrease the velocity threshold and re-analyze the data.

Noisy Velocity Signals

Note : There are two types of noise that affect mechanical systems - signal noise and system noise. Signal noise corresponds to measurement error, while system noise corresponds to actual physical motion that is unaccounted-for by your model (e.g. vibration). If SysId suggests that your system is noisy, you must figure out which of the two types of noise is at play - signal noise is often easier to eliminate than system noise.



Many FRC setups suffer from poorly-installed encoders - errors in shaft concentricity (for optical encoders) and magnet location (For magnetic encoders) can both contribute to noisy velocity signals, as can inappropriate filtering settings. Encoder noise will be immediately visible in your diagnostic plots, as can be seen above. Encoder noise is especially common on the toughbox mini gearboxes provided in the kit of parts.

System parameters can sometimes be accurately determined even from data polluted by encoder noise by increasing the window size setting. However, this sort of encoder noise is problematic for robot code much the same way it is problematic for the system identification tool. As the root cause of the noise is not known, it is recommended to try a different encoder setup if this is observed, either by moving the encoders to a different shaft, replacing them with a different type of encoder, or increasing the sample per average in project generation (adds an additional layer of filtering).

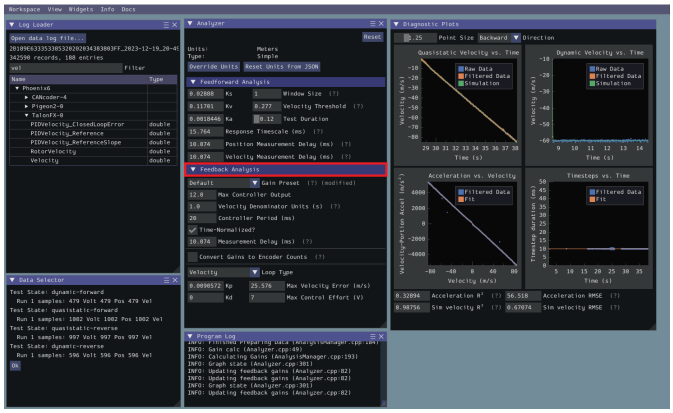
32.5.6 Analyzing Data

Feedforward Analysis

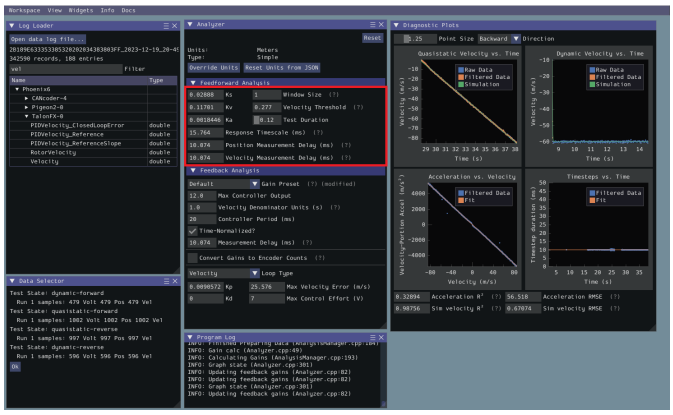
Note : For information on what the calculated feedback gains mean, see *The Permanent-Magnet DC Motor Feedforward Equation*. For information on using the calculated feedback gains in code, see *feedforward control*.

Click the dropdown arrow on the *Feedforward* Section.

Note : If you would like to change units, you will have to press the *Override Units* button and fill out the information on the popup.



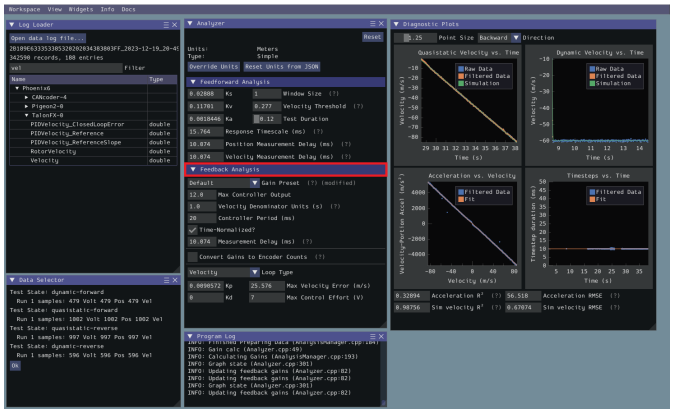
The computed mechanism system parameters will then be displayed.



Feedback Analysis

Important : These gains are, in effect, « educated guesses » - they are not guaranteed to be perfect, and should be viewed as a « starting point » for further tuning.

To view the feedback constants, click on the dropdown arrow on the *Feedback* section.



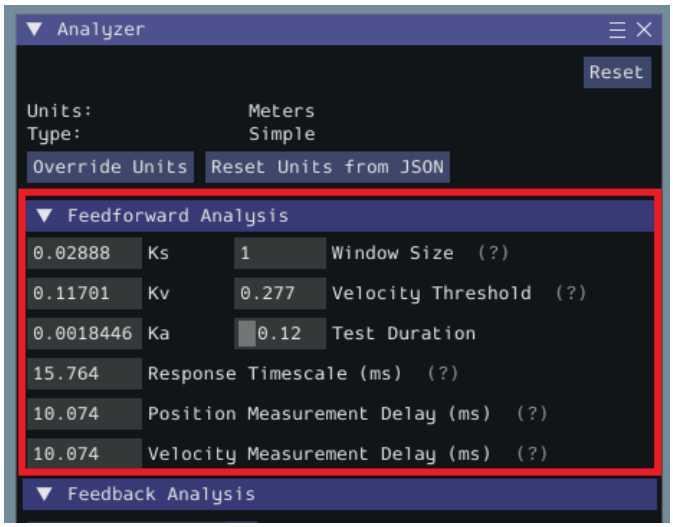
This view can be used to calculate optimal feedback gains for a PD or P controller for your mechanism (via *LQR*).

Enter Controller Parameters

Note : The « Spark Max » preset assumes that the user has configured the controller to operate in the units of analysis with the SPARK MAX API’s position/velocity scaling factor feature.

The calculated feedforward gains are *dimensioned quantities*. Unfortunately, not much attention is often paid to the units of PID gains in FRC® controls, and so the various typical options for PID controller implementations differ in their unit conventions (which are often not made clear to the user).

To specify the correct settings for your PID controller, use the following options.



— *Gain Settings Preset* This drop-down menu will auto-populate the remaining fields with likely settings for one of a number of common FRC controller setups. Note that some settings, such as post-encoder gearing, PPR, and the presence of a follower motor must still be manually specified (as the analyzer has no way of knowing these without user input), and that others may vary from the given defaults depending on user setup.
— *Controller Period* This is the execution period of the control loop, in seconds. The default RIO loop rate is 50Hz, corresponding to a period of 0.02s. The onboard controllers on most « smart controllers » run at 1Khz, or a period of 0.001s.
— *Max Controller Output* This is the maximum value of the controller output, with respect to the PID calculation. Most controllers calculate outputs with a maximum value of 1, but Talon controllers have a maximum output of 1023.
— *Time-Normalized Controller* This specifies whether the PID calculation is normalized to the period of execution, which affects the scaling of the D gain.
— *Controller Type* This specifies whether the controller is an onboard RIO loop, or is running on a smart motor controller such as a Talon or a SPARK MAX.
— *Post-Encoder Gearing* This specifies the gearing between the encoder and the mechanism itself. This is necessary for control loops that do not allow user-specified unit scaling in their PID computations (e.g. those running on Talons). This will be disabled if not relevant.
— *Encoder EPR* This specifies the edges-per-revolution (not cycles per revolution) of the encoder used, which is needed in the same cases as Post-Encoder Gearing.
— *Has Follower* Whether there is a motor controller following the controller running the control loop, if the control loop is being run on a peripheral device. This changes the effective loop period.
— *Follower Update Period* The rate at which the follower (if present) is updated. By default, this is 100Hz (every 0.01s) for the Talon SRX, Talon FX, and the SPARK MAX, but can be changed.

Note : If you select a smart motor controller as the preset (e.g. TalonSRX, SPARK MAX, etc.) the *Convert Gains* checkbox will be automatically checked. This means the tool will convert your gains so that they can be used through the smart motor controller's PID methods. Therefore, if you would like to use WPILib's PID Loops, you must uncheck that box.

Measurement Delays

Note : If you are using default smart motor controller settings or WPILib PID Control without additional filtering, SysId handles this for you.

Many « smart motor controllers » (such as the `Talon SRX`, `Venom`, `Talon FX`, and `SPARK MAX`) apply substantial *low-pass filtering* to their encoder velocity measurements, which can introduce a significant amount of phase lag. This can cause the calculated gains for velocity loops to be unstable. This can be accounted for with the *Measurement Delay* box.

However, the measurement delays have already been calculated for the default settings of the previously mentioned motor controllers so for most users this is handled by selecting the right preset in *Gain Settings Preset*.

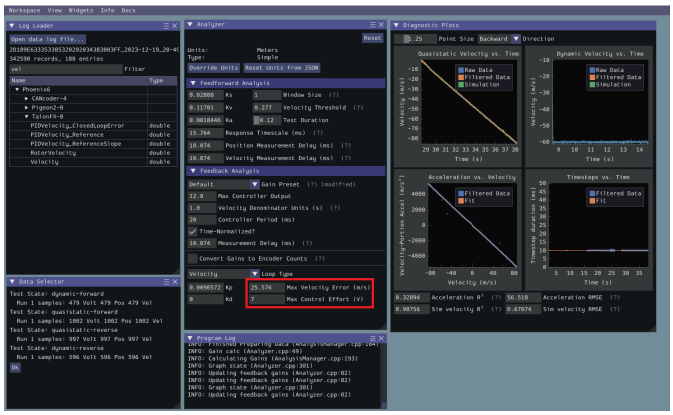
The following only applies if the user decides to implement their own custom filtering settings (e.g. adding a moving average filter to a WPILib PID loop or changing smart motorcontroller measurement period and/or measurement window size) as the measurement delay must be recalculated. Here is the general formula that can be used for filters with moving windows (e.g. median filter + moving average filter) :

$$d = \frac{T(n-1)}{2}$$

Where `T` is the period at which measurements are sampled (RIO default is 20 ms) and `n` is the size of the moving window used.

Specify Optimality Criteria

Finally, the user must specify what will be considered an « optimal » controller. This takes the form of desired tolerances for the system error and control effort - note that it is *not* guaranteed that the system will obey these tolerances at all times.

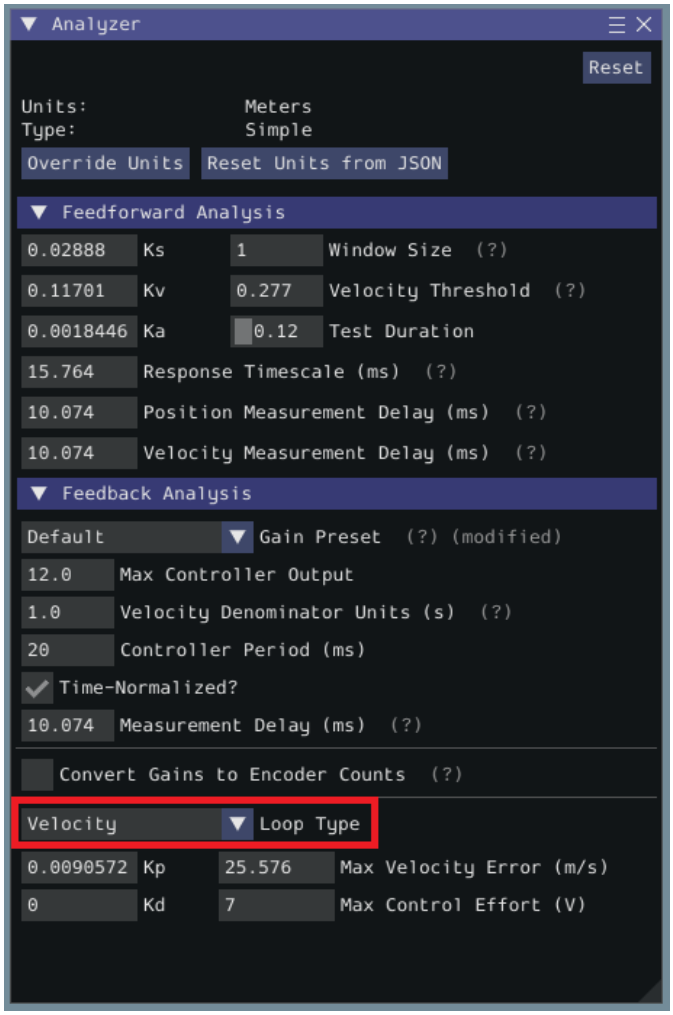


As a rule, smaller values for the *Max Acceptable Error* and larger values for the *Max Acceptable Control Effort* will result in larger gains - this will result in larger control efforts, which can grant better setpoint-tracking but may cause more violent behavior and greater wear on components.

The *Max Acceptable Control Effort* should never exceed 12V, as that corresponds to full battery voltage, and ideally should be somewhat lower than this.

Select Loop Type

It is typical to control mechanisms with both position and velocity PIDs, depending on application. Either can be selected using the drop-down *Loop Type* menu.



32.5.7 Utilitaires et outils supplémentaires

Cette page couvre principalement des informations utiles relatives aux fonctionnalités supplémentaires fournies par cet outil.

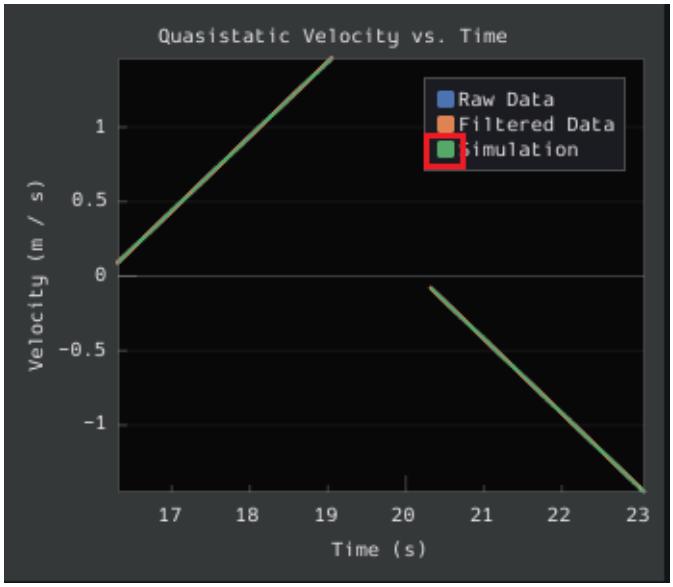
Conseils relatifs au framework ImGui

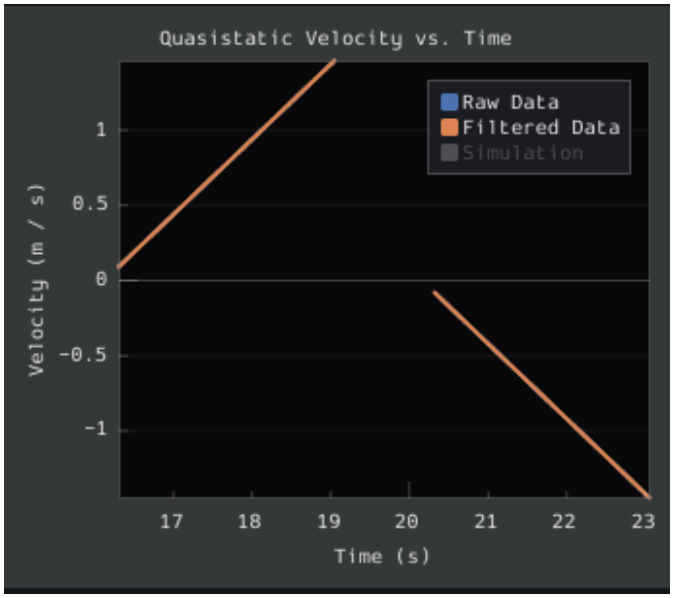
Les fonctionnalités suivantes sont essentiellement très pratiques avec le framework ImGui utilisé par SysId :

Affichage et masquage des données de tracé

To add or remove certain data from the plots, click on the color of the data that you would like to hide or remove.

Par exemple, si nous voulons masquer les données de simulation, nous pouvons cliquer sur la boîte de couleur verte.

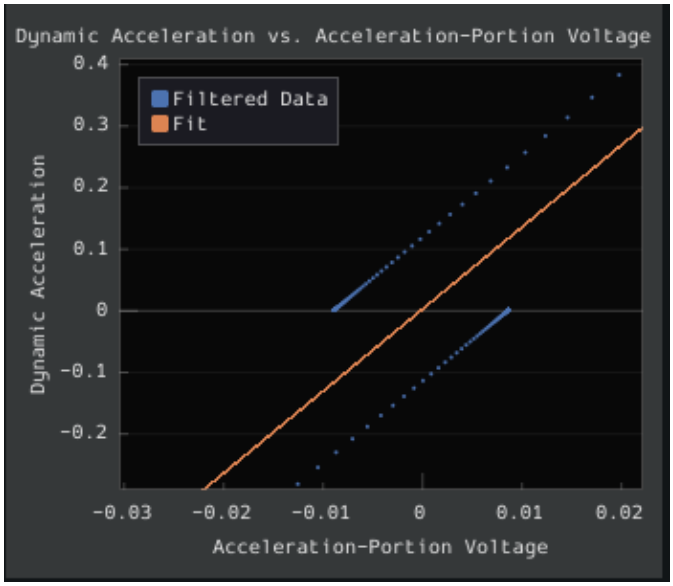




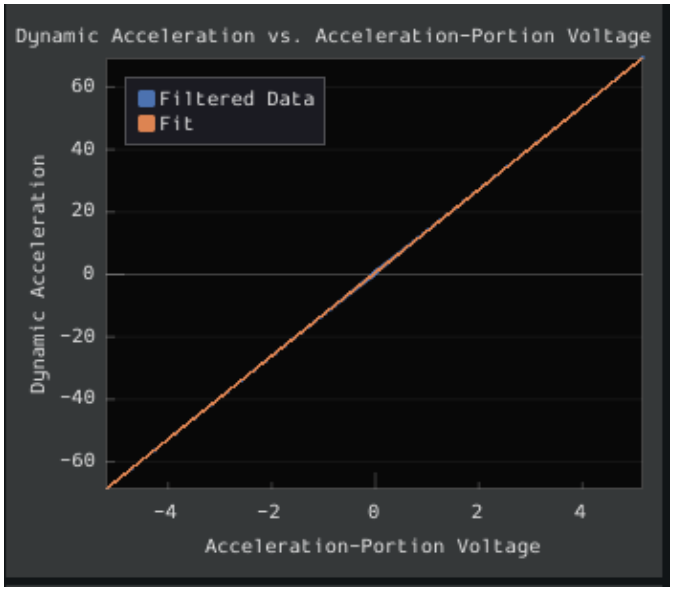
Auto Sizing Plots

If you zoom in to plots and want to revert back to the normally sized plots, just double click on the plot and it will automatically resize it.

Here is a plot that is zoomed in :



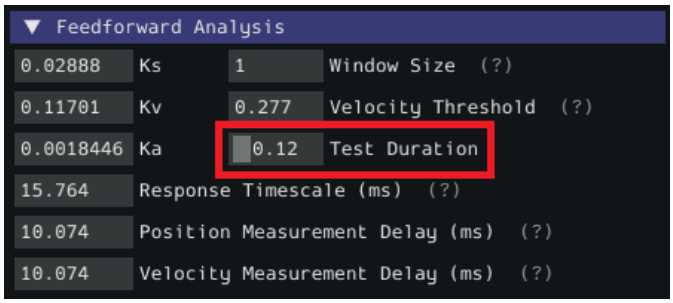
After double clicking, it is automatically resized :



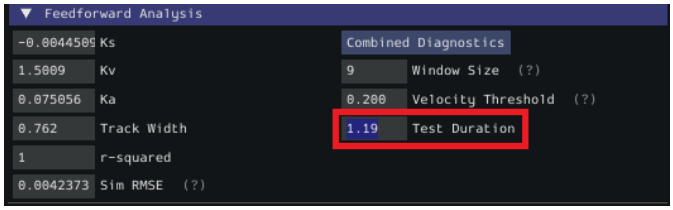
Setting Slider Values

To set the value of a slider as a number rather than sliding the widget, you can CTRL + Click the slider and it will allow you to input a number.

Here is a regular slider :



Here is the input after double clicking the slider :



32.6 Contrôleurs

Cette section décrit diverses classes de contrôleur de rétroaction et de feedforward WPILib qui sont utiles pour contrôler le mouvement des mécanismes du robot, ainsi que des classes de profilage de mouvement qui peuvent générer automatiquement des points de consigne à utiliser avec ces contrôleurs.

32.6.1 Contrôle PID dans WPILib

Note : This article focuses on in-code implementation of PID control in WPILib. For a conceptual explanation of the working of a PIDController, see *Une introduction aux PIDs*

Note : Pour un guide sur la mise en œuvre du contrôle PID via un *environnement basé sur les commandes*, regardez *Contrôle PID via PIDSubsystems et PIDCommands*.

WPILib supports PID control of mechanisms through the `PIDController` class (Java, C++, `Python`). This class handles the feedback loop calculation for the user, as well as offering methods for returning the error, setting tolerances, and checking if the control loop has reached its setpoint within the specified tolerances.

L'utilisation de la classe PIDController

La construction d'un objet PIDController

Note : Bien que l'objet `PIDController` puisse être utilisé de manière asynchrone, il ne fournit *aucune* fonctionnalité de sécurité des « threads » - la responsabilité du bon fonctionnement threadsafe est entièrement laissée à l'utilisateur. Par conséquant, l'utilisation asynchrone est recommandée uniquement pour les équipes avancées.

Afin d'utiliser la fonctionnalité de contrôle PID de WPILib, les utilisateurs doivent d'abord construire un objet `PIDController` avec les gains souhaités :

JAVA

```
// Creates a PIDController with gains kP, kI, and kD
PIDController pid = new PIDController(kP, kI, kD);
```

C++

```
// Creates a PIDController with gains kP, kI, and kD
frc::PIDController pid{kP, kI, kD};
```

PYTHON

```
from wpimath.controller import PIDController

# Creates a PIDController with gains kP, kI, and kD
pid = PIDController(kP, kI, kD)
```

Un quatrième paramètre facultatif peut être fourni au constructeur, spécifiant la période durant laquelle le contrôleur sera exécuté. L'objet `PIDController` est principalement destiné à une utilisation synchrone à partir de la boucle du robot principal, et cette valeur par défaut est donc de 20 mSec.

Utilisation de la sortie de boucle de rétroaction

Note : Le `PIDController` assume que la méthode `calculate()` est appelée régulièrement à un intervalle cohérent avec la période configurée. Le non-respect de cette consigne entraînera un comportement de boucle involontaire.

L'utilisation de `PIDController` est simple : il suffit d'appeler la méthode `calculate()` depuis la boucle principale du robot (par exemple la méthode `autonomousPeriodic()` du robot) :

JAVA

```
// Calculates the output of the PID algorithm based on the sensor reading
// and sends it to a motor
motor.set(pid.calculate(encoder.getDistance(), setpoint));
```

C++

```
// Calculates the output of the PID algorithm based on the sensor reading
// and sends it to a motor
motor.Set(pid.Calculate(encoder.GetDistance(), setpoint));
```

PYTHON

```
# Calculates the output of the PID algorithm based on the sensor reading
# and sends it to a motor
motor.set(pid.calculate(encoder.getDistance(), setpoint))
```

La vérification des erreurs

Note : `getPositionError()` et `getVelocityError()` sont définis avec l'hypothèse que la boucle contrôle une position - si toutefois la boucle contrôle plutôt une vitesse, ces dernières renvoient respectivement l'erreur de vitesse et l'erreur d'accélération.

L'erreur actuelle de la variable de processus mesurée est retournée par la fonction `getPositionError()` , tandis que sa dérivée est retournée par la fonction `getVelocityError ()` :

La spécification et la vérification des tolérances

Note : Si seulement une tolérance de position est spécifiée, la tolérance de vitesse par défaut est l'infini.

Note : Comme ci-dessus, «position» fait référence à la mesure de la variable de processus et «vitesse» à sa dérivée - ainsi, pour une boucle de vitesse, ce sont en fait la vitesse et l'accélération, respectivement.

Parfois, il est utile de savoir si un contrôleur a suivi le point de consigne dans une tolérance donnée - par exemple, pour déterminer si une commande doit être terminée ou (tout en suivant un profil de mouvement) si le mouvement est entravé et doit être re -prévu.

Pour ce faire, nous devons d'abord spécifier les tolérances avec la méthode `setTolerance()`; ensuite, nous pouvons le vérifier avec la méthode `atSetpoint()`.

JAVA

```
// Sets the error tolerance to 5, and the error derivative tolerance to 10 per second
pid.setTolerance(5, 10);

// Returns true if the error is less than 5 units, and the
// error derivative is less than 10 units
pid.atSetpoint();
```

C++

```
// Sets the error tolerance to 5, and the error derivative tolerance to 10 per second
pid.SetTolerance(5, 10);

// Returns true if the error is less than 5 units, and the
// error derivative is less than 10 units
pid.AtSetpoint();
```

PYTHON

```
# Sets the error tolerance to 5, and the error derivative tolerance to 10 per second
pid.setTolerance(5, 10)

# Returns true if the error is less than 5 units, and the
# error derivative is less than 10 units
pid.atSetpoint()
```

Réinitialisation du contrôleur

Il est parfois souhaitable d'effacer l'état interne (le paramètre le plus important étant l'accumulateur intégré) d'un `PIDController`, car il peut ne plus être valide (par exemple, lorsque le `PIDController` a été désactivé puis réactivé) . Cela peut être accompli en appelant la méthode `reset()`.

La définition d'une valeur d'intégrateur maximale

Note : Les intégrateurs introduisent l'instabilité et l'hystérésis dans les systèmes de boucle de rétroaction. Il est fortement recommandé aux équipes d'éviter d'utiliser le gain intégral sauf si aucune autre solution ne fera l'affaire - très souvent, les problèmes qui peuvent être résolus avec un intégrateur peuvent être mieux résolus en utilisant un outil plus précis, comme le *feedforward*.

Un problème typique rencontré lors de l'utilisation de la rétroaction intégrale est un « emballement » excessif qui fait que le système dépasse largement le point de consigne. Cela peut être atténué de plusieurs manières - la classe WPILib `PIDController` applique un limiteur de plage d'intégrateur pour aider les équipes à surmonter ce problème.

Par défaut, la contribution de sortie totale du gain intégral est limitée entre -1,0 et 1,0.

Les limites de plage peuvent être augmentées ou diminuées à l'aide de la méthode `setIntegratorRange()`.

JAVA

```
// The integral gain term will never add or subtract more than 0.5 from
// the total loop output
pid.setIntegratorRange(-0.5, 0.5);
```

C++

```
// The integral gain term will never add or subtract more than 0.5 from
// the total loop output
pid.SetIntegratorRange(-0.5, 0.5);
```

PYTHON

```
# The integral gain term will never add or subtract more than 0.5 from
# the total loop output
pid.setIntegratorRange(-0.5, 0.5)
```

Disabling Integral Gain if the Error is Too High

Another way integral « wind-up » can be alleviated is by limiting the error range where integral gain is active. This can be achieved by setting `IZone`. If the error is more than `IZone`, the total accumulated error is reset, disabling integral gain. When the error is equal to or less than IZone, integral gain is enabled.

By default, `IZone` is disabled.

`IZone` may be set using the `setIZone()` method. To disable it, set it to infinity.

JAVA

```
// Disable IZone
pid.setIZone(Double.POSITIVE_INFINITY);

// Integral gain will not be applied if the absolute value of the error is
// more than 2
pid.setIZone(2);
```

C++

```
// Disable IZone
pid.SetIZone(std::numeric_limits<double>::infinity());

// Integral gain will not be applied if the absolute value of the error is
// more than 2
pid.SetIZone(2);
```

PYTHON

```
# Disable IZone
pid.setIZone(math.inf)

# Integral gain will not be applied if the absolute value of the error is
# more than 2
pid.setIZone(2)
```

Le réglage de l'entrée continue

Avertissement : Si votre mécanisme n'est pas capable d'un mouvement de rotation entièrement continu (par exemple, une tourelle sans bague collectrice, dont les fils se tordent en tournant), il ne faut pas alors utiliser une entrée continue sauf si vous avez mis en place une fonction de sécurité supplémentaire pour empêcher le mécanisme de passer sa limite !

Avertissement : La fonction d'entrée continue ne limite *pas* automatiquement vos valeurs d'entrée - assurez-vous que vos valeurs d'entrée, lorsque vous utilisez cette fonction, ne sont jamais en dehors de la plage spécifiée !

Some process variables (such as the angle of a turret) are measured on a circular scale, rather than a linear one - that is, each « end » of the process variable range corresponds to the same point in reality (e.g. 360 degrees and 0 degrees). In such a configuration, there are two possible values for any given error, corresponding to which way around the circle the error is measured. It is usually best to use the smaller of these errors.

Pour configurer un `PIDController` pour le faire automatiquement, utilisez la méthode `enableContinuousInput()` :

JAVA

```
// Enables continuous input on a range from -180 to 180
pid.enableContinuousInput(-180, 180);
```

C++

```
// Enables continuous input on a range from -180 to 180
pid.EnableContinuousInput(-180, 180);
```

PYTHON

```
# Enables continuous input on a range from -180 to 180
pid.enableContinuousInput(-180, 180)
```

Le cramponnage de la sortie du contrôleur

JAVA

```
// Clamps the controller output to between -0.5 and 0.5
MathUtil.clamp(pid.calculate(encoder.getDistance(), setpoint), -0.5, 0.5);
```

C++

```
// Clamps the controller output to between -0.5 and 0.5
std::clamp(pid.Calculate(encoder.GetDistance(), setpoint), -0.5, 0.5);
```

PYTHON

```
# Python doesn't have a builtin clamp function
def clamp(v, minval, maxval):
    return max(min(v, maxval), minval)

# Clamps the controller output to between -0.5 and 0.5
clamp(pid.calculate(encoder.getDistance(), setpoint), -0.5, 0.5)
```

32.6.2 Commande prédictive ou par anticipation en WPILib

Note : This article focuses on in-code implementation of feedforward control in WPILib. For a conceptual explanation of the feedforward equations used by WPILib, see *Introduction to DC Motor Feedforward*

You may have used feedback control (such as PID) for reference tracking (making a system's output follow a desired reference signal). While this is effective, it's a reactionary measure ; the system won't start applying control effort until the system is already behind. If we could tell the controller about the desired movement and required input beforehand, the system could react quicker and the feedback controller could do less work. A controller that feeds information forward into the plant like this is called a feedforward controller.

A feedforward controller injects information about the system's dynamics (like a mathematical model does) or the intended movement. Feedforward handles parts of the control actions we already know must be applied to make a system track a reference, then feedback compensates for what we do not or cannot know about the system's behavior at runtime.

Les classes FeedForward WPILib

WPILib fournit un certain nombre de classes pour aider les utilisateurs à implémenter une commande feedforward pour leurs mécanismes. À bien des égards, un feedforward précis est plus important que la rétroaction au contrôle efficace d'un mécanisme. Comme la plupart des mécanismes en FRC® obéissent étroitement aux équations bien comprises du système, en commençant par un feedforward précis est à la fois facile et extrêmement bénéfique pour le contrôle précis et robuste du mécanisme

The WPILib feedforward classes closely match the available mechanism characterization tools available in the *SysId toolsuite*. The system identification toolsuite can be used to quickly and effectively determine the correct gains for each type of feedforward. If you are unable to empirically characterize your mechanism (due to space and/or time constraints), reasonable estimates of `kG`, `kV`, and `kA` can be obtained by fairly simple computation, and are also available from ReCalc. `kS` is nearly impossible to model, and must be measured empirically.

WPILib fournit actuellement les trois classes suivantes pour le contrôle Feedforward
- *SimpleMotorFeedforward* (Java, C++, `Python`)
- *ArmFeedforward* (Java, C++, `Python`)
- *ElevatorFeedforward* (Java, C++, `Python`)

La classe « SimpleMotorFeedforward »

Note : En C ++, la classe `SimpleMotorFeedforward` est basée sur le type d'unité utilisé pour les mesures de distance, qui peut être angulaire ou linéaire. Les gains transmis *doivent* avoir des unités cohérentes avec les unités de distance, sinon une erreur de compilation sera générée. `kS` doit avoir des unités de `volts`, `kV` doit avoir des unités de `volts * secondes / distance` et `kA` doit avoir des unités de `volts * secondes^2 / distance`. Pour plus d'informations sur les unités C ++, voir *La librairie d'unités C++*.

Note : Les composants Java Feedforward calculent les sorties en unités déterminées par les unités des gains Feedforward fournis par l'utilisateur. Les utilisateurs *doivent* maintenir la cohérence des unités, car WPILibJ ne dispose pas d'un système d'unités de type sécurisé.

Note : The API documentation for Python feedforward components indicate which unit is being used as `wpimath.units.NAME`. Users *must* take care to use correct units, as Python does not have a type-safe unit system.

La classe `SimpleMotorFeedforward` calcule les valeurs de Feedforward pour les mécanismes qui se composent de moteurs à courant continu à aimant permanent (comme le CIM, Bag, 775...) sans charge externe autre que le frottement et l'inertie, tels que les volants d'inertie et les entraînements de robots.

Pour créer un `SimpleMotorFeedforward`, il suffit de le construire avec les gains requis :

Note : Le gain `kA` peut être omis, et s'il l'est, sera par défaut à une valeur de zéro. Pour de nombreux mécanismes, en particulier ceux avec peu d'inertie, ce n'est pas nécessaire.

JAVA

```
// Create a new SimpleMotorFeedforward with gains kS, kV, and kA
SimpleMotorFeedforward feedforward = new SimpleMotorFeedforward(kS, kV, kA);
```

C++

```
// Create a new SimpleMotorFeedforward with gains kS, kV, and kA
// Distance is measured in meters
frc::SimpleMotorFeedforward<units::meters> feedforward(kS, kV, kA);
```

PYTHON

```
from wpimath.controller import SimpleMotorFeedforwardMeters

# Create a new SimpleMotorFeedforward with gains kS, kV, and kA
# Distance is measured in meters
feedforward = SimpleMotorFeedforwardMeters(kS, kV, kA)
```

Pour calculer le feedforward, il suffit d'appeler la méthode `calculate()` avec la vitesse et l'accélération désirées du moteur :

Note : L'argument d'accélération peut être omis de l'appel `calculate()`, et si c'est le cas, sa valeur par défaut sera zéro. Cela doit être fait chaque fois qu'il n'y a pas de point de consigne d'accélération clairement défini.

JAVA

```
// Calculates the feedforward for a velocity of 10 units/second and an acceleration␣
↪of 20 units/second^2
// Units are determined by the units of the gains passed in at construction.
feedforward.calculate(10, 20);
```

C++

```
// Calculates the feedforward for a velocity of 10 meters/second and an acceleration␣
↪of 20 meters/second^2
// Output is in volts
feedforward.Calculate(10_mps, 20_mps_sq);
```

PYTHON

```
# Calculates the feedforward for a velocity of 10 meters/second and an acceleration␣
↪of 20 meters/second^2
# Output is in volts
feedforward.calculate(10, 20)
```

La classe ArmFeedforward

Note : En C++, la classe `ArmFeedforward` suppose que les distances sont angulaires et non linéaires. Les gains transmis *doivent* avoir des unités cohérentes avec l'unité angulaire, sinon une erreur de compilation sera générée. `kS` et `kG` doivent avoir des unités de `volts`, `kV` doivent avoir des unités de `volts * secondes / radians`, et `kA` doivent avoir des unités de `volts * secondes^2 / radians`. Pour plus d'informations sur les unités C++, consultez *La librairie d'unités C++*.

Note : Les composants Java Feedforward calculent les sorties en unités déterminées par les unités des gains Feedforward fournis par l'utilisateur. Les utilisateurs *doivent* maintenir la cohérence des unités, car WPILibJ ne dispose pas d'un système d'unités de type sécurisé.

Note : The API documentation for Python feedforward components indicate which unit is being used as `wpimath.units.NAME`. Users *must* take care to use correct units, as Python does not have a type-safe unit system.

La classe `ArmFeedforward` calcule les valeurs de Feedforward pour les bras de levier qui sont contrôlés directement par un moteur à courant continu à aimant permanent, avec une charge externe de frottement, d'inertie et de masse du bras. Il s'agit d'un modèle précis qui simule la plupart des bras de levier utilisés en FRC.

Pour créer un `ArmFeedforward`, il suffit de le construire avec les gains requis :

Note : Le gain kA peut être omis, et s'il l'est, sera par défaut à une valeur de zéro. Pour de nombreux mécanismes, en particulier ceux avec peu d'inertie, ce n'est pas nécessaire.

JAVA

```
// Create a new ArmFeedforward with gains kS, kG, kV, and kA
ArmFeedforward feedforward = new ArmFeedforward(kS, kG, kV, kA);
```

C++

```
// Create a new ArmFeedforward with gains kS, kG, kV, and kA
frc::ArmFeedforward feedforward(kS, kG, kV, kA);
```

PYTHON

```
from wpimath.controller import ArmFeedforward

# Create a new ArmFeedforward with gains kS, kG, kV, and kA
feedforward = ArmFeedforward(kS, kG, kV, kA)
```

Pour calculer le Feedforward, il suffit d'appeler la méthode `calculate()` avec la position, la vitesse et l'accélération souhaitées du bras de levier :

Note : L'argument d'accélération peut être omis de l'appel `calculate()`, et si c'est le cas, sa valeur par défaut sera zéro. Cela doit être fait chaque fois qu'il n'y a pas de point de consigne d'accélération clairement défini.

JAVA

```
// Calculates the feedforward for a position of 1 units, a velocity of 2 units/second,
↪ and
// an acceleration of 3 units/second^2
// Units are determined by the units of the gains passed in at construction.
feedforward.calculate(1, 2, 3);
```

C++

```
// Calculates the feedforward for a position of 1 radians, a velocity of 2 radians/
↪second, and
// an acceleration of 3 radians/second^2
// Output is in volts
feedforward.Calculate(1_rad, 2_rad_per_s, 3_rad/(1_s * 1_s));
```

PYTHON

```
# Calculates the feedforward for a position of 1 radians, a velocity of 2 radians/
↪second, and
# an acceleration of 3 radians/second^2
# Output is in volts
feedforward.calculate(1, 2, 3)
```

La classe ElevatorFeedforward

Note : In C++, the passed-in gains *must* have units consistent with the distance units, or a compile-time error will be thrown. `kS` and `kG` should have units of `volts`, `kV` should have units of `volts * seconds / distance`, and `kA` should have units of `volts * seconds^2 / distance`. For more information on C++ units, see *La librairie d'unités C++*.

Note : Les composants Java Feedforward calculent les sorties en unités déterminées par les unités des gains Feedforward fournis par l'utilisateur. Les utilisateurs *doivent* maintenir la cohérence des unités, car WPILibJ ne dispose pas d'un système d'unités de type sécurisé.

Note : The API documentation for Python feedforward components indicate which unit is being used as `wpimath.units.NAME`. Users *must* take care to use correct units, as Python does not have a type-safe unit system.

La classe `ElevatorFeedforward` calcule les feed-back pour un mécanisme de type ascenseur qui se composent de moteurs à courant continu à aimant permanent chargés par le frottement, l'inertie et la masse de charge déplacée par l'ascenseur. Il s'agit d'un modèle précis de la plupart des mécanismes d'ascenseurs en FRC.

Pour créer un `ElevatorFeedforward`, il suffit de le construire avec les gains requis :

Note : Le gain `kA` peut être omis, et s'il l'est, sera par défaut à une valeur de zéro. Pour de nombreux mécanismes, en particulier ceux avec peu d'inertie, ce n'est pas nécessaire.

JAVA

```
// Create a new ElevatorFeedforward with gains kS, kG, kV, and kA
ElevatorFeedforward feedforward = new ElevatorFeedforward(kS, kG, kV, kA);
```

C++

```
// Create a new ElevatorFeedforward with gains kS, kV, and kA
// Distance is measured in meters
frc::ElevatorFeedforward feedforward(kS, kG, kV, kA);
```

PYTHON

```
from wpimath.controller import ElevatorFeedforward

# Create a new ElevatorFeedforward with gains kS, kV, and kA
# Distance is measured in meters
feedforward = ElevatorFeedforward(kS, kG, kV, kA)
```

Pour calculer le feedforward, il suffit d'appeler la méthode `calculate()` avec la vitesse et l'accélération désirées du moteur :

Note : L'argument d'accélération peut être omis de l'appel `calculate()`, et si c'est le cas, sa valeur par défaut sera zéro. Cela doit être fait chaque fois qu'il n'y a pas de point de consigne d'accélération clairement défini.

JAVA

```
// Calculates the feedforward for a velocity of 20 units/second
// and an acceleration of 30 units/second^2
// Units are determined by the units of the gains passed in at construction.
feedforward.calculate(20, 30);
```

C++

```
// Calculates the feedforward for a velocity of 20 meters/second
// and an acceleration of 30 meters/second^2
// Output is in volts
feedforward.Calculate(20_mps, 30_mps_sq);
```

PYTHON

```
# Calculates the feedforward for a velocity of 20 meters/second
# and an acceleration of 30 meters/second^2
# Output is in volts
feedforward.calculate(20, 30)
```

Utilisation de Feedforward pour contrôler les mécanismes

Note : Since feedforward voltages are physically meaningful, it is best to use the `setVoltage()` (Java, C++, Python) method when applying them to motors to compensate for « voltage sag » from the battery.

Le contrôle par anticipation (Feedforward) peut être utilisé à lui seul, sans contrôleur de rétroaction. Ceci est connu sous le nom de contrôle « en boucle ouverte », et pour de nombreux mécanismes (en particulier les moteurs qui font avancer le robot) peuvent être parfaitement satisfaisants. Un `SimpleMotorFeedforward` peut être utilisé pour contrôler un entraînement de robot comme suit :

JAVA

```
public void tankDriveWithFeedforward(double leftVelocity, double rightVelocity) {
  leftMotor.setVoltage(feedforward.calculate(leftVelocity));
  rightMotor.setVoltage(feedForward.calculate(rightVelocity));
}
```

C++

```
void TankDriveWithFeedforward(units::meters_per_second_t leftVelocity,
                              units::meters_per_second_t rightVelocity) {
  leftMotor.SetVoltage(feedforward.Calculate(leftVelocity));
  rightMotor.SetVoltage(feedforward.Calculate(rightVelocity));
}
```

PYTHON

```
def tankDriveWithFeedforward(self, leftVelocity: float, rightVelocity: float):
    self.leftMotor.setVoltage(feedForward.calculate(leftVelocity))
    self.rightMotor.setVoltage(feedForward.calculate(rightVelocity))
```

32.6.3 Combinaison des contrôles Feedforward ou Prédictif et PID

Note : N.D.T Le terme Feedforward sera quelquefois utilisé pour alléger le texte. Cet article couvre l'implémentation dans le code du contrôle Feedforward/PID combiné avec les librairies de classes fournies par WPILib. Une documentation décrivant ces concepts de façon plus détaillée est à venir.

Les contrôleurs de commande par anticipation et de rétroaction peuvent être utilisé séparément, cependant ils sont plus efficaces lorsqu'ils sont combinés ensemble. La combinaison de ces deux méthodes de contrôle est d'une simplicité enfantine - on ajoute simplement leurs sorties respectives…

L'utilisation d'un Feedforward avec un PIDController

Les utilisateurs peuvent ajouter n'importe quelle information de leur choix à la sortie du contrôleur avant de l'envoyer à leurs moteurs :

JAVA

```
// Adds a feedforward to the loop output before sending it to the motor
motor.setVoltage(pid.calculate(encoder.getDistance(), setpoint) + feedforward);
```

C++

```
// Adds a feedforward to the loop output before sending it to the motor
motor.SetVoltage(pid.Calculate(encoder.GetDistance(), setpoint) + feedforward);
```

PYTHON

```
# Adds a feedforward to the loop output before sending it to the motor
motor.setVoltage(pid.calculate(encoder.getDistance(), setpoint) + feedforward)
```

En outre, la prédiction est une fonctionnalité entièrement distincte de la rétroaction, et n'a donc aucune raison d'être traitée dans le même objet contrôleur, car cela viole la séparation des préoccupations. WPILib est livré avec plusieurs classes d'aide pour calculer des tensions de prédiction précises pour des mécanismes communs en FRC® - pour plus d'informations, consulter *Commande prédictive ou par anticipation en WPILib*.

L'utilisation des composants Feedforward avec PID

Note : Since feedforward voltages are physically meaningful, it is best to use the `setVoltage()` (Java, C++, Python) method when applying them to motors to compensate for « voltage sag » from the battery.

À quoi pourrait ressembler un exemple plus complet de contrôle Feedforward/PID combiné ? Considerez *l'exemple de conduite* de la section de contôle anticipé (Feedforward). Nous pouvons facilement modifier les exemples pour ajouter de la rétroaction (avec une composante `SimpleMotorFeedforward`) :

JAVA

```
public void tankDriveWithFeedforwardPID(double leftVelocitySetpoint, double␣
↪rightVelocitySetpoint) {
  leftMotor.setVoltage(feedforward.calculate(leftVelocitySetpoint)
      + leftPID.calculate(leftEncoder.getRate(), leftVelocitySetpoint));
  rightMotor.setVoltage(feedForward.calculate(rightVelocitySetpoint)
      + rightPID.calculate(rightEncoder.getRate(), rightVelocitySetpoint));
}
```

C++

```
void TankDriveWithFeedforwardPID(units::meters_per_second_t leftVelocitySetpoint,
                                 units::meters_per_second_t rightVelocitySetpoint) {
  leftMotor.SetVoltage(feedforward.Calculate(leftVelocitySetpoint)
      + leftPID.Calculate(leftEncoder.getRate(), leftVelocitySetpoint.value()));
  rightMotor.SetVoltage(feedforward.Calculate(rightVelocitySetpoint)
      + rightPID.Calculate(rightEncoder.getRate(), rightVelocitySetpoint.value()));
}
```

PYTHON

```
def tank_drive_with_feedforward_PID(
    left_velocity_setpoint: float,
    right_velocity_setpoint: float,
) -> None:
    leftMotor.setVoltage(
        feedforward.calculate(left_velocity_setpoint)
        + leftPID.calculate(leftEncoder.getRate(), left_velocity_setpoint)
    )
    rightMotor.setVoltage(
        feedforward.calculate(right_velocity_setpoint)
        + rightPID.calculate(rightEncoder.getRate(), right_velocity_setpoint)
    )
```

D'autre types de mécanismes peut être traité de la même façon.

32.6.4 Profils de mouvement trapézoïdal dans WPILib

Note : Cet article couvre la génération de profils de mouvement trapézoïdaux. Une documentation plus détaillée décrivant les concepts impliqués est à venir.

Note : Pour un guide sur la mise en œuvre de la classe `TrapezoidProfile` dans le cadre du *command-based framework* voir *Profilage de mouvement via TrapezoidProfileSubsystems et TrapezoidProfileCommands*.

Note : Lorsque la classe `TrapezoidProfile` est utilisée seule, elle est mieux servie quand elle est pairée avec un contrôleur de type « intelligent » (tel qu'un contrôleur de moteur avec une fonctionnalité PID intégrée). Pour l'intégrer à la classe `PIDController` de WPILib, voir *Combinaison du profilage de mouvement et du contrôle PID avec ProfiledPIDController*.

Bien que le contrôle par anticipation (Feedforward) et la rétroaction offrent des moyens pratiques d'atteindre un point de consigne donné, nous sommes souvent confrontés au problème de la génération de points de consigne pour nos mécanismes. Bien que l'approche naïve de commander immédiatement un mécanisme à son état souhaité puisse fonctionner, elle est souvent sous-optimale. Pour améliorer la gestion de nos mécanismes, nous voulons commander ces derniers en suivant une *séquence* de points de consigne qui interpolent en douceur la transition entre son état actuel et son état cible souhaité.

To help users do this, WPILib provides a `TrapezoidProfile` class (Java, C++, `Python`).

La création d'un profil trapézoïdal

Note : En C ++, la classe `TrapezoidProfile` est basée sur le type d'unité utilisé pour les mesures de distance, qui peut être angulaire ou linéaire. Les valeurs transmises *doivent* avoir des unités cohérentes avec les unités de distance, sinon une erreur de compilation sera levée. Pour plus d'informations sur les unités C ++, voir *La librairie d'unités C++*.

Les contraintes

Note : Les différentes *classes de Feedforward* fournissent des méthodes pour calculer la vitesse et l'accélération maximales d'un mécanisme, de façon simultanée. Celles-ci peuvent être très utiles pour calculer les contraintes de mouvement appropriées générer un `TrapezoidProfile`

In order to create a trapezoidal motion profile, we must first impose some constraints on the desired motion. Namely, we must specify a maximum velocity and acceleration that the mechanism will be expected to achieve during the motion. To do this, we create an instance of the `TrapezoidProfile.Constraints` class (Java, C++, `Python`) :

JAVA

```
// Creates a new set of trapezoidal motion profile constraints
// Max velocity of 10 meters per second
// Max acceleration of 20 meters per second squared
new TrapezoidProfile.Constraints(10, 20);
```

C++

```
// Creates a new set of trapezoidal motion profile constraints
// Max velocity of 10 meters per second
// Max acceleration of 20 meters per second squared
frc::TrapezoidProfile<units::meters>::Constraints{10_mps, 20_mps_sq};
```

PYTHON

```
from wpimath.trajectory import TrapezoidProfile

# Creates a new set of trapezoidal motion profile constraints
# Max velocity of 10 meters per second
# Max acceleration of 20 meters per second squared
TrapezoidProfile.Constraints(10, 20)
```

Les états de départ et de fin

Next, we must specify the desired starting and ending states for our mechanisms using the `TrapezoidProfile.State` class (Java, C++, Python). Each state has a position and a velocity :

JAVA

```
// Creates a new state with a position of 5 meters
// and a velocity of 0 meters per second
new TrapezoidProfile.State(5, 0);
```

C++

```
// Creates a new state with a position of 5 meters
// and a velocity of 0 meters per second
frc::TrapezoidProfile<units::meters>::State{5_m, 0_mps};
```

PYTHON

```
from wpimath.trajectory import TrapezoidProfile

# Creates a new state with a position of 5 meters
# and a velocity of 0 meters per second
TrapezoidProfile.State(5, 0)
```

Combiner tout cela ensemble

Note : C ++ est souvent capable d'inférer le type des classes internes, et donc une simple liste d'initialisation (sans le nom de la classe) peut être envoyée en tant que paramètre. Les noms de classe complets sont inclus dans l'exemple ci-dessous pour plus de clarté.

Now that we know how to create a set of constraints and the desired start/end states, we are ready to create our motion profile. The `TrapezoidProfile` constructor takes 1 parameter : the constraints.

JAVA

```
// Creates a new TrapezoidProfile
// Profile will have a max vel of 5 meters per second
// Profile will have a max acceleration of 10 meters per second squared
TrapezoidProfile profile = new TrapezoidProfile(new TrapezoidProfile.Constraints(5,␣
↪10));
```

C++

```
// Creates a new TrapezoidProfile
// Profile will have a max vel of 5 meters per second
// Profile will have a max acceleration of 10 meters per second squared
frc::TrapezoidProfile<units::meters> profile{
  frc::TrapezoidProfile<units::meters>::Constraints{5_mps, 10_mps_sq}};
```

PYTHON

```
from wpimath.trajectory import TrapezoidProfile

# Creates a new TrapezoidProfile
# Profile will have a max vel of 5 meters per second
# Profile will have a max acceleration of 10 meters per second squared
profile = TrapezoidProfile(TrapezoidProfile.Constraints(5, 10))
```

L'utilisation de `TrapezoidProfile`

Échantillonnage du profil

Once we've created a `TrapezoidProfile`, using it is very simple : to get the profile state at the given time after the profile has started, call the `calculate()` method with the goal state and initial state :

JAVA

```
// Profile will start stationary at zero position
// Profile will end stationary at 5 meters
// Returns the motion profile state after 5 seconds of motion
profile.calculate(5, new TrapezoidProfile.State(0, 0), new TrapezoidProfile.State(5,␣
↪0));
```

C++

```
// Profile will start stationary at zero position
// Profile will end stationary at 5 meters
// Returns the motion profile state after 5 seconds of motion
profile.Calculate(5_s,
frc::TrapezoidProfile<units::meters>::State{0_m, 0_mps},
frc::TrapezoidProfile<units::meters>::State{5_m, 0_mps});
```

PYTHON

```
# Profile will start stationary at zero position
# Profile will end stationary at 5 meters
# Returns the motion profile state after 5 seconds of motion
profile.calculate(5, TrapezoidProfile.State(0, 0), TrapezoidProfile.State(5, 0))
```

Utiliser l'État

The `calculate` method returns a `TrapezoidProfile.State` class (the same one that was used to specify the initial/end states when calculating the profile state). To use this for actual control, simply pass the contained position and velocity values to whatever controller you wish (for example, a PIDController) :

JAVA

```
var setpoint = profile.calculate(elapsedTime, initialState, goalState);
controller.calculate(encoder.getDistance(), setpoint.position);
```

C++

```
auto setpoint = profile.Calculate(elapsedTime, initialState, goalState);
controller.Calculate(encoder.GetDistance(), setpoint.position.value());
```

PYTHON

```
setpoint = profile.calculate(elapsedTime, initialState, goalState)
controller.calculate(encoder.getDistance(), setpoint.position)
```

Un exemple d'utilisation complet

Note : In this example, the initial state is re-computed every timestep. This is a somewhat different usage technique than is detailed above, but works according to the same principles - the profile is sampled at a time corresponding to the loop period to get the setpoint for the next loop iteration.

A more complete example of `TrapezoidProfile` usage is provided in the ElevatorTrapezoidProfile example project (Java, C++, Python) :

JAVA

```
5  package edu.wpi.first.wpilibj.examples.elevatortrapezoidprofile;
6
7  import edu.wpi.first.math.controller.SimpleMotorFeedforward;
8  import edu.wpi.first.math.trajectory.TrapezoidProfile;
9  import edu.wpi.first.wpilibj.Joystick;
10 import edu.wpi.first.wpilibj.TimedRobot;
11
12 public class Robot extends TimedRobot {
13   private static double kDt = 0.02;
14
15   private final Joystick m_joystick = new Joystick(1);
16   private final ExampleSmartMotorController m_motor = new␣
   ↪ExampleSmartMotorController(1);
17   // Note: These gains are fake, and will have to be tuned for your robot.
18   private final SimpleMotorFeedforward m_feedforward = new SimpleMotorFeedforward(1,␣
   ↪1.5);
19
20   // Create a motion profile with the given maximum velocity and maximum
21   // acceleration constraints for the next setpoint.
22   private final TrapezoidProfile m_profile =
```

(suite sur la page suivante)

(suite de la page précédente)

```
      new TrapezoidProfile(new TrapezoidProfile.Constraints(1.75, 0.75));
  private TrapezoidProfile.State m_goal = new TrapezoidProfile.State();
  private TrapezoidProfile.State m_setpoint = new TrapezoidProfile.State();

  @Override
  public void robotInit() {
    // Note: These gains are fake, and will have to be tuned for your robot.
    m_motor.setPID(1.3, 0.0, 0.7);
  }

  @Override
  public void teleopPeriodic() {
    if (m_joystick.getRawButtonPressed(2)) {
      m_goal = new TrapezoidProfile.State(5, 0);
    } else if (m_joystick.getRawButtonPressed(3)) {
      m_goal = new TrapezoidProfile.State();
    }

    // Retrieve the profiled setpoint for the next timestep. This setpoint moves
    // toward the goal while obeying the constraints.
    m_setpoint = m_profile.calculate(kDt, m_setpoint, m_goal);

    // Send setpoint to offboard controller PID
    m_motor.setSetpoint(
        ExampleSmartMotorController.PIDMode.kPosition,
        m_setpoint.position,
        m_feedforward.calculate(m_setpoint.velocity) / 12.0);
  }
}
```

C++

```
#include <numbers>

#include <frc/Joystick.h>
#include <frc/TimedRobot.h>
#include <frc/controller/SimpleMotorFeedforward.h>
#include <frc/trajectory/TrapezoidProfile.h>
#include <units/acceleration.h>
#include <units/length.h>
#include <units/time.h>
#include <units/velocity.h>
#include <units/voltage.h>

#include "ExampleSmartMotorController.h"

class Robot : public frc::TimedRobot {
 public:
  static constexpr units::second_t kDt = 20_ms;

  Robot() {
    // Note: These gains are fake, and will have to be tuned for your robot.
    m_motor.SetPID(1.3, 0.0, 0.7);
  }
```

(suite sur la page suivante)

(suite de la page précédente)

```

  void TeleopPeriodic() override {
    if (m_joystick.GetRawButtonPressed(2)) {
      m_goal = {5_m, 0_mps};
    } else if (m_joystick.GetRawButtonPressed(3)) {
      m_goal = {0_m, 0_mps};
    }

    // Retrieve the profiled setpoint for the next timestep. This setpoint moves
    // toward the goal while obeying the constraints.
    m_setpoint = m_profile.Calculate(kDt, m_setpoint, m_goal);

    // Send setpoint to offboard controller PID
    m_motor.SetSetpoint(ExampleSmartMotorController::PIDMode::kPosition,
                        m_setpoint.position.value(),
                        m_feedforward.Calculate(m_setpoint.velocity) / 12_V);
  }

 private:
  frc::Joystick m_joystick{1};
  ExampleSmartMotorController m_motor{1};
  frc::SimpleMotorFeedforward<units::meters> m_feedforward{
      // Note: These gains are fake, and will have to be tuned for your robot.
      1_V, 1.5_V * 1_s / 1_m};

  // Create a motion profile with the given maximum velocity and maximum
  // acceleration constraints for the next setpoint.
  frc::TrapezoidProfile<units::meters> m_profile{{1.75_mps, 0.75_mps_sq}};
  frc::TrapezoidProfile<units::meters>::State m_goal;
  frc::TrapezoidProfile<units::meters>::State m_setpoint;
};

#ifndef RUNNING_FRC_TESTS
int main() {
  return frc::StartRobot<Robot>();
}
#endif
```

PYTHON

```
import wpilib
import wpimath.controller
import wpimath.trajectory
import examplesmartmotorcontroller


class MyRobot(wpilib.TimedRobot):
    kDt = 0.02

    def robotInit(self):
        self.joystick = wpilib.Joystick(1)
        self.motor = examplesmartmotorcontroller.ExampleSmartMotorController(1)
        # Note: These gains are fake, and will have to be tuned for your robot.
        self.feedforward = wpimath.controller.SimpleMotorFeedforwardMeters(1, 1.5)
```

(suite sur la page suivante)

(suite de la page précédente)

```

        self.constraints = wpimath.trajectory.TrapezoidProfile.Constraints(1.75, 0.75)

        self.goal = wpimath.trajectory.TrapezoidProfile.State()
        self.setpoint = wpimath.trajectory.TrapezoidProfile.State()

        # Note: These gains are fake, and will have to be tuned for your robot.
        self.motor.setPID(1.3, 0.0, 0.7)

    def teleopPeriodic(self):
        if self.joystick.getRawButtonPressed(2):
            self.goal = wpimath.trajectory.TrapezoidProfile.State(5, 0)
        elif self.joystick.getRawButtonPressed(3):
            self.goal = wpimath.trajectory.TrapezoidProfile.State(0, 0)

        # Create a motion profile with the given maximum velocity and maximum
        # acceleration constraints for the next setpoint, the desired goal, and the
        # current setpoint.
        profile = wpimath.trajectory.TrapezoidProfile(
            self.constraints, self.goal, self.setpoint
        )

        # Retrieve the profiled setpoint for the next timestep. This setpoint moves
        # toward the goal while obeying the constraints.
        self.setpoint = profile.calculate(self.kDt)

        # Send setpoint to offboard controller PID
        self.motor.setSetPoint(
            examplesmartmotorcontroller.ExampleSmartMotorController.PIDMode.kPosition,
            self.setpoint.position,
            self.feedforward.calculate(self.setpoint.velocity) / 12,
        )
```

32.6.5 Combinaison du profilage de mouvement et du contrôle PID avec ProfiledPIDController

Note : Pour un guide sur l'implémentation de la classe `ProfiledPIDController` dans le cadre du *command-based framework* framework, see *Combinaison du profilage de mouvement et du PID dans les commandes*.

Dans l'article précédent, nous avons vu comment utiliser la classe `TrapezoidProfile` pour créer et utiliser un profil de mouvement trapézoïdal. L'exemple de code de cet article montre comment créer manuellement la classe `TrapezoidProfile` avec la fonction de contrôle PID externe d'un contrôleur de moteur « intelligent ».

This combination of functionality (a motion profile for generating setpoints combined with a PID controller for following them) is extremely common. To facilitate this, WPILib comes with a `ProfiledPIDController` class (Java, C++, Python) that does most of the work of combining these two functionalities. The API of the `ProfiledPIDController` is very similar to that of the `PIDController`, allowing users to add motion profiling to a PID-controlled mechanism with very few changes to their code.

L'utilisation de la classe ProfiledPIDController

Note : En C ++, la classe `ProfiledPIDController` est basée sur le type d'unité utilisé pour les mesures de distance, qui peut être angulaire ou linéaire. Les valeurs transmises *doivent* avoir des unités cohérentes avec les unités de distance, sinon une erreur de compilation sera levée. Pour plus d'informations sur les unités C++, voir *La librairie d'unités C++*.

Note : La plupart des fonctionnalités de `ProfiledPIDController` sont relativement identiques à celle de `PIDController`. Par conséquent, cet article ne couvrira que les fonctionnalités qui ont été substantiellement modifiées pour s'adapter à la fonctionnalité de profilage de mouvement. Pour plus d'informations sur les fonctionnalités standard de `PIDController`, voir *Contrôle PID dans WPILib*.

La construction d'un objet ProfiledPIDController

Note : C ++ est souvent capable d'inférer le type des classes internes, et donc une simple liste d'initialisation (sans le nom de la classe) peut être envoyée en tant que paramètre. Le nom complet de la classe est inclus dans l'exemple ci-dessous pour plus de clarté.

La création d'un `ProfiledPIDController` est presque identique à *creating a PIDController*. La seule différence est la nécessité de fournir un ensemble de *trapezoidal profile constraints*, qui seront automatiquement transmises aux instances du « TrapezoidProfile » :

JAVA

```
// Creates a ProfiledPIDController
// Max velocity is 5 meters per second
// Max acceleration is 10 meters per second
ProfiledPIDController controller = new ProfiledPIDController(
  kP, kI, kD,
  new TrapezoidProfile.Constraints(5, 10));
```

C++

```
// Creates a ProfiledPIDController
// Max velocity is 5 meters per second
// Max acceleration is 10 meters per second
frc::ProfiledPIDController<units::meters> controller(
  kP, kI, kD,
  frc::TrapezoidProfile<units::meters>::Constraints{5_mps, 10_mps_sq});
```

PYTHON

```
from wpimath.controller import ProfiledPIDController
from wpimath.trajectory import TrapezoidProfile

# Creates a ProfiledPIDController
# Max velocity is 5 meters per second
# Max acceleration is 10 meters per second
controller = ProfiledPIDController(
  kP, kI, kD,
  TrapezoidProfile.Constraints(5, 10))
```

Objectif vs point de consigne

Une différence majeure entre un `PIDController` normal et un `ProfiledPIDController` est que le *point de consigne* réel de la boucle de régulation n'est pas directement spécifié par l'utilisateur. Au contraire, l'utilisateur spécifie une position ou un état souhaité, ou *objectif*, et le point de consigne pour le contrôleur est calculé automatiquement à partir du profil de mouvement généré entre l'état actuel et l'état objectif. Alors que la commande pour appeler cette classe semble essentiellement identique :

JAVA

```
// Calculates the output of the PID algorithm based on the sensor reading
// and sends it to a motor
motor.set(controller.calculate(encoder.getDistance(), goal));
```

C++

```
// Calculates the output of the PID algorithm based on the sensor reading
// and sends it to a motor
motor.Set(controller.Calculate(encoder.GetDistance(), goal));
```

PYTHON

```
# Calculates the output of the PID algorithm based on the sensor reading
# and sends it to a motor
motor.set(controller.calculate(encoder.getDistance(), goal))
```

La valeur `objectif` spécifiée (qui peut être une valeur de position ou un `TrapezoidProfile.State`, si une vitesse non nulle est souhaitée) n'est *pas* nécessairement le point de consigne *actuel* de la boucle - plutôt, elle est le point de consigne *éventuel* une fois le profil généré terminé.

L'obtention / l'utilisation du point de consigne

Étant donné que l'objectif `ProfiledPIDController` diffère du point de consigne, il est souvent souhaitable d'interroger le contrôleur pour obtenir le point de consigne actuel (par exemple, pour obtenir des valeurs à utiliser avec *feedforward*). Cela peut être fait avec la méthode `getSetpoint()`.

Le point de consigne renvoyé peut alors être utilisé comme dans l'exemple suivant.

JAVA

```
double lastSpeed = 0;
double lastTime = Timer.getFPGATimestamp();

// Controls a simple motor's position using a SimpleMotorFeedforward
// and a ProfiledPIDController
public void goToPosition(double goalPosition) {
  double pidVal = controller.calculate(encoder.getDistance(), goalPosition);
  double acceleration = (controller.getSetpoint().velocity - lastSpeed) / (Timer.
↪getFPGATimestamp() - lastTime);
  motor.setVoltage(
      pidVal
      + feedforward.calculate(controller.getSetpoint().velocity, acceleration));
  lastSpeed = controller.getSetpoint().velocity;
  lastTime = Timer.getFPGATimestamp();
}
```

C++

```
units::meters_per_second_t lastSpeed = 0_mps;
units::second_t lastTime = frc2::Timer::GetFPGATimestamp();

// Controls a simple motor's position using a SimpleMotorFeedforward
// and a ProfiledPIDController
void GoToPosition(units::meter_t goalPosition) {
  auto pidVal = controller.Calculate(units::meter_t{encoder.GetDistance()},␣
↪goalPosition);
  auto acceleration = (controller.GetSetpoint().velocity - lastSpeed) /
      (frc2::Timer::GetFPGATimestamp() - lastTime);
  motor.SetVoltage(
       pidVal +
      feedforward.Calculate(controller.GetSetpoint().velocity, acceleration));
  lastSpeed = controller.GetSetpoint().velocity;
  lastTime = frc2::Timer::GetFPGATimestamp();
}
```

PYTHON

```
from wpilib import Timer
from wpilib.controller import ProfiledPIDController
from wpilib.controller import SimpleMotorFeedforward


def __init__(self):

    # Assuming encoder, motor, controller are already defined
    self.lastSpeed = 0
    self.lastTime = Timer.getFPGATimestamp()

    # Assuming feedforward is a SimpleMotorFeedforward object
    self.feedforward = SimpleMotorFeedforward(ks=0.0, kv=0.0, ka=0.0)

def goToPosition(self, goalPosition: float):

    pidVal = self.controller.calculate(self.encoder.getDistance(), goalPosition)
    acceleration = (self.controller.getSetpoint().velocity - self.lastSpeed) / (Timer.
↪getFPGATimestamp() - self.lastTime)

    self.motor.setVoltage(
        pidVal
        + self.feedforward.calculate(self.controller.getSetpoint().velocity,␣
↪acceleration))

    self.lastSpeed = controller.getSetpoint().velocity
    self.lastTime = Timer.getFPGATimestamp()
```

Un exemple complet d'utilisation

A more complete example of `ProfiledPIDController` usage is provided in the ElevatorProfilePID example project (Java, C++, Python) :

JAVA

```
5  package edu.wpi.first.wpilibj.examples.elevatorprofiledpid;
6
7  import edu.wpi.first.math.controller.ElevatorFeedforward;
8  import edu.wpi.first.math.controller.ProfiledPIDController;
9  import edu.wpi.first.math.trajectory.TrapezoidProfile;
10 import edu.wpi.first.wpilibj.Encoder;
11 import edu.wpi.first.wpilibj.Joystick;
12 import edu.wpi.first.wpilibj.TimedRobot;
13 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
14
15 @SuppressWarnings("PMD.RedundantFieldInitializer")
16 public class Robot extends TimedRobot {
17   private static double kDt = 0.02;
18   private static double kMaxVelocity = 1.75;
19   private static double kMaxAcceleration = 0.75;
20   private static double kP = 1.3;
```

(suite sur la page suivante)

(suite de la page précédente)

```
  private static double kI = 0.0;
  private static double kD = 0.7;
  private static double kS = 1.1;
  private static double kG = 1.2;
  private static double kV = 1.3;

  private final Joystick m_joystick = new Joystick(1);
  private final Encoder m_encoder = new Encoder(1, 2);
  private final PWMSparkMax m_motor = new PWMSparkMax(1);

  // Create a PID controller whose setpoint's change is subject to maximum
  // velocity and acceleration constraints.
  private final TrapezoidProfile.Constraints m_constraints =
      new TrapezoidProfile.Constraints(kMaxVelocity, kMaxAcceleration);
  private final ProfiledPIDController m_controller =
      new ProfiledPIDController(kP, kI, kD, m_constraints, kDt);
  private final ElevatorFeedforward m_feedforward = new ElevatorFeedforward(kS, kG,␣
↪kV);

  @Override
  public void robotInit() {
    m_encoder.setDistancePerPulse(1.0 / 360.0 * 2.0 * Math.PI * 1.5);
  }

  @Override
  public void teleopPeriodic() {
    if (m_joystick.getRawButtonPressed(2)) {
      m_controller.setGoal(5);
    } else if (m_joystick.getRawButtonPressed(3)) {
      m_controller.setGoal(0);
    }

    // Run controller and update motor output
    m_motor.setVoltage(
        m_controller.calculate(m_encoder.getDistance())
            + m_feedforward.calculate(m_controller.getSetpoint().velocity));
  }
}
```

C++

```
#include <numbers>

#include <frc/Encoder.h>
#include <frc/Joystick.h>
#include <frc/TimedRobot.h>
#include <frc/controller/ElevatorFeedforward.h>
#include <frc/controller/ProfiledPIDController.h>
#include <frc/motorcontrol/PWMSparkMax.h>
#include <frc/trajectory/TrapezoidProfile.h>
#include <units/acceleration.h>
#include <units/length.h>
#include <units/time.h>
#include <units/velocity.h>
```

(suite sur la page suivante)

(suite de la page précédente)

```cpp
#include <units/voltage.h>

class Robot : public frc::TimedRobot {
 public:
  static constexpr units::second_t kDt = 20_ms;

  Robot() {
    m_encoder.SetDistancePerPulse(1.0 / 360.0 * 2.0 * std::numbers::pi * 1.5);
  }

  void TeleopPeriodic() override {
    if (m_joystick.GetRawButtonPressed(2)) {
      m_controller.SetGoal(5_m);
    } else if (m_joystick.GetRawButtonPressed(3)) {
      m_controller.SetGoal(0_m);
    }

    // Run controller and update motor output
    m_motor.SetVoltage(
        units::volt_t{
            m_controller.Calculate(units::meter_t{m_encoder.GetDistance()})} +
        m_feedforward.Calculate(m_controller.GetSetpoint().velocity));
  }

 private:
  static constexpr units::meters_per_second_t kMaxVelocity = 1.75_mps;
  static constexpr units::meters_per_second_squared_t kMaxAcceleration =
      0.75_mps_sq;
  static constexpr double kP = 1.3;
  static constexpr double kI = 0.0;
  static constexpr double kD = 0.7;
  static constexpr units::volt_t kS = 1.1_V;
  static constexpr units::volt_t kG = 1.2_V;
  static constexpr auto kV = 1.3_V / 1_mps;

  frc::Joystick m_joystick{1};
  frc::Encoder m_encoder{1, 2};
  frc::PWMSparkMax m_motor{1};

  // Create a PID controller whose setpoint's change is subject to maximum
  // velocity and acceleration constraints.
  frc::TrapezoidProfile<units::meters>::Constraints m_constraints{
      kMaxVelocity, kMaxAcceleration};
  frc::ProfiledPIDController<units::meters> m_controller{kP, kI, kD,
                                                         m_constraints, kDt};
  frc::ElevatorFeedforward m_feedforward{kS, kG, kV};
};

#ifndef RUNNING_FRC_TESTS
int main() {
  return frc::StartRobot<Robot>();
}
#endif
```

PYTHON

```python
import wpilib
import wpimath.controller
import wpimath.trajectory
import math


class MyRobot(wpilib.TimedRobot):
    kDt = 0.02

    def robotInit(self) -> None:
        self.joystick = wpilib.Joystick(1)
        self.encoder = wpilib.Encoder(1, 2)
        self.motor = wpilib.PWMSparkMax(1)

        # Create a PID controller whose setpoint's change is subject to maximum
        # velocity and acceleration constraints.
        self.constraints = wpimath.trajectory.TrapezoidProfile.Constraints(1.75, 0.75)
        self.controller = wpimath.controller.ProfiledPIDController(
            1.3, 0, 0.7, self.constraints, self.kDt
        )

        self.encoder.setDistancePerPulse(1 / 360 * 2 * math.pi * 1.5)

    def teleopPeriodic(self) -> None:
        if self.joystick.getRawButtonPressed(2):
            self.controller.setGoal(5)
        elif self.joystick.getRawButtonPressed(3):
            self.controller.setGoal(0)

        # Run controller and update motor output
        self.motor.set(self.controller.calculate(self.encoder.getDistance()))
```

32.6.6 Commande Tout ou Rien en abrégé TOR avec contrôleur Bang-Bang

La commande « Bang-bang » ou « Tout ou rien » est une stratégie de contrôle qui n'utilise que deux états : activé (lorsque la mesure est inférieure au point de consigne) et désactivé (dans le cas contraire). Ceci est à peu près équivalent à une boucle proportionnelle avec un gain infini.

Cette approche peut sembler au départ être une mauvaise stratégie de contrôle, car les boucles PID ont la réputation de devenir instables à mesure que les gains deviennent importants - et en effet, c'est une *très mauvaise idée d'utiliser un contrôleur bang-bang sur tout système autre que le contrôle de la vitesse d'un mécanisme à haute inertie*.

Cependant, lors du contrôle de la vitesse des mécanismes à haute inertie sous des charges variables (comme le volant d'inertie d'un tireur), un contrôleur bang-bang peut offrir un temps de récupération plus rapide et donc de meilleures performances/plus cohérentes qu'un contrôleur proportionnel. Contrairement à une boucle P ordinaire, un contrôleur bang-bang est *asymétrique* - c'est-à-dire que le contrôleur est activé lorsque la variable du procédé est en dessous du point de consigne et ne fait rien d'autre. Cela permet de rendre l'effort du contrôle anticipé aussi important que possible sans risque de produire des oscillations destructrices lorsque la boucle de contrôle tenterait de corriger un dépassement résultant.

Asymmetric bang-bang control is provided in WPILib by the BangBangController class (Java, C++, Python).

Construction d'un objet BangBangController

Étant donné qu'un contrôleur bang-bang n'a aucun gain, il n'a pas besoin de constructeur avec argument (on peut éventuellement spécifier la tolérance du contrôleur utilisé par le paramètre `atSetpoint`, mais ce n'est pas obligatoire).

JAVA

```
// Creates a BangBangController
BangBangController controller = new BangBangController();
```

C++

```
// Creates a BangBangController
frc::BangBangController controller;
```

PYTHON

```
from wpimath.controller import BangBangController

# Creates a BangBangController
controller = BangBangController()
```

Utilisation d'un objet BangBangController

Avertissement : La commande Bang-bang utilise un algorithme extrêmement agressif qui repose sur l'asymétrie de réponse pour assurer la stabilité du système. *Assurez-vous* que vos contrôleurs de moteurs ont été configurés en « coast mode » avant d'essayer de les commander avec un contrôleur bang-bang, sinon le freinage résultant du mode « break » combattra le contrôleur et provoquera une oscillation potentiellement destructrice pour le système.

L'utilisation d'un contrôleur bang-bang est facile :

JAVA

```
// Controls a motor with the output of the BangBang controller
motor.set(controller.calculate(encoder.getRate(), setpoint));
```

C++

```
// Controls a motor with the output of the BangBang controller
motor.Set(controller.Calculate(encoder.GetRate(), setpoint));
```

PYTHON

```
# Controls a motor with the output of the BangBang controller
motor.set(controller.calculate(encoder.getRate(), setpoint))
```

Combinaison de la commande Bang Bang et de la commande par anticipation

Comme dans le cas d'un commande PID, les meilleurs résultats sont obtenus en conjonction avec une commande *prédictive* qui fournit la tension nécessaire pour maintenir la sortie du système à la vitesse souhaitée, de sorte que le contrôleur bang-bang n'est responsable que de l'élimination des perturbations. Étant donné que le contrôleur bang-bang ne peut *que* corriger dans la commande prédictive, il serait sûrement préférable d'utiliser une estimation légèrement conservatrice de la commande prédictive pour s'assurer que le tireur ne dépasse pas la vitesse souhaitée.

JAVA

```
// Controls a motor with the output of the BangBang controller and a feedforward
// Shrinks the feedforward slightly to avoid overspeeding the shooter
motor.setVoltage(controller.calculate(encoder.getRate(), setpoint) * 12.0 + 0.9 *␣
↪feedforward.calculate(setpoint));
```

C++

```
// Controls a motor with the output of the BangBang controller and a feedforward
// Shrinks the feedforward slightly to avoid overspeeding the shooter
motor.SetVoltage(controller.Calculate(encoder.GetRate(), setpoint) * 12.0 + 0.9 *␣
↪feedforward.Calculate(setpoint));
```

PYTHON

```
# Controls a motor with the output of the BangBang controller and a feedforward
motor.setVoltage(controller.calculate(encoder.getRate(), setpoint) * 12.0 + 0.9 *␣
↪feedforward.calculate(setpoint))
```

32.7 Génération de Trajectoire et Repérage avec WPILib

Cette section décrit la prise en charge par la WPILib de la génération des trajectoires spline paramétrisées et le suivi ces trajectoires avec les entraînements courants des robots en FRC®.

32.7.1 La génération de trajectoire

WPILib contient des classes qui servent à générer des trajectoires. Une trajectoire est une courbe lisse, comportant des vitesses et des accélérations en chacun de ses points et qui relie deux extrémités sur le terrain. La génération et le suivi des trajectoires sont des opérations extrêment imporantes pour effectuer des tâches en mode autonome. Au lieu d'une simple routine autonome - qui consiste à avancer, à s'arrêter, à tourner de 90 degrés vers la droite, puis à avancer - l'utilisation de trajectoires permet de se déplacer le long d'une courbe lisse. Ce qui a pour avantage d'accélérer les routines autonomes et, donc, laisse plus de temps pour effectuer d'autres tâches ; et lorsque l'utilisation de trajectoire est bien mise en œuvre, les déplacements en mode autonome plus beaucoup plus précis.

Cet article explique comment générer une trajectoire. Les prochains articles de cette série examineront comment suivre réellement la trajectoire générée. Afin de pouvoir suivre une trajectoire donnée, votre robot doit être capable de faire tout d'abord les mesures suivantes, en utilisant certains senseurs :

- La mesure de la position et la vitesse de chaque côté du robot. Un encodeur est le meilleur moyen de le faire ; cependant, d'autres options peuvent inclure des capteurs optiques de débit (optical flow sensors), etc.
- Un moyen de mesurer l'angle ou la vitesse angulaire du châssis du robot. Un gyroscope est la meilleure façon de procéder. Bien que la vitesse angulaire puisse être calculée en utilisant les vitesses obtenues par les encodeurs, cette méthode n'est PAS recommandée en raison du glissement éventuel des roues lors d'un virage (Scrubbing).

Si vous cherchez un moyen plus simple d'effectuer une navigation autonome, voir *la section relative au déplacement sur une distance donnée*.

Courbes d'interpolation (Splines)

(N.D.T. Le mot anglais Spline sera utilisé dans cette section pour alléger le texte). Une Spline fait référence à un ensemble de courbes qui interpolent entre les points. Considérez-les comme des points de connexion, sauf avec des courbes. WPILib prend en charge deux types de Splines : « hermite cubic clamped » et « hermite quintic ».

- « Hermite cubic clamped » : C'est l'option recommandée pour la plupart des utilisateurs. La génération de trajectoires à l'aide de ces Splines implique de spécifier les coordonnées (x, y) de tous les points et les caps (direction angulaire) pour les points de

départ et de fin. Les changements de cap à l'intérieur de la Spline sont automatiquement calculés pour assurer une courbure continue (taux de changement du cap) sur tout le long du parcours.
- Hermite quintic : Il s'agit d'une option plus avancée qui nécessite que l'utilisateur spécifie les coordonnées (x, y) et les caps (direction angulaire) pour *tous* les points internes au parcours. Cela doit être utilisé si vous n'êtes pas satisfait des trajectoires générées par les Splines cubiques décrites ci-dessus, ou si vous souhaitez un contrôle plus fin des caps sur points intérieurs au parcours.

Les splines sont utilisées comme un outil pour générer des trajectoires ; cependant, la spline elle-même n'a aucune information sur les vitesses et les accélérations. Par conséquent, il n'est pas recommandé d'utiliser directement les classes de splines. Afin de générer un chemin lisse avec des vitesses et des accélérations, une *trajectoire* doit être générée.

La création de la configuration de trajectoire

Une configuration doit être créée afin de générer une trajectoire. La configuration contient des informations sur les contraintes spéciales, dont la vitesse maximale, l'accélération maximale, en plus de la vitesse de départ et de la vitesse de fin. La configuration contient également des informations sur l'opportunité d'inverser la trajectoire (le robot recule sur certains points internes). La classe `TrajectoryConfig` doit être utilisée pour construire une configuration. Le constructeur de cette classe prend deux arguments, la vitesse maximale et l'accélération maximale. Les autres champs (`startVelocity`, `endVelocity`, `inversé`, `contraintes`) ont par défaut des valeurs raisonnables (`0`, `0`, `false`, `{}`) lorsque l'objet est créé. Si vous souhaitez modifier les valeurs de l'un de ces champs, vous pouvez appeler les méthodes suivantes :
- `setStartVelocity(double startVelocityMetersPerSecond)` (Java/Python) / `SetStartVelocity(units::meters_per_second_t startVelocity)` (C++)
- `setEndVelocity(double endVelocityMetersPerSecond)` (Java/Python) / `SetEndVelocity(units::meters_per_second_t endVelocity)` (C++)
- `setReversed(boolean reversed)` (Java/Python) / `SetReversed(bool reversed)` (C++)
- `addConstraint(TrajectoryConstraint constraint)` (Java/Python) / `AddConstraint(TrajectoryConstraint constraint)` (C++)

Note : La propriété `reversed` représente simplement si le robot recule durant son parcours. Si vous spécifiez quatre points de cheminement, a, b, c et d, le robot continuera de voyager dans le même ordre à travers les points de cheminement lorsque le drapeau `reversed` est réglé sur `true` (vrai). Cela signifie également que vous devez tenir compte de la direction du robot lorsque vous fournissez les points de cheminement. Par exemple, si votre robot fait face au mur de votre station d'alliance et se déplace vers l'arrière vers un élément situé sur le terrain, le point de cheminement de départ doit avoir une rotation de 180 degrés.

La génération de la trajectoire

La méthode utilisée pour générer une trajectoire est `generateTrajectory(...)`. Il existe quatre « surcharges » (Overloads) pour cette méthode. Deux qui utilisent des Splines cubiques et les deux autres qui utilisent des splines quintiques. Pour chaque type de Spline, il existe deux façons de construire une trajectoire. Les méthodes les plus simples utilisent des surcharges qui acceptent les objets `Pose2d`.

Pour les Splines cubiques, cette méthode accepte deux objets `Pose2d`, un pour le point de cheminement de départ et un pour le point de cheminement de fin. La méthode utilise un vecteur d'objets `Translation2d` qui représentent les points de cheminement intérieurs. Les

caps à ces points intérieurs sont déterminés automatiquement pour assurer une courbure continue. Pour les splines quintiques, la méthode prend simplement une liste d'objets `Pose2d`, chaque `Pose2d` représentant un point et un cap sur le champ.

La surcharge qui est plus complexe accepte les « vecteurs de contrôle » pour les Splines. Cette méthode est utilisée lors de la génération de trajectoires avec « Pathweaver », où vous pouvez contrôler l'amplitude du vecteur tangent à chaque point. La classe `ControlVector` se compose de deux tableaux `double`. Chaque tableau représente une dimension (x ou y) et ses éléments représentent les dérivées à ce point. Par exemple, la valeur à l'élément 0 du tableau x représente la coordonnée x (dérivée 0), la valeur à l'élément 1 représente la dérivée 1 dans la dimension x et ainsi de suite.

Lorsque vous utilisez des Splines cubiques, la taille du tableau doit permettre 2 éléments (dérivées 0 et 1), tandis que lorsque vous utilisez des Splines quintiques, la longueur du tableau doit être 3 (dérivée 0, 1 et 2). À moins que vous ne sachiez exactement ce que vous faites, la première méthode (et la plus simple) est FORTEMENT recommandée pour générer manuellement des trajectoires. (c'est-à-dire lorsque vous n'utilisez pas de fichiers JSON Pathweaver).

Voici un exemple de génération d'une trajectoire à l'aide de splines cubiques serrées pour le jeu 2018, FIRST Power Up :

Java

```
class ExampleTrajectory {
  public void generateTrajectory() {

    // 2018 cross scale auto waypoints.
    var sideStart = new Pose2d(Units.feetToMeters(1.54), Units.feetToMeters(23.23),
        Rotation2d.fromDegrees(-180));
    var crossScale = new Pose2d(Units.feetToMeters(23.7), Units.feetToMeters(6.8),
        Rotation2d.fromDegrees(-160));

    var interiorWaypoints = new ArrayList<Translation2d>();
    interiorWaypoints.add(new Translation2d(Units.feetToMeters(14.54), Units.
↪feetToMeters(23.23)));
    interiorWaypoints.add(new Translation2d(Units.feetToMeters(21.04), Units.
↪feetToMeters(18.23)));

    TrajectoryConfig config = new TrajectoryConfig(Units.feetToMeters(12), Units.
↪feetToMeters(12));
    config.setReversed(true);

    var trajectory = TrajectoryGenerator.generateTrajectory(
        sideStart,
        interiorWaypoints,
        crossScale,
        config);
  }
}
```

C++

```
void GenerateTrajectory() {
  // 2018 cross scale auto waypoints
  const frc::Pose2d sideStart{1.54_ft, 23.23_ft, frc::Rotation2d(180_deg)};
  const frc::Pose2d crossScale{23.7_ft, 6.8_ft, frc::Rotation2d(-160_deg)};

  std::vector<frc::Translation2d> interiorWaypoints{
      frc::Translation2d{14.54_ft, 23.23_ft},
      frc::Translation2d{21.04_ft, 18.23_ft}};

  frc::TrajectoryConfig config{12_fps, 12_fps_sq};
  config.SetReversed(true);

  auto trajectory = frc::TrajectoryGenerator::GenerateTrajectory(
      sideStart, interiorWaypoints, crossScale, config);
}
```

Python

```
def generateTrajectory():
    # 2018 cross scale auto waypoints.
    sideStart = Pose2d.fromFeet(1.54, 23.23, Rotation2d.fromDegrees(-180))
    crossScale = Pose2d.fromFeet(23.7, 6.8, Rotation2d.fromDegrees(-160))

    interiorWaypoints = [
        Translation2d.fromFeet(14.54, 23.23),
        Translation2d.fromFeet(21.04, 18.23),
    ]

    config = TrajectoryConfig.fromFps(12, 12)
    config.setReversed(True)

    trajectory = TrajectoryGenerator.generateTrajectory(
        sideStart, interiorWaypoints, crossScale, config
    )
```

Note : The Java code utilizes the Units utility, for easy unit conversions.

Note : La génération d'une trajectoire typique prend environ 10 ms à 25 ms. Ce n'est pas long, mais il est tout de même fortement recommandé de générer toutes les trajectoires au démarrage (`robotInit`).

Concaténation des trajectoires

Trajectories in Java can be combined into a single trajectory using the `concatenate(trajectory)` function. C++/Python users can simply add (+) the two trajectories together.

Avertissement : Il appartient à l'utilisateur de s'assurer que la fin de la trajectoire initiale et le début de la trajectoire ajoutée correspondent. Il est également de la responsabilité de l'utilisateur de s'assurer que les vitesses de début et de fin de ses trajectoires correspondent.

JAVA

```
var trajectoryOne =
TrajectoryGenerator.generateTrajectory(
   new Pose2d(0, 0, Rotation2d.fromDegrees(0)),
   List.of(new Translation2d(1, 1), new Translation2d(2, -1)),
   new Pose2d(3, 0, Rotation2d.fromDegrees(0)),
   new TrajectoryConfig(Units.feetToMeters(3.0), Units.feetToMeters(3.0)));

var trajectoryTwo =
TrajectoryGenerator.generateTrajectory(
   new Pose2d(3, 0, Rotation2d.fromDegrees(0)),
   List.of(new Translation2d(4, 4), new Translation2d(6, 3)),
   new Pose2d(6, 0, Rotation2d.fromDegrees(0)),
   new TrajectoryConfig(Units.feetToMeters(3.0), Units.feetToMeters(3.0)));

var concatTraj = trajectoryOne.concatenate(trajectoryTwo);
```

C++

```
auto trajectoryOne = frc::TrajectoryGenerator::GenerateTrajectory(
   frc::Pose2d(0_m, 0_m, 0_rad),
   {frc::Translation2d(1_m, 1_m), frc::Translation2d(2_m, -1_m)},
   frc::Pose2d(3_m, 0_m, 0_rad), frc::TrajectoryConfig(3_fps, 3_fps_sq));

auto trajectoryTwo = frc::TrajectoryGenerator::GenerateTrajectory(
   frc::Pose2d(3_m, 0_m, 0_rad),
   {frc::Translation2d(4_m, 4_m), frc::Translation2d(5_m, 3_m)},
   frc::Pose2d(6_m, 0_m, 0_rad), frc::TrajectoryConfig(3_fps, 3_fps_sq));

auto concatTraj = m_trajectoryOne + m_trajectoryTwo;
```

PYTHON

```python
from wpimath.geometry import Pose2d, Rotation2d, Translation2d
from wpimath.trajectory import TrajectoryGenerator, TrajectoryConfig


trajectoryOne = TrajectoryGenerator.generateTrajectory(
   Pose2d(0, 0, Rotation2d.fromDegrees(0)),
   [Translation2d(1, 1), Translation2d(2, -1)],
   Pose2d(3, 0, Rotation2d.fromDegrees(0)),
   TrajectoryConfig.fromFps(3.0, 3.0),
)

trajectoryTwo = TrajectoryGenerator.generateTrajectory(
   Pose2d(3, 0, Rotation2d.fromDegrees(0)),
   [Translation2d(4, 4), Translation2d(6, 3)],
   Pose2d(6, 0, Rotation2d.fromDegrees(0)),
   TrajectoryConfig.fromFps(3.0, 3.0),
)

concatTraj = trajectoryOne + trajectoryTwo
```

32.7.2 Les contraintes de trajectoire

Dans *l'article précédent*, vous avez peut-être remarqué qu'aucune contrainte personnalisée n'a été ajoutée lors de la génération des trajectoires. Les contraintes personnalisées permettent aux utilisateurs d'imposer plus de restrictions sur la vitesse et l'accélération aux points le long de la trajectoire en fonction de l'emplacement et de la courbure.

Par exemple, une contrainte personnalisée peut maintenir la vitesse de la trajectoire sous un certain seuil dans une certaine région ou ralentir le robot près des virages à des fins de stabilité.

Les contraintes fournies par WPILib

WPILib inclut un ensemble de contraintes prédéfinies que les utilisateurs peuvent utiliser lors de la génération de trajectoires. La liste des contraintes fournies par WPILib est la suivante :

- `CentripetalAccelerationConstraint` : Limite l'accélération centripète du robot lorsqu'il se déplace le long de la trajectoire. Cela peut aider à ralentir le robot dans les virages serrés.
- `DifferentialDriveKinematicsConstraint` : limite la vitesse du robot autour des virages de sorte qu'aucune roue d'un robot à entraînement différentiel ne dépasse une vitesse maximale spécifiée.
- `DifferentialDriveVoltageConstraint` : limite l'accélération d'un robot à entraînement différentiel de sorte qu'aucune tension de commande ne dépasse un voltage maximum spécifié.
- `EllipticalRegionConstraint` : impose une contrainte uniquement dans une région elliptique du champ.
- `MaxVelocityConstraint` : Impose une contrainte de vitesse maximale. Cela peut être composé avec la `EllipticalRegionConstraint` ou la `RectangularRegionConstraint` pour limiter la vitesse du robot uniquement dans une région spécifique.

— `MecanumDriveKinematicsConstraint` : limite la vitesse du robot dans les virages de sorte qu'aucune roue d'un robot avec entraînement Mécanum ne dépasse une vitesse maximale spécifiée.
— `RectangularRegionConstraint` : impose une contrainte uniquement dans une région spécifique du terrain (de forme rectangulaire).
— `SwerveDriveKinematicsConstraint` : Limite la vitesse du robot autour des virages de telle sorte qu'aucune roue d'un robot de type Swerve ne dépasse une vitesse maximale spécifiée.

Note : La `DifferentialDriveVoltageConstraint` garantit que les commandes de tension calculées ne dépassent pas le voltage maximum spécifié en utilisant un *modèle Feedforward*. Si le robot s'écarte de la référence pendant le suivi du parcours, la tension commandée pourrait être supérieure au voltage maximum spécifié.

La création d'une contrainte personnalisée

Les utilisateurs peuvent créer leur propre contrainte en appelant l'interface `TrajectoryConstraint`.

JAVA

```
@Override
public double getMaxVelocityMetersPerSecond(Pose2d poseMeters, double␣
↪curvatureRadPerMeter,
                                            double velocityMetersPerSecond) {
  // code here
}

@Override
public MinMax getMinMaxAccelerationMetersPerSecondSq(Pose2d poseMeters,
                                                     double curvatureRadPerMeter,
                                                     double velocityMetersPerSecond) {
  // code here
}
```

C++

```
units::meters_per_second_t MaxVelocity(
const Pose2d& pose, units::curvature_t curvature,
units::meters_per_second_t velocity) override {
  // code here
}

MinMax MinMaxAcceleration(const Pose2d& pose, units::curvature_t curvature,
                          units::meters_per_second_t speed) override {
  // code here
}
```

PYTHON

```python
from wpimath import units
from wpimath.geometry import Pose2d
from wpimath.trajectory.constraint import TrajectoryConstraint


class MyConstraint(TrajectoryConstraint):
    def maxVelocity(
        self,
        pose: Pose2d,
        curvature: units.radians_per_meter,
        velocity: units.meters_per_second,
    ) -> units.meters_per_second:
        ...

    def minMaxAcceleration(
        self,
        pose: Pose2d,
        curvature: units.radians_per_meter,
        speed: units.meters_per_second,
    ) -> TrajectoryConstraint.MinMax:
        ...
```

La méthode `MaxVelocity` retourne la vitesse maximale autorisée pour la pose, la courbure et la vitesse d'origine données de la trajectoire sans aucune contrainte. La méthode `MinMaxAcceleration` retourne l'accélération minimale et maximale autorisée pour la pose donnée, la courbure et la vitesse.

Voir le code source (Java, C++) pour les contraintes fournies par WPILib pour plus d'exemples sur la façon d'écrire vos propres contraintes de trajectoire personnalisées.

32.7.3 La manipulation des trajectoires

Une fois qu'une trajectoire a été générée, vous pouvez en extraire des informations à l'aide de certaines méthodes. Ces méthodes seront utiles lors de l'écriture de code pour suivre ces trajectoires.

Obtenir la durée totale de la trajectoire

Because all trajectories have timestamps at each point, the amount of time it should take for a robot to traverse the entire trajectory is pre-determined. The `TotalTime()` (C++) / `getTotalTimeSeconds()` (Java) / `totalTime` (Python) method can be used to determine the time it takes to traverse the trajectory.

JAVA

```
// Get the total time of the trajectory in seconds
double duration = trajectory.getTotalTimeSeconds();
```

C++

```
// Get the total time of the trajectory
units::second_t duration = trajectory.TotalTime();
```

PYTHON

```
# Get the total time of the trajectory
duration = trajectory.totalTime()
```

Échantillonnage de la trajectoire

The trajectory can be sampled at various timesteps to get the pose, velocity, and acceleration at that point. The `Sample(units::second_t time)` (C++) / `sample(double timeSeconds)` (Java/Python) method can be used to sample the trajectory at any timestep. The parameter refers to the amount of time passed since 0 seconds (the starting point of the trajectory). This method returns a `Trajectory::Sample` with information about that sample point.

JAVA

```
// Sample the trajectory at 1.2 seconds. This represents where the robot
// should be after 1.2 seconds of traversal.
Trajectory.Sample point = trajectory.sample(1.2);
```

C++

```
// Sample the trajectory at 1.2 seconds. This represents where the robot
// should be after 1.2 seconds of traversal.
Trajectory::State point = trajectory.Sample(1.2_s);
```

PYTHON

```
# Sample the trajectory at 1.2 seconds. This represents where the robot
# should be after 1.2 seconds of traversal.
point = trajectory.sample(1.2)
```

La structure `Trajectory::Sample` contient plusieurs informations sur le point d'échantillonnage :

- `t` : Le temps écoulé depuis le début de la trajectoire jusqu'au point d'échantillonnage.
- `velocity` : la vélocité au point d'échantillonnage.
- `acceleration` : l'accélération au point d'échantillonnage.
- `pose` : la pose (x, y, cap) au point d'échantillonnage.
- `curvature` : la courbure (taux de changement de cap par rapport à la distance le long de la trajectoire) au point d'échantillonnage.

Remarque : La vitesse angulaire au point d'échantillonnage peut être calculée en multipliant la vitesse par la courbure.

Obtention de tous les états de la trajectoire (avancé)

A more advanced user can get a list of all states of the trajectory by calling the `States()` (C++) / `getStates()` (Java) / `states` (Python) method. Each state represents a point on the trajectory. *When the trajectory is created* using the `TrajectoryGenerator::GenerateTrajectory(...)` method, a list of trajectory points / states are created. When the user samples the trajectory at a particular timestep, a new sample point is interpolated between two existing points / states in the list.

32.7.4 La transformation des trajectoires

Les trajectoires peuvent être transformées d'un système de coordonnées à un autre et déplacées dans un système de coordonnées à l'aide des méthodes `relativeTo` et `transformBy`. Ces méthodes sont utiles pour déplacer des trajectoires dans l'espace ou redéfinir une trajectoire déjà existante dans un autre cadre de référence.

Note : Aucune de ces méthodes ne modifie la forme de la trajectoire d'origine.

La méthode `relativeTo`

La méthode `relativeTo` permet de redéfinir une trajectoire déjà existante dans un autre référentiel. Cette méthode prend un argument : une pose, (via un objet `Pose2d`) qui est définie par rapport au système de coordonnées courant, qui représente l'origine du nouveau système de coordonnées.

Par exemple, une trajectoire définie dans le système de coordonnées A peut être redéfinie dans le système de coordonnées B, dont l'origine est à (3, 3, 30 degrés) dans le système de coordonnées A, en utilisant la méthode `relativeTo`.

JAVA

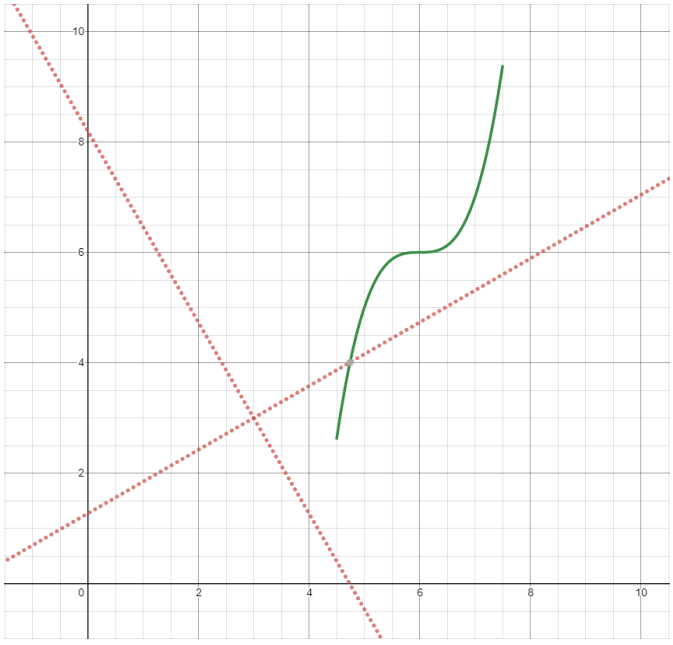
```
Pose2d bOrigin = new Pose2d(3, 3, Rotation2d.fromDegrees(30));
Trajectory bTrajectory = aTrajectory.relativeTo(bOrigin);
```

C++

```
frc::Pose2d bOrigin{3_m, 3_m, frc::Rotation2d(30_deg)};
frc::Trajectory bTrajectory = aTrajectory.RelativeTo(bOrigin);
```

PYTHON

```
from wpimath.geometry import Pose2d, Rotation2d

bOrigin = Pose2d(3, 3, Rotation2d.fromDegrees(30))
bTrajectory = aTrajectory.relativeTo(bOrigin)
```



Dans le diagramme ci-dessus, la trajectoire originale (`aTrajectory` dans le code ci-dessus) a été définie dans le système de coordonnées A, représenté par les axes noirs. Les axes rouges, situés à (3, 3) et 30° par rapport au système de coordonnées d'origine, représentent le système de coordonnées B. Appeler `relativeTo` sur `aTrajectory` redéfinira toutes les poses de la trajectoire pour être par rapport au système de coordonnées B (axes rouges).

La méthode `transformBy`

La méthode `transformBy` peut être utilisée pour déplacer (c'est-à-dire traduire et faire pivoter) une trajectoire dans un système de coordonnées. Cette méthode prend un seul argument : soit une transformation (via un objet `Transform2d`) qui mappe la position initiale actuelle de la trajectoire à une position initiale souhaitée de la même trajectoire.

Par exemple, on peut vouloir transformer une trajectoire qui commence à (2, 2, 30 degrés) pour la faire commencer à (4, 4, 50 degrés) en utilisant la méthode `transformBy`.

JAVA

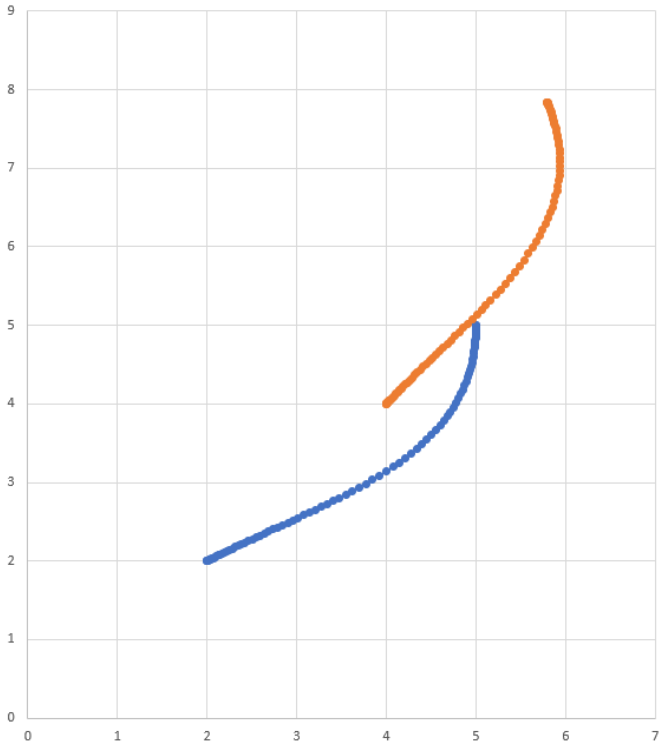
```
Transform2d transform = new Pose2d(4, 4, Rotation2d.fromDegrees(50)).minus(trajectory.
↪getInitialPose());
Trajectory newTrajectory = trajectory.transformBy(transform);
```

C++

```
frc::Transform2d transform = Pose2d(4_m, 4_m, Rotation2d(50_deg)) - trajectory.
↪InitialPose();
frc::Trajectory newTrajectory = trajectory.TransformBy(transform);
```

PYTHON

```
from wpimath.geometry import Pose2d, Rotation2d

transform = Pose2d(4, 4, Rotation2d.fromDegrees(50)) - trajectory.initialPose()
newTrajectory = trajectory.transformBy(transform)
```



Dans le diagramme ci-dessus, la trajectoire d'origine, qui commence à (2, 2) et à 30 ° est visible en bleu. Après avoir appliqué la transformation ci-dessus, l'emplacement de départ de la trajectoire résultante est changé en (4, 4) à 50 °. La trajectoire résultante est visible en orange.

32.7.5 Contrôleur Ramsete

Le contrôleur Ramsete est un tracker de trajectoire intégré à WPILib. Ce tracker peut être utilisé pour suivre avec précision les trajectoires avec correction des perturbations mineures.

Construire l'objet contrôleur Ramsete

Le contrôleur Ramsete doit être initialisé avec deux gains, à savoir `b` et `zeta`. Des valeurs plus grandes de `b` rendent la convergence plus agressive comme un terme proportionnel tandis que des valeurs plus grandes de `zeta` fournissent plus d'amortissement dans la réponse. Ces gains de contrôleur dictent uniquement la manière dont le contrôleur produira les vitesses ajustées. Cela n'affecte PAS le suivi de la vitesse réelle du robot. Cela signifie que ces gains de contrôleur sont généralement indépendants du robot.

Note : Les gains de `2.0` et `0.7` pour `b` et `zeta` ont été testés à plusieurs reprises pour produire des résultats souhaitables lorsque toutes les unités étaient en mètres. En tant que tel, un constructeur qui ne passe aucun argument à `RamseteController` aura ses gains réglés par défaut à ces valeurs.

JAVA

```
// Using the default constructor of RamseteController. Here
// the gains are initialized to 2.0 and 0.7.
RamseteController controller1 = new RamseteController();

// Using the secondary constructor of RamseteController where
// the user can choose any other gains.
RamseteController controller2 = new RamseteController(2.1, 0.8);
```

C++

```
// Using the default constructor of RamseteController. Here
// the gains are initialized to 2.0 and 0.7.
frc::RamseteController controller1;

// Using the secondary constructor of RamseteController where
// the user can choose any other gains.
frc::RamseteController controller2{2.1, 0.8};
```

PYTHON

```
from wpimath.controller import RamseteController

# Using the default constructor of RamseteController. Here
# the gains are initialized to 2.0 and 0.7.
controller1 = RamseteController()

# Using the secondary constructor of RamseteController where
# the user can choose any other gains.
controller2 = RamseteController(2.1, 0.8)
```

Obtention de vitesses ajustées

Le contrôleur Ramsete renvoie des «vitesses ajustées» de sorte que lorsque le robot respecte ces vitesses, il atteint avec précision le point cible. Le contrôleur doit être mis à jour périodiquement avec le nouvel objectif. Le point cible comprend une pose souhaitée, une vitesse linéaire souhaitée et une vitesse angulaire souhaitée. De plus, la position actuelle du robot doit également être mise à jour périodiquement. Le contrôleur utilise ces quatre arguments pour renvoyer la vitesse linéaire et angulaire ajustée. Les utilisateurs doivent commander leur robot à ces vitesses linéaires et angulaires pour obtenir un suivi de trajectoire optimal.

Note : The « goal pose » represents the position that the robot should be at a particular timestep when tracking the trajectory. It does NOT represent the final endpoint of the trajectory.

The controller can be updated using the `Calculate` (C++) / `calculate` (Java/Python) method. There are two overloads for this method. Both of these overloads accept the current robot position as the first parameter. For the other parameters, one of these overloads takes in

the goal as three separate parameters (pose, linear velocity, and angular velocity) whereas the other overload accepts a `Trajectory.State` object, which contains information about the goal pose. For its ease, users should use the latter method when tracking trajectories.

JAVA

```
Trajectory.State goal = trajectory.sample(3.4); // sample the trajectory at 3.4␣
↪seconds from the beginning
ChassisSpeeds adjustedSpeeds = controller.calculate(currentRobotPose, goal);
```

C++

```
const Trajectory::State goal = trajectory.Sample(3.4_s); // sample the trajectory at␣
↪3.4 seconds from the beginning
ChassisSpeeds adjustedSpeeds = controller.Calculate(currentRobotPose, goal);
```

PYTHON

```
goal = trajectory.sample(3.4)  # sample the trajectory at 3.4 seconds from the␣
↪beginning
adjustedSpeeds = controller.calculate(currentRobotPose, goal)
```

Ces calculs doivent être effectués à chaque itération de boucle, avec une position et un objectif du robot mis à jour.

Utilisation des vitesses ajustées

Les vitesses ajustées sont de type `ChassisSpeeds`, qui contient un `vx` (vitesse linéaire dans la direction avant), un `vy` (vitesse linéaire dans la direction latérale) et un `omega` (vitesse angulaire autour du centre du châssis du robot). Étant donné que le contrôleur Ramsete est un contrôleur pour les robots non holonomiques (robots qui ne peuvent pas se déplacer latéralement), l'objet à vitesses ajustées a un `vy` de zéro.

Les vitesses ajustées renvoyées peuvent être converties en vitesses utilisables en utilisant les classes cinématiques pour votre type de transmission. Par exemple, les vitesses ajustées peuvent être converties en vitesses gauche et droite pour un entraînement différentiel à l'aide d'un objet `DifferentialDriveKinematics`.

JAVA

```
ChassisSpeeds adjustedSpeeds = controller.calculate(currentRobotPose, goal);
DifferentialDriveWheelSpeeds wheelSpeeds = kinematics.toWheelSpeeds(adjustedSpeeds);
double left = wheelSpeeds.leftMetersPerSecond;
double right = wheelSpeeds.rightMetersPerSecond;
```

C++

```
ChassisSpeeds adjustedSpeeds = controller.Calculate(currentRobotPose, goal);
DifferentialDriveWheelSpeeds wheelSpeeds = kinematics.ToWheelSpeeds(adjustedSpeeds);
auto [left, right] = kinematics.ToWheelSpeeds(adjustedSpeeds);
```

PYTHON

```
adjustedSpeeds = controller.calculate(currentRobotPose, goal)
wheelSpeeds = kinematics.toWheelSpeeds(adjustedSpeeds)
left = wheelSpeeds.left
right = wheelSpeeds.right
```

Because these new left and right velocities are still speeds and not voltages, two PID Controllers, one for each side may be used to track these velocities. Either the WPILib PIDController (C++, Java, Python) can be used, or the Velocity PID feature on smart motor controllers such as the TalonSRX and the SPARK MAX can be used.

Ramsete dans un environnement basé sur les commandes (Command -based Framework)

Par souci de facilité pour les utilisateurs, une classe `RamseteCommand` est intégrée à WPILib. Pour un didacticiel complet sur la mise en œuvre d'un mode autonome qui suit une trajectoire précise à l'aide de RamseteCommand, voir *Didacticiel sur la pratique des trajectoires*.

32.7.6 Contrôleur pour entraînement de type holonomique

Le contrôleur pour entraînement de type holonomique permet de tracer la trajectoire pour un robot ayant une transmission holonomique (par exemple, à embardée (Swerve) ou Mécanum, etc.). Ceci peut être utilisé pour suivre avec précision une trajectoire avec la possibilité de corriger des perturbations mineures.

Construire un contrôleur holonomique

Le contrôleur d'entraînement holonomique doit être instancié avec 2 contrôleurs PID et 1 contrôleur PID profilé.

Note : Pour plus d'informations sur le contrôle PID, voir *Contrôle PID dans WPILib*.

Les 2 contrôleurs PID sont des contrôleurs qui doivent corriger les erreurs respectivement dans les directions x et y relatives au champ. Par exemple, si les 2 premiers arguments sont respectivement `PIDController(1, 0, 0)` et `PIDController(1.2, 0, 0)`, le contrôleur de lecteur holonomique ajoutera un mètre supplémentaire par seconde dans le x direction pour chaque mètre d'erreur dans la direction x et ajoutera 1,2 mètre supplémentaire par seconde dans la direction y pour chaque mètre d'erreur dans la direction y.

Le paramètre final est un `ProfiledPIDController` pour la rotation du robot. Étant donné que la dynamique de rotation d'une transmission holonomique est découplée du mouvement

dans les directions x et y, les utilisateurs peuvent définir des références de cap personnalisées lorsque le but est de suivre une trajectoire. Ces références d'en-tête sont profilées en fonction des paramètres définis dans le `ProfiledPIDController`.

JAVA

```
var controller = new HolonomicDriveController(
  new PIDController(1, 0, 0), new PIDController(1, 0, 0),
  new ProfiledPIDController(1, 0, 0,
    new TrapezoidProfile.Constraints(6.28, 3.14)));
// Here, our rotation profile constraints were a max velocity
// of 1 rotation per second and a max acceleration of 180 degrees
// per second squared.
```

C++

```
frc::HolonomicDriveController controller{
  frc::PIDController{1, 0, 0}, frc::PIDController{1, 0, 0},
  frc::ProfiledPIDController<units::radian>{
    1, 0, 0, frc::TrapezoidProfile<units::radian>::Constraints{
      6.28_rad_per_s, 3.14_rad_per_s / 1_s}}};
// Here, our rotation profile constraints were a max velocity
// of 1 rotation per second and a max acceleration of 180 degrees
// per second squared.
```

PYTHON

```
from wpimath.controller import (
   HolonomicDriveController,
   PIDController,
   ProfiledPIDControllerRadians,
)
from wpimath.trajectory import TrapezoidProfileRadians

controller = HolonomicDriveController(
   PIDController(1, 0, 0),
   PIDController(1, 0, 0),
   ProfiledPIDControllerRadians(
      1, 0, 0, TrapezoidProfileRadians.Constraints(6.28, 3.14)
   ),
)
# Here, our rotation profile constraints were a max velocity
# of 1 rotation per second and a max acceleration of 180 degrees
# per second squared.
```

Obtenir des vitesses ajustées

Le contrôleur d'entraînement holonomique renvoie des « vitesses ajustées » de telle sorte que lorsque le robot suit ces vitesses, il atteint avec précision le point cible. Le contrôleur doit être mis à jour périodiquement avec le nouvel objectif. L'objectif est composé d'une pose désirée, d'une vitesse linéaire et d'un cap souhaités.

Note : The « goal pose » represents the position that the robot should be at a particular timestamp when tracking the trajectory. It does NOT represent the trajectory's endpoint.

The controller can be updated using the `Calculate` (C++) / `calculate` (Java/Python) method. There are two overloads for this method. Both of these overloads accept the current robot position as the first parameter and the desired heading as the last parameter. For the middle parameters, one overload accepts the desired pose and the linear velocity reference while the other accepts a `Trajectory.State` object, which contains information about the goal pose. The latter method is preferred for tracking trajectories.

JAVA

```
// Sample the trajectory at 3.4 seconds from the beginning.
Trajectory.State goal = trajectory.sample(3.4);

// Get the adjusted speeds. Here, we want the robot to be facing
// 70 degrees (in the field-relative coordinate system).
ChassisSpeeds adjustedSpeeds = controller.calculate(
  currentRobotPose, goal, Rotation2d.fromDegrees(70.0));
```

C++

```
// Sample the trajectoty at 3.4 seconds from the beginning.
const auto goal = trajectory.Sample(3.4_s);

// Get the adjusted speeds. Here, we want the robot to be facing
// 70 degrees (in the field-relative coordinate system).
const auto adjustedSpeeds = controller.Calculate(
  currentRobotPose, goal, 70_deg);
```

PYTHON

```
from wpimath.geometry import Rotation2d

# Sample the trajectory at 3.4 seconds from the beginning.
goal = trajectory.sample(3.4)

# Get the adjusted speeds. Here, we want the robot to be facing
# 70 degrees (in the field-relative coordinate system).
adjustedSpeeds = controller.calculate(
   currentRobotPose, goal, Rotation2d.fromDegrees(70.0)
)
```

Utilisation des vitesses ajustées

Les vitesses ajustées sont de type `ChassisSpeeds`, qui contient un `vx` (vitesse linéaire dans le sens avant), un `vy` (vitesse linéaire dans le sens latéral) et un `omega` (vitesse angulaire autour du centre du châssis du robot).

Les vitesses ajustées renvoyées peuvent être converties en vitesses utilisables à l'aide des classes de cinématique correspondant à votre type de transmission. Dans l'exemple de code ci-dessous, nous supposerons un robot à entraînement de type Swerve ; cependant, le code cinématique est exactement le même pour un robot avec entraînement Mécanum sauf en utilisant `MecanumDriveKinematics`.

JAVA

```
SwerveModuleState[] moduleStates = kinematics.toSwerveModuleStates(adjustedSpeeds);

SwerveModuleState frontLeft = moduleStates[0];
SwerveModuleState frontRight = moduleStates[1];
SwerveModuleState backLeft = moduleStates[2];
SwerveModuleState backRight = moduleStates[3];
```

C++

```
auto [fl, fr, bl, br] = kinematics.ToSwerveModuleStates(adjustedSpeeds);
```

PYTHON

```
fl, fr, bl, br = kinematics.toSwerveModuleStates(adjustedSpeeds)
```

Étant donné que les états de module pour un entraînement de type Swerve sont définis par des vitesses et des angles, vous devrez utiliser des contrôleurs PID pour définir ces vitesses et ces angles.

32.7.7 Dépannage

Dépannage des échecs complets

Il y a un certain nombre de choses qui peuvent amener votre robot à complètement avoir le mauvais comportement. La liste de vérification ci-dessous couvre les erreurs les plus courantes.

- Mon robot ne bouge pas.
  - Produisez-vous réellement vos moteurs ?
  - Une `MalformedSplineException` est-elle imprimée sur le poste de conduite ? Si oui, passez à la section `MalformedSplineException` ci-dessous.
  - Votre trajectoire est-elle très courte ou dans les mauvaises unités ?
- Mon robot pivote pour suivre la trajectoire dans l'autre sens.
  - Les titres de début et de fin de votre trajectoire sont-ils incorrects ?
  - Le gyroscope de votre robot est-il réinitialisé dans la mauvaise direction ?

  - *Avez-vous le drapeau inversé mal réglé ?*
  - Vos angles gyroscopiques sont-ils positifs dans le sens horaire ? Si oui, vous devez inverser le signe (nouvel angle = 0 - angle gyro).
- Mon robot roule en ligne droite même s'il doit tourner.
  - Votre gyroscope est-il correctement configuré et renvoie-t-il de bonnes données ?
  - Utilisez-vous les bonnes unités de mesure lorsque vous passez votre cap gyroscopique à votre objet odométrique ?
  - La largeur spécifiée de votre robot (track width) est-elle correcte ? Est-ce dans les bonnes unités ?
- Je reçois un message `MalformedSplineException` sur le poste de pilotage et le robot ne bouge pas.
  - *Avez-vous le drapeau inversé mal réglé ?*
  - Avez-vous deux points de cheminement très proches l'un de l'autre avec des caps diamétralement opposés ?
  - Avez-vous deux points de cheminement avec les mêmes (ou presque les mêmes) coordonnées ?
- Mon robot va trop loin.
  - Vos conversions d'unités d'encodeur sont-elles correctement configurées ?
  - Vos encodeurs sont-ils connectés ?
- Mon robot fait surtout ce qu'il faut, mais c'est un peu inexact.
  - Passez à la section suivante.

Dépannage des mauvaises performances

Note : Cette section concerne principalement le dépannage des performances de suivi de trajectoire médiocres, comme une erreur de positionnement de 1 mètre, et non les défaillances catastrophiques comme les erreurs de compilation, les robots se retournant et allant dans la mauvaise direction, ou les erreurs comme `MalformedSplineException`s.

Note : Cette section est conçue pour les robots à entraînement différentiel, mais la plupart des idées peuvent être adaptées pour les robots de type Swerve ou Mécanum

---

Les mauvaises performances de suivi de trajectoire peuvent être difficiles à dépanner. Bien que le générateur de trajectoire et le suiveur soient faciles à utiliser et performants tel quel, il y a des situations où votre robot n'accomplit pas toute sa trajectoire. Le générateur de trajectoire et les suiveurs ont de nombreux ajustements logiciels et mécaniques, il peut donc être difficile de savoir par où commencer, en particulier parce qu'il est difficile de localiser la source des problèmes de trajectoire à partir du comportement général du robot.

Parce qu'il peut être si difficile de localiser la section du générateur de trajectoire et des suiveurs qui est fautive, une approche systématique, étape par étape est recommandée pour les mauvaises performances de suivi générales (comme exemple, le robot a dévié de 20 degrés ou est hors-parcours de quelques pieds). Les étapes ci-dessous sont répertoriées dans l'ordre dans lequel vous devez les effectuer ; il est important de suivre cet ordre afin de pouvoir isoler les effets des différentes étapes les unes des autres.

Note : Les exemples ci-dessous mettent des valeurs de diagnostic sur les *NetworkTables*. La façon la plus simple de représenter graphiquement ces valeurs est d'utiliser les capacités graphiques de *Shuffleboard*.

Vérifier l'odométrie

Si votre odométrie est mauvaise, votre contrôleur Ramsete peut se comporter de façon erronée, car il modifie les vitesses cibles de votre robot en fonction de l'endroit où votre odométrie pense que le robot se trouve.

Note : *Envoi de la pose et de la trajectoire de votre robot à field2d* peut aider à vérifier que votre robot se déplace correctement par rapport à sa trajectoire.

---

1. Configurez votre code pour enregistrer la position de votre robot après chaque mise à jour de l'odométrie :

JAVA

```
NetworkTableEntry m_xEntry = NetworkTableInstance.getDefault().getTable(
↪"troubleshooting").getEntry("X");
NetworkTableEntry m_yEntry = NetworkTableInstance.getDefault().getTable(
↪"troubleshooting").getEntry("Y");

@Override
public void periodic() {
    // Update the odometry in the periodic block
    m_odometry.update(Rotation2d.fromDegrees(getHeading()), m_leftEncoder.
↪getDistance(),
        m_rightEncoder.getDistance());

    var translation = m_odometry.getPoseMeters().getTranslation();
    m_xEntry.setNumber(translation.getX());
    m_yEntry.setNumber(translation.getY());
}
```

C++

```
NetworkTableEntry m_xEntry = nt::NetworkTableInstance::GetDefault().GetTable(
↪"troubleshooting")->GetEntry("X");
NetworkTableEntry m_yEntry = nt::NetworkTableInstance::GetDefault().GetTable(
↪"troubleshooting")->GetEntry("Y");

void DriveSubsystem::Periodic() {
    // Implementation of subsystem periodic method goes here.
    m_odometry.Update(frc::Rotation2d(units::degree_t(GetHeading())),
                      units::meter_t(m_leftEncoder.GetDistance()),
                      units::meter_t(m_rightEncoder.GetDistance()));

    auto translation = m_odometry.GetPose().Translation();
    m_xEntry.SetDouble(translation.X().value());
    m_yEntry.SetDouble(translation.Y().value());
}
```

2. Disposez un ruban à mesurer parallèle à votre robot et poussez votre robot d'environ un mètre le long du ruban à mesurer. Disposez un ruban à mesurer le long de l'axe Y

et recommencez, en poussant votre robot d'un mètre le long de l'axe X et d'un mètre le long de l'axe Y en arc de cercle.

3. Compare X and Y reported by the robot to actual X and Y. If X is off by more than 5 centimeters in the first test then you should check that you measured your wheel diameter correctly, and that your wheels are not worn down. If the second test is off by more than 5 centimeters in either X or Y then your track width (distance from the center of the left wheel to the center of the right wheel) may be incorrect ; if you're sure that you measured the track width correctly with a tape measure then your robot's wheels may be slipping in a way that is not accounted for by track width, so try increasing the track width number or measuring it programmatically.

Vérifier la fonction Feedforward

Si vos informations sont mauvaises, les contrôleurs P de chaque côté du robot ne suivront pas aussi bien et votre `DifferentialDriveVoltageConstraint` ne limitera pas l'accélération de votre robot avec précision. Nous voulons surtout désactiver les contrôleurs P de la roue afin de pouvoir isoler et tester les feedforwards.

1. Tout d'abord, nous devons désactiver le paramètre P pour neutraliser le PID pour chaque roue. Réglez le gain `P` à 0 pour chaque contrôleur. Dans l'exemple `RamseteCommand`, vous définiriez `kPDriveVel` à 0 :

JAVA

```
123             new PIDController(DriveConstants.kPDriveVel, 0, 0),
124             new PIDController(DriveConstants.kPDriveVel, 0, 0),
```

C++

```
81      frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
82      frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
```

1. Ensuite, nous voulons désactiver le contrôleur Ramsete pour faciliter l'isolement de notre comportement problématique. Pour ce faire, invoquer simplement la méthode `setEnabled(false)` sur `RamseteController` passé à votre `RamseteCommand` :

JAVA

```
RamseteController m_disabledRamsete = new RamseteController();
m_disabledRamsete.setEnabled(false);

// Be sure to pass your new disabledRamsete variable
RamseteCommand ramseteCommand = new RamseteCommand(
    exampleTrajectory,
    m_robotDrive::getPose,
    m_disabledRamsete,
    ...
);
```

C++

```
frc::RamseteController m_disabledRamsete;
m_disabledRamsete.SetEnabled(false);

// Be sure to pass your new disabledRamsete variable
frc2::RamseteCommand ramseteCommand(
  exampleTrajectory,
  [this]() { return m_drive.GetPose(); },
  m_disabledRamsete,
  ...
);
```

3. Enfin, nous devons enregistrer la vitesse de roue souhaitée et la vitesse de roue réelle (vous devez mettre les vitesses réelles et souhaitées sur le même graphique si vous utilisez Shuffleboard, ou si votre logiciel graphique a cette capacité) :

JAVA

```
var table = NetworkTableInstance.getDefault().getTable("troubleshooting");
var leftReference = table.getEntry("left_reference");
var leftMeasurement = table.getEntry("left_measurement");
var rightReference = table.getEntry("right_reference");
var rightMeasurement = table.getEntry("right_measurement");

var leftController = new PIDController(kPDriveVel, 0, 0);
var rightController = new PIDController(kPDriveVel, 0, 0);
RamseteCommand ramseteCommand = new RamseteCommand(
    exampleTrajectory,
    m_robotDrive::getPose,
    disabledRamsete, // Pass in disabledRamsete here
    new SimpleMotorFeedforward(ksVolts, kvVoltSecondsPerMeter,␣
↪kaVoltSecondsSquaredPerMeter),
    kDriveKinematics,
    m_robotDrive::getWheelSpeeds,
    leftController,
    rightController,
    // RamseteCommand passes volts to the callback
    (leftVolts, rightVolts) -> {
        m_robotDrive.tankDriveVolts(leftVolts, rightVolts);

        leftMeasurement.setNumber(m_robotDrive.getWheelSpeeds().leftMetersPerSecond);
        leftReference.setNumber(leftController.getSetpoint());

        rightMeasurement.setNumber(m_robotDrive.getWheelSpeeds().
↪rightMetersPerSecond);
        rightReference.setNumber(rightController.getSetpoint());
    },
    m_robotDrive
);
```

C++

```
auto table =
    nt::NetworkTableInstance::GetDefault().GetTable("troubleshooting");
auto leftRef = table->GetEntry("left_reference");
auto leftMeas = table->GetEntry("left_measurement");
auto rightRef = table->GetEntry("right_reference");
auto rightMeas = table->GetEntry("right_measurement");

frc::PIDController leftController(DriveConstants::kPDriveVel, 0, 0);
frc::PIDController rightController(DriveConstants::kPDriveVel, 0, 0);
frc2::RamseteCommand ramseteCommand(
    exampleTrajectory, [this]() { return m_drive.GetPose(); },
    frc::RamseteController(AutoConstants::kRamseteB,
                           AutoConstants::kRamseteZeta),
    frc::SimpleMotorFeedforward<units::meters>(
        DriveConstants::ks, DriveConstants::kv, DriveConstants::ka),
    DriveConstants::kDriveKinematics,
    [this] { return m_drive.GetWheelSpeeds(); }, leftController,
    rightController,
    [=](auto left, auto right) {
        auto leftReference = leftRef;
        auto leftMeasurement = leftMeas;
        auto rightReference = rightRef;
        auto rightMeasurement = rightMeas;

        m_drive.TankDriveVolts(left, right);

        leftMeasurement.SetDouble(m_drive.GetWheelSpeeds().left.value());
        leftReference.SetDouble(leftController.GetSetpoint());

        rightMeasurement.SetDouble(m_drive.GetWheelSpeeds().right.value());
        rightReference.SetDouble(rightController.GetSetpoint());
    },
    {&m_drive});
```

4. Exécutez le robot sur une variété de trajectoires (courbe et ligne droite) et vérifiez si la vitesse réelle suit la vitesse souhaitée en regardant les graphiques de NetworkTables.
5. Si les valeurs souhaitées et réelles sont *beaucoup* différentes, vous devez vérifier si le diamètre de la roue et l'"encoderEPR que vous avez utilisés pour l'identification du système étaient corrects. Si vous avez vérifié que vos unités et vos conversions sont correctes, vous devriez essayer de requalifier au même étage que celui sur lequel vous testez pour voir si vous pouvez obtenir de meilleures données.

Vérifiez le gain proportionnel P

Si vous avez effectué l'étape précédente et que le problème est réglé, alors le problème initial se situe peut-être dans la boucle de contrôle. Dans cette étape, nous allons vérifier que les paramètres P pour les contrôleurs de roue soient bien réglés. Si vous utilisez Java, nous voulons désactiver Ramsete afin que nous puissions simplement visualiser nos contrôleurs PF par eux-mêmes.

1. Vous devez réutiliser tout le code de l'étape précédente qui enregistre la vitesse réelle par rapport à la vitesse souhaitée (et le code qui désactive Ramsete, si vous utilisez Java), sauf que **le gain P doit être ramené à sa valeur initiale**

2. Exécutez à nouveau le robot sur une variété de trajectoires et vérifiez que vos graphiques réels et souhaités semblent bons
3. Si les graphiques ne semblent pas bons (c'est-à-dire que la vitesse réelle est très différente de celle souhaitée), vous devriez essayer de régler votre gain P et de réexécuter vos trajectoires de test.

Vérifier les contraintes

Note : Assurez-vous que votre gain P est différent de zéro pour cette étape et que vous avez toujours le code de prises de données ajouté dans les étapes précédentes. Si vous utilisez Java, vous devez supprimer le code pour désactiver Ramsete.

Si votre problème de précision a persisté à travers toutes les étapes précédentes, vous pourriez avoir un problème avec vos contraintes. Vous trouverez ci-dessous une liste des symptômes que les différentes contraintes disponibles présenteront lorsqu'elles sont mal réglées.

Vérifiez une seule contrainte à la fois ! Supprimez les autres contraintes, ajustez l'unique contrainte restante et répétez ce processus pour chaque contrainte que vous souhaitez vérifier. La liste de contrôle ci-dessous suppose que vous n'utilisez qu'une seule contrainte à la fois.

— `DifferentialDriveVoltageConstraint` :
  — Si votre robot accélère très lentement, il est possible que la tension maximale (voltage) pour cette contrainte soit trop faible.
  — Si votre robot n'atteint pas la fin du chemin, les données d'identification de votre système peuvent être problématiques.
— `DifferentialDriveKinematicsConstraint` :
  — Si le cap de votre robot est orienté vers la mauvaise direction, il est possible que la vitesse maximale permise du contrôleur soit trop faible ou trop élevée. La seule façon de vérifier cela est d'ajuster la vitesse maximale à une valeur différente et de voir ce qui se passe.
— `CentripetalAccelerationConstraint` :
  — Si le cap de votre robot pointe vers la mauvaise direction, ce paramètre ci-haut pourrait être le coupable. Si votre robot ne semble pas tourner suffisamment, vous devez augmenter l'accélération centripète maximale, mais s'il semble faire des virages serrés trop rapidement, vous devez diminuer l'accélération centripète maximale.

Vérifier les points de cheminement de la trajectoire

Il est possible que votre trajectoire elle-même ne soit pas très pilotable. Essayez de déplacer certains points de cheminement (et les caps aux points de cheminement, le cas échéant) pour réduire les virages serrés.

32.8 Contrôle basé sur l'espace-état (State-Space) et le modèle avec WPILib

Cette section fournit une introduction et décrit la prise en charge de WPILib pour le contrôle de l'espace-état.

32.8.1 Introduction au Contrôle de l'Espace-État

Note : Cet article est tiré de Controls Engineering in FRC par Tyler Veness avec permission.

Du PID au Contrôle à base d'un modèle

Quand nous réglons les contrôleurs PID, nous nous concentrons le réglage des paramètres du contrôleur liés à l'erreur actuel, du passé et du futur. (termes P, I et D) au lieu de nous concentrer aux états du système sous-jacent. Bien que cette approche fonctionne dans beaucoup de situations, c'est une vue incomplète de la réalité du système étudié.

Le contrôle à partir d'un modèle se concentre sur le développement d'un modèle précis du *système* (mécanisme) que nous essayons de contrôler. Ces modèles aident à affiner les *gains* choisis pour les commandes de rétroaction basées sur les réponses phyiques du système, au lieu d'avoir un *gain* proportionnel arbitraire dérivé de tests. Cela nous permet non seulement de prédire comment un système va réagir, mais aussi de tester nos contrôleurs sans avoir un robot physique en notre possession et gagner du temps pendant le débogage des simples bogues .

Note : State-space control makes extensive use of linear algebra. More on linear algebra in modern control theory, including an introduction to linear algebra and resources, can be found in Chapter 5 of Controls Engineering in FRC.

Si vous avez utilisé les classes feedforward de WPILib pour `SimpleMotorFeedforward` ou ses classes sœurs, ou utilisé SysId pour choisir PID gains `<gain> pour vous, vous êtes déjà familiarisé avec le contrôle basé sur un modèle ! Les ``kv` et ka gains `<gain> peuvent être utilisés pour décrire comment un moteur (ou un bras, ou une transmission) réagira à la tension. Nous pouvons mettre ces constantes dans la notation d'espace d'état standard en utilisant le ``LinearSystem` de WPILib, ce que nous ferons dans un article ultérieur.

Vocabulaire

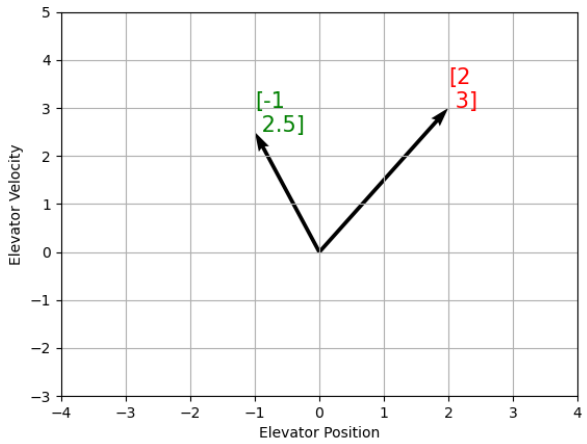
Pour le vocabulaire de base qui sera utilisé tout au long de cet article, consultez le *Glossaire*.

Introduction à l'algèbre linéaire

Pour une courte et intuitive introduction aux concepts de base de l'algèbre linéaire, nous vous recommandons les chapitres 1 à 4 de la série Essence of Linear Algebra par 3Blue1Brown (Vecteurs, qu'est-ce même ?, Combinaisons linéaires, couverture, vecteurs de base, transformations linéaires et matrices, et multiplication de matrices comme composition).

Qu'est-ce que c'est que l'espace-état ?

Recall that 2D space has two axes : x and y. We represent locations within this space as a pair of numbers packaged in a vector, and each coordinate is a measure of how far to move along the corresponding axis. State-space is a *Cartesian coordinate system* with an axis for each state variable, and we represent locations within it the same way we do for 2D space : with a list of numbers in a vector. Each element in the vector corresponds to a state of the system. This example shows two example state vectors in the state-space of an elevator model with the states $[\text{position}, \text{velocity}]$:



Dans cette image, les vecteurs qui représentent les états dans l'espace-état sont des flèches. À partir de maintenant ces vecteurs seront représentés simplement par un point à la pointe du vecteur, mais souvenez-vous que le reste du vecteur est toujours là.

En plus de l'état, les *entrées* et *sorties* sont representées aussi comme des vecteurs. Puisque la correspondance des états actuels et les entrées au changement de l'état est un système d'équations, c'est naturel d'écrire cette relation sous forme de matrice. Cette équation matricielle peut être écrite dans notation d'espace-état.

Qu'est-ce que c'est que la notation espace-état ?

La notation espace-état est un ensemble d'équations matricielles qui décrit comment un système évoluera dans le temps. Ces équations relient le changement de l'état $\dot{\mathbf{x}}$, et la sortie $\mathbf{y}$, à des combinaisons linéaires du vecteur d'état actuel $\mathbf{x}$ et le vecteur d'entrée $\mathbf{u}$.

Le contrôle de l'espace-état peut traiter des systèmes en temps continu ou en temps discret. Dans le cas du temps continu, le taux de variation de l'état du système $\mathbf{x}$ est exprimé comme une combinaison linéaire de l'état courant $\mathbf{x}$ et de l'entrée $\mathbf{u}$.

En revanche, les systèmes à temps discret expriment l'état du système à notre prochain pas de temps $\mathbf{x}_{k+1}$ basé sur l'état actuel $\mathbf{x}_k$ et l'entrée $\mathbf{u}_k$, où k est le pas de temps actuel et $k+1$ est le prochain pas de temps.

Dans les deux formes de temps continu et discret, le vecteur de sortie **y** est exprimé comme une combinaison linéaire de l'état actuel et de l'entrée. Dans de nombreux cas, la sortie est un sous-ensemble de l'état du système et n'a aucune contribution de l'entrée actuelle.

Lors de la modélisation de systèmes, nous dérivons d'abord la représentation en temps continu parce que les équations de mouvement sont naturellement écrites comme le taux de changement de l'état d'un système comme une combinaison linéaire de son état actuel et des entrées. Nous convertissons cette représentation en temps discret sur le robot parce que nous mettons à jour le système par intervalles de temps discrets au lieu de le faire en continu.

Les deux ensembles d'équations suivantes constituent la forme standard de notation espace-état en temps continu et en temps discret :

$$\text{Continuous} : \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$$\text{Discrete} : \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$
$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k$$

A	system matrix	**x**	state vector
B	input matrix	**u**	input vector
C	output matrix	**y**	output vector
D	feedthrough matrix		

Un système d'espace d'état à temps continu peut être converti en un système à temps discret grâce à un processus appelé discrétisation.

Note : Dans la forme à temps discret, l'état du système est maintenu constant entre les mises à jour. Cela signifie que nous ne pouvons réagir aux perturbations aussi rapidement que notre estimation d'état est mise à jour. Mettre à jour notre estimation plus rapidement peut aider à améliorer les performances, jusqu'à un certain point. La classe `Notifier` de WPILib peut être utilisée si des mises à jour plus rapides que la boucle principale du robot sont souhaitées.

Note : Bien que les matrices en temps continu et en temps discret A, B, C et D d'un système portent les mêmes noms, elles ne sont pas équivalentes. Les matrices à temps continu décrivent le taux de changement de l'état, **x**, tandis que les matrices à temps discret décrivent l'état du système au prochain intervalle de temps en fonction de l'état actuel et de l'entrée.

Important : LinearSystem de WPILib prend des matrices système en temps continu et les convertit en interne en forme de temps discret si nécessaire.

State-space Notation Example : Flywheel from Kv and Ka

Recall that we can model the motion of a flywheel connected to a brushed DC motor with the equation $V = K_v \cdot v + K_a \cdot a$, where V is voltage output, v is the flywheel's angular velocity and a is its angular acceleration. This equation can be rewritten as $a = \frac{V - K_v \cdot v}{K_a}$, or $a = \frac{-K_v}{K_a} \cdot v + \frac{1}{K_a} \cdot V$. Notice anything familiar ? This equation relates the angular acceleration of the flywheel to its angular velocity and the voltage applied.

Nous pouvons convertir cette équation en notation d'espace d'états. Nous pouvons créer un système avec un état (vitesse), une *entrée* (tension), et une sortie (vitesse). En rappelant que la première dérivée de la vitesse est l'accélération, nous pouvons écrire notre équation comme suit, en remplaçant la vitesse par **x**, l'accélération par $\dot{\mathbf{x}}$, et la tension **V** par **u** :

$$\dot{\mathbf{x}} = \left[\frac{-K_v}{K_a}\right]\mathbf{x} + \left[\frac{1}{K_a}\right]\mathbf{u}$$

The output and state are the same, so the output equation is the following :

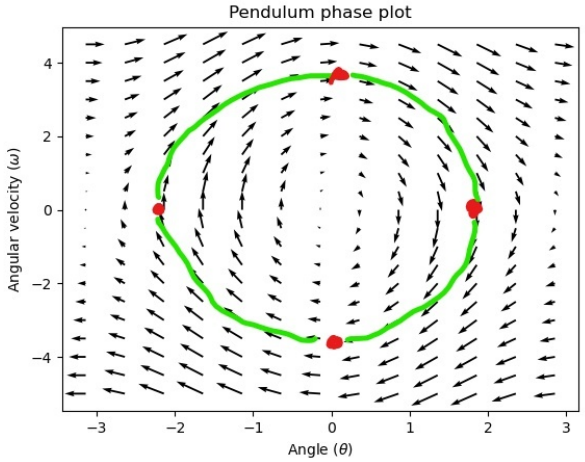
$$\mathbf{y} = [1]\,\mathbf{x} + [0]\,\mathbf{u}$$

That's it ! That's the state-space model of a system for which we have the K_v and K_a constants. This same math is used in system identification to model flywheels and drivetrain velocity systems.

Visualisation des réponses de l'espace d'état : portrait de phase

A *phase portrait* can help give a visual intuition for the response of a system in state-space. The vectors on the graph have their roots at some point **x** in state-space, and point in the direction of $\dot{\mathbf{x}}$, the direction that the system will evolve over time. This example shows a model of a pendulum with the states of angle and angular velocity.

Pour tracer une trajectoire potentielle qu'un système pourrait emprunter à travers l'espace d'états, choisissez un point de départ et suivez les flèches. Dans cet exemple, nous pourrions commencer par $[-2, 0]$. À partir de là, la vitesse augmente à mesure que nous nous balançons verticalement et commence à diminuer jusqu'à ce que nous atteignions l'extrême opposé du balancement. Ce cycle de rotation autour de l'origine se répète indéfiniment.



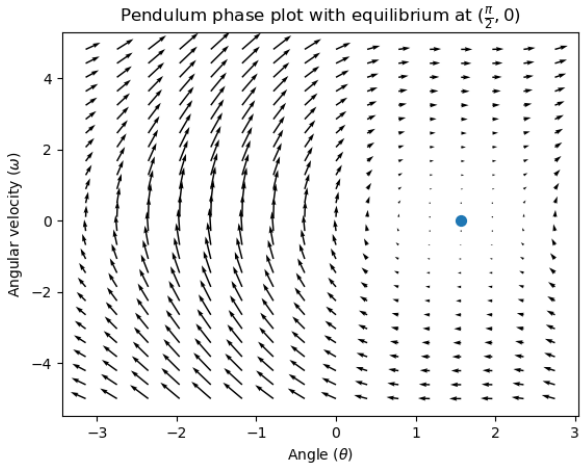
Notez que près des bords du portrait de phase, l'axe X s'enroule comme une rotation de π radians dans le sens antihoraire et une rotation de π radians dans le sens horaire se terminera au même point.

Pour en savoir plus sur les équations différentielles et les portraits de phase, voir la vidéo sur les équations différentielles de 3Blue1Brown – ils font un excellent travail d'animation de l'espace de phase du pendule vers 15 :30 minutes après le début du vidéo.

Visualiser le Feedforward

Ce portrait de phase montre les réponses « en boucle ouverte » du système - c'est-à-dire comment il réagira si nous laissons l'état évoluer naturellement. Si nous voulons, par exemple, équilibrer le pendule horizontal (à $(\frac{\pi}{2}, 0)$ dans l'espace d'états), nous aurions besoin d'appliquer une commande d'entrée pour contrer la tendance en boucle ouverte du pendule à basculer vers le bas. C'est ce que la commande prédictive essaie de faire - faire en sorte que notre portrait de phase ait un équilibre à la position de *référence* (ou point de consigne) dans l'espace d'états.

Looking at our phase portrait from before, we can see that at $(\frac{\pi}{2}, 0)$ in state space, gravity is pulling the pendulum down with some *torque* T, and producing some downward angular acceleration with magnitude $\frac{\tau}{I}$, where I is angular *moment of inertia* of the pendulum. If we want to create an equilibrium at our *reference* of $(\frac{\pi}{2}, 0)$, we would need to apply an *input* can counteract the system's natural tendency to swing downward. The goal here is to solve the equation $\mathbf{0} = \mathbf{Ax} + \mathbf{Bu}$ for $\mathbf{u}$. Below is shown a phase portrait where we apply a constant *input* that opposes the force of gravity at $(\frac{\pi}{2}, 0)$:



Commande de rétroaction

Dans le cas d'un moteur à courant continu, avec juste un modèle mathématique et une connaissance de tous les états actuels du système (c'est-à-dire la vitesse angulaire), nous pouvons prédire tous les états futurs compte tenu des futures entrées de tension. Mais si le système est perturbé d'une manière qui n'est pas modélisée par nos équations, comme une charge ou un frottement inattendu, la vitesse angulaire du moteur s'écartera du modèle au fil du temps. Pour lutter contre cela, nous pouvons donner au moteur des commandes correctives à l'aide d'un contrôleur de rétroaction.

Un contrôleur PID est une forme de commande de rétroaction. Le contrôle de l'espace d'états utilise souvent la loi de commande suivante, où $\mathbf{K}$ est la matrice de *gain* d'une commande, $\mathbf{r}$ est l'état de *référence*, et $\mathbf{x}$ est l'état actuel dans l'espace d'états. La différence entre ces

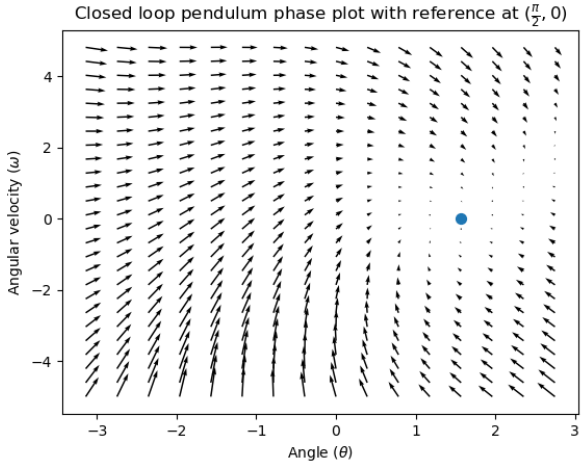
deux vecteurs, $\mathbf{r} - \mathbf{x}$, correspond à l'erreur.

$$\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$$

Cette loi de commande correspond à un contrôleur proportionnel pour chaque état de notre système. Les contrôleurs proportionnels créent des « ressorts » définis par logiciel qui tirent l'état de notre système vers notre état de référence dans l'espace d'états. Dans le cas où le système commandé a des états de position et de vitesse, la loi de commande ci-dessus se comportera comme un contrôleur PD, qui essaie également de ramener l'erreur de position et de vitesse à zéro.

Montrons un exemple de cette loi de commande en action. Nous utiliserons le système de pendule décrit plus haut, où le pendule oscillant encercle l'origine dans l'espace d'états. Le cas où **K** est la matrice nulle (une matrice dont tous les éléments sont des zéros) équivaut à choisir des gains P et D nuls - aucune *entrée* de commande serait appliquée, et le diagramme de phase serait identique à celui présenté ci-dessus.

Pour ajouter une rétroaction, nous choisissons arbitrairement un **K** de [2, 2], où notre *entrée* du pendule est une accélération angulaire. Ce K signifie que pour chaque *erreur* de position de 1 radian, l'accélération angulaire serait de 2 radians par seconde au carré ; en d'autres mots, nous accélérons de 2 radians par seconde au carré pour chaque radian par seconde term :*d'erreur*. Essayez de suivre une flèche en provenance de quelque part dans l'espace d'états vers l'intérieur - quelles que soient les conditions initiales, l'état convergera vers la *référence* plutôt que de tourner sans fin comme ce serait le cas avec une commande avec rétroaction pure.



Mais comment choisir une matrice de *gain* K optimale pour notre système ? Alors que nous pouvons choisir manuellement les *gains* et simuler la réponse du système ou l'ajuster sur le robot comme un contrôleur PID, la théorie de commande moderne a une meilleure réponse : le régulateur linéaire-quadratique (LQR).

Le régulateur linéaire-quadratique

Comme la commande basée sur un modèle signifie que nous pouvons prédire les états futurs d'un système étant données une condition initiale et des entrées de commande futures, nous pouvons choisir une matrice de *gain* math :*mathbf {K}* mathématiquement optimale. À cette fin, nous devons d'abord définir à quoi correspondrait une « bonne » ou « mauvaise » matrice **K**. Pour ce faire, nous additionnons dans le temps le carré de l'erreur et de l'entrée de commande, ceci nous donne un nombre représentant à quel point notre loi de contrôle sera « mauvaise ». Si nous minimisons cette somme, nous arrivons à la loi de commande optimale recherchée.

LQR : Définition

Linear-Quadratic Regulators work by finding a *control law* that minimizes the following cost function, which weights the sum of *error* and *control effort* over time, subject to the linear *system* dynamics $\mathbf{x_{k+1}} = \mathbf{Ax_k} + \mathbf{Bu_k}$.

$$J = \sum_{k=0}^{\infty} \left(\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right)$$

The *control law* that minimizes **J** can be written as $\mathbf{u} = \mathbf{K}(\mathbf{r_k} - \mathbf{x_k})$, where $r_k - x_k$ is the *error*.

Note : Les matrices **Q** et **R** d'un modèle n'ont pas besoin de discrétisation, mais les matrices **K** calculées pour les *systèmes* en temps continu et en temps discret seront différentes.

LQR : réglage

Tout comme les contrôleurs PID peuvent être réglés en ajustant leurs gains, nous voulons également changer la façon dont notre loi de commande équilibre notre erreur et notre entrée. Par exemple, un vaisseau spatial peut vouloir minimiser le carburant qu'il dépense pour atteindre une référence donnée, tandis qu'un bras robotique à grande vitesse peut avoir besoin de réagir rapidement aux perturbations.

Nous pouvons pondérer l'erreur et contrôler l'effort dans notre modèle LQR avec les matrices **Q** et **R**. Dans notre fonction de coût (qui décrit la « mauvaise » performance de notre loi de commande), **Q** et **R** pondèrent notre erreur et les entrées de commande l'une par rapport à l'autre. Dans l'exemple de vaisseau spatial ci-dessus, nous pourrions utiliser une matrice **Q** avec des valeurs relativement faibles pour montrer que nous ne voulons pas pénaliser fortement l'erreur, tandis que notre **R** pourrait être élevée pour montrer que dépenser du carburant n'est pas souhaitable.

Avec WPILib, la classe LQR prend un vecteur d'excursions d'état maximum souhaitées et d'efforts de contrôle et les convertit en interne en matrices Q et R complètes avec la règle de Bryson. Nous utilisons souvent des minuscules **q** et **r** pour faire référence à ces vecteurs, et **Q** et **R** pour faire référence aux matrices.

Increasing the **q** elements would make the LQR less heavily weight large errors, and the resulting *control law* will behave more conservatively. This has a similar effect to penalizing *control effort* more heavily by decreasing **r**'s elements.

Similarly, decreasing the **q** elements would make the LQR penalize large errors more heavily, and the resulting *control law* will behave more aggressively. This has a similar effect to penalizing *control effort* less heavily by increasing **r** elements.

Par exemple, nous pourrions utiliser les Q et R suivants pour un système d'élévateur avec des états de position et de vitesse.

JAVA

```
// Example system -- must be changed to match your robot.
LinearSystem<N2, N1, N1> elevatorSystem = LinearSystemId.identifyPositionSystem(5, 0.
↪5);
LinearQuadraticRegulator<N2, N1, N1> controller = new␣
↪LinearQuadraticRegulator(elevatorSystem,
      // q's elements
      VecBuilder.fill(0.02, 0.4),
      // r's elements
      VecBuilder.fill(12.0),
      // our dt
      0.020);
```

C++

```
// Example system -- must be changed to match your robot.
   LinearSystem<2, 1, 1> elevatorSystem =␣
↪frc::LinearSystemId::IdentifyVelocitySystem(5, 0.5);
   LinearQuadraticRegulator<2, 1> controller{
      elevatorSystem,
      // q's elements
      {0.02, 0.4},
      // r's elements
      {12.0},
      // our dt
      0.020_s};
```

PYTHON

```
from wpimath.controller import LinearQuadraticRegulator_2_1
from wpimath.system.plant import LinearSystemId


# Example system -- must be changed to match your robot.
elevatorSystem = LinearSystemId.identifyPositionSystemMeters(5, 0.5)
controller = LinearQuadraticRegulator_2_1(
   elevatorSystem,
   # q's elements
   (0.02, 0.4),
   # r's elements
   (12.0,),
   # our dt
   0.020,
)
```

LQR : exemple d'application

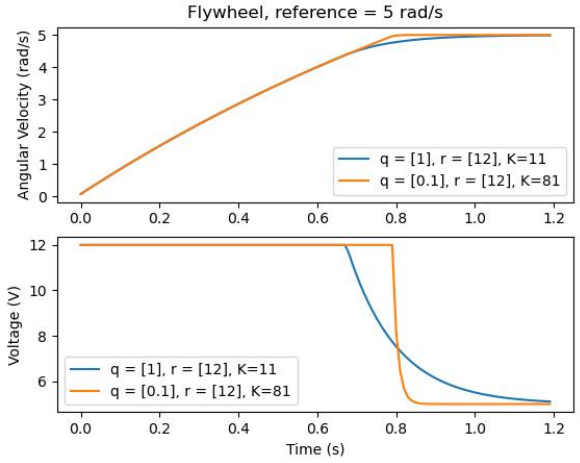
Let's apply a Linear-Quadratic Regulator to a real-world example. Say we have a flywheel velocity system determined through system identification to have $K_v = 1 \frac{\text{volts}}{\text{radian per second}}$ and $K_a = 1.5 \frac{\text{volts}}{\text{radian per second squared}}$. Using the flywheel example above, we have the following linear *system* :

$$\mathbf{x} = \left[\frac{-K_v}{K_a}\right] v + \left[\frac{1}{K_a}\right] V$$

Nous choisissons arbitrairement une excursion d'état désirée (erreur maximale) de $q = [0.1$ rad/sec$]$, et un **r** de $[12$ volts$]$. Après discrétisation avec un pas de temps de 20ms, on trouve un *gain* de $\mathbf{K} = 81$. Ce *gain* K agit comme la composante proportionnelle d'une boucle PID sur la vitesse du volant d'inertie.

Let's adjust **q** and **r**. We know that increasing the q elements or decreasing the **r** elements we use to create **Q** and **R** would make our controller more heavily penalize *control effort*, analogous to trying to driving a car more conservatively to improve fuel economy. In fact, if we increase our *error* tolerance q from 0.1 to 1.0, our *gain* matrix **K** drops from ~81 to ~11. Similarly, decreasing our maximum voltage r from 12.0 to 1.2 decreases **K**.

Le graphique suivant montre la vitesse angulaire du volant d'inertie et la tension appliquée au fil du temps avec deux *gain*s. Nous pouvons voir comment un *gain* plus élevé permettra au système d'atteindre la référence plus rapidement (à t = 0.8 seconde), tout en maintenant notre moteur saturé à 12V plus longtemps. C'est exactement la même chose que d'augmenter le gain P d'un régulateur PID d'un facteur ~ 8x.



LQR et compensation de latence de mesure

Souvent, nos capteurs ont un retard associé à leurs mesures. Par exemple, le contrôleur de moteur SPARK MAX sur CAN peut avoir jusqu'à 30 ms de retard associé aux mesures de vitesse.

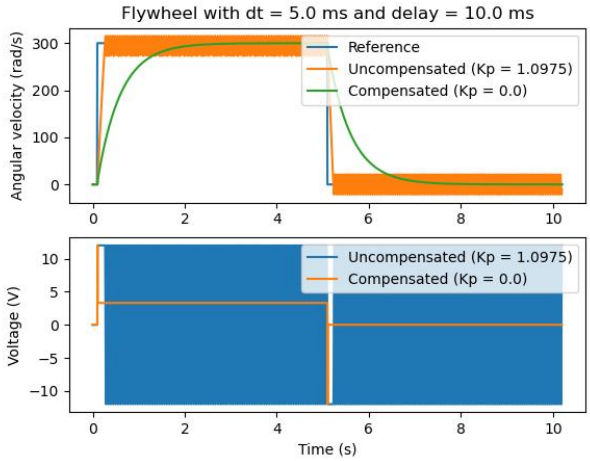
Ce décalage signifie que notre contrôleur de rétroaction générera des commandes de tension basées sur des estimations d'état du passé. Cela a souvent pour effet d'introduire de l'instabilité et des oscillations dans notre système, comme le montre le graphique ci-dessous.

Cependant, nous pouvons modéliser notre contrôleur pour contrôler où le *state* du système est retardé dans le futur. Cela réduira la matrice *gain* du LQR **K**, échangeant les performances

du contrôleur contre la stabilité. La formule ci-dessous, qui ajuste la matrice *gain* pour tenir compte du retard, est également utilisée dans l'identification du système.

$$\mathbf{K_{compensated}} = \mathbf{K} \cdot (\mathbf{A} - \mathbf{BK})^{\text{delay}/dt}$$

Multiplier $\mathbf{K}$ par $\mathbf{A} - \mathbf{BK}$ avance essentiellement les gains d'un pas de temps. Dans ce cas, on multiplie par $(\mathbf{A} - \mathbf{BK})^{\text{delay}/dt}$ pour avancer les gains par le retard de la mesure.



Note : Cela peut avoir pour effet de réduire $\mathbf{K}$ à zéro, désactivant ainsi la commande de rétroaction.

Note : Le contrôleur de moteur SPARK MAX utilise un filtre à réponse impulsionnelle finie ou RIF à 40 prises avec un retard de 19.5 ms, et les trames d'état sont envoyées par défaut tous les 20ms.

Le code ci-dessous montre comment régler le gain K du contrôleur LQR pour les délais d'entrée du capteur :

JAVA

```
// Adjust our LQR's controller for 25 ms of sensor input delay. We
// provide the linear system, discretization timestep, and the sensor
// input delay as arguments.
controller.latencyCompensate(elevatorSystem, 0.02, 0.025);
```

C++

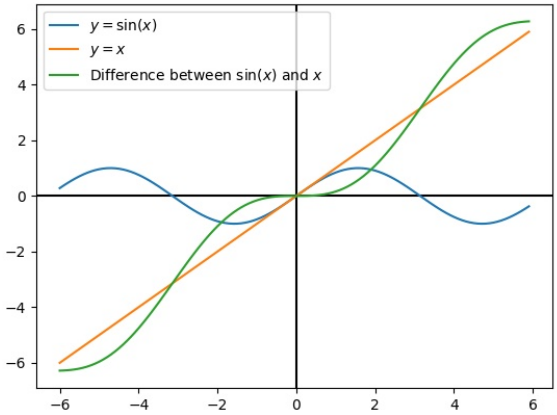
```
// Adjust our LQR's controller for 25 ms of sensor input delay. We
// provide the linear system, discretization timestep, and the sensor
// input delay as arguments.
controller.LatencyCompensate(elevatorSystem, 20_ms, 25_ms);
```

PYTHON

```
# Adjust our LQR's controller for 25 ms of sensor input delay. We
# provide the linear system, discretization timestep, and the sensor
# input delay as arguments.
controller.latencyCompensate(elevatorSystem, 0.020, 0.025)
```

Linéarisation

La linéarisation est un outil utilisé pour approximer les fonctions non linéaires et les systèmes d'espace d'états en faisant usage de systèmes linéaires. Dans l'espace bidimensionnel, les fonctions linéaires sont des lignes droites tandis que les fonctions non linéaires sont courbes. Un exemple courant de fonction non linéaire et de son approximation linéaire correspondante est : $y = \sin x$. Cette fonction peut être approximée par $y = x$ près de zéro. Cette approximation est précise lorsqu'elle est proche de $x = 0$, mais perd de la précision lorsque nous nous éloignons du point de linéarisation. Par exemple, l'approximation $\sin x \approx x$ est précise à 0.02 près de 0.5 radians de $y = 0$, mais perd rapidement sa précision au-delà. Dans l'image suivante, nous voyons $y = \sin x$, $y = x$ et la différence entre l'approximation et la vraie valeur de $\sin x$ à x.



Nous pouvons également linéariser des systèmes d'espace d'états en faisant appel à une *dynamique* non-linéaire. Pour ce faire, choisissons un point $\mathbf{x}$ dans l'espace d'états et en utilisons-le comme entrée de nos fonctions non linéaires. Comme dans l'exemple ci-dessus, cela fonctionne bien pour les états proches du point sur lequel le système a été linéarisé, mais peut rapidement diverger davantage de cet état.

32.8.2 Description de l'espace-état d'un controleur

Note : Avant de poursuivre la lecture de ce didacticiel, il est recommandé aux lecteurs d'avoir pris connaissance du document *Introduction au Contrôle de l'Espace-État*.

L'objectif de ce didacticiel est de fournir des instructions de « bout à bout » sur la mise en œuvre dans l'espace d'état d'un contrôleur pour un volant d'inertie. En parcourant ce didacticiel, les lecteurs apprendront à :

1. Create an accurate state-space model of a flywheel using *system identification* or *CAD* software.
2. Implémenter un filtre de Kalman pour filtrer les mesures de vitesse de l'encodeur sans décalage.
3. implémenter une commande *LQR* de rétroaction qui, lorsqu'elle est combinée avec le modèle de commande prédictive, générera les tensions *d'entrée* afin de conduire le volant d'inertie à la *référence*.

Ce tutoriel se veut être simple pour faciliter la tâche aux équipes n'ayant pas beaucoup d'expertise en programmation. Bien que la bibliothèque WPILib offre une flexibilité significative dans la manière dont ses fonctionnalités de contrôle dans l'espace d'état sont implémentées, parcourir attentivement l'implémentation décrite dans ce didacticiel devrait fournir aux équipes une structure de base qui peut être réutilisée pour une variété de systèmes espace-état.

The full example is available in the state-space flywheel (Java/C++/Python) and state-space flywheel system identification (Java/C++/Python) example projects.

Pourquoi utiliser le contrôle espace-état ?

Étant donné que la commande de l'espace-état se concentre sur la création d'un modèle précis de notre système, nous pouvons adéquatement prédire comment notre *modèle* va répondre aux *entrées* des commandes. Cela nous permet de simuler nos mécanismes sans accès à un robot physique, ainsi que de choisir facilement les *gains* que nous savons parfaitement fonctionnels. Avoir un modèle nous permet également de créer des filtres sans retard, comme les filtres de Kalman, pour filtrer de manière optimale les signaux obtenus à partir des capteurs.

Modélisation de notre volant d'inertie

Rappelons que les systèmes continus espace-état sont modélisés à l'aide du système d'équations suivant :

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

Where *x-dot* is the rate of change of the *system*'s *state*, $\mathbf{x}$ is the system's current state, $\mathbf{u}$ is the *input* to the system, and $\mathbf{y}$ is the system's *output*.

Utilisons ce système d'équations pour modéliser notre volant d'inertie de deux manières différentes. Nous allons d'abord le modéliser en utilisant identification du système à l'aide de la suite d'outils SysId, puis le modéliser en fonction du moment d'inertie du moteur et du volant.

The first step of building up our state-space system is picking our system's states. We can pick anything we want as a state – we could pick completely unrelated states if we wanted – but it helps to pick states that are important. We can include *hidden states* in our state (such as elevator velocity if we were only able to measure its position) and let our Kalman Filter estimate their values. Remember that the states we choose will be driven towards their respective *references* by the feedback controller (typically the *Linear-Quadratic Regulator* since it's optimal).

For our flywheel, we care only about one state : its velocity. While we could chose to also model its acceleration, the inclusion of this state isn't necessary for our system.

Ensuite, nous identifions les *entrées* dans notre système. Les entréss peuvent être considérées comme des paramètres que nous pouvons mettre « dans » notre système pour en changer l'état. Dans le cas du volant d'inertie (et de nombreux autres mécanismes à simple articulation en FRC®), nous n'avons qu'une seule entrée : la tension appliquée sur le moteur. En choisissant la tension comme entrée (plutôt qu'un paramètre comme le cycle de travail du moteur), nous pouvons compenser la réduction de la tension de la batterie à mesure que la charge de la batterie augmente.

Un système espace-état en temps continu écrit *x-point*, ou le taux de variation instantané de l'état du *système*, comme proportionnel à l'état actuel et ses *entrées*. Puisque notre état est la vitesse angulaire, $\mathbf{x}$ sera l'accélération angulaire du volant d'inertie.

Ensuite, nous allons modéliser notre volant d'inertie comme un système espace-état en temps continu. Le `LinearSystem` de WPILib convertira ce système en temps discret selon un mécanisme interne. Relisez *la notation espace-état* pour en savoir d'avantage sur les systèmes en temps continu et en temps discret.

Modélisation par identification du système

To rewrite this in state-space notation using *system identification*, we recall from the flywheel *state-space notation example*, where we rewrote the following equation in terms of $\mathbf{a}$.

$$V = kV \cdot \mathbf{v} + kA \cdot \mathbf{a}$$
$$\mathbf{a} = \mathbf{v} = \left[\frac{-kV}{kA}\right] v + \left[\frac{1}{kA}\right] V$$

Où $\mathbf{v}$ est la vitesse du volant d'inertie, $\mathbf{a}$ et $\mathbf{v}$ sont l'accélération de la roue d'inertie, et V est la tension. En reformulant ce qui précède selon la représentation conventionnelle $\mathbf{x}$ pour le vecteur d'état et $\mathbf{u}$ pour le vecteur d'entrée, nous trouvons :

$$\mathbf{x} = \left[\frac{-kV}{kA}\right] \mathbf{x} + \left[\frac{1}{kA}\right] \mathbf{u}$$

La deuxième partie de la notation espace-état relie les termes actuels du système *état* et *entrées* à la sortie. Dans le cas d'un volant d'inertie, notre vecteur de sortie $\mathbf{y}$ (ou toute autre grandeur que nos capteurs peuvent mesurer) est la vitesse de notre volant d'inertie, qui se trouve également être un élément de notre vecteur d'état $\mathbf{x}$. Par conséquent, notre matrice de sortie est $\mathbf{C} = \begin{bmatrix}1\end{bmatrix}$, et notre matrice du système de la traversée est $\mathbf{D} = \begin{bmatrix}0\end{bmatrix}$. En reformulant ce qui précède selon la notation espace-état en temps continu conduit au résultat suivant.

$$\mathbf{x} = \begin{bmatrix}\frac{-kV}{kA}\end{bmatrix}\mathbf{x} + \begin{bmatrix}\frac{1}{kA}\end{bmatrix}\mathbf{u}$$
$$\mathbf{y} = \begin{bmatrix}1\end{bmatrix}\mathbf{x} + \begin{bmatrix}0\end{bmatrix}\mathbf{u}$$

La classe `LinearSystem` contient des méthodes permettant de créer facilement des systèmes espace-état identifiés à l'aide de la méthode d'identification de système. Cet exemple montre un modèle volant d'inertie avec un kV de 0.023 et un kA de 0.001 :

Java

```
  // Volts per (radian per second)
  private static final double kFlywheelKv = 0.023;

  // Volts per (radian per second squared)
  private static final double kFlywheelKa = 0.001;

  // The plant holds a state-space model of our flywheel. This system has the␣
↪following properties:
  //
  // States: [velocity], in radians per second.
  // Inputs (what we can "put in"): [voltage], in volts.
  // Outputs (what we can measure): [velocity], in radians per second.
  //
  // The Kv and Ka constants are found using the FRC Characterization toolsuite.
  private final LinearSystem<N1, N1, N1> m_flywheelPlant =
      LinearSystemId.identifyVelocitySystem(kFlywheelKv, kFlywheelKa);
```

C++

```
#include <frc/system/plant/LinearSystemId.h>
```

```
  // Volts per (radian per second)
  static constexpr auto kFlywheelKv = 0.023_V / 1_rad_per_s;

  // Volts per (radian per second squared)
  static constexpr auto kFlywheelKa = 0.001_V / 1_rad_per_s_sq;

  // The plant holds a state-space model of our flywheel. This system has the
  // following properties:
  //
  // States: [velocity], in radians per second.
  // Inputs (what we can "put in"): [voltage], in volts.
  // Outputs (what we can measure): [velocity], in radians per second.
  //
  // The Kv and Ka constants are found using the FRC Characterization toolsuite.
  frc::LinearSystem<1, 1, 1> m_flywheelPlant =
      frc::LinearSystemId::IdentifyVelocitySystem<units::radian>(kFlywheelKv,
                                                                 kFlywheelKa);
```

Python

```
# Volts per (radian per second)
kFlywheelKv = 0.023

# Volts per (radian per second squared)
kFlywheelKa = 0.001
```

```
        # The plant holds a state-space model of our flywheel. This system has the following properties:
        #
        # States: [velocity], in radians per second.
        # Inputs (what we can "put in"): [voltage], in volts.
        # Outputs (what we can measure): [velocity], in radians per second.
        #
        # The Kv and Ka constants are found using the FRC Characterization toolsuite.
        self.flywheelPlant = (
            wpimath.system.plant.LinearSystemId.identifyVelocitySystemRadians(
                kFlywheelKv, kFlywheelKa
            )
        )
```

Modélisation à l'aide du moment d'inertie du volant d'inertie et de l'engrenage.

A flywheel can also be modeled without access to a physical robot, using information about the motors, gearing and flywheel's *moment of inertia*. A full derivation of this model is presented in Section 12.3 of Controls Engineering in FRC.

The `LinearSystem` class contains methods to easily create a model of a flywheel from the flywheel's motors, gearing and *moment of inertia*. The moment of inertia can be calculated using *CAD* software or using physics. The examples used here are detailed in the flywheel example project (Java/C++/Python).

Note : Pour les classes espace-état de la WPILib, l'engrenage est modélisé par une fonction de sortie sur entrée – c'est-à-dire que si le volant d'inertie tourne plus lentement que les moteurs, ce nombre devrait être supérieur à un.

Note : La classe C++ LinearSystem utilise *librairie d'unités C++* pour prévenir les mélanges des unités et se conformer à l'analyse dimensionnelle.

Java

```
  private static final double kFlywheelMomentOfInertia = 0.00032; // kg * m^2

  // Reduction between motors and encoder, as output over input. If the flywheel␣
↪spins slower than
  // the motors, this number should be greater than one.
  private static final double kFlywheelGearing = 1.0;

  // The plant holds a state-space model of our flywheel. This system has the␣
↪following properties:
  //
  // States: [velocity], in radians per second.
  // Inputs (what we can "put in"): [voltage], in volts.
  // Outputs (what we can measure): [velocity], in radians per second.
  private final LinearSystem<N1, N1, N1> m_flywheelPlant =
      LinearSystemId.createFlywheelSystem(
          DCMotor.getNEO(2), kFlywheelMomentOfInertia, kFlywheelGearing);
```

C++

```
#include <frc/system/plant/LinearSystemId.h>
```

```
#include <frc/system/plant/LinearSystemId.h>
  static constexpr units::kilogram_square_meter_t kFlywheelMomentOfInertia =
      0.00032_kg_sq_m;

  // Reduction between motors and encoder, as output over input. If the flywheel
  // spins slower than the motors, this number should be greater than one.
  static constexpr double kFlywheelGearing = 1.0;

  // The plant holds a state-space model of our flywheel. This system has the
  // following properties:
  //
  // States: [velocity], in radians per second.
  // Inputs (what we can "put in"): [voltage], in volts.
  // Outputs (what we can measure): [velocity], in radians per second.
  frc::LinearSystem<1, 1, 1> m_flywheelPlant =
      frc::LinearSystemId::FlywheelSystem(
          frc::DCMotor::NEO(2), kFlywheelMomentOfInertia, kFlywheelGearing);
```

Python

```
kFlywheelMomentOfInertia = 0.00032  # kg/m^2

# Reduction between motors and encoder, as output over input. If the flywheel spins␣
↪slower than
# the motors, this number should be greater than one.
kFlywheelGearing = 1
```

```
        # The plant holds a state-space model of our flywheel. This system has the␣
↪following properties:
```

(suite sur la page suivante)

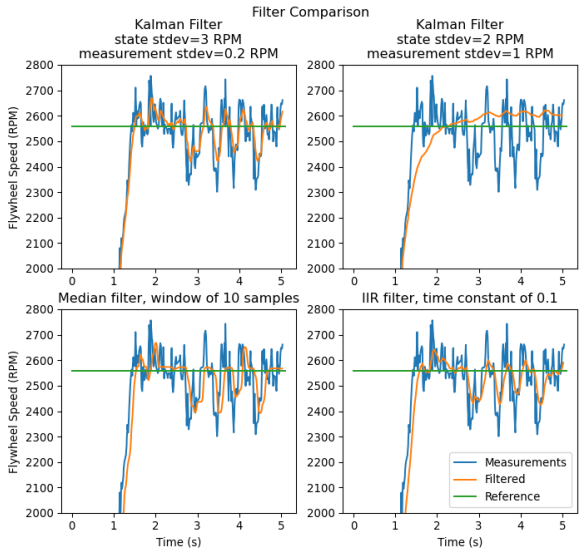
(suite de la page précédente)

```
        #
        # States: [velocity], in radians per second.
        # Inputs (what we can "put in"): [voltage], in volts.
        # Outputs (what we can measure): [velocity], in radians per second.
        self.flywheelPlant = wpimath.system.plant.LinearSystemId.flywheelSystem(
            wpimath.system.plant.DCMotor.NEO(2),
            kFlywheelMomentOfInertia,
            kFlywheelGearing,
        )
```

Filtres de Kalman : Observation de l'état du volant d'inertie

Les filtres de Kalman sont utilisés pour filtrer nos mesures de vitesse à l'aide de notre modèle espace-état pour générer une estimation de l'état $\hat{x}$. Comme notre modèle de volant est linéaire, nous pouvons utiliser un filtre de Kalman pour estimer la vitesse du volant. Le filtre de Kalman fournie par la WPILib prend un `système linéaire` ou `LinearSystem` (que nous avons trouvé ci-dessus), ainsi que des écarts-types des mesures du modèle et des capteurs. Nous pouvons ajuster le « lissage » de notre de l'état estimé en ajustant ces poids. Les déviations standard plus importants de l'état feront en sorte que le filtre « se conforme moins » à notre estimation de l'état et favorisera plus fortement les nouvelles mesures, tandis que des valeurs plus grandes de déviations standard auront un effet contraire.

Dans le cas d'un volant d'inertie, nous commençons par un écart type d'état de 3 rad/s et un écart type de mesure de 0.01 rad/s. Le choix de ces valeurs est à la discrétion de l'utilisateur - ces poids ont produit un filtre qui était tolérant à certains bruits, mais dont l'estimation de l'état a rapidement réagi aux perturbations externes pour *un* volant d'inertie - et doivent être réglés pour créer un filtre qui soit conforme à votre volant spécifique. Les tracés des états, des mesures, des entrées, des références et des sorties au fil du temps sont un excellent moyen visuel de régler les filtres de Kalman.



Le graphique ci-dessus montre deux filtres de Kalman réglés différemment, ainsi qu'un *filtre IIR simple-pôle* et un *Filtre médian*. Ces données ont été recueillies avec un tireur sur ~5 secondes, et quatre balles ont été lancées à travers le tireur (comme on le voit dans les quatre

creux de vitesse). Bien qu'il n'y ait pas de règles strictes sur le choix du bon état et des écarts-types des mesures, ils devraient en général être réglés pour se rapprocher suffisamment du modèle pour éliminer le bruit tout en réagissant rapidement aux perturbations externes.

Comme le contrôleur de rétroaction calcule l'erreur à l'aide du paramètre x-chapeau estimé par le filtre de Kalman, le contrôleur ne réagira aux perturbations qu'aussi rapidement que l'estimation d'état du filtre change. Dans le graphique ci-dessus, la courbe située en haut et à gauche (avec un écart type d'état de 3.0 et un écart type de mesure de 0.2) a produit un filtre qui réagit rapidement aux perturbations tout en rejetant le bruit, tandis que la courbe située en haut et à droite montre un filtre qui a été à peine affecté par les baisses de vitesse.

Java

```
48  // The observer fuses our encoder data and voltage inputs to reject noise.
49  private final KalmanFilter<N1, N1, N1> m_observer =
50      new KalmanFilter<>(
51          Nat.N1(),
52          Nat.N1(),
53          m_flywheelPlant,
54          VecBuilder.fill(3.0), // How accurate we think our model is
55          VecBuilder.fill(0.01), // How accurate we think our encoder
56          // data is
57          0.020);
```

C++

```
13  #include <frc/estimator/KalmanFilter.h>
```

```
48  // The observer fuses our encoder data and voltage inputs to reject noise.
49  frc::KalmanFilter<1, 1, 1> m_observer{
50      m_flywheelPlant,
51      {3.0},   // How accurate we think our model is
52      {0.01},  // How accurate we think our encoder data is
53      20_ms};
```

Python

```
48        # The observer fuses our encoder data and voltage inputs to reject noise.
49        self.observer = wpimath.estimator.KalmanFilter_1_1_1(
50            self.flywheelPlant,
51            [3],  # How accurate we think our model is
52            [0.01],  # How accurate we think our encoder data is
53            0.020,
54        )
```

Étant donné que les filtres de Kalman utilisent notre modèle espace-état dans *L'étape de prédiction*, il est important que notre modèle soit aussi précis que possible. Une façon de vérifier cela est d'enregistrer la tension d'entrée et la vitesse d'entrée d'un volant d'inertie au fil du temps, et de rejouer ces données en appelant uniquement `predict` sur le filtre de Kalman. Ensuite, les gains kV et kA (ou moment d'inertie et autres constantes) peuvent être ajustés jusqu'à ce que le modèle corresponde étroitement aux données enregistrées.

Régulateurs linéaires-quadratiques et commande prédictive inverse de procédé

Le régulateur linéaire-quadratique trouve un contrôleur de rétroaction pour amener notre *système* volant d'inertie à sa *référence*. Parce que notre volant n'a qu'un seul état, la loi de commande choisie par notre LQR sera sous la forme $\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$ où $\mathbf{K}$ est une matrice 1x1 ; en d'autres termes, la loi de commande choisie par notre LQR est simplement un contrôleur proportionnel, ou un contrôleur PID avec seulement un gain P. Ce gain est choisi par notre LQR en fonction de l'excursion d'état et des efforts de contrôle que nous lui transmettons. En savoir plus sur le réglage des contrôleurs LQR, consultez *LQR application example*.

Tout comme le `SimpleMotorFeedforward` peut être utilisé pour générer des tensions d'entrée de la commande prédictive étant donnés les constantes kS, kV, et kA, la classe `Plant Inversion Feedforward` génère *feedforward* les tensions d'entrée étant donné le système espace-état. Les commandes de tension générées par la classe `LinearSystemLoop` sont la somme des entrées des commandes prédictive et de rétroaction.

Java

```
  // A LQR uses feedback to create voltage commands.
  private final LinearQuadraticRegulator<N1, N1, N1> m_controller =
      new LinearQuadraticRegulator<>(
          m_flywheelPlant,
          VecBuilder.fill(8.0), // qelms. Velocity error tolerance, in radians per second. Decrease
          // this to more heavily penalize state excursion, or make the controller behave more
          // aggressively.
          VecBuilder.fill(12.0), // relms. Control effort (voltage) tolerance. Decrease this to more
          // heavily penalize control effort, or make the controller less aggressive. 12 is a good
          // starting point because that is the (approximate) maximum voltage of a battery.
          0.020); // Nominal time between loops. 0.020 for TimedRobot, but can be
  // lower if using notifiers.
```

C++

```
#include <frc/controller/LinearQuadraticRegulator.h>
```

```
  // A LQR uses feedback to create voltage commands.
  frc::LinearQuadraticRegulator<1, 1> m_controller{
      m_flywheelPlant,
      // qelms. Velocity error tolerance, in radians per second. Decrease this
      // to more heavily penalize state excursion, or make the controller behave
      // more aggressively.
      {8.0},
      // relms. Control effort (voltage) tolerance. Decrease this to more
      // heavily penalize control effort, or make the controller less
      // aggressive. 12 is a good starting point because that is the
      // (approximate) maximum voltage of a battery.
      {12.0},
      // Nominal time between loops. 20ms for TimedRobot, but can be lower if
```

(suite sur la page suivante)

(suite de la page précédente)

```
67      // using notifiers.
68      20_ms};
69
70  // The state-space loop combines a controller, observer, feedforward and plant
71  // for easy control.
72  frc::LinearSystemLoop<1, 1, 1> m_loop{m_flywheelPlant, m_controller,
73                                        m_observer, 12_V, 20_ms};
```

Python

```
56          # A LQR uses feedback to create voltage commands.
57          self.controller = wpimath.controller.LinearQuadraticRegulator_1_1(
58              self.flywheelPlant,
59              [8],  # qelms. Velocity error tolerance, in radians per second. Decrease
60              # this to more heavily penalize state excursion, or make the controller␣
↪behave more
61              # aggressively.
62              [12],  # relms. Control effort (voltage) tolerance. Decrease this to more
63              # heavily penalize control effort, or make the controller less aggressive.
↪ 12 is a good
64              # starting point because that is the (approximate) maximum voltage of a␣
↪battery.
65              0.020,  # Nominal time between loops. 0.020 for TimedRobot, but can be␣
↪lower if using notifiers.
66          )
```

En rassemblant le tout : LinearSystemLoop

LinearSystemLoop combine notre système, notre contrôleur et notre observateur que nous avons créés plus tôt. Le constructeur montré instancie également un `PlantInversionFeedforward`.

Java

```
72  // The state-space loop combines a controller, observer, feedforward and plant for␣
↪easy control.
73  private final LinearSystemLoop<N1, N1, N1> m_loop =
74      new LinearSystemLoop<>(m_flywheelPlant, m_controller, m_observer, 12.0, 0.020);
```

C++

```
15  #include <frc/system/LinearSystemLoop.h>
```

```
71  // The state-space loop combines a controller, observer, feedforward and plant
72  // for easy control.
73  frc::LinearSystemLoop<1, 1, 1> m_loop{m_flywheelPlant, m_controller,
74                                        m_observer, 12_V, 20_ms};
```

Python

```
        # The state-space loop combines a controller, observer, feedforward and plant for easy control.
        self.loop = wpimath.system.LinearSystemLoop_1_1_1(
            self.flywheelPlant, self.controller, self.observer, 12.0, 0.020
        )
```

Une fois que nous avons obtenu notre `LinearSystemLoop`, la dernière chose à faire est de l'exécuter. Pour ce faire, nous mettrons périodiquement à jour notre filtre de Kalman avec nos nouvelles mesures de vitesse d'encodeur et lui appliquerons de nouvelles commandes de tension. Dans cette optique, nous avons d'abord défini la *référence*, puis `corrigé` avec la vitesse actuelle du volant d'inertie, `prédit` le filtre de Kalman dans le prochain pas de temps, et appliqué les entrées générées à l'aide de `getU`.

Java

```
  @Override
  public void teleopPeriodic() {
    // Sets the target speed of our flywheel. This is similar to setting the setpoint of a
    // PID controller.
    if (m_joystick.getTriggerPressed()) {
      // We just pressed the trigger, so let's set our next reference
      m_loop.setNextR(VecBuilder.fill(kSpinupRadPerSec));
    } else if (m_joystick.getTriggerReleased()) {
      // We just released the trigger, so let's spin down
      m_loop.setNextR(VecBuilder.fill(0.0));
    }

    // Correct our Kalman filter's state vector estimate with encoder data.
    m_loop.correct(VecBuilder.fill(m_encoder.getRate()));

    // Update our LQR to generate new voltage commands and use the voltages to predict the next
    // state with out Kalman filter.
    m_loop.predict(0.020);

    // Send the new calculated voltage to the motors.
    // voltage = duty cycle * battery voltage, so
    // duty cycle = voltage / battery voltage
    double nextVoltage = m_loop.getU(0);
    m_motor.setVoltage(nextVoltage);
  }
}
```

C++

```cpp
#include <numbers>

#include <frc/DriverStation.h>
#include <frc/Encoder.h>
#include <frc/TimedRobot.h>
#include <frc/XboxController.h>
#include <frc/controller/LinearQuadraticRegulator.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/estimator/KalmanFilter.h>
#include <frc/motorcontrol/PWMSparkMax.h>
#include <frc/system/LinearSystemLoop.h>
#include <frc/system/plant/DCMotor.h>
#include <frc/system/plant/LinearSystemId.h>
```

```cpp
  void TeleopPeriodic() override {
    // Sets the target speed of our flywheel. This is similar to setting the
    // setpoint of a PID controller.
    if (m_joystick.GetRightBumper()) {
      // We pressed the bumper, so let's set our next reference
      m_loop.SetNextR(frc::Vectord<1>{kSpinup.value()});
    } else {
      // We released the bumper, so let's spin down
      m_loop.SetNextR(frc::Vectord<1>{0.0});
    }

    // Correct our Kalman filter's state vector estimate with encoder data.
    m_loop.Correct(frc::Vectord<1>{m_encoder.GetRate()});

    // Update our LQR to generate new voltage commands and use the voltages to
    // predict the next state with out Kalman filter.
    m_loop.Predict(20_ms);

    // Send the new calculated voltage to the motors.
    // voltage = duty cycle * battery voltage, so
    // duty cycle = voltage / battery voltage
    m_motor.SetVoltage(units::volt_t{m_loop.U(0)});
  }
```

Python

```python
    def teleopPeriodic(self) -> None:
        # Sets the target speed of our flywheel. This is similar to setting the␣
↪setpoint of a
        # PID controller.
        if self.joystick.getTriggerPressed():
            # We just pressed the trigger, so let's set our next reference
            self.loop.setNextR([kSpinUpRadPerSec])

        elif self.joystick.getTriggerReleased():
            # We just released the trigger, so let's spin down
            self.loop.setNextR([0.0])

        # Correct our Kalman filter's state vector estimate with encoder data.
```

(suite sur la page suivante)

(suite de la page précédente)

```
        self.loop.correct([self.encoder.getRate()])

        # Update our LQR to generate new voltage commands and use the voltages to␣
↪predict the next
        # state with out Kalman filter.
        self.loop.predict(0.020)

        # Send the new calculated voltage to the motors.
        # voltage = duty cycle * battery voltage, so
        # duty cycle = voltage / battery voltage
        nextVoltage = self.loop.U()
        self.motor.setVoltage(nextVoltage)
```

Angle Wrap with LQR

Mechanisms with a continuous angle can have that angle wrapped by calling the code below instead of `lqr.Calculate(x, r)`.

JAVA

```
var error = lqr.getR().minus(x);
error.set(0, 0, MathUtil.angleModulus(error.get(0, 0)));
var u = lqr.getK().times(error);
```

C++

```
Eigen::Vector<double, 2> error = lqr.R() - x;
error(0) = frc::AngleModulus(units::radian_t{error(0)}).value();
Eigen::Vector<double, 2> u = lqr.K() * error;
```

PYTHON

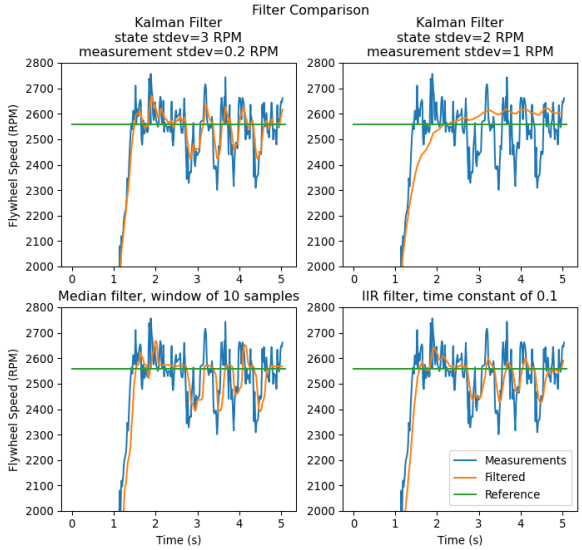
```
error = lqr.R() - x
error[0] = wpimath.angleModulus(error[0])
u = lqr.K() * error
```

32.8.3 Observateurs d'état et filtres de Kalman

Les observateurs d'état combinent des informations sur le comportement d'un système et des mesures externes pour estimer l'état du système. Un observateur commun utilisé pour les systèmes linéaires est le filtre de Kalman. Les filtres de Kalman sont avantageux par rapport aux autres *filtres* car ils fusionnent les mesures d'un ou plusieurs capteurs avec un modèle d'espace d'état du système pour estimer de manière optimale l'état d'un système.

Cette image montre les mesures de vitesse d'un volant d'inertie au fil du temps, exécutées à travers une variété de filtres différents. Notez qu'un filtre de Kalman bien réglé ne montre aucun décalage de mesure pendant la rotation du volant tout en rejetant les données bruyantes

et en réagissant rapidement aux perturbations lorsque les billes le traversent. Vous trouverez plus d'informations sur les filtres dans la :ref :` section filtres <docs/software/advanced-controls/filters/index :Filters>`.



Les fonctions gaussiennes

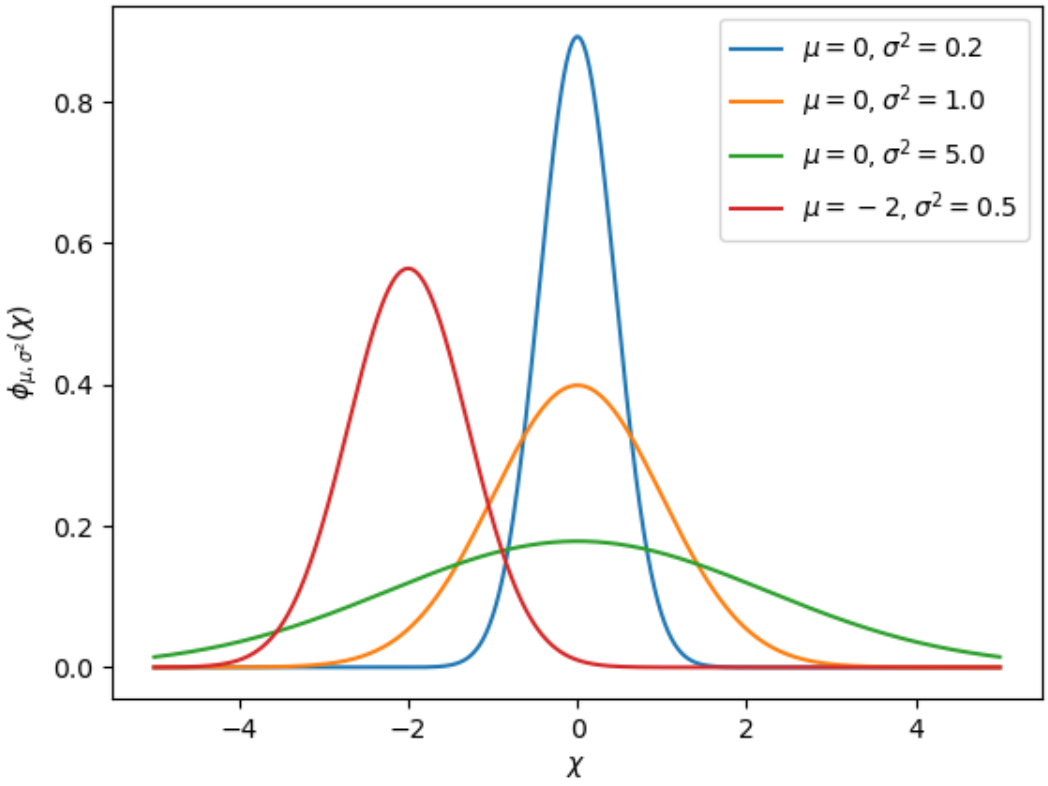
Kalman filters utilize a *Gaussian distribution* to model the noise in a process[1]. In the case of a Kalman filter, the estimated *state* of the system is the mean, while the variance is a measure of how certain (or uncertain) the filter is about the true *state*.

L'idée de variance et de covariance est au cœur de la fonction d'un filtre de Kalman. La covariance est une mesure de la manière dont deux variables aléatoires sont corrélées. Dans un système à un seul état, la matrice de covariance est simplement $\text{cov}(\mathbf{x_1}, \mathbf{x_1})$, ou une matrice contenant la variance $\text{var}(\mathbf{x_1})$ de l'état x_1. L'amplitude de cette variance est le carré de l'écart type de la fonction gaussienne décrivant l'estimation de l'état actuel. Des valeurs relativement élevées de covariance peuvent indiquer des données bruyantes, tandis que de petites covariances peuvent indiquer que le filtre est plus sûr de son estimation. Rappelez-vous que les valeurs « grandes » et « petites » de la variance ou de la covariance sont relatives à l'unité de base utilisée - par exemple, si $\mathbf{x_1}$ a été mesurée en mètres, $\text{cov}(\mathbf{x_1}, \mathbf{x_1})$ serait en mètres carrés.

Les matrices de covariance sont écrites sous la forme suivante :

$$\square = \begin{bmatrix} \text{cov}(x_1, x_1) & \text{cov}(x_1, x_2) & \dots & \text{cov}(x_1, x_n) \\ \text{cov}(x_2, x_1) & \text{cov}(x_2, x_2) & \dots & \text{cov}(x_1, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{cov}(x_n, x_1) & \text{cov}(x_n, x_2) & \dots & \text{cov}(x_n, x_n) \end{bmatrix}$$

1. In a real robot, noise comes from all sorts of sources. Stray electromagnetic radiation adds extra voltages to sensor readings, vibrations and temperature variations throw off inertial measurement units, gear lash causes encoders to have inaccuracies when directions change… all sorts of things. It's important to realize that, by themselves, each of these sources of « noise » aren't guaranteed to follow any pattern. Some of them might be the « white noise » random vibrations you've probably heard on the radio. Others might be « pops » or single-loop errors. Others might be nominally zero, but strongly correlated with events on the robot. However, the *Central Limit Theorem* shows mathematically that regardless of how the individual sources of noise are distributed, as we add more and more of them up their combined effect eventually is distributed like a Gaussian. Since we do not know the exact individual sources of noise, the best choice of a model we can make is indeed that Gaussian function.


$\mu = 0, \sigma^2 = 0.2$
$\mu = 0, \sigma^2 = 1.0$
$\mu = 0, \sigma^2 = 5.0$
$\mu = -2, \sigma^2 = 0.5$
0.8
0.6
0.4
0.2
0.0
$\phi_{\mu,\sigma^2}(\chi)$
−4
−2
0
2
4
χ

Les filtres de Kalman

Important : Il est important de développer une intuition de ce que fait réellement un filtre de Kalman. Le livre Kalman and Bayesian Filters in Python par Roger Labbe fournit une excellente introduction visuelle et intéractive aux filtres bayésiens. Les filtres de Kalman de la WPILib utilisent l'algèbre linéaire pour rendre plus accessible les mathématiques, mais les idées sont similaires au cas unidimensionnel. Nous vous suggérons de lire le chapitre 4 pour avoir une idée de ce que font ces filtres.

To summarize, Kalman filters (and all Bayesian filters) have two parts : prediction and correction. Prediction projects our state estimate forward in time according to our system's dynamics, and correct steers the estimated state towards the measured state. While filters often perform both in the same timestep, it's not strictly necessary – For example, WPILib's pose estimators call predict frequently, and correct only when new measurement data is available (for example, from a low-framerate vision system).

Ce qui suit montre les équations d'un filtre de Kalman à temps discret :

Predict step

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{A}\hat{\mathbf{x}}_{k}^{+} + \mathbf{B}\mathbf{u}_k$$
$$\mathbf{P}_{k+1}^{-} = \mathbf{A}\mathbf{P}_{k}^{-}\mathbf{A}^T + \square\mathbf{Q}\square^T$$

Update step

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1}^{-}\mathbf{C}^T(\mathbf{C}\mathbf{P}_{k+1}^{-}\mathbf{C}^T + \mathbf{R})^{-1}$$
$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{C}\hat{\mathbf{x}}_{k+1}^{-} - \mathbf{D}\mathbf{u}_{k+1})$$
$$\mathbf{P}_{k+1}^{+} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C})\mathbf{P}_{k+1}^{-}$$

A	system matrix	$\hat{\mathbf{x}}$	state estimate vector
B	input matrix	**u**	input vector
C	output matrix	**y**	output vector
D	feedthrough matrix	$\square$	process noise intensity vector
P	error covariance matrix	**Q**	process noise covariance matrix
K	Kalman gain matrix	**R**	measurement noise covariance matrix

L'estimation d'état **x**, avec **P**, décrivent la moyenne et la covariance de la fonction gaussienne qui décrit l'estimation par notre filtre de l'état réel du système.

Les matrices de covariance du bruit de processus et de mesure

Les matrices de covariance du bruit de processus et de mesure **Q** et **R** décrivent la variance de chacun de nos états et mesures. N'oubliez pas que pour une fonction gaussienne, la variance est le carré de l'écart type de la fonction. Dans un WPILib, Q et R sont des matrices diagonales dont les diagonales contiennent leurs variances respectives. Par exemple, un filtre de Kalman avec les états $\begin{bmatrix}\text{position}\\ \text{velocity}\end{bmatrix}$ et les mesures $\begin{bmatrix}\text{position}\end{bmatrix}$ avec les écarts-types d'état $\begin{bmatrix}0.1\\ 1.0\end{bmatrix}$ `$etl'cart - typedemesure : math$ :begin{bmatrix}0.01end{bmatrix}` aurait les matrices suivantes **Q** et **R** :

$$Q = \begin{bmatrix}0.01 & 0\\ 0 & 1.0\end{bmatrix}, R = \begin{bmatrix}0.0001\end{bmatrix}$$

Matrice de covariance d'erreur

La matrice de covariance d'erreur **P** décrit la covariance de l'estimation d'état $\hat{\mathbf{x}}$. De manière informelle, **P** décrit notre certitude quant à l'état estimé. Si **P** est grand, notre incertitude sur l'état vrai est grande. Inversement, un **P** avec des éléments plus petits impliquerait moins d'incertitude sur notre véritable état.

Au fur et à mesure que nous projetons le modèle vers l'avant, **P** augmente à mesure que notre certitude sur l'état réel du système diminue.

L'étape de prédiction

En prédiction, notre estimation d'état est mise à jour selon la dynamique du système linéaire $\mathbf{x} = \mathbf{Ax} + \mathbf{Bu}$. De plus, notre covariance d'erreur **P** augmente selon la matrice de covariance du bruit de processus **Q**. Des valeurs plus grandes de :math :*mathbf{Q}* feront croître notre covariance d'erreur **P** plus rapidement. Ceci **P** est utilisé dans l'étape de correction pour pondérer le modèle et les mesures.

L'étape correcte

À l'étape correcte, notre estimation d'état est mise à jour pour inclure de nouvelles informations de mesure. Cette nouvelle information est pondérée par rapport à l'estimation d'état $\hat{\mathbf{x}}$ par le gain de Kalman **K**. Les grandes valeurs de **K** influencent plus fortement les mesures entrantes, tandis que les valeurs plus petites de **K** influencent plus fortement notre prédiction d'état. Parce que **K** est lié à **P**, des valeurs plus grandes de **P** augmenteront **K** et influencer encore plus les mesures. Si, par exemple, un filtre est utilisé pour une longue période, une valeur élevée de **P** pondérerait fortement les nouvelles informations.

Enfin, la covariance d'erreur **P** diminue pour augmenter notre confiance dans l'estimation de l'état.

Le réglage des filtres Kalman

Les constructeurs des classes de filtre Kalman de WPILib utilisent un système linéaire, un vecteur des écarts-types de bruit de processus et des écarts-types de bruit de mesure. Ceux-ci sont convertis en matrices **Q** et **R** en remplissant les diagonales avec le carré des écarts-types, ou variances, de chaque état ou mesure. En diminuant l'écart type d'un état (et donc son entrée correspondante dans **Q**), le filtre se fera moins confiance aux mesures entrantes. De même, l'augmentation de l'écart type d'un état fera davantage confiance aux mesures entrantes. Il en va de même pour les écarts-types de mesure - la diminution d'une entrée augmentra la confiance du filtre par rapport à la mesure entrante pour l'état correspondant, tandis que son augmentation diminuera la confiance dans la mesure.

Java

```
  // The observer fuses our encoder data and voltage inputs to reject noise.
  private final KalmanFilter<N1, N1, N1> m_observer =
      new KalmanFilter<>(
          Nat.N1(),
          Nat.N1(),
          m_flywheelPlant,
          VecBuilder.fill(3.0), // How accurate we think our model is
          VecBuilder.fill(0.01), // How accurate we think our encoder
          // data is
          0.020);
```

C++

```
#include <numbers>

#include <frc/DriverStation.h>
#include <frc/Encoder.h>
#include <frc/TimedRobot.h>
#include <frc/XboxController.h>
#include <frc/controller/LinearQuadraticRegulator.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/estimator/KalmanFilter.h>
#include <frc/motorcontrol/PWMSparkMax.h>
#include <frc/system/LinearSystemLoop.h>
#include <frc/system/plant/DCMotor.h>
#include <frc/system/plant/LinearSystemId.h>
#include <units/angular_velocity.h>
```

```
  // The observer fuses our encoder data and voltage inputs to reject noise.
  frc::KalmanFilter<1, 1, 1> m_observer{
      m_flywheelPlant,
      {3.0},   // How accurate we think our model is
      {0.01},  // How accurate we think our encoder data is
      20_ms};
```

Python

```
        # The observer fuses our encoder data and voltage inputs to reject noise.
        self.observer = wpimath.estimator.KalmanFilter_1_1_1(
            self.flywheelPlant,
            [3],  # How accurate we think our model is
            [0.01],  # How accurate we think our encoder data is
            0.020,
        )
```

Footnotes

32.8.4 Pose Estimators

WPILib includes pose estimators for differential, swerve and mecanum drivetrains. These estimators are designed to be drop-in replacements for the existing *odometry* classes that also support fusing latency-compensated robot pose estimates with encoder and gyro measurements. These estimators can account for encoder drift and noisy vision data. These estimators can behave identically to their corresponding odometry classes if only `update` is called on these estimators.

Pose estimators estimate robot position using a state-space system with the states $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$, which can represent robot position as a `Pose2d`. WPILib includes `DifferentialDrivePoseEstimator`, `SwerveDrivePoseEstimator` and `MecanumDrivePoseEstimator` to estimate robot position. In these, users call `update` periodically with encoder and gyro measurements (same as the odometry classes) to update the robot's estimated position. When the robot receives measurements of its field-relative position (encoded as a `Pose2d`) from sensors such as computer vision or V-SLAM, the pose estimator latency-compensates the measurement to accurately estimate robot position.

Here's how to initialize a `DifferentialDrivePoseEstimator` :

JAVA

```
  private final DifferentialDrivePoseEstimator m_poseEstimator =
      new DifferentialDrivePoseEstimator(
          m_kinematics,
          m_gyro.getRotation2d(),
          m_leftEncoder.getDistance(),
          m_rightEncoder.getDistance(),
          new Pose2d(),
          VecBuilder.fill(0.05, 0.05, Units.degreesToRadians(5)),
          VecBuilder.fill(0.5, 0.5, Units.degreesToRadians(30)));
```

C++

```
  frc::DifferentialDrivePoseEstimator m_poseEstimator{
      m_kinematics,
      m_gyro.GetRotation2d(),
      units::meter_t{m_leftEncoder.GetDistance()},
      units::meter_t{m_rightEncoder.GetDistance()},
      frc::Pose2d{},
      {0.01, 0.01, 0.01},
      {0.1, 0.1, 0.1}};
```

Add odometry measurements every loop by calling `Update()`.

JAVA

```
    m_poseEstimator.update(
        m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
↪getDistance());
```

C++

```
  m_poseEstimator.Update(m_gyro.GetRotation2d(),
                         units::meter_t{m_leftEncoder.GetDistance()},
                         units::meter_t{m_rightEncoder.GetDistance()});
```

Add vision pose measurements occasionally by calling `AddVisionMeasurement()`.

JAVA

```
    // Compute the robot's field-relative position exclusively from vision␣
↪measurements.
    Pose3d visionMeasurement3d =
        objectToRobotPose(m_objectInField, m_robotToCamera, m_cameraToObjectEntry);

    // Convert robot pose from Pose3d to Pose2d needed to apply vision measurements.
    Pose2d visionMeasurement2d = visionMeasurement3d.toPose2d();
```

C++

```
  // Compute the robot's field-relative position exclusively from vision
  // measurements.
  frc::Pose3d visionMeasurement3d = ObjectToRobotPose(
      m_objectInField, m_robotToCamera, m_cameraToObjectEntryRef);

  // Convert robot's pose from Pose3d to Pose2d needed to apply vision
  // measurements.
  frc::Pose2d visionMeasurement2d = visionMeasurement3d.ToPose2d();

  // Apply vision measurements. For simulation purposes only, we don't input a
  // latency delay -- on a real robot, this must be calculated based either on
  // known latency or timestamps.
  m_poseEstimator.AddVisionMeasurement(visionMeasurement2d,
                                       frc::Timer::GetFPGATimestamp());
```

Réglage des estimateurs de pose

All pose estimators offer user-customizable standard deviations for model and measurements (defaults are used if you don't provide them). Standard deviation is a measure of how spread out the noise is for a random signal. Giving a state a smaller standard deviation means it will be trusted more during data fusion.

For example, increasing the standard deviation for measurements (as one might do for a noisy signal) would lead to the estimator trusting its state estimate more than the incoming measurements. On the field, this might mean that the filter can reject noisy vision data well, at the cost of being slow to correct for model deviations. While these values can be estimated beforehand, they very much depend on the unique setup of each robot and global measurement method.

When incorporating AprilTag poses, make the vision heading standard deviation very large, make the gyro heading standard deviation small, and scale the vision x and y standard deviation by distance from the tag.

32.8.5 Débogage des modèles et contrôleurs d'espace d'état

Vérification des signes

L'une des causes les plus courantes de bogues avec les contrôleurs d'espace d'état est le mauvais agencement des signes (+ ou -). Par exemple, les modèles inclus dans WPILib s'attendent à ce qu'une tension positive entraîne une accélération positive, et vice versa. Si l'application d'une tension positive ne fait pas accélérer le mécanisme vers l'avant, ou si le déplacement «vers l'avant» fait diminuer le codeur (ou d'autres lectures de capteur), ils doivent être inversés de sorte qu'une entrée de tension positive entraîne une lecture de codeur positive. Par exemple, si j'applique une *entrée* de $[12, 12]^T$ (vitesse maximale vers l'avant pour les moteurs gauche et droit) à ma base pilotable de type différentielle, mes roues devraient propulser mon robot « vers l'avant » (le long de l'axe + X), et pour mes encodeurs de lire une vitesse positive.

Important : Le WPILib `DifferentialDrive`, par défaut, n'inverse aucun moteur. Vous devrez peut-être appeler la méthode `setInverted(true)` sur l'objet contrôleur de moteur pour inverser afin que l'entrée positive crée un mouvement vers l'avant.

L'importance des graphiques

Des données fiables de l'état du *système*, *entrée*s et sorties en fonction du temps sont importantes lors du débogage des contrôleurs et des observateurs d'espace-état. Une approche courante consiste à envoyer ces données sur les NetworkTables et à utiliser des outils tels que le *Shuffleboard*, qui nous permettent à la fois de représenter graphiquement les données en temps réel et aussi bien que les enregistrer dans un fichier CSV pour un traçage ultérieur avec des outils tels que Google Sheets, Excel ou Python.

Note : Par défaut, les NetworkTables sont limités à un taux de rafraîchissement de 10 hz. Pour les tests, cette limitation peut être contournée avec l'extrait de code suivant pour soumettre des données jusqu'à 100hz. Ce code doit être exécuté périodiquement pour forcer la

publication de nouvelles données.

Danger : Cette opération enverra des données supplémentaires (jusqu'à 100hz) sur NetworkTables, pouvant ainsi causer des retards avec le code utilisateur et les dashboards des robots. Ce qui augmentera également l'utilisation du réseau. C'est souvent une bonne idée de désactiver cette fonctionnalité pendant les compétitions.

JAVA

```
@Override
public void robotPeriodic() {
   NetworkTableInstance.getDefault().flush();
}
```

C++

```
void RobotPeriodic() {
   NetworkTableInstance::GetDefault().Flush();
}
```

PYTHON

```
from ntcore import NetworkTableInstance

def robotPeriodic(self):
   NetworkTableInstance.getDefault().flush()
```

Compensation du retard d'entrée

Souvent, certaines données d'entrée du capteur (c'est-à-dire les lectures de vitesse) peuvent être retardées en raison du filtrage intégré que les contrôleurs de moteur intelligents ont tendance à effectuer. Par défaut, le gain K du LQR ne suppose aucun retard d'entrée, donc l'introduction d'un retard significatif de l'ordre de dizaines de millisecondes peut provoquer une instabilité. Pour lutter contre cela, le gain K du LQR peut être réduit, échangeant les performances contre la stabilité. Un exemple de code permettant de compenser cette latence de manière mathématiquement rigoureuse est disponible :ref : *ici <docs / software / advanced-controls / state-space / state-space-intro : LQR and Measurement Latency Compensation>*.

32.9 Glossaire des commandes

bang-bang control
: A very simple, no-tuning-required closed-loop control technique. It simply « turns on » the *control effort* when the *process variable* is too small, and « turns off » the control effort when the process variable is too big. It works well in some cases, but not all. See « Bang-bang » control on Wikipedia for more info.

Cartesian coordinate system
: A set of points in space where each point is described by a set of numbers, indicating its *coordinates* within that space. These coordinates are an expression of the *orthogonal* distance of each point from a set of fixed, orthogonal axes (IE, a « rectangular » system). 2-dimension and 3-dimension spaces are most common in FRC (and likely what was learned in algebra 1), but any number of dimensions is theoretically possible. See Cartesian coordinate system on Wikipedia for more info.

churning losses
: Complex friction-like forces arising from the fact that when gears and bearings rotate, they must displace liquid lubricant. This reduces the efficiency of rotating mechanisms.

control signal
: The driving signal sent to a *plant* by a *controller*, usually quantified as a voltage.

Effort de contrôle
: *Control signal*

la loi de commande
: Une formule mathématique qui génère des *entrées* pour conduire un *Système* vers un *état*, étant donné l'état courant. Un exemple courant est la loi de contrôle $\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$

manette
: Utilisé en position ou rétroaction négative avec l' *Usine* pour amener un *état du système* souhaité en forçant la différence entre le signal de *Référence* et la Sortie vers zéro.

convolution
: A mathematical operation that calculates a weighted moving average of one function, with the weights assigned by a second function. A common way to « filter » sensor input is to apply a *convolution* to it, using a carefully-chosen filtering function. See convolution. on Wikipedia for more info.

counter-electromotive force
: A *voltage* generated in a spinning motor. The voltage is a result of the fact that has a coil of wire rotating near a magnet. See Counter-electromotive_force on Wikipedia for more info.

current
: The flow of electrons through a conductor. Current is described with a unit of « Amps » (or simply « A »), and is measured at a single point in a circuit. One amp is equal to 6241509074000000000 electrons moving past the measurement point in one second.

dynamique
: Une branche de la physique concernée par le mouvement des corps sous l'action des forces. Dans le contrôle moderne, les systèmes évoluent en fonction de leur dynamique.

derivative
: A mathematical operation which evaluates the « rate-of-change » of a function at a given point. See derivative on Wikipedia for more info.

Erreur
: *Référence* moins la Sortie ou *état*.

exponential search
: An iterative process of finding a specific value within a wide search range by applying a multiplicative factor to the search value. See exponential search on Wikipedia for more info.

exponential smoothing
: A very common way to implement a simple low-pass filter, using an exponential window function in a *convolution* with an input signal. The convolution operation simplifies down to a very simple set of math operations on the current input and previous output. See exponential smoothing on Wikipedia for more info.

le gain
: A scalar value that relates the magnitude of an input signal to the magnitude of an output signal. For example, `gain` in `output = gain * input`. A gain greater than one would amplify an input signal, while a gain less than one would dampen an input signal. A negative gain would negate the input signal.

Gaussian distribution
: A special mathematical function that describes distributions of averages. The graph of a Gaussian function is a « bell curve » shape. This function is described by its mean (the location of the « peak » of the bell curve) and variance (a measure of how « spread out » the bell curve is). See Gaussian distribution on Wikipedia for more info.

gradient
: The *derivative*, but applied to a function with multiple inputs. As a result, the output is both the magnitude of the rate of change, and the vector direction along which it occurs.

l'état caché
: Un *état* qui ne peut pas être mesuré directement, mais dont la *dynamique* peut être liée à d'autres états.

Entrée
: Une entrée pour *Usine* (d'où le nom) qui peut être utilisée pour changer l'état de *Usine*.
  — Ex. Un volant d'inertie aura 1 entrée : la tension du moteur qui l'entraîne.
  — Ex. Un groupe motopropulseur peut avoir 2 entrées : les tensions des moteurs gauche et droit.

  Les entrées sont souvent représentées par la variable **u**, un vecteur de colonne avec une entrée par :term :*Entrées* au *Système*.

least-squares regression
: A curve-fitting technique which picks a curve to minimizes the *square* of the error between the fitted curve, and the actual measured data. See ordinary least-squares regression on Wikipedia for more info.

LQR
: Linear-Quadratic Regulator - A feedback control scheme which seeks to operate a system in a « most optimal » or « lowest cost » manner, in the sense of minimizing the square of some « cost function » that represents a combination of system error and control effort. This requires an accurate mathematical model of the system being controlled, and function describing the « cost » of any given system state. See LQR on Wikipedia for more info.

la mesure
: Les mesures sont des Sorties qui sont mesurées à partir d'un procédé, ou un système physique, à l'aide de capteurs.

Modèle
: Un ensemble d'équations mathématiques qui reflète certains aspects du comportement physique d'un *Système*.

Observateur
Dans la théorie du contrôle, un système qui fournit une estimation de l" *état* interne d'un *système* réel à partir de mesures des Entrées et de la Sortie du *système*. WPILib inclut une classe de filtre Kalman pour l'observation des systèmes linéaires, et les classes ExtendedKalmanFilter et UnscentedKalmanFilter pour les systèmes non linéaires.

orthogonal
Having the property of being independent, or lacking mutual influence. For example, two lines are orthogonal if moving any number of units along one line causes zero displacement along the other line. In a *cartesian coordinate system*, orthogonal lines are often said to have 90-degree angles between each other.

Le paramètre Output (Sortie)
Mesures des capteurs. Il peut y avoir plus de mesures que d'états. Ces sorties sont utilisées dans l'étape «correcte» des filtres de Kalman.
— Ex. Un volant peut avoir une sortie d'un encodeur qui mesure sa vitesse.
— Ex. Un groupe motopropulseur peut utiliser solvePNP et V-SLAM pour trouver sa position x/y/cap sur le terrain. C'est bien qu'il y ait 6 mesures (résoudrePNP x/y/cap et V-SLAM x/y/cap) et 3 états (robot x/y/cap).
Les sorties d'un *Système* sont souvent représentées en utilisant la variable **y**, un vecteur de colonne avec une entrée par Sortie (ou chose que nous pouvons mesurer). Par exemple, si notre *Système* avait des états pour la vitesse et l'accélération mais que notre capteur ne pouvait mesurer que la vitesse, notre vecteur Sortie inclurait seulement la vitesse du *Système*.

phase portrait
A graph of a function's value and its *derivative* as they change in time, given some initial starting conditions. They are useful for analyzing system behavior (stable/unstable operating points, limit cycles, etc.) given a certain set of parameters or starting conditions. See phase portrait on Wikipedia for more info.

PID
Proportional-Integral-Derivative - A feedback controller which calculates a *control signal* from a weighted sum of the *error*, the rate of change of the error, and an accumulated sum of previous errors. See PID controller. on Wikipedia for more info.

Usine
Le *Système* ou l'ensemble des actionneurs contrôlés.

Variable de procédé
Le terme utilisé pour décrire la sortie de l' *Usine* dans le contexte du contrôle PID.

r-squared
A statistical measurement of how well a model predicts a set of data, representing the fraction of the observed variation in the independent variable that is accurately predicted by the model. The value typically runs from 0.0 (a terrible fit, equivalent to just guessing the average value of your independent variable) to 1.0 (a perfect fit). See Coefficient_of_determination on Wikipedia for more info.

Référence
L'état souhaité. Cette valeur est utilisée comme point de référence pour le calcul d'erreur d'un contrôleur.

Temps de montée
Le temps qu'un *Système* prend pour atteindre initialement la *Référence* après avoir appliqué une *Impulsion d'entrée*.

RMSE
Root Mean Squared Error - Statistical measurement of how well a curve is fit to a set of data. It is calculated as the square root of the average (mean) of the squares of all the errors between the actual sample and the curve fit. It has units of the original input data. See Root Mean Squared Error on Wikipedia for more info.

Point de consigne
Le terme utilisé pour décrire la *Référence* d'un contrôleur PID.

Temps de stabilisation
Le temps qu'un *Système* prend pour se stabiliser au point de *Référence* après l'application d'une *Impulsion d'entrée*.

signum function
A non-continuous function that expresses the « sign » of its input. It is equal to -1 for all negative input numbers, 0 for an input of 0, and 1 for all positive input numbers. See signum function, on Wikipedia for more info.

état
Une caractéristique d'un *Système* (par exemple, la vitesse) qui peut être utilisée pour déterminer le comportement futur du *Système*. Dans la notation d'espace d'états, l'état d'un système est écrit sous la forme d'un vecteur de colonne décrivant sa position dans l'espace d'états.
— Ex. Un système de transmission peut avoir les états $beginbmatrixx$ y $theta\ endbmatrix$ pour décrire sa position sur le terrain.
— Ex. Un système d'ascenseur peut avoir les états $beginbmatrix\ textposition$ $textvitesse\ endbmatrix$ pour décrire sa hauteur et sa vitesse actuelles.
L'état du *Système* est souvent représenté par la variable **x**, un vecteur de colonne avec une entrée par **:terme :`état`**.

statistically robust
The property of a data processing algorithm which makes it resilient to a noisy or outlier-prone data set. Designing statistically robust algorithms on robots is important because real-world sensor data can often be unpredictable, but unexpected robot behavior is never desirable. See Robust Statistics on Wikipedia for more info.

Erreur régime permanant
Erreur obtenue après que le *système* a atteint l'équilibre.

Impulsion d'entrée
Une *Entrée* de *Système* prenant la forme 0 pour $t < 0$ et une constante supérieure à 0 pour $t \geq 0$. Une impulsion d'entrée qui a pour valeur 1 pour $t \geq 0$ est appelée une impulsion d'entrée unitaire.

Réponse d'impulsion
La réponse du *Système* à une *Impulsion d'entrée*.

Système
Un terme englobant l' *Usine* et son intéraction avec le Contrôleur et l' *Observateur*, qui est traité comme une seule entité. Mathématiquement parlant, un *Système* mappe les *Entrées* aux *Sorties* via une combinaison linéaire d' *États*.

l'identification du système
Le processus de capture d'un *systèmes dynamique* dans un modèle mathématique à l'aide de données mesurées. La suite d'outils SysId utilise l'identification du système pour trouver les termes kS, kV et kA.

Réponse du système
Le comportement d'un *Système* dans le temps pour une *Entrée* donnée.

voltage
The measurement of how much an electric field is « pushing » electrons through a circuit. It is sometimes called « Electromotive Force », or « EMF ». It is measured in units of « Volts ». It always is defined between *two* points in a circuit. If one electron travels between two points that have one volt of EMF between them, it will have been accelerated to the point of having $\frac{1}{6241509074000000000}$ joules of energy.

viscous drag
: The force generated from an object moving *relatively* slowly through non-turbulent fluid. In this region, the force is roughly proportional to the *velocity* of the object. It describes the most common type of « air resistance » an FRC robot would encounter, as well as losses in a gearbox from displacing grease. See Drag (physics) on Wikipedia for more info.

x-point
: $\dot{\mathbf{x}}$, ou x-point : la dérivée du vecteur d'état $\mathbf{x}$. Si le *système* avait juste une vitesse comme *état*, alors $\dot{\mathbf{x}}$ représenterait l'accélération du *système*.

x-châpeau
: $\hat{\mathbf{x}}$, ou x-châpeau : l'estimation de l'état d'un système, tel qu'il serait estimé par un *observateur*.

33

Fonctionnalités Pratiques

Cette section couvre des fonctionnalités générales pratiques qui peut être utilisé avec d'auture fonctionnalités de programmation avancées.

33.1 Planification des Fonctions à des Fréquences Personnalisées

La méthode `addPeriodic()` de `TimedRobot` nous permettons d'exécuter des méthodes personnalisées à un rythme plus rapide que le taux de mise à jour périodique par défault de `TimedRobot` (20 ms). Précédemment, les équipes auraient dû créer un `Notifier` pour exécuter les contrôleurs de rétroaction plus souvent que la période de boucle de `TimedRobot` de 20 ms (Exécuter `TimedRobot` plus souvent que ça n'est pas conseillé). Maintenant, les utilisateurs peuvent exécuter les contrôleurs de rétroaction plus souvent que la boucle principale du robot, mais de manière synchrone avec les fonctions périodiques de `TimedRobot`, ce qui élimine les problèmes potentiels de sécurité des threads.

La méthode `addPeriodic()` (Java) / `AddPeriodic()` (C++) prend un lambda (la fonction à exécuter), ainsi que la période demandée et un offset optionnel à partir de l'heure de départ commune. La troisième argument optionnel est utile pour la planification d'une fonction dans un créneau horaire différent par rapport aux autre méthodes périodiques de `TimedRobot`.

Note : Les unités pour la période et le décalage sont les secondes en Java. En C++, la *bibliothèque d'unités* peut être utilisée pour spécifier une période et un décalage dans n'importe quelle unité de temps.

Java

```
public class Robot extends TimedRobot {
    private Joystick m_joystick = new Joystick(0);
    private Encoder m_encoder = new Encoder(1, 2);
    private Spark m_motor = new Spark(1);
    private PIDController m_controller = new PIDController(1.0, 0.0, 0.5, 0.
↪01);

    public Robot() {
        addPeriodic(() -> {
            m_motor.set(m_controller.calculate(m_encoder.getRate()));
        }, 0.01, 0.005);
    }

    @Override
    public teleopPeriodic() {
        if (m_joystick.getRawButtonPressed(1)) {
            if (m_controller.getSetpoint() == 0.0) {
                m_controller.setSetpoint(30.0);
            } else {
                m_controller.setSetpoint(0.0);
            }
        }
    }
  }
```

C++ (Header ou en-tête)

```
class Robot : public frc::TimedRobot {
 private:
  frc::Joystick m_joystick{0};
  frc::Encoder m_encoder{1, 2};
  frc::Spark m_motor{1};
  frc::PIDController m_controller{1.0, 0.0, 0.5, 10_ms};

  Robot();

  void TeleopPeriodic() override;
};
```

C++ (Source)

```
void Robot::Robot() {
  AddPeriodic([&] {
    m_motor.Set(m_controller.Calculate(m_encoder.GetRate()));
  }, 10_ms, 5_ms);
}

void Robot::TeleopPeriodic() {
  if (m_joystick.GetRawButtonPressed(1)) {
    if (m_controller.GetSetpoint() == 0.0) {
```

(suite sur la page suivante)

(suite de la page précédente)

```
      m_controller.SetSetpoint(30.0);
    } else {
      m_controller.SetSetpoint(0.0);
    }
  }
}
```

La méthode `teleopPeriodic()` dans cet exemple s'exécute toutes les 20 ms, et la mise à jour du contrôleur est exécutée toutes les 10 ms avec un décalage de 5 ms à partir du moment où `teleopPeriodic()` s'exécute de sorte que leurs tranches de temps ne sont pas en conflit (par exemple, `teleopPeriodic()` fonctionne à 0 ms, 20 ms, 40 ms, etc. ; le contrôleur fonctionne à 5 ms, 15 ms, 25 ms, etc.).

33.2 Event-Based Programming With EventLoop

Many operations in robot code are driven by certain conditions ; buttons are one common example. Conditions can be polled with an *imperative programming* style by using an `if` statement in a periodic method. As an alternative, WPILib offers an *event-driven programming* style of API in the shape of the `EventLoop` and `BooleanEvent` classes.

Note : The example code here is taken from the EventLoop example project (Java/C++).

33.2.1 EventLoop

The `EventLoop` class is a « container » for pairs of conditions and actions, which can be polled using the `poll()`/`Poll()` method. When polled, every condition will be queried and if it returns `true` the action associated with the condition will be executed.

JAVA

```
private final EventLoop m_loop = new EventLoop();
private final Joystick m_joystick = new Joystick(0);
@Override
public void robotPeriodic() {
  // poll all the bindings
  m_loop.poll();
}
```

C++

```
frc::EventLoop m_loop{};
void RobotPeriodic() override { m_loop.Poll(); }
```

Avertissement : The `EventLoop`'s `poll()` method should be called consistently in a `*Periodic()` method. Failure to do this will result in unintended loop behavior.

33.2.2 BooleanEvent

The `BooleanEvent` class represents a boolean condition : a `BooleanSupplier` (Java) / `std::function<bool()>` (C++).

To bind a callback action to the condition, use `ifHigh()`/`IfHigh()` :

JAVA

```
BooleanEvent atTargetVelocity =
    new BooleanEvent(m_loop, m_controller::atSetpoint)
        // debounce for more stability
        .debounce(0.2);

// if we're at the target velocity, kick the ball into the shooter wheel
atTargetVelocity.ifHigh(() -> m_kicker.set(0.7));
```

C++

```
frc::BooleanEvent atTargetVelocity =
    frc::BooleanEvent(
        &m_loop,
        [&controller = m_controller] { return controller.AtSetpoint(); })
        // debounce for more stability
        .Debounce(0.2_s);

// if we're at the target velocity, kick the ball into the shooter wheel
atTargetVelocity.IfHigh([&kicker = m_kicker] { kicker.Set(0.7); });
```

N'oubliez pas que la liaison des boutons est *déclarative* : les liaisons ne doivent être déclarées qu'une seule fois, idéalement pendant l'initialisation du robot. La librarie gère tout le reste.

33.2.3 Composing Conditions

`BooleanEvent` objects can be composed to create composite conditions. In C++ this is done using operators when applicable, other cases and all compositions in Java are done using methods.

and() / &&

The `and()`/&& composes two `BooleanEvent` conditions into a third condition that returns `true` only when **both** of the conditions return `true`.

JAVA

```
    // if the thumb button is held
    intakeButton
        // and there is not a ball at the kicker
        .and(isBallAtKicker.negate())
        // activate the intake
        .ifHigh(() -> m_intake.set(0.5));
```

C++

```
    // if the thumb button is held
    (intakeButton
     // and there is not a ball at the kicker
     && !isBallAtKicker)
        // activate the intake
        .IfHigh([&intake = m_intake] { intake.Set(0.5); });
```

or() / ||

The `or()`/`||` composes two `BooleanEvent` conditions into a third condition that returns `true` only when **either** of the conditions return `true`.

JAVA

```
    // if the thumb button is not held
    intakeButton
        .negate()
        // or there is a ball in the kicker
        .or(isBallAtKicker)
        // stop the intake
        .ifHigh(m_intake::stopMotor);
```

C++

```
    // if the thumb button is not held
    (!intakeButton
     // or there is a ball in the kicker
     || isBallAtKicker)
        // stop the intake
        .IfHigh([&intake = m_intake] { intake.Set(0.0); });
```

negate() / !

The `negate()/!` composes one `BooleanEvent` condition into another condition that returns the opposite of what the original conditional did.

JAVA

```
        // and there is not a ball at the kicker
        .and(isBallAtKicker.negate())
```

C++

```
     // and there is not a ball at the kicker
     && !isBallAtKicker)
```

debounce() / Debounce()

To avoid rapid repeated activation, conditions (especially those originating from digital inputs) can be debounced with the *WPILib Debouncer class* using the *debounce* method :

JAVA

```
    BooleanEvent atTargetVelocity =
        new BooleanEvent(m_loop, m_controller::atSetpoint)
            // debounce for more stability
            .debounce(0.2);
```

C++

```
    frc::BooleanEvent atTargetVelocity =
        frc::BooleanEvent(
            &m_loop,
            [&controller = m_controller] { return controller.AtSetpoint(); })
            // debounce for more stability
            .Debounce(0.2_s);
```

rising(), falling()

Often times it is desired to bind an action not to the *current* state of a condition, but instead to when that state *changes*. For example, binding an action to when a button is newly pressed as opposed to when it is held. This is what the `rising()` and `falling()` decorators do : `rising()` will return a condition that is `true` only when the original condition returned `true` in the *current* polling and `false` in the *previous* polling ; `falling()` returns a condition that returns `true` only on a transition from `true` to `false`.

> **Avertissement :** Due to the « memory » these conditions have, do not use the same instance in multiple places.

JAVA

```
    // when we stop being at the target velocity, it means the ball was shot
    atTargetVelocity
        .falling()
        // so stop the kicker
        .ifHigh(m_kicker::stopMotor);
```

C++

```
    // when we stop being at the target velocity, it means the ball was shot
    atTargetVelocity
        .Falling()
        // so stop the kicker
        .IfHigh([&kicker = m_kicker] { kicker.Set(0.0); });
```

Downcasting `BooleanEvent` Objects

To convert `BooleanEvent` objects to other types, most commonly the `Trigger` subclass used for *binding commands to conditions*, the generic `castTo()`/`CastTo()` decorator exists :

JAVA

```
Trigger trigger = booleanEvent.castTo(Trigger::new);
```

C++

```
frc2::Trigger trigger = booleanEvent.CastTo<frc2::Trigger>();
```

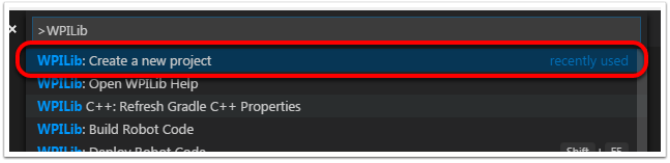
Note : In Java, the parameter expects a method reference to a constructor accepting an `EventLoop` instance and a `BooleanSupplier`. Due to the lack of method references, this parameter is defaulted in C++ as long as a constructor of the form `Type(frc::EventLoop*, std::function<bool()>)` exists.

34

Exemples fournis de Projets WPILib

Avertissement : Bien qu'on ne ménage aucun effort pour maintenir les exemples WPILib fonctionnels, ils ne sont *pas* destinés à être utilisés « tel quel ». À tout le moins, les valeurs constantes spécifiques aux robots devront être modifiées pour que le code fonctionne sur le robot de l'utilisateur. Beaucoup de constantes empiriques ont des valeurs « factices » à des fins de démonstration. Les utilisateurs sont fortement encouragés à écrire leur propre code (à partir de zéro ou à partir d'un modèle existant) plutôt que de copier le code d'exemple et de l'utiliser tel quel.

Les exemples de projets WPILib illustrent un grand nombre de fonctionnalités de bibliothèque et de modèles d'utilisation. Les projets vont de la simple démonstration d'une seule fonctionnalité à des programmes robotiques complets et compétitifs. Tous ces exemples sont disponibles dans VS Code en entrant `Ctrl+Shift+P`, puis en sélectionnant *WPILib : Create a new project* et en choisissant l'exemple.



34.1 Exemples de base

Ces exemples démontrent la fonctionnalité de base/minimale du robot. Ils sont utiles pour des équipes recrues qui s'initient à la programmation de robots, mais sont grandement limités en terme de fonctionnalité.

- **Arcade Drive** (Java, C++, Python) : Demonstrates a simple differential drive implementation using « arcade »-style controls through the `DifferentialDrive` class.
- **Arcade Drive Xbox Controller** (Java, C++, Python) : Demonstrates the same functionality seen in the previous example, except using an `XboxController` instead of an ordinary joystick.
- **Getting Started** (Java, C++, Python) : Demonstrates a simple autonomous routine that drives forwards for two seconds at half speed.

— **Mecanum Drive** (Java, C++, Python) : Demonstrates a simple mecanum drive implementation using the `MecanumDrive` class.
— **Motor Controller** (Java, C++, Python) : Demonstrates how to control the output of a motor with a joystick with an encoder to read motor position.
— **Simple Vision** (Java, C++, Python) : Demonstrates how to stream video from a USB camera to the dashboard.
— **Relay** (Java, C++, Python) : Demonstrates the use of the `Relay` class to control a relay output with a set of joystick buttons.
— **Solenoids** (Java, C++, Python) : Demonstrates the use of the `Solenoid` and `DoubleSolenoid` classes to control solenoid outputs with a set of joystick buttons.
— **TankDrive** (Java, C++, Python) : Demonstrates a simple differential drive implementation using « tank »-style controls through the `DifferentialDrive` class.
— **Tank Drive Xbox Controller** (Java, C++, Python) : Demonstrates the same functionality seen in the previous example, except using an `XboxController` instead of an ordinary joystick.

34.2 Exemples de contrôle (commande)

Ces exemples illustrent les implémentations dans la WPILib de contrôles de robot courants. Les capteurs peuvent être présents, mais ils ne constituent pas le point focal dans ces exemples.

— **DifferentialDriveBot** (Java, C++, Python) : Demonstrates an advanced differential drive implementation, including encoder-and-gyro odometry through the `DifferentialDriveOdometry` class, and composition with PID velocity control through the `DifferentialDriveKinematics` and `PIDController` classes.
— **Elevator with profiled PID controller** (Java, C++, Python) : Demonstrates the use of the `ProfiledPIDController` class to control the position of an elevator mechanism.
— **Elevator with trapezoid profiled PID** (Java, C++, Python) : Demonstrates the use of the `TrapezoidProfile` class in conjunction with a « smart motor controller » to control the position of an elevator mechanism.
— **Gyro Mecanum** (Java, C++, Python) : Demonstrates field-oriented control of a mecanum robot through the `MecanumDrive` class in conjunction with a gyro.
— **MecanumBot** (Java, C++, Python) : Demonstrates an advanced mecanum drive implementation, including encoder-and-gyro odometry through the `MecanumDriveOdometry` class, and composition with PID velocity control through the `MecanumDriveKinematics` and `PIDController` classes.
— **PotentiometerPID** (Java, C++, Python) : Demonstrates the use of the `PIDController` class and a potentiometer to control the position of an elevator mechanism.
— **RamseteController** (Java, C++, Python) : Demonstrates the use of the `RamseteController` class to follow a trajectory during the autonomous period.
— **SwerveBot** (Java, C++, Python) : Demonstrates an advanced swerve drive implementation, including encoder-and-gyro odometry through the `SwerveDriveOdometry` class, and composition with PID position and velocity control through the `SwerveDriveKinematics` and `PIDController` classes.
— **UltrasonicPID** (Java, C++, Python) : Demonstrates the use of the `PIDController` class in conjunction with an ultrasonic sensor to drive to a set distance from an object.

34.3 Exemples de programmation de capteurs

Ces exemples démontrent la lecture de signaux à partir des capteurs et le traitement des données à l'aide de WPILib. Le contrôle des mécanismes peut être présent, mais ce n'est pas le point focal de ces exemples.

- **Axis Camera Sample** (Java, C++) : Démontre l'utilisation d'OpenCV et d'un Axis Netcam pour superposer un rectangle sur un flux vidéo capturé et le diffuser sur le Dashboard.
- **Power Distribution CAN Monitoring** (Java, C++, Python) : Demonstrates obtaining sensor information from a Power Distribution module over CAN using the `PowerDistribution` class.
- **Duty Cycle Encoder** (Java, C++, Python) : Demonstrates the use of the `DutyCycleEncoder` class to read values from a PWM-type absolute encoder.
- **DutyCycleInput** (Java, C++, Python) : Demonstrates the use of the `DutyCycleInput` class to read the frequency and fractional duty cycle of a *PWM* input.
- **Encoder** (Java, C++, Python) : Demonstrates the use of the `Encoder` class to read values from a quadrature encoder.
- **Gyro** (Java, C++, Python) : Demonstrates the use of the `AnalogGyro` class to measure robot heading and stabilize driving.
- **Intermediate Vision** (Java, C++, Python) : Demonstrates the use of OpenCV and a USB camera to overlay a rectangle on a captured video feed and stream it to the dashboard.
- **AprilTagsVision** (Java, C++) : Demonstrates on-roboRIO detection of AprilTags using an attached USB camera.
- **Ultrasonic** (Java, C++, Python) : Demonstrates the use of the `Ultrasonic` class to read data from an ultrasonic sensor in conjunction with the `MedianFilter` class to reduce signal noise.
- **SysIdRoutine** (Java, C++) : Demonstrates the use of the SysIdRoutine API to gather characterization data for a differential drivetrain.

34.4 Exemples de programmes basés sur l'architecture orientée Commande

Ces exemples démontrent l'utilisation du *cadre de développement orienté Commande*.

- **ArmBot** (Java, C++, Python) : Demonstrates the use of a `ProfiledPIDSubsystem` to control a robot arm.
- **ArmBotOffboard** (Java, C++, Python) : Demonstrates the use of a `TrapezoidProfileSubsystem` in conjunction with a « smart motor controller » to control a robot arm.
- **DriveDistanceOffboard** (Java, C++, Python) : Demonstrates the use of a `TrapezoidProfileCommand` in conjunction with a « smart motor controller » to drive forward by a set distance with a trapezoidal motion profile.
- **FrisbeeBot** (Java, C++, Python) : A complete set of robot code for a simple frisbee-shooting robot typical of the 2013 FRC® game *Ultimate Ascent*. Demonstrates simple PID control through the `PIDSubystem` class.
- **Gears Bot** (Java, C++) : Un ensemble complet de codes de robot pour le robot de démonstration WPI, GearsBot.
- **Gyro Drive Commands** (Java, C++, Python) : Demonstrates the use of `PIDCommand` and `ProfiledPIDCommand` in conjunction with a gyro to turn a robot to face a specified heading and to stabilize heading while driving.
- **Inlined Hatchbot** (Java, C++, Python) : A complete set of robot code for a simple hatch-delivery bot typical of the 2019 FRC game *Destination : Deep Space*. Commands

are written in an « inline » style, in which explicit subclassing of `Command` is avoided.
- **Traditional Hatchbot** (Java, C++, Python) : A complete set of robot code for a simple hatch-delivery bot typical of the 2019 FRC game *Destination : Deep Space*. Commands are written in a « traditional » style, in which subclasses of `Command` are written for each robot action.
- **MecanumControllerCommand** (Java, C++) : Démontre la génération et le suivi de trajectoire pour un entraînement de type mécanum à l'aide des classes `TrajectoryGenerator` et `MecanumControllerCommand`.
- **RamseteCommand** (Java, C++, Python) : Demonstrates trajectory generation and following with a differential drive using the `TrajectoryGenerator` and `RamseteCommand` classes. A matching step-by-step tutorial can be found *here*.
- **Select Command Example** (Java, C++, Python) : Demonstrates the use of the `SelectCommand` class to run one of a selection of commands depending on a runtime-evaluated condition.
- **SwerveControllerCommand** (Java, C++) : Démontre la génération et le suivi de trajectoire pour un entraînement de type swerve à l'aide des classes `TrajectoryGenerator` et `SwerveControllerCommand`.

34.5 Exemples d'espace d'état

Ces exemples illustrent l'utilisation de la *commande espace-d'état*.
- **StateSpaceFlywheel** (Java, C++, Python) : Demonstrates state-space control of a flywheel.
- **StateSpaceFlywheelSysId** (Java, C++, Python) : Demonstrates state-space control using SysId's System Identification for controlling a flywheel.
- **StateSpaceElevator** (Java, C++) : Démontre le contrôle de l'espace d'état d'un élévateur.
- **StateSpaceArm** (Java, C++) : Démontre le contrôle de l'espace d'état d'un bras.
- **StateSpaceDriveSimulation** (Java, C++) : Démontre le contrôle de l'espace d'états d'une transmission différentielle en combinaison avec un contrôleur de suivi de trajectoire RAMSETE et une classe Field2d.

34.6 Exemples de physique de simulation

Ces exemples démontrent l'utilisation de la simulation physique.
- **ElevatorSimulation** (Java, C++, Python) : Demonstrates the use of physics simulation with a simple elevator.
- **ArmSimulation** (Java, C++, Python) : Demonstrates the use of physics simulation with a simple single-jointed arm.
- **StateSpaceDriveSimulation** (Java, C++) : Démontre le contrôle de l'espace d'états d'une transmission différentielle en combinaison avec un contrôleur de suivi de trajectoire RAMSETE et une classe Field2d.
- **SimpleDifferentialDriveSimulation** (Java, C++) : Un exemple simple d'une base pilotable qui peut être utilisée dans la simulation.

34.7 Exemples divers

Ces exemples illustrent une variété de fonctionnalités WPILib qui ne s'inscrivent dans aucune des catégories ci-dessus.

- **Addressable LED** (Java, C++, Python) : Demonstrates the use of the `AddressableLED` class to control RGB LEDs for robot decoration and/or driver feedback.
- **DMA** (Java, C++) : Demonstrates the use of DMA (Direct Memory Access) to read from sensors without using the RoboRIO's CPU.
- **HAL** (C++) : Démontre l'utilisation de HAL (Hardware Abstraction Layer) sans utiliser le reste de WPILib. Cet exemple est destiné aux utilisateurs avancés (C++ uniquement).
- **HID Rumble** (Java, C++, Python) : Demonstrates the use of the « rumble » functionality for tactile feedback on supported HIDs (such as XboxControllers).
- **Shuffleboard** (Java, C++, Python) : Demonstrates configuring tab/widget layouts on the « Shuffleboard » dashboard from robot code through the `Shuffleboard` class's fluent builder API.
- **RomiReference** (Java, C++, Python) : A command based example of how to run the *Romi robot*.
- **Mechanism2d** (Java, C++, Python) : A simple example of using *Mechanism2d*.

35

Exemples de projets tiers

Cette liste vous aide à trouver des exemples de programmes à utiliser avec des appareils tiers. Vous pouvez trouver de l'aide pour bon nombre de ces tiers sur la page *Ressources d'assistance*.

- Cross The Road Electronics (CTRE)
- Kauai Labs (navX)
- Limelight (additional examples, called tutorials, can be found on the left)
- PhotonVision
- REV Robotics

36

Composants matériels - Notions de base

36.1 Les meilleures techniques de câblage

Astuce : The article *Intro to FRC Robot Wiring* walks through the details of what connects where to wire up the FRC Control System and this article provides some additional « Best Practices » that may increase reliability and make maintenance easier. Take a look at *Preemptive Troubleshooting* for more tips and tricks.

36.1.1 Vibration et choc

Un robot FRC® est un environnement incroyablement brutal quand il s'agit de vibrations et de charges de choc. Bien que de nombreux appareils électroniques spécifiques à FRC soient largement testés pour la robustesse mécanique dans ces conditions, quelques composants, comme la radio, ne sont pas spécifiquement conçus pour une utilisation sur une plate-forme mobile. Prendre des mesures pour réduire le choc et les vibrations auxquelles ces composants sont exposés peut aider à réduire les défaillances. Quelques suggestions qui peuvent réduire les défaillances mécaniques

- Isolation des vibrations - Assurez-vous d'isoler tous les composants qui créent des vibrations excessives, telles que les compresseurs, à l'aide des « isolateurs de vibration ». Cela aidera à réduire les vibrations sur le robot qui peuvent desserrer les attaches et causer une défaillance de fatigue prématurée sur certains composants électroniques.
- Pare-chocs - Utilisez des pare-chocs pour couvrir autant que possible votre robot lors votre conception. Bien que les règles exigent une couverture spécifique des pare-chocs autour des coins de votre robot, maximiser l'utilisation de pare-chocs augmente la probabilité que toutes les collisions soient amorties par vos pare-chocs. Les pare-chocs réduisent considérablement les "forces g" subies lors d'une collision par rapport à une collision directe sur une surface non protégée du robot, réduisant le choc subi par l'électronique et diminuant le risque d'une défaillance liée au choc.
- Montage d'amortisseur de choc - Vous pouvez choisir d'installer des amortisseurs de choc sur une partie ou la totalité de vos composants électroniques pour réduire davantage les forces qu'ils subissent dans les collisions de robots. Ceci est particulièrement utile pour la radio du robot et d'autres appareils électroniques tels que les coprocesseurs, qui peuvent ne pas être conçus pour une utilisation sur les plates-formes

mobiles. Isolateurs de vibrations, ressorts, mousses ou montage à matériaux flexibles tous peuvent réduire les forces de choc observées par ces composants

36.1.2 Redondance

Malheureusement, il y a peu d'endroits dans le système de contrôle FRC où la redondance est possible. Tirer parti des possibilités de redondance peut accroître la fiabilité. Un exemple est le raccordement de la prise a barillet d'alimentation de la radio en plus de la connexion de PoE fournie. Cela garantit que si l'un des câbles est endommagé ou délogé, l'autre maintiendra l'alimentation de la radio. Gardez un œil sur d'autres zones potentielles pour fournir la redondance lors du câblage et la programmation de votre robot.

36.1.3 Économiseur de ports

Pour toutes les connexions sur le Robot ou sur la station de pilotage qui peuvent être fréquemment branchées et débranchées (telles que les joysticks DS, DS Ethernet, attache USB au roboRIO et attache Ethernet) il est possible d'utiliser un « Économiseur de ports"" ce qui peut réduire considérablement le risque d'endommager le port des appareils. Ce type de dispositif peut servir à une double fonction, à la fois en réduisant le nombre de cycles que le port sur l'appareil électronique subit ainsi qu'en relocalisant la connexion à un endroit plus pratique. Assurez-vous de sécuriser l'Économiseur de port (voir l'élément suivant) pour éviter les dommages au port.

36.1.4 Gestion des câbles et attaches mécaniques

L'un des composants les plus essentiels à la fiabilité et à l'entretien des robots est une bonne gestion du câblage ainsi que des attaches mécaniques de ceux-ci . Une bonne gestion des fils est composée de quelques composants :

- Assurez-vous que les câbles sont de la bonne longueur. Toute longueur de câble excédentaire est plus difficile à gérer. Si vous utilisez un câble très long provenant d'un câblage prémonté d'un manufacturier, fixer le supplément dans un petit paquet à l'aide d'attaches de câble séparées avant de fixer le reste du câble.
- Assurez-vous que les câbles sont fixés à proximité des points de connexion, avec suffisamment de mou pour éviter de mettre la pression sur les connecteurs. C'est ce qu'on appelle le soulagement des forces de traction de câble et il est essentiel de minimiser la probabilité qu'un câble soit débranché ou qu'un fil se détache à un point de connexion (il s'agit généralement de concentrateurs de contrainte mécanique).
- Sécurisez les câbles à proximité de tous les composants en mouvement. Assurez-vous que tous les fils sont sécurisés et protégés contre les composants en mouvement, même si les composants en mouvement devaient se plier ou se déplacer sur une grande distance.
- Sécurisez les câbles à des points supplémentaires au besoin pour garder le câblage propre et en ordre. Prenez soin de ne pas trop attacher les fils ; si les fils sont fixés dans trop d'endroits, ceci peut rendre le dépannage et la maintenance plus difficile.

36.1.5 Documentation

Une excellente façon de faciliter l'entretien et le dépannage est de rédiger la documentation décrivant ce qui est connecté au robot et la localisation de chaque connexion. Il existe un certain nombre de façons de créer ce type de documentation pouvant consister en des diagrammes de câblage complets, à des tableaux excel ou une liste rapide des fonctions qui sont attachées à quels canaux. De nombreuses équipes intègrent également ces listes à l'étiquetage (voir la prochaine section).

Quand un fil est accidentellement coupé, qu'un mécanisme devient défectueux ou qu'un composant brûle, il sera beaucoup plus facile à effectuer la réparation si vous avez une certaine documentation pour vous indiquer ce qui est connecté où sans avoir à tracer le câblage tout au long (même si votre câblage est propre !)

36.1.6 Étiquetage

L'étiquetage est un excellent moyen de compléter la documentation de câblage décrite ci-dessus. Il existe de nombreuses stratégies différentes pour étiqueter le câblage et l'électronique, le tout avec leurs propres avantages et inconvénients. Les étiquettes pour l'électronique et les drapeaux pour les fils peuvent être faits à la main, ou en utilisant un appareil qui imprime les étiquettes (certains peuvent également utiliser des étiquettes thermo rétractable sur les câbles). Vous pouvez aussi utiliser différentes couleurs de ruban électrique ou des drapeaux d'étiquetage pour indiquer différentes choses. Quel que soit le système que vous choisissez, assurez-vous de comprendre comment il complète votre documentation et assurez-vous que tous les membres de votre équipe le connaisse.

36.1.7 Vérification des câbles et des branchements

Une fois que tout le câblage sur le robot est complet, assurez-vous de vérifier chaque connexion, en tirant sur chacune, pour s'assurer que tout est sécurisé. En outre, assurez-vous qu'aucun brin de fil errant « en cuivre nu » ne sorte d'un point de connexion et qu'aucune connexion non isolée soit exposée. Si des connexions se détachent lors des tests, ou tout brin de fil en cuivre nu soient découverts, refaites la connexion à nouveau et assurez-vous qu'une deuxième personne la vérifie une fois terminée.

Une cause commune qui crée de mauvaises connexions est le mauvais raccordement aux connecteurs à vis ou aux attaches à écrous et boulons. Pour tous ces types de connexion sur le robot (par exemple, les connexions de batterie, le disjoncteur principal, PDP, roboRIO), assurez-vous que les attaches soient bien serrées. Pour les connexions de style écrou et boulon, assurez-vous que le fil/terminal ne bouge pas lorsqu'on le tire ; si vous pouvez faire pivoter votre câble de batterie ou votre connexion de disjoncteur principal en saisissant le terminal et en le tirant, la connexion n'est pas assez serrée.

Une autre source commune de défaillances sont les fusibles sur le côté du PDP. Assurez-vous que ces fusibles sont complètement insérés ; vous devrez peut-être appliquer plus de force que vous le pensez pour les insérer complètement. Si les fusibles sont bien insérés, ils seront probablement difficiles ou impossibles à enlever à la main.

Les connexions enfichable Snap-in telles que le connecteur SB-50 doivent être sécurisées à l'aide de clips ou de attaches de câble pour s'assurer qu'elles ne se détachent pas pendant les impacts.

36.1.8 Revérifiez tôt et souvent

Revérifiez l'ensemble du système électrique aussi soigneusement que possible après avoir joué le premier match ou deux (ou faire des tests très vigoureux). Les premiers impacts que le robot subit peuvent desserrer les attaches ou exposer des problèmes.

Créez une liste de contrôle pour la vérification régulière des connexions électriques. Ä titre d'exemple, les fixations de rotation telles que les connexions de batterie et de PDP doivent être vérifiées tous les 1-3 matchs. Les connexions de type ressort tels que les connecteurs WAGO et Weidmuller n'ont probablement besoin d'être vérifiées qu'une fois par événement. Assurez-vous que l'équipe sait qui est responsable de remplir la liste de vérification et comment elle documentera ce qui est fait, ou à faire.

36.1.9 Entretien et manipulation de la batterie

Prenez bien soin de vos batteries ! Une mauvaise batterie peut facilement causer une défaillance majeure sur le robot pendant un match. Étiquetez toutes vos batteries pour vous aider à suivre l'utilisation pendant l'événement. De nombreuses équipes inscrivent également des informations telles que l'âge de la batterie sur cette étiquette.

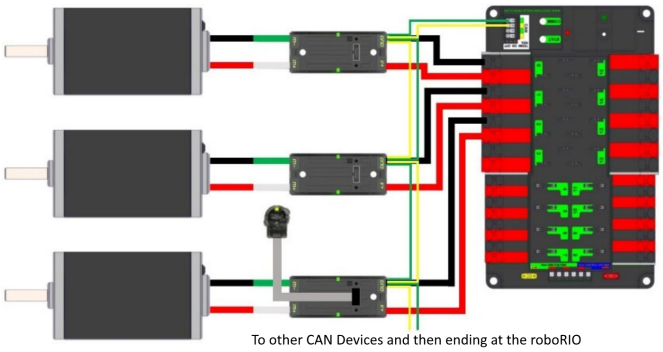
- Ne soulevez jamais ou ne transportez jamais la batterie par ses fils ! Le transport des batteries par les fils a le potentiel d'endommager la connexion interne entre les terminaux et les plaques, tout en augmentant considérablement la résistance interne et en va causer une dégradation de la performance du robot.
- Identifiez toute batterie qui a été échappée sur le sol jusqu'à ce qu'un test complet puisse être effectué. En plus des connexions terminales mentionnées, une batterie qui a été échappée au sol peut aussi contenir des cellules individuelles endommagées à l'interne. Ces dommages ne peuvent pas être décelé à l'aide d'un simple test de tension, mais seront vite détectés lorsque la batterie sera placée sous la charge.
- Faire la rotation de l'utilisation des batteries uniformément. Cela permet de s'assurer que les batteries ont le plus de temps pour se charger et se reposer et qu'elles se dégradent uniformément (nombre égal de cycles de charge/décharge)
- Charger les batteries uniformément et faites un essai de charge si possible pour surveiller l'état de santé. Il existe un certain nombre de produits disponibles dans le commerce que les équipes utilisent pour tester la charge des batteries, dont au moins une conçue spécifiquement pour FRC. Un test de charge peut fournir un indicateur de la santé de la batterie en mesurant la résistance interne. Cette mesure est beaucoup plus significative quand il s'agit de correspondre aux performances qu'un simple mesure de tension sans charge fourni par un multimètre.

36.1.10 Vérification des journaux DS (DS logs)

Après chaque match, passez en revue les journaux DS pour voir à quoi ressemble la tension de la batterie et l'utilisation du courant. Une fois que vous avez établi la consommation normale de ces éléments pour votre robot, vous pouvez être en mesure de repérer les problèmes potentiels (mauvaises batteries, moteurs défaillants, liaison mécanique) avant qu'ils ne deviennent des défaillances critiques.

36.2 Bases du câblage du réseau CAN

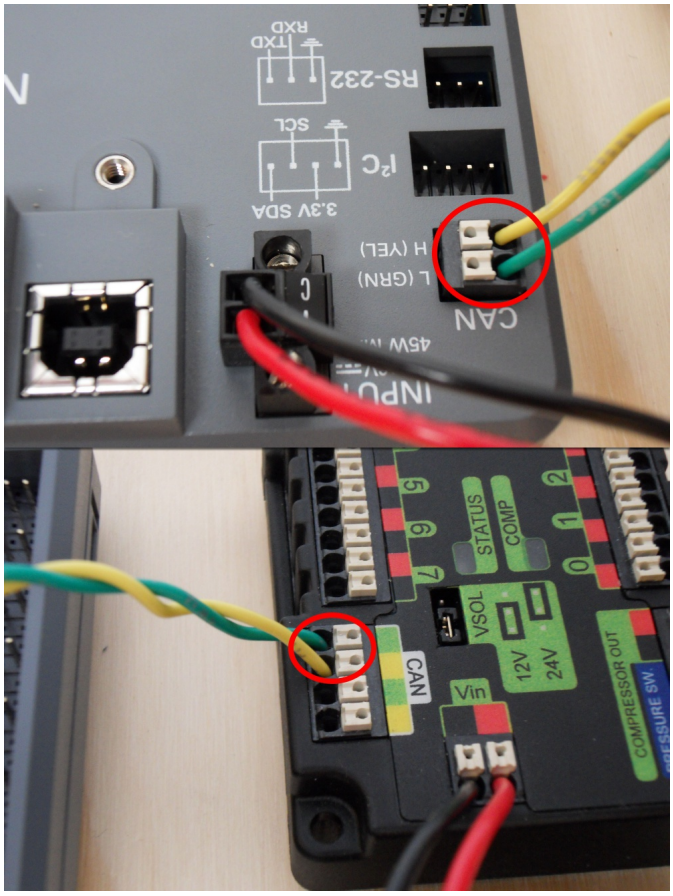
CAN is a two wire network that is designed to facilitate communication between multiple devices on your robot. It is recommended that CAN on your robot follow a « daisy-chain » topology. This means that the CAN wiring should usually start at your roboRIO and go into and out of each device successively until finally ending at the *PDP*.



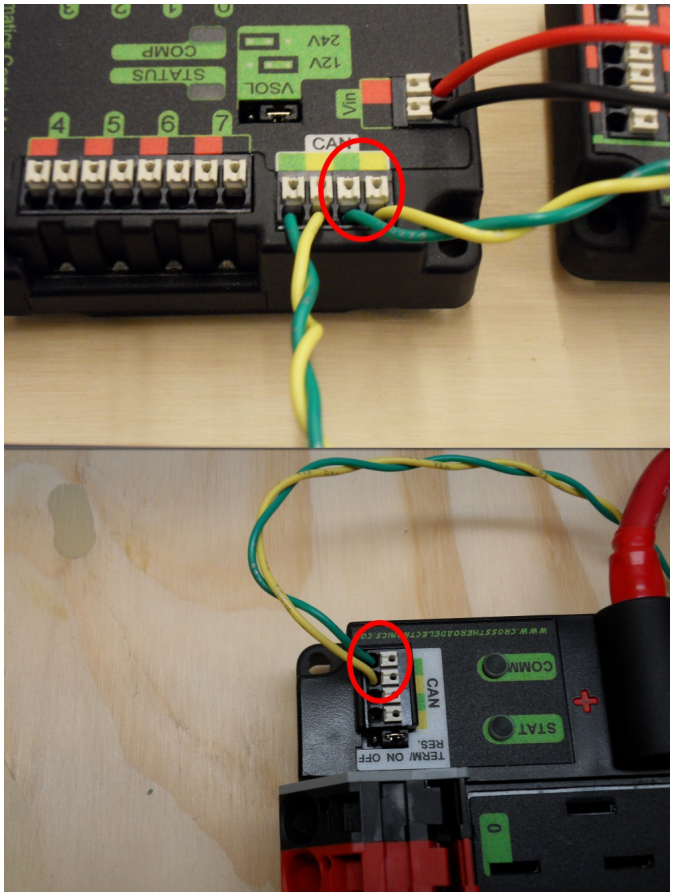
36.2.1 Câblage standard

Le CAN est généralement câblé avec du fil jaune et du fil vert avec le jaune agissant comme le signal CAN-High et le vert comme le signal CAN-Low. Beaucoup d'appareils présentent ce schéma de couleurs jaune et vert pour indiquer comment les fils doivent être branchés.

Câblage du réseau CAN du roboRIO au PCM.



Câblage du réseau CAN du PCM au PDP.



36.2.2 Borne

Il est recommandé que le câblage commence au roboRIO et se termine au PDP parce que le réseau CAN doit être terminé par des résistances de 120 Ω et celles-ci sont intégrées dans ces deux dispositifs. Le PDP est livré avec le jumper de la résistance de terminaison du réseau CAN en position « ON ». Il est recommandé de laisser le jumper dans cette position et de placer tous les nœuds CAN supplémentaires entre le roboRIO et le PDP (laissant le PDP comme terminaison du réseau CAN). Si vous souhaitez placer le PDP au milieu du réseau CAN (en utilisant les deux paires de terminaux du PDP CAN), déplacez le jumper à la position « OFF » et placez votre propre résistance de terminaison de 120 Ω à la fin de la chaîne de votre du réseau CAN.

36.3 Wiring Pneumatics - CTRE Pneumatic Control Module

This page describes wiring pneumatics with the CTRE Pneumatic Control Module (PCM). For instructions on wiring pneumatics with the REV Pneumatic Hub (PH) see *this page*.

Indication : For pneumatics safety & mechanical requirements, consult this year's Robot Construction rules. For mechanical design guidelines, the FIRST Pneumatics Manual is located here

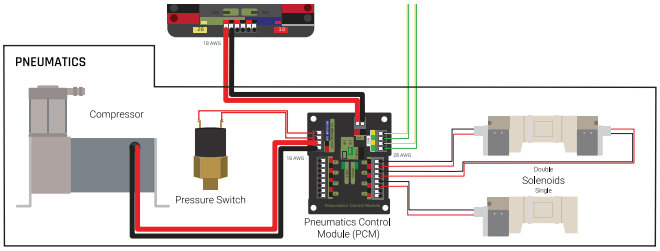
---

36.3.1 Wiring Overview

A single PCM will support most pneumatics applications, providing an output for the compressor, input for the pressure switch, and outputs for up to 8 solenoid channels (12V or 24V selectable). The module is connected to the roboRIO over the *CAN* bus and powered via 12V from the PDP or PDH.

For complicated robot designs requiring more channels or multiple solenoid voltages, additional PCMs or PHs can be added to the control system.

36.3.2 PCM Power and Control Wiring



The first PCM on your robot can be wired from the PDP VRM/PCM connectors on the end of the PDP or from a 15 amp or 20 amp port on the PDH (20 amp recommended if controlling a compressor). The PCM is connected to the roboRIO via CAN and can be placed anywhere in the middle of the CAN chain (or on the end with a custom terminator). For more details on wiring a single PCM, see *Alimentation des composantes pneumatiques (optionnel)*

Additional PCMs can be wired to a standard WAGO connector on the side of the PDP and protected with a 20A or smaller circuit breaker. Additional PCMs should also be placed anywhere in the middle of the CAN chain.

36.3.3 The Compressor

The compressor can be wired directly to the Compressor Out connectors on the PCM. If additional length is required, make sure to use 18 AWG wire or larger for the extension.

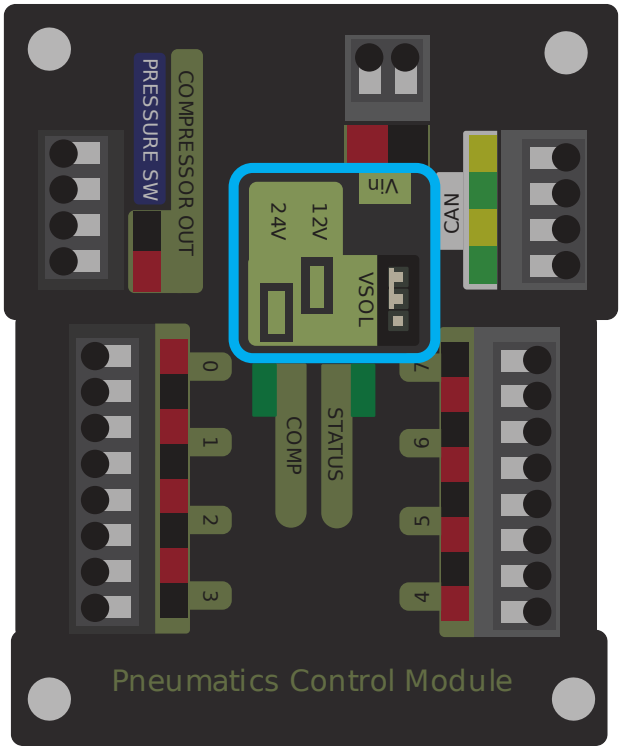
36.3.4 The Pressure Switch

The pressure switch should be connected directly to the pressure switch input terminals on the PCM. There is no polarity on the input terminals or on the pressure switch itself, either terminal on the PCM can be connected to either terminal on the switch. Ring or spade terminals are recommended for the connection to the switch screws (note that the screws are slightly larger than #6, but can be threaded through a ring terminal with a hole for a #6 screw such as the terminals shown in the image).

36.3.5 Solenoids

Each solenoid channel should be wired directly to a numbered pair of terminals on the PCM. A single acting solenoid will use one numbered terminal pair. A double acting solenoid will use two pairs. If your solenoid does not come with color coded wiring, check the datasheet to make sure to wire with the proper polarity.

36.3.6 Solenoid Voltage Jumper



The PCM is capable of powering either 12V or 24V solenoids, but all solenoids connected to a single PCM must be the same voltage. The PCM ships with the jumper in the 12V position as shown in the image. To use 24V solenoids move the jumper from the left two pins (as shown in the image) to the right two pins. The overlay on the PCM also indicates which position corresponds to which voltage. You may need to use a tool such as a small screwdriver, small pair of pliers, or a pair of tweezers to remove the jumper.

36.4 Wiring Pneumatics - REV Pneumatic Hub

This page describes wiring pneumatics with the REV Pneumatic Hub (*PH*). For instructions on wiring pneumatics with the CTRE Pneumatic Control Module (*PCM*) see *this page*.

Indication : For pneumatics safety & mechanical requirements, consult this year's Robot Construction rules. For mechanical design guidelines, the FIRST Pneumatics Manual is located here

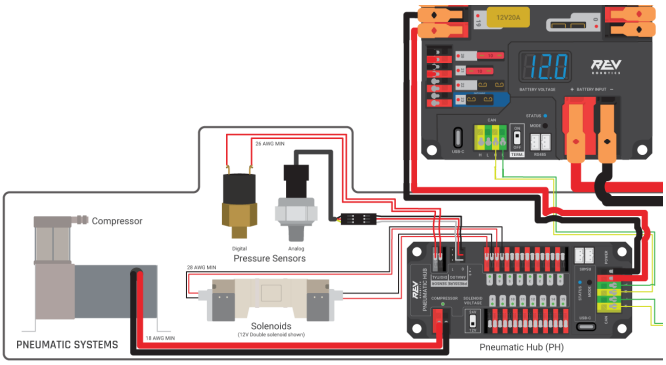
---

36.4.1 Wiring Overview

A single PH will support most pneumatics applications, providing an output for the compressor, input for a pressure switch, and outputs for up to 16 solenoid channels (12V or 24V selectable). The module is connected to the roboRIO over the *CAN* bus and powered via 12V from the PDP/PDH.

For complicated robot designs requiring more channels or multiple solenoid voltages, additional PHs or PCMs can be added to the control system.

36.4.2 PCM Power and Control Wiring



The first PH on your robot can be wired from the PDP VRM/PCM connectors on the end of the PDP or from a 15A or 20A port on the PDH (20 amp recommended if controlling a compressor). The PH is connected to the roboRIO via CAN and can be placed anywhere in the middle of the CAN chain (or on the end with a custom terminator). For more details on wiring a single PCM, see *Alimentation des composantes pneumatiques (optionnel)*

Additional PHs can be wired to a standard WAGO connector on the side of the PDP and protected with a 20A or smaller circuit breaker or to a 15A port on the PDH. Additional PHs should also be placed anywhere in the middle of the CAN chain.

36.4.3 The Compressor

The compressor can be wired directly to the Compressor connectors on the PH. If additional length is required, make sure to use 18 AWG wire or larger for the extension.

36.4.4 The Pressure Switch

The PH has two options for detecting pressure, a digital pressure switch, or an analog pressure switch.

Digital

A digital pressure switch should be connected directly to the digital pressure sensor input terminals on the PCM. There is no polarity on the input terminals or on the pressure switch itself, either terminal on the PH can be connected to either terminal on the switch. Ring or spade terminals are recommended for the connection to the switch screws (note that the screws are slightly larger than #6, but can be threaded through a ring terminal with a hole for a #6 screw such as the terminals shown in the image).

Analog

An analog pressure switch (REV-11-1107 can be connected directly to the analog pressure sensor port 0 input terminals. Using an analog pressure sensor allows reading the pressure in the pneumatic system through code and setting custom trigger thresholds for turning on and off the compressor.

Avertissement : The Analog Pressure Sensor port is a very tight fit and requires special attention. See REV Wiring an Analog Pressure Sensor for more tips

36.4.5 Solenoids

Each solenoid channel should be wired directly to a numbered pair of terminals on the PH. A single acting solenoid will use one numbered terminal pair. A double acting solenoid will use two pairs. If your solenoid does not come with color coded wiring, check the datasheet to make sure to wire with the proper polarity.

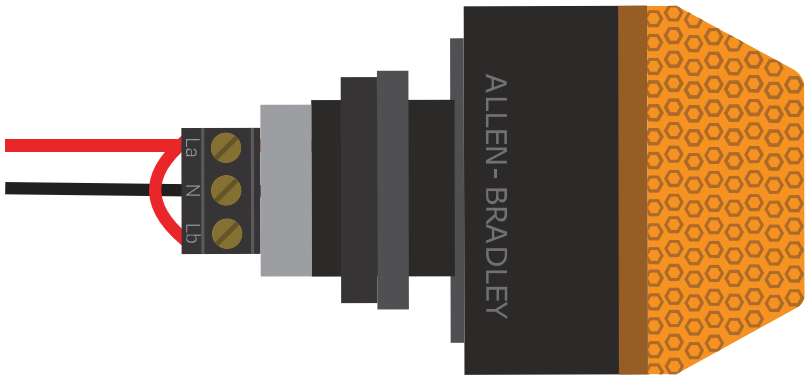
36.4.6 Solenoid Voltage Switch

The PH is capable of powering either 12V or 24V solenoids, but all solenoids connected to a single PH must be the same voltage. Set the voltage switch to the appropriate voltage for solenoids prior to use.

36.5 Signification des témoins lumineux

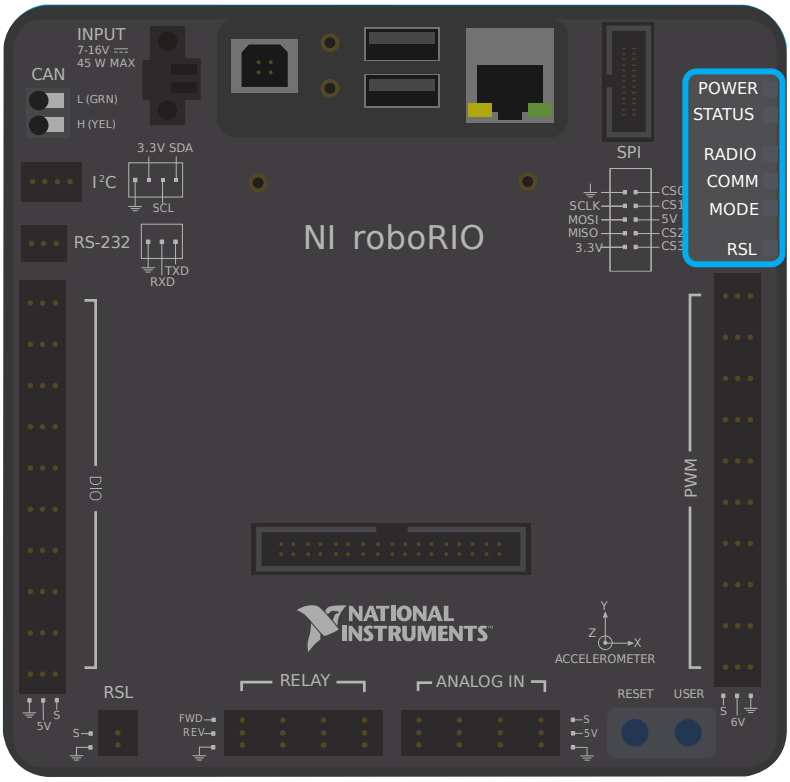
De nombreux composants du système de contrôle FRC® disposent de voyants lumineux qui peuvent être utilisés pour diagnostiquer rapidement les problèmes avec votre robot. Ce guide présente chacun des composants matériels et décrit la signification des voyants. Photos et informations d'Innovation FIRST et Cross the Road Electronics.

36.5.1 Témoin diagnostique (RSL)



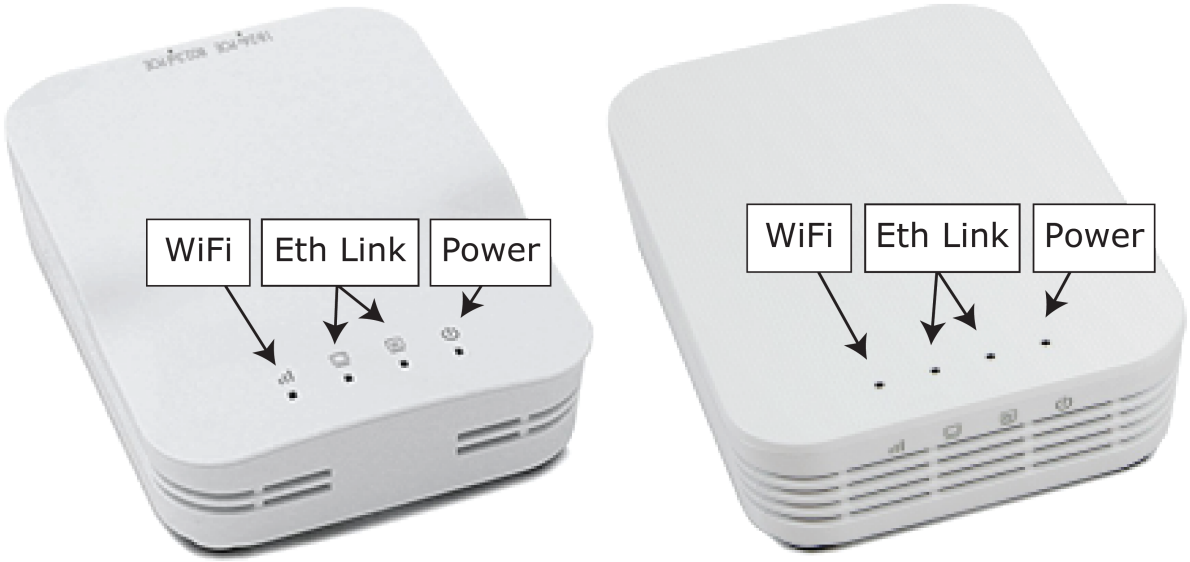
Allumé	Robot alimenté et désactivé
Allumé clignotant	Robot alimenté et activé
Éteint	Robot non alimenté, roboRIO non alimenté ou RSL mal câblée

36.5.2 roboRIO



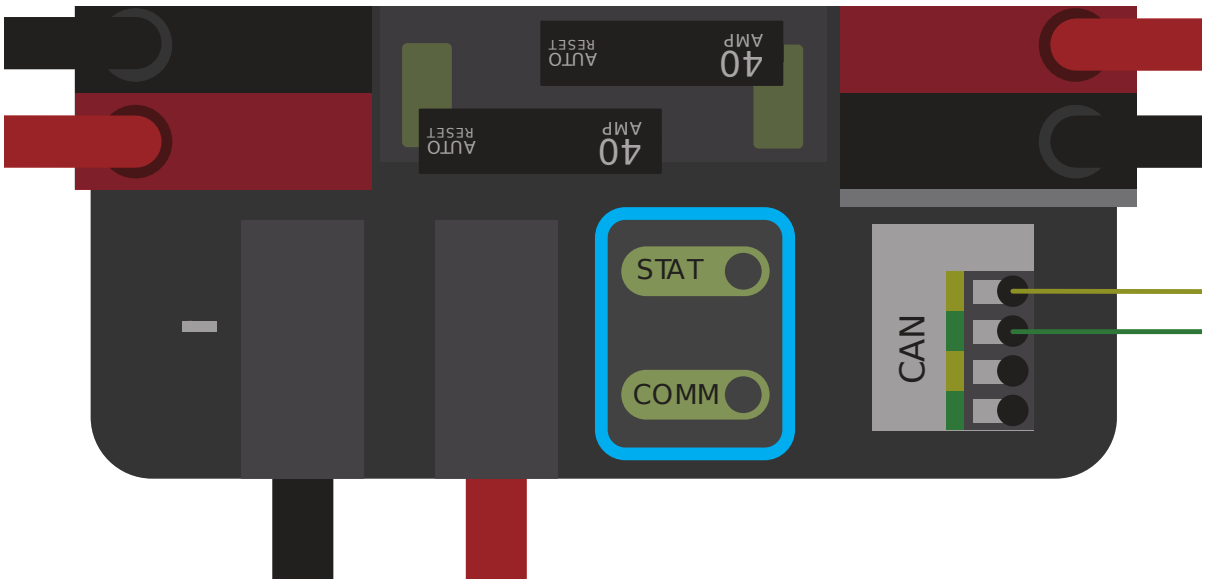
Power	Vert	Alimentation OK
	Ambre	Protection contre les sous-tensions déclenchée, sorties désactivées
	Rouge	Faute d'alimentation, vérifier les connexions pour un court-circuit
Status	Allumée pendant que le contrôleur démarre, devrait s'éteindre ensuite	
	2 clignotements	Erreur logicielle, ré-imager le roboRIO
	3 clignotements	Mode sans échec, redémarrer le roboRIO, ré-imager si ce n'est pas résolu
	4 clignotements	Le logiciel a planté deux fois sans redémarrer, redémarrer le roboRIO, ré-imager si ce n'est pas résolu
	Clignotement constant ou reste allumée	Erreur irrécupérable
Radio	N'est pas implémenté pour le moment	
Comm	Éteint	Aucune communication
	Rouge	Communication avec le Driver Station, mais pas de code utilisateur en cours d'exécution
	Rouge clignotant	Arrêt d'urgence déclenché
	Vert	Bonne communication avec le Driver Station
Mode	Éteint	Sorties désactivées (robot désactivé, sous-tension, etc.)
	Orange	Autonome activé
	Vert	Téléopération activée
	Rouge	Test activé
RSL	*Voir plus haut*	

36.5.3 Radio OpenMesh



Power	Bleu	Allumée ou en démarrage
	Bleu clignotant	En démarrage
Eth Link	Bleu	Lien effectué
	Bleu clignotant	Trafic présent
WiFi	Éteint	Mode passerelle, non-lié ou firmware non-FRC
	Rouge	Point d'accès, non-lié
	Jaune/orange	Point d'accès, lié
	Vert	Mode passerelle, lié

36.5.4 Panneau de distribution de puissance (PDP)



Témoins Status/Comm du PDP

Témoin	Clignotant	Lent
Vert	Aucune erreur - robot activé	Aucune erreur - robot désactivé
Orange	N/A	Erreur
Rouge	N/A	Aucune communication CAN

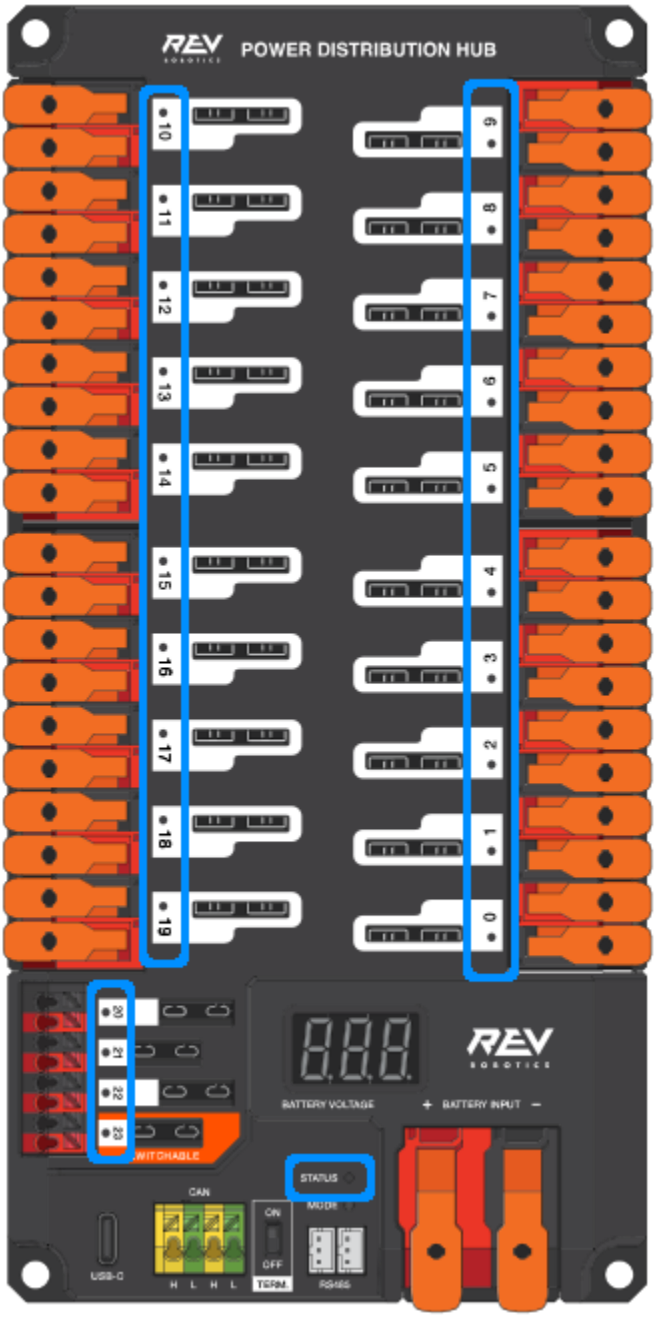
Astuce : Si un indicateur du PDP affiche plus d'une couleur, voir le tableau des états particuliers des témoins du PDP ci-dessous. Pour plus d'informations sur la résolution des problèmes avec le PDP, consulter le manuel utilisateur du PDP.

Note : Note that the No *CAN* Comm fault will occur if the PDP cannot communicate with the roboRIO via CAN Bus.

États particuliers des témoins du PDP

Couleurs des témoins	Problème
Rouge/orange	Matériel endommagé
Vert/orange	En démarrage
Aucun témoin	Aucune alimentation/polarité incorrecte

36.5.5 Concentrateur de distribution d'alimentation



Note : Ces modèles de voyants ne s'appliquent qu'à la version 21.1.7 et ultérieure du micrologiciel

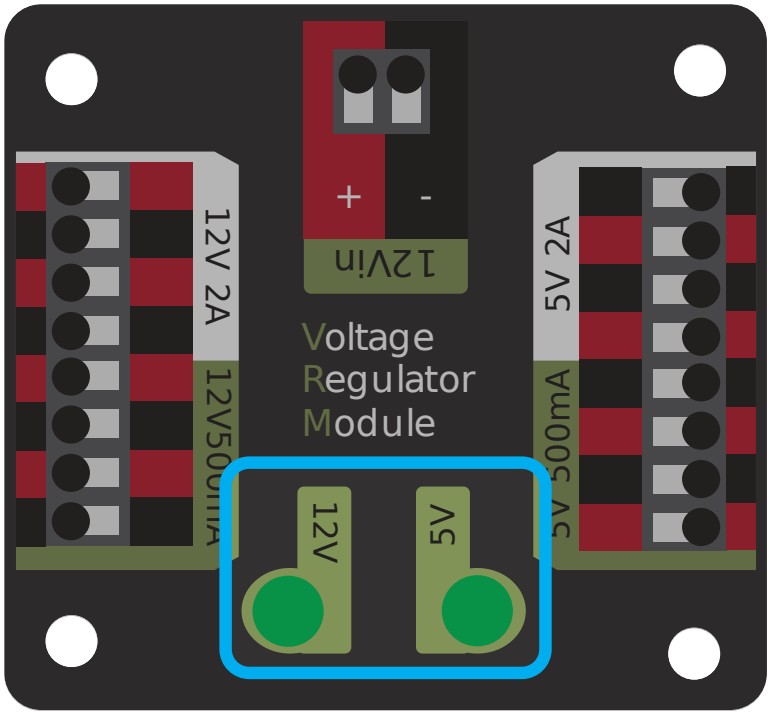
36.5.6 Voyant d'état PDH

Couleur de la DEL	Statut
Bleu	Appareil allumé mais aucune communication établie
Vert	Communication principale avec roboRIO établie
Magenta clignotant	Délai d'attente de maintien en vie
Turquoise	Heartbeat secondaire (connecté au client matériel REV)
Clignotant Orange/Bleu	Faible batterie
Clignotant Orange/Jaune	Erreur sur le bus CAN
Clignotant Orange/Turquoise	Faute du Matériel
Clignotant Orange/Rouge	Fail Safe
Clignotant Orange/Magenta	Erreur de surintensité de courant

DELs pour canaux

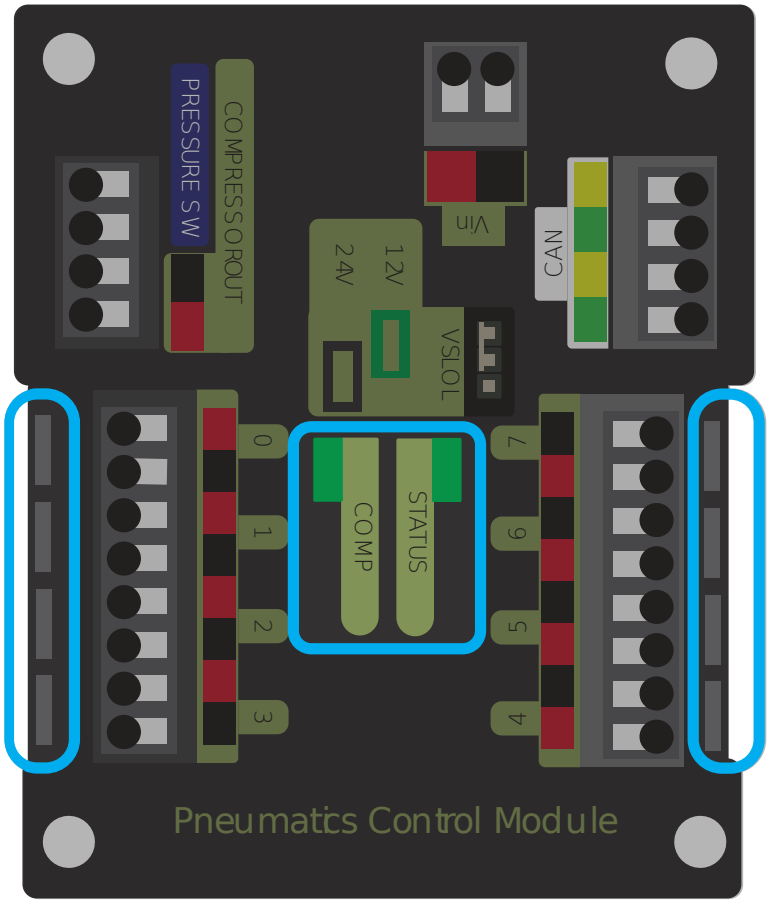
Couleur de la DEL	Statut
Éteint	Le canal est sous tension et fonctionne comme prévu
Rouge	Le canal n'a PAS de tension et il y a un défaut actif. Vérifiez s'il y a un disjoncteur/fusible déclenché ou manquant
Rouge clignotant	Défaut persistant sur le canal. Vérifiez s'il y a un disjoncteur/fusible déclenché.

36.5.7 Régulateur de tension (VRM)



Les témoins sur le VRM indiquent l'état des deux alimentations électriques. Si les alimentations fonctionnent de manière adéquate, les deux témoins devraient être allumés en vert. Si les témoins sont faibles ou éteints, la sortie pourrait avoir un court-circuit ou tirer trop de courant.

36.5.8 Module de contrôle pneumatique (PCM)



Témoin du PCM

Témoin	Clignotant	Lent	Long
Vert	Aucune erreur - robot activé	Erreur	N/A
Oran	N/A	Erreur	N/A
Roug	N/A	Aucune communication CAN ou erreur d'un solénoïde (la lumière propre au solénoïde clignotera)	Erreur du compresseur

Astuce : Si un indicateur du PCM affiche plus d'une couleur, voir le tableau des états particuliers des témoins du PCM ci-dessous. Pour plus d'informations sur la résolution des problèmes avec le PCM, consulter le manuel utilisateur du PCM.

Note : Noter que l'erreur No CAN Comm ou Aucune communication CAN ne s'affichera pas si le PCM ne peut communiquer avec aucune autre composante, si le PCM et le PDP peuvent communiquer entre eux, mais pas le roboRIO

États particuliers des témoins du PCM

Témoin	Problèmes
Rouge/orange	Matériel endommagé
Vert/orange	En démarrage
Aucun témoin	Pas d'alimentation/polarité incorecte

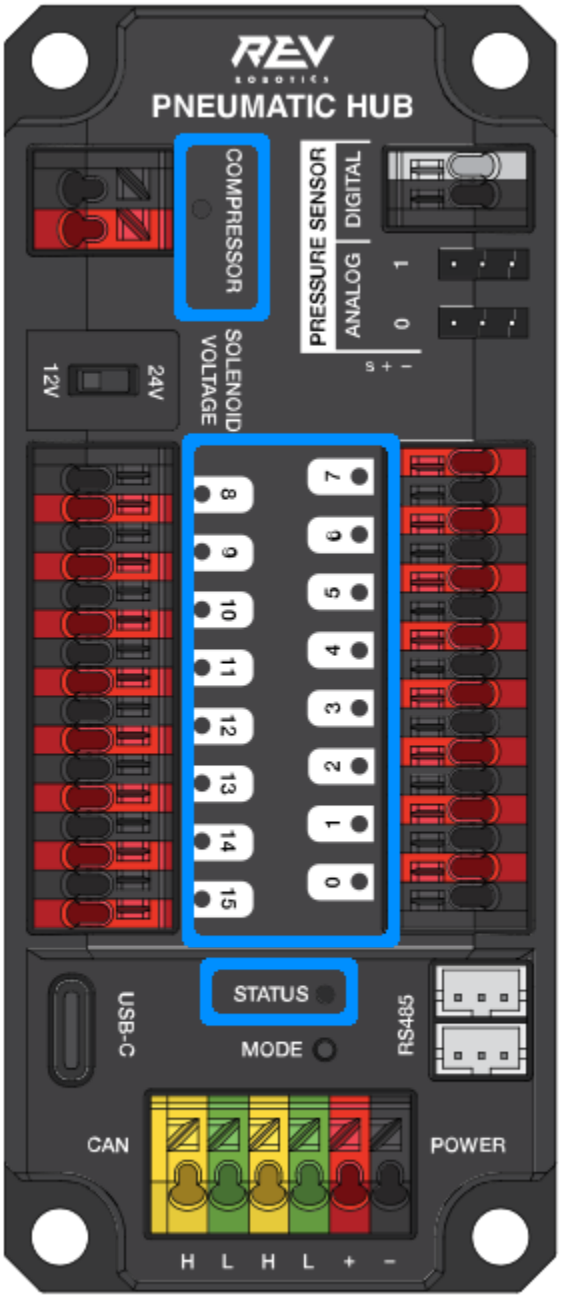
Témoin Comp du PCM

Ce témoin est le témoin du compresseur. Celui-ci sera vert lorsque la sortie du compresseur est active (compresseur alimenté) et éteint lorsque la sortie du compresseur n'est pas active

Témoins des canaux solénoïdes du PCM

Ces témoins sont allumés rouge si le canal du solénoïde est alimenté et éteints s'il n'est pas alimenté.

36.5.9 Concentrateur pneumatique



Note : Ces modèles de voyants ne s'appliquent qu'à la version 21.1.7 et ultérieure du micrologiciel

Voyant de statut PH

Couleur de la DEL	Statut
Bleu	Appareil allumé mais aucune communication établie
Vert	Communication principale établie
Magenta clignotant	Délai d'attente de maintien en vie
Turquoise	Heartbeat secondaire (connecté au client REV HW)
Clignotant Orange/Bleu	Faute du Matériel
Clignotant Orange/Jaune	Erreur sur le bus CAN
Clignotant Orange/Rouge	Fail Safe
Clignotant Orange/Magenta	Erreur de surintensité de courant
Clignotant Orange/Vert	Clignotant Orange/Vert

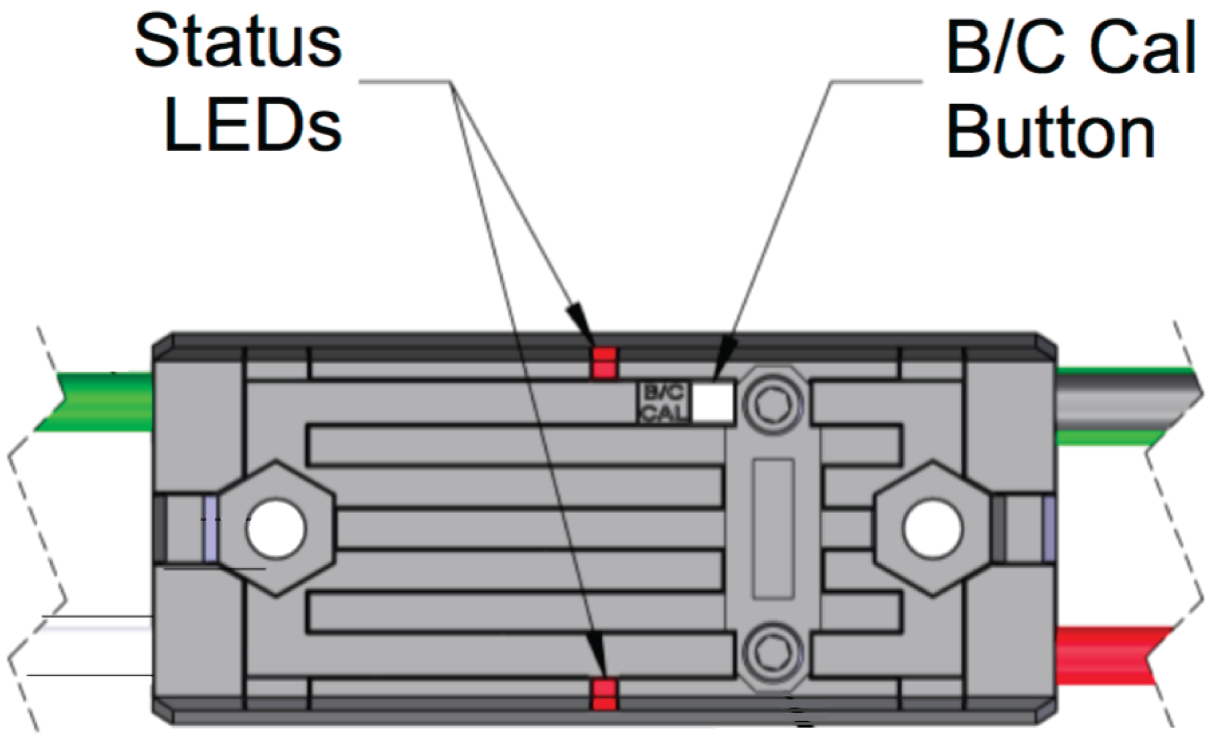
DEL du compresseur

Couleur de la DEL	Statut
Vert	Compresseur activé
Noire	Compresseur arrêté

DEL du solénoïde

Couleur de la DEL	Statut
Vert	Solénoïde activé
Noire	Solénoïde désactivé

36.5.10 Contrôleurs de moteurs Talon SRX , Victor SPX & Talon FX



Témoins en opération normale

Témoins	Couleurs	État du Talon SRX
Les deux	Vert clignotant	Signal positif appliqué. Vitesse du clignotement proportionnelle au rapport cyclique.
Les deux	Rouge clignotant	Signal négatif appliqué. Vitesse du clignotement proportionnelle au rapport cyclique.
Aucun	Aucun	Aucune alimentation n'est connectée au Talon SRX
En alternance	Éteint/orang	Bus CAN détecté, robot désactivé
En alternance	Éteint/rouge lent	Bus CAN/PWM non-détecté
En alternance	Éteint/rouge rapide	Erreur détectée
En alternance	Rouge/orang	Matériel endommagé
Témoins clignotant vers (M-)	Éteint/rouge	Interrupteur de fin de course avant ou limite logicielle avant
Témoins clignotant vers (M+)	Éteint/rouge	Interrupteur de fin de course arrière ou limite logicielle arrière
Témoin 1 seulement (plus près de M+/V+)	Vert/orange	En démarrage
Témoins clignotant vers (M+)	Éteint/orang	Défaut thermique / Arrêt (Talon FX Seulement)

Témoins durant la calibration

Témoins	État du Talon SRX
Rouge et vert clignotant	Mode de calibration
Vert clignotant	Calibration effectuée avec succès
Rouge clignotant	Calibration échouée

Témoin B/C CAL

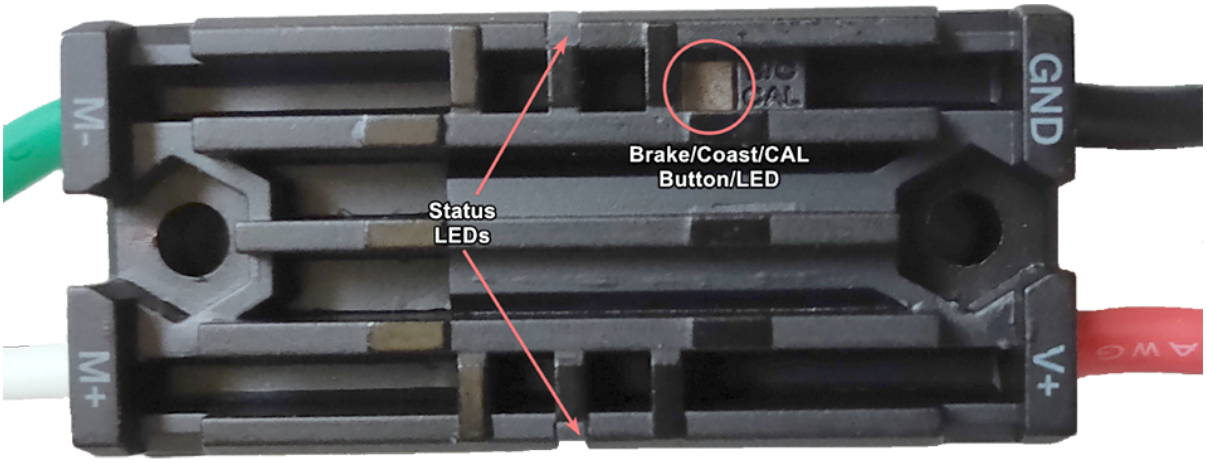
Couleur du témoin B/C CAL	État du Talon SRX
Rouge	Mode frein
Éteint	Mode roue libre

36.5.11 Contrôleur de moteur SPARK-MAX

Operating Mode	Idle Mode	State	Color/Pattern	
Brushed	Brake	No Signal	Blue Blink	
		Valid Signal	Blue Solid	
	Coast	No Signal	Yellow Blink	
		Valid Signal	Yellow Solid	
Brushless	Brake	No Signal	Cyan Blink	
		Valid Signal	Cyan Solid	
	Coast	No Signal	Magenta Blink	
		Valid Signal	Magenta Solid	
Partial Forward	-	-	Green Blink	
Full Forward	-	-	Green Solid	
Partial Reverse	-	-	Red Blink	
Full Reverse	-	-	Red Solid	
Forward Limit	-	-	Green/White Blink	
Reverse Limit	-	-	Red/White Blink	
Firmware Update Mode	-	-	Dark (LED off)	
Fault Conditions				
12V Missing	-	-	Orange/Blue Slow Blink	
Brushless Encoder Error	-	-	Orange/Magenta Slow Blink	
Gate Driver Fault	-	-	Orange/Cyan Slow Blink	
CAN Fault	-	-	Orange/Yellow Slow Blink	

36.5.12 SPARK (REV Robotics)

36.5.13 Contrôleur de moteur Victor-SP

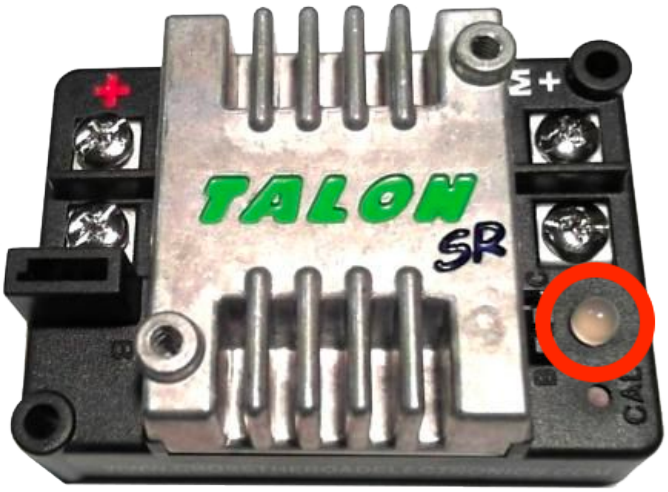


Bouton/témoin frein/roue libre/cal - rouge si le contrôleur est en mode frein, éteint si le contrôleur est en mode roue libre

Statut

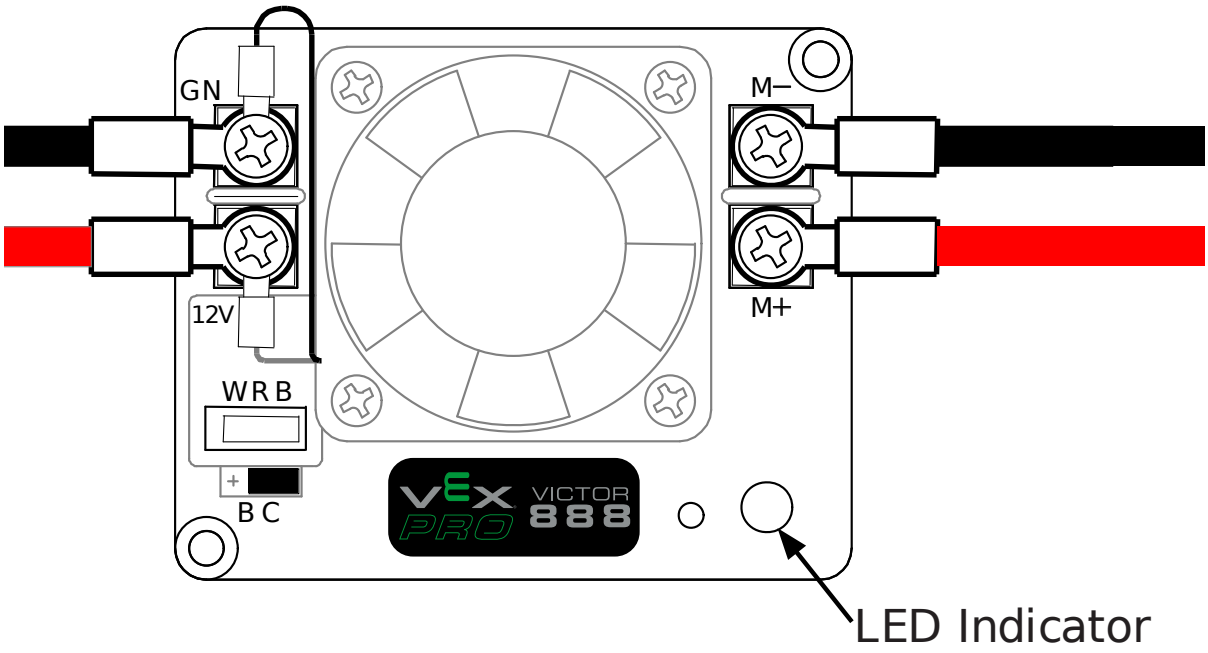
Vert	Solide	Avant (pleine puissance)
	Allumé clignotant	Proportionnel au voltage de sortie (avant)
Rouge	Solide	Arrière (pleine puissance)
	Allumé clignotant	Proportionnel au voltage de sortie (avant)
Orang	Solide	Robot FRC désactivé, signal PWM perdu ou signal dans la zone morte (+/- 4%)
Rouge	Allumé clignotant	Prêt pour la calibration. Plusieurs clignotements verts indiquent que la calibration s'est effectuée avec succès et plusieurs clignotements rouges indiquent que la calibration a échoué.

36.5.14 Contrôleur de moteur Talon



Vert	Solide	Avant (pleine puissance)
	Allumé clignotant	Proportionnel au voltage de sortie (avant)
Rouge	Solide	Arrière (pleine puissance)
	Allumé clignotant	Proportionnel au voltage de sortie (arrière)
Orange	Solide	Aucun appareil CAN n'est connecté
	Allumé clignotant	Désactivé, signal PWM perdu, robot FRC désactivé ou signal dans la zone morte (+/- 4%)
Éteint		Aucune alimentation au Talon
Rouge	Clignotant	Prêt pour la calibration. Plusieurs clignotements verts indiquent que la calibration s'est effectuée avec succès et plusieurs clignotements rouges indiquent que la calibration a échoué.

36.5.15 Contrôleur de moteur Victor888

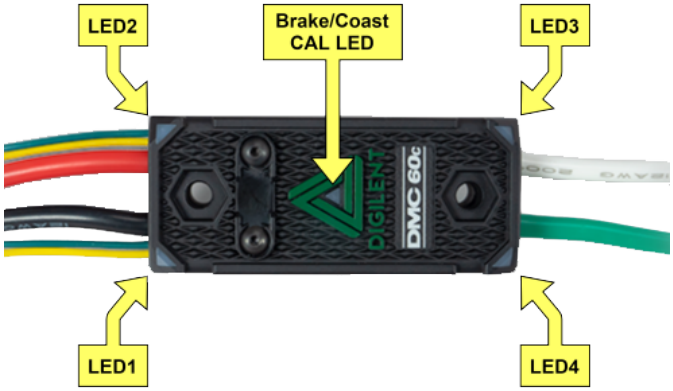


Vert	Solide	Avant (pleine puissance)
	Allumé clignotant	Calibration effectuée avec succès
Rouge	Solide	Arrière (pleine puissance)
	Allumé clignotant	Calibration échouée
Orange	Solide	Neutre/frein
Rouge/vert	Allumé clignotant	Mode de calibration

36.5.16 Contrôleur de moteur Jaguar

État des témoins	Statut du module
Conditions de fonctionnement normales	
Jaune	Neutre (vitesse à 0)
Vert clignotant rapidement	Avant
Rouge clignotant rapidement	Arrière
Vert	Avant (pleine vitesse)
Rouge	Arrière (pleine vitesse)
États d'erreur	
Jaune clignotant lentement	Perte du signal PWM ou réseau
Jaune clignotant rapidement	ID CAN invalide
Rouge clignotant lentement	Erreur de voltage, température ou d'interrupteur de fin de course
Rouge et jaune clignotant lentement	État d'erreur
Calibration ou état CAN	
Rouge et vert clignotant	Mode de calibration activé
Rouge et jaune clignotant	Échec de la calibration
Vert et jaune clignotant	Succès de la calibration
Vert clignotant lentement	Mode d'assignation de l'ID CAN
Jaune clignotant rapidement	ID CAN (compter le nombre de clignotements pour déterminer l'ID CAN)
Jaune clignotant	ID CAN invalide (programmée à 0) en attente d'une ID valide

36.5.17 Digilent DMC-60



Le DMC60C contient quatre témoins RVB et un témoin Brake/Coast CAL. Les quatre témoins RVB sont situés dans les coins et sont utilisés pour indiquer le statut pendant l'opération normale ainsi que lorsqu'une erreur se produit. Le témoin Brake/Coast CAL est positionné au centre du triangle et est utilisé pour indiquer le paramètre frein/roue libre actuel. Lorsque le témoin central est éteint, le contrôleur opère en mode roue libre. Lorsque le témoin est allumé, le contrôleur opère en mode frein. Il est possible d'alterner entre ces deux modes en pressant puis relâchant le centre du triangle.

Lors de l'alimentation, les témoins affichent du bleu, devenant continuellement plus lumineux. Cette séquence dure approximativement cinq secondes. Pendant ce démarrage, le contrôleur ne répondra pas à aucun signal d'entrée et les sorties seront désactivées. Après que le démarrage soit complété, le contrôleur est prêt à être utilisé normalement et ce qui est affiché sur les témoins est une fonction du signal d'entrée appliqué ainsi que des erreurs présentes. Assumant qu'aucune erreur ne s'est produite, les témoins afficheront ceci :

Signal PWM appliqué	État des témoins
Aucun signal en entrée ou signal PWM invalide	Les témoins du haut (1 et 2) et ceux du bas (3 et 4) sont illuminés rouge ou éteints en alternance.
Signal PWM neutre	Les 4 témoins sont illuminés orange.
Signal PWM positif	Les témoins clignotent en vert de manière horaire (témoins 1 → 2 → 3 → 4 → 1). La vitesse à laquelle ce cycle se produit est proportionnel au rapport cyclique de la sortie et devient plus rapide lorsque le rapport cyclique augmente. À un rapport cyclique de 100%, les 4 témoins sont illuminés en vert.
Signal PWM négatif	Les témoins clignotent en rouge de manière anti-horaire (témoins 1 → 4 → 3 → 2 → 1). La vitesse à laquelle ce cycle se produit est proportionnel au rapport cyclique de la sortie et devient plus rapide lorsque le rapport cyclique augmente. À un rapport cyclique de 100%, les 4 témoins sont illuminés en rouge.

État du contrôle par bus CAN	État des témoins
Aucun signal en entrée ou erreur du bus CAN détectée	Les témoins du haut (1 et 2) et ceux du bas (3 et 4) sont illuminés rouge ou éteints en alternance.
Aucune trame de données CAN reçue dans les dernières 100ms ou la dernière trame de données spécifiait mode-NoDrive (sortie désactivée)	Les témoins du haut (1 et 2) et ceux du bas (3 et 4) sont illuminés orange ou éteints en alternance.
Trame de données CAN valide reçue dans les dernières 100ms. Le mode de contrôle spécifié a résulté en l'application d'un rapport cyclique neutre à la sortie	Les 4 témoins sont illuminés orange.
Trame de données CAN valide reçue dans les dernières 100ms. Le mode de contrôle spécifié a résulté en l'application d'un rapport cyclique positif à la sortie	Les témoins clignotent en vert de manière horaire (témoins 1 → 2 → 3 → 4 → 1). La vitesse à laquelle ce cycle se produit est proportionnel au rapport cyclique de la sortie et devient plus rapide lorsque le rapport cyclique augmente. À un rapport cyclique de 100%, les 4 témoins sont illuminés en vert.
Trame de données CAN valide reçue dans les dernières 100ms. Le mode de contrôle spécifié a résulté en l'application d'un rapport cyclique négatif à la sortie	Les témoins clignotent en rouge de manière anti-horaire (témoins 1 → 4 → 3 → 2 → 1). La vitesse à laquelle ce cycle se produit est proportionnel au rapport cyclique de la sortie et devient plus rapide lorsque le rapport cyclique augmente. À un rapport cyclique de 100%, les 4 témoins sont illuminés en rouge.

Codes couleur d'erreur

Lorsqu'une erreur est détectée, le rapport cyclique en sortie est réduit à 0% et l'erreur est signalée. La sortie reste ensuite désactivée pour 3 secondes. Les témoins 1 à 4 sont utilisés pour indiquer l'état d'erreur. L'état d'erreur est indiqué en alternant entre les témoins du haut (1 et 2) et ceux du bas (3 et 4) étant illuminés rouge ou éteints. La couleur des témoins du bas dépend de l'erreur détectée. Le tableau ci-dessous décris à quelle erreur correspondent les différentes couleurs des témoins du bas.

Couleur	Surchauffe	Sous-tension
Vert	Allumé	Éteint
Bleu	Éteint	Allumé
Cyan / aqua	Allumé	Allumé

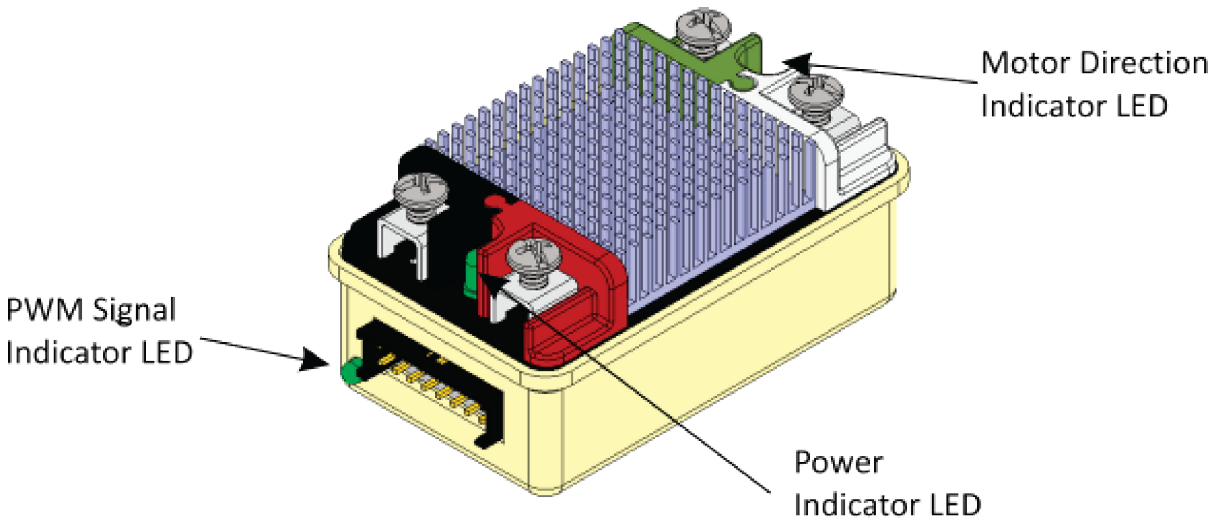
Mode frein/roue libre

Lorsque le témoin du centre est éteint, le contrôleur opère en mode roue libre. Lorsque le témoin du centre est illuminé, le contrôleur opère en mode frein. Le mode peut être changé en pressant le centre du triangle puis en le relâchant.

36.5.18 Contrôleur de moteur Venom

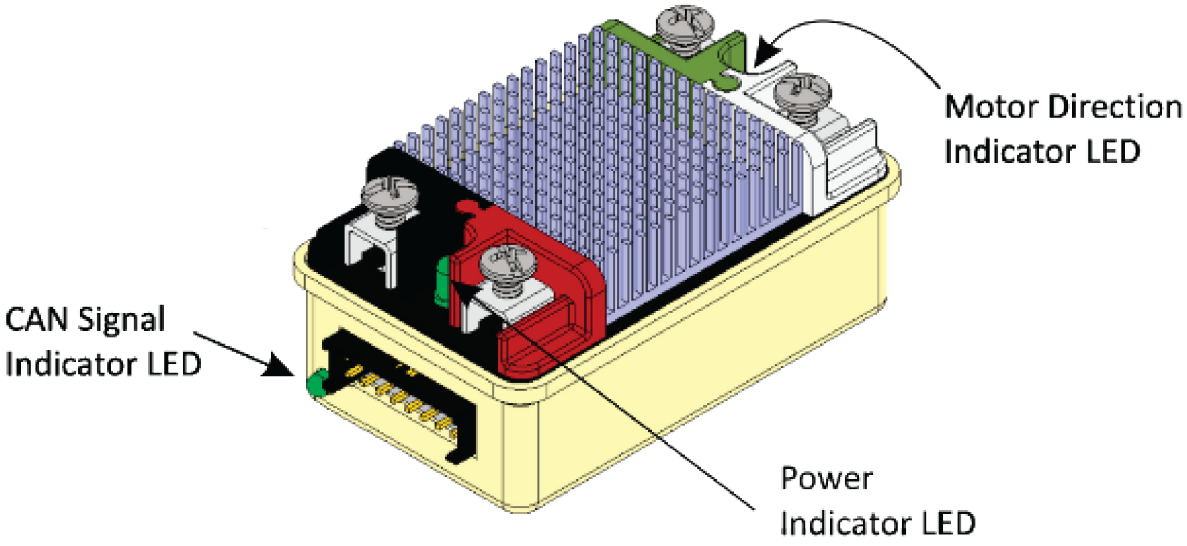
LED Pattern	Description
	Venom is initializing. This state should last less than 40ms after power up.
	The 'Identify Device' feature is active. This pattern in used to locate a particular Playing With Fusion device when multiple are installed on a robot. See the Motor Configuration section for more information
	Venom is initialized and in PWM mode. Waiting for a valid 1.0 to 2.0 ms PWM pulse.
	Venom successfully entered a valid CAN or PWM control mode. No Faults are active and motion may be commanded
	Venom is initialized and detected a valid CAN bus
	CAN communication fault. Check harness connections and but termination
	Missing heartbeat in CAN control mode. Ensure device ID matches device ID used by CANVenom class. See the Motor Configuration section for more information and instructions to change/verify the device ID
	Lead motor heartbeat is missing while in Follow The Leader mode.
	The lead motor ID is same as the motor ID. One Venom cannot follow itself. Ensure the leader and follower have different IDs
	An invalid control mode was specified by the roboRIO. This should not occur when using PlayingWithFusionDriver. Contact PWF Technical support.
	Another Venom with the same device ID was detected on the CAN bus. All Venom device IDs must be unique
	The forward limit switch is enabled and is active
	The reverse limit switch is enabled and is active
	Motor temperature is too high
	Average motor current is too high

36.5.19 Mindsensors SD540B (PWM)



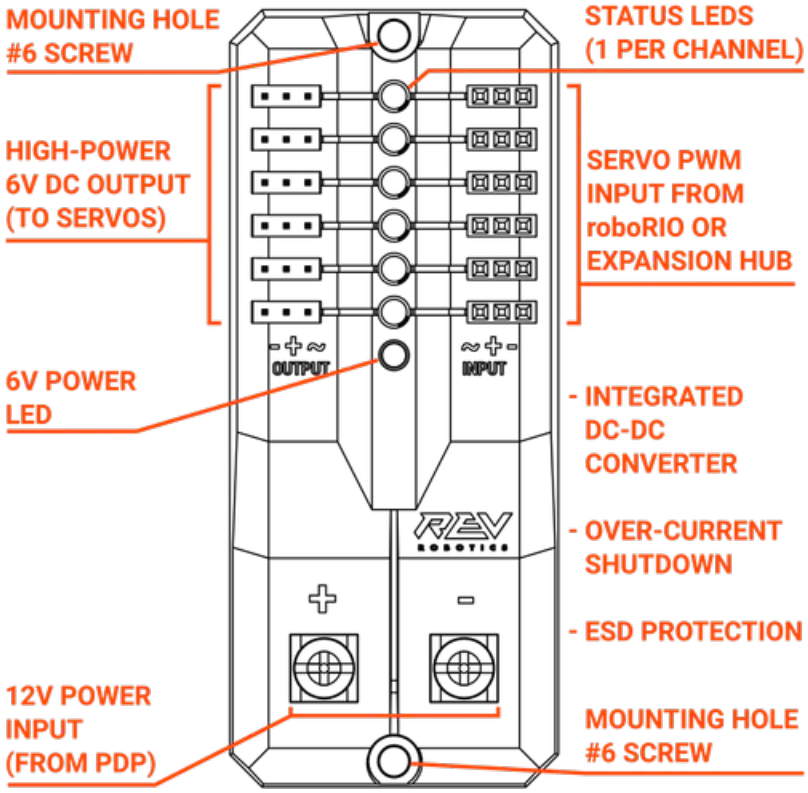
Témoin d'alimentation	Éteint	Aucune alimentation en entrée
	Rouge	Alimentation présente en entrée
Témoin moteur	Rouge	Avant
	Vert	Arrière
Témoin de signal PWM	Rouge	Aucun signal PWM valide n'es détecté
	Vert	Un signal PWM valide est détecté

36.5.20 Mindsensors SD540C (CAN Bus)



Témoin d'alimentation	Éteint	Aucune alimentation en entrée
	Rouge	Alimentation présente en entrée
Témoin moteur	Rouge	Avant
	Vert	Arrière
Témoin de signal CAN	Clignotant rapidement	Aucun appareil CAN n'est connecté
	Éteint	Connecté au roboRIO et le Driver Station est ouvert

36.5.21 REV Robotics Servo Power Module



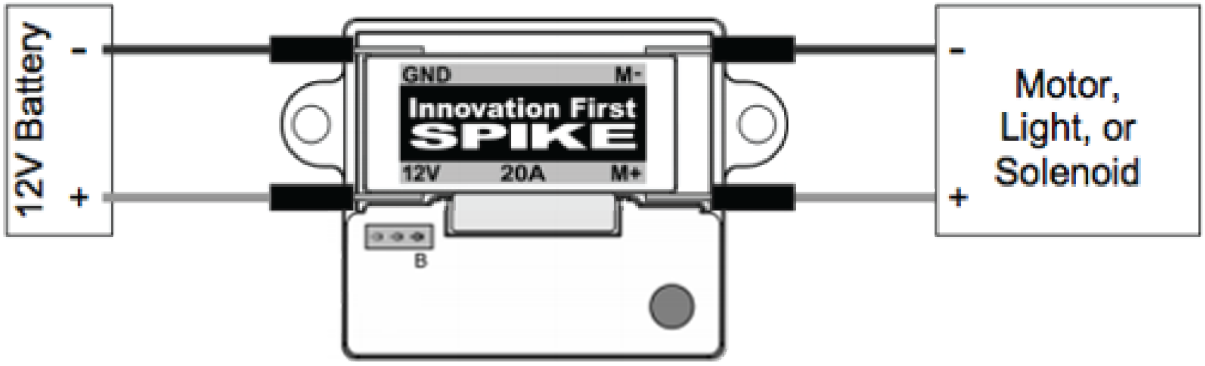
Témoins d'état

Each channel has a corresponding status LED that will indicate the sensed state of the connected *PWM* signal. The table below describes each state's corresponding LED pattern.

État	Témoin
Aucun signal	Ambre clignotant
Signal négatif	Rouge
Signal neutre	Ambre
Signal positif	Vert

— Témoin d'alimentation 6V éteint, faible out clignotant avec une alimentation appliquée = excès de courant

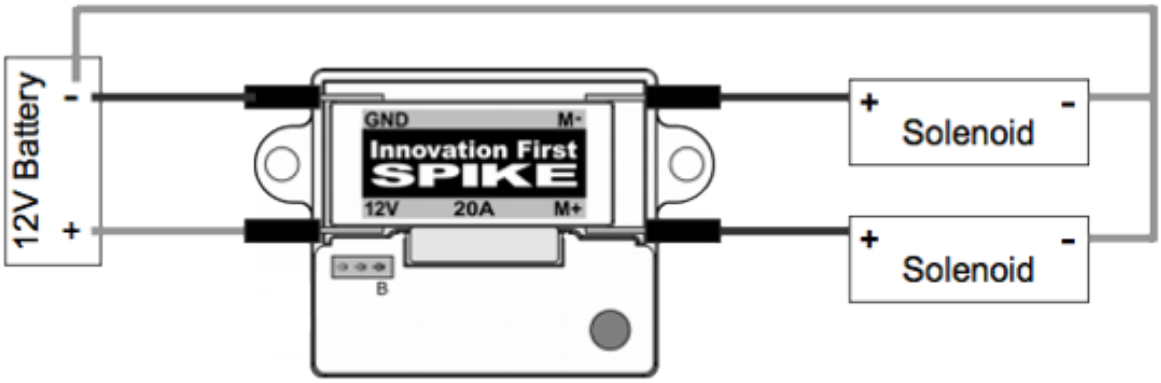
36.5.22 Relais Spike configuré en tant que moteur, lumière ou interrupteur solénoïde



Entrées		Sorties		Témoin	Fonction
Avant (blanc)	Arrière (rouge)	M+	M-		
Éteint	Éteint	GND	GND	Orange	Éteint/frein (défaut)
Allumé	Éteint	+12v	GND	Vert	Moteur tournant dans une direction
Éteint	Allumé	GND	+12v	Rouge	Moteur tournant dans la direction opposée
Allumé	Allumé	+12v	+12v	Éteint	Éteint/frein

Note : Le frein se réfère à l'arrêt dynamique du moteur par le court-circuitage des terminaux du moteur. Ceci n'est pas optionnel lorsque le relais est désactivé.

36.5.23 Relais Spike configuré pour un ou deux solénoïdes



Entrées		Sorties		Témoin	Fonction
Avant (blanc)	**Arrière (rouge)**	**M+**	**M-**		
Éteint	Éteint	GND	GND	Orange	Les deux solénoïdes sont éteints (défaut)
Allumé	Éteint	+12v	GND	Vert	Solénoïde connecté à M+ est alimenté
Éteint	Allumé	GND	+12v	Rouge	Solénoïde connecté à M- est alimenté
Allumé	Allumé	+12v	+12v	Éteint	Les deux solénoïdes sont alimentés

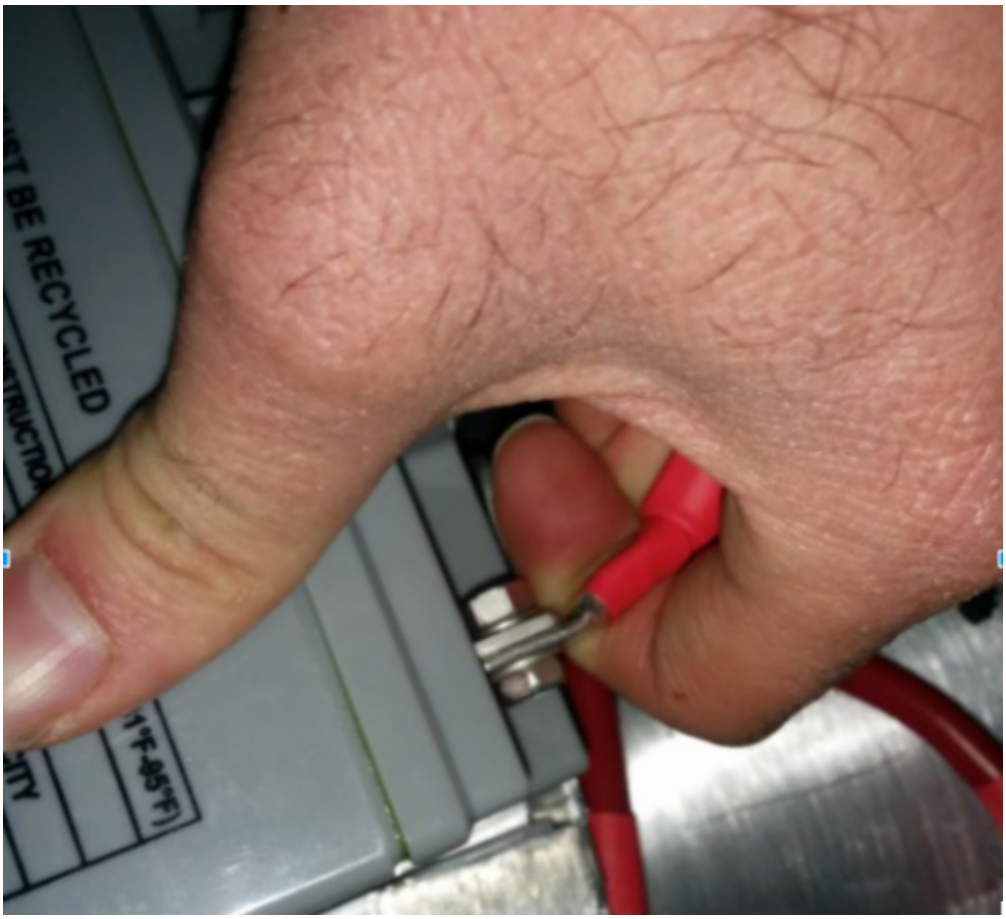
36.5.24 Encodeur CANCoder

Couleur de la DEL	Intensité de la DEL	Détection du bus CAN	Force du champ magnétique	Description
Éteint	Éteint			Le CANCoder n'est pas alimenté
Jaune/Ve	Lumineux			L'appareil est dans le chargeur de démarrage. Référez-vous au manuel d'utilisateur pour plus d'information.
Clignotement rouge lent	Lumineux	Le bus CAN a été perdu		
Clignotement rouge rapide	Faible	Bus CAN jamais détecté depuis le démarrage	L'aimant est en dehors de la plage (<25mT or >135mT)	
Clignotement jaune rapide			L'aimant est dans la plage avec une précision légèrement réduite (25-45mT ou 75-135mT)	
Clignotement vert rapide			L'aimant est dans la plage (entre 45mT et 75mT)	
Clignotement rouge rapide	Lumineux	Bus CAN présent	L'aimant est en dehors de la plage (<25mT or >135mT)	
Clignotement jaune rapide			L'aimant est dans la plage avec une précision légèrement réduite (25-45mT ou 75-135mT)	
Clignotement vert rapide			L'aimant est dans la plage (entre 45mT et 75mT)	

36.6 Dépannage du robot

Note : Pendant la Compétition de robotique FIRST, les robots subissent plusieurs chocs lorsqu'ils sont conduits sur le terrain de jeu. Il est donc important de s'assurer que les connexions soient bien ajustées, que les pièces soient boulonnées de manière sécuritaire et que le tout soit assemblé de sorte que le robot ne se brise pas en se déplaçant sur le terrain.

36.6.1 Vérifier les bornes de la batterie et les connexions

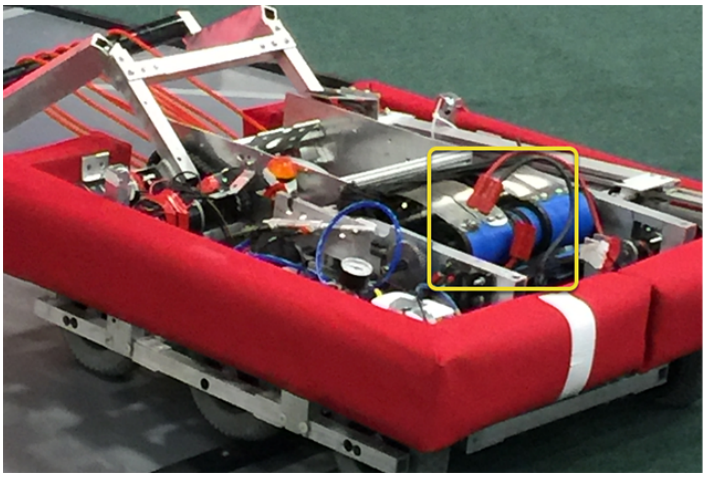


Le ruban qui devrait couvrir la connexion de la batterie dans ces exemples a été retirée pour illustrer ce qui se passe. Sur vos robots, les connexions doivent être couvertes.

Faire bouger les connecteurs allant à la batterie dans un mouvement de va-et-vient. Souvent, ceux-ci ce sont desserrés parce que les vis qui les retiennent sont mal vissées, ou parfois le sertissage n'est pas complètement sécure. Cependant, vous ne pouvez que déceler les très mauvais cas de cette manière, car souvent le ruban électrique qui entoure la connexion rigidifie cette dernière, et vous donne un résultat qui semble correct. L'utilisation d'un voltmètre pour déceler les chutes de tension, ou encore du petit gadget appelé Battery Beak, vous aidera à identifier ce problème.

Appliquez une force considérable sur le câble de la batterie à 90 degrés pour essayer de déplacer la direction du câble quittant la batterie, en cas de succès, la connexion n'était pas assez serrée pour commencer et elle devrait être refaite. Cet *article* contient des informations plus détaillées sur la batterie.

36.6.2 Fixation de la batterie au robot



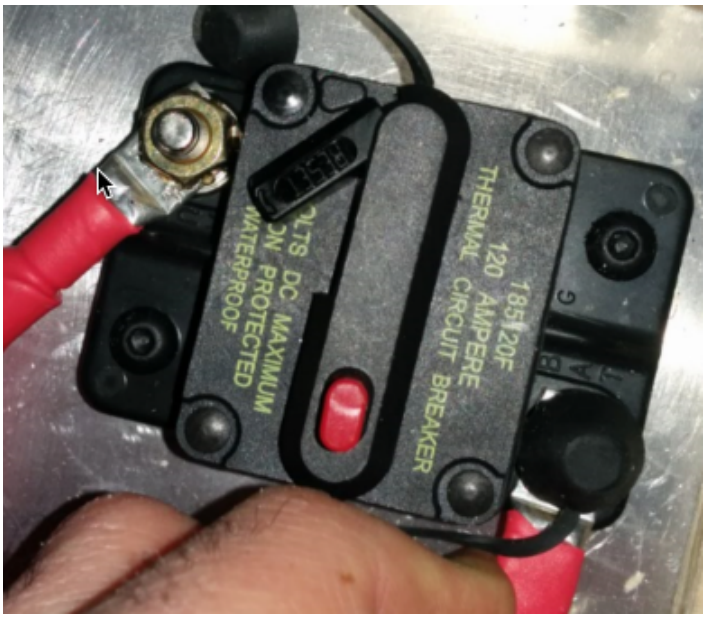
In almost every event we see at least one robot where a not properly secured battery connector (the large Anderson) comes apart and disconnects power from the robot. This has happened in championship matches on the Einstein and everywhere else. Its an easy to ensure that this doesn't happen to you by securing the two connectors by wrapping a tie wrap around the connection. 10 or 12 tie wraps for the peace of mind during an event is not a high price to pay to guarantee that you will not have the problem of this robot from an actual event after a bumpy ride over a defense. Also, secure your battery to the chassis with hook and loop tape or another method, especially in games with rough defense, obstacles or climbing.

36.6.3 Sécuriser le connecteur de batterie et les câbles d'alimentation principaux

Un connecteur de batterie desserré (côté robot : le grand Anderson SB) peut tirer les câbles d'alimentation principaux lorsque la batterie est remplacée. Si les fils d'alimentation principaux sont desserrés, cette traction peut se rendre jusqu'aux cosses à sertir attachées au disjoncteur de 120 A ou au panneau de distribution d'alimentation (PDP), plier la cosse et, avec le temps et la fatigue du métal, provoquer la rupture de l'extrémité de la cosse. La mise en place de quelques attaches sur les câbles d'alimentation principaux en les fixant au châssis et le boulonnage du connecteur de batterie côté robot peut empêcher cela et faciliter la connexion de la batterie.

36.6.4 Disjoncteur principal 120A

Note : Assurez-vous que les écrous sont serrés fermement et que le disjoncteur est fixé à un élément rigide.



Appliquez une forte force de torsion pour essayer de faire tourner la portion sertie. Si elle tourne, l'écrou n'est pas assez serré. Après avoir serré l'écrou, recommencez le test en essayant à nouveau de faire tourner la partie sertie.

L'écrou d'origine a une rondelle en étoile, qui peut s'user avec le temps : il peut être nécessaire de vérifier celles-ci après quelques matches, en particulier si le connecteur de batterie côté robot n'est pas fixé au châssis.

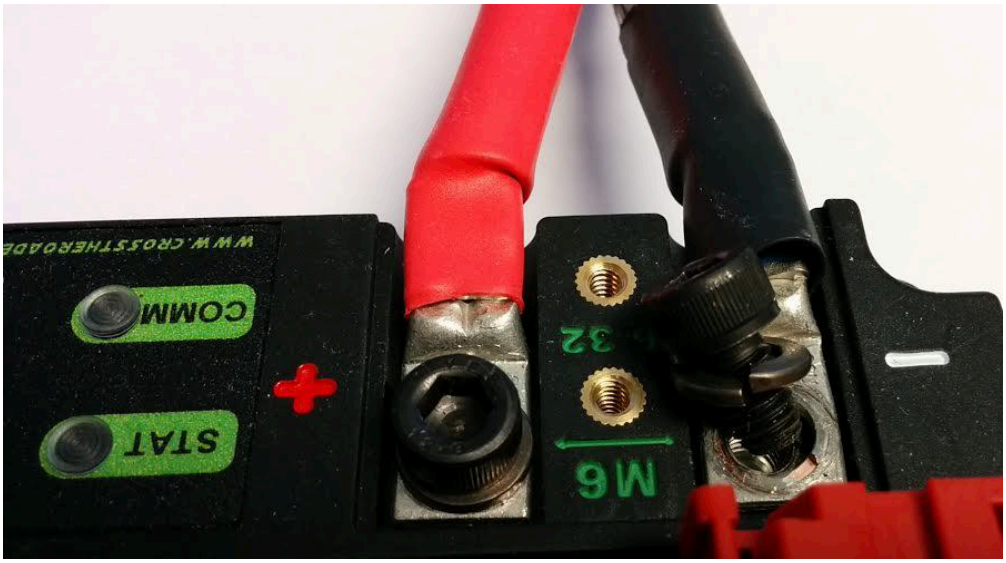
L'écrou a un filetage 1/4-28, ce qui est relativement rare : assurez-vous que vous avez le bon pas si l'écrou est remplacé.

Parce que le goujon métallique est juste moulé dans le boîtier, de temps en temps, vous pouvez casser le goujon. Ne vous inquiétez pas, remplacez simplement l'assemblage.

Lorsqu'il est soumis à plusieurs saisons de compétition, le disjoncteur principal est sensible aux dommages de fatigue dus aux vibrations et à l'utilisation, et peut commencer à s'ouvrir sous l'impact. Chaque fois que la fonction de fusible thermique est déclenchée, il peut devenir

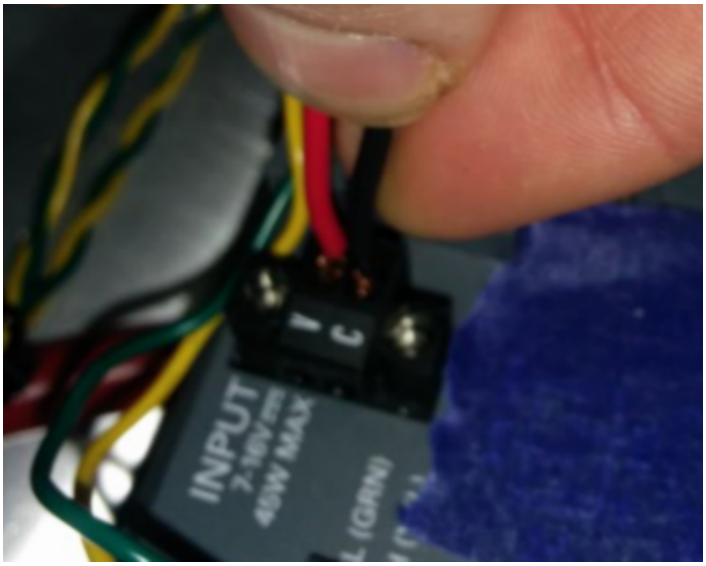
progressivement plus facile de déclencher. De nombreuses équipes de vétérans commencent chaque saison avec un nouveau disjoncteur principal et apportent avec eux des disjoncteurs de rechange.

36.6.5 Panneau de Distribution de Puissance (PDP)



Assurez-vous que des rondelles fendues ont été placées sous les vis du PDP, mais ce n'est pas facile à confirmer visuellement, et parfois vous ne pouvez pas voir. Vous pouvez alors vérifier en retirant le capot sur le boîtier du PDP. De plus, si vous appliquez une pression entre les fils rouge et noir, l'un vers l'autre, vous pouvez parfois déceler les connexions vraiment mal fixées.

36.6.6 Test de robustesse des branchements



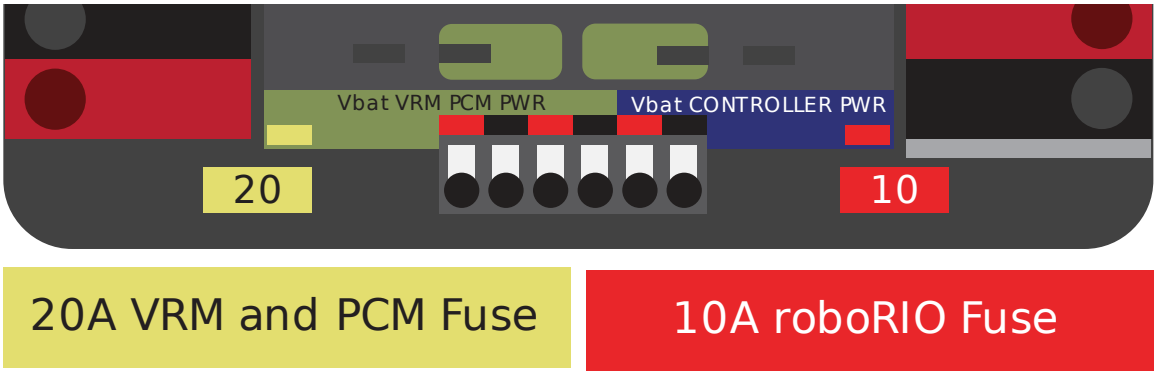
Il est important de vérifier les connecteurs de type Weidmuller pour la puissance, la sortie du compresseur, le connecteur d'alimentation roboRIO et l'alimentation de la radio. On a qu'a tirer sur les connexions, comme indiqué. Assurez-vous qu'aucun fil se retire du connecteur.

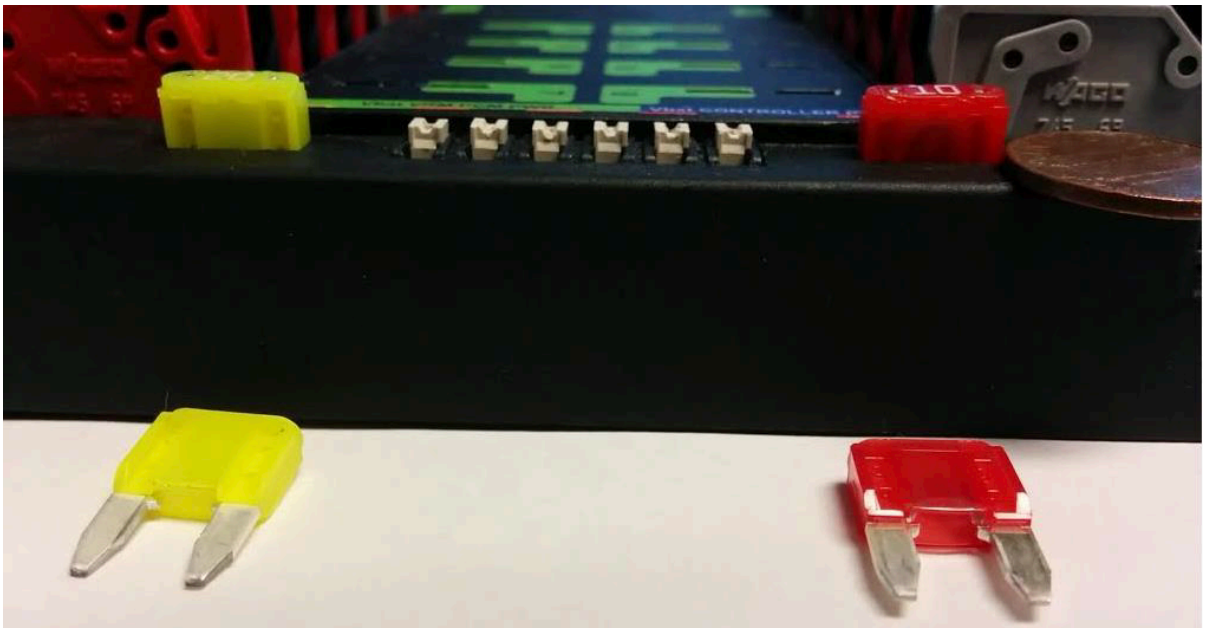
Recherchez des courts-circuits possibles ou imminents avec des connexions de type Weidmuller qui sont proches les unes des autres et dont la partie dénudée du fil est exposée. Toujours réduire au maximum l'exposition de fils dénudées.

Les connecteurs à fourche peuvent également échouer en raison de sertissages inappropriés, alors testez-les également.

36.6.7 Fusible à lamelles

Assurez-vous de placer le fusible 20A (jaune) à gauche et le fusible 10A (rouge) à droite.





Avertissement : Veillez à ce que les fusibles soient bien insérés dans les porte-fusibles. Les fusibles doivent pénétrer au moins aussi loin que la figure ci-dessous (les fusibles de différentes marques ont des longueurs de lames différentes). Il devrait être presque impossible de retirer le fusible à mains nues (sans utiliser une pince à long bec). Si cela n'est pas fait correctement, le robot et/ou la radio peuvent présenter des problèmes de connectivité intermittents.

Si vous pouvez retirer les fusibles à lame à la main, ils ne sont pas complètement insérés. Assurez-vous qu'ils sont complètement insérés dans le PDP afin qu'ils ne ressortent pas pendant le fonctionnement du robot.

36.6.8 Limaille de fer dans le roboRIO

La limaille est formée de copeaux fins de métal, d'abrasifs ou d'autres matériaux produits par une opération d'usinage. Souvent, des modifications doivent être apportées à un robot après que toute la partie électronique a été installée. Le circuit imprimé du roboRIO est recouvert d'un vernis protecteur, mais cela ne garantit pas de façon absolue que la présence de limaille puisse court-circuiter les traces ou les composants à l'intérieur du boîtier du roboRIO. Dans tous les cas, vous devez veiller à ce qu'aucun copeau de métal, limaille ou autre ne se retrouve dans le roboRIO ou dans l'un des autres composants electronique. En particulier, les connecteurs à 3 broches qui sont exposés en périphérie du roboRIO sont un endroit où les copeaux peuvent entrer dans le boîtier. Un examen approfondi des quatre côtés du boîtier avec une lampe de poche est généralement suffisant pour trouver les très mauvaises cas.

36.6.9 Connecteur à barillet de la radio

Make sure the correct barrel jack is used, not one that is too small and falls out for no reason. This isn't common, but ask an *FTA* and every once in awhile a team will use some random barrel jack that is not sized correctly, and it falls out in a match on first contact.

36.6.10 Câble Ethernet

Si le câble Ethernet RIO vers radio ne contient pas le clip qui verrouille le connecteur, procurez-vous un autre câble. C'est un problème courant qui se produit plusieurs fois dans chaque compétition. Assurez-vous que vos câbles sont sécurisés. Le clip se rompt souvent, en particulier lorsque le câble reste accroché lors de l'assemblage électrique.

36.6.11 Câbles relâchés

Les câbles doivent être serrés, en particulier le câble d'alimentation radio et Ethernet. Les câbles d'alimentation radio ne sont pas retenus avec beaucoup de force de frottement, et tomberont facilement sous un choc (même s'il s'agit du bon conncecteur), ou si le câble est lâche et peut se déplacer facilement.

Le câble Ethernet est également assez lourd, s'il est autorisé à pivoter librement, le clip en plastique peut ne pas être suffisant pour maintenir les connecteurs à broches Ethernet en circuit.

36.6.12 Reproduire un problème dans le puits.

Au-delà des secousses normales des câbles pendant que le robot est alimenté et attaché, il est suggéré qu'un côté du robot soit levé de quelques centimètres, puis de le laisser retomber afin de lui infliger une secousse. La conduite sur le terrain, en particulier contre les défenseurs, sera souvent très violente, ce qui permet de s'assurer que rien ne tombe ou se brise. Il vaut mieux que le robot échoue dans le puits plutôt qu'au milieu d'un match.

Lorsque vous effectuez ce test, il est important d'être connecté via Ethernet et non par USB, sinon vous ne testez pas tous les chemins critiques.

36.6.13 Vérification des logiciels embarqués (Firmware) et des versions

Les inspecteurs de robots font cela, mais vous devriez le faire aussi, cela aide les inspecteurs de robots et ils l'apprécient. Et cela garantit que vous utilisez le code le plus récent, avec un minimum de bogues. Vous ne voulez pas perdre une partie à cause d'un logiciel de système de contrôle obsolète sur votre robot !

36.6.14 Vérification de la station de pilotage

Nous observons souvent des problèmes avec la station de pilotage. Vous devez :
- Apporter TOUJOURS le câble d'alimentation de l'ordinateur portable sur le terrain, peu importe l'état de sa batterie, vous êtes toujours autorisé à le brancher sur le terrain.
- Vérifier les paramètres d'alimentation et de veille, désactivez la veille et la mise en veille prolongée, les économiseurs d'écran, etc.
- Désactiver l'alimentation des ports USB (dev manager)
- Désactiver l'alimentation des ports Ethernet (dev manager)
- Désactiver l'antivirus Windows Defender
- Désactiver le pare-feu
- Fermez toutes les applications à l'exception de DS/Dashboard lorsque vous êtes sur le terrain.
- Vérifiez qu'il n'y a rien d'inutile en cours d'exécution dans la barre d'applications dans le menu Démarrer (en bas à droite)

36.6.15 Outils pratiques

Il ne semble jamais y avoir assez de lumière à l'intérieur des robots, du moins pas assez pour scruter les points de connexion critiques, alors pensez à utiliser une lampe de poche LED pour inspecter les connexions de votre robot. Ils sont disponibles chez Home Depot ou dans tout magasin de quincaillerie/automobile.

Un outil WAGO est un bel outil pour refaire les connexions Weidmuller avec des fils multibrins. Souvent, je vais faire une démonstration d'un raccord pour montrer à l'équipe, puis leur faire faire le reste en utilisant l'outil WAGO pour appuyer sur le piston blanc pendant qu'ils insèrent le fil multibrins. L'angle de l'outil WAGO rend cela particulièrement utile.

36.7 Batterie de Robot

L'alimentation électrique pour un robot FRC est une batterie anti-déversement (plomb-acide scellé) 12V 18Ah, capable de fournir brièvement plus de 180A et un arc de plus de 500A lorsqu'elle est complètement chargée. L'assemblage de la Batterie de Robot inclut la batterie *COTS*, les câbles avec contacts, et un connecteur Anderson SB. Les équipes sont encouragées à se procurer plusieurs Batteries de Robot.

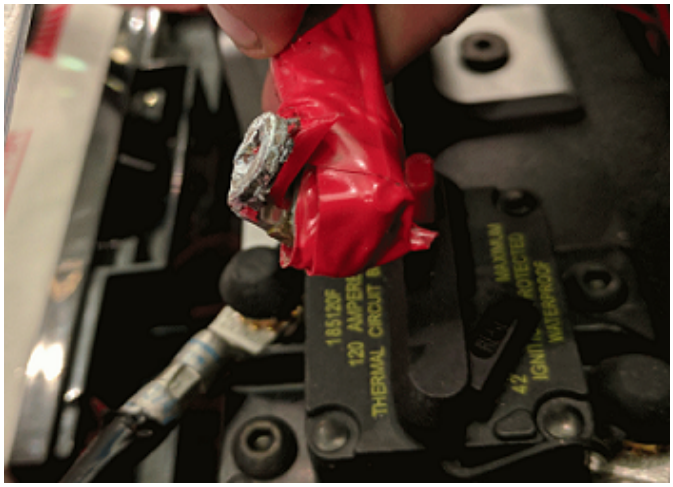
36.7.1 Batterie COTS

Les Règles de Robot dans le Manuel du Jeu spécifient une batterie anti-déversement plomb-acide scellé prémonté d'un manufacturier (COTS) qui répond à des critères spécifiques et fournit des exemples des numéros de pièce légaux de plusieurs manufacturiers.

36.7.2 Sécurité et Manipulation de la Batterie

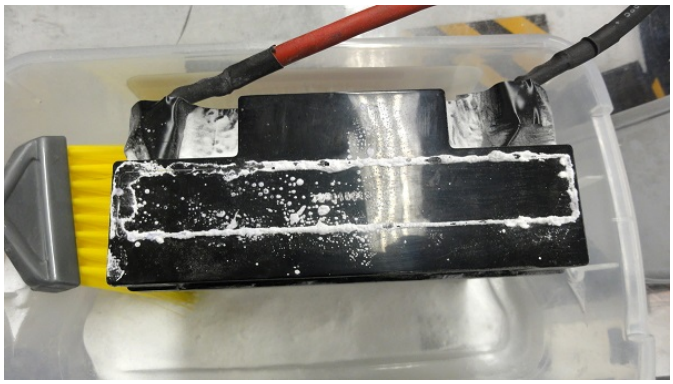
Une batterie saine est **toujours** « en marche » et les terminaux sont **toujours** actifs. Si les bornes de la batterie sont court-circuitées - par exemple, une clé ou autre objet metallique touche les deux terminaux - toute l'énergie stockée sera libérée dans un arc dangereux. Ce risque est à l'origine d'un large éventail de bonnes pratiques, comme couvrir les terminaux lorsque la batterie est en stockage, seulement les découvrir et travailler sur un terminal ou une polarité à la fois. Aussi, garder les contacts SB entièrement inséré dans les connecteurs.

Ne *JAMAIS* transporter une batterie par les câbles et évitez toujours de les tirer. La traction dans les câbles commencera à endommager les connexions et les terminaux (cosses). Au fil du temps, les dommages dus à la fatigue peuvent s'additionner jusqu'à ce que la borne entière de la batterie se déchire du boîtier ! Même si elle n'est pas clairement brisée, les dommages de fatigue internes peuvent augmenter la résistance interne de la batterie, usant prématurément la batterie. La batterie ne sera pas en mesure de fournier la même quantité de courant avec une résistance interne augmentée, ou si les *connecteurs ne sont pas serrés*.



La chute des batteries peut plier les plaques internes et entraîner des problèmes de performances, créer des renflements ou même fissurer le boîtier de la batterie. Alors que la plupart des batteries FRC utilisent la technologie Absorbent Glass Mat [AGM] ou Gel pour des raisons de sécurité et de performance, lorsqu'une cellule est perforée, une petite quantité d'acide de batterie peut encore s'échapper. C'est l'une des raisons pour lesquelles FIRST recommande aux équipes de disposer d'un kit de déversement de batterie.

Enfin, certains chargeurs de batterie plus anciens sans fonctions de «mode de maintenance» peuvent *surcharger* la batterie, entraînant l'ébullition d'une partie de l'acide de la batterie.



Les batteries endommagées doivent être mises au rebut en toute sécurité dès que possible. Tous les détaillants qui vendent de grosses batteries SLA, comme des batteries de voiture, devraient pouvoir s'en débarrasser pour vous. Ils peuvent facturer une somme modique ou fournir un petit «remboursement des frais de base», selon la législation de votre état.

Danger : N'ESSAYEZ PAS de «réparer» des batteries endommagées ou non fonctionnelles.

36.7.3 Construction de batterie et outils

Fils de batterie

Les fils de batterie doivent être en cuivre, de taille minimale (section transversale) 6 AWG (16 mm2, 7 SWG) et de longueur maximale 12″, codés par couleur pour la polarité, avec un connecteur Anderson SB. dans le kit de pièces et sont vendus par les fournisseurs FRC.

Câbles de branchement

Le cuivre étamé, recuit ou revêtu est autorisé. N'utilisez pas de CCA (aluminium revêtu de cuivre), d'aluminium ou de tout autre métal de base autre que le cuivre. Le métal conducteur est normalement imprimé à l'extérieur de l'isolation avec les autres calibres de câble.

La taille de fil 6AWG est suffisante pour presque tous les robots et convient aux contacts SB50 standard. Un petit nombre d'équipes adoptent des tailles de fil plus grandes pour des avantages de performance marginaux.

Un fil de plus grand nombre de torons (parfois vendu comme «Flex» ou «fil de soudage») a un rayon de courbure plus petit, ce qui le rend plus facile à acheminer, et une limite de fatigue plus élevée. Il n'y a pas d'exigence de nombre de brins, mais 84/25 (fil de raccordement 84 brins « flex ») et 259/30 (259 brins « fil de soudage ») seront tous deux *beaucoup* plus faciles à travailler que 19/0.0372 (raccordement 19 brins fil).

L'isolation doit être codée par couleur selon le manuel du jeu : à partir de 2021, le fil + 12Vdc doit être rouge, blanc, marron, jaune ou noir avec bande et le fil de terre (fil de retour) doit être noir ou bleu. Il n'y a pas d'exigence de température d'isolement explicite, mais toute isolation noircie ou endommagée signifie que le fil doit être remplacé : La valeur de 105 ° C est suffisante et même des valeurs plus basses sont acceptables. Il n'y a pas d'exigence de tension d'isolement, une valeur inférieure est signifie une isolation plus mince.

Connecteur SB

Le connecteur Anderson SB peut être le SB50 rose/rouge standard ou un autre connecteur Anderson SB. Il est *FORTEMENT* recommandé aux équipes d'utiliser le SB50 rose/rouge pour l'interopérabilité : les autres couleurs et tailles de boîtiers ne sont pas compatibles, et ainsi, vous ne pourrez pas emprunter de batteries ou de chargeurs d'autres équipes, si vous en avez besoin.

Suivez les instructions du fabricant pour sertir les contacts et assembler les fils dans les connecteurs Anderson SB. Un petit tournevis à tête plate peut aider à insérer les contacts (appuyez sur le contact, pas sur l'isolation du fil), ou il peut aider à désengager le loquet interne si le contact est dans la mauvaise fente ou à l'envers.

Cosses de batterie

Les cosses de compression («cosses à sertir») pour les languettes de batterie à boulon n ° 10 (ou M5) (diamètre du trou ~ 0.2″ ou ~ 5 mm) sont disponibles en ligne et chez les fournisseurs de composants électriques, et sont vendues selon les tailles de fil acceptées en AWG (ou mm2) et diamètre du plot (« taille du boulon », « diamètre du trou »). Les fournisseurs haut de gamme feront également la distinction entre le nombre de brins Standard (~ 19) et Flex (> 80) dans leurs catalogues de cosses. Certains fournisseurs proposent également des cosses à angle droit, en plus de styles droits plus courants. Suivez les instructions du fabricant pour sertir les cosses.

Les cosses à vis sont légales, mais non recommandées. Si vous utilisez des cosses de borne à vis, utilisez le tournevis de taille de pointe appropriée pour serrer la borne. Vérifiez fréquemment le serrage des bornes car elles peuvent se desserrer avec le temps.

Cosse du terminal de la batterie à la connexion postérieure

Un écrou et un boulon n ° 10 ou M5 relient la cosse du câble de la batterie à la languette de la batterie.

Avertissement : La patte et la languette doivent être en contact direct, cuivre sur cuivre : ne pas mettre de rondelle d'aucune sorte les séparant.

Certaines batteries sont livrées avec des boulons à languette dans l'emballage : ils peuvent être utilisées ou préférablement remplacées par des boulons en acier plus solides. C'est une bonne idée d'ajouter une rondelle de blocage fonctionnelle, comme une rondelle en étoile # 10 ou un système de rondelle nordlock, en plus d'un écrou de blocage en nylon («nylock»). N'utilisez qu'un seul style de rondelle de blocage pour chaque connexion. Même si le fabricant fournit des rondelles de blocage à anneau fendu dans l'emballage, vous n'êtes pas obligé de les utiliser.

Ces connexions doivent être très serrées pour la fiabilité. Tout mouvement de l'ergot pendant le fonctionnement peut interrompre l'alimentation du robot, entraînant des redémarrages du robot et des déconnexions sur le terrain pendant 30 secondes ou plus.

Cette connexion doit également être entièrement couverte pour la sécurité électrique ; du ruban isolant fonctionnera, mais une gaine thermorétractable qui s'adapte sur toute la connexion est recommandée. Des taux de rétrécissement élevés (minimum 3 :1, recommandé 4 :1) faciliteront l'application de la gaine thermorétractable. La thermorétraction doublée d'adhésif est autorisée. Assurez-vous que *tout* le cuivre est couvert ! La gaine thermorétrac-

table doit être «réparée» avec du ruban isolant si du cuivre apparaît.

Chargeurs de batterie

Il existe de nombreux bons chargeurs de batterie «intelligents» COTS conçus pour des batteries SLA 12 V, d'une capacité nominale de 6 A ou moins par batterie, avec des fonctionnalités de «mode maintenance». Les chargeurs de plus de 6 A ne sont pas autorisés dans les fosses FRC.

Les chargeurs utilisés en compétition doivent utiliser des connecteurs Anderson SB. La fixation d'un câble de batterie de connecteur COTS SB aux câbles du chargeur à l'aide de serre-fils ou de bornes à vis de taille appropriée est rapide et simple (assurez-vous de couvrir tout cuivre exposé avec une gaine thermorétractable ou du ruban isolant). Les contacts de connecteur SB sont également disponibles pour les fils de plus petite taille, si l'équipe dispose d'une capacité de sertissage.

> **Avertissement :** Après avoir fixé le SB, vérifiez les polarités du chargeur avec un multimètre avant de brancher la première batterie.

Certains fournisseurs de FRC vendent des chargeurs avec des connecteurs SB50 rouges préconnectés.

Outils d'évaluation de la batterie

Chargeur de batterie

Si votre chargeur de batterie est doté d'un indicateur de mode de maintenance, tel qu'une LED VERTE, vous pouvez utiliser cet indicateur pour vous dire si vous êtes PRÊT. Certains chargeurs alterneront périodiquement entre «CHARGING» et «READY». Il s'agit d'un comportement de «maintenance», parfois associé au refroidissement de la batterie et à la capacité d'accepter plus de charge.

Affichage et journal des évènements de votre Driver Station

Lorsque le robot est branché et connecté à l'ordinateur portable de la station de pilotage, la tension de la batterie est affichée sur le logiciel NI Driver Station.

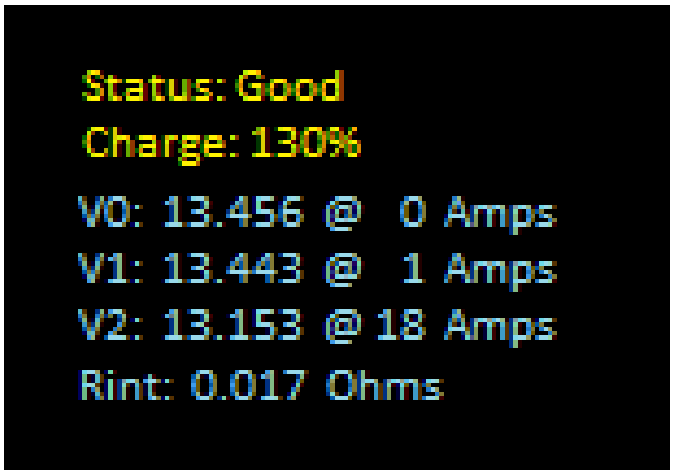
Après avoir terminé une session de conduite, vous pouvez réviser le *journal de capture des tensions de batterie*

Voltmètre ou **Multimètre** portatif

Une lecture de tension sur le connecteur SB d'une batterie déconnectée vous donnera un aperçu de ce que le Voc (tension circuit ouvert, ou «tension flottante») est dans l'état «déchargé». En général, le Voc n'est pas une méthode recommandée pour comprendre l'état de la batterie : la tension en circuit ouvert n'est pas aussi utile que la combinaison de la résistance interne et des tensions à des charges spécifiques fournies par un testeur de charge (ou analyseur de batterie).

Testeur de charge

Un testeur de charge de batterie peut être utilisé comme un moyen rapide de déterminer l'état de fonctionnement d'une batterie. Il peut fournir des informations telles que : tension de charge ouverte, tension sous charge, résistance interne et état de charge. Ces mesures peuvent être utilisées pour confirmer rapidement qu'une batterie est prête pour un match et même aider à identifier certains problèmes à long terme avec la batterie.



La résistance interne idéale doit être inférieure à 0.015 Ohms. La spécification du fabricant pour la plupart des batteries est de 0.011 Ohms. Si une batterie dépasse 0.020 Ohms, il est judicieux d'envisager de ne pas utiliser cette batterie pour les matchs en compétition.

Si une batterie présente des tensions significativement plus basses aux charges de courant de test les plus élevés, il se peut qu'elle ne se recharge pas ou doive être retirée.

36.7.4 Comprendre les tensions de batterie

Une «batterie 12 V» est tout sauf 12.0 V.

Complètement chargée, une batterie peut être de 12.7 à 13.5 volts en circuit ouvert (Voc). La tension en circuit ouvert est mesurée avec *aucune charge* connectée.

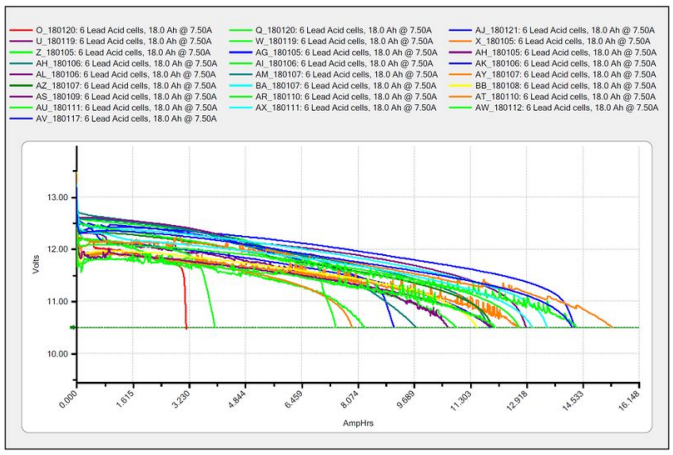
Une fois qu'une charge (comme un robot) est connectée et qu'une quantité de courant circule, la tension de la batterie chutera. Donc, si vous vérifiez une batterie avec un voltmètre, et qu'il lit 13.2, puis connectez-le à votre robot et allumez-le, il affichera une valeur inférieure, peut-être 12.9 sur l'écran de la station de conduite. Ces chiffres varient avec chaque batterie et chaque robot , voir Caractérisation ci-dessous. Une fois que votre robot commence à fonctionner, il consommera plus de courant et la tension diminuera davantage.

Les batteries lisant 12.5 V sur un robot inactif doivent être échangées et chargées avant un match. Remplacez toujours les batteries avant que le robot ne commence à atteindre les seuils de sécurité contre les baisses de tension (demeurant à basse tension sur l'écran de la station de conduite), car le fonctionnement dans des plages de basse tension risque d'endommager la batterie de manière permanente ; ce comportement peut se produire à divers états Voc en fonction de l'état de la batterie, du fabricant de la batterie et de la conception du robot. L'état de charge de la batterie doit être maintenu à plus de 50% pour la longévité de la batterie.

La tension et le courant de la batterie dépendent également de la température : les batteries froides sont des batteries heureuses.

Caractérisation de la batterie

Un analyseur de batterie peut être utilisé pour effectuer une inspection détaillée et une comparaison des performances de la batterie.



Il fournira des graphiques des performances de la batterie au fil du temps. Ce test prend beaucoup de temps (environ deux heures), il est donc moins adapté aux tests en compétition. Il est recommandé d'exécuter ce test sur chaque batterie chaque année pour surveiller et suivre ses performances. Cela déterminera comment il doit être utilisé : matchs, entraînement, test ou élimination.

À la charge d'essai standard de 7.5 ampères, les batteries de compétition doivent avoir une valeur nominale d'au moins 11.5 ampères-heure. Tout ce qui est inférieur à cela ne doit être utilisé que pour la pratique ou d'autres cas d'utilisation moins exigeants.

Longévité de la batterie

Une batterie est conçue pour environ 1200 cycles de charge/recharge normaux. Les courants élevés requis pour un match FRC réduisent cette durée de vie à environ 400 cycles. Ces cycles sont destinés à être à décharge relativement faible, d'environ 13.5 à 12 ou 12.5 volts. Un cycle profond de la batterie (la faire fonctionner complètement avec une grosse charge) l'endommagera.

Les batteries durent plus longtemps si elles sont maintenues complètement chargées lorsqu'elles ne sont pas utilisées, soit en les rechargeant régulièrement, soit en utilisant un chargeur de maintenance. Les batteries chutent d'environ 0,1 V chaque mois de non-utilisation.

Les batteries doivent être tenues à l'écart de la chaleur et du froid extrêmes. Cela signifie généralement stocker les batteries dans une zone à température contrôlée : un placard de classe est généralement bien, une remise externe exposée aux éléments est plus risqué.

36.7.5 Meilleures pratiques en matière de batterie

- Utilisez uniquement une batterie chargée pour les matchs de compétition. Si vous êtes dans une situation où vous êtes à court de batteries chargées, veuillez demander de l'aide à une équipe expérimentée ! Personne ne veut voir un robot mort sur le terrain (*brownout*) en raison d'une batterie défectueuse ou non chargée.
- Il est fortement recommandé aux équipes d'utiliser des outils correctement évalués et des pratiques de contrôle qualité strictes pour les processus de sertissage (demandez l'aide des équipes locales de vétérans ou d'un électricien commercial), ou d'utiliser des câbles de batterie fabriqués par le fournisseur.
- Attendez que les batteries refroidissent après le match avant de les recharger : le boîtier ne doit pas être chaud au toucher, quinze minutes suffisent généralement.
- Les équipes devraient envisager d'acheter plusieurs nouvelles batteries chaque année pour avoir des batteries fraîches pour les match plus compétitifs. Les matchs d'élimination peuvent nécessiter de nombreuses batteries et il se peut que le temps de recharge ne soit pas suffisant.

— Un chargeur de batterie multi-banque vous permet de charger plus d'une batterie à la fois. De nombreuses équipes construisent un chariot de robot pour leurs batteries et chargeurs, ce qui facilite le transport et le stockage.
— C'est une bonne idée d'identifier en permanence chaque batterie avec au moins : le numéro d'équipe, l'année et un identifiant unique.
— Les équipes peuvent également vouloir utiliser un truc amovible (autocollants, étiqueteuse, etc.) pour identifier l'usage de cette batterie en fonction de ses données de performance et de la date du dernier test de l'analyseur.

— L'utilisation de témoins de batterie (un morceau de plastique placé dans le connecteur de batterie) est un moyen courant d'indiquer qu'une batterie a été chargée. Les témoins de batterie peuvent également être facilement imprimés en 3D.
— Des poignées pour contacts SB50 peuvent être achetées ou imprimées en 3D pour éviter de tirer sur les fils lors de la connexion ou de la déconnexion des batteries. N'utilisez pas ces poignées pour supporter le poids de la batterie.

— Certaines équipes cousent des sangles de transport de batterie à partir d'anciennes ceintures de sécurité ou d'un autre nylon plat qui s'ajustent autour de la batterie pour éviter d'être transporté par des fils.

— Les clips de bord d'attache de câble (cable tie edge clips) peuvent être utilisés avec des cosses à sertir à 90 degrés pour réduire la tension sur les câbles de batterie.

37

Didacticiels des composants matériels

Note : Les pages de cette section de la documentation contiennent des supports médiatiques qui ne sont visibles qu'à partir de la version web de la documentation

37.1 Moteurs pour applications en robotique

L'une des décisions de conception les plus importantes que les équipes doivent faire face est la sélection et la conception des systèmes de moteur sur leur robot. Trop souvent un moteur incompatible est choisi pour une conception particulière ce qui donne des performances réduites et, parfois encore pire, une défaillance de moteurs occasionné par une consommation excessive de courant. Dans cette série de vidéos, le professeur Ken Stafford de wpi explique comment fonctionnent les moteurs, comment concevoir des systèmes pour fonctionner à des performances maximales ainsi qu'un exemple de conception pour un système de robot.

37.2 Détection et Capteurs

Sans aucun capteurs ou senseurs, les robots sont seulement des véhicules radiocommandés. Les capteurs permettent aux robot de contrôler l'opération interne des système mécaniques du robot, et donnent la capacité d'interagir avec l'environnement externe au robot. Dans ces vidéos, le professeur Craig Putnam du WPI nous parle de plusieurs classes de capteurs, comments ils sont utilisés, et fournit des conseils à propos de quelles capteur est le plus adéquat pour votre application.

37.3 Systèmes pneumatiques

Le système pneumatique est un dispositif d'actionnement souvent sous-utilisé qui peut être utilisé sur les robots. Il y a beaucoup d'avantages à utiliser la technologie pneumatique par rapport à l'utilisation des moteurs. Dans cette vidéo, le professeur Ken Stafford décrit les caractéristiques de la pneumatique, les applications avec des robots ainsi que le calcul du système de bonne taille pour une application.

37.4 Transmission de puissance

Le choix des bons moteurs pour une application précise va de pair avec la transmission adéquate de la puissance de ces moteurs à l'endroit où elle est requise. L'utilisation d'engrenages ou de chaînes et de pignons sont deux moyens efficaces d'adapter la puissance du moteur à l'application entraînée. Dans cette vidéo, le doctorant en Génie Robotique à WPI Michael Delph nous parle de la transmission de puissance, y compris comment choisir les bons rapports d'engrenage ou de chaînes et de pignons afin d'obtenir les performances maximales dans la conception de votre robot.

38 Capteurs

38.1 Bref aperçu des capteurs

Note : Cette section couvre le matériel des capteurs, pas l'utilisation de capteurs dans le code. Pour un guide des capteurs logiciels, voir *Un bref aperçu du logiciel associé aux capteurs*.

Pour être efficaces, il est souvent essentiel que les robots puissent recueillir des informations sur leur environnement. Les dispositifs qui fournissent une rétroaction au robot sur l'état de son environnement sont appelés « capteurs ». Il existe une grande variété de capteurs disponibles pour les équipes FRC® pour mesurer à peu près tout, du positionnement sur le terrain à l'orientation du robot en passant par le positionnement moteur/mécanisme. L'utilisation de capteurs est une compétence absolument cruciale pour réussir sur le terrain en compétition ; alors que la plupart des jeux FRC comportent des tâches qui peuvent être accomplies par un robot « aveugle », les meilleurs robots comptent fortement sur des capteurs pour accomplir des tâches de jeu aussi rapidement et de manière fiable que possible.

De plus, les capteurs peuvent être extrêmement importants pour la sécurité du robot - de nombreux mécanismes de robot sont capables de se briser s'ils sont mal programmés ou poussés hors-limites. Les capteurs offrent une protection contre cela, permettant aux robots, par exemple, de désactiver un moteur si un mécanisme bute contre un arrêt d'urgence.

38.1.1 Types de capteurs

Les capteurs utilisés dans FRC peuvent généralement être classés de deux manières différentes : par fonction et par protocole de communication. La première catégorie est pertinente pour la conception de robots ; la dernière pour le câblage et la programmation.

Capteurs par fonction

Les capteurs peuvent fournir de l'information sur une variété d'aspects différents de l'état du robot. Les fonctions de capteur communes à FRC incluent :

- *Commutateurs de proximité*
  - Détecteurs de proximité mécaniques (« fins de course »)
  - Détecteurs de proximité magnétiques
  - Détecteurs de proximité inductifs
  - Détecteurs de proximité photoélectriques
- Capteurs de distance
  - *Capteurs à ultrasons*
  - *Télémètres triangulaires*
  - *LIDAR*
- Capteurs de rotation d'arbre
  - *Encodeurs*
  - *Potentiomètres*
- *Accéléromètres*
- *Gyroscopes*

Capteurs par protocole de communication

Pour qu'un capteur soit utile, il doit être capable de «parler» au roboRIO. Il existe plusieurs méthodes par lesquelles les capteurs peuvent communiquer et passer leur données au roboRIO :

- *Entrée analogique*
- *Entrée numérique*
- *Bus série*

En général, la prise en charge des capteurs qui communiquent via des entrées analogiques et numériques est simple, tandis que la communication sur le bus série peut être plus compliquée.

38.2 Entrées analogiques

Note : Cette section traite du matériel d'entrée analogique. Pour un guide logiciel sur les entrées analogiques, voir *Entrées analogiques - Partie logicielle*.

Un signal analogique est un signal dont la valeur peut se situer n'importe où dans un intervalle continu. Cela diffère d'un *signal numérique*, qui ne peut prendre qu'une seule valeur discrète parmi plusieurs. Les ports d'entrée analogique du roboRIO permettent la mesure de signaux analogiques qui ont des valeurs entre 0 et 5 Volts.

En pratique, il n'y a aucun moyen de mesurer un « vrai » signal analogique avec un appareil numérique tel qu'un ordinateur (comme le roboRIO). Par conséquent, les entrées analogiques

sont réellement mesurées comme un signal numérique à 12 bits - qui est une résolution assez élevée [1].

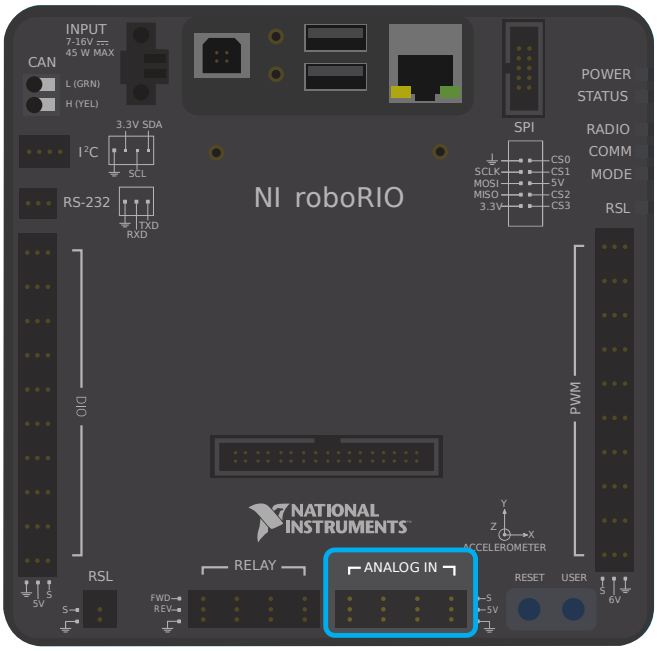
Les entrées analogiques sont généralement (mais pas toujours !) utilisées pour les capteurs dont les mesures varient en continu sur une plage, tels que *capteurs ultrasoniques* et *potentiomètres*, car ils fournissent une tension proportionnelle à leurs mesures.

38.2.1 Connexion aux ports d'entrée analogiques roboRIO

Note : Quatre entrées analogiques supplémentaires sont disponibles via le port d'extension « MXP ». Pour les utiliser, une carte supplémentaire qui se connecte au MXP est nécessaire.

Avertissement : Consultez toujours les spécifications techniques du capteur que vous utilisez *avant* de le câbler au roboRIO, afin de vous assurer que le bon fil est connecté à la bonne broche. Ne pas le faire peut endommager le capteur ou le roboRIO.

Avertissement : Ne **Jamais** connecter directement la broche d'alimentation à la broche de mise à la terre sur n'importe quel port du roboRIO ! Ce court-circuit déclenchera des fonctions de protection sur le roboRIO et générer comportement inattendu.



Le roboRIO dispose de 4 ports d'entrée analogique intégrés (numérotés 0-3), comme le montre l'image ci-dessus. Chaque port a trois broches - signal (« S »), alimentation (« V ») et masse (»⏚ »). Les broches « alimentation » et « masse » sont utilisées pour alimenter les capteurs périphériques qui se connectent aux ports d'entrée analogique - il existe une différence

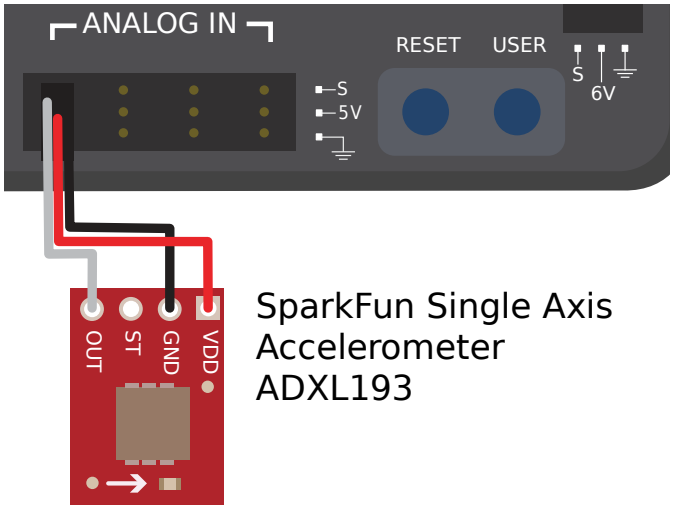
1. Une résolution de 12 bits donne 2^{12}, soit 4096 valeurs différentes. Pour une plage de 5 V, c'est une résolution effective d'environ 1,2 mV ou 0,0012 V. L'accéléromètre étant précis à plus ou moins 50 mV (dans ses spécifications techniques), donc cette résolution n'est pas le facteur limitant la précision de la mesure.

de potentiel constante de 5 V entre les broches « alimentation » et « masse »[2]. La broche de signal est la broche sur laquelle le signal est réellement mesuré.

Connexion d'un capteur à un seul port d'entrée analogique

Note : Certains capteurs (tels que *potentiomètres*) peuvent avoir des connexions d'alimentation et de masse interchangeables.

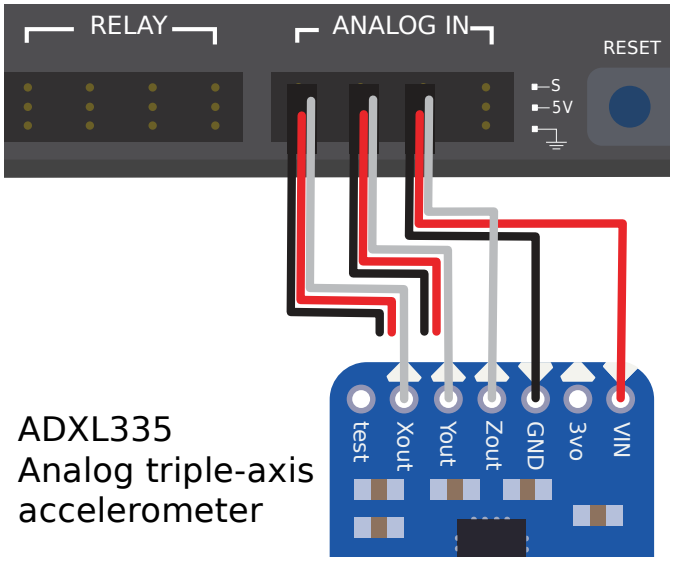
La plupart des capteurs qui se connectent aux ports d'entrée analogiques auront trois fils - signal, alimentation et masse - correspondant précisément aux trois broches des ports d'entrée analogiques. Ils doivent être connectés en conséquence.



Connexion d'un capteur à plusieurs ports d'entrée analogique

Some sensors may need to connect to multiple analog input ports in order to function. In general, these sensors will only ever require a single power and a single ground pin - only the signal pin of the additional port(s) will be needed. The image below is shows an analog accelerometer that requires three analog input ports, but similar wiring can be used for analog sensors requiring two analog input ports.

2. Toutes les broches d'alimentation sont en fait connectées à un seul rail, comme toutes les broches de masse - il n'est pas nécessaire d'utiliser les broches d'alimentation / de masse correspondant à une broche de signal donnée



38.2.2 Notes de bas de page

38.3 Potentiomètres analogiques

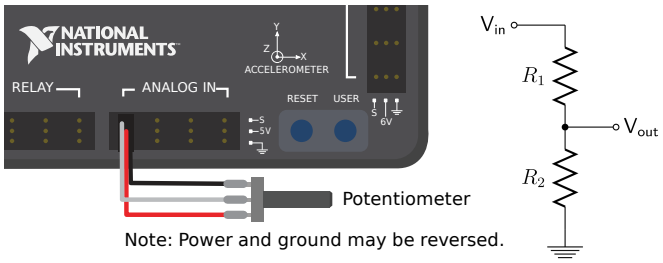
Note : Cette section couvre le matériel du potentiomètre analogique. Pour un guide logiciel sur les potentiomètres analogiques, voir *Potentiomètres analogiques - Partie logicielle*.

Avertissement : Les potentiomètres ont généralement une plage de déplacement limitée de façon mécanique. Les utilisateurs doivent faire attention que leurs mécanismes ne tournent pas leurs potentiomètres au-delà de leur course maximale (environ 270 degrés pour les modèles à un seul tour), car cela endommagerait ou détruirait le potentiomètre.

En dehors *quadrature encoders*, une autre façon courante de mesurer la rotation pour des robots FRC® est l'usage de potentiomètres analogiques. Un potentiomètre est simplement une résistance variable - comme l'arbre du potentiomètre tourne, la résistance change (généralement linéairement). Placer cette résistance dans un diviseur de tension permet à l'utilisateur de mesurer facilement la résistance en mesurant la tension à travers le potentiomètre, qui peut ensuite être utilisée pour calculer la position angulaire de l'arbre.

38.3.1 Câblage d'un potentiomètre analogique

Comme le suggèrent les noms, les potentiomètres analogiques se connectent aux ports d'"*entrées analogiques* du roboRIO. Cependant, pour comprendre comment câbler exactement les potentiomètres, il est important de comprendre leur construction.



L'image ci-dessus à gauche montre un potentiomètre typique. Il y a trois broches, tout comme il y en a sur les entrées analogiques du RIO. La broche du milieu est la broche de signal, tandis que les broches extérieures peuvent être reliés soit l'une à l'alimentation et l'autre à la masse, ou vice-versa.

Comme mentionné précédemment, un potentiomètre est un diviseur de tension, comme le montre le schéma de droite. Lorsque l'arbre du potentiomètre tourne, les résistances R1 et R2 changent ; cependant, leur somme reste constante [1]. Ainsi, la tension à travers l'ensemble du potentiomètre reste constante (pour le roboRIO, ce serait 5 volts), mais la tension entre la broche de signal et la tension ou la broche de masse varie linéairement lorsque l'arbre tourne.

Comme le circuit est symétrique, il est réversible - cela permet à l'utilisateur de choisir à quelle extrémité du trajet la tension mesurée est nulle et à quelle extrémité elle est de 5 volts. Pour inverser la directionnalité du capteur, on peut simplement le câbler à l'envers ! Assurez-vous de vérifier la directionnalité de votre potentiomètre avec un multimètre pour vous assurer qu'il est dans l'orientation souhaitée avant de souder vos fils aux contacts.

38.3.2 Notes de bas de page

38.4 Entrées numériques

Note : Cette section couvre les entrées numériques. Pour un guide logiciel sur les entrées numériques, voir *Entrées numériques - Partie logicielle*.

Un signal numérique est un signal qui peut être dans l'un de plusieurs états discrets. Dans la grande majorité des cas, le signal est la tension dans un fil, et il n'y a que deux états pour un signal numérique - haut ou bas (également dénotés 1 et 0, ou vrai et faux, respectivement).

Les ports d'entrée-sortie numériques (ou « DIO ») intégrés du roboRIO fonctionnent sur 5V, donc « haut » correspond à un signal de 5V et « bas » à un signal de 0V [1] [2].

1. La façon dont un potentiomètre fonctionne réellement est généralement de faire en sorte que l'arbre du potentiomètre contrôle la position d'un contact, ou « balai » le long d'une piste semi-circulaire de longueur fixe qui est composée d'un elément résistif - la résistance est proportionnelle à la distance entre le balai et une des extrémités de la piste.

1. Plus précisément, le signal se lit «haut» lorsqu'il monte au-dessus de 2.0 V et se lit «bas» lorsqu'il retombe en dessous de 0.8 V - le comportement entre ces deux seuils est indeterminé (soit « haut » ou »bas »).

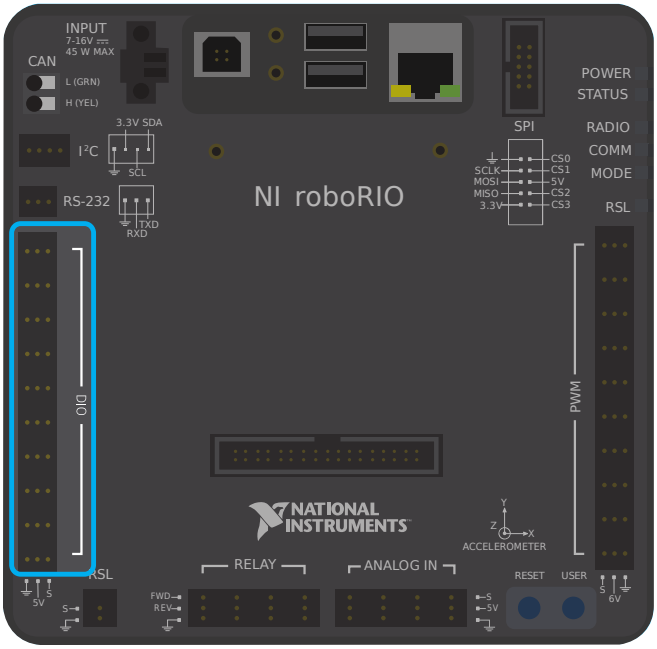
2. Le roboRIO offre également une logique 3.3 V via le port d'extension « MXP » ; cependant, son utilisation est

38.4.1 Connexion aux ports roboRIO DIO

Note : Des ports DIO supplémentaires sont disponibles via le port d'extension « MXP ». Pour les utiliser, une carte supplémentaire qui se connecte au MXP est nécessaire.

Avertissement : Consultez toujours les spécifications techniques du capteur que vous utilisez *avant* de câbler le capteur, pour vous assurer que le bon fil est connecté à la bonne broche sur le roboRIO. Le non-respect de cette consigne peut endommager soit le capteur et/ou le roboRIO.

Avertissement : Ne **jamais** connectez directement la broche d'alimentation à la broche de mise à la terre sur n'importe quel port du roboRIO ! Ce court-circuit déclenchera des fonctions de protection sur le roboRIO et entraînera un comportement inattendu.



Le roboRIO possède 10 ports DIO intégrés (numérotés de 0 à 9), comme le montre l'image ci-dessus. Chaque port a trois broches - signal (« S »), alimentation (« V ») et masse (»□ »). Les broches « alimentation » et « masse » sont utilisées pour alimenter les capteurs périphériques qui se connectent aux ports DIO - il y a une différence de potentiel constante de 5 V entre les broches « alimentation » et « masse » [3] - l'état « haut » correspond au voltage sur la broche (5V), et la « masse » à l'état « bas » (0V). La broche de signal est la broche sur laquelle le

beaucoup moins courante que le 5V.

3. Toutes les broches d'alimentation sont en fait connectées à un seul rail, comme toutes les broches de terre - il n'est pas nécessaire d'utiliser les broches d'alimentation / de terre correspondant à une broche de signal donnée.

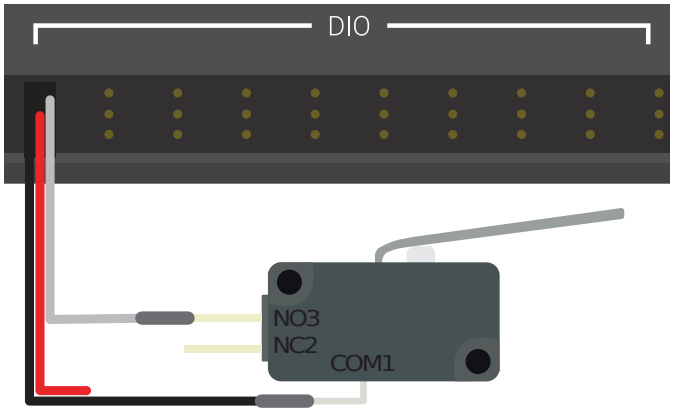
signal est réellement mesuré (ou, lorsqu'elle est utilisée comme sortie, c'est la broche qui envoie le signal).

Tous les ports DIO ont des résistances «pull-up» intégrées entre les broches d'alimentation et les broches de signal - celles-ci garantissent que lorsque la broche de signal est «flottante» (c'est-à-dire qu'elle n'est connectée à aucun circuit), elles restent constamment dans un état «haut».

Connexion d'un simple commutateur à un port DIO

Le dispositif le plus simple pouvant être connecté à un port DIO est un commutateur (comme un *interrupteur de fin de course*). Lorsqu'un commutateur est correctement connecté à un port DIO, le port indiquera « haut » lorsque le circuit est ouvert et « bas » lorsque le circuit est fermé.

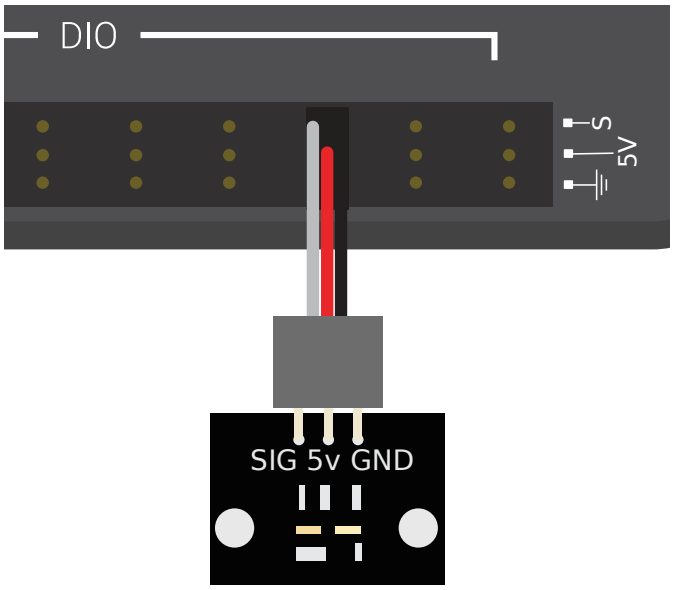
Un simple interrupteur n'a pas besoin d'être alimenté et n'a donc que deux fils. Les commutateurs doivent être câblés entre les broches *signal* et *masse* du port DIO. Lorsque le circuit de commutation est ouvert, la broche de signal flottera et la résistance de pull-up s'assurera qu'elle indique « haut ». Lorsque le circuit de commutation est fermé, il se connectera directement au rail de mise à la terre, et indiquera donc «bas».



Limit Switch or Micro Switch

Connexion d'un capteur alimenté à un port DIO

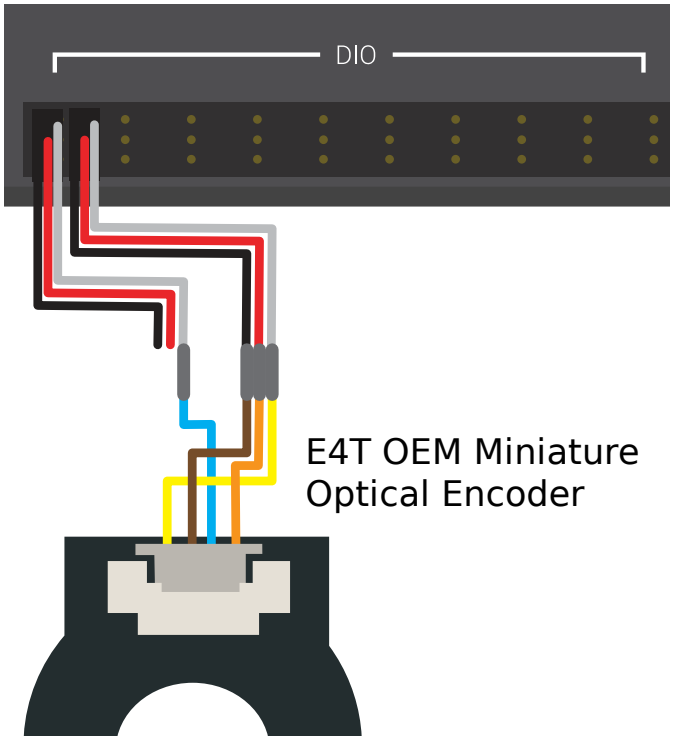
De nombreux capteurs numériques (comme la plupart des détecteurs de proximité sans contact) nécessitent de l'énergie pour fonctionner. Un capteur alimenté aura généralement trois fils - signal, alimentation et masse. Ceux-ci doivent être connectés aux broches correspondantes du port DIO.



WCP Hall Effect Sensor

Connexion d'un capteur utilisant plusieurs ports DIO

Certains capteurs (tels que les *encodeurs en quadrature*) peuvent avoir besoin de se connecter à plusieurs ports DIO pour fonctionner. En général, ces capteurs ne nécessiteront qu'une seule alimentation et une seule broche de terre - seule la broche de signal du ou des ports supplémentaires sera nécessaire.



38.4.2 Notes de bas de page

38.5 Commutateurs de proximité

Note : Cette section couvre les dispositifs de détection de proximité. Pour un guide d'utilisation des détecteurs de proximité dans le logiciel, voir *Entrées numériques - Partie logicielle*.

L'une des tâches les plus courantes sur un robot consiste à détecter lorsqu'un objet (qu'il s'agisse d'un mécanisme, d'une pièce de jeu ou d'un élément de terrain) se trouve à une certaine distance d'un point connu du robot. Ce type de détection est accompli par un « commutateur de proximité ».

38.5.1 Fonctionnement du commutateur de proximité

Les commutateurs de proximité sont comme des interrupteurs - ils font fonctionner un circuit entre un état « ouvert » (dans lequel il n'y a *pas* de connectivité à travers le circuit) et un circuit « fermé » (dans lequel il y en *a*) . Ainsi, les détecteurs de proximité génèrent un signal numérique et, par conséquent, ils sont presque toujours connectés aux ports d" *entrées numériques* du roboRIO.

Les interrupteurs de proximité peuvent être soit « normalement ouverts ou NO », dans ces interrupteurs l'activation de l'interrupteur ferme le circuit, soit « normalement fermés ou NC », dans ce cas l'activation de l'interrupteur ouvre le circuit. Certains commutateurs offrent *à la fois* un circuit NO et NC relié au même interrupteur. Dans la pratique, la différence effective entre un interrupteur de type NO et un interrupteur de type NC est le comportement

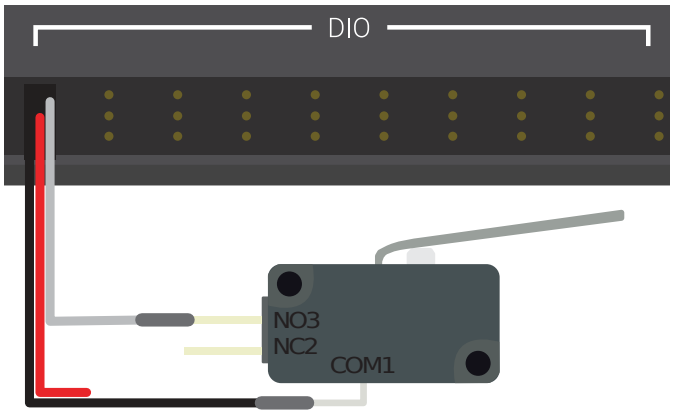
du système dans le cas où le câblage de l'interrupteur présence une défaillance, car une défaillance du câblage entraînera presque toujours un circuit ouvert. Les interrupteurs de type NC sont souvent « plus sûrs », en ce sens qu'une défaillance du câblage fait en sorte que le système se comporte comme si l'interrupteur était appuyé - puisque les interrupteurs sont souvent utilisés pour empêcher un mécanisme de s'endommager, ce qui atténue les risques de dommages au mécanisme en cas de défaillance du câblage.

Les entrées numériques sur le roboRIO ont des résistances pull-up qui feront monter l'entrée au niveau haut (ce qui correspond à la valeur 1) lorsque le commutateur est ouvert, mais lorsque le commutateur se ferme la valeur passe à 0 puisque l'entrée est maintenant connectée à la masse.

38.5.2 Types de commutateurs de proximité

Il existe plusieurs types de détecteurs de proximité qui sont couramment utilisés en FRC® :
- *Interrupteurs de proximité mécaniques (« interrupteurs de fin de course »)*
- *Interrupteurs de proximité magnétiques*
- *Interrupteurs de proximité inductifs*
- *Interrupteurs de proximité photoélectriques*
- *Interrupteurs de proximité dans le temps de vol*

Interrupteurs de proximité mécaniques («interrupteurs de fin de course»)



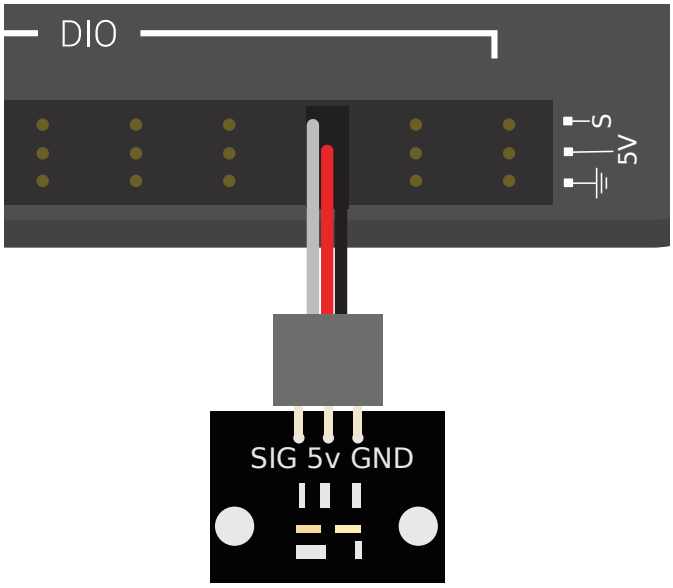
Limit Switch or Micro Switch

Les interrupteurs de proximité mécaniques (plus communément appelés « interrupteurs de fin de course ») sont probablement l'interrupteur de proximité le plus couramment utilisé en FRC, en raison de leur simplicité, leur facilité d'utilisation et de leur faible coût. Un interrupteur de fin de course est tout simplement un interrupteur attaché à un bras mécanique, généralement aux limites de la course du bras. Monté sur le bras, l'interrupteur de fin de course est activé lorsqu'il touche un objet présent sur le trajet du bras.

Les interrupteurs de fin de course varient en taille, en géométrie (relativement au bras de commutation) et au déplacement requis pour activer l'interrupteur. Bien que les interrupteurs de fin de course soient assez bon marché, leur implémentation mécanique dans un robot est parfois moins fiable que les alternatives sans contact. Cependant, ils sont extrêmement polyvalents, car ils peuvent être déclenchés par tout objet physique capable de déplacer le bras de commutation.

Voir cet *article* pour écrire le logiciel pour les interrupteurs de fin de course.

Interrupteurs de proximité magnétiques



WCP Hall Effect Sensor

Les interrupteurs de proximité magnétiques sont activés lorsqu'un aimant se trouve dans une certaine plage du capteur. En conséquence, ce sont des commutateurs « sans contact » - ils ne nécessitent pas de contact avec l'objet détecté.

Il existe deux principaux types d'interrupteurs de proximité magnétique - les interrupteurs Reed et les capteurs à effet Hall. Dans un interrupteur Reed, un champ magnétique crée une paire de contacts métalliques flexibles (les « reeds ») qui, en se touchant, ferment un circuit électrique. Un capteur à effet Hall, quant à lui, détecte la tension induite transversalement à travers un conducteur parcouru par un courant électrique. De ces deux types d'interrupteurs, les capteurs à effet Hall sont généralement les moins chers et les plus fiables. Sur la photo ci-dessus vous pouvez voir un capteur à effet Hall produit par West Coast Products.

Les détecteurs de proximité magnétiques peuvent être « unipolaires », « bipolaires » ou « omnipolaires ». Un interrupteur unipolaire s'active et se désactive en fonction de la présence d'un pôle donné de l'aimant (soit le Nord ou le Sud, selon le détecteur). Un interrupteur bipolaire s'active à partir de la proximité d'un pôle et se désactive lorsque le pôle opposé est présenté. Un interrupteur omnipolaire s'activera en présence de l'un ou l'autre des pôles et se désactivera en l'absence d'aimant.

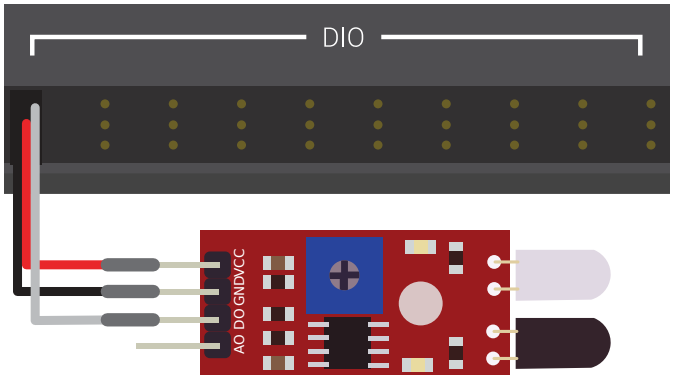
Bien que les détecteurs de proximité magnétiques soient souvent plus fiables que leurs homologues mécaniques, ils nécessitent que l'utilisateur monte un aimant sur l'objet à détecter - ils sont donc principalement utilisés pour détecter le positionnement d'un mécanisme intégré au robot.

Interrupteurs de proximité inductifs

Les interrupteurs de proximité inductifs sont activés lorsqu'un matériau conducteur se trouve dans une certaine plage du capteur. Tout comme les interrupteurs magnétiques de proximité, ce sont des détecteurs «sans contact».

Les interrupteurs de proximité inductifs sont utilisés à de nombreuses fins identiques aux interrupteurs de proximité magnétiques. Leur nature plus générale (s'activant en présence de tout conducteur, plutôt que d'un simple aimant) peut être un avantage, ou un inconvéniant, selon la nature de l'application.

Interrupteurs de proximité photoélectriques

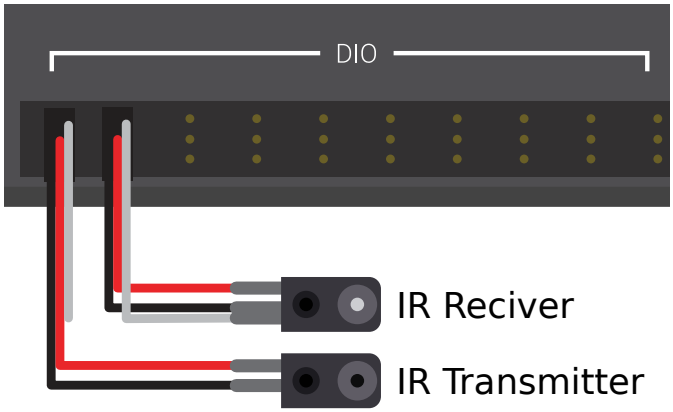


IR Digital Obstacle Avoidance Sensc

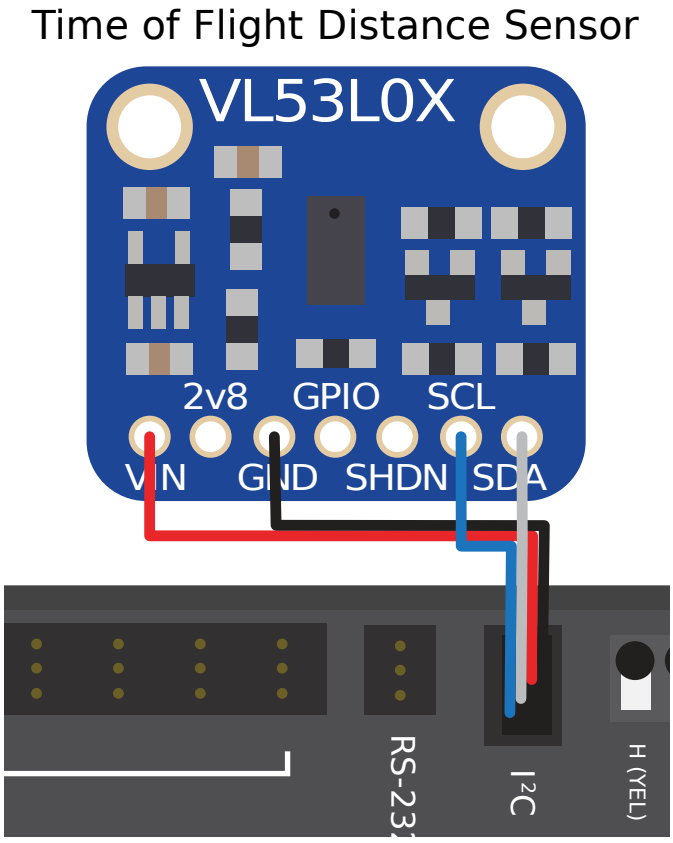
Photoelectric proximity switches are another type of no-contact proximity switch in widespread use in FRC. Photoelectric proximity switches contain a light source (usually an IR laser) and a photoelectric sensor that activates the switch when the detected light (which bounces off of the sensor target) exceeds a given threshold. One such sensor is the IR Obstacle Avoidance Module pictured above.

Étant donné que les interrupteurs de proximité photoélectriques mesurent la quantité de lumière réfléchie, leur plage de déclenchement est souvent incohérente entre différents matériaux - en conséquence, la plupart des interrupteurs photoélectriques ont un point d'activation réglable (généralement déterminé en tournant une vis quelque part sur le corps du capteur). D'autre part, les interrupteurs photoélectriques sont également extrêmement polyvalents, car ils peuvent détecter une plus grande variété d'objets que les autres types de commutateurs sans contact.

Photoelectric sensors are also often used in a « beam break » configuration, in which the emitter is separate from the sensor. These typically activate when an object is interposed between the emitter and the sensor. Pictured below is a beam break sensor with an IR LED transmitter and IR receiver.



Interrupteurs de proximité à temps de vol



Les commutateurs de proximité à temps de vol sont de conception plus récentes, et on les retrouve moins dans FRC. Ils utilisent une source de lumière concentrée, comme un mini-laser, et mesurent le temps entre l'émission de lumière et le moment où le récepteur la détecte. En utilisant la vitesse de la lumière comme référence, ils peuvent ainsi calculer la distance parcourue et effectuer une mesure de distance très précise pour une très petite zone cible. La portée de ce type de capteur peut varier considérablement, entre 30 mm et environ 1000 mm pour le capteur VL53L0X illustré ci-dessus. Il existe également des versions à plus longue portée. Vous trouverez plus d'informations sur les capteurs de temps de vol dans cet article et plus sur les circuits dans cet article.

38.6 Encodeurs

Note : Cette section traite des encodeurs. Pour un guide logiciel sur les encodeurs, voir *Encodeurs - Partie logicielle*.

L'usage d'encodeurs est de loin la méthode la plus courante pour mesurer le mouvement de rotation en FRC®, et pour cause - ils sont bon marché, faciles à utiliser et fiables. Lorsqu'ils produisent des signaux numériques, ils sont moins sujets au bruit et aux interférences que les appareils analogiques (tels que les *potentiomètres*).

38.6.1 Types d'encodeurs

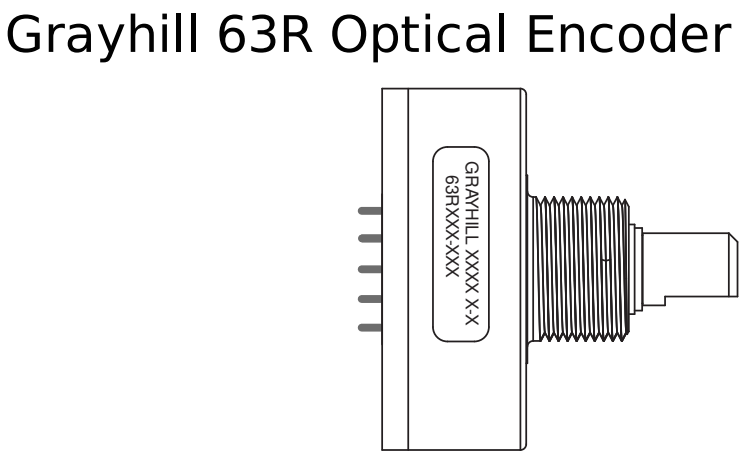
En FRC, il existe trois principales façons de brancher physiquement les encodeurs.
- *Encodeurs à arbre*
- *Encodeurs sur arbre*
- *Encodeurs magnétiques*

Ces encodeurs varient dans la façon dont ils sont montés sur le mécanisme en question. En plus de ces types d'encodeurs, de nombreux mécanismes FRC sont livrés avec des encodeurs en quadrature intégrés dans leur boîtier.

Il existe également trois manières principales de communiquer les données de l'encodeur en FRC :
- *Encodeurs à quadrature*
- *Encodeurs de rapport cyclique*
- *Encodeurs analogiques*

Note : Certains encodeurs peuvent prendre en charge plusieurs méthodes de communication.

Encodeurs à arbre

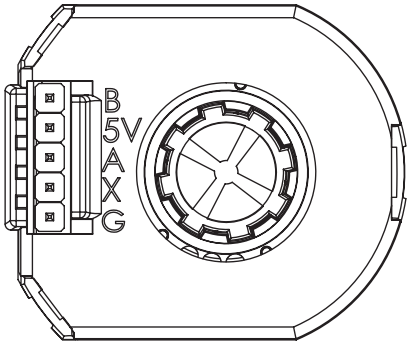


Les encodeurs à arbre ont un corps scellé avec un arbre qui en sort et qui doit être couplé en rotation à un mécanisme. Cela se fait souvent avec un couplage de faisceau hélicoïdal, ou, à moindre coût, avec une longueur de tubes flexibles (tels que les tubes chirurgicaux ou tubes pneumatiques), fixé avec des attaches de câble et/ou adhésif à chaque extrémité. De nombreuses boîtes de vitesses FRC commerciales ont des points de montage spécialement conçus pour les encodeurs à arbre.

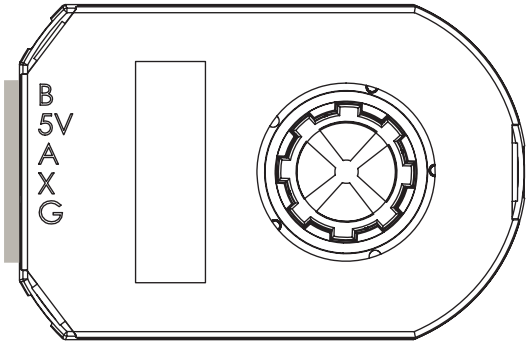
Exemples d'encodeurs à arbre :
- Grayhill 63r
- US Digital MA3

Encodeurs sur arbre

AM10 Series Modular Incremental Encoder



AMT103

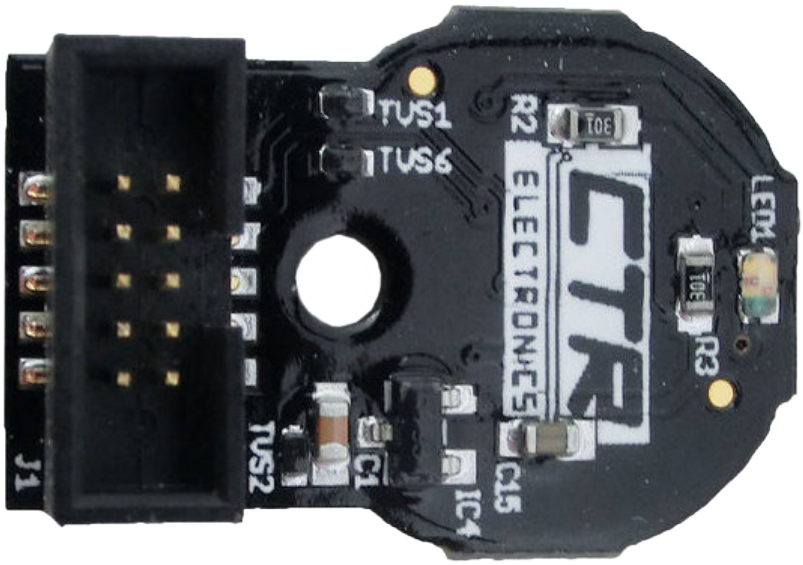


AMT102

Les encodeurs sur arbre sont couplés à un arbre en s'ajustant *autour* de celui-ci, formant un accouplement à friction entre l'arbre et un moyeu rotatif à l'intérieur de l'encodeur.

Exemples d'encodeurs sur arbre :
- AMT103-V disponible via FIRST Choice
- CIMcoder
- REV Through Bore Encoder
- US Digital E4T

Encodeurs magnétiques



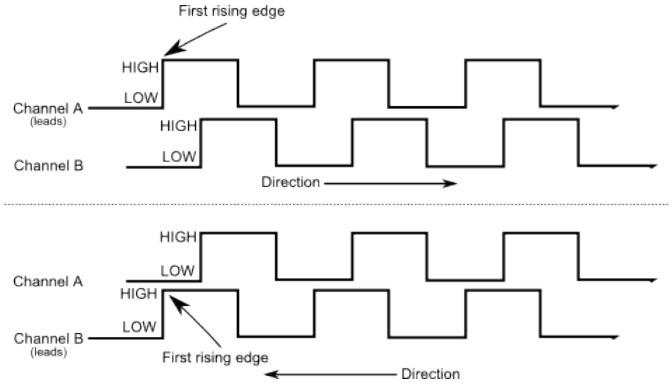
Les encodeurs magnétiques ne nécessitent aucun accouplement mécanique à l'arbre ; ils suivent plutôt l'orientation d'un aimant qui est fixé à l'arbre. Bien que la nature sans contact des encodeurs magnétiques puisse être pratique, ils nécessitent souvent une construction précise afin de s'assurer que l'aimant est correctement positionné par rapport à l'encodeur.

Exemples d'encodeurs magnétiques :

— CTRE Mag Encoder
— Thrifty Absolute Magnetic Encoder
— Team 221 Lamprey2

Encodeurs à quadrature

Le terme « quadrature » fait référence à la méthode par laquelle le mouvement est mesuré / codé. Un codeur en quadrature produit deux impulsions carrées qui sont déphasées de 90 degrés l'une par rapport à l'autre, comme le montre l'image ci-dessous :

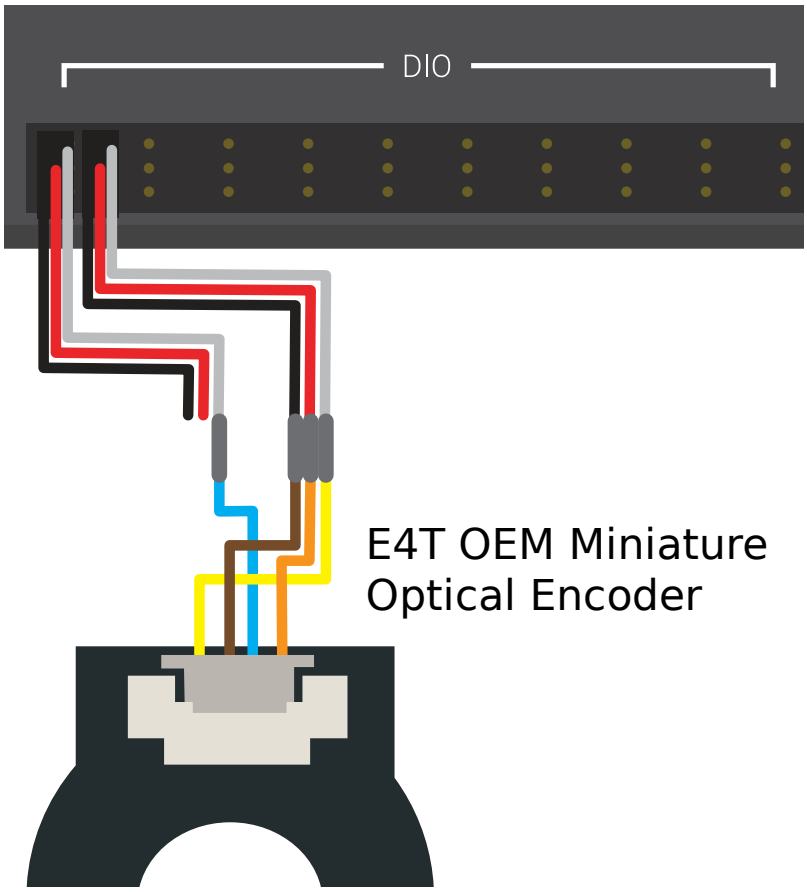


Ainsi, sur les deux canaux, il y a quatre « fronts » totaux par période (d'où « quad »). L'utilisation de deux impulsions déphasées permet de déterminer sans ambiguïté la direction du mouvement en déterminant quelle impulsion « est en avance » sur autre.

Comme chaque impulsion d'onde carrée est un signal numérique, les encodeurs en quadrature se connectent aux ports *entrée numérique* du roboRIO.

Exemples d'encodeurs à quadrature :
- AMT103-V disponible via FIRST Choice
- CIMcoder
- CTRE Mag Encoder
- Grayhill 63r
- REV Through Bore Encoder
- US Digital E4T

Câblage d'un encodeur à quadrature



Les encodeurs en quadrature, tels que l'"encodeur optique miniature OEM E4T, peuvent être câblés à deux ports d'entrée numériques, tel qu'indiqué ci-dessus.

Index

Certains encodeurs en quadrature ont une troisième broche d'index qui émet une impulsion lorsque l'encodeur effectue un tour.

Résolution de l'encodeur à quadrature

Avertissement : Les acronymes « CPR » et « PPR » sont utilisés indifféremment par plusieurs sources pour désigner les deux fronts de montée par révolution (PPR = Pulses per Revolution) *et* les cycles par révolution (CPR = Cycles per Revolution), de sorte que l'acronyme seul ne suffit pas pour dire ce que le nombre associé à la valeur représente « dans la vraie vie ». Il est préférable de se fier à la documentation technique de chaque encodeur pour obtenir l'information souhaitée.

Comme les encodeurs mesurent la rotation avec des impulsions numériques, la précision de la mesure est limitée par le nombre d'impulsions pour un angle donné, ou une rotation complète. C'est ce que l'on appelle la «résolution» de l'encodeur, et elle est traditionnellement mesurée de deux manières différentes : fronts par tour ou cycles par tour.

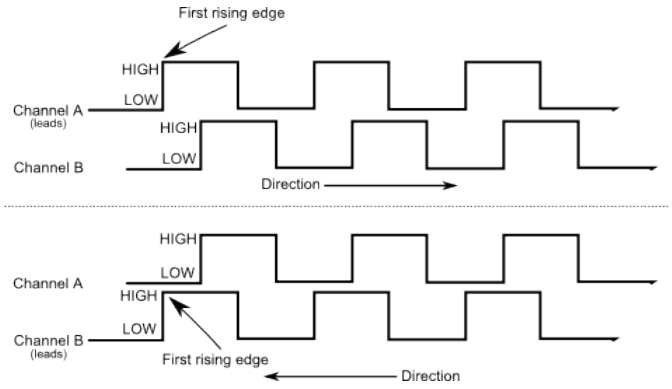
Le terme *Fronts par révolution* fait référence au nombre total de transitions (soit du haut vers le bas, ou bas vers le haut) sur les deux canaux, pour un tour complet de l'arbre de l'encodeur. Une période complète contient *quatre* fronts, et un tour contient plusieurs périodes.

Le terme *Cycles par tour* fait référence au nombre total de *périodes complètes* des deux canaux par tour de l'arbre de l'encodeur. Une période complète est *un* cycle.

Ainsi, une résolution indiquée en fronts par tour a une valeur quatre fois supérieure à la même résolution indiquée en cycles par tour.

En général, la résolution de votre encodeur en fronts par révolution doit être supérieure à la plus petite erreur acceptable de positionnement. Ainsi, si vous voulez obtenir une précision angulaire de plus ou moins un degré dans un mécanisme, vous devriez utiliser un encodeur avec une résolution au moins supérieure à 360 fronts par tour.

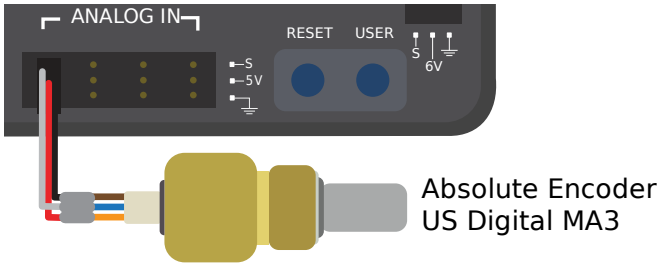
Encodeurs de rapport cyclique



Les encodeurs de rapport cyclique se connectent à une seule entrée numérique sur le roboRIO. Ils génèrent une impulsion dont la longueur est proportionnelle à la position absolue de l'encodeur.

Exemples d'encodeurs Duty Cycle :
- AndyMark Mag Encoder
- CTRE Mag Encoder
- REV Through Bore Encoder
- Team 221 Lamprey2
- US Digital MA3

Encodeurs analogiques



Un encodeur analogique se connecte à une entrée analogique du roboRIO. Il produit une tension proportionnelle à sa position absolue.

Exemples d'encodeurs analogiques :
- Team 221 Lamprey2
- Thrifty Absolute Magnetic Encoder
- US Digital MA3

38.7 Gyroscopes

Note : Cette section couvre le matériel du gyroscope. Pour un guide logiciel sur les gyroscopes, voir *Gyroscopes - Partie logicielle*.

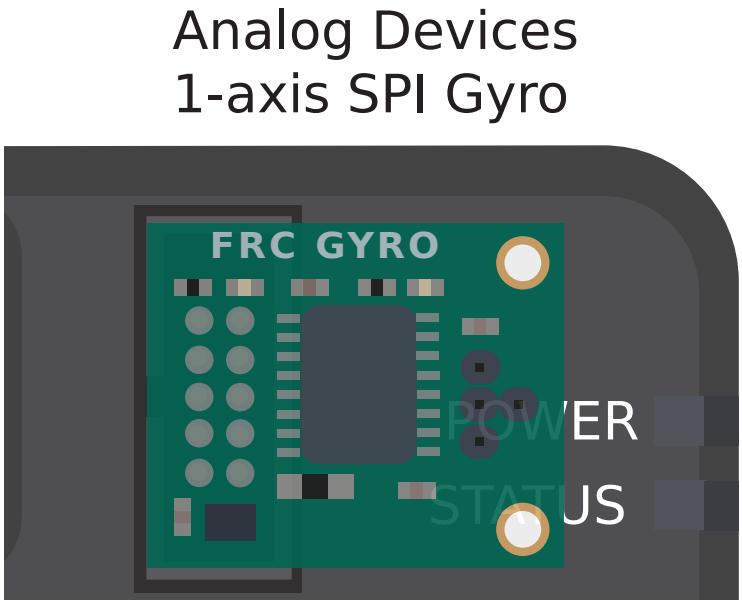
Les gyroscopes sont des dispositifs qui mesurent le taux de rotation. Ceux-ci sont particulièrement utiles pour stabiliser la conduite du robot ou pour mesurer le cap ou l'inclinaison en intégrant (additionnant) les mesures de vitesse pour obtenir une mesure du déplacement angulaire total.

Plusieurs dispositifs populaires en FRC® connus sous le nom de *IMUs* (Inertial Measurement Units) combinent gyros à 3 axes, accéléromètres et autres capteurs de position en un seul appareil. Voici quelques exemples populaires :
- IMU Analog Devices ADIS16448 et ADIS 16470
- **`IMU CTRE Pigeon<https ://store.ctr-electronics.com/gadgeteer-pigeon-imu/>`__**
- Kauai Labs NavX

38.7.1 Types de gyroscopes

Il existe deux types de gyroscopes couramment utilisés en FRC : les gyroscopes à axe unique, les gyroscopes à trois axes et les IMU, qui comprennent souvent un gyroscope à 3 axes.

Gyroscopes à axe unique



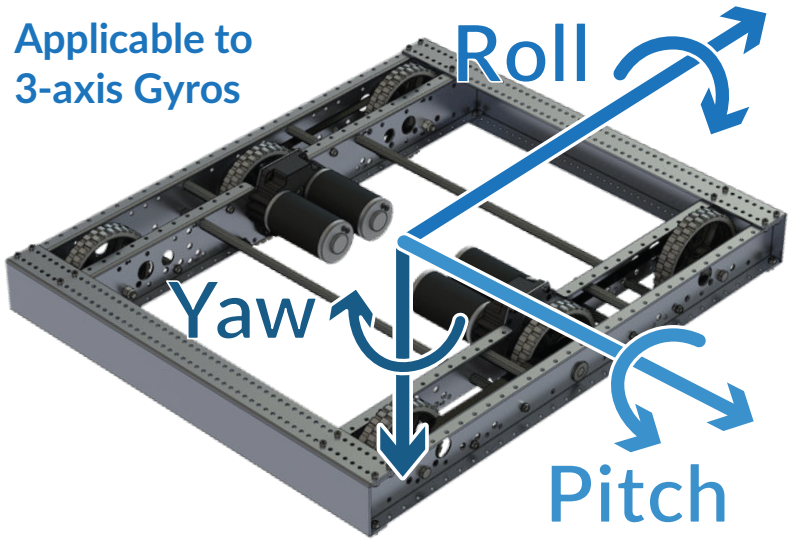
Selon leur nom, les gyroscopes à axe unique mesurent le taux de rotation autour d'un seul axe. Cet axe est généralement spécifié sur le composant physique, et son installation dans la bonne orientation, de telle sorte que l'axe désiré est mesuré, est très important. Certains gyroscopes à axe unique peuvent produire une tension analogique correspondant au taux de rotation mesuré, et ainsi se connecter aux ports **:doc :`d'entrée analogiques <analog-inputs-hardware> ` du roboRIO. D'autres gyroscopes à axe unique, tels que le `ADXRS450 <https ://wiki.analog.com/first>`__** sur la photo ci-dessus, utilisent plutôt le *port SPI* du roboRIO.

Le module gyroscopique FRC Analog Devices ADXRS450 qui faisait partie de FIRST Choice ces dernières années est un gyroscope à axe unique couramment utilisé.

Gyroscopes à trois axes

Les gyroscopes à trois axes mesurent le taux de rotation autour des trois axes spatiaux (généralement étiquetés x, y et z). Le mouvement autour de ces axes est appelé tangage, lacet et roulis (ou respectivement pitch, yaw et roll en anglais).

Le module IMU pour FRC **`Analog Devices ADIS16470<https ://www.analog.com/en/landing-pages/001/first.html>`__** qui faisait partie de FIRST Choice ces dernières années est un gyroscope à trois axes couramment utilisé.



Note : Le système de coordonnées illustré ci-dessus est souvent utilisé pour les gyroscopes à trois axes, car il s'agit d'une convention en avionique. Notez que d'autres systèmes de coordonnées sont utilisés en mathématiques et référencés dans WPILib. Veuillez vous référer au *diagramme suivant* pour les axes référencés dans le logiciel.

Les gyroscopes périphériques à trois axes peuvent simplement produire trois tensions analogiques (et donc se connecter aux *ports d'entrée analogiques*, ou (plus communément) ils peuvent communiquer avec l'un des *bus série* du roboRIO.

38.8 Ultrasons

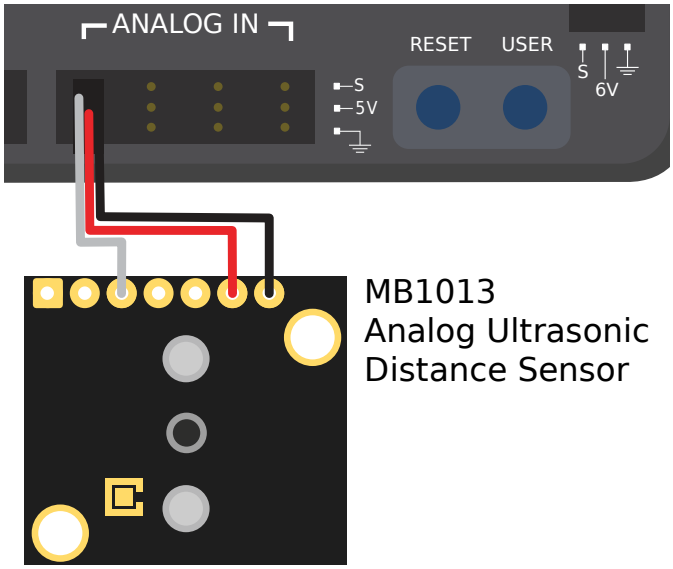
Note : Cette section couvre les dispositifs de capteurs à ultrasons. Pour un guide logiciel sur les ultrasons, voir *Ultrasons - Partie logicielle*.

Les télémètres à ultrasons sont quelques-uns des télémètres les plus couramment utilisés en FRC®. Ils sont bon marché, faciles à utiliser et assez fiables. Les télémètres à ultrasons fonctionnent en émettant une impulsion sonore à haute fréquence, puis en mesurant le temps qu'il faut à l'écho pour atteindre le capteur après avoir rebondi sur la cible. À partir du temps mesuré et de la vitesse du son dans l'air, il est possible de calculer la distance par rapport à la cible.

38.8.1 Types de capteurs ultrasons

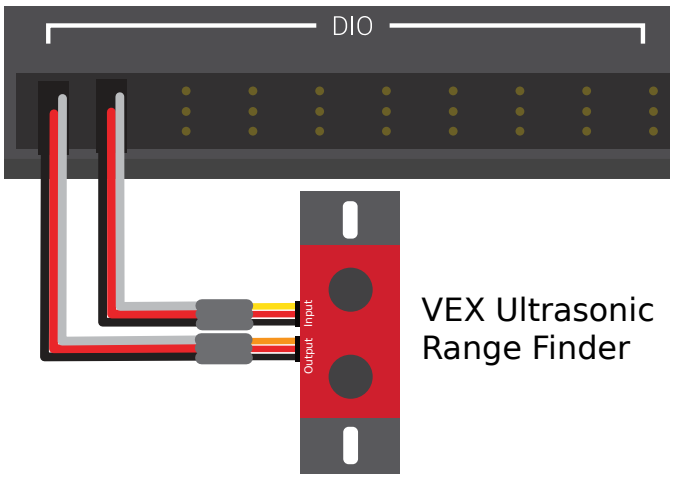
Bien que tous les télémètres à ultrasons fonctionnent selon le principe de «réponse ping» décrit ci-dessus, ils peuvent varier dans leur façon de communiquer avec le roboRIO.

Ultrasons avec sorties analogiques



Les détecteurs à ultrasons analogiques produisent une simple tension analogique correspondant à la distance à la cible, et se connectent ainsi à un port analog input. L'utilisateur devra calibrer la conversion tension-distance dans *software*.

Capteurs ultrasoniques à réponse ping



Alors que, comme mentionné, tous les capteurs ultrasoniques sont des dispositifs fonctionnellement à réponse ping, celui-ci est configuré pour se connecter *à la fois une entrée numérique et une sortie numérique*. La sortie numérique est utilisée pour envoyer le ping, tandis que l'entrée est utilisée pour lire la réponse.

Capteurs ultrasoniques avec bus série

Certains capteurs à ultrasons plus compliqués peuvent communiquer avec le RIO sur l'un des *bus série*, tels que RS-232.

38.8.2 Avertissements

Les capteurs à ultrasons sont généralement assez faciles à utiliser, mais il y a quelques mises en garde. Comme les ultrasons fonctionnent en mesurant le temps entre l'impulsion et son écho, ils mesurent généralement la distance uniquement à la cible *la plus proche* dans leur plage. Il est donc extrêmement important de choisir le bon capteur pour le travail. La documentation des capteurs à ultrasons comprendra généralement une image du «diagramme de faisceau» qui montre la forme de la «fenêtre» dans laquelle les ultrasons détecteront une cible - faites très attention à cela lors de la sélection de votre capteur.

Les capteurs à ultrasons sont également sensibles aux interférences d'autres capteurs à ultrasons. Afin de minimiser cela, le roboRIO peut exécuter des ultrasons à réponse ping de manière alternée, ou « round-robin » - cependant, en compétition, il n'y a aucun moyen sûr de s'assurer que les interférences des capteurs montés sur d'autres robots ne viennent pas entrer en conflit avec nos capteurs.

Enfin, les capteurs ultrasoniques peuvent être dans l'impossibilité de détecter des objets qui absorbent les ondes sonores ou qui les redirigent de manière étrange. Ainsi, ils fonctionnent mieux pour détecter des objets durs et plats.

38.9 Accéléromètres

Les accéléromètres sont des capteurs courants utilisés pour mesurer l'accélération.

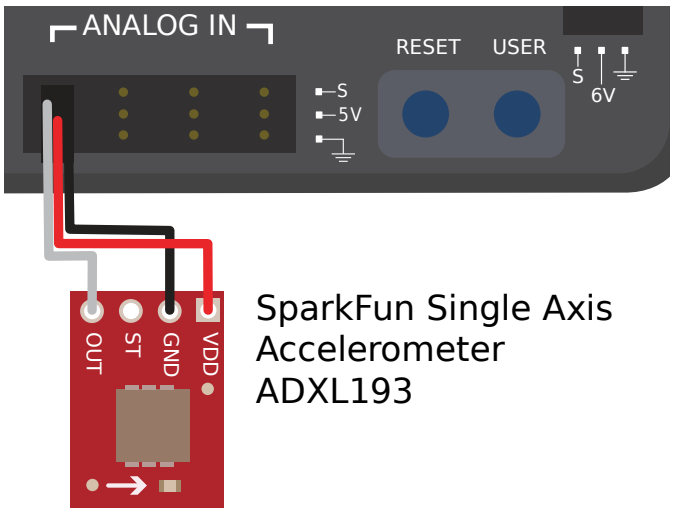
En principe, des mesures précises de l'accélération peuvent être intégrées deux fois de suite et utilisées pour suivre la position (de la même manière que la mesure du taux de rotation à partir d'un gyroscope peut être intégrée pour déterminer le cap) - cependant, dans la pratique, les accéléromètres qui sont disponibles dans gamme de prix légale autorisée en FRC® ne sont quasiment pas précis pour cet usage. Toutefois, les accéléromètres sont toujours utiles pour un certain nombre de tâches en FRC.

Le roboRIO contient un *accéléromètre à trois axes intégré* que toutes les équipes peuvent utiliser, mais les équipes qui recherchent des mesures plus précises peuvent acheter et utilisez également un accéléromètre périphérique.

38.9.1 Types d'accéléromètres

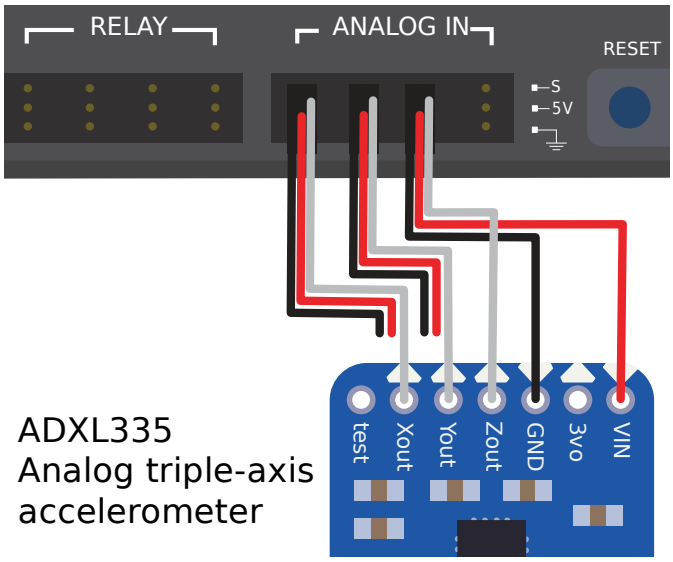
Il existe trois types d'accéléromètres couramment utilisés en FRC : les accéléromètres à axe unique, les accéléromètres à axes multiples et les IMU.

Accéléromètres à axe unique



Comme leur nom l'indique, les accéléromètres à axe unique mesurent l'accélération le long d'un axe unique. Cet axe est généralement spécifié sur l'appareil physique, et le montage de l'appareil dans la bonne orientation est primordial. afin que l'axe souhaité soit mesuré convenablement. Les accéléromètres à axe unique produisent généralement une tension analogique correspondant à l'accélération mesurée, et se connectent donc aux *ports d'entrée analogiques* du roboRIO.

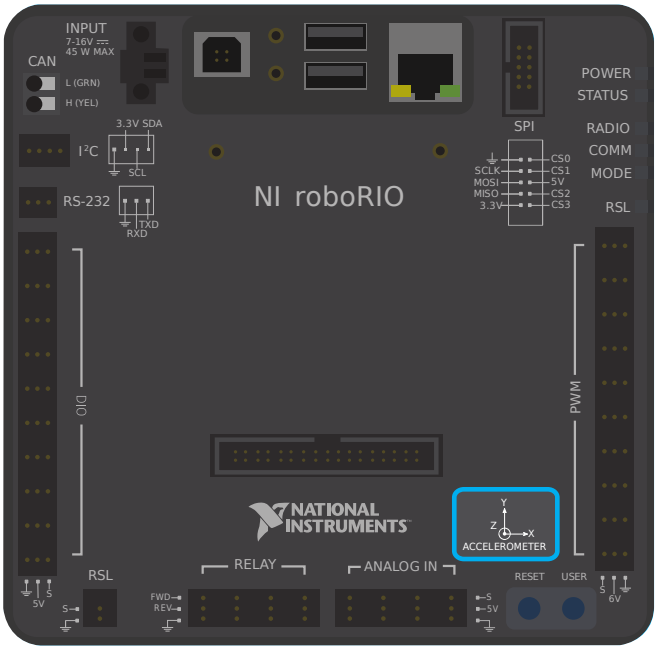
Accéléromètres multi-axes



Les accéléromètres multi-axes mesurent l'accélération le long de plusieurs axes spatiaux. L'accéléromètre intégré du roboRIO est un accéléromètre à trois axes.

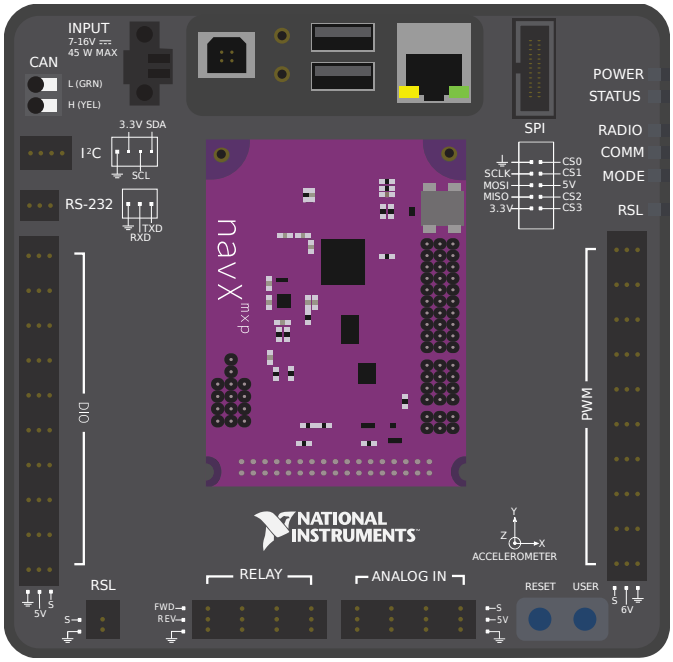
Les accéléromètres multi-axes peuvent simplement produire plusieurs tensions analogiques (et donc se connecter à plusieurs *ports d'entrée analogiques*, ou (plus communément) ils peuvent communiquer de façon sérielle avec l'un des *bus série* du roboRIO.

L'accéléromètre intégré du roboRIO



Le roboRIO dispose d'un accéléromètre intégré, qui ne nécessite aucune connexion externe. Vous pouvez trouver plus de détails sur son utilisation dans la *section Accéléromètre intégré* de la documentation du logiciel.

IMU (unités de mesure inertielle)



Plusieurs dispositifs FRC populaires (appelés «unités de mesure inertielles» ou «IMU») combinent à la fois un accéléromètre et un gyroscope. Voici quelques exemple des modèles les

plus populaires :

- Analog Devices ADIS16448 and ADIS 16470 IMUs
- **`IMU CTRE Pigeon<https ://store.ctr-electronics.com/gadgeteer-pigeon-imu/>`__**
- Kauai Labs NavX

38.10 Les capteurs de type LIDAR

Les capteurs LIDAR (détection de lumière et de portée) sont une variété de télémètres qui connaissent une utilisation croissante en FRC®.

Les capteurs LIDAR fonctionnent de manière assez similaire aux *capteurs à ultrasons*, mais utilisent la lumière au lieu du son. Un laser est pulsé et le capteur mesure le temps nécessaire pour que l'impulsion rebondisse.

38.10.1 Types de LIDAR

Il existe deux types de capteurs LIDAR couramment utilisés dans les FRC actuels : le LIDAR à 1 dimension et le LIDAR à 2 dimensions.

LIDAR 1 dimension

Un capteur LIDAR unidimensionnel (1D) fonctionne un peu comme un capteur à ultrasons - il mesure la distance à l'objet le plus proche le long d'une ligne devant lui. Les capteurs LIDAR 1D peuvent souvent être plus fiables que les ultrasons, car ils ont des «profils de faisceau»

plus étroits et sont moins sensibles aux interférences. Sur la photo ci-dessus, le Capteur de distance optique Garmin LIDAR-Lite.

Les capteurs 1D LIDAR génèrent normalement une tension analogique proportionnelle à la distance mesurée et se connectent ainsi aux ports *d'entrée analogiques* du roboRIO ou ceux qui ont un port série à l'un des *bus série du roboRIO*.

LIDAR bidimensionnel

Un capteur LIDAR bidimensionnel (2D) mesure la distance dans toutes les directions dans un plan. Généralement, ceci est réalisé (plus ou moins) en plaçant simplement un capteur LIDAR 1D sur une plaque tournante qui tourne à une vitesse constante.

Étant donné que, par nature, les capteurs LIDAR 2D doivent renvoyer une grande quantité de données au roboRIO, ils se connectent presque toujours à l'un des *bus série* du roboRIO.

38.10.2 Avertissements

Les capteurs LIDAR souffrent de quelques inconvénients courants :

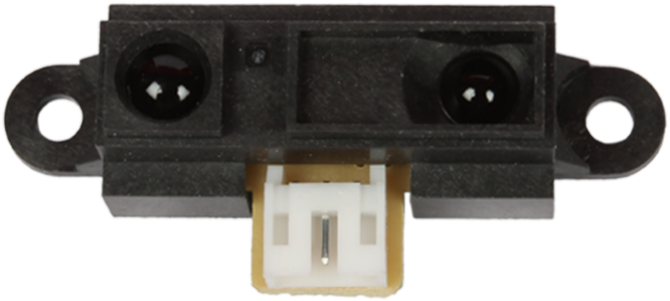
Comme les ultrasons, le LIDAR se fie sur la réflexion de l'impulsion émise vers le capteur. Ainsi, le LIDAR dépend de manière critique de la réflectivité du matériau dans la longueur d'onde du laser. La paroi de champ FRC est en polycarbonate (qui est parfaitement légal pour une utilisation FRC), cependant, ce matériau a tendance à être transparent dans la longueur d'onde infrarouge . Ainsi, le capteur LIDAR a de la difficulté à détecter la barrière du terrain de jeu.

Les capteurs 2D LIDAR (dans la fourchette de prix légale pour une utilisation FRC) ont tendance à être assez flous et le traitement de leurs données mesurées (connu sous le nom de « nuage de points ») peut impliquer beaucoup de logiciels complexes. De plus, il existe très peu de capteurs LIDAR 2D spécialement conçus pour FRC, de sorte que le support logiciel a tendance à être rare.

Comme les capteurs 2D LIDAR reposent sur une platine (table tournante) pour fonctionner, leur taux de mise à jour est limité par la vitesse à laquelle la platine tourne. Pour les capteurs dans la gamme de prix légale pour FRC, cela signifie souvent que la mise à jour des données est lente, ce qui peut limiter la capacité de détecter des cibles lorsque le robot se déplace rapidement.

De plus, comme les capteurs LIDAR 2D sont limités en résolution *angulaire*, la résolution *spatiale* du nuage de points est pire lorsque les cibles sont plus éloignées.

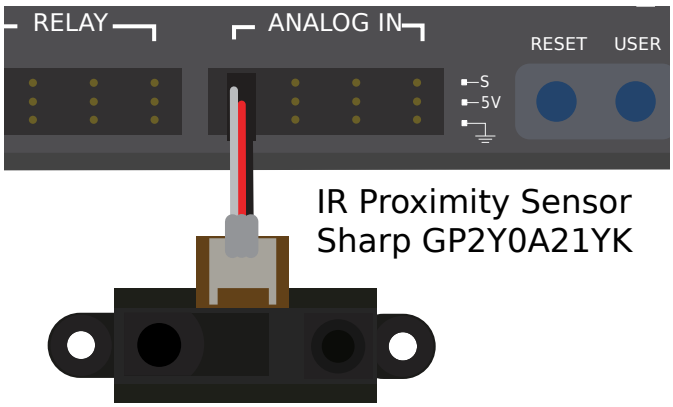
38.11 Télémètres triangulaires



Les télémètres de triangulation (souvent appelés « télémètres IR », car ils fonctionnent couramment dans la bande de longueur d'onde infrarouge) sont un autre type commun de télémètre utilisé en FRC®. Le capteur ci-dessus est un capteur commun de marque Sharp

Contrairement à *LIDAR*, les télémètres triangulaires ne mesurent pas le temps entre l'émission d'une impulsion et la réception d'une réflexion. Au contraire, la plupart des télémètres IR fonctionnent en émettant un faisceau constant à un léger angle et en mesurant la position du faisceau réfléchi. Plus le point de contact du faisceau réfléchi est proche de l'émetteur, plus l'objet est proche du capteur.

38.11.1 Utilisation de télémètres IR



Les télémètres IR émettent généralement une tension analogique proportionnelle à la distance de la cible, et se connectent ainsi aux ports *entrée analogique* sur le RIO.

38.11.2 Avertissements

Les télémètres IR présentent des inconvénients similaires aux capteurs LIDAR 1D - ils sont très sensibles à la réflectivité de la cible dans la longueur d'onde du laser émis.

De plus, bien que les télémètres infrarouges aient tendance à offrir une meilleure résolution que les capteurs LIDAR lorsqu'ils mesurent à de courtes distances, ils sont généralement plus sensibles aux différences d'orientation de la cible, *surtout* si la cible est hautement réfléchissante (comme un miroir).

38.12 Bus Séries

En plus des entrées *numériques* et *analogiques*, le roboRIO fournit aussi plusieurs méthodes de communication série avec des appareils périphérique.

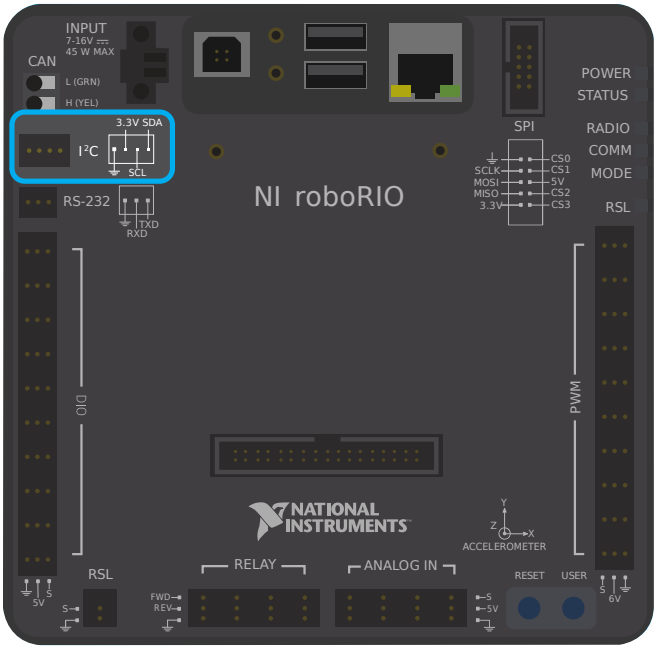
Les entrées numériques et analogiques sont très limitées avec la quantité de données qu'elles peuvent transmettre. Les bus série permettent l'usage de protocoles de communication beaucoup plus robustes et à plus large bande passante avec des senseurs qui collectent un grand nombre de données, commes des Unités de Mesure Inertielle (IMUs) ou des détecteurs LiDAR 2D.

38.12.1 Types de bus série supportés.

Le roboRIO supporte plusieurs protocoles de communications série :
- *I2C*
- *SPI*
- *RS-232*
- *USB Host*
- *Bus CAN*

De plus, le roboRIO prend en charge les communications avec les périphériques via le bus *CAN*. Cependant, comme le FRC® Le protocole CAN est assez idiosyncratique, relativement peu de capteurs périphériques le prennent en charge (bien qu'il soit largement utilisé pour les contrôleurs de moteur).

38.12.2 I2C



I²C Port

Figure 6 and Table 5 describe the I²C port pins and signals.

Figure 6. I²C Port Pinout

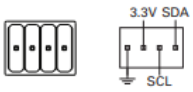


Table 5. I²C Port Signal Descriptions

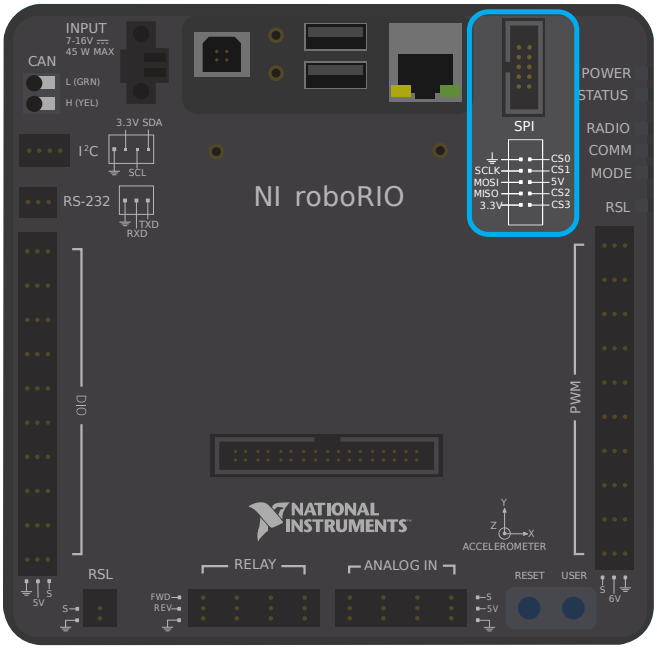
Signal Name	Direction	Description
GND	—	Reference for digital lines and +3.3 V power output.
3.3V	Output	+3.3 V power output.
SCL	Input or Output	I²C lines with 3.3 V output, 3.3 V/ 5 V-compatible input. Refer to the *I²C Lines* section for more information.
SDA	Input or Output	

Pour communiquer avec des périphériques via *I2C*, chaque broche doit être connectée à sa broche correspondante sur l'appareil. I2C permet aux utilisateurs de câbler une « chaîne » d'appareils esclaves à un seul port, à condition que ces appareils aient des identifiants distincts.

Le bus I2C peut également être utilisé via le « port d'extension MXP »_. Le bus I2C sur le *MXP* est indépendant. Par exemple, un appareil sur le bus principal peut avoir le même ID qu'un appareil sur le bus MXP.

Avertissement : Assurez-vous de vous familiariser avec le problème connu suivant avant d'utiliser le port I2C intégré : docs/yearly-overview/known-issues :I2C intégré provoquant des blocages du système

38.12.3 SPI



SPI Port

Figure 13 and Table 12 describe the SPI port pins and signals.

Figure 13. SPI Port Pinout

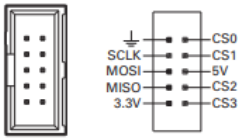


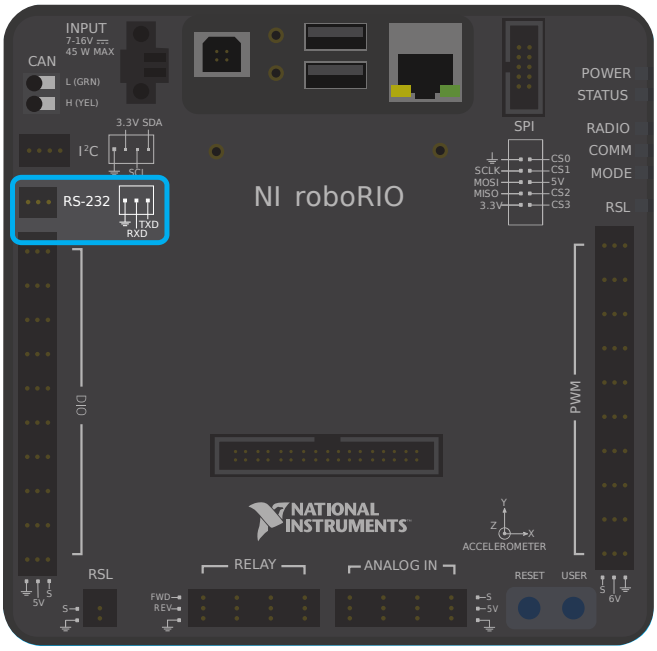
Table 12. SPI Port Signal Descriptions

Signal Name	Direction	Description
3.3V	Output	+3.3 V power output.
5V	Output	+5 V power output.
CS <0..3>	Output	SPI with 3.3 V output, 3.3 V/5 V-compatible input. Refer to the *SPI Lines* section for more information.
SCLK	Output	
MOSI	Output	
MISO	Input	
GND	—	Reference for digital lines and +3.3 V and +5.5 V power output.

Pour communiquer avec des périphériques via *SPI*, chaque broche doit être connectée à sa broche correspondante sur l'appareil. Le port SPI prend en charge les communications avec jusqu'à quatre appareils (correspondant aux broches Chip Select (CS) 0-3 sur le schéma ci-dessus).

Le bus SPI peut aussi être utilisé via le *port d'expansion MXP*. Le port MXP fournit une horloge indépendante (CLK), et des lignes d'entrée/sortie (MISO et MOSI) et un CS supplémentaire.

38.12.4 RS-232



RS-232 Port

Figure 7 and Table 6 describe the RS-232 port pins and signals.

Figure 7. RS-232 Serial Port Pinout

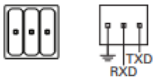


Table 6. RS-232 Serial Port Signal Descriptions

Signal Name	Direction	Description
TXD	Output	Serial transmit output with ±5 V to ±15 V signal levels. Refer to the *UART and RS-232 Lines* section for more information.
RXD	Input	Serial receive input with ±15 V input voltage range. Refer to the *UART and RS-232 Lines* section for more information.
GND	—	Reference for digital lines.

Pour communiquer aux dispositifs périphériques via RS-232, chaque broche doit être reliée à la broche correspondante sur le dispositif. Faire attention au sens de la transmission, quelquefois le signal RX doit être relié au TX et vice-versa.

Le bus RS-232 peut aussi être utilisé via le *port d'expansion MXP*.

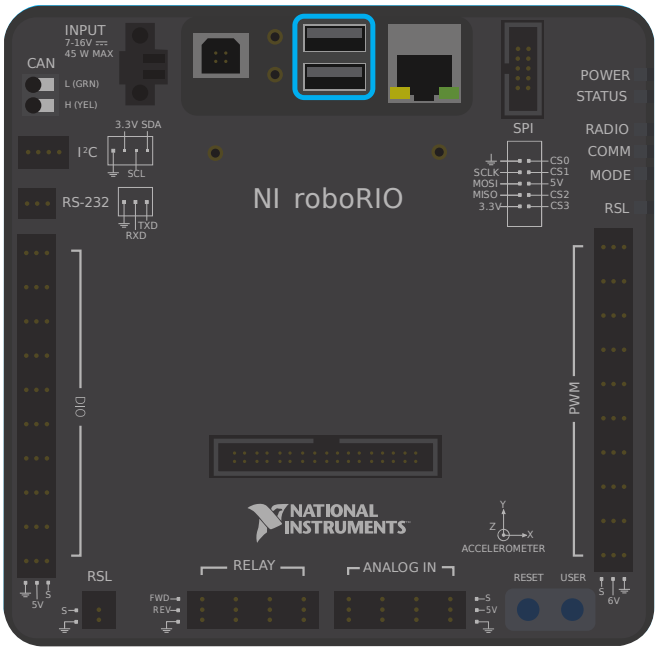
Le port série RS-232 du roboRIO utilise les niveaux de signalisation RS-232 (+/- 15 Volts). le port série du MXP utilise les niveaux de signalisation CMOS (+/- 3.3 Volts).

Note : Par défaut, le port RS-232 intégré est utilisé par la console série du roboRIO. Afin de l'utiliser pour un périphérique externe, la console série doit être désactivée et pour ce faire consultez la section *Installation de l'image dans votre roboRIO* ou la section *Tableau de bord Web du roboRIO*.

38.12.5 Client USB

L'un des ports USB du roboRIO est un port client USB-B ou USB. Celui-ci peut être connecté à des appareils, tels qu'un ordinateur Driver Station, avec un câble USB standard.

38.12.6 USB Host



Deux des ports USB du roboRIO sont un port hôte USB-A ou USB. Ceux-ci peuvent être connectés à des dispositifs, tels que des caméras ou des capteurs, avec un câble USB standard.

38.12.7 Port d'extension MXP

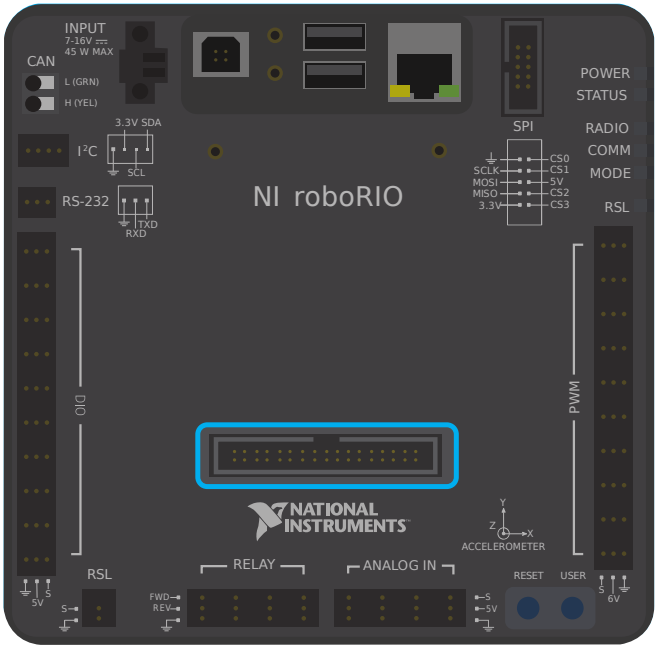
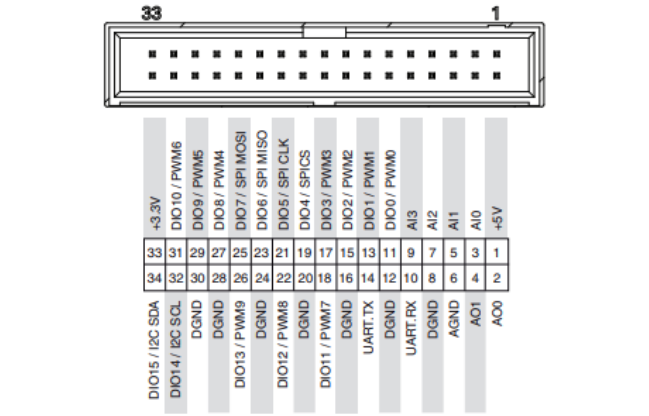


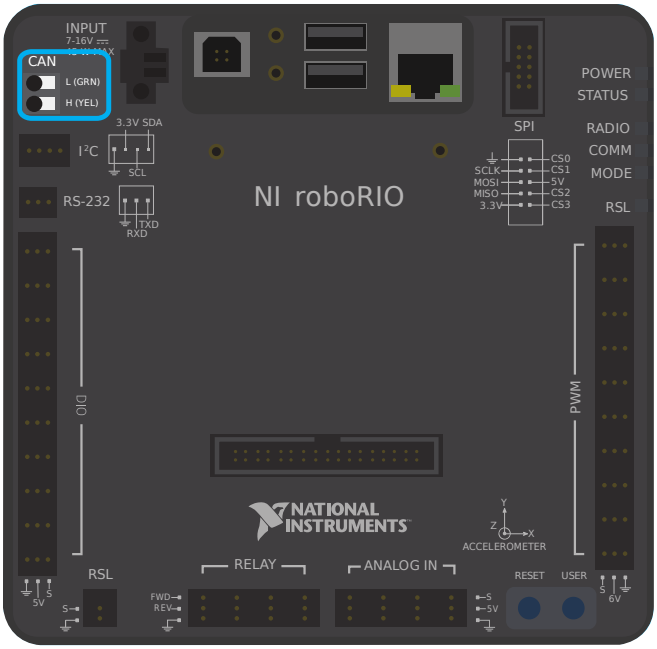
Figure 4. MXP Pinout



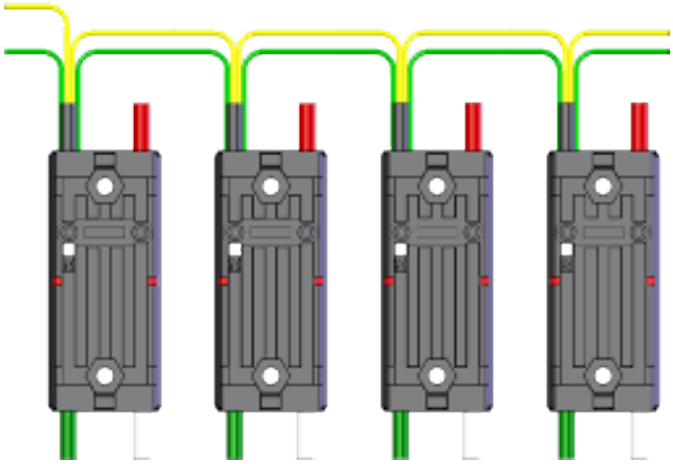
Plusieurs bus série sont également disponibles pour une utilisation via le port d'extension *MXP* du roboRIO. Ce port permet aux utilisateurs d'utiliser de nombreuses entrées supplémentaires *digital* et *analog*, ainsi que les différents bus série.

De nombreux périphériques se connectent directement au port MXP pour plus de commodité, ne nécessitant aucun câblage de la part de l'utilisateur.

38.12.8 Bus CAN



De plus, le roboRIO prend en charge les communications avec les périphériques via le bus CAN. Cependant, comme le protocole FRC CAN est assez particulier, relativement peu de capteurs périphériques le prennent en charge (bien qu'il soit largement utilisé pour les contrôleurs de moteur). L'un des avantages de l'utilisation du protocole de bus CAN est que les périphériques peuvent être connectés en « guirlande », ou « daisy-chain », comme illustré ci-dessous. Si l'alimentation est coupée de n'importe quel dispositif situé sur la chaîne, les signaux de données pourront toujours continuer de circuler sur la chaîne.



Plusieurs capteurs utilisent principalement le bus CAN. Quelques exemples :

- Capteur de distance/distance basé sur le temps de vol CAN de playwithfusion.com
- Capteurs basés sur TalonSRX, tels que le Gadgeteer Pigeon IMU et l'encodeur SRX MAG Encoder
- CANifier
- Capteurs de surveillance de puissance intégrés dans le *Panneau de distribution de puissance CTRE (PDP)* et le *Concentrateur de distribution de puissance REV (PDH)*

Pour plus d'informations sur l'utilisation des périphériques connectés au bus CAN, référez-vous à l'article sur l'utilisation des *dispositifs CAN*.

39

Premiers pas avec Romi

The Romi is a small and inexpensive robot designed for learning about programming FRC robots. All the same tools used for programming full-sized FRC robots can be used to program the Romi. The Romi comes with two drive motors with integrated wheel encoders. It also includes an *IMU* sensor that can be used for measuring headings and accelerations. Using it is as simple as writing a robot program, and running it on your computer. It will command the Romi to follow the steps in the program.

Astuce : Un cours d'apprentissage de la programmation à l'aide du robot Romi est disponible via Thinkscape. L'information sur le cours est disponible ici

---

39.1 Matériel (Hardware) et assemblage de Romi

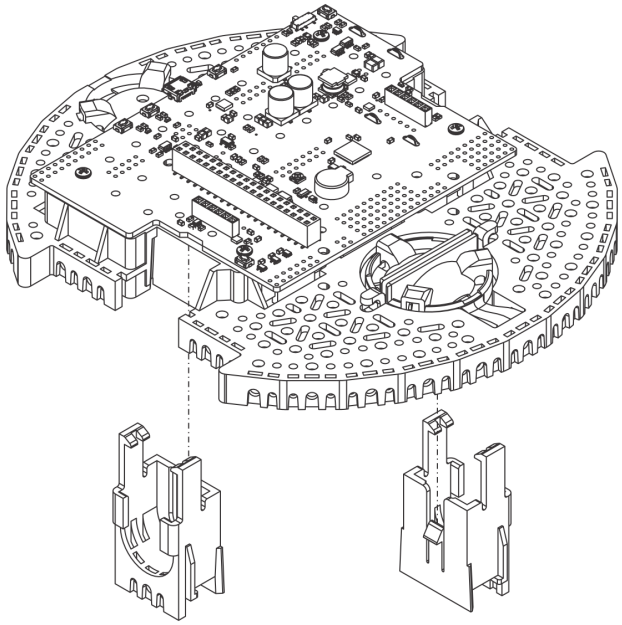
Pour démarrer avec le Romi, vous aurez besoin du matériel nécessaire.

1. Romi Kit vendu par Pololu – La commande bénéficie d'une livraison gratuite
2. Raspberry Pi – 3B+ ou 4
3. Carte Micro SD de 8GB (ou plus)
4. Lecteur de carte Micro SD - si vous n'en avez pas déjà un
5. 6 piles AA - Les piles rechargeables sont recommendées (n'oubliez pas le chargeur)

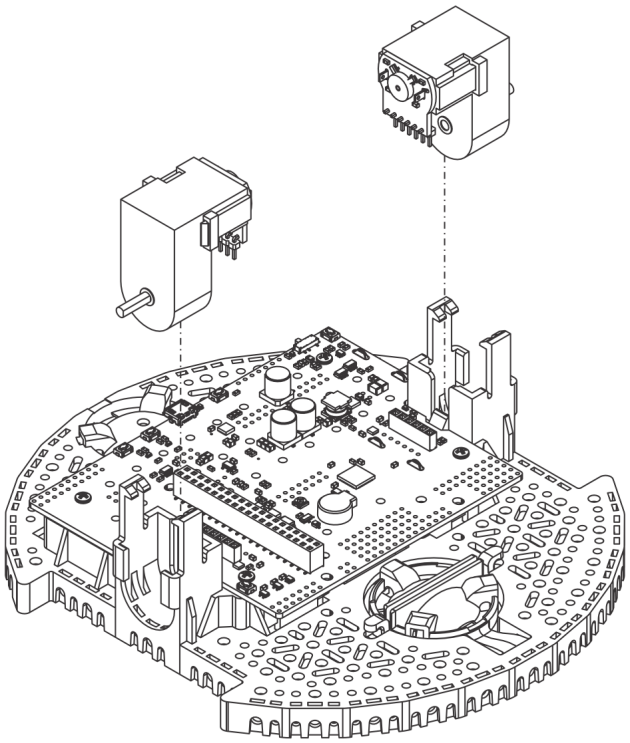
39.1.1 Assemblage

Le kit de robot Romi pour FIRST est pré-soudé et doit être assemblé avant de pouvoir être utilisé. Une fois que vous avez rassemblé tous les matériaux, vous pouvez commencer l'assemblage :

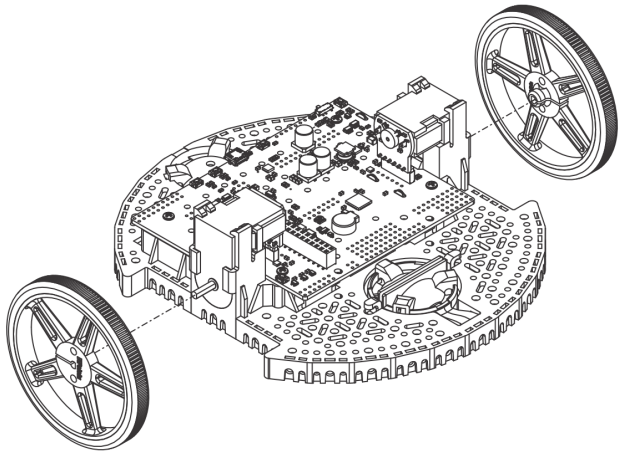
1. Alignez les clips du moteur avec le châssis comme indiqué et appuyez-les fermement dans le châssis jusqu'à ce que le bas des clips soit au même niveau que le bas du châssis (vous pouvez entendre plusieurs clics).



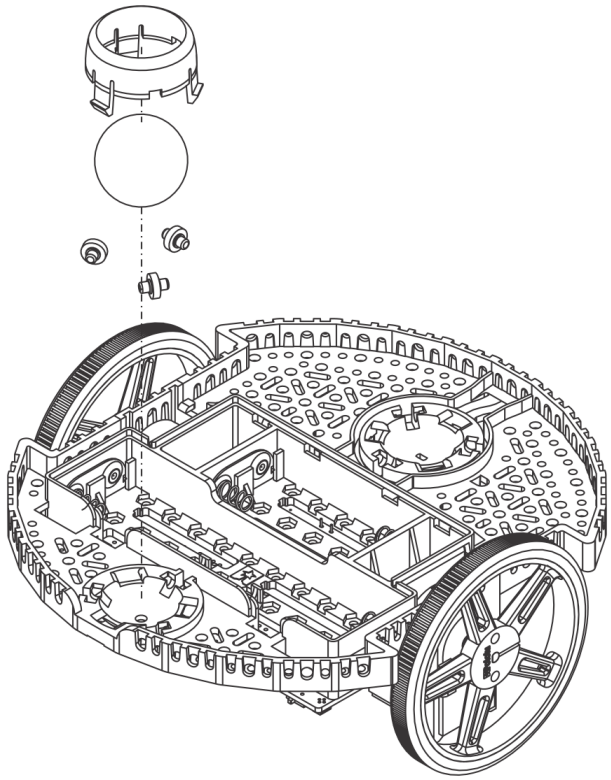
2. Poussez les mini moteurs en plastique dans les clips du moteur jusqu'à ce qu'ils s'enclenchent. Notez que le moteur bloque la libération du clip, donc si vous devez retirer un support de moteur plus tard, vous devrez d'abord retirer le moteur. Les mini moteurs en plastique fournis avec le kit ont des arbres étendus pour permettre la liason mécanique aux encodeurs en quadrature pour le contrôle de position.



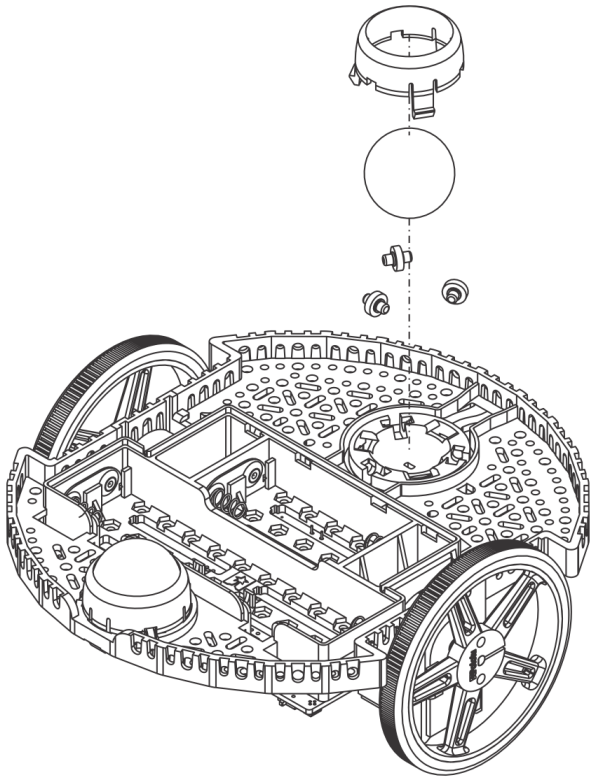
3. Insérer les roues sur les arbres de sortie des moteurs jusqu'à ce que l'arbre du moteur affleure la face extérieure de la roue. Une façon de procéder consiste à placer la roue sur une surface plane et à aligner le châssis avec elle de sorte que la partie plate de l'arbre en D du moteur s'aligne correctement avec la roue. Ensuite, abaissez le châssis en enfonçant l'arbre du moteur dans la roue jusqu'à ce qu'il entre en contact avec la surface.



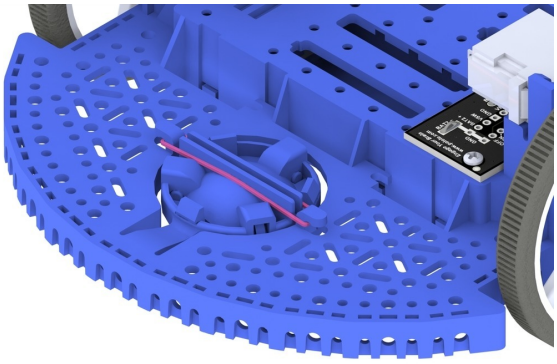
4. Retournez le châssis et placez les trois rouleaux arrière dans les découpes du châssis. Placez la boule en plastique de 1″ au-dessus des trois rouleaux. Poussez ensuite le clip de retenue de la roulette sur la bille et dans le châssis pour que les trois pieds s'enclenchent dans leurs trous respectifs.



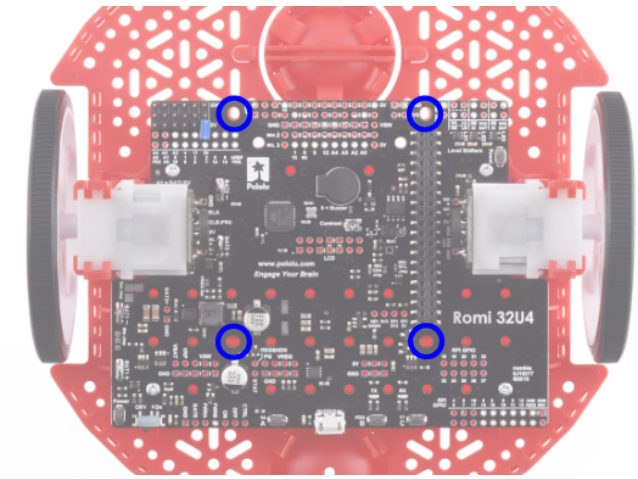
5. Répétez l'opération pour les rouleaux avant afin qu'il y ait un rouleau à l'avant et à l'arrière du robot.



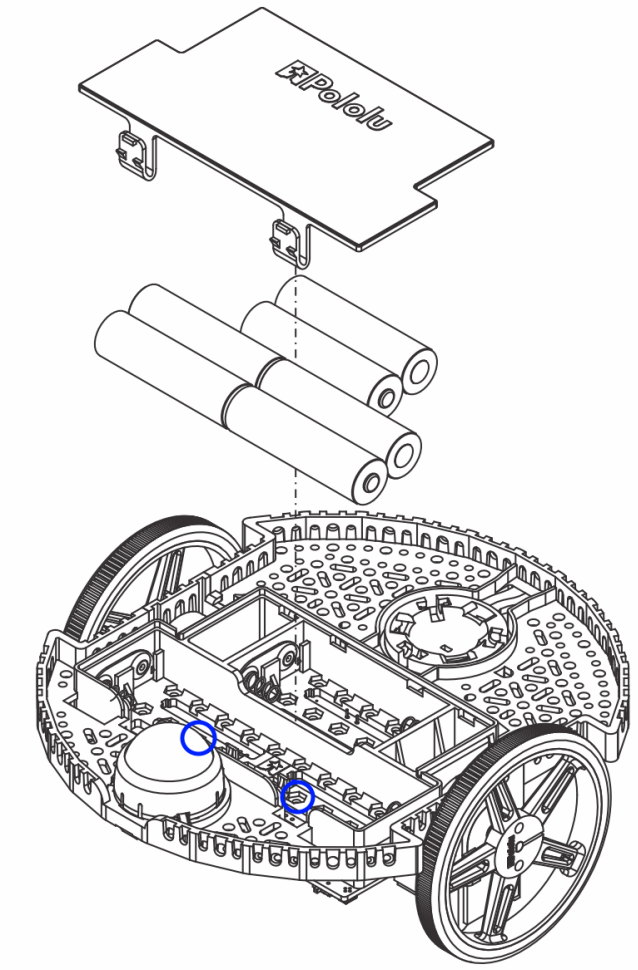
6. En option : le rouleau à bille avant est soutenue par un bras flexible qui agit comme un système de suspension. Si vous souhaitez le rendre plus rigide, vous pouvez enrouler une bande de caoutchouc autour des deux crochets situés de chaque côté de la roulette à billes sur le côté supérieur du châssis.



7. Installez les socles pour soutenir la carte Raspberry Pi. Deux socles (côté fil vers le bas) montent dans les trous sur le côté de la carte du Romi la plus proche de l'étiquette « Romi 32U4 » comme le montre l'image. Les écrous de ces socles sont à l'intérieur du compartiment de la batterie. Les deux autres socles s'insèrent dans les trous de l'autre côté de la carte. Pour les attacher, vous aurez besoin d'une pince à bec effilé pour tenir l'écrou pendant que vous vissez dans les socles. Les trous encerclés de l'image ci-dessous montrent où les socles doivent s'insérer.



8. Le châssis fonctionne avec quatre ou six piles AA (nous vous recommandons d'utiliser des piles AA NiMH rechargeables). L'orientation correcte des batteries est indiquée par les trous en forme de batterie dans le châssis Romi ainsi que par les voyants + et - dans le châssis lui-même.



9. Fixez la carte Raspberry Pi à l'envers, en alignant soigneusement le connecteur 2x20 broches sur le Pi avec la prise 2x20 broches sur le Romi. Poussez avec une pression uniforme en prenant soin de ne plier aucune des broches. Une fois insérée, utilisez les

vis fournies pour fixer la carte Raspberry Pi aux entretoises installées lors de l'étape précédente.

Note : Deux des vis nécessiteront de placer un écrou dans un trou hexagonal à l'intérieur du compartiment de la batterie. Les emplacements sont indiqués par les cercles bleus dans l'image ci-dessus.

L'assemblage de votre châssis Romi est maintenant terminé !

39.2 Installation de l'image dans votre Romi

Le Romi dispose de 2 cartes à microprocesseur :

1. A **Raspberry Pi** that handles high-level communication with the robot program running on the desktop and
2. A **Romi 32U4 Control Board** that handles low-level motor and sensor operation.

Both boards need to have firmware installed so that the robot operates properly.

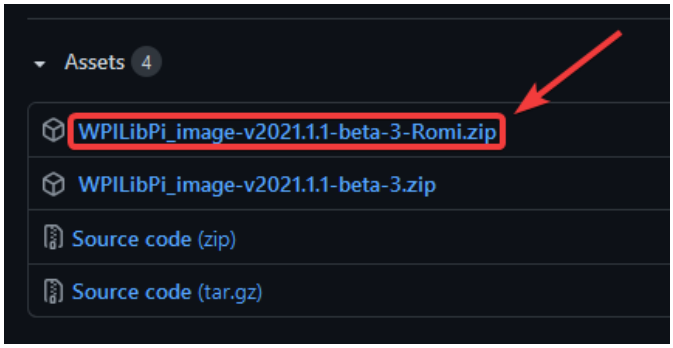
39.2.1 Raspberry Pi

Téléchargement

Le firmware du Raspberry Pi est basé sur WPILibPi (anciennement FRCVision) et doit être téléchargé et écrit sur la carte Raspberry Pi micro SD. Cliquez sur `Assets` au bas de la description pour voir les fichiers images disponibles :

Romi WPILibPi

Assurez-vous de télécharger la version Romi et non la version standard de WPILibPi. La version Romi est pourvue du suffixe `-Romi`. Voir l'image ci-dessous pour un exemple.



Installation de l'image

La procédure d'installation de l'image est décrite ici : *WPILibPi Installation*.

Configuration du réseau sans fil

Réalisez les étapes suivantes pour que votre Raspberry Pi soit prêt à être utilisé avec le Romi :

1. Allumez le Romi en glissant l'interrupteur d'alimentation sur la carte Romi 32U4 jusqu'à la position on. La première fois qu'il est mis sous tension avec une nouvelle image, il faudra environ 2-3 minutes pour démarrer pendant qu'il redimensionne le système de fichiers et redémarre. Les démarrages suivants se feront en moins d'une minute.
2. À l'aide de votre ordinateur, connectez-vous au réseau WiFi Romi à l'aide du SSID `WPILibPi-<number>` (où `<number>` est basé sur le numéro de série du Raspberry Pi) avec le mot de passe WPA2 `WPILib2021!`.

Note : Si le Raspberry Pi est allumé dans un environnement avec plusieurs Raspberry Pis pourvus de WPILibPI en cours d'exécution, le SSID pour un Raspberry Pi particulier est également annoncé de manière audible à travers le port du casque. Le SSID par défaut est également écrit sur le fichier `/boot/default-ssid.txt` qui peut être lu en insérant la carte SD (via un lecteur) dans un ordinateur et en ouvrant la partition `boot`.

---

3. Ouvrez un navigateur Web et connectez-vous au dashboard du Raspberry Pi soit à l'adresse `http://10.0.0.2/` ou à l'adresse `http://wpilibpi.local/`.

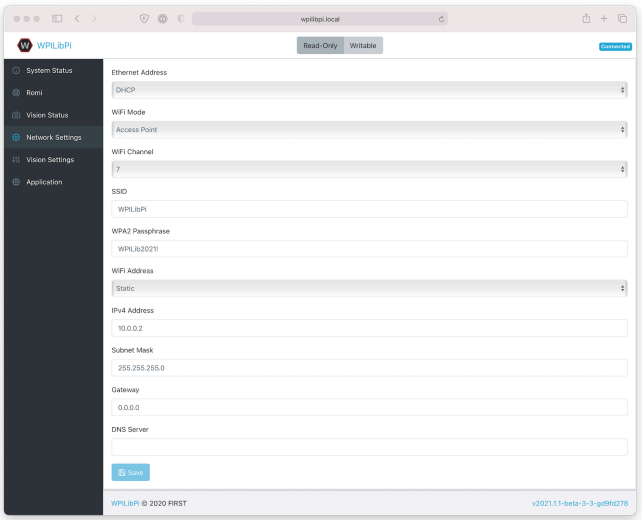
Note : L'image démarre en lecture seule par défaut, il est donc nécessaire de cliquer sur le bouton `Writable` pour apporter des modifications. Une fois les modifications effectuées, cliquez sur le bouton `Read-Only` pour éviter la corruption de la mémoire.

---

4. Sélectionnez *Writable* en haut de la page Web du dashboard.
5. Changez le mot de passe par défaut de votre Romi en inscrivant un nouveau mot de passe dans le champ `WPA2 Passphrase`.
6. Appuyez sur le bouton *Save* en bas de la page pour enregistrer les modifications.
7. Change the network SSID to a unique name if you plan on operating your Romi on a wireless network with other Romis.

8. Reconnectez-vous au réseau WiFi de Romi avec le nouveau mot de passe que vous avez défini.

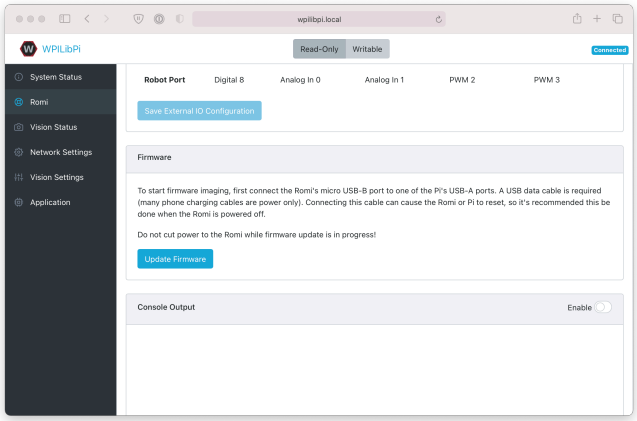
Assurez-vous de configurer le Dashboard à `Read-only` lorsque toutes les modifications auront été effectuées.



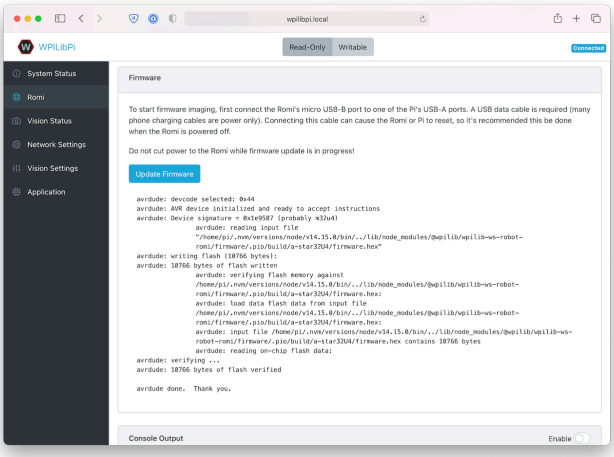
39.2.2 Carte de contrôle 32U4

Le Raspberry Pi peut maintenant être utilisé pour installer l'image du firmware à la carte de contrôle 32U4

1. Éteignez le Romi
2. Connectez un câble USB A à micro-B de l'un des ports USB du Raspberry Pi aux ports micro USB de la carte de contrôle 32U4.
3. Allumez le Romi et connectez-vous à son réseau Wifi et connectez-vous au dashboard web comme lors des étapes précédentes.
4. Sur la page de configuration de `Romi` appuyez sur le bouton *Update Firmware*.



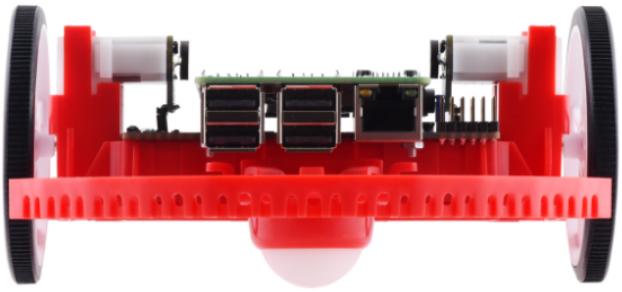
Une console apparaîtra affichant un log du processus de déploiement du micrologiciel. Une fois le micrologiciel déployé sur la carte de contrôle 32U4, le message `avrdude done. Thank you.` apparaîtra.



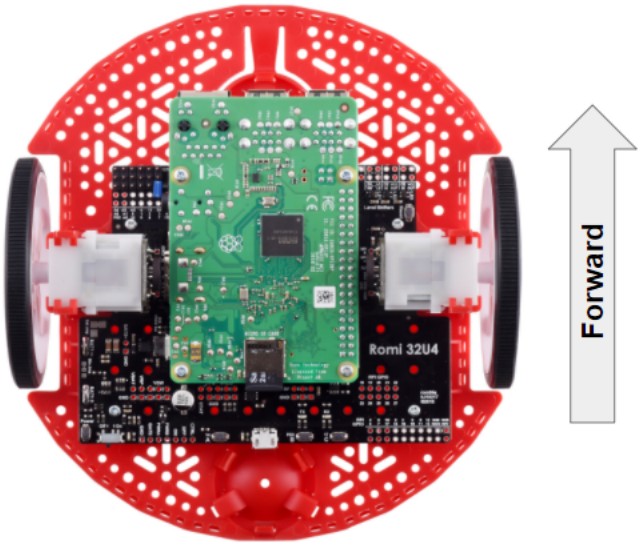
39.3 Apprendre à connaître votre Romi

39.3.1 Conventions directionnelles

L'avant de Romi est correspond à l'endroit où se trouvent les ports USB du Raspberry Pi, les ports GPIO et la roue de roulette suspendue.



Dans toute la documentation de Romi, les références à la conduite vers l'avant utilisent la définition ci-dessus de « l'avant ».



39.3.2 Matériel, capteurs et GPIO

Le Romi possède le matériel et les périphériques intégrés suivants :
- 2x motoréducteurs avec encodeurs
- 1x unité de mesure inertielle (IMU)
- 3x LED (verte, jaune, rouge)
- 3 boutons poussoirs (marqués A, B et C)
- 5x canaux GPIO configurables (EXT)
- Buzer ou vibreur

Note : Le Buzzer ou vibreur n'est pas actuellement pris en charge par WPILib.

Moteurs, roues et encodeurs

Les moteurs utilisés sur le Romi ont une réduction de 120 :1 et une vitesse de sortie à vide de 150 RPM à 4.5V. Le courant libre est de 0.13 ampères et le courant de décrochage est de 1.25 ampères. Le couple de décrochage est de 25 oz-in (0.1765 N-m) mais l'embrayage de sécurité intégré peut commencer à glisser à des couples inférieurs.

Les roues ont un diamètre de 70mm (2.751″). Ils ont une voie (trackwidth) de 141mm (5.551″).

Les encodeurs sont connectés directement à l'arbre de sortie du moteur et ont 12 comptes par révolution (CPR). Avec le rapport de démultiplication fourni, ceci correspond à 1440 comptes par rotation de roue.

The motor *PWM* channels are listed in the table below.

Canal	Composant matériel Romi
PWM 0	Moteur gauche
PWM 1	Moteur droit

Note : Le moteur droit tournera dans une direction inverse lorsque la sortie positive est appliquée. Ainsi, le code relatif au contrôleur de moteur correspondant doit être inversé dans le code du robot.

Les canaux relatifs aux encodeurs sont répertoriés dans le tableau ci-dessous.

Canal	Composant matériel Romi
DIO 4	Canal de quadrature encodeur gauche A
DIO 5	Canal de quadrature encodeur gauche B
DIO 6	Canal de quadrature encodeur droit A
DIO 7	Canal de quadrature encodeur droit B

Note : Par défaut, la lecture de valeurs croissantes des encodeurs se produit lorsque le Romi avance.

L'unité de mesure inertielle

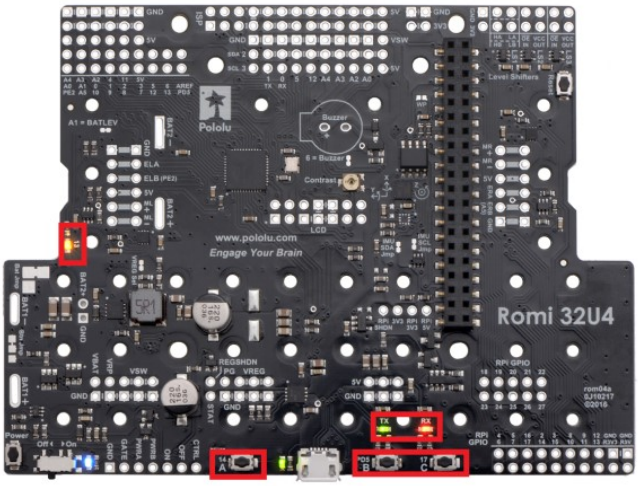
Le Romi comprend une unité de mesure inertielle (IMU) STMicroelectronics LSM6DS33 qui contient un gyroscope à 3 axes et un accéléromètre à 3 axes.

L'accéléromètre a une sensibilité sélectionnable de 2G, 4G, 8G et 16G. Le gyroscope a une sensibilité sélectionnable de 125 degrés par seconde (DPS), 250 DPS, 500 DPS, 1000 DPS et 2000 DPS.

L'interface utilisateur Web Romi fournit également un moyen de calibrer le gyroscope et de mesurer ses décalages d'origine avant de l'utiliser avec le code du robot.

LEDs et boutons-poussoirs intégrés

La carte de contrôle Romi 32U4 dispose de 3 boutons-poussoirs et de 3 LEDs embarqués qui sont exposés en tant que canaux d'E/S numériques (DIO) au code du robot.

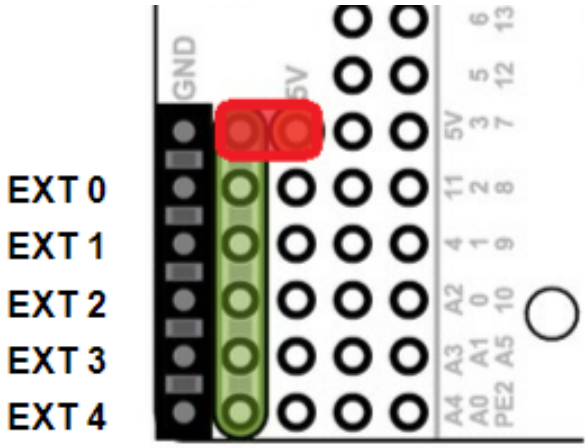


Canal DIO	Composant matériel Romi
DIO 0	Bouton A (entrée uniquement)
DIO 1	Bouton B (entrée), LED verte (sortie)
DIO 2	Bouton C (entrée), LED rouge (sortie)
DIO 3	LED jaune (sortie uniquement)

Writes to DIO 0, 4, 5, 6 and 7 will result in a *no-op*.

Broches GPIO configurables

La carte de contrôle dispose de 5 broches GPIO configurables (désignées EXT0 à EXT4) qui permettent à un utilisateur de connecter des capteurs et des actionneurs externes au Romi.



Les 5 broches prennent en charge les modes suivants : E/S numériques, Entrée analogique et PWM (à l'exception de EXT 0, qui ne prend en charge que les E/S numériques et PWM). Le mode des ports peut être configuré avec *l'interface utilisateur Web Romi*.

Les canaux GPIO sont exposés via une interface de style servo à 3 broches, avec des connexions pour la masse (ground), l'alimentation et le signal (la connexion à la masse étant la plus proche du bord de la carte et le signal le plus proche de l'intérieur de la carte).

Les connexions d'alimentation pour les broches GPIO sont initialement laissées sans connexion, mais peuvent être connectées à l'alimentation 5V intégrée du Romi à l'aide d'un cavalier pour connecter la broche 5V au bus d'alimentation (comme on le voit dans l'image ci-dessus). De plus, si plus si nous avons besoin de plus d'énergie que ce le Romi peut fournir, l'utilisateur peut fournir sa propre alimentation 5V et la connecter directement au bus d'alimentation et aux broches de mise à la terre.

Configuration par défaut de broches GPIO

Le tableau ci-dessous montre la configuration par défaut des broches GPIO (EXT0 à EXT4). *L'interface utilisateur Web Romi* permet à l'utilisateur de personnaliser les fonctions des 5 broches GPIO configurables. L'interface utilisateur fournira également à l'écran les correspondances de canaux/périphériques WPILib appropriés une fois la configuration des E/S terminée.

Canal	Ext Pin
DIO 8	EXT0
Analog In 0	EXT1
Analog In 1	EXT2
PWM 2	EXT3
PWM 3	EXT4

39.4 Support matériel Romi

Le robot Romi, ayant une architecture matérielle différente de celle d'un roboRIO, est compatible avec un sous-ensemble de composants du système de contrôle FRC couramment utilisés.

39.4.1 Matériel compatible

En général, le Romi est compatible avec les éléments suivants :

- Dispositifs d'entrée/sortie numériques simples (par exemple, commutateurs, LED)
- Dispositifs de sortie standard de style RC *PWM* (par exemple, servos, contrôleurs de moteur basés sur PWM)
- Capteurs d'entrée analogique (par exemple, capteurs de distance qui signalent la distance sous forme de tension)

39.4.2 Matériel incompatible

En raison de limitations matérielles, le robot Romi n'est pas compatible avec les éléments suivants :

- Encodeurs autres que les encodeurs intégrés Romi
- Capteurs à ultrasons de style « Ping » (qui nécessitent 2 canaux DIO)
- Capteurs basés sur le temps
- Appareils basés sur CAN
- Buzzer intégré Romi

39.4.3 Classes compatibles

Toutes les classes répertoriées ici sont prises en charge par le robot Romi. Si une classe n'est pas répertoriée ici, supposez qu'elle n'est pas prise en charge et qu'elle *ne fonctionnera pas*.

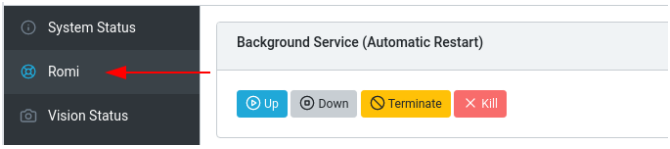
- Contrôleurs de moteurs PWM (i.e. `Spark`)
- `Encoder`
- `AnalogInput`
- `DigitalInput`
- `DigitalOutput`
- `Servo`
- `BuiltInAccelerometer`

Les classes suivantes sont fournies par Romi Vendordep, c'est à dire le manufacturier du Romi.

- `RomiGyro`
- `RomiMotor`
- `OnboardIO`

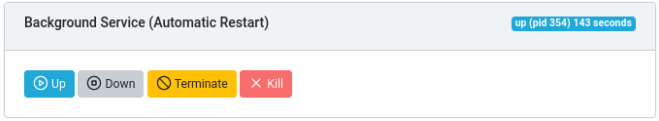
39.5 Interface utilisateur Web de Romi

L'interface utilisateur web Romi est installée comme partie intégrante de l'image WPILibPi Raspberry Pi. Elle est accessible en cliquant sur l'onglet Romi dans la barre de navigation de l'interface utilisateur web WPILibPi principale.



Le reste de cette section passera en revue les différentes parties de l'interface utilisateur Web Romi et décrira les fonctionnalités pertinentes.

39.5.1 État du service Web en arrière-plan



Cette section de l'interface utilisateur Web Romi fournit des informations sur le service Web Romi en cours d'exécution (ce qui permet à WPILib de parler au Romi). L'interface utilisateur fournit des contrôles pour activer/désactiver le service et affiche le temps de fonctionnement actuel du service Web.

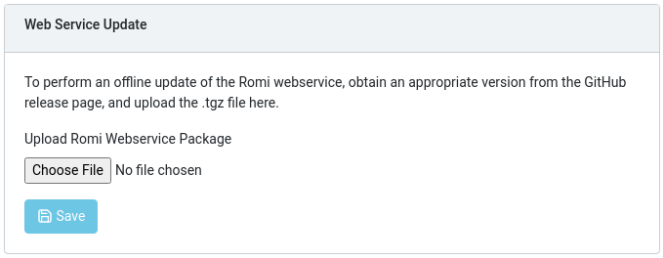
Note : Les utilisateurs n'auront pas besoin d'utiliser souvent la fonctionnalité de cette section, mais cela peut être utile pour le dépannage.

39.5.2 Statut du Romi

Cette section fournit des informations sur le Romi, y compris la version du service, la tension de la batterie et si le micrologiciel actuellement installé sur la carte Romi 32U4 est compatible ou non avec la version actuelle du service Web.

Note : Si le micrologiciel n'est pas compatible, voir la section *Installation de l'image dans votre Romi*

39.5.3 Mise à jour du service Web

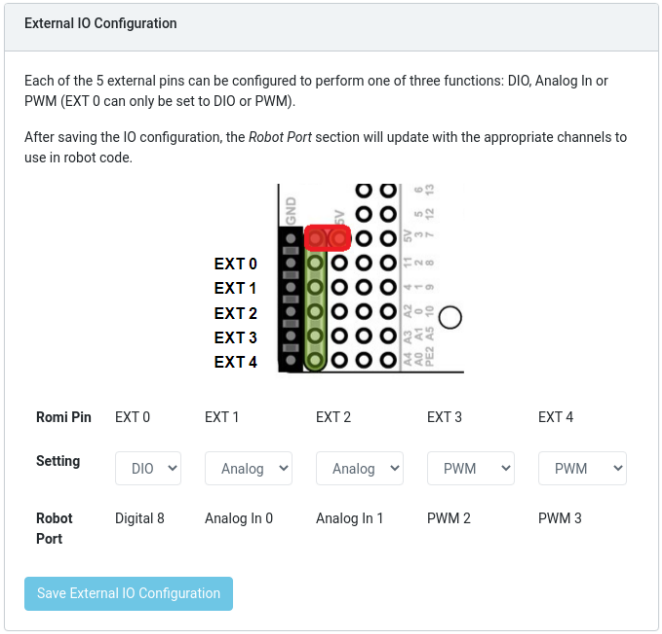


Note : Le Raspberry Pi doit être en mode **Writable** pour que cette section fonctionne.

L'image Romi WPILibPi est fournie avec la dernière version (au moment de la publication) du service Web Romi. Pour prendre en charge la mise à niveau vers des versions plus récentes du service Web Romi, cette section permet aux utilisateurs de télécharger un bundle pré-construit qui peut être obtenu via le service Web Romi Page des versions de GitHub.

Pour effectuer une mise à niveau, téléchargez le fichier .tgz approprié à partir de la page Versions de GitHub. Ensuite, sélectionnez le fichier .tgz téléchargé et cliquez sur *Save*. Le kit de services Web mis à jour sera téléchargé sur le Raspberry Pi et installé. Après un court instant, la section Statut Romi devrait se mettre à jour avec les dernières informations de version.

39.5.4 Configuration I/O (Éntrées-Sorties) externe



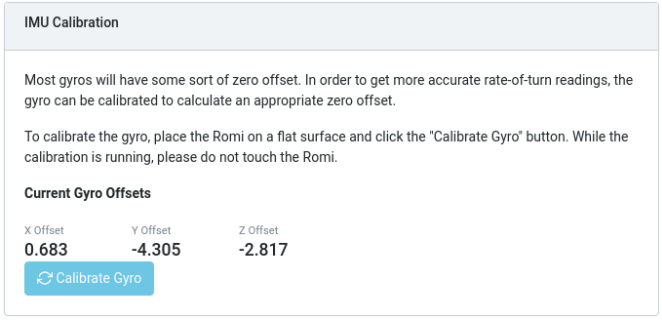
Cette section permet aux utilisateurs de configurer les 5 canaux GPIO externes sur le Romi.

Note : Le Raspberry Pi doit être en mode **Writable** pour que cette section fonctionne.

To change the configuration of a GPIO channel, select an appropriate option from the drop-down lists. All channels (with the exception of EXT 0) support Digital IO, Analog In and *PWM* as channel types. Once the appropriate selections are made, click on *Save External IO Configuration*. The web service will then restart and pick up the new IO configuration.

La ligne «Robot Port» fournit le mappage WPILib approprié pour chaque canal GPIO configuré. Par exemple, EXT 0 est configuré comme canal I/O numérique et sera accessible dans WPILib en tant que canal 8 DigitalInput (ou DigitalOutput).

39.5.5 Étalonnage de l'unité IMU



Note : Le Raspberry Pi doit être en mode **Writable** pour que cette section fonctionne.

Cette section permet aux utilisateurs de calibrer le gyroscope sur le Romi. Les gyroscopes ont généralement une sorte d'erreur de décalage zéro, et l'étalonnage permet au Romi de calculer le décalage et de l'utiliser dans les calculs.

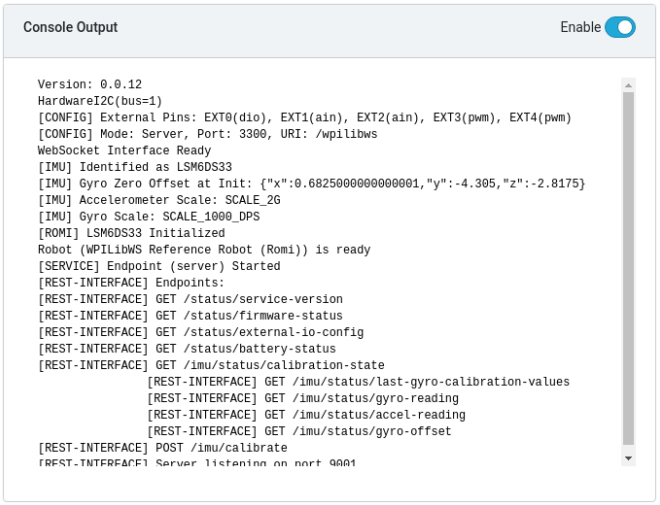
Pour commencer l'étalonnage, placez le Romi sur une surface plane et stable. Ensuite, cliquez sur le bouton *Calibrate Gyro*. Une barre de progression apparaîtra, indiquant le processus d'étalonnage en cours. Une fois l'étalonnage terminé, les dernières valeurs de décalage seront affichées à l'écran et enregistrées auprès du service Web Romi.

Ces valeurs de décalage sont enregistrées sur la carte SD et persistent entre les redémarrages.

39.5.6 Micrologiciel

Note : Voir la section sur *Installation de l'image dans votre Romi*

39.5.7 Sortie de la console



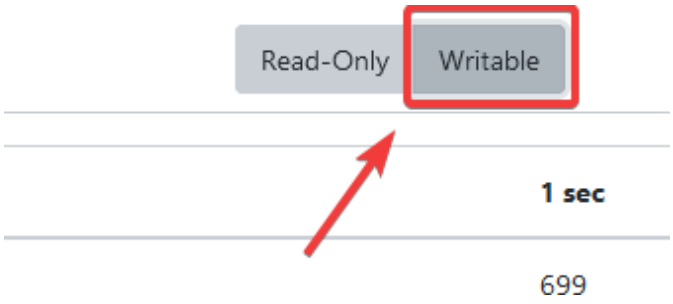
Lorsqu'elle est activée, cette section permet aux utilisateurs d'afficher la sortie de console brute fournie par le service Web Romi. Ceci est utile pour résoudre les problèmes avec le Romi, ou simplement pour en savoir plus sur ce qui se passe dans les coulisses.

39.5.8 Mode passerelle

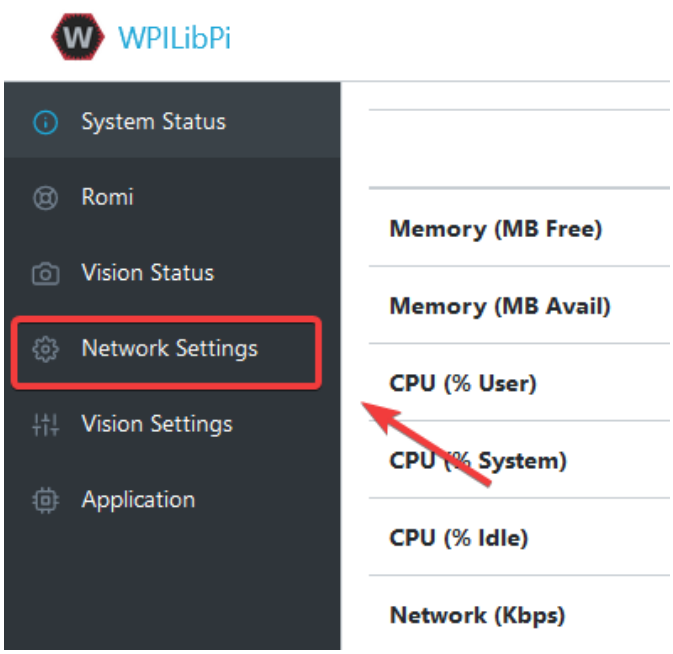
Le mode Bridge ou passerelle permet à votre robot Romi de se connecter à un réseau WiFi au lieu d'agir comme un point d'accès (AP). Ceci est particulièrement utile dans les environnements d'apprentissage à distance, car vous pouvez utiliser Internet tout en utilisant le Romi sans matériel supplémentaire.

Note : Il est peu probable que le mode passerelle fonctionne correctement dans les environnements de réseau restreints (institutions d'enseignement).

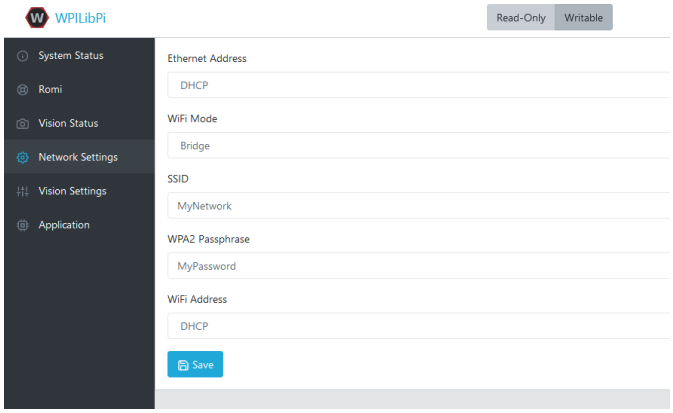
1. Activer *Writable* dans le menu du haut.



2. Cliquez sur *Network Settings*.



3. Les configurations réseau suivantes doivent être appliquées :



— **Ethernet** : DHCP
— **WiFi Mode** : Bridge
— **SSID** : SSID (name) nom de votre réseau
— **WPA2 Passphrase** : Mot de passe de votre réseau wifi
— **WiFi Address** : DHCP

Une fois que les configurations sont appliquées, s'il vous plaît redémarrer le Romi. Vous devriez maintenant être en mesure de naviguer vers `wpilibpi.local` dans votre navigateur Web lorsque vous êtes connecté à votre réseau spécifié.

Impossible d'accéder à Romi

Si le Romi présente des configurations exactes de ses paramètres de passerelle et que vous ne pouvez pas y accéder, nous avons quelques solutions de contournement.

— Connectez-vous à Romi par câble Ethernet
— Réinstallez l'image du Romi

Certains réseaux restreints peuvent interférer avec le nom d'hôte de la résolution Romi, vous pouvez résoudre ce problème en utilisant Angry IP Scanner pour trouver l'adresse IP.

Avertissement : Angry IP Scanner est signalé par certains antivirus comme spyware comme il envoie des requêtes ping à des dispositifs sur votre réseau ! C'est une application sûre !

39.6 Programmation de Romi

L'écriture d'un programme pour le Romi est très similaire à l'écriture d'un programme pour un robot FRC ordinaire. En fait, tous les mêmes outils (Visual Studio Code, Driver Station, SmartDashboard, etc.) peuvent être utilisés avec le Romi.

39.6.1 Créer un programme Romi

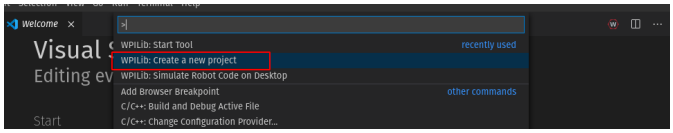
Créer un nouveau programme pour un Romi, c'est comme créer un programme FRC normal, similaire aux étapes de programmation repertoriées dans la section :doc :` de Zéro à Robot </docs/zero-to-robot/step-4/index>`.

WPILib est livré avec deux modèles pour les projets Romi, dont un basé sur TimedRobot et un modèle de projet orienté commandes. En outre, un exemple d'un projet est fourni qui présente certaines des fonctionnalités intégrées du Romi. Cet article décrit la création d'un projet à partir de cet exemple.

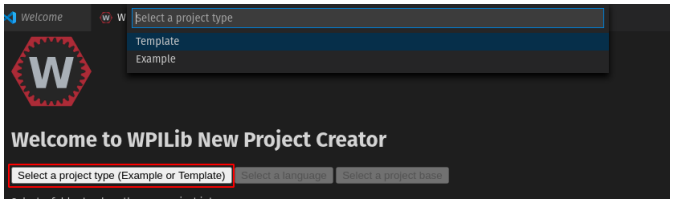
Note : In order to program the Romi using C++, a compatible C++ desktop compiler must be installed. See *Robot Simulation - Additional C++ Dependency*.

Création d'un nouveau projet WPILib Romi

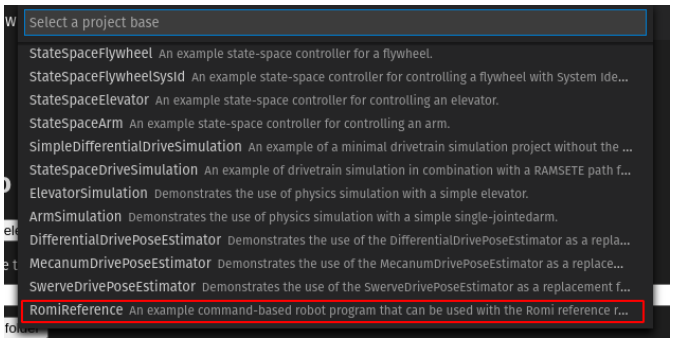
Affichez la palette de commandes Visual Studio Code avec `Ctrl+Maj+P`, et tapez « New project ». Sélectionnez la commande « Create a new project » :



Cela fera apparaître la « New Project Creator Window ». De là, cliquez sur « Select a project type (Example or Template), et choisir « Example » dans la boîte qui apparaît :



Ensuite, une liste d'exemples apparaîtra. Faites défiler la liste pour trouver l'exemple « RomiReference » :



Remplissez le reste des champs dans le « New Project Creator » et cliquez sur « Generate Project » pour créer le nouveau projet de robot.

Exécution d'un programme Romi

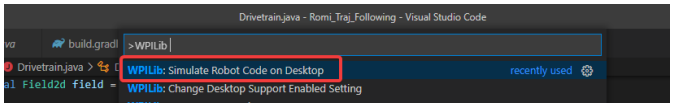
Une fois le projet de robot généré, il est essentiellement prêt à fonctionner. Le projet dispose d'une classe `Drivetrain` fournie avec l'installation logicielle et d'une commande par défaut associée qui vous permet de conduire le Romi à l'aide d'un joystick.

Un aspect où un projet Romi diffère d'un projet de robot FRC ordinaire est que le code n'est pas déployé directement sur le Romi. À la place, un projet Romi s'exécute sur votre ordinateur de développement et exploite l'environnement de simulation WPILib pour communiquer avec le robot Romi.

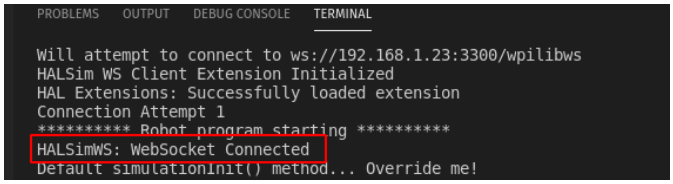
Pour exécuter un programme Romi, assurez-vous d'abord que votre Romi est sous tension. Ensuite, connectez-vous au réseau WiFi `WPILibPi-` diffusé par le Romi. Si vous avez modifié les paramètres du réseau Romi (par exemple, pour le connecter à votre propre réseau WiFi), vous pouvez modifier l'adresse IP que votre programme utilise pour se connecter au Romi. Pour ce faire, ouvrez le fichier `build.gradle` et mettez à jour la ligne `wpi.sim.envVar` avec l'adresse IP appropriée.

```
//Sets the websocket client remote host.
wpi.sim.envVar("HALSIMWS_HOST", "10.0.0.2")
wpi.sim.addWebsocketsServer().defaultEnabled = true
wpi.sim.addWebsocketsClient().defaultEnabled = true
```

Maintenant, pour démarrer votre code de robot Romi, ouvrez la palette de commandes WPILib (tapez `Ctrl+Shift+P`) et sélectionnez « Simulate Robot Code », ou appuyez sur `F5`.



Si tout se passe bien, vous devriez voir une ligne dans la sortie de la console qui lit « HALSimWS : WebSocket Connected » :



Votre code Romi est maintenant en cours d'exécution !

39.7 Programmation du Romi (LabVIEW)

L'écriture d'un programme LabVIEW pour le Romi est très similaire à l'écriture d'un programme pour un robot standard basé sur roboRIO. En fait, tous les mêmes outils peuvent être utilisés avec le Romi.

39.7.1 Créer un projet Romi

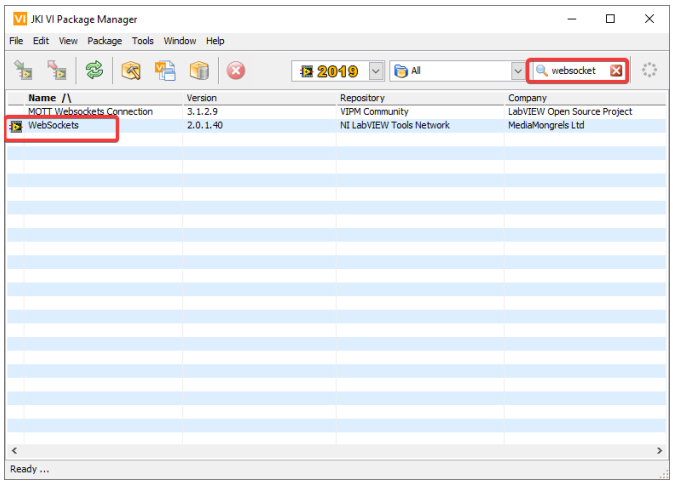
Créer un nouveau programme pour un Romi n'est pas différent de créer un FRC normal |reg| similaire aux étapes de programmation Zero To Robot. Au départ, vous souhaiterez peut-être créer un projet distinct à utiliser uniquement sur le Romi, car le matériel Romi peut être connecté à des ports différents de ceux de votre robot roboRIO.

The Romi Robot used *PWM* ports 0 and 1 for left and right side respectively.

Installer le WebSockets VI

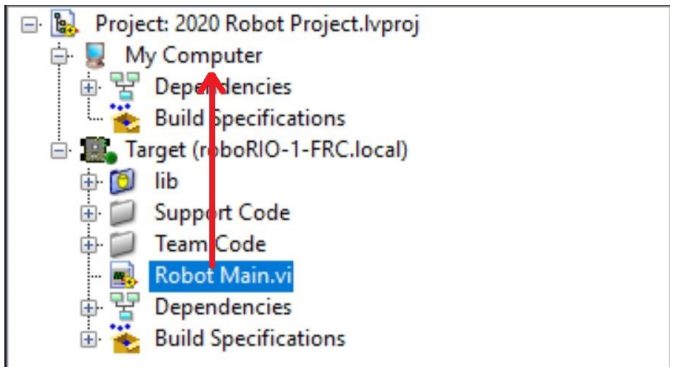
Un aspect où un projet Romi diffère d'un FRC régulier |reg| robot est que le code n'est pas déployé directement sur le Romi. Au lieu de cela, un projet Romi s'exécute sur votre ordinateur de développement et exploite le cadre de simulation WPILib pour communiquer avec le robot Romi. WebSockets est le protocole que LabVIEW utilise pour converser avec le Romi.

Ouvrez l'application *VI Package Manager*. Tapez `websockets` dans le champ de recherche en haut à droite. Sélectionnez le VI par *LabVIEW Tools Network*



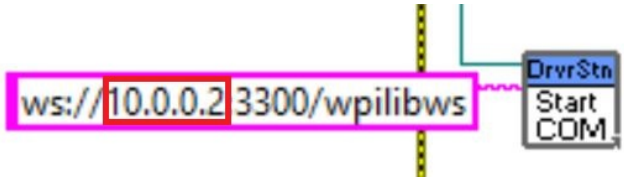
Modification de la cible du projet

La première étape nécessaire pour exécuter votre programme LabVIEW sur le Romi est de changer la cible en Desktop. Pour changer la cible du projet, localisez le Robot Main VI dans l'explorateur de projet et cliquez dessus et faites-le glisser de la section `Cible` vers la section `Poste de travail`.



Définition de l'adresse IP cible

Par défaut, votre programme LabVIEW tentera de se connecter à un Romi avec l'adresse IP `10.0.0.2`. Si vous souhaitez utiliser une adresse IP différente, vous pouvez la spécifier comme entrée du `Driver Station Start Communication VI` dans `Robot Main`. Localisez le terminal d'entrée rose pour `Simulation URL` puis faites un clic droit et sélectionnez *Create Constant* pour créer une constante pré-remplie avec la valeur par défaut. Vous pouvez ensuite modifier la partie adresse IP du texte, en prenant soin de laisser la section protocole (au début) et le port et le suffixe (à la fin) identiques.



Exécution d'un programme Romi

Pour exécuter un programme Romi, assurez-vous d'abord que votre Romi est sous tension. Une fois connecté au réseau `WPILibPi-` diffusé par le Romi, appuyez sur la flèche blanche *Run* pour lancer le programme Romi sur votre ordinateur.

Votre code Romi est maintenant en cours d'exécution ! Le programme tentera automatiquement de se connecter soit à l'adresse IP que vous avez spécifiée, soit à la valeur par défaut si vous n'avez pas spécifié d'adresse IP.

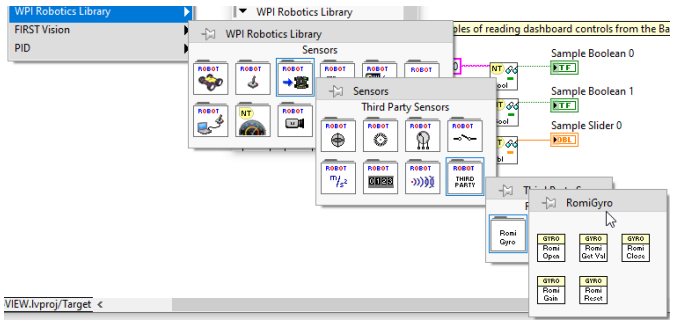
Il est recommandé d'exécuter le logiciel Driver Station sur le même ordinateur que le code LabVIEW. Une fois que votre programme se connecte avec succès à la Driver Station, il notifiera automatiquement à la Driver Station que le code est en cours d'exécution sur le bureau, permettant à la Driver Station de se connecter sans que vous ne modifiiez aucune information à l'intérieur de la Driver Station. Ensuite, vous devrez pointer la Driver Station vers votre Romi. Cela se fait en définissant le numéro d'équipe sur `127.0.0.1`. Vous pouvez ensuite utiliser les commandes de la Driver Station pour définir le mode robot et activer/désactiver normalement.

Note : If your robot code is unable to connect to the Romi, the Driver Station will also show no connectivity.

Utilisation du gyroscope ou de l'encodeur

Le gyroscope disponible sur le Romi est disponible à l'aide des fonctions RomiGyro. Celui-ci est situé sous

```
- WPI Robotics Library
   - Sensors
      - Third Party Libraries
         - RomiGyro
```



Les encodeurs peuvent être utilisés en utilisant la fonction d'encodeur standard. Les ports DIO sont :

— Left (4, 5)
— Right (6, 7)

40

Getting Started with XRP

The XRP is a small and inexpensive robot designed for learning about programming FRC robots. All the same tools used for programming full-sized FRC robots can be used to program the XRP. The XRP comes with two drive motors with integrated wheel encoders. It also includes an *IMU* sensor that can be used for measuring headings and accelerations. Using it is as simple as writing a robot program, and running it on your computer. It will command the XRP to follow the steps in the program.

The XRP provides a similar use case as the *Romi* with similar functionality, albeit using a lower power processor and is overall lower in cost.

40.1 Matériel, assemblage et imagerie XRP

Pour démarrer avec le XRP, vous aurez besoin du matériel suivant.

1. Kit XRP de SparkFun ou de DigiKey - Disponible à prix réduit pour les établissements d'enseignement ou les équipes FIRST. Consultez les fournisseurs individuels pour plus de détails.
2. Câble micro-USB - Assurez-vous qu'il s'agit d'un câble de données
3. 4 piles AA - Rechargeable (exemple) est le meilleur (n'oubliez pas le chargeur)

40.1.1 Assemblage

Note : Consultez les instructions d'assemblage dans le Guide de l'utilisateur XRP.

Vous devez suivre les instructions jusqu'au point où le bras XRP est monté sur le servo.

40.1.2 Imager votre XRP

Le XRP utilise un Raspberry Pi Pico W comme processeur principal. Un firmware spécial devra être installé pour que le robot fonctionne correctement.

Téléchargement

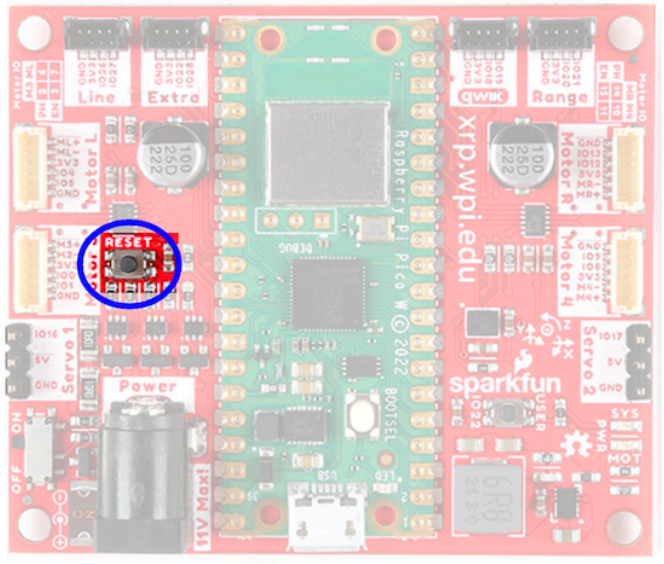
Le firmware XRP doit être téléchargé et écrit sur le Pico W. Cliquez sur « Assets » en bas de la description pour voir les fichiers image disponibles :

XRP-WPILib Firmware

Installation de l'image

Pour créer une image du XRP, effectuez les étapes suivantes :

1. Extraire le contenu du fichier ZIP du firmware. Vous devriez obtenir un fichier `.uf2`.
2. Branchez le XRP sur votre ordinateur avec un câble Micro-USB. Vous devriez voir une LED d'alimentation rouge qui s'allume.
3. Tout en maintenant enfoncé le bouton « BOOTSEL » (le bouton blanc sur le Pico W vert, près du connecteur USB), appuyez rapidement sur le bouton de réinitialisation (encerclé ci-dessous), puis relâchez le bouton « BOOTSEL ».

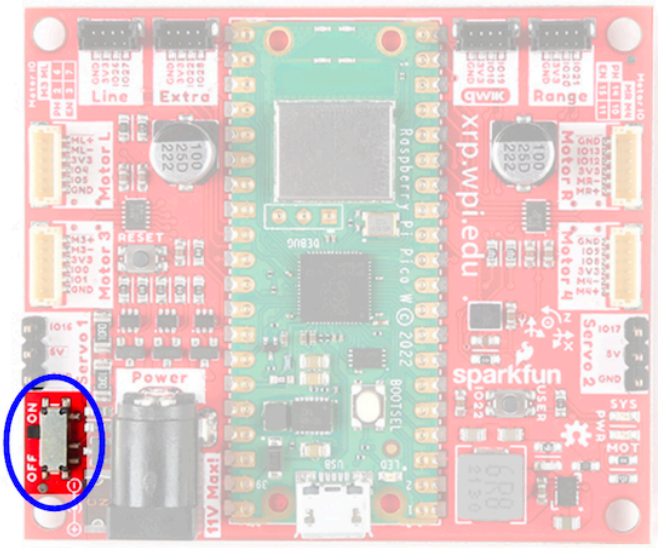


4. La carte se déconnectera temporairement de votre ordinateur, puis se reconnectera en tant que périphérique de stockage USB nommé « RPI-RP2 ».
5. Faites glisser le fichier du firmware `.uf2` dans le lecteur `RPI-RP2` et il mettra automatiquement à jour le firmware.
6. Une fois terminé, le périphérique de stockage USB `RPI-RP2` se déconnectera. À ce stade, vous pouvez déconnecter la carte XRP de votre ordinateur et la faire fonctionner sur batterie.

Premier démarrage

Effectuez les étapes suivantes pour préparer votre XRP à l'utilisation :

1. Assurez-vous que 4 piles AA sont installées
2. Allumez le XRP en faisant glisser l'interrupteur d'alimentation (encerclé ci-dessous) de la carte XRP en position marche. Une LED d'alimentation rouge s'allumera.



3. À l'aide de votre ordinateur, connectez-vous au réseau WiFi XRP en utilisant le SSID `XRP-<IDENT>` (où `<IDENT>` est basé sur l'ID unique du Pico W) avec la phrase secrète WPA2 `xrp-wpilib`.

Note : Si vous allumez le XRP dans un environnement avec plusieurs autres XRP, le SSID peut également être trouvé en connectant le XRP à un ordinateur, en accédant au périphérique de stockage USB (`PICODISK`) qui apparaît et en ouvrant le fichier `xrp-status Fichier .txt`.

---

4. Ouvrez un navigateur Web et connectez-vous à l'interface utilisateur Web à l'adresse `http://192.168.42.1:5000`. Si la page se charge, vous avez établi la connectivité avec le XRP.

Note : Plus d'informations sur l'interface utilisateur Web et la configuration peuvent être trouvées dans la section Web UI.

40.2 Apprendre à connaître votre XRP

40.2.1 Démarrage du XRP

Au démarrage (lorsque le XRP est alimenté via la batterie ou via USB), les événements suivants se produisent :

1. L'IMU se calibrera elle-même. Cela dure environ 3 à 5 secondes et sera indiqué par le clignotement rapide de la LED verte. Idéalement, le XRP doit être placé sur une surface plane avant la mise sous tension et, si nécessaire, les utilisateurs peuvent appuyer sur le bouton de réinitialisation pour redémarrer le processus d'étalonnage du micrologiciel et de l'IMU.
2. Le réseau sera configuré en fonction des paramètres de configuration. Consultez la section sur :doc :`l'interface utilisateur Web ` pour plus d'informations sur la configuration des paramètres réseau. Par défaut, le XRP diffusera son propre point d'accès WiFi.
3. Après cela, le XRP est prêt à être utilisé.

40.2.2 Matériel (Hardware), capteurs et GPIO

Le XRP dispose du matériel/périphériques intégrés suivants :
— 2x motoréducteurs avec encodeurs
— 2x connecteurs motoréducteur supplémentaires avec support encodeur (marqués Motor3 et Motor4)
— 2x connecteurs servo (marqués Servo1 et Servo2)
— 1x unité de mesure inertielle (IMU)
— 1x LED (verte)
— 1x bouton-poussoir (marqué USER)
— 1x capteur de suivi de ligne (exposé comme 2 entrées analogiques)
— 1x capteur à ultrasons de style PING (utilise 2 broches IO numériques, exposées comme une entrée analogique)

Moteurs, roues et encodeurs

Les moteurs utilisés sur le XRP ont une réduction de 48,75 :1 et une vitesse de sortie à vide de 90 tr/min à 4,5 V.

Les roues ont un diamètre de 60 mm (2,3622 »). Elles ont une largeur de voie de 155 mm (6,1 »).

Les encodeurs sont connectés directement à l'arbre de sortie du moteur et disposent de 12 comptes par révolution (CPR). Avec le rapport de démultiplication fourni, cela génère 585 comptes par tour de roue.

Les canaux du moteur sont répertoriés dans le tableau ci-dessous.

Note : Nous utilisons ici des « canaux moteur » au lieu de « canaux PWM » car le XRP nécessite l'utilisation d'un objet spécial `XRPMotor` dans le code WPILib pour interagir avec le matériel.

Canal	Composant matériel XRP
XRPMotor 0	Moteur gauche
XRPMotor 1	Moteur droit
XRPMotor 2	Moteur 3
XRPMotor 3	Moteur 4

Note : Le moteur droit tournera vers l'arrière lorsqu'une sortie positive est appliquée. Ainsi, le contrôleur de moteur correspondant doit être inversé dans le code du robot.

Les canaux de servo sont répertoriés dans le tableau ci-dessous.

Note : Nous utilisons ici des « canaux servo » au lieu de « canaux PWM » car le XRP nécessite l'utilisation d'un objet `XRPServo` spécial dans le code WPILib pour interagir avec le matériel.

Canal	Composant matériel XRP
XRPServo 4	Servo 1
XRPServo 5	Servo 2

Les canaux relatifs aux encodeurs sont répertoriés dans le tableau ci-dessous.

Canal	Composant matériel XRP
DIO 4	Canal de quadrature encodeur gauche A
DIO 5	Canal de quadrature encodeur gauche B
DIO 6	Canal de quadrature encodeur droit A
DIO 7	Canal de quadrature encodeur droit B
DIO 8	Canal A en quadrature de l'encodeur Motor3
DIO 9	Canal B en quadrature de l'encodeur Motor3
DIO 10	Canal A en quadrature de l'encodeur Motor4
DIO 11	Canal B en quadrature de l'encodeur Motor4

Note : Par défaut, les encodeurs comptent lorsque le XRP avance.

L'unité de mesure inertielle

Le XRP comprend une unité de mesure inertielle (IMU) STMicroelectronics LSM6DSOX qui contient un gyroscope à 3 axes et un accéléromètre à 3 axes.

Le XRP calibrera le gyroscope et l'accéléromètre à chaque démarrage (la LED intégrée clignotera rapidement pendant environ 3 à 5 secondes au démarrage).

LED et bouton poussoir intégrés

Le XRP dispose d'un bouton-poussoir (étiqueté USER) et d'une LED verte intégrée qui sont exposés en tant que canaux Digital IO (DIO) au code du robot.

Canal DIO	Composant matériel XRP
DIO 0	Bouton USER
DIO 1	LED verte

Note : Les DIO 2 et 3 sont réservées à une utilisation future du système.

Capteur de suivi de ligne (réflectance)

Lorsqu'il est assemblé conformément aux instructions, le XRP prend en charge un capteur de suivi de ligne avec 2 éléments de détection. Chaque élément de détection mesure la réflectance et les expose en tant que canaux d'entrée analogique au code du robot. Les valeurs renvoyées vont de 0 V (blanc pur) à 5 V (noir pur).

Canal d'entrée analogique	Composant matériel XRP
Entrée analogique 0	Capteur de réflectance gauche
Entrée analogique 1	Capteur de réflexion droit

Capteur à ultrasons

Lorsqu'il est assemblé conformément aux instructions, le XRP prend en charge un capteur à ultrasons de style PING. Ceci est exposé en tant que canal AnalogInput au code du robot. Les valeurs renvoyées vont de 0 V (20 mm) à 5 V (4 000 mm).

Canal d'entrée analogique	Composant matériel XRP
Entrée analogique 2	Capteur à ultrasons

40.3 Prise en charge matérielle XRP

Le robot XRP, ayant une architecture matérielle différente de celle d'un roboRIO, est compatible avec un sous-ensemble de composants du système de contrôle FRC couramment utilisés.

40.3.1 Matériel compatible

En général, le XRP est compatible avec les éléments suivants :

- Moteurs CC Hobby avec encodeurs intégrés (connecteur 6 broches)
- Dispositifs de sortie standard de style RC *PWM* (par exemple, servos, contrôleurs de moteur basés sur PWM)
- Capteurs à ultrasons de style « Ping » (uniquement lorsqu'ils sont connectés au port RANGE)

40.3.2 Matériel incompatible

En raison de limitations matérielles, le XRP n'est pas compatible avec les éléments suivants :

- Encodeurs autres que ceux déjà intégrés aux moteurs hobby
- Capteurs basés sur le temps
- Appareils basés sur CAN

40.3.3 Classes compatibles

Toutes les classes répertoriées ici sont prises en charge par le XRP. Si une classe n'est pas répertoriée ici, supposez qu'elle n'est pas prise en charge et qu'elle *ne fonctionnera pas*.

- `Encoder`
- `AnalogInput`
- `DigitalInput`
- `DigitalOutput`
- `BuiltInAccelerometer`

Note : Les classes de contrôleur de moteur PWM (par exemple « Spark ») et « Servo » ne sont pas prises en charge. Le XRP nécessite l'utilisation de classes spécialisées « XRPMotor » et « XRPServo ».

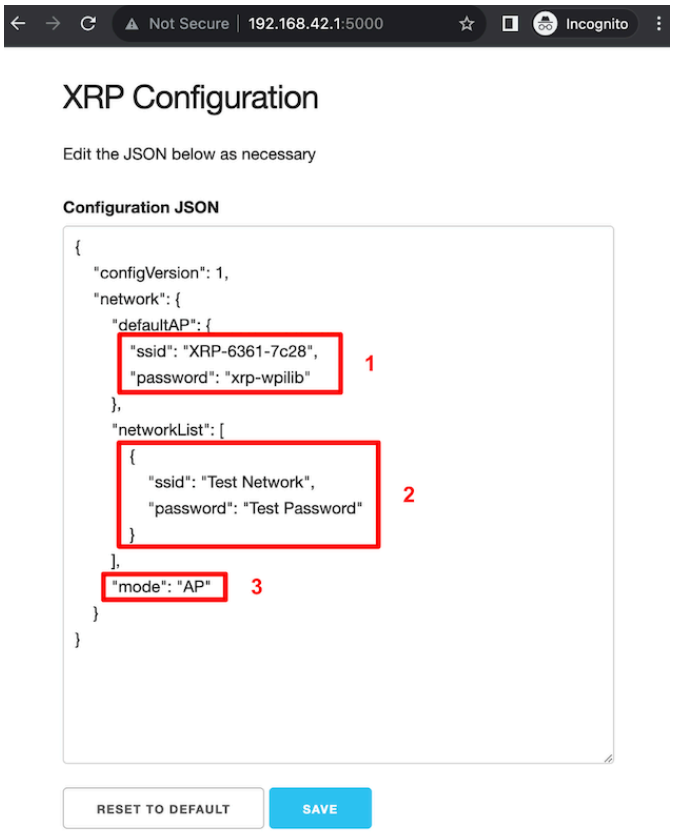
Les classes suivantes sont fournies par XRP Vendordep (intégré à WPILib).

- `XRPGyro`
- `XRPMotor`
- `XRPServo`
- `XRPOnBoardIO`

40.4 The XRP Web UI

The XRP provides a simple Web UI for configuration. It is accessible at `http://<IP Address of XRP>:5000`. By default, this is `http://192.168.42.1:5000`.

The XRP configuration is a simple JSON object that allows a user to configure the network settings of the XRP.



40.4.1 Switching Network Modes

Box 3 in the image above shows the field that needs to be changed in order to switch the XRP from Access Point mode to/from Station mode. In Access Point (`AP`) mode, the XRP will broadcast a WiFi network. In Station (`STA`) mode, the XRP will connect to an existing WiFi network. Update the `mode` field with the appropriate value (`AP`/`STA`).

40.4.2 Setting up a default Access Point (AP)

By default, the XRP will operate in Access Point mode, where it broadcasts a WiFi network. Box 1 in the image above shows which fields control the settings for the AP SSID and passphrase.

If the operating mode is set to `AP`, the access point information will be used to create the WiFi Access Point. If the mode is set to `STA` (station) and the XRP is unable to connect to any of the listed WiFi networks, then it will fall back to AP mode, again, using the information specified in box 1.

40.4.3 Connecting to an existing WiFi network

Box 2 in the image above shows an example of listing a WiFi network that you want the XRP to connect to. the `networkList` array can be populated with as many preferred networks as you would like (following the same format as Box 2). When set to `STA` mode, the XRP will attempt to connect to each listed network in order. If none of the networks are available, the XRP will fallback into AP mode.

Note : If you are unsure about what mode the XRP is operating in, or which WiFi network it is connected to, you can connect the XRP to a computer via a USB cable. A USB storage device named `PICODISK` will appear, and the `xrp-status.txt` file within it will list the appropriate network information.

40.5 Programmation du XRP

L'écriture d'un programme pour le XRP est très similaire à l'écriture d'un programme pour un robot FRC classique. En fait, tous les mêmes outils (Visual Studio Code, Driver Station, SmartDashboard, etc.) peuvent être utilisés avec le XRP.

40.5.1 Création d'un programme XRP

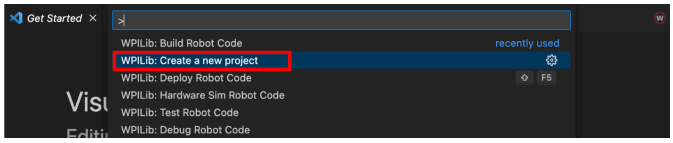
Créer un nouveau programme pour un XRP revient à créer un programme FRC normal, similaire aux étapes de programmation :doc :`Zero To Robot `.

WPILib est livré avec deux modèles pour les projets XRP, dont un basé sur TimedRobot et un modèle de projet basé sur des commandes. De plus, un exemple de projet est fourni qui présente certaines des fonctionnalités intégrées du XRP et montre comment utiliser les classes XRP exposées par le fournisseur. Cet article explique comment créer un projet à partir de cet exemple.

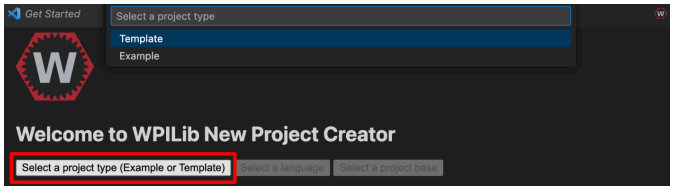
Note : In order to program the XRP using C++, a compatible C++ desktop compiler must be installed. See *Robot Simulation - Additional C++ Dependency*.

Création d'un nouveau projet WPILib XRP

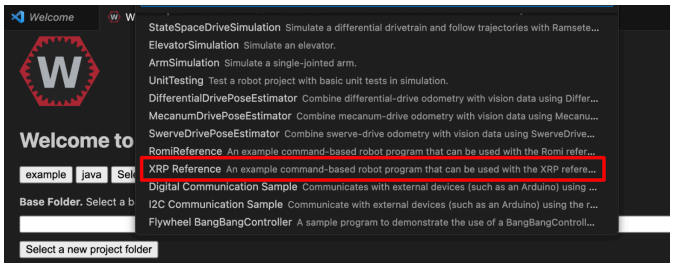
Affichez la palette de commandes Visual Studio Code avec `Ctrl+Maj+P`, et tapez « New project ». Sélectionnez la commande « Create a new project » :



Cela fera apparaître « New Project Creator Window ». À partir de là, cliquez sur « Select a project type (Example or Template) » et choisissez « Example » dans l'invite qui apparaît :



Ensuite, une liste d'exemples apparaîtra. Faites défiler la liste pour trouver l'exemple « XRP Reference » :



Remplissez le reste des champs dans le « New Project Creator » et cliquez sur « Generate Project » pour créer le nouveau projet de robot.

Exécuter un programme XRP

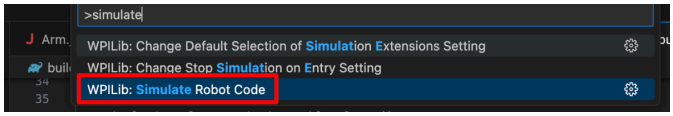
Une fois le projet de robot généré, il est essentiellement prêt à être exécuté. Le projet dispose d'une classe « Drivetrain » prédéfinie et d'une commande par défaut associée qui vous permet de piloter le XRP à l'aide d'un joystick.

Un aspect où un projet XRP diffère d'un projet de robot FRC classique est que le code n'est pas déployé directement sur le XRP. Au lieu de cela, un projet XRP s'exécute sur votre ordinateur de développement et exploite le framework de simulation WPILib pour communiquer avec le XRP.

Pour exécuter un programme XRP, assurez-vous d'abord que votre XRP est sous tension. Ensuite, connectez-vous au réseau WiFi `XRP-<IDENT>` diffusé par le XRP. Si vous avez modifié les paramètres réseau du XRP (par exemple, pour le connecter à votre propre réseau), vous pouvez modifier l'adresse IP que votre programme utilise pour se connecter au XRP. Pour ce faire, ouvrez le fichier `build.gradle` et mettez à jour la ligne `wpi.sim.envVar` avec l'adresse IP appropriée.

```
43 //Sets the XRP Client Host
44 wpi.sim.envVar("HALSIMXRP_HOST", "192.168.42.1")
45 wpi.sim.addXRPClient().defaultEnabled = true
```

Maintenant, pour démarrer votre code de robot XRP, ouvrez la palette de commandes WPILib (tapez `Ctrl+Shift+P`) et sélectionnez « Simulate Robot Code », ou appuyez sur `F5`.



Si tout se passe bien, vous devriez voir l'interface graphique de simulation apparaître et voir les valeurs du gyroscope et de l'accéléromètre se mettre à jour.

Votre code XRP est maintenant exécuté !

41

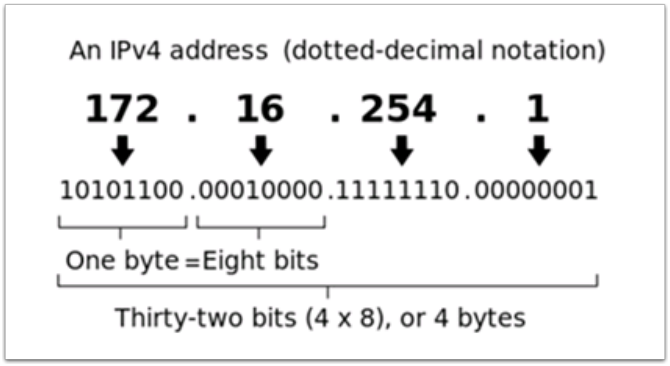
Introduction à la réseautique

Cette section décrit la configuration et l'utilisation de base du robot relatives à la communication entre la Driver Station et le roboRIO.

41.1 Bases de la réseautique

41.1.1 Qu'est-ce qu'une adresse IP ?

Une adresse IP est une série unique de nombres, séparés par des points qui identifie chaque périphérique sur un réseau. Chaque adresse IP est divisée en 4 sections (octets) allant de 0 à 255.



Comme indiqué ci-dessus, cela signifie que chaque adresse IP est une adresse de 32 bits ce qui signifie qu'il y a 2^{32} adresses, ou près de 4.300.000.000 adresses possibles. Cependant, la plupart d'entre elles sont utilisées publiquement dans des applications telles que les serveurs Web.

Cela soulève notre **premier élément clé** de l'adressage IP : Chaque appareille sur le réseau doit avoir une adresse IP unique. Il n'est pas possible pour deux appareils d'avoir la même adresse IP, sinon des collisions se produiront.

Since there are only 4 billion addresses, and there are more than 4 billion computers connected to the internet, we need to be as efficient as possible with giving out IP addresses. This brings us to public vs. private addresses.

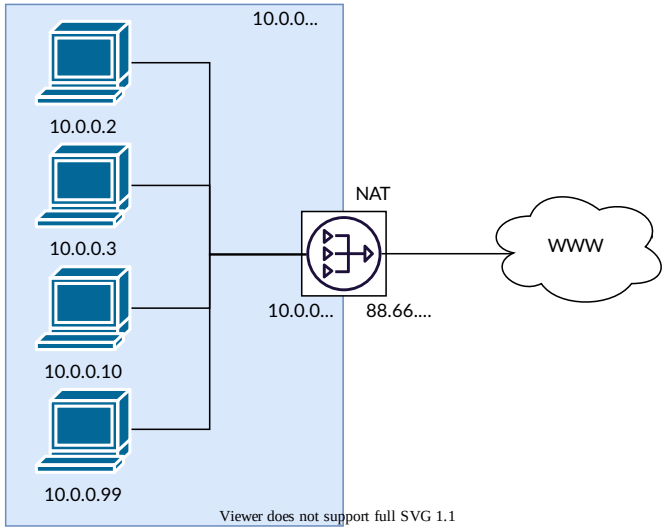
41.1.2 Adresses IP publiques vs privées

Pour être efficace avec l'utilisation des adresses IP, l'idée de « plages d'adresses IP réservées » a été implémentée. En bref, cela signifie qu'il existe des plages d'adresses IP qui ne seront jamais attribuées à des serveurs Web, et ne seront utilisées que pour les réseaux locaux, tels que ceux de votre maison.

Key point #2 : Unless you are directly connecting to your internet provider's basic modem (no router function), your device will have an IP Address in one of these ranges. This means that at any local network, such as : your school, work office, home, etc., your device will 99% of the time have an IP address in a range listed below :

Class	Bits	Start Address	End Address	Number of Addresses
A	24	10.0.0.0	10.255.255.255	16,777,216
B	20	172.16.0.0	172.31.255.255	1,048,576
C	16	192.168.0.0	192.168.255.255	65,536

Ces plages réservées nous permettent d'attribuer une « adresse IP sans réserve » à une maison entière, puis d'utiliser plusieurs adresses dans une plage réservée pour connecter plus d'un ordinateur à Internet. Un processus sur le routeur Internet de la maison connu sous le nom de **NAT** (Traduction d'adresses réseau), gère le processus de suivi de l'IP privée qui demande des données, en utilisant l'IP publique pour demander ces données à partir d'Internet, puis en transmettant les données retournées à l'IP privée qui les a demandés. Ce qui nous permet d'utiliser les mêmes adresses IP réservées pour de nombreux réseaux locaux, sans provoquer de conflits. Une image de ce processus est présentée ci-dessous.



Note : Pour les réseaux FRC®, nous utiliserons la plage `10.0.0.0`. Cette plage nous permet d'utiliser le format `10.TE.AM.xx` pour les adresses IP, alors que l'utilisation des réseaux de classe B ou C ne permettrait qu'à un sous-ensemble d'équipes de suivre le format. Un exemple d'utilisation de ce format serait `10.17.50.1` pour FRC Team 1750.

41.1.3 Comment ces adresses sont-elles attribuées ?

We've covered the basics of what IP addresses are, and which IP addresses we will use for the FRC competition, so now we need to discuss how these addresses will get assigned to the devices on our network. We already stated above that we can't have two devices on the same network with the same IP Address, so we need a way to be sure that every device receives an address without overlapping. This can be done Dynamically (automatic), or Statically (manual).

Dynamiquement

L'attribution dynamique d'adresses IP signifie que nous laissons un périphérique sur le réseau gérer les attributions d'adresses IP. Cela se fait par l'intermédiaire du protocole de configuration dynamique des hôtes (DHCP). DHCP contient de nombreux composants, mais pour la portée de ce document, nous allons le considérer comme un service qui gère automatiquement le réseau. Chaque fois que vous branchez un nouvel appareil au réseau, le service DHCP voit le nouvel appareil, puis lui fournit une adresse IP disponible ainsi que les autres paramètres réseau requis pour que l'appareil communique. Cela peut signifier qu'il y a des moments où nous ne connaissons pas l'adresse IP exacte de chaque appareil.

Qu'est-ce qu'un serveur DHCP ?

A *DHCP* server is a device that runs the DHCP service to monitor the network for new devices to configure. In larger businesses, this could be a dedicated computer running the DHCP service and that computer would be the DHCP server. For home networks, FRC networks, and other smaller networks, the DHCP service is usually running on the router ; in this case, the router is the DHCP server.

Cela signifie que si jamais vous vous retrouvez dans une situation où vous devez avoir un serveur DHCP assignant des adresses IP à vos périphériques réseau, il est aussi simple que de trouver le routeur domestique le plus proche, et de le brancher.

Statiquement

L'attribution statique d'adresses IP signifie que nous disons manuellement à chaque appareil du réseau quelle adresse IP nous voulons qu'il ait. Cette configuration se produit à travers un paramètre sur chaque appareil. En désactivant DHCP sur le réseau et en assignant les adresses manuellement, nous avons l'avantage de connaître l'adresse IP exacte de chaque appareil sur le réseau, mais parce que nous avons attribué chacune manuellement et il n'y a pas de service de suivi des adresses IP utilisées, nous devons nous-mêmes garder une trace de cette information . Tout en définissant statiquement les adresses IP, nous devons faire attention à ne pas attribuer d'adresses en double, et nous devons être sûrs que nous configurons correctement les autres paramètres du réseau (tels que le masque de sous-réseau et la passerelle par défaut) sur chaque appareil.

41.1.4 Qu'est-ce qu'un lien-local ?

Si un périphérique n'a pas d'adresse IP, il ne peut pas communiquer sur un réseau. Cela peut devenir un problème si nous avons un périphérique qui est configuré pour acquérir dynamiquement son adresse à partir d'un serveur DHCP, mais qu'il n'y a pas de serveur DHCP sur le réseau. Par exemple, lorsque vous avez un ordinateur portable directement connecté à un roboRIO et les deux sont configurés pour acquérir dynamiquement une adresse IP. Aucun des deux appareils n'est un serveur DHCP, et comme ils sont les deux seuls appareils sur le réseau, des adresses IP ne leur seront donc pas attribuées automatiquement.

Les adresses de type lien-local nous donnent un ensemble standard d'adresses auxquelles nous pouvons « nous replier » si un périphérique configuré pour acquérir dynamiquement n'est pas en mesure d'acquérir une adresse. Si cela se produit, l'appareil s'attribuera une adresse IP dans la plage d'adresses `169.254.xx.yy` ; il s'agit d'une adresse lien-local. Dans notre roboRIO et l'exemple d'ordinateur ci-dessus, les deux appareils se rendront compte qu'il ne leur a pas été affecté d'adresse IP et s'assigneront eux-mêmes une adresse lien-local. Une fois que leur auront été attribuées deux adresses dans la plage `169.254.xx.yy`, ils seront dans le même réseau et seront en mesure de communiquer, même s'ils ont été réglés sur dynamique et qu'un serveur DHCP n'a pas attribué d'adresses.

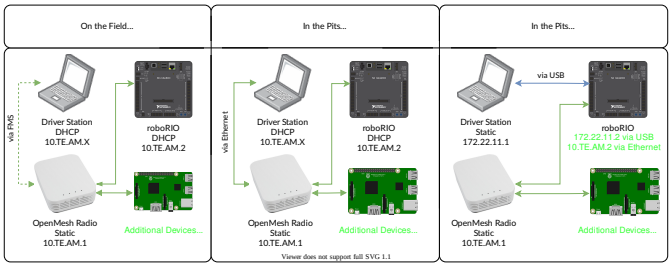
41.1.5 Adressage IP pour FRC

Voir *IP Networking Article* pour plus d'informations.

Mélange de configurations dynamique et statique

Sur le terrain, l'équipe ne doit pas déceler de problèmes avec le fait que les périphériques soient réglés statiquement dans la plage `10.TE.AM.xx` et de voir le terrain assigner des adresses DHCP tant qu'il n'y a pas de conflits d'adresses IP comme mentionné dans la section ci-dessus.

Dans les puits, une équipe peut rencontrer des problèmes avec le mélange des appareils Statique et DHCP pour la raison suivante. Comme mentionné ci-dessus, les périphériques DHCP se rabattront sur une adresse lien-local (`169.254.xx.yy`) si un serveur n'est pas présent. Pour les périphériques statiques, l'adresse IP sera toujours la même. Si le serveur DHCP n'est pas présent et que le roboRIO, la station de pilotage et l'ordinateur portable se rabattent sur des adresses de type lien-local, les périphériques réglés statiquement dans la plage 10.TE.AM.xx seront dans un réseau différent et non visibles pour ceux qui ont des adresses de type lien-local. Une description visuelle de ceci est fournie ci-dessous :



Avertissement : Lorsqu'il est connecté via USB au roboRIO, une configuration *Transfert de port* est nécessaire pour accéder aux appareils connectés à la radio OpenMesh (sur le réseau vert indiqué ci-dessus).

Ports réseau disponibles

Please see R704 of the 2024 Game Manual for information regarding available network ports.

41.1.6 mDNS

mDNS, ou multicast Domain Name System est un protocole qui nous permet de bénéficier des fonctionnalités de DNS, sans avoir un serveur DNS sur le réseau. Pour que cela soit plus clair, prenons un peu de recul et parlons de ce qu'est le DNS.

DNS : qu'est-ce que c'est ?

DNS (Domain Name System) can become a complex topic, but for the scope of this paper, we are going to just look at the high-level overview of DNS. In the most basic explanation, DNS is what allows us to relate human-friendly names for network devices to IP Addresses, and keep track of those IP addresses if they change.

Example 1 : Let's look at the site `www.google.com`. The IP address for this site is `172.217.164.132`, however that is not very user-friendly to remember !

Whenever a user types `www.google.com` into their computer, the computer contacts the DNS server (a setting provided by DHCP !) and asks what is the IP address on file for `www.google.com`. The DNS server returns the IP address and then the computer is able to use that to connect to the Google website.

Exemple 2 : Sur votre réseau domestique, vous disposez d'un serveur nommé `MYCOMPUTER` auquel vous souhaitez vous connecter à partir de votre ordinateur portable. Votre réseau utilise DHCP de sorte que vous ne connaissez pas l'adresse IP de `MYCOMPUTER`, cependant le DNS vous permet de vous connecter uniquement en utilisant le nom `MYCOMPUTER`. De plus, à chaque fois que les attributions DHCP s'actualisent, `MYCOMPUTER` peut se retrouver avec une adresse différente, mais parce que vous vous connectez en utilisant le nom `MYCOMPUTER` au lieu d'une adresse IP spécifique, l'enregistrement DNS est mis à jour et vous êtes toujours en mesure de vous connecter.

This is the second benefit to DNS and the most relevant for FRC. With DNS, if we reference devices by their friendly name instead of IP Address, we don't have to change anything in our program if the IP Address changes. DNS will keep track of the changes and return the new address if it ever changes.

DNS pour FRC

On the field and in the pits, there is no DNS server that allows us to perform the lookups like we do for the Google website, but we'd still like to have the benefits of not remembering every IP Address, and not having to guess at every device's address if DHCP assigns a different address than we expect. This is where mDNS comes into the picture.

Le mDNS nous offre les mêmes avantages que le DNS traditionnel, mais est juste implémenté d'une manière qui ne nécessite pas un serveur. Chaque fois qu'un utilisateur demande à se connecter à un appareil à l'aide d'un nom convivial, le mDNS envoie un message demandant à l'appareil portant ce nom de s'identifier. L'appareil portant ce nom envoie alors un message de retour, y compris son adresse IP afin que tous les appareils du réseau puissent mettre à jour leurs informations. Le mDNS est ce qui nous permet de nous référer à notre roboRIO comme `roboRIO-TEAM-FRC.local` et de le faire connecter sur un réseau DHCP.

Note : Si un périphérique utilisé pour la FRC ne prend pas en charge le mDNS, il lui sera attribué une adresse IP dans la plage 10.TE.AM.20 - 10.TE.AM.255, mais nous ne saurons pas l'adresse IP exacte pour nous connecter et nous ne serons pas, non plus, en mesure d'utiliser un nom convivial comme auparavant. Dans ce cas, ce périphérique devrait disposer d'une adresse IP statique.

---

mDNS - Principes

Multicast Domain Name System (mDNS) is a system which allows for resolution of hostnames to IP addresses on small networks with no dedicated name server. To resolve a hostname a device sends out a multicast message to the network querying for the device. The device then responds with a multicast message containing its IP. Devices on the network can store this information in a cache so subsequent requests for this address can be resolved from the cache without repeating the network query.

mDNS - Fournisseurs

Pour utiliser mDNS, une implémentation mDNS doit être installée sur votre PC. Voici quelques implémentations mDNS courantes pour chaque importante plate-forme :

Windows :
- **NI mDNS Responder :** Installé avec NI FRC Game Tools
- **Apple Bonjour :** Installé avec iTunes

OSX :
- **Apple Bonjour :** Installé par défaut

Linux :
- **nss-mDNS/Avahi/Zeroconf :** Installé et activé par défaut sur certaines variantes Linux (comme Ubuntu ou Mint). Peut avoir besoin d'être installé ou activé sur d'autres (comme Arch)

mDNS - Pare-feu

Note : Selon la configuration de votre PC, aucun changement ne peut être nécessaire, cette section est fournie pour vous aider dans la résolution des problèmes.

Pour fonctionner correctement, mDNS doit être autorisé à passer à travers votre pare-feu. Étant donné que le trafic réseau provient de l'implémentation mDNS et non directement de la Drive Station ou de l'IDE, l'autorisation de ces seules applications à passer à travers votre pare-feu pourrait ne pas suffire. Il existe deux façons principales de résoudre les problèmes de pare-feu mDNS :

- Ajouter une exception d'application/service pour l'implémentation du mDNS (NI mDNS Responder est `C:\Program Files\National Instruments\Shared\mDNS Responder\nimdnsResponder.exe`)
- Ajoutez une exception de port pour le trafic vers/à partir de UDP 5353. Plages IP :
  - `10.0.0.0 - 10.255.255.255`
  - `172.16.0.0 - 172.31.255.255`
  - `192.168.0.0 - 192.168.255.255`
  - `169.254.0.0 - 169.254.255.255`
  - `224.0.0.251`

mDNS - Prise en charge des navigateurs

La plupart des navigateurs Web devraient être en mesure d'utiliser l'adresse mDNS pour accéder au serveur web du roboRIO tant qu'un fournisseur mDNS est installé. Ces navigateurs incluent Microsoft Edge, Firefox et Google Chrome.

41.1.7 USB

Si vous utilisez l'interface USB, aucune configuration réseau n'est requise (vous devez toutefois vous assurer d'avoir préalablement installé les outils de jeu *Installer les Outils de Jeu FRC* afin de fournir le pilote USB roboRIO). Le pilote roboRIO configurera automatiquement l'adresse IP de l'hôte (votre ordinateur) et du roboRIO et le logiciel mentionné ci-dessus devrait être en mesure de localiser et d'utiliser votre roboRIO.

41.1.8 Ethernet/Sans fil

The *Programmation de votre Radio* will enable the DHCP server on the OpenMesh radio in the home use case (AP mode), if you are putting the OpenMesh in bridge mode and using a router, you can enable DHCP addressing on the router. The bridge is set to the same team-based IP address as before (`10.TE.AM.1`) and will hand out DHCP address from `10.TE.AM.20` to `10.TE.AM.199`. When connected to the field, *FMS* will also hand out addresses in the same IP range.

41.1.9 Résumé

Les adresses IP sont ce qui nous permet de communiquer avec des appareils sur un réseau. En FRC, ces adresses seront dans la plage 10.TE.AM.xx si nous sommes connectés à un serveur DHCP ou ces adresses IP sont attribuées statiquement, ou dans la plage de lien local `169.254.xx.yy` si les périphériques sont configurés sur DHCP, mais il n'y a pas de serveur présent. Pour plus d'informations sur le fonctionnement des adresses IP, voir cet article de Microsoft.

Si tous les appareils du réseau prennent en charge le mDNS, tous les appareils peuvent être configurés sur DHCP et invoqués à l'aide de leurs appellations conviviales (ex. `roboRIO-TEAM-FRC.local`). Si certains périphériques ne prennent pas en charge le mDNS, ils devront être configurés pour utiliser des adresses IP statiques.

Si tous les périphériques sont configurés pour utiliser des attributions d'adresses IP DHCP ou statiques (avec des paramètres statiques corrects), la communication doit fonctionner à la fois dans le puits et sur le terrain sans que des modifications ne soient nécessaires. S'il existe un mélange de certains périphériques Statique et DHCP, les périphériques statiques se connecteront sur le terrain, mais ne se connecteront pas dans le puis. Cette situation peut être résolue soit en définissant tous les périphériques sur des paramètres statiques, soit en laissant les paramètres actuels et en fournissant un serveur DHCP dans le puits.

41.2 IP Configurations

Note : Ce document décrit la configuration d'adresses IP utilisées lors d'événements, tant sur les terrains que dans les puits, les problèmes potentiels et les solutions de rechange.

41.2.1 Notation IP TE.AM

La notation TE.AM est utilisée dans le cadre des IP à de nombreux endroits dans ce document. Cette notation fait référence à la séparation de votre numéro d'équipe à quatre chiffres en paire à deux chiffres pour les octets d'adresses IP.

Exemple : `10.TE.AM.2`

Team 12 - `10.0.12.2`

Team 122 - `10.1.22.2`

Team 1212 - `10.12.12.2`

Team 1202 - `10.12.2.2`

Team 1220 - `10.12.20.2`

Team 3456 - `10.34.56.2`

41.2.2 Sur le terrain

Cette section décrit la réseautique lorsqu'on est connecté au réseau du terrain pour un match

Configuration DHCP pour usage sur le terrain

The Field Network runs a *DHCP* server with pools for each team that will hand out addresses in the range of `10.TE.AM.20` to `10.TE.AM.199` with a subnet mask of `255.255.255.0`, and a default gateway of `10.TE.AM.4`. When configured for an event, the Team Radio runs a DHCP server with a pool for devices onboard the robot that will hand out addresses in the range of 10.TE.AM.200 to 10.TE.AM.219 with a subnet mask of 255.255.255.0, and a gateway of 10.TE.AM.1.

- OpenMesh OM5P-AN ou OM5P-AC radio - Statique `10.TE.AM.1` programmée par le Kiosk
- roboRIO - DHCP `10.TE.AM.2` attribuée par la radio du robot
- Driver Station - DHCP (« Obtain an IP address automatically ») 10.TE.AM.X assignée par le terrain
- Caméra IP (si elle est utilisée) - DHCP `10.TE.AM.Y` attribuée par la radio du robot
- Autres appareils (s'ils sont utilisés) - DHCP `10.TE.AM.Z` attribuée par la radio du robot

Configuration statique pour usage sur le terrain

Il est également possible de configurer des adresses IP statiques sur vos appareils pour prendre en charge les périphériques ou les logiciels qui ne prennent pas en charge mDNS. Ce faisant, vous devez vous assurer d'éviter les adresses qui seront utilisées lorsque le robot sera sur le réseau du terrain de jeu. Ces adresses sont `10.TE.AM.1` pour la radio OpenMesh, `10.TE.AM.4` pour le routeur du terrain, et tout ce qui est supérieur à `10.TE.AM.20` qui peut être affecté à un périphérique configuré pour DHCP ou réservé. La configuration du réseau du roboRIO peut être définie à partir du webdashboard.

- OpenMesh radio - Statique `10.TE.AM.1` programmée par le Kiosk
- roboRIO - Statique `10.TE.AM.2` serait un choix raisonnable, masque sous-réseau de ""255.255.255.0"" (par défaut)
- Driver Station - Static `10.TE.AM.5` would be a reasonable choice, subnet mask **must** be `255.0.0.0` to enable the DS to reach both the robot and *FMS* Server, without additionally configuring the default gateway. If a static address is assigned and the subnet mask is set to `255.255.255.0`, then the default gateway must be configured to `10.TE.AM.4`.
- Caméra IP (si elle est utilisée) - Statique `10.TE.AM.11` serait un choix raisonnable, sous-réseau `255.255.255.0` devrait parfaitement convenir
- Autres appareils - Statique `10.TE.AM.6-.10` ou `.12-.19` (.11 si la caméra n'est pas présente) sous-réseau `255.255.255.0`

41.2.3 Dans le puits

Note : Nouveau pour 2018 : Il y a maintenant un serveur DHCP qui fonctionne du côté câblé de la radio du robot dans la configuration de l'événement.

Configuration DHCP pour usage dans le puits

- OpenMesh radio - Statique `10.TE.AM.1` programmée par le Kiosk.
- roboRIO - `10.TE.AM.2`, attribuée par la radio du robot
- Driver Station - DHCP (« Obtain an IP address automatically »), `10.TE.AM.X`, attribuée par la radio du robot
- Caméra IP (si elle est utilisée) - DHCP, `10.TE.AM.Y`, attribuée par la radio du robot
- Autres appareils (s'ils sont utilisés) - DHCP, `10.TE.AM.Z`, attribuée par la radio du robot

Configuration statique pour usage dans le puits

Il est également possible de configurer des adresses IP statiques sur vos appareils pour accomoder des périphériques ou des logiciels qui ne prennent pas en charge le mDNS. Ce faisant, vous devez vous assurer d'éviter les adresses qui seront utilisées lorsque le robot est sur le réseau du terrain de jeu. Ces adresses sont `10.TE.AM.1` pour la radio OpenMesh et `10.TE.AM.4` pour le routeur du terrain de jeu.

41.3 Dépannage des problèmes réseaux du roboRIO

The roboRIO and FRC® tools use dynamic IP addresses (*DHCP*) for network connectivity. This article describes steps for troubleshooting networking connectivity between your PC and your roboRIO

41.3.1 Ping le roboRIO à l'aide de mDNS

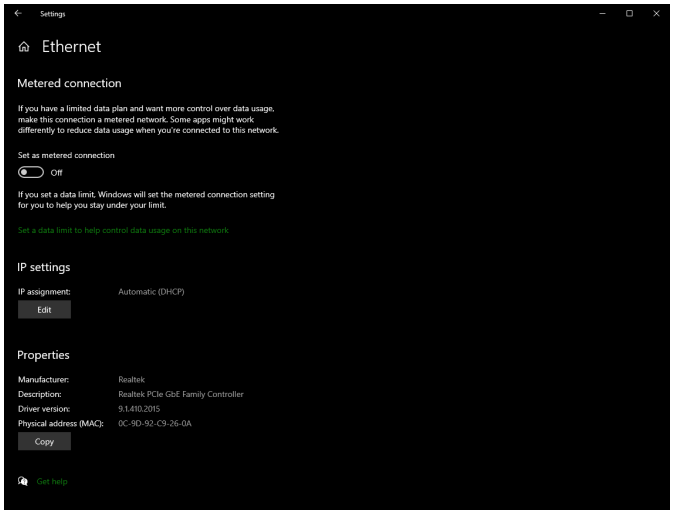
La première étape pour identifier les problèmes de réseau du roboRIO est de distinguer s'il s'agit d'un problème d'application ou d'un problème de réseau général. Pour ce faire, cliquez sur **Démarrer -> taper cmd -> appuyez sur Entrée** pour ouvrir l'invite de commandes. Tapez `ping roboRIO-####-FRC.local` où `####` est votre numéro d'équipe (sans zéros en position de tête) et appuyez sur Entrer. Si l'interrogation ping réussit, le problème est probablement avec l'application spécifique, vérifiez la configuration de votre numéro d'équipe dans l'application et vérifiez votre configuration de pare-feu.

41.3.2 Envoyez un ping à l'adresse IP du roboRIO

S'il n'y a pas de réponse, essayez d'envoyer une requête ping à `10.TE.AM.2` (: ref :*TE.AM IP Notation <docs / networking / networking-introduction / ip-configurations : TE.AM IP Notation>*) en utilisant l'invite de commande comme décrit ci-dessus. Si cela fonctionne, vous rencontrez un problème pour résoudre l'adresse mDNS sur votre PC. Les deux causes les plus courantes sont l'absence d'un résolveur mDNS installé sur le système et d'un serveur DNS sur le réseau qui tente de résoudre l'adresse .local à l'aide de DNS standard.

— Vérifiez que vous avez installé un résolveur mDNS sur votre système. Sous Windows, cette tâche est généralement remplie par le NI FRC Game Tools. Pour plus d'informations sur les résolveurs mDNS, reférez-vous au document *Network Basics article*.
— Déconnectez votre ordinateur de tout autre réseau et assurez-vous que l'OM5P-AN est configuré comme point d'accès, à l'aide de :ref :`FRC Radio Configuration Utility <docs / zero-to-robot / step-3 / radio-programmation : Programming votre radio> `. La suppression de tout autre routeur du système permettra de vérifier qu'aucun serveur DNS n'est à l'origine du problème.

41.3.3 Si le Ping échoue



Si la commande ping de l'adresse IP échoue, vous pouvez avoir un problème avec la configuration réseau du PC. Le PC doit être configuré sur **Automatique**. Pour vérifier cela, cliquez sur : *Start -> Settings -> Network & Internet*. En fonction de votre réseau, sélectionnez *Wifi* ou *Ethernet*. Cliquez ensuite sur votre réseau connecté. Faites défiler jusqu'à **IP settings** et cliquez sur *Edit* et assurez-vous que l'option *Automatic (DHCP)* est sélectionnée.

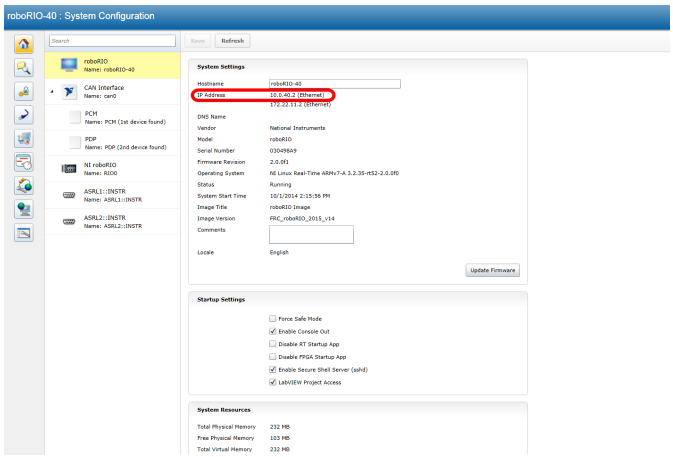
41.3.4 Dépannage de la connexion USB

Si vous essayez de résoudre le problème de la connexion USB, essayez une interrogation ping de l'adresse IP du roboRIO. Tant qu'il n'y a qu'un seul roboRIO connecté au PC, il doit être configuré à 172.22.11.2. Si ce cette interrogation ping échoue, assurez-vous d'avoir le roboRIO connecté et alimenté, et que vous avez installé les outils NI FRC Game Tools. Les outils de jeu installent les pilotes roboRIO nécessaires à la connexion USB.

Si ce ping réussit, mais que le ping .local échoue, il est probable que le nom d'hôte roboRIO soit configuré incorrectement, soit que vous soyez connecté à un serveur DNS qui tente de résoudre l'adresse .local.

— Verify that your roboRIO has been imaged for your team number : *roboRIO 1 roboRIO 2*. This sets the hostname used by mDNS.
— *Disable all other network adapters*

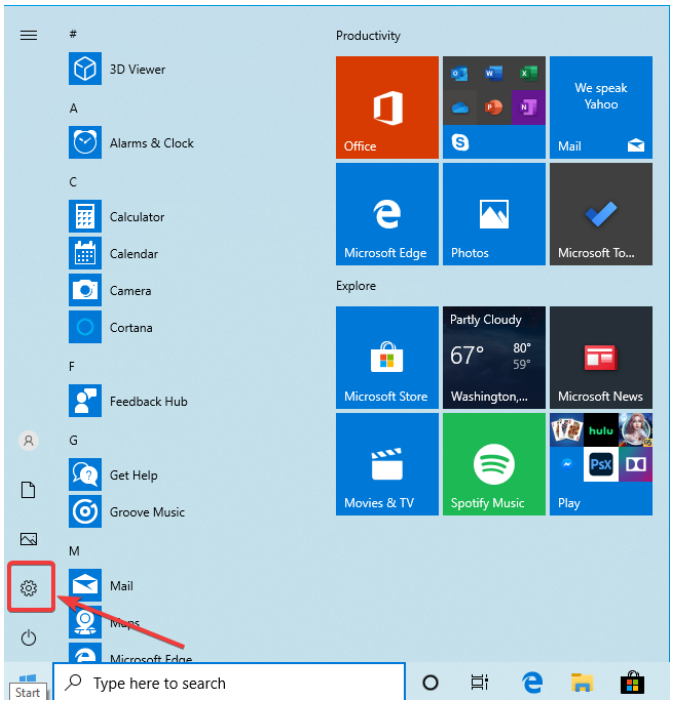
41.3.5 Connexion Ethernet



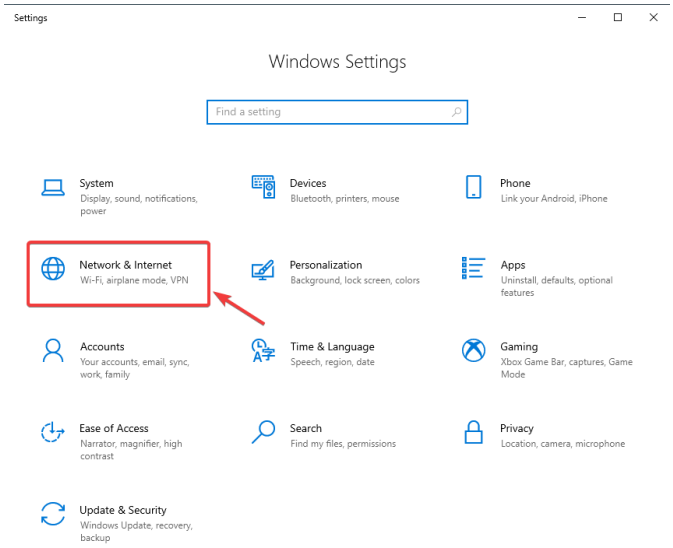
Si vous dépannez une connexion Ethernet, il peut être utile de s'assurer d'abord que vous pouvez vous connecter au roboRIO à l'aide de la connexion USB. À l'aide de la connexion USB, ouvrez le *roboRIO webdashboard* et vérifiez que le roboRIO a une adresse IP sur l'interface ethernet. Si vous directement connecté au roboRIO cette adresse doit être auto-assignée à `169.*.*.*`, si vous êtes connecté à la radio OM5P-AN, il doit s'agir d'une adresse de la forme `10.TE.AM.XX` où TEAM est votre numéro d'équipe FRC à quatre chiffres. Si la seule adresse IP ici est l'adresse USB, vérifiez la connexion ethernet physique du roboRIO .

41.3.6 Désactivation des adaptateurs réseau

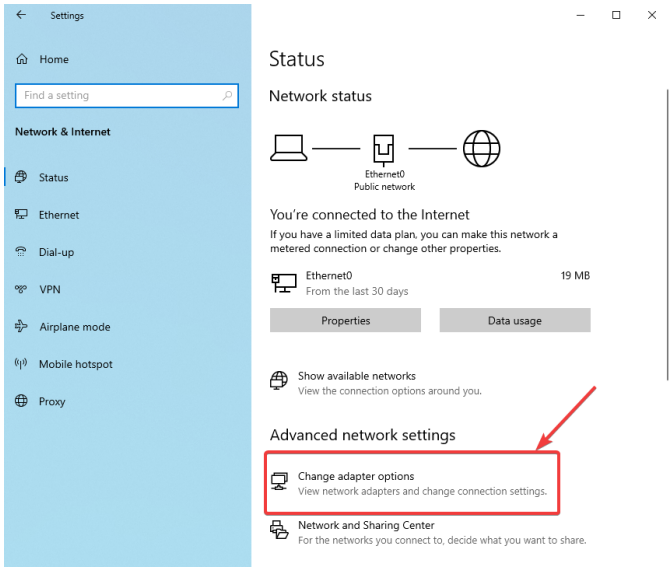
Ce n'est pas toujours la même chose que d'éteindre les adaptateurs avec un bouton physique ou de mettre le PC en mode avion. Les étapes suivantes fournissent plus de détails sur la désactivation des adaptateurs.



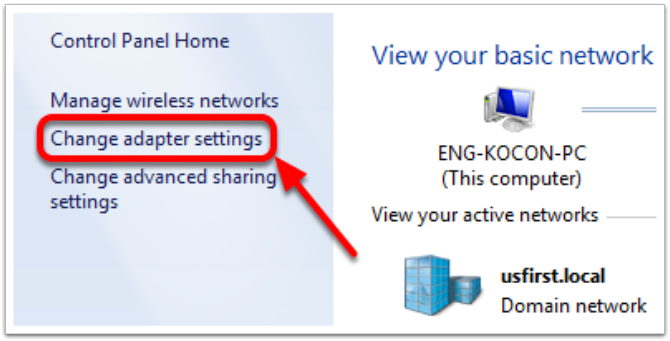
Ouvrez l'application Paramètres en cliquant sur l'icône des paramètres.



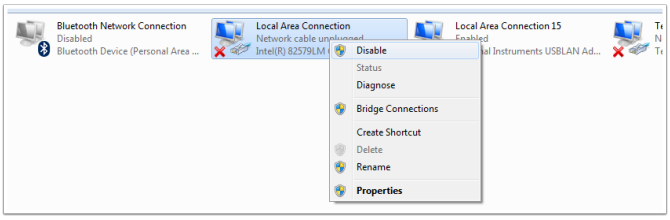
Choisissez la catégorie *Réseau et Internet*.



Cliquez sur *Changer les options de l'adaptateur*.



Dans le volet gauche, cliquez sur *Modifier les paramètres de l'adaptateur*.



Pour chaque adaptateur autre que celui connecté à la radio, faites un clic droit sur l'adaptateur et sélectionnez *Désactiver* dans le menu.

41.3.7 Proxies

— Proxies. Le fait d'avoir un proxy activé peut causer des problèmes avec le réseau du roboRIO.

41.4 Configuration du pare-feu Windows

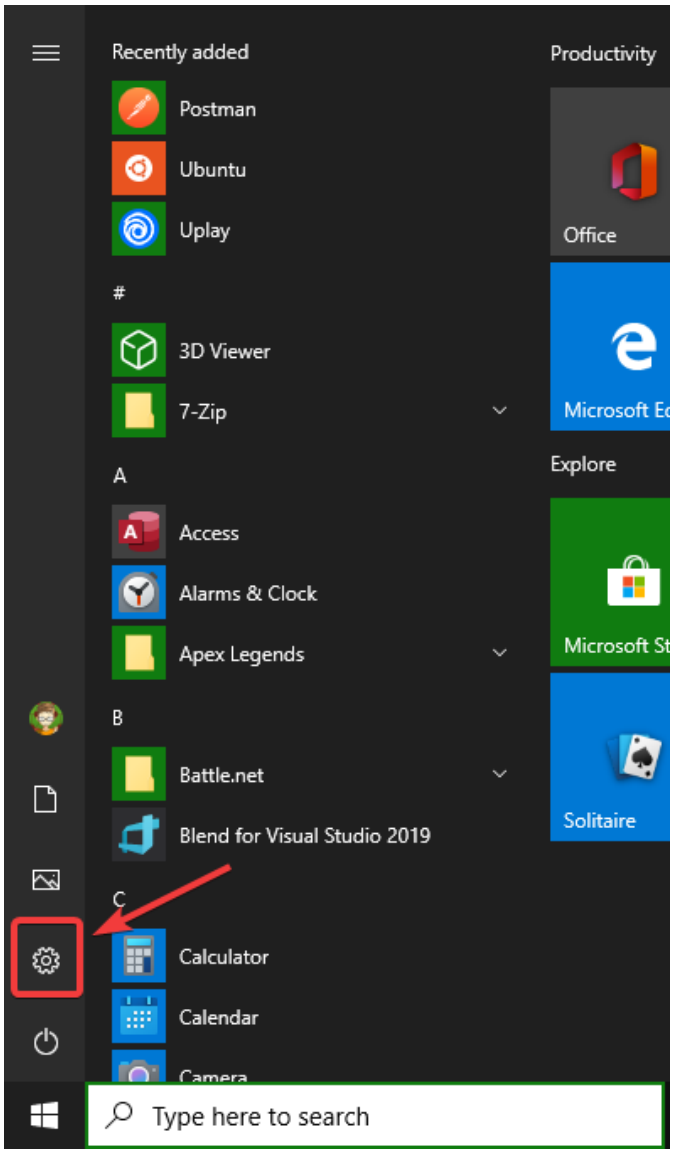
De nombreux outils de programmation utilisés dans FRC® besoin d'un accès au réseau pour diverses raisons. Selon la configuration exacte, le pare-feu Windows peut potentiellement interférer avec cet accès pour un ou plusieurs de ces programmes.

41.4.1 Désactiver le pare-feu Windows

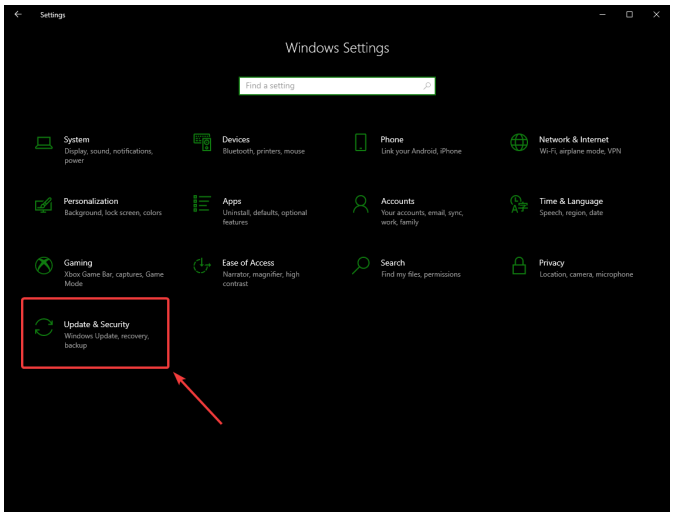
Important : La désactivation de votre pare-feu nécessite des privilèges d'administrateur sur le PC. Notez également que la désactivation du pare-feu n'est pas recommandée pour les ordinateurs qui se connectent à Internet.

La solution la plus simple consiste à désactiver le pare-feu Windows. Les équipes doivent être conscientes que cela rend le portable potentiellement plus vulnérable aux attaques de logiciels malveillants lors de la connexion à Internet.

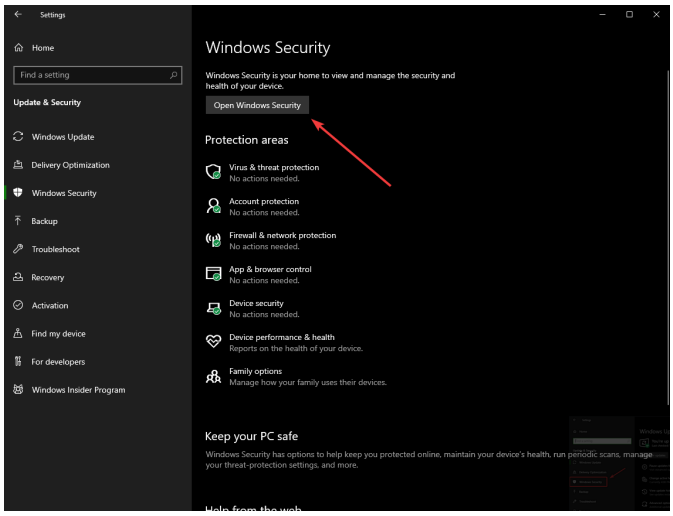
Cliquez sur *Start -> Settings*



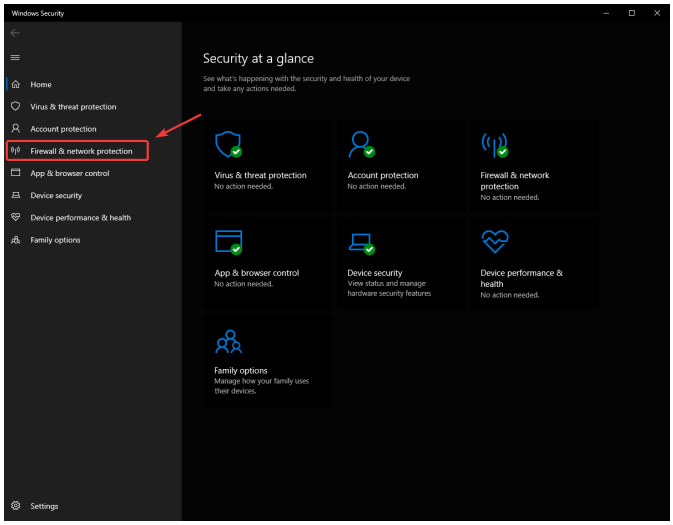
Cliquez sur *Update & Security*



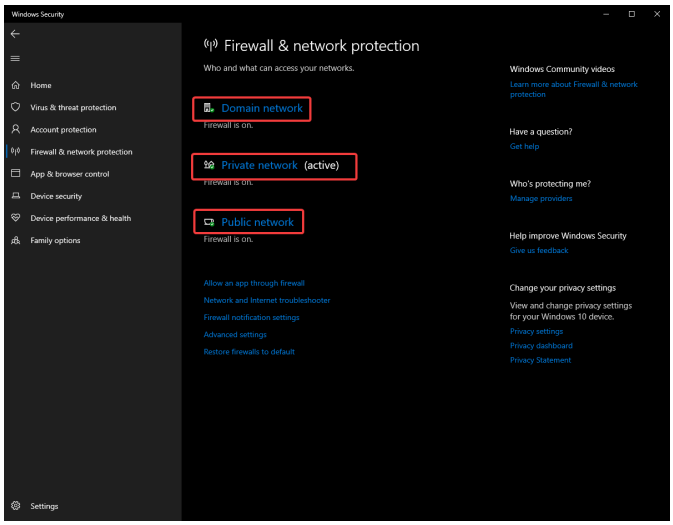
Dans le panneau de droite, sélectionner *Open Windows Security*



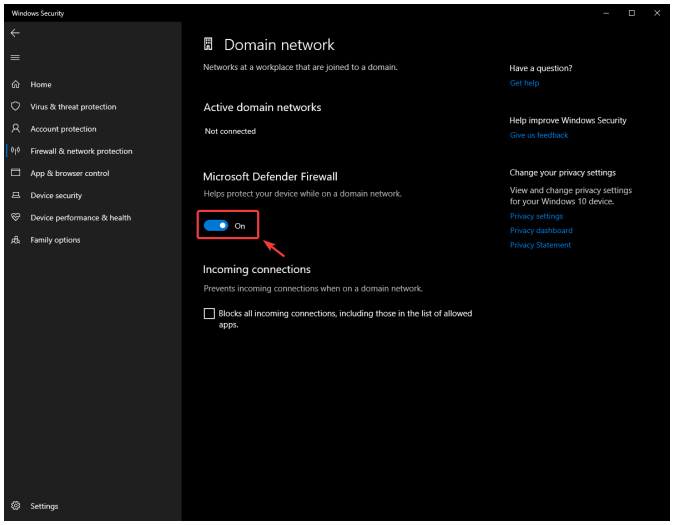
Dans le volet gauche, sélectionnez *Firewall and network protection*



Cliquez sur ** chacune ** des options mises en évidence



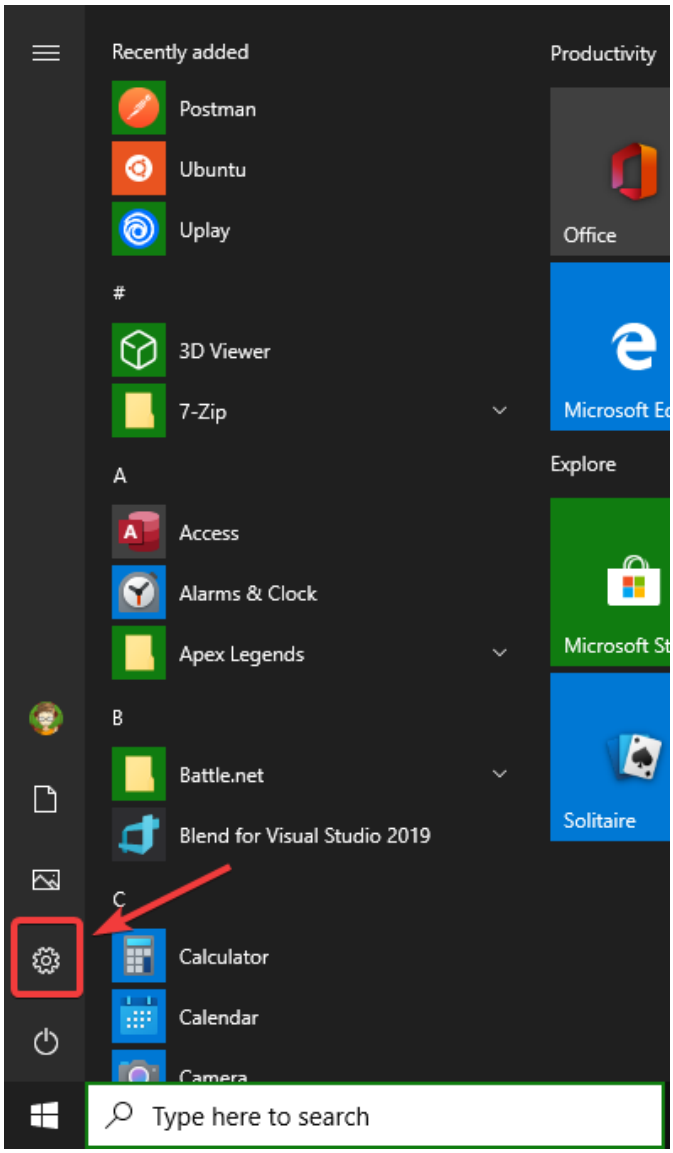
Cliquez ensuite sur le bouton ** ON ** pour le désactiver.



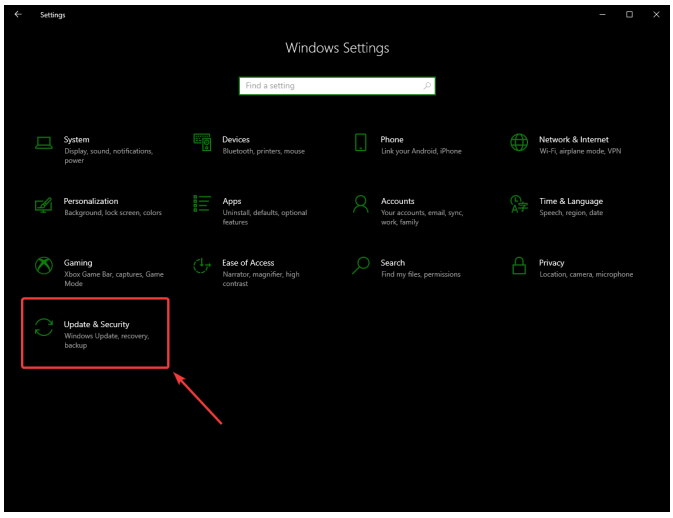
41.4.2 Liste blanche des applications

Vous pouvez également ajouter des exceptions au pare-feu pour tous les programmes FRC avec lesquels vous rencontrez des problèmes.

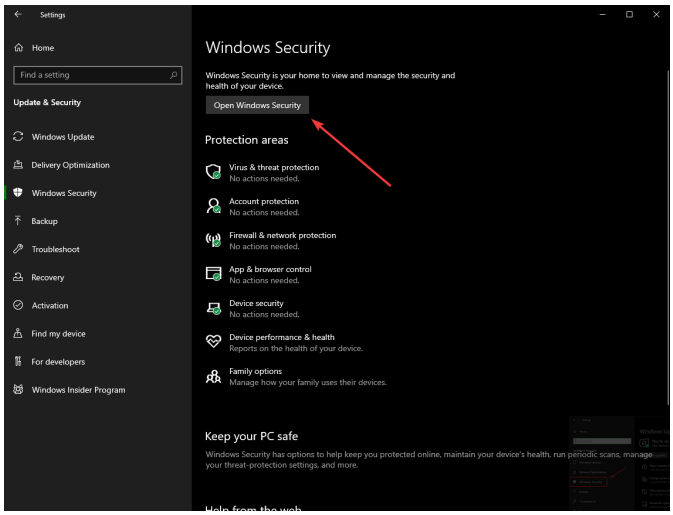
Cliquez sur *Start -> Settings*



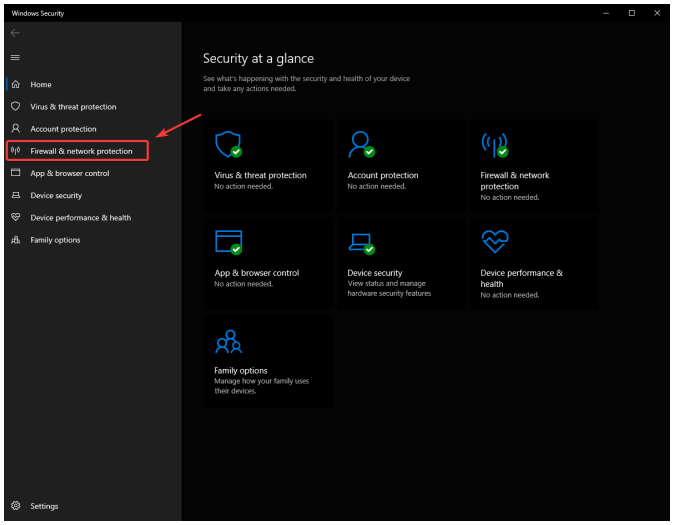
Cliquez sur *Update & Security*



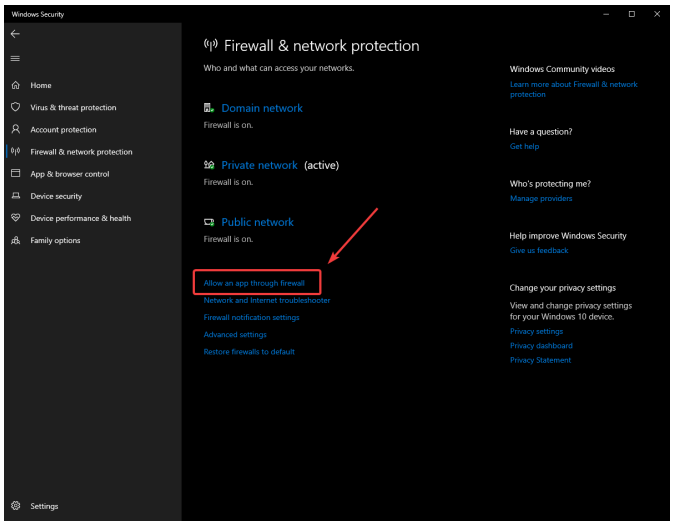
Dans le panneau de droite, sélectionner *Open Windows Security*



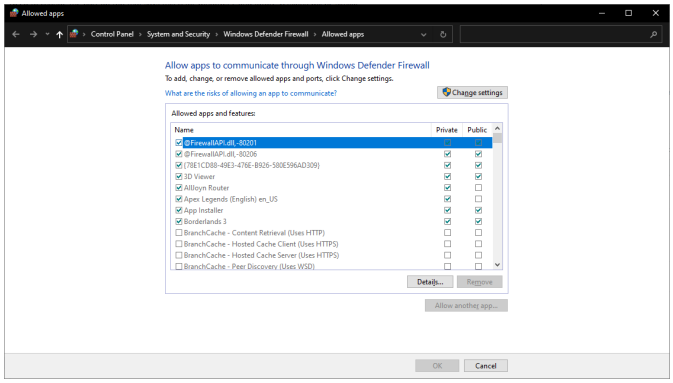
Dans le volet gauche, sélectionnez *Firewall and network protection*



En bas de la fenêtre, sélectionnez *Allow an app through firewall*



Pour chaque programme FRC avec lequel vous rencontrez un problème, assurez-vous qu'il apparaît dans la liste et qu'il est coché dans chacune des 3 colonnes. Si vous avez besoin de modifier un paramètre, vous devez cliquer sur le bouton *Change settings* en haut à droite avant de modifier les paramètres. Si le programme n'est pas du tout dans la liste, cliquez sur le bouton *Allow another program…* et accédez à l'emplacement du programme pour l'ajouter.



41.5 Mesure de l'utilisation de la bande passante

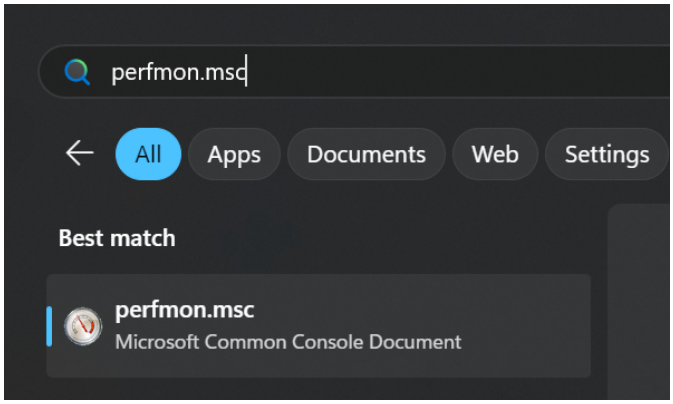
On the FRC® Field each team is allocated limited network bandwidth (see R704 in the 2024 manual). The FMS Whitepaper provides more information on determining the bandwidth usage of the Axis camera, but some teams may wish to measure their overall bandwidth consumption. This document details how to make that measurement.

Note : Les équipes peuvent simuler la limitation de la bande passante à la maison à l'aide de l'utilitaire de configuration de pont FRC avec la case à cocher de bande passante cochée.

41.5.1 Measuring Bandwidth Using the Performance Monitor (Win 7/10)

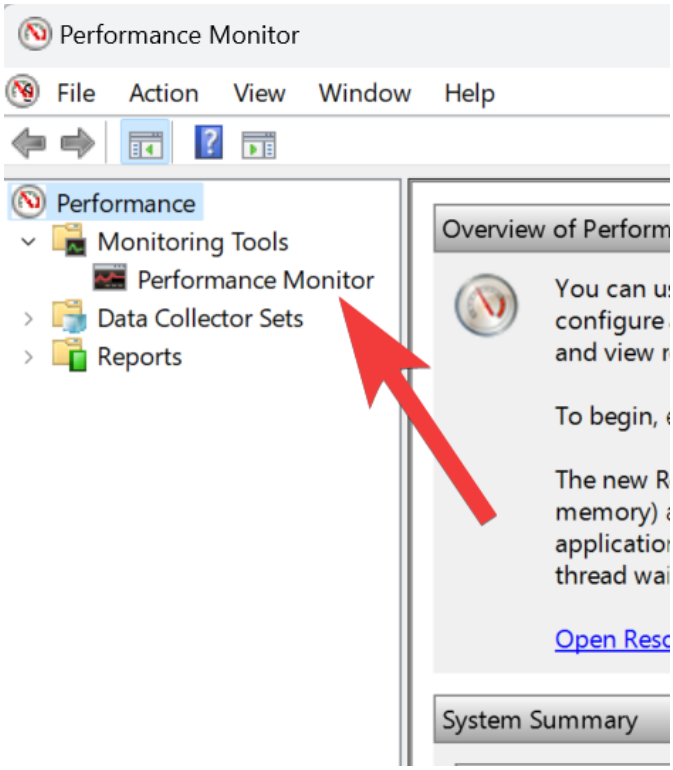
Windows contient un outil intégré appelé Performance Monitor qui peut être utilisé pour évaluer l'utilisation de la bande passante sur une interface réseau.

Démarrer *Performance Monitor*



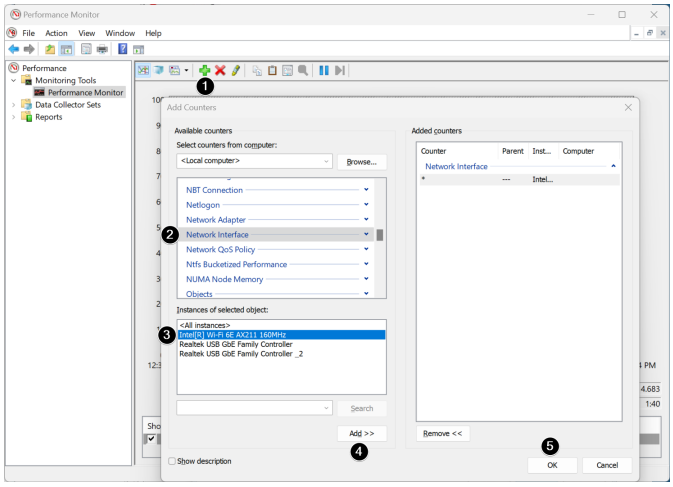
Open the Start menu and in the search box, type `perfmon.msc` and press Enter.

Ouvrir le moniteur en temps réel *Real-Time Monitor*



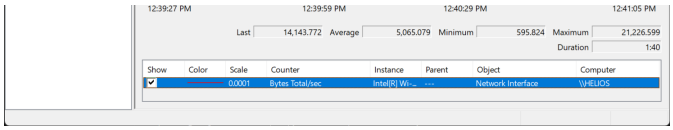
In the left pane, click *Performance Monitor* to display the real-time monitor.

Ajouter un compteur réseau



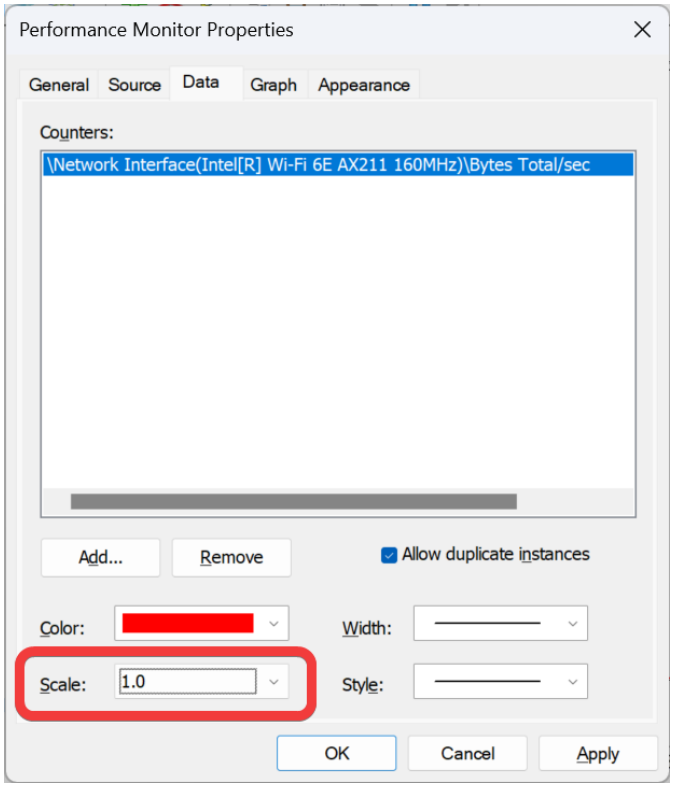
1. Cliquez sur le signe plus vert en haut de l'écran pour ajouter un compteur.
2. Dans le volet supérieur gauche, recherchez et cliquez sur `Network Interface` pour le sélectionner.
3. Dans le volet inférieur gauche, recherchez l'interface réseau souhaitée (ou *All instances* pour surveiller toutes les interfaces).
4. Click *Add >>* to add the counter to the right pane.
5. Click *OK* to add the counters to the graph.

Retirer des compteurs



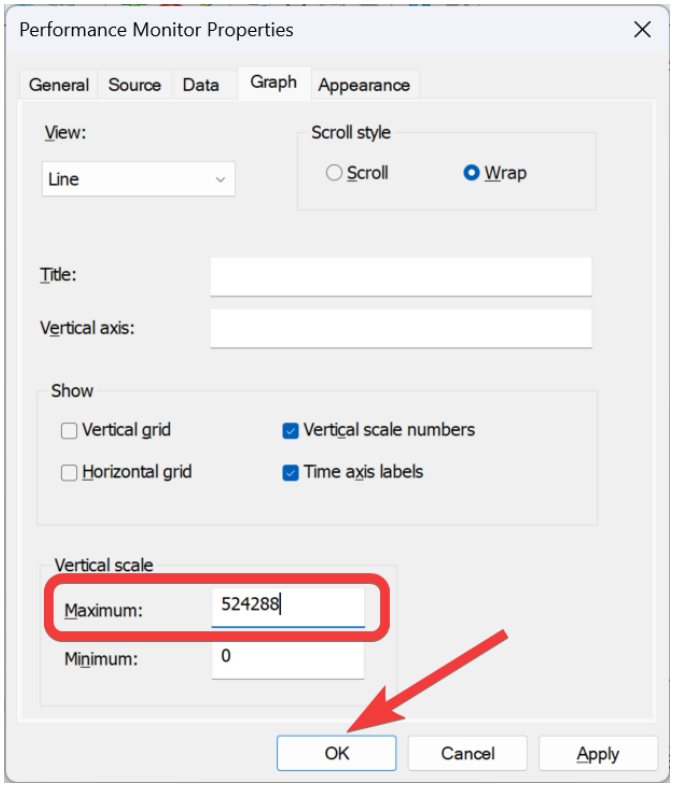
In the bottom pane, select each counter other than `Bytes Total/sec` and press the `Delete` key. The `Bytes Total/sec` entry should be the only entry remaining in the pane.

Configurer les propriétés des données



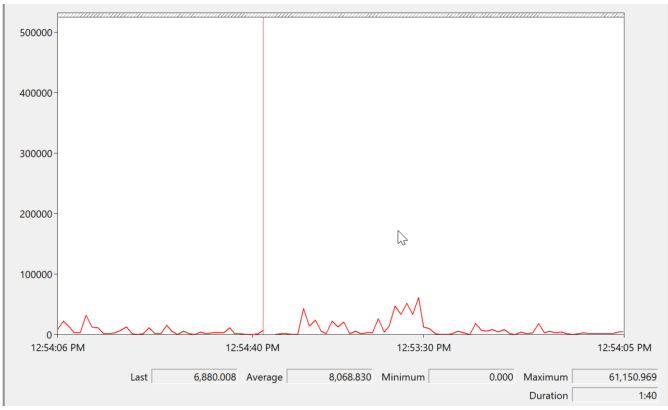
Press `Ctrl+Q` to bring up the Properties window. Click on the dropdown next to `Scale` and select `1.0`. Then click on the *Graph* tab.

Configurer le graphique



In the `Maximum Box` under `Vertical Scale` enter 524288 (this is 4 Megabits converted to Bytes). If desired, turn on the horizontal grid by checking the box. Then click *OK* to close the dialog.

Observer l'usage de la bande passante

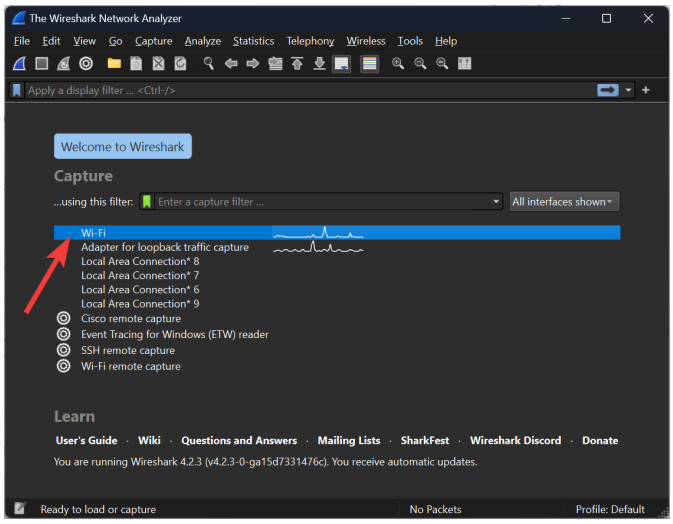


You may now connect to your robot as normal over the selected interface (if you haven't done so already). The graph will show the total bandwidth usage of the connection, with the bandwidth cap at the top of the graph. The Last, Average, Min and Max values are also displayed at the bottom of the graph. Note that these values are in Bytes/Second meaning the cap is 524,288. With just the Driver Station open you should see a flat line at ~100000 Bytes/Second.

41.5.2 Mesurer l'usage de la bande passante avec Wireshark

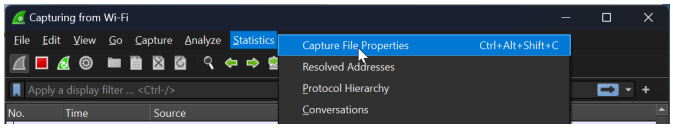
If you can not use performance monitor, you will need to install a 3rd party program to monitor bandwidth usage. One program that can be used for this purpose is Wireshark. Download and install the latest version of Wireshark for your version of Windows from this page. After installation is complete, locate and open Wireshark. Connect your computer to your robot, open the Driver Station and any Dashboard or custom programs you may be using.

Sélectionner l'interface et lancer la capture



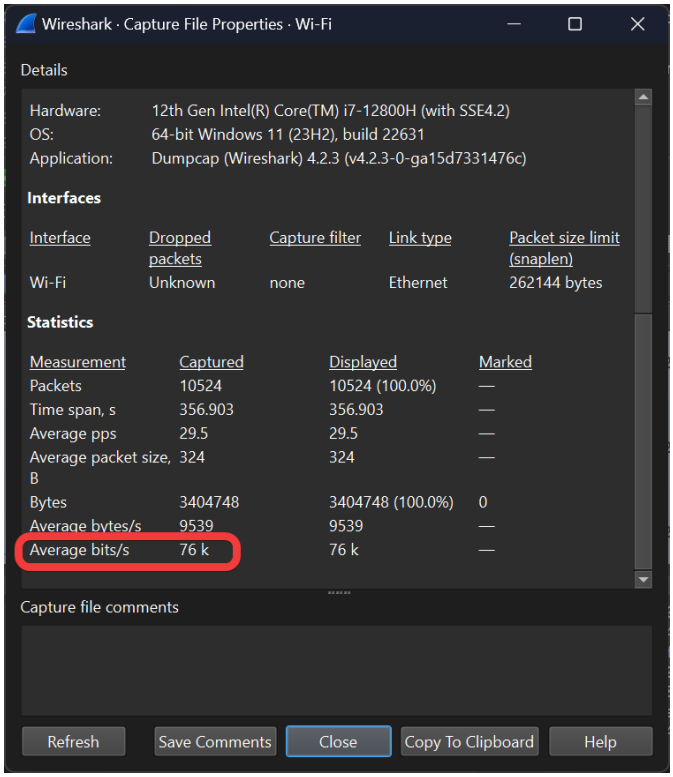
In the Wireshark program on the left side, double click the interface you are using to connect to the robot.

Open Capture File Properties



Let the capture run for at least 1 minute, then click *Statistics* then *Capture File Properties*.

Utilisation de la bande passante



Average bandwidth usage, in bits/second is displayed near the bottom of the window.

41.6 Modification à la radio OM5P-AC

Le contexte d'utilisation prévu pour la radio OM5P-AC ne le soumet pas aux mêmes chocs et forces qu'elle subit dans l'Environnemene FRC®. Si la radio est soumise à une pression importante sur le fond du boîtier, il est possible de provoquer un redémarrage de la radio en court-circuitant un écran métallique en dessous de la radio à quelques fils métalliques exposés sur le fond du boitier. Cet article détaille une modification à apporter à la radio pour empêcher ce scénario.

Avertissement : Il faut une pression importante appliquée au fond du boîtier pour provoquer un tel redémarrage. La plupart des problèmes de redémarrage de la radio FRC peuvent être attribués à l'alimentation électrique sous une forme ou une autre. Nous recommandons d'atténuer ce risque via un montage stratégique de la radio plutôt que d'ouvrir et de modifier la radio (et risquer d'endommager les composants internes délicats) :

- Évitez d'utiliser les fonctionnalités « mounting tab » en bas de la radio.
- Vous souhaiterez peut-être installer la radio pour permettre une certaine absorption des chocs. Un détail peut prévenir beaucoup. Le montage de la radio à l'aide de bandes auto-agrippantes ou sur une surface du robot à faible flexion (plastique ou tôle, etc.) peut réduire considérablement les forces subies par la radio.

41.6.1 Ouvrir le boîtier de la radio

Note : La radio OpenMesh OM5P-AC n'est pas conçue pour être réparable par l'utilisateur. Les utilisateurs effectuent cette modification à leurs propres risques. Assurez-vous de travailler lentement et soigneusement pour éviter d'endommager les composants internes tels que les câbles d'antenne radio.

Vis du boîtier

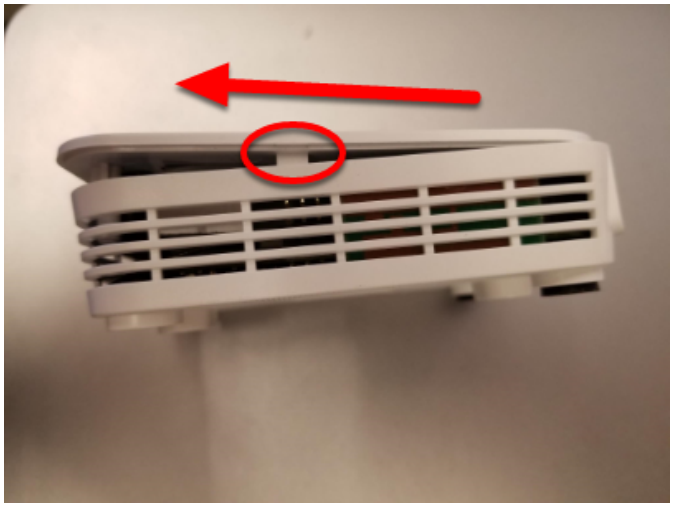
Repérez les deux pattes en caoutchouc à l'avant de la radio, puis écartez-les de la radio à l'aide de vos ongles, d'un petit tournevis plat, etc. À l'aide d'un petit tournevis cruciforme, retirez les deux vis sous les pattes.

Loquets latéraux

Il y a un petit loquet sur le couvercle de la radio près du milieu de chaque bord long (vous pouvez voir ces verrous plus clairement dans l'image suivante). Faites glisser un ongle ou un outil très mince le long de l'espace entre le couvercle et le boîtier d'avant en arrière vers le milieu de la radio, vous devriez entendre un petit bruit lorsque vous vous approchez du milieu de la radio. Répétez de l'autre côté (remarque : il n'est pas difficile de reverrouiller accidentellement le premier côté en faisant cela, assurez-vous que les deux côtés sont déverrouillés avant de continuer). Le couvercle de la radio doit maintenant être légèrement ouvert sur la face avant, comme indiqué dans l'image ci-dessus.

Retirez le couvercle

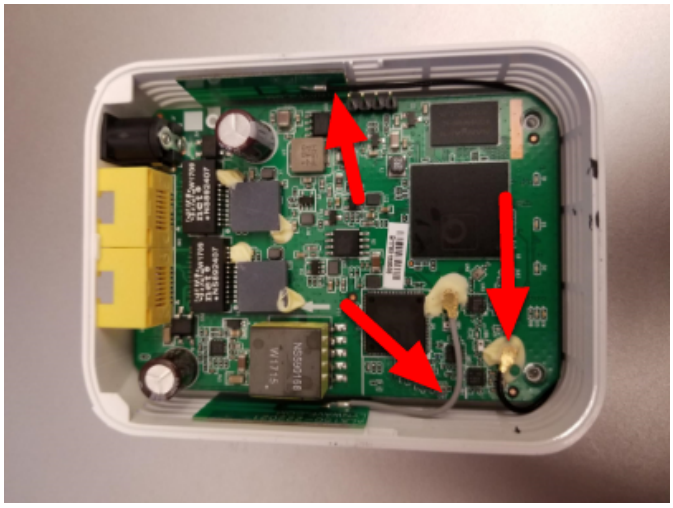
Avertissement : La carte peut coller au couvercle lorsque vous le retirez en raison des coussinets du dissipateur thermique. Regardez à travers les ouvertures de la radio lorsque vous retirez le couvercle pour voir si la carte suit le couvercle ; si c'est le cas, vous devrez peut-être insérer un petit outil pour maintenir la carte vers le bas et la séparer du couvercle. Nous recommandons un petit tournevis, inséré par le coin avant du côté du connecteur de l'alimentation, juste au-dessus du trou de vis. Défilez à l'image avec le couvercle retiré pour voir à quoi ressemble la carte à cet endroit.



Pour commencer à retirer le couvercle, faites-le glisser vers l'avant (en le soulevant légèrement) jusqu'à ce que les porte-vis atteignent l'avant du boîtier (vous devrez peut-être appliquer une pression sur les zones de verrouillage tout en faisant cela.

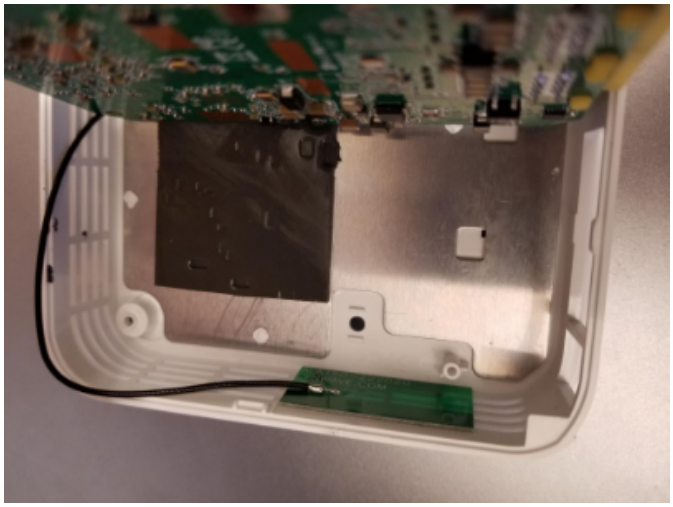
Ensuite, commencez à faire pivoter légèrement le couvercle du côté du connecteur d'alimentation comme indiqué, tout en continuant à soulever. Cela décrochera le couvercle du petit triangle visible dans le coin supérieur droit. Continuez à tourner légèrement dans cette direction tout en poussant le coin supérieur gauche vers le connecteur de l'alimentation (n'essayez pas de soulever plus loin à cette étape) pour défaire une jonction similaire dans le coin supérieur gauche. Soulevez ensuite complètement le couvercle du corps.

Retirez la carte



Avertissement : Notez la position des fils d'antenne montrés dans l'image ci-dessus. Ces fils et leurs connecteurs sont fragiles, veillez à ne pas les endommager lors des étapes suivantes.

Pour retirer la carte, nous vous recommandons de saisir avec vos doigts un ou les deux ports réseau (comme illustré) et de pousser vers l'intérieur (vers l'avant de la radio) et vers le haut jusqu'à ce que les ports réseau et le connecteur d'alimentation soient délogés du boîtier.



Inclinez la carte vers le haut (vers le câble d'antenne court et gris) pour exposer le blindage métallique en dessous.

Note : À cette étape, vous pouvez remarquer qu'il y a un petit bouton de réinitialisation sur le dessous de la carte et qui est plus grand que le trou dans le boîtier. Notez que le fait d'appuyer sur le bouton de réinitialisation alors que le micrologiciel FRC est installé n'a aucun effet et que percer le boîtier de la radio n'est pas une modification autorisée.

41.6.2 Appliquez du ruban adhésif

Appliquez un morceau de ruban électrique adhésif sur le blindage métallique dans la zone juste à l'intérieur des ouvertures du port réseau et du connecteur d'alimentation. Cela empêchera les fils exposés sur le dessous de la carte de court-circuiter sur cette plaque.

41.6.3 Réassemblez la radio

Procédez au réassemblage de la radio en inversant les instructions de démontage.

- Déposez la carte vers le bas, en vous assurant qu'elle s'aligne avec les trous de vis près de l'avant et qu'elle repose correctement.
- Faites glisser le couvercle sur la butée de retenue arrière gauche en le déplaçant de droite à gauche. Prenez soin du condensateur dans cette zone.
- Faites pivoter le couvercle, appuyez vers le bas et faites glisser la butée de retenue arrière droite.
- Appuyez fermement sur l'avant / le milieu du couvercle pour asseoir les loquets.
- Replacez les 2 vis des pattes avant.
- Replacez les pattes avant.

42

Utilitaires réseau

42.1 Transfert de port

La classe PortForwarder offre un moyen facile de transférer les ports locaux vers un autre hôte/port. Ceci est utile pour fournir un moyen d'accéder aux périphériques connectés par Ethernet à partir d'un ordinateur connecté au port USB du roboRIO. Cette classe agit comme un transitaire de port TCP brut, ce qui signifie que vous pouvez transférer des connexions telles que SSH.

42.1.1 Transférer un port distant

Souvent, les équipes souhaitent se connecter directement au roboRIO pour contrôler leur robot. La classe PortForwarding (Java, C++) peut être utilisée pour transférer la connection du Raspberry Pi pour l'utilisation en port transféré durant ce temps. La classe PortForwarding établit un pont entre l'appareil distant et le client. Pour transférer un port en Java, faites simplement `PortForwarder.add(int port, String remoteName, int remotePort)`.

JAVA

```
@Override
public void robotInit() {
   PortForwarder.add(8888, "wpilibpi.local", 80);
}
```

C++

```
void Robot::RobotInit {
   wpi::PortForwarder::GetInstance().Add(8888, "wpilibpi.local", 80);
}
```

PYTHON

```
wpinet.PortForwarder.getInstance().add(8888, "wpilibpi.local", 80)
```

Important : Vous **ne pouvez pas** utiliser un port inférieur à 1024 comme port redirigé local. Il est également important de noter que vous ne pouvez **pas** utiliser des URL complètes (`http://wpilibpi.local`) et ne devez utiliser que des adresses IP ou des noms DNS.

42.1.2 Supprimer un port transféré

Pour arrêter le transfert vers un port spécifié, il suffit d'appeler `remove(int port)` avec comme port le numéro de port. Si vous appelez `remove()` sur un port qui n'est pas transmis, il ne se passera rien.

JAVA

```
@Override
public void robotInit() {
   PortForwarder.remove(8888);
}
```

C++

```
void Robot::RobotInit {
   wpi::PortForwarder::GetInstance().Remove(8888);
}
```

PYTHON

```
wpinet.PortForwarder.getInstance().remove(8888)
```

43

Contribuer au projet frc-docs

43.1 Instructions concernant les contributions

Welcome to the contribution guidelines for the frc-docs project. If you are unfamiliar to writing in the reStructuredText format, please read up on it here.

Important : *FIRST*® conserve tous les droits sur la documentation et les images fournies. Le crédit pour les articles/mises à jour sera accessible dans GitHub commit history.

43.1.1 Mission

La mission de WPILib est de permettre *aux équipes de la Compétition de robotique FIRST* de se concentrer à rédiger du code lié au jeu de la saison plutôt que de se buter à des détails techniques - « élever le plancher sans abaisser le plafond ». Nous voulons permettre aux équipes avec peu de connaissances en programmation ou sans l'expertise de mentors de performer autant que possible, sans gêner les prouesses d'équipes avec plus de ressources avancées en matière de programmation. Les ressources concernant les composantes du système de contrôle du Kit de pièces se trouvent directement dans la bibliothèque. Nous tentons également de pourvoir équitablement les caractéristiques principales de chaque langage (Java, C++, et LabVIEW de NI), de telle sorte qu'il n'y ait pas de désavantage pour les équipes à choisir un langage plus qu'un autre.

Ces documents constituent un apprentissage de base pour toutes les équipes en Compétition de robotique *FIRST*. Toute contribution au projet doit respecter les principes suivants.

- Une documentation issue de la communauté. Les sources sont disponibles publiquement et la communauté peut y contribuer.
- Une documentation structurée, mise en page adéquatement et concise. La documentation doit être concise et facile à lire, tant du point de vue rédaction que diffusion.
- Pertinence. La documentation doit porter sur la Compétition de robotique *FIRST* .

Consultez *Conventions de style* concernant la mise en forme des documents.

43.1.2 Processus de révision

frc-docs utilise un processus de révision spécial pour la gestion du site principal `/stable/` et du site de développement `/latest/`. Ce flux est détaillé ci-dessous.

Pendant la saison :
- Validation (commit) effectuée sur la branche `main`
  - Mises à jour `/stable/` et `/latest/` sur le site web

À la fin de la saison :
- Le référentiel est étiqueté avec l'année, à des fins d'archivage

Hors-saison :
- La branche `/stable/` est verrouillée sur le dernier commit de la saison
- Validation (commit) effectuée sur la branche `main`
  - Seules les mises à jour `/latest/` sur le site de documentation

43.1.3 Création d'un PR

Les PR doivent être envoyés au référentiel frc-docs repo on GitHub. sur GitHub. Ils doivent pointer vers la branche `principale` et non celle `stable`.

43.1.4 Création de nouveaux contenus

Merci d'avoir contribué au projet frc-docs Il y a deux ou trois choses que vous devriez savoir avant de commencer !

Où placer les articles ?

L'emplacement pour de nouveaux articles peut être un sujet très discutable. Les articles autonomes qui entrent bien dans une catégorie déjà soumise devraient être placés dans la catégorie des sujets mentionnés (la documentation sur un sujet relatif à la simulation devrait être placée dans la section simulation). Cependant, les choses peuvent devenir assez compliquées quand un article combine ou fait référence à deux sections existantes distinctes. Dans ce cas, nous conseillons à l'auteur d'ouvrir une question sur le dépôt pour faire en sorte que l'on puisse en discuter avant d'ouvrir le PR.

Note : Toutes les nouvelles publications seront soumises à un processus de révision avant d'être intégrées. Le processus de révision est assuré par des membres de l'équipe WPILib. Les nouvelles publications doivent porter sur les équipements ou les logiciels officiellement reconnus par *FIRST*. De la documentation portant sur des bibliothèques ou des capteurs non officiels « ne sera pas » acceptée. Le processus peut prendre un certain temps ; soyez patient.

Où placer les sections ?

Les sections sont assez délicates, car elles contiennent une grande quantité de contenu. Nous conseillons à l'auteur de signaler le problème afin de recueillir des opinions avant d'ouvrir une RP.

Relier d'autres articles

Dans le cas où l'article fait référence au contenu décrit dans un autre article, l'auteur doit faire de son mieux pour établir un lien vers cet article dans la première référence.

Imaginez que nous avons le contenu suivant dans un tutoriel relatif à la base pilotable :

```
Teams may often need to test their robot code outside of a competition.␣
↪:ref:`Simulation <link-to-simulation:simulation>` is a means to achieve this.␣
↪Simulation offers teams a way to unit test and test their robot code without ever␣
↪needing a robot.
```

Notez comment seule la première instance de Simulation est liée. C'est cette structure que l'auteur doit suivre. Il y a des moments où un article est lié a différents sujets dans son contenu. Si vous référez les différents types de contenu dans l'article, vous devez lier chaque nouvelle référence une fois (sauf dans les situations où l'auteur l'a jugé autrement approprié).

43.2 Conventions de style

Ce document contient les différentes directives spécifiques à RST/Sphinx relatives au projet frc-docs. Pour les directives relatives aux différents projets codés selon la WPILib, voir the WPILib GitHub

43.2.1 Noms de fichier

N'utilisez que des caractères alphanumériques minuscules et le symbole de soustraction - (moins).

Pour les documents qui porteront le même titre pour l'équipement et la programmation, ajoutez « Hardware » ou « Software » à la fin du nom du document. IE, `ultrasonics-hardware.rst`

Utilisez l'extension `.rst` comme terminaison de nom de fichier.

Note : Si vous rencontrez des problèmes pour éditer des fichiers avec l'extension `.rst`, l'éditeur de texte recommandé est VS Code avec l'extension rST.

43.2.2 Texte

Tout le contenu textuel devrait apparaître sur une seule ligne. Pour faciliter la lecture, utilisez la fonction de retour à la ligne de votre éditeur de texte.

Utilisez ces dénominations pour ces termes :
- roboRIO (pas RoboRIO, roboRio, ni RoboRio)
- LabVIEW (pas labview ni LabView)
- Visual Studio Code (VS Code) (pas vscode, VScode, vs code, etc.)
- macOS (pas Mac OS, Mac OSX, Mac OS X, Mac, Mac OS, etc.)
- GitHub (pas github, Github, etc.)
- PowerShell (pas powershell, Powershell, etc.)
- Linux (pas linux)
- Java (pas java)

Utilisez le jeu de caractères ASCII pour l'anglais. Pour les caractères spéciaux (e.g. symboles grecs) utilisez les jeux standards de caractères.

Utilisez `.. math::` pour les équations et `:math:` pour les équations intégrées. Une liste d'astuces LaTeX pour les équations est disponible ici

Utilisez des chaînes de caractères pour les noms de fichier, les fonctions et les noms de variable.

L'utilisation des marques enregistrées *FIRST*® et FRC® doit respecter la Politique suivante. Spécifiquement, lorsque possible (i.e. non imbriquée dans une balise ou dans le titre d'un document), la première occurrence de la marque de commerce doit porter le symbole ® et toutes les mentions de *FIRST* doivent être en italiques. Le symbole ® peut être ajouté en insérant `.. include:: <isonum.txt>` au début du document, puis en utilisant `*FIRST*\ |reg|` ou `FRC\ |reg|`.

Les termes couramment utilisés doivent être ajoutés au *Glossaire FRC*. Vous pouvez référencer des éléments dans le glossaire en utilisant ``:term:`deprecated```.

43.2.3 Espaces

Indentation

Une indentation doit *toujours* correspondre au niveau précédent d'indentation *sauf* si vous créez un nouveau bloc de contenu.

L'indentation d'instructions de contenu sur une nouvelle ligne `.. toctree::` doit être de *3* espacements.

Lignes vierges

Il doit y avoir 1 ligne vierge séparant des blocs de texte simples et les titres de section. Il *doit* y avoir 1 ligne vierge séparant des blocs de texte *et* des directives de contenu.

Espacement intérieur

Utilisez un espacement entre les phrases.

43.2.4 Entêtes

Les entêtes doivent présenter la structure suivante. Le soulignement d'entête doit correspondre au nombre de caractères de l'entête lui-même.

1. = pour les titres de document. À *ne pas* utiliser plus d'*une fois* par article.
2. - pour les sections de document
3. ^ pour des sous-sections de document
4. ~ pour des sous-sous-sections de document
5. Si vous devez subdiviser la structure davantage, vous êtes mal engagé.

Utilisez la casse de titre pour les entêtes.

43.2.5 Listes

Les listes doivent comporter une nouvelle ligne entre chaque niveau d'indentation. Le niveau supérieur doit avoir une indentation de `0`, et les sous-listes subséquentes doivent avoir une indentation débutant au premier caractère de l'indentation précédente.

```
- Block one
- Block two
- Block three

  - Sub 1
  - Sub 2

- Block four
```

43.2.6 Blocs de code

Tous les blocs de code doivent être dans un langage spécifique.

1. Exception : Du contenu dont la mise en page doit être préservée et sans langage. Utilisez plutôt `text`.

Respectez le Guide de formatage WPILib pour des modèles de codage C++ et Java. Par exemple, utilisez deux espacements pour les indentations en C++ et Java.

43.2.7 RLI (Inclusion Littéralle à distance)

Lorsque cela est possible, au lieu d'utiliser des blocs de code, un RLI doit être utilisé. Cela extrait les lignes de code directement de GitHub, le plus souvent en utilisant les exemples de programmes. Cela maintient automatiquement le code à jour avec toutes les modifications apportées. Le format d'un RLI est :

```
.. rli:: https://raw.githubusercontent.com/wpilibsuite/allwpilib/v2024.3.2/
↪wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/ramsetecontroller/
↪Robot.java
   :language: java
   :lines: 44-61
   :linenos:
   :lineno-start: 44

.. rli:: https://raw.githubusercontent.com/wpilibsuite/allwpilib/v2024.3.2/
↪wpilibcExamples/src/main/cpp/examples/RamseteController/cpp/Robot.cpp
   :language: c++
   :lines: 18-30
   :linenos:
   :lineno-start: 18
```

Assurez-vous de créer un lien vers la version brute du fichier sur GitHub, il y a un bouton Raw pratique dans le coin supérieur droit de la page.

43.2.8 Onglets

Pour créer des onglets de code dans un article, vous pouvez utiliser la directive `.. tab-set-code::`. Vous pouvez utiliser les directives `code-block` et `rli` à l'intérieur. Le format est :

```
.. tab-set-code::

   .. rli:: https://raw.githubusercontent.com/wpilibsuite/allwpilib/v2024.3.2/
↪wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/ramsetecontroller/
↪Robot.java
      :language: java
      :lines: 44-61
      :linenos:
      :lineno-start: 44


   .. code-block:: c++
      // Start the timer.
      m_timer.Start();

      // Send Field2d to SmartDashboard.
      frc::SmartDashboard::PutData(&m_field);

      // Reset the drivetrain's odometry to the starting pose of the trajectory.
      m_drive.ResetOdometry(m_trajectory.InitialPose());

      // Send our generated trajectory to Field2d.
      m_field.GetObject("traj")->SetTrajectory(m_trajectory);
```

Si vous avez besoin d'utiliser plus d'un onglet par langue, plusieurs RLI par langue ou des onglets de texte, vous pouvez utiliser la directive `.. tab-set::` et `.. tab-item::`. Le format est le suivant :

```
.. tab-set::

   .. tab-item:: Title
```

(suite sur la page suivante)

(suite de la page précédente)

```
    :sync: sync-id

       Content
```

Cet exemple utilise l'argument de synchronisation pour permettre à tous les onglets avec la même clé d'être synchronisés ensemble. Cela signifie que lorsque vous cliquez sur un onglet, tous les onglets avec la même clé s'ouvriront.

Si vous avez un mélange de directives « tab-set » et « tab-set-code » sur une page, vous pouvez les synchroniser en définissant l'identifiant de synchronisation sur les directives « tab-item » avec la valeur « tabcode-LANGUAGE ». Par exemple, un onglet Java aurait un identifiant de synchronisation de « tabcode-java ».

43.2.9 Avertissements

Le libellé des avertissements (liste ici) doit être sur la même ligne que l'avertissement lui-même. Il existe cependant des exceptions à cette règle, quand il y a plusieurs sections de contenu dans un même avertissement. Généralement, il n'est pas recommandé d'avoir plusieurs sections de contenu dans un avertissement.

Usage correcte

```
.. warning:: This is a warning!
```

Usage INCORRECTE

```
.. warning::
   This is a warning!
```

43.2.10 Liens

Liens internes

Les liens internes seront générés automatiquement en se basant sur le nom de fichier ReStructuredText et sur le titre de section.

Par exemple, il y a plusieurs manières de lier les sections et les documents.

Utilisez ce format pour référencer une section de document. Vous devez utiliser le chemin absolu du document. `:ref:`docs/software/hardware-apis/sensors/ultrasonics-software:Analog ultrasonics`` est rendu comme *Capteurs à ultrasons analogiques*.

Utilisez ce format pour référencer une section du même document. Notez le trait de soulignement unique. `` `Images`_ `` est rendu comme *Images*.

Utilisez ce format pour référencer le niveau supérieur d'un document. Vous pouvez utiliser des chemins relatifs `:doc:`build-instructions`` est rendu comme *Instructions de compilaton* Ou pour utiliser des chemins absolus, mettez une barre oblique au début du chemin `:doc:`/docs/software/hardware-apis/sensors/ultrasonics-software`` est rendu comme *Ultrasons - Partie logicielle*. Notez que le texte rendu est le titre de la section principale de la page cible, quel que soit le nom du fichier cible.

Lorsque vous utilisez `:ref:` ou `:doc:`, vous pouvez personnaliser le texte affiché en entourant le lien réel avec des crochets angulaires `<>` et en ajoutant le texte personnalisé entre le premier accent grave (backtick) ```` ` ```` et le premier crochet angulaire `<`. Par exemple ``:ref:`texte personnalisé <docs/software/hardware-apis/sensors/ultrasonics-software:Ultrasoniques analogiques>`` est rendu en texte personnalisé.

Liens externes

Il est préférable de formater les liens externes comme des hyperliens anonymes. À noter, les **deux** soulignements suivant le texte. Dans la situation où un seul soulignement est utilisé, des erreurs peuvent se produire lors de la compilation du document.

```
Hi there, `this is a link <https://example.com>`__ and it's pretty cool!
```

Cependant, dans les cas où le même lien doit être référencé à plusieurs reprises, la syntaxe suivante est acceptée.

```
Hi there, `this is a link`_ and it's pretty cool!

..  _this is a link: https://example.com
```

43.2.11 Images

Les images doivent être introduites avec `1` nouvelle ligne séparant le contenu et l'instruction.

La taille de chaque image (même au format vectoriel) ne doit pas dépasser `500` kilo-octets. Veuillez réduire la résolution d'image et utiliser un algorithme de compression efficace.

```
.. image:: images/my-article/my-image.png
   :alt: Always add alt text here describing the image.
```

Fichiers image

Les fichiers d'images doivent être stockés dans le répertoire du document, sous-répertoire `document-name/images`.

They should follow the naming scheme of `short-description.png`, where the name of the image is a short description of what the image shows. This should be less than `24` characters.

They should be of the `.png` or `.jpg` image extension. `.gif` is unacceptable due to storage and accessibility concerns.

Note : Accessibility is important! Images should be marked with a `:alt:` directive.

```
.. image:: images/my-document/my-image.png
   :alt: An example image
```

Images vectorielles

Les fichiers SVG sont supportés via l'extension Sphinx `svg2pdfconverter`.

Utilisez-les comme vous le feriez de tout autre format d'image.

Note : Assurez-vous que les images incorporées au document vecteur ne le gonflent pas pour dépasser la limite de 500 Ko.

```
.. image:: images/my-document/my-image.svg
   :alt: Always add alt text here describing the image.
```

Croquis Draw.io

Les croquis ou diagrammes Draw.io (ou diagrams.net) sont supportés en utilisant des fichiers `svg` avec métadonnées `.drawio` incorporées, ce qui permet au fichier `svg` d'être la source du croquis et d'être affiché comme un fichier vectoriel régulier.

Utilisez-les simplement comme vous le feriez pour toute autre image vectorielle ou toute autre format d'image.

```
.. image:: diagrams/my-document/diagram-1.drawio.svg
   :alt: Always add alt text here describing the image.
```

Fichiers Draw.io

Les fichiers Draw.io suivent presque le même schéma de nomenclature que les images normales. Pour garder la trace des fichiers contenant les métadonnées `.drawio` intégrées, ajoutez un `.drawio` à la fin du nom de fichier mais avant l'extension, ce qui signifie que le nom du fichier doit être `document-title-1.drawio.svg` etc. De plus, les croquis ou diagrammes doivent être stockés dans le répertoire du document, dans un sous-dossier nommé `diagrams`.

Pour les détails concernant la sauvegarde d'un croquis ou diagramme en tant que `.svg` avec des métadonnées, consultez *Instructions de sauvegarde pour Draw.io*.

Avertissement : Ne modifiez aucun fichier du dossier `diagrams` ou se terminant par `.drawio.svg` avec un programme autre que draw.io, sinon vous risquez d'endommager les métadonnées du fichier, ce qui le rendrait inutilisable.

43.2.12 Extensions de fichiers

Les extensions de fichiers doivent utiliser la mise en forme du code. Par exemple, utilisez :

```
``.png``
```

Au lieu de :

```
.png
".png"
"``.png``"
```

43.2.13 Table des matières (TOC)

Chaque catégorie doit contenir un `index.rst`. Ce fichier d'index doit avoir une profondeur `maxdepth` de 1. Les sous-catégories sont acceptables, avec une profondeur maximale `maxdepth` de 1.

Le fichier `index.rst` de catégorie peut être ajouté au fichier index de la racine dans `source/index.rst`.

43.2.14 Exemples

```
Title
=====
This is an example article

.. code-block:: java

   System.out.println("Hello World");

Section
-------
This is a section!
```

43.2.15 Notes importantes !

Cette liste n'est pas exhaustive et les administrateurs se réservent le droit de la modifier. Les modifications seront identifiées dans ce document.

43.3 Instructions de compilaton

Ce document présente l'information nécessaire pour compiler les versions HTML, PDF, et EPUB du site frc-docs. Le projet frc-docs utilise Sphinx comme générateur de documentation. Ce document suppose également que vous avez une connaissance de base de Git et des commandes de la console.

43.3.1 Prérequis

Assurez-vous que Git est installé et que la bibliothèque frc-docs est dupliquée en utilisant `git clone https://github.com/wpilibsuite/frc-docs.git`.

Éditeurs de texte / IDE

Pour le développement, nous recommandons l'utilisation de VS Code avec l'extension reStructuredText. Cela dit, n'importe quel éditeur de texte peut être utilisé.

Par défaut, l'extension reStructuredText active la vérification lint avec toutes les fonctionnalités doc8 activées. Comme frc-docs ne suit pas la vérification lint de longueur de ligne, ajoutez ce qui suit à votre code VS `settings.json` pour désactiver la vérification de la longueur de ligne.

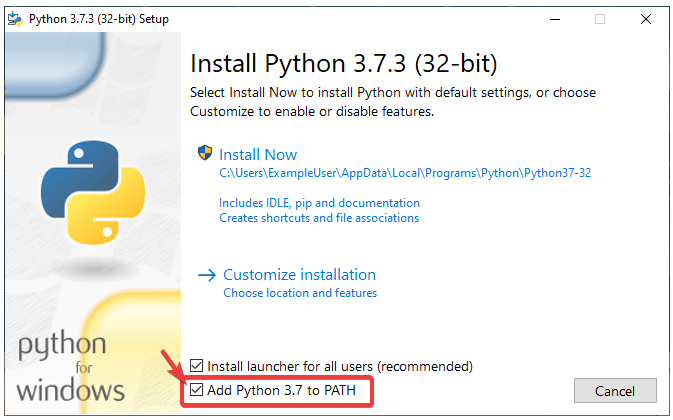
```
"restructuredtext.linter.doc8.extraArgs": [
   "--ignore D001"
]
```

Windows

Note : MikTeX et `rsvg-convert` ne sont pas requis pour la construction HTML, ils ne sont requis que pour les constructions Windows PDF.

— Python 3.9
— Perl
— MiKTeX (Seulement requis pour les constructions PDF)
— rsvg-convert (seulement requis pour la création de fichiers pdf)

Assurez-vous que Python fait partie de votre chaîne Path en sélectionnant l'option **Add Python to PATH** lors de l'installation de Python.



Une fois Python installée, ouvrez Powershell, puis localisez le répertoire frc-docs. Exécutez la commande suivante : `pip install -r source/requirements.txt`

Installez les modules MikTex manquants en accédant au répertoire frc-docs, puis en exécutant la commande suivante dans Powershell : `miktex --verbose packages require --package-id-file miktex-packages.txt`

Linux (Ubuntu)

```
$ sudo apt update
$ sudo apt install python3 python3-pip
$ python3 -m pip install -U pip setuptools wheel
$ python3 -m pip install -r source/requirements.txt
$ sudo apt install -y texlive-latex-recommended texlive-fonts-recommended texlive-
↪latex-extra latexmk texlive-lang-greek texlive-luatex texlive-xetex texlive-fonts-
↪extra dvipng librsvg2-bin
```

43.3.2 Compilation

Ouvrez une console ou une fenêtre Powershell Window et localisez le répertoire frc-docs qui a été dupliqué.

```
PS > cd "%USERPROFILE%\Documents"
PS C:\Users\Example\Documents> git clone https://github.com/wpilibsuite/frc-docs.git
Cloning into 'frc-docs'...
remote: Enumerating objects: 217, done.
remote: Counting objects: 100% (217/217), done.
remote: Compressing objects: 100% (196/196), done.
remote: Total 2587 (delta 50), reused 68 (delta 21), pack-reused 2370
Receiving objects: 100% (2587/2587), 42.68MiB | 20.32 MiB/s, done.
Receiving deltas: 100% (1138/1138), done/
PS C:\Users\Example\Documents> cd frc-docs
PS C:\Users\Example\Documents\frc-docs>
```

Vérificateur syntaxique

Note : Le vérificateur syntaxique ne vérifiera pas les fins de phrases sur Windows à cause d'un bug lié aux fins de phrases. Consultez ce problème pour plus d'information.

Il est suggéré de vérifier les changements apportés avec le vérificateur syntaxique. Sinon, la compilation automatique **échouera**. Pour vérifier, exécutez `.\make lint`

Vérificateur de lien

Le vérificateur de lien s'assure que tous les liens dans la documentation sont correctement définis. Sinon, la compilation automatique **échouera**. Pour vérifier, exécutez `.\make linkcheck`

Vérificateur de taille des images

Exécutez `.\make sizecheck` pour vérifier que toutes les images font moins de 500ko. Les excès feront *échouer* l'intégration continue. Des exceptions seront permises au cas par cas et elles seront ajoutées à la liste `IMAGE_SIZE_EXCLUSIONS` du fichier de configuration.

Rediriger la vérification

Les fichiers qui ont été déplacés ou renommés doivent avoir leur nouvel emplacement (ou remplacé par 404) dans le fichier `redirects.txt` dans la `source`.

Le rédacteur de redirection ajoutera automatiquement les fichiers renommés ou déplacés au fichier de redirection. Run `.\make rediraffewritediff`.

Note : si un fichier est à la fois déplacé et substantiellement modifié, le rédacteur de la redirection ne l'ajoutera pas au fichier `redirects.txt`, et `redirects.txt` aura besoin d'être mis à jour manuellement.

Le vérificateur de redirection s'assure qu'il existe des redirections valides pour tous les fichiers. Cela ** fera ** échouer le buildbot s'il ne réussit pas. Pour vérifier, exécutez `.\make rediraffecheckdiff` pour vérifier que tous les fichiers sont redirigés. De plus, une version HTML peut devoir être exécutée pour garantir que tous les fichiers sont correctement redirigés.

Compilation en HTML

Exécutez `.\make html` pour générer le contenu en HTML. Le contenu HTML est situé dans le répertoire `build/html` à la racine de la bibliothèque.

43.3.3 Compilation en PDF

Avertissement : Notez que la compilation PDF sous Windows peut causer la distorsion d'image au format SVG. Ceci est dû au manque de ressources librsvg2-bin sur Windows.

Exécutez `.\make latexpdf` pour générer le contenu en PDF. Le contenu PDF est situé dans le répertoire `build/latex` à la racine de la bibliothèque.

43.3.4 Compilation en EPUB

Exécutez `.\make epub` pour générer le contenu en EPUB. Le contenu EPUB est situé dans le répertoire `build/epub` à la racine de la bibliothèque.

43.3.5 Ajout de librairies tierces Python

Important : Après avoir modifié les dépendances frc-docs de quelque manière que ce soit, `requirements.txt` doit être régénéré en exécutant `poetry export -f requirements.txt --output source/requirements.txt --without-hashes` à partir de la racine du repo.

frc-docs utilise Poetry pour gérer ses dépendances afin de s'assurer que les versions sont reproductibles.

Note : Poetry n'est ** pas ** nécessaire pour créer et contribuer au contenu frc-docs. Il est * uniquement * utilisé pour la gestion des dépendances.

Installation de Poetry

Assurez-vous que Poetry est installé. Exécutez la commande suivante : `` pip install poetry``

Ajouter une dépendance

Ajoutez la dépendance à la section `[tool.poetry.dependencies]` de `pyproject.toml`. Assurez-vous de spécifier une version exacte. Ensuite, exécutez la commande suivante : `poetry lock --no-update`

Mise à jour d'une dépendance de niveau supérieur

Mettez à jour la version de la dépendance dans la section `[tool.poetry.dependencies]` de `pyproject.toml`. Ensuite, exécutez la commande suivante : `poetry lock --no-update`

Mise à jour des dépendances masquées

Exécutez la commande suivante : `poetry lock`

43.4 Instructions de sauvegarde pour Draw.io

Avertissement : Ne modifiez aucun fichier du dossier `diagrams` ou se terminant par `.drawio.svg` avec un programme autre que draw.io, sinon vous risquez d'endommager les métadonnées du fichier, ce qui le rendrait inutilisable.

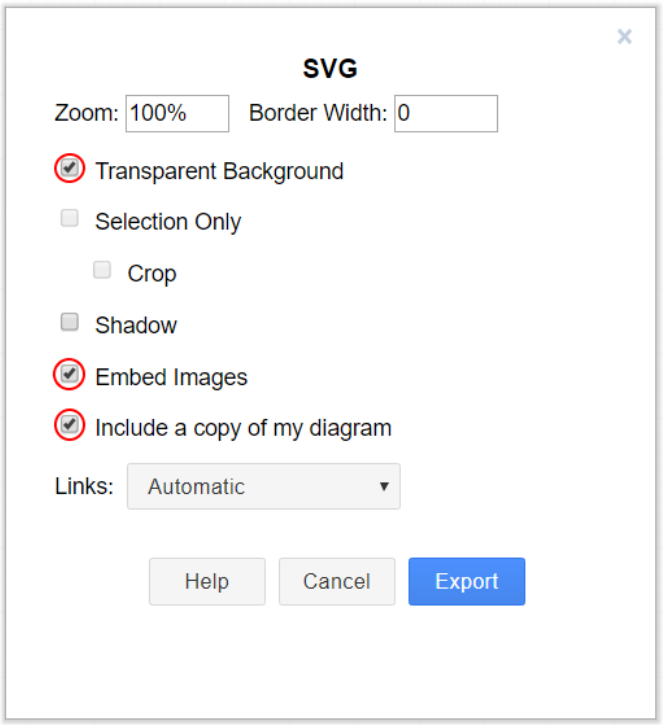
Les croquis Draw.io (ou diagrams.net) sont supportés si sauvegardés au format `svg`, avec métadonnées XML intégrées au document source draw.io (normalement d'extension `.drawio`). Ceci permet aux images d'agir comme sources pour les diagrammes qui pourront éventuellement être modifiés, et être affichées comme des fichiers `svg` réguliers.

Il existe quelques méthodes pour enregistrer un diagramme avec les métadonnées incorporées, mais l'utilisation du menu d'exportation est préférable car elle nous permet d'incorporer des images dans le diagramme, sans quoi elles risquent de ne pas s'afficher correctement sur les documents.

Cette méthode s'applique aux versions bureau et Web de draw.io, cette dernière étant disponible à diagrams.net.

Pour exporter, sélectionnez `File - Export as - SVG...`. Sélectionnez les options `Include a copy of my diagram` pour incorporer les métadonnées du croquis, et `Embed Images` pour incorporer les fichiers d'image du diagramme à être affichés dans le document. De plus, sélectionnez l'option `Transparent Background` pour que l'arrière-plan s'affiche correctement.

Le menu d'exportation devrait ressembler à ceci :



Cliquez simplement `Export` et sélectionnez la destination du fichier, puis sauvegardez-le.

Note : Lors de la sauvegarde, respectez les consignes de formatage ici *Fichiers Draw.io*

43.5 Traductions

frc-docs prend en charge les traductions à l'aide de l'utilitaire Web Transifex. frc-docs a été traduit en espagnol - Mexique (es_MX), Français - Canada (fr_CA) et turc - Turquie (tr_TR). Chinois - Chine (zh_CN), Hébreu - Israël (he_IL), et Portugais - Brésil (pt_BR) ont des traductions en cours. Les traducteurs qui parlent couramment *les deux* l'anglais et l'une des langues spécifiées seraient grandement appréciés pour contribuer aux traductions. Même une fois qu'une traduction est terminée, elle doit être mise à jour pour suivre les changements dans les frc-docs.

43.5.1 Plan de travail

Voici les étapes à suivre pour la traduction de frc-docs.

1. Inscrivez-vous à Transifex et demandez à rejoindre le projet frc-docs project, puis demandez l'accès spécifique à la langue sur laquelle vous souhaitez travailler.
2. Rejoignez les discussions GitHub ! Il s'agit d'un moyen de communication direct avec l'équipe WPILib. Vous pouvez l'utiliser pour nous poser des questions d'une manière rapide et simplifiée.
3. Vous pourriez être contacté et questionné concernant le langage à traduire avant d'avoir accès au projet de traduction frc-docs.
4. Traduisez !

43.5.2 Liens

Les liens doivent être conservés dans leur syntaxe d'origine. Pour traduire un lien, vous pouvez remplacer le texte TRANSLATE ME (celui-ci sera remplacé par le titre anglais) par la traduction appropriée.

Un exemple de texte à traduire :

```
For complete wiring instructions/diagrams, please see the :doc:`Wiring the FRC␣
↪Control System Document <Wiring the FRC Control System document>`.
```

où l'expression `Wiring the FRC Control System Document` doit être traduite.

```
For complete wiring instructions/diagrams, please see the :doc:`TRANSLATED TEXT
↪<Wiring the FRC Control System document>`.
```

Un autre exemple :

```
For complete wiring instructions/diagrams, please see the :ref:`TRANSLATED TEXT <docs/
↪zero-to-robot/step-1/how-to-wire-a-simple-robot:How to Wire an FRC Robot>`
```

43.5.3 Publier une traduction

Les traductions sont extraites de Transifex et publiées quotidiennement.

43.5.4 Exactitude

Les traductions doivent exactement respecter le texte original. Si le texte en anglais devrait être amélioré, rédigez un PR ou soulevez un problème sur frc-docs bibliothèque. Ces suggestions pourront être traduites et intégrées.

43.6 Principaux traducteurs

43.6.1 Mandarin (Chinois)

— 8192 Dhc
— Atlus Zhang
— Jiangshan Gong
— Keseterg
— Michael Zhao
— Ningxi Huang
— Ran Xin
— Team 5308
— Tianrui Wu
— Tianshuang Zhang
— Xun Sun
— Yitong Zhao
— Yuhao Li
— □□ □
— □□ □
— □□ □
— □□ □
— □□ □
— □□ □
— □ Sherry

43.6.2 Français

— Alexandra Schneider
— Andre Theberge
— Andy Chang
— Austin Shalit
— Dalton Smith
— Daniel Renaud
— Étienne Beaulac
— Félix Giffard
— Kaitlyn Kenwell
— Laura Luna Bedard
— Marc Lalonde
— Martin Regimbald
— Martin Rioux
— Regis Bekale
— Sami G.-D.
— Sidney Lavoie
— Youdlain Marcellus

43.6.3 Portugais

— Amanda Carolina Wilmsen
— Bibiana Oliveira
— Bruno Osio
— Bruno Toso
— Gabriel Silveira
— Gabriela Tomaz Do Amaral Ribeiro
— Günther Steinmeier
— Luca Carvalho
— Lucas Fontes Francisco
— Maria Eduarda Grabin Gisse
— Matheus Heitor Timm Chanan
— Miguel Ramos
— Miguel S. Ramos
— Natan Feijó Tristão
— Nathany Santiago
— Pedro Henrique Dias Pellicioli
— Rodrigo Anholon Novo
— Tales Dias De Almeida Silva
— Vinícius Castro
— Vinícius Gabriel Da Silva

43.6.4 Espagnol

— Austin Shalit
— Cesar Ernesto
— Diana Ramos
— Diego Lozano Rangel
— Fernanda Reveles
— Fernando Soltero
— Gibrán Verástegui
— Heber Sepúlveda
— Heriberto Gutierrez
— Hugo Espino
— Luis_Hernández
— Mariano
— Miguel Angel De León Adame
— Óscar Ariel Gutiérrez
— Paulina Maynez
— Pierre Cote
— Rodrigo Acosta
— Román Hernandez Sosa
— Sofia Fernandez
— Zara Moreno

43.6.5 Turc

— Hasan Bilgin
— Müfit Alkaya
— Esra Özemre
— Ceren Oktemer
— Demet T
— Demet Tumkaya
— Lal Serdaroğlu
— Melis Aldeniz
— Çağan Uslu
— Duru Ünlü
— Arhan Ünay
— Ada Zagyapan
— Doruk Akdoğan
— Müfit Alkaya_3390
— Mayra Şengel
— Tuna Özer
— Duru Hatipoğlu
— Elif Akın
— Ece Yiğit
— Nesrin Serra Köşkeroğlu

43.6.6 Hébreu

— Aric Radzin
— Dalton Smith
— Itay Ziv
— Ofek Ashery
— Shai Grossman
— Starlight220
— Yotam Shlomi

44

Développer avec allwpilib

Important : Ce document contient des informations pour les développeurs *de* la WPILib. Il n'est donc pas destiné à la programmation des robots FRC®.

Voici une liste de liens vers les diverses documentations du référentiel allwpilib

44.1 Démarrage rapide

Ci-dessous figure une liste d'instructions qui vous guidera au travers du clonage, de la construction et de la publication des fichiers binaires dans un projet de robot en utilisant la allwpililb locale. Ce guide de démarrage rapide n'est pas conçu comme un remplacement de l'information présente plus loin dans le document.

- Clonez le référentiel avec `git clone https://github.com/wpilibsuite/allwpilib.git`
- Construisez le référentiel avec `./gradlew build` ou `./gradlew build --build-cache` si vous possédez une connexion à internet
- Publiez les artéfacts localement en utilisant `./gradlew publish`
- Mettez à jour votre <https://github.com/wpilibsuite/allwpilib/blob/main/DevelopmentBuilds.md>`__ `build.gradle` du projet de robot pour utiliser les artéfacts <https://github.com/wpilibsuite/allwpilib/blob/main/DevelopmentBuilds.md>`__

44.2 Référentiel de base

44.3 NetworkTables

Index

A

accéléromètre, **583**
AM, **583**
AprilTags, **583**
auto, **583**

B

back-EMF, **583**
bang-bang control, **1226**
boolean, **583**

C

C++, **584**
CAD, **583**
call stack, **583**
CAM, **583**
CAN, **583**
cap, **586**
Cartesian coordinate system, **1226**
CC, 70
CD, **584**
central limit theorem, **584**
churning losses, **1226**
Châssis KOP, **586**
CIM, **584**
Classical Mechanics, **584**
composition, **584**
control signal, **1226**
convolution, **1226**
COTS, **584**
counter-electromotive force, **1226**
CRTP, **584**
CSA, **584**
CTRE, **584**
current, **1226**
CXX, 70

D

declarative programming, **584**
dependency injection, **584**
derivative, **1226**
design pattern, **585**
DHCP, **585**
dynamique, **1226**

E

Effort de contrôle, **1226**
encapsulation, **585**
entry, **585**
Entrée, **1227**
enumeration, **585**
EPA, **585**
Erreur, **1226**
Erreur régime permanant, **1229**
event-driven programming, **585**
exponential search, **1227**
exponential smoothing, **1227**

F

FIRST, **585**
FLL, **585**
floating point, **585**
FMS, **585**
FPGA, **585**
FRC, **585**
FTA, **586**
FTC, **586**

G

Gaussian distribution, **1227**
GDC, **586**
GP, **586**
gradient, **1227**
GradleRIO, **586**
gyroscope, **586**

I

I2C, **586**
imperative programming, **586**
Impulsion d'entrée, **1229**
IMU, **586**

J

Java, **586**
JSON, **586**

K

KOP, **586**

L

l'identification du système, **1229**
l'état caché, **1227**
la loi de commande, **1226**
la mesure, **1227**
LabVIEW, **586**
le gain, **1227**
Le paramètre Output (*Sortie*), **1228**
least-squares regression, **1227**
LED, **586**
LQR, **1227**

M

manette, **1226**
mass, **587**
Modèle, **1227**
moment of inertia, **587**
mutable, **587**
MXP, **587**

N

NetworkTables, **587**
no-op, **587**

O

Observateur, **1228**
obsolète, **584**
odometry, **587**
OPR, **587**
orthogonal, **1228**

P

PCM, **587**
PDH, **587**
PDP, **587**
permanent-magnet DC motor, **587**
persistent, **587**
PH, **587**
phase portrait, **1228**
PID, **1228**
Point de consigne, **1229**
pose, **588**
pose estimation, **588**
property, **588**
publisher, **588**
PWM, **588**
Python, **588**
Python Enhancement Proposals
    PEP 600, 69

R

r-squared, **1228**
RAII, **588**
recursive composition, **588**
retained, **588**
retro-reflection, **588**
REV, **588**
RMSE, **1228**
RPM, **589**
RSL, **589**
Référence, **1228**
Réponse d'impulsion, **1229**
Réponse du système, **1229**

S

serialized, **589**
signum function, **1229**
simulation, **589**
software library, **589**
solenoid valve, **589**
SPI, **589**
state machine, **589**
statistically robust, **1229**
subscriber, **589**
Système, **1229**

T

TBA, **589**
telemetry, **589**
Temps de montée, **1228**
Temps de stabilisation, **1229**
topic, **589**
torque, **589**
trajectoire, **590**
transitory, **590**
téléop, **589**

U

Usine, **1228**

V

variable d'environnement
    CC, 70
    CXX, 70
Variable de procédé, **1228**
viscous drag, **1230**
voltage, **1229**
VRM, **590**

W

WCP, **590**
WFA, **590**

X

x-châpeau, **1230**
x-point, **1230**

##

état, **1229**