FIRST Robotics Competition

WPILib

18 de mayo de 2024

Cero a Robot

¡Bienvenido a la Documentación de Sistema de Control de *FIRST*® Robotics Competition! ¡Este sitio contiene todo lo que necesitas saber para programar un robot de competencia!

Las traducciones comunitarias se encuentran en varios idiomas en el menú inferior izquierdo.

Equipos que regresan

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.

Registro de cambios

Problemas Conocidos

Inicio Rapido

Nuevos Equipos

¡El tutorial de Cero a Robot lo guiará a través de la preparación, cableado y la programación básica del robot!

Ve a Zero-to-Robot

Hardware Overview

Vista general de los componentes disponibles de hardware para los equipos.

Ve a Hardware Overview

Software Overview

Vista general de los componentes y herramientas de software disponibles para los equipos.

Ve a Software Overview

Programming Basics

Documentación que es útil a lo largo del proceso de programación de un equipo.

Ver artículos

Programación Avanzada

Es la documentación para los equipos más veteranos. Incluye contenido como Path Planning y Kinematics.

Ver artículos

Hardware

Tutoriales de hardware y contenido disponible para equipos.

Ver artículos

Romi y los robots XRP

Los robots Romi y XRP son plataformas de bajo costo para practicar la programación de WPILib.

Vea artículos de Romi

Vea artículos de XRP

Documentación de la API

Java, C++, and Python class documentation.

Java

C++

Python

Herramientas de software

Herramientas esenciales como son FRC Driver Station, Dashboards, roboRIO Imaging Tool y mas.

Ver artículos

Proyectos de ejemplo

Esta sección muestra los proyectos de ejemplo disponibles a los que los equipos pueden hacer referencia en VS Code.

Ver artículos

Referencia rápida de la luz de estado

Guía de referencia rápida para las luces de estado en una variedad de hardware de FRC.

Ver artículo

Bibliotecas de terceros

Tutorial sobre cómo agregar bibliotecas de terceros como CTRE y REV a su proyecto de robot.

Ver artículo

1 Introducción

Bienvenido a la documentación oficial del Sistema de Control de *FIRST*® Robotics Competition y los paquetes de software de WPILib. Esta página es el recurso principal que documenta el uso del Sistema de Control de FRC® (incluido el cableado, la configuración y el software), así como las bibliotecas y herramientas de WPILib.

1.1 ¿Es nuevo en programación?

Estas páginas cubren los detalles de las bibliotecas de WPILib y el sistema de control de FRC y no describen los conceptos básicos del uso de los lenguajes de programación compatibles. Si desea recursos para aprender los lenguajes de programación compatibles, consulte las recomendaciones a continuación:

Nota: Puede continuar con esta sección Zero-to-Robot para obtener un robot básico que funcione sin conocimiento del lenguaje de programación. Para ir más allá, necesita familiarizarse con el lenguaje en el que elija programar.

1.1.1 Java

- Code Academy
- Head First Java 2nd Edition es una introducción muy amigable a la programación en Java (ISBN-10: 0596009208).

1.1.2 C++

- LearnCPP
- Programming: Principles and Practice Using C++ 2nd Edition es un introducción a C++ por el creador del lenguaje (ISBN-10: 0321992784).
- C++ Primer Plus 6th Edition (ISBN-10: 0321776402).

1.1.3 LabVIEW

- NI Aprenda LabVIEW

1.1.4 Python

- List of various guides to learn Python

1.2 Zero to 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

Paso 1: construyendo tu robot

Se puede encontrar una descripción general del hardware del sistema de control disponible: *aquí*.

2.1 Introduction to FRC Robot Wiring

Nota: This document details the wiring of a basic electronics board for the kitbot or to allow basic drivetrain testing.

Some images shown in this section reflect the setup for a Robot Control System using SPARK or SPARK MAX Motor Controllers. Wiring diagram and layout should be similar for other motor controllers. Where appropriate, two sets of images are provided to show connections using controllers with and without integrated wires.

2.1.1 Overview

REV

CTR

2.1.2 Gather Materials

Locate the following control system components and tools

- Kit Materials:
  - Power Distribution Hub (*PDH*) / Power Distribution Panel (*PDP*)
  - roboRIO
  - Pneumatics Hub (*PH*) / Pneumatics Control Module (*PCM*)
  - Radio Power Module (*RPM*) / Voltage Regulator Module (*VRM*)

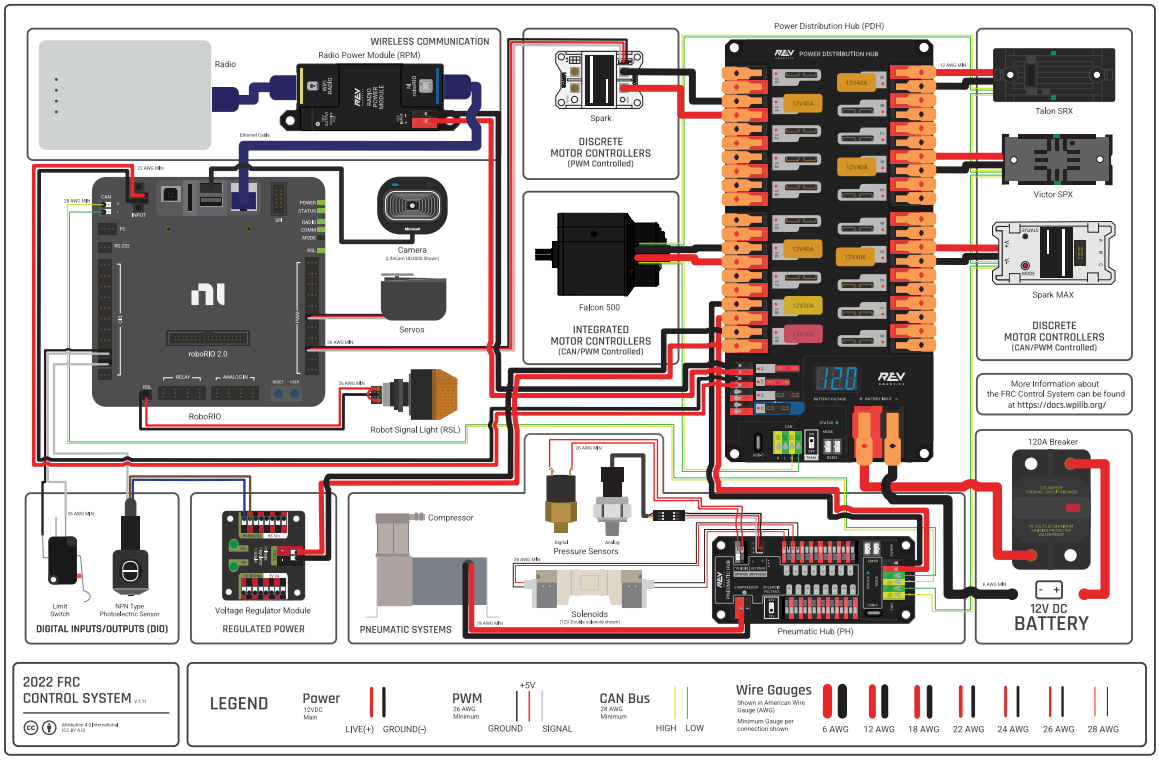


Figura 1: Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

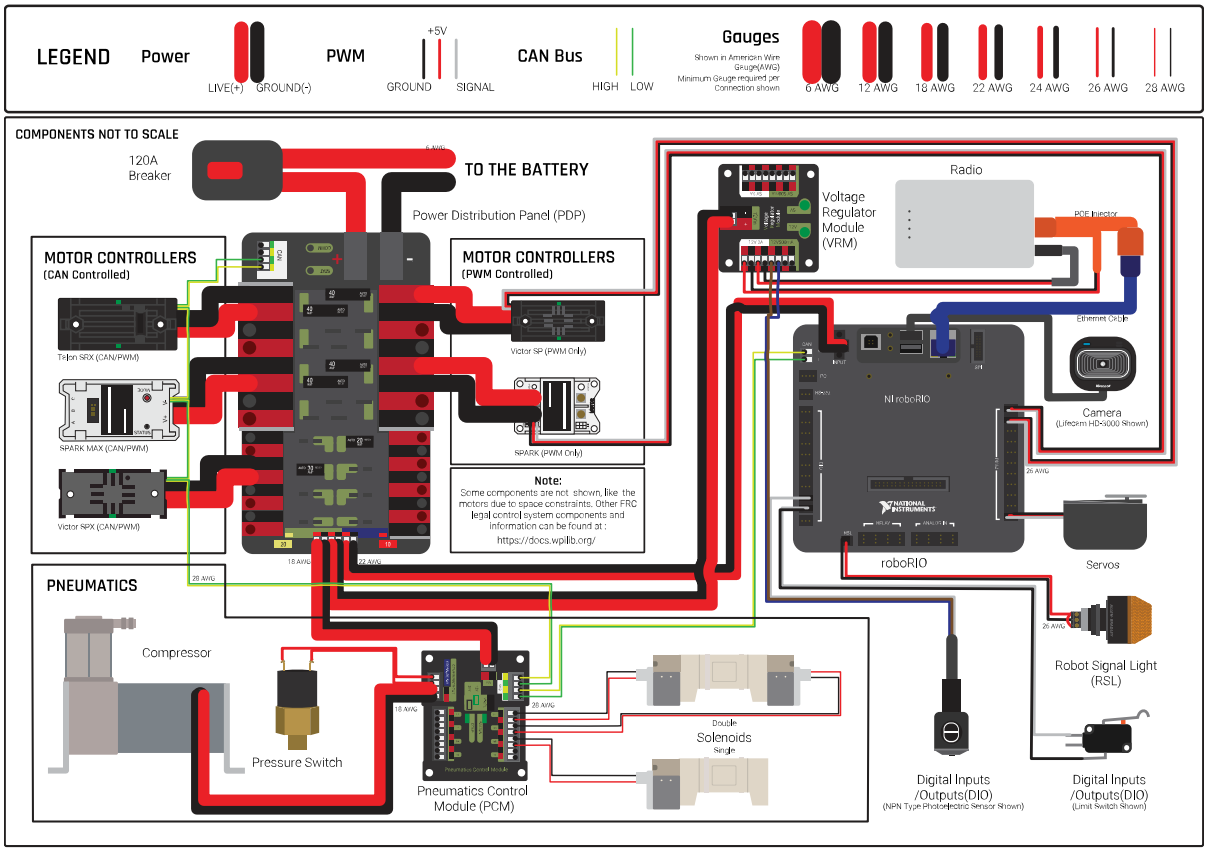


Figura 2: Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

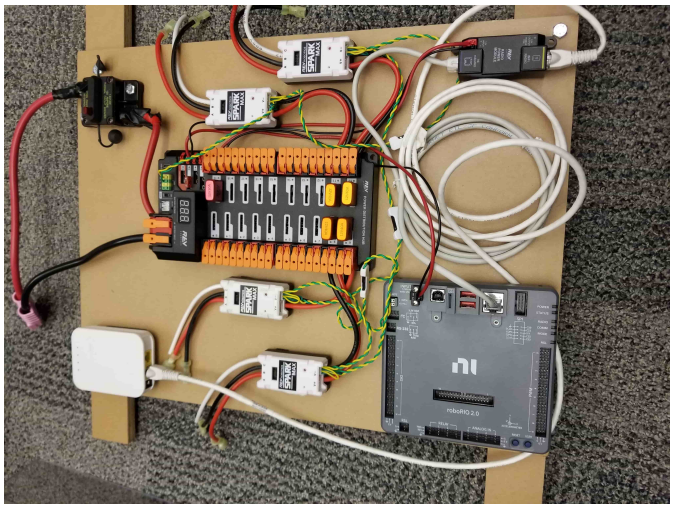
  - OpenMesh radio (with power cable and Ethernet cable)
  - Robot Signal Light (*RSL*)
  - 4x SPARK MAX or other motor controllers
  - 2x *PWM* y-cables
  - 120A Circuit breaker
  - 4x 40A Circuit breaker
  - 6 AWG (16 mm^2) Red wire
  - 10 AWG (6 mm^2) Red/Black wire
  - 18 AWG (1 mm^2) Red/Black wire
  - 22 AWG (0.5 mm^2) Yellow/Green twisted *CAN* cable
  - 8x Pairs of 10-12 AWG (4 - 6 mm^2) (Yellow) quick disconnect terminals (16x ring terminals if using integrated wire controllers)
  - 2x Anderson SB50 battery connectors
  - 6 AWG (16 mm^2) Terminal lugs
  - 12V Battery
  - Red/Black Electrical tape
  - Dual Lock material or fasteners
  - Zip ties
  - 1/4» or 1/2» (6-12 mm) plywood
- Tools Required:
  - Wago Tool or small flat-head screwdriver
  - Very small flat head screwdriver (eyeglass repair size)
  - Wire cutters, strippers, and crimpers
  - 7/16» (11 mm may work if imperial is unavailable) box end wrench or nut driver
  - Additional 7/16» wrench/nut driver or Philips head screw driver
  - For CTR PDP only: 5 mm Hex key (3/16» may work if metric is unavailable)
  - For CTR PDP only: 1/16» Hex key

2.1.3 Create the Base for the Control System

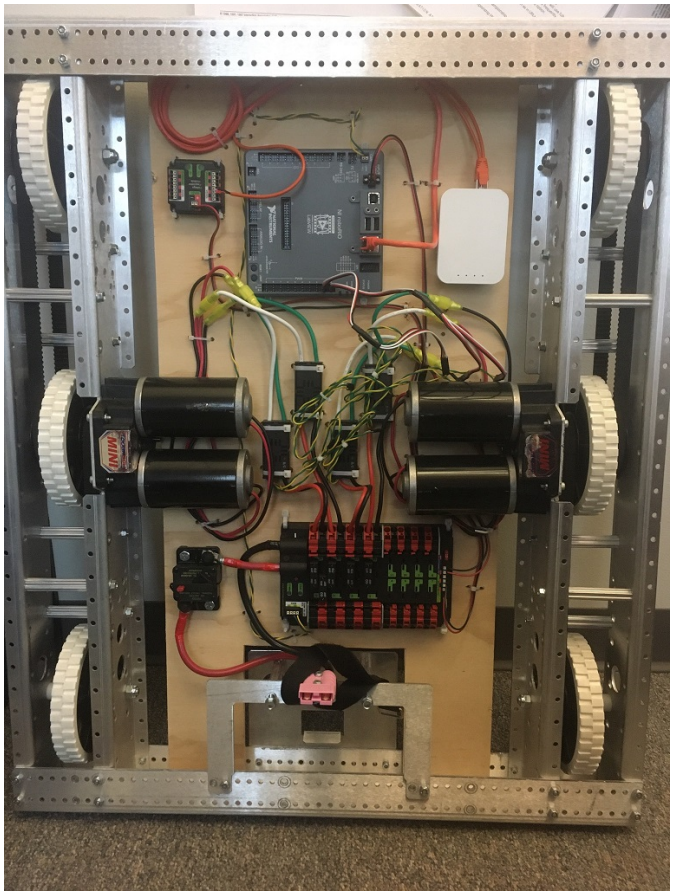
For a test board, cut piece of 1/4» or 1/2» (6-12 mm) material (wood or plastic) approximately 24» x 16» (60 x 40 cm). For a Robot Quick Build control board see the supporting documentation for the proper size board for the chosen chassis configuration.

2.1.4 Layout the Core Control System Components

REV

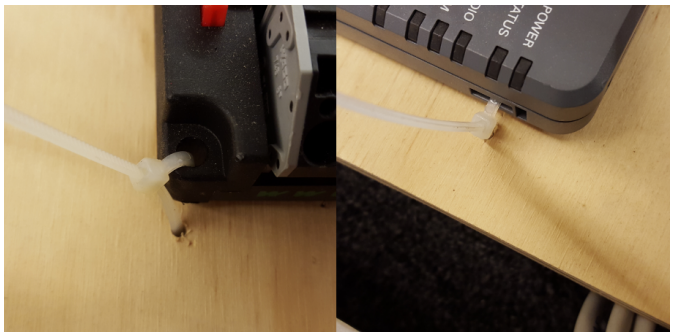


CTR



Lay out the components on the board. An example layout is shown in the image above.

2.1.5 Fasten Components



Using the Dual Lock or hardware, fasten all components to the board. Note that in many FRC games robot-to-robot contact may be substantial and Dual Lock alone is unlikely to stand up as a fastener for many electronic components. Teams may wish to use nut and bolt fasteners or (as shown in the image above) cable ties, with or without Dual Lock to secure devices to the board.

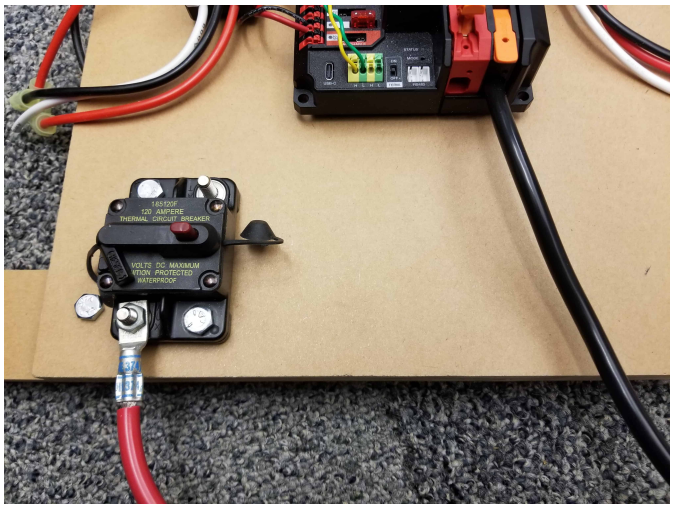
2.1.6 Attach Robot Side Battery Connector

REV

The next step will involve using the Wago connectors on the PDH. To use the Wago connectors, open the lever, insert the wire, then close the lever. Two sizes of Wago connector are found on the PDH:

- Main power connectors: Accept 4 - 18 AWG (.75 - 25 mm^2), strip 20 mm (~3/4»)
- High current channel connectors: Accept 8 - 24 AWG (.25 - 10 mm^2), strip 12 mm (~1/2»)

To maximize pullout force and minimize connection resistance wires should not be tinned (and ideally not twisted) before inserting into the Wago connector.



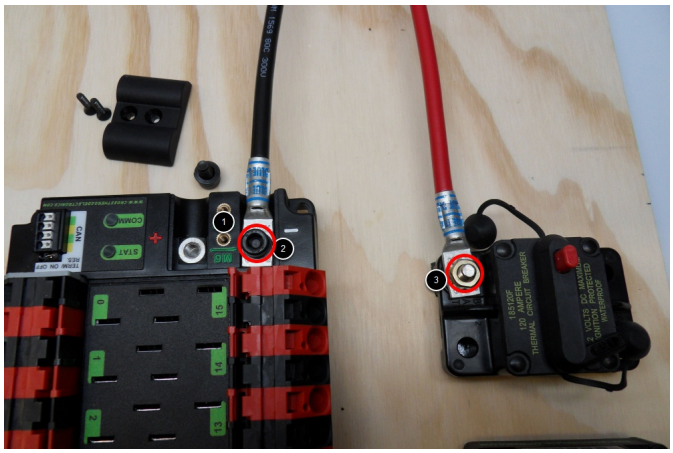
Requires: Battery Connector, 6 AWG (16 mm^2) terminal lugs, 7/16» (11 mm) Box end

Attach terminal lug to positive (red) wire of battery connector. Strip .75» off the black wire.

Lift the lever above the black main power input terminal on the PDH until it clicks into place. Insert the wire. Pull the lever down to secure the wire.

Using a 7/16» (11 mm) box end wrench, remove the nut on the «Batt» side of the main breaker and secure the positive terminal of the battery connector

CTR



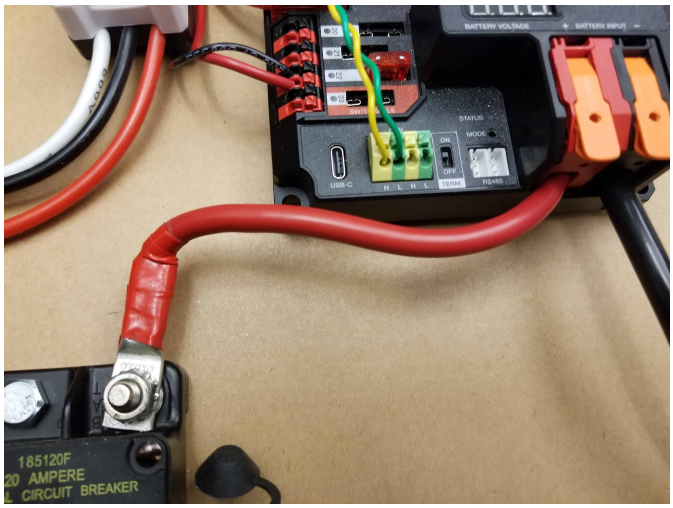
Requires: Battery Connector, 6 AWG (16 mm^2) terminal lugs, 1/16» Allen, 5 mm Allen, 7/16» (11 mm) Box end

Attach terminal lugs to battery connector.

1. Using a 1/16» Allen wrench, remove the two screws securing the PDP terminal cover.
2. Using a 5 mm Allen wrench (3/16»), remove the negative (-) bolt and washer from the PDP and fasten the negative terminal of the battery connector.
3. Using a 7/16» (11 mm) box end wrench, remove the nut on the «Batt» side of the main breaker and secure the positive terminal of the battery connector

2.1.7 Wire Breaker to Power Distribution

REV

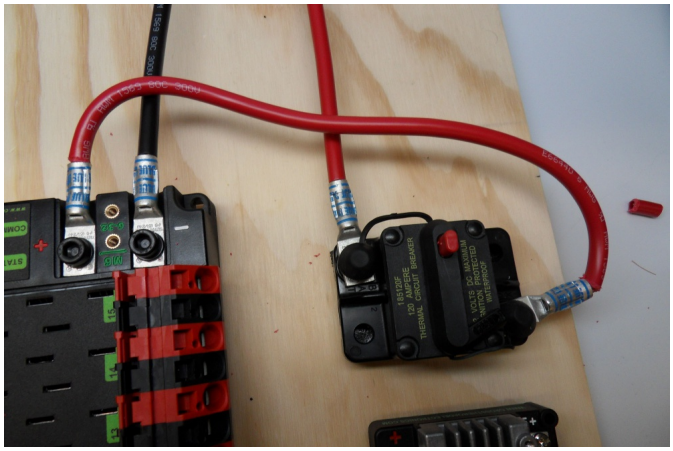


Requires: 6 AWG (16 mm^2) red wire, 1x 6 AWG (16 mm^2) terminal lugs, 7/16» (11 mm) wrench

Secure one terminal lug to the end of the 6 AWG (16 mm^2) red wire. Using the 7/16» (11 mm) wrench, remove the nut from the «AUX» side of the 120A main breaker and place the terminal over the stud. Loosely secure the nut (you may wish to remove it shortly to cut and strip the other end of the wire). Measure out the length of wire required to reach the positive terminal of the PDH.

1. Cut and strip the other end of the red wire.
2. Using the 7/16» (11 mm) wrench, secure the wire to the «AUX» side of the 120A main breaker.
3. Lift the lever on the positive (red) input terminal of the PDH, insert the wire, then close the terminal.

CTR



Requires: 6 AWG (16 mm^2) red wire, 2x 6 AWG (16 mm^2) terminal lugs, 5 mm Allen, 7/16» (11 mm) box end

Secure one terminal lug to the end of the 6 AWG (16 mm^2) red wire. Using the 7/16» (11 mm) box end, remove the nut from the «AUX» side of the 120A main breaker and place the terminal over the stud. Loosely secure the nut (you may wish to remove it shortly to cut, strip, and crimp the other end of the wire). Measure out the length of wire required to reach the positive terminal of the PDP.

1. Cut, strip, and crimp the terminal to the 2nd end of the red 6 AWG (16 mm^2) wire.
2. Using the 7/16» (11 mm) box end, secure the wire to the «AUX» side of the 120A main breaker.
3. Using the 5 mm Allen wrench, secure the other end to the PDP positive terminal.

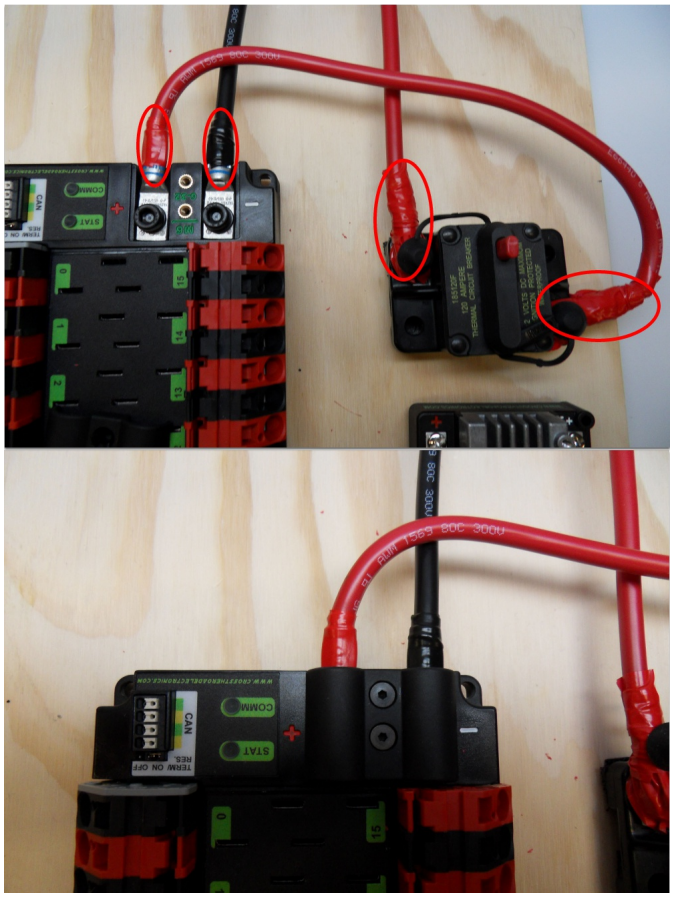
2.1.8 Insulate power connections

REV

Requires: Electrical tape

Using electrical tape, insulate the two connections to the 120A breaker.

CTR

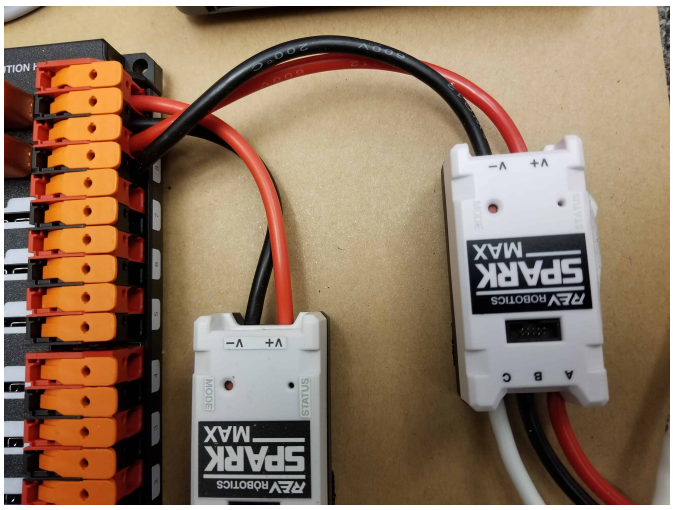


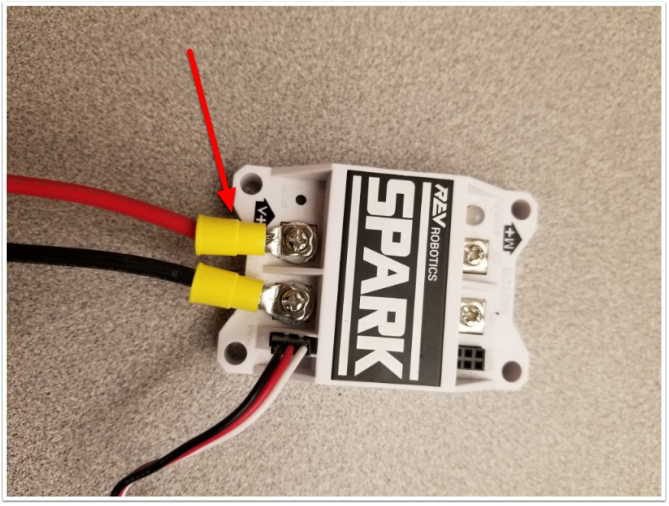
Requires: 1/16» Allen, Electrical tape

1. Using electrical tape, insulate the two connections to the 120A breaker. Also insulate any part of the PDP terminals which will be exposed when the cover is replaced.
2. Using the 1/16» Allen wrench, replace the PDP terminal cover

2.1.9 Motor Controller Power

REV





Requires: Wire Stripper Terminal Controllers only: 10 or 12 AWG (4 - 6 mm^2) wire , 10 or 12 AWG (4 - 6 mm^2) fork/ring terminals, wire crimper

For SPARK MAX or other wire integrated motor controllers (top image):

- Cut and strip the red and black power input wires, then insert into one of the Wago terminal pairs.

For terminal motor controllers (bottom image):

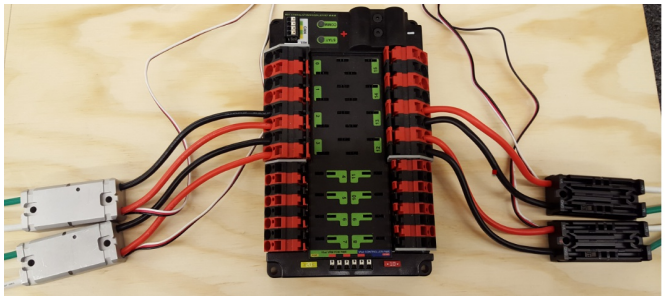
1. Cut red and black wire to appropriate length to reach from one of the Wago terminal pairs to the input side of the motor controller (with a little extra for the length that will be inserted into the terminals on each end)
2. Strip one end of each of the wires, then insert into the Wago terminals.
3. Strip the other end of each wire, and crimp on a ring or fork terminal
4. Attach the terminal to the motor controller input terminals (red to +, black to -)

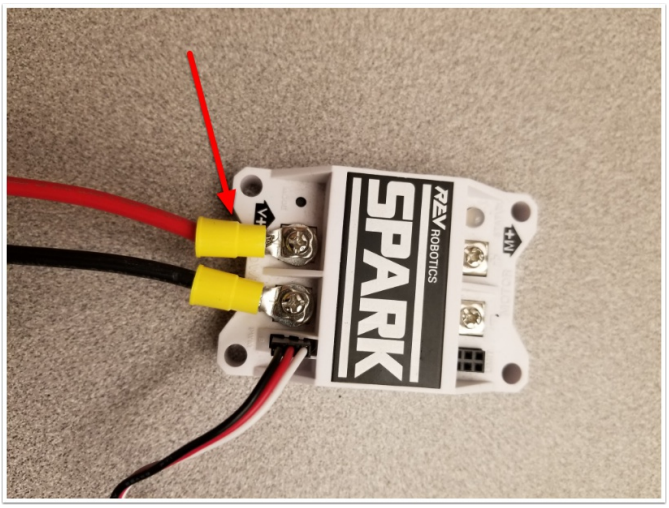
CTR

The next step will involve using the Wago connectors on the PDP. To use the Wago connectors, insert a small flat blade screwdriver into the rectangular hole at a shallow angle then angle the screwdriver upwards as you continue to press in to actuate the lever, opening the terminal. Two sizes of Wago connector are found on the PDP:

- Small Wago connector: Accepts 10 - 24 AWG (0.25 - 6 mm^2), strip 11-12 mm (~7/16»)
- Large Wago connector: Accepts 6 - 12 AWG (4 - 16 mm^2), strip 12-13 mm (~1/2»)

To maximize pullout force and minimize connection resistance wires should not be tinned (and ideally not twisted) before inserting into the Wago connector.





Requires: Wire Stripper, Small Flat Screwdriver, Terminal Controllers only: 10 or 12 AWG (4 - 6 mm^2) wire, 10 or 12 AWG (4 - 6 mm^2) fork/ring terminals, wire crimper

For SPARK MAX or other wire integrated motor controllers (top image):

- Cut and strip the red and black power input wires, then insert into one of the 40A (larger) Wago terminal pairs.

For terminal motor controllers (bottom image):

1. Cut red and black wire to appropriate length to reach from one of the 40A (larger) Wago terminal pairs to the input side of the motor controller (with a little extra for the length that will be inserted into the terminals on each end)
2. Strip one end of each of the wires, then insert into the Wago terminals.
3. Strip the other end of each wire, and crimp on a ring or fork terminal
4. Attach the terminal to the motor controller input terminals (red to +, black to -)

2.1.10 Weidmuller Connectors

A number of the CAN and power connectors in the system use a Weidmuller LSF series wire-to-board connector. There are a few things to keep in mind when using this connector for best results:

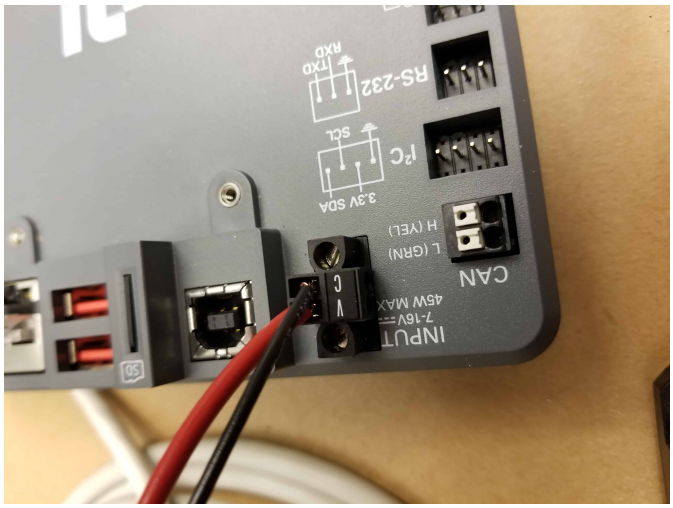
- Wire should be 16 AWG (1.5 mm^2) to 24 AWG (0.25 mm^2) (consult rules to verify required gauge for power wiring)
- Wire ends should be stripped approximately 5/16 (~8 mm)»
- To insert or remove the wire, press down on the corresponding «button» to open the terminal

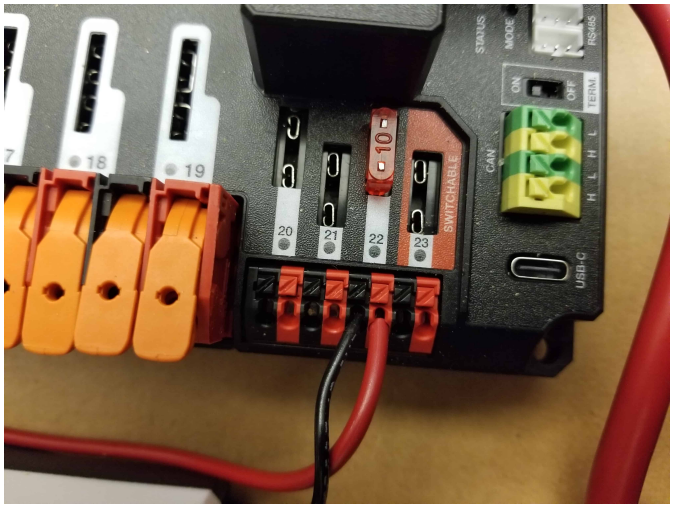
After making the connection check to be sure that it is clean and secure:

- Verify that there are no «whiskers» outside the connector that may cause a short circuit
- Tug on the wire to verify that it is seated fully. If the wire comes out and is the correct gauge it needs to be inserted further and/or stripped back further. Occasionally the terminal may remain stuck open with the wire inserted and the button released even if the wire is stripped and inserted properly; in these cases wiggling the wire in and out a small amount will often allow the connector to latch shut and grip the wire.

2.1.11 roboRIO Power

REV

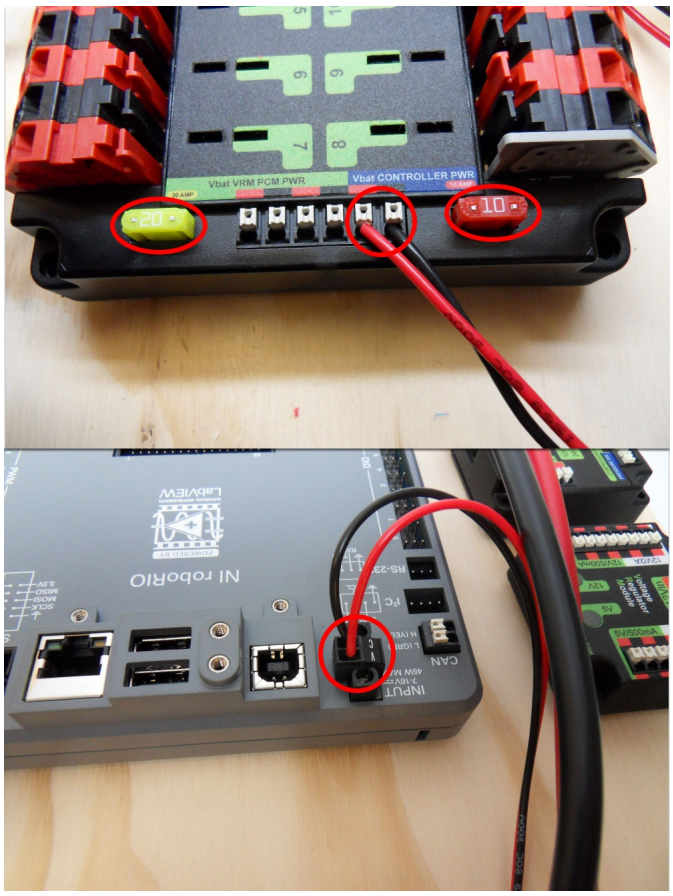




Requires: 10A mini fuse, Wire stripper, very small flat screwdriver, 18 AWG (1 mm^2) Red and Black

1. Insert the 10A fuse into the PDH in one of the non-switchable fused channels (20-22).
2. Strip ~5/16» (~8 mm) on both the red and black 18 AWG (1 mm^2) wire and connect to the corresponding terminals on the PDH channel where the fuse was installed
3. Measure the required length to reach the power input on the roboRIO. Take care to leave enough length to route the wires around any other components such as the battery and to allow for any strain relief or cable management.
4. Cut and strip the wire.
5. Using a very small flat screwdriver connect the wires to the power input connector of the roboRIO (red to V, black to C). Also make sure that the power connector is screwed down securely to the roboRIO.

CTR

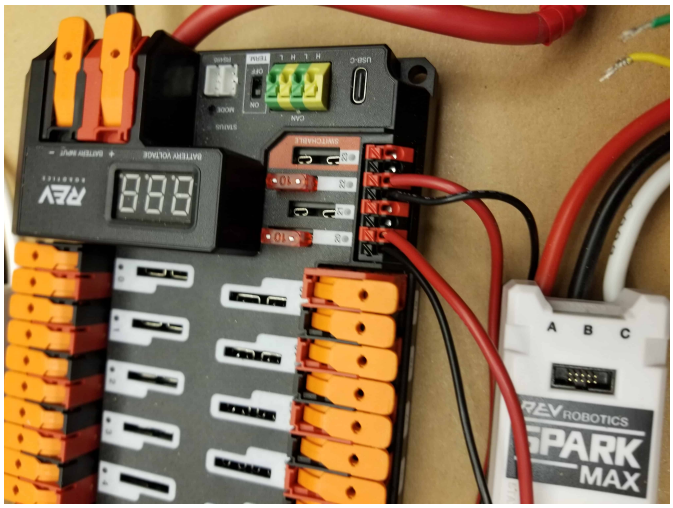


Requires: 10A/20A mini fuses, Wire stripper, very small flat screwdriver, 18 AWG (1 mm^2) Red and Black

1. Insert the 10A and 20A mini fuses in the PDP in the locations shown on the silk screen (and in the image above)
2. Strip ~5/16» (~8 mm) on both the red and black 18 AWG (1 mm^2) wire and connect to the «Vbat Controller PWR» terminals on the PDB
3. Measure the required length to reach the power input on the roboRIO. Take care to leave enough length to route the wires around any other components such as the battery and to allow for any strain relief or cable management.
4. Cut and strip the wire.
5. Using a very small flat screwdriver connect the wires to the power input connector of the roboRIO (red to V, black to C). Also make sure that the power connector is screwed down securely to the roboRIO.

2.1.12 Radio Power

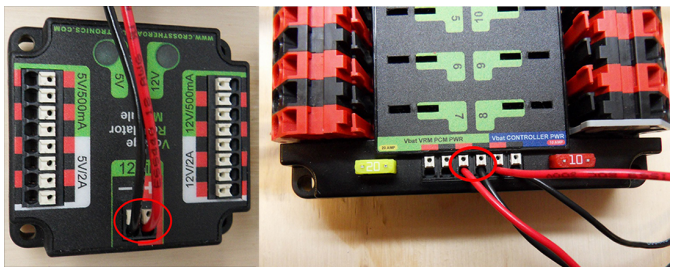
REV



Requires: Wire stripper, small flat screwdriver (optional), 18 AWG (1 mm^2) red and black wire:

1. Insert the 10A fuse into the PDH in one of the non-switchable fused channels (20-22).
2. Strip ~5/16» (~8 mm) on the end of the red and black 18 AWG (1 mm^2) wire and connect the wire to the corresponding terminals on the PDH.
3. Measure the length required to reach the «12V Input» terminals on the Radio Power Module. Take care to leave enough length to route the wires around any other components such as the battery and to allow for any strain relief or cable management.
4. Cut and strip ~5/16» (~8 mm) from the end of the wire.
5. Connect the wire to the RPM 12V Input terminals.

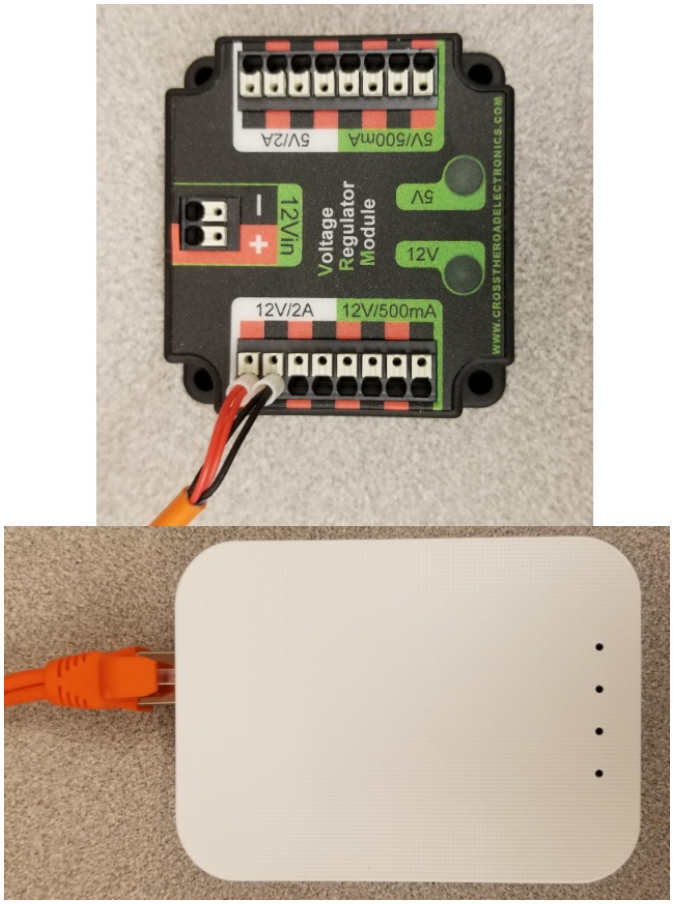
CTR



Requires: Wire stripper, small flat screwdriver (optional), 18 AWG (1 mm^2) red and black wire:

1. Strip ~5/16» (~8 mm) on the end of the red and black 18 AWG (1 mm^2) wire.
2. Connect the wire to one of the two terminal pairs labeled «Vbat VRM PCM PWR» on the PDP.
3. Measure the length required to reach the «12Vin» terminals on the VRM. Take care to leave enough length to route the wires around any other components such as the battery and to allow for any strain relief or cable management.
4. Cut and strip ~5/16» (~8 mm) from the end of the wire.
5. Connect the wire to the VRM 12Vin terminals.

Advertencia: DO NOT connect the Rev passive POE injector cable directly to the roboRIO. The roboRIO MUST connect to the socket end of the cable using an additional Ethernet cable as shown in the next step.

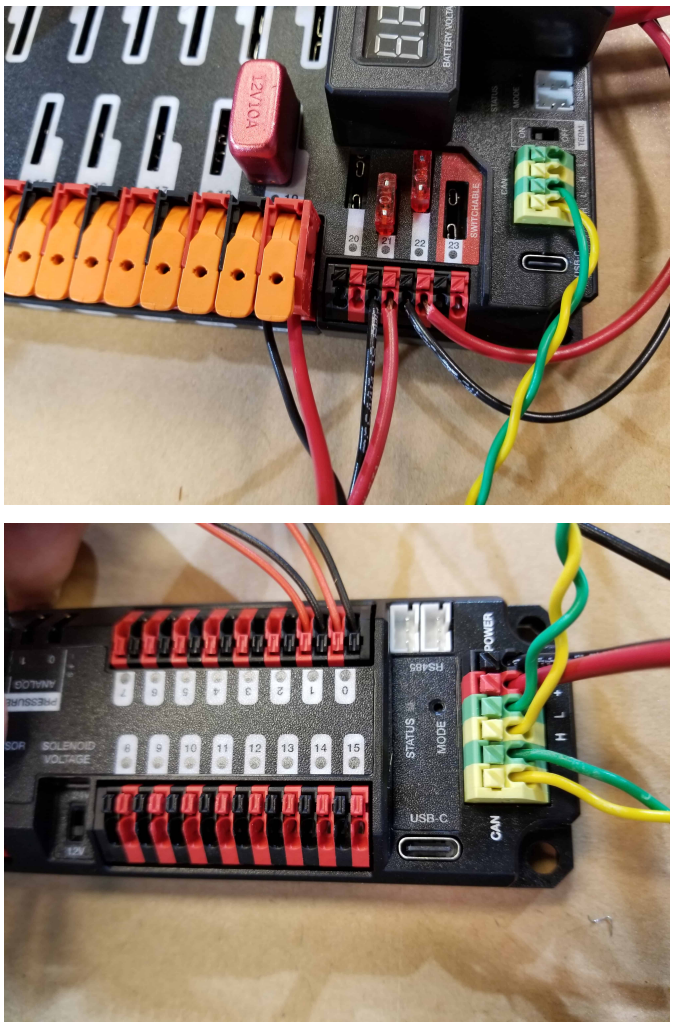


Requires: Small flat screwdriver (optional), Rev radio PoE cable

1. Insert the ferrules of the passive PoE injector cable into the corresponding colored terminals on the 12V/2A section of the VRM.
2. Connect the RJ45 (Ethernet) plug end of the cable into the Ethernet port on the radio closest to the barrel connector (labeled 18-24v POE)

2.1.13 Pneumatics Power (Optional)

REV



Requires: Wire stripper, small flat screwdriver (optional), 18 AWG (1 mm^2) red and black wire

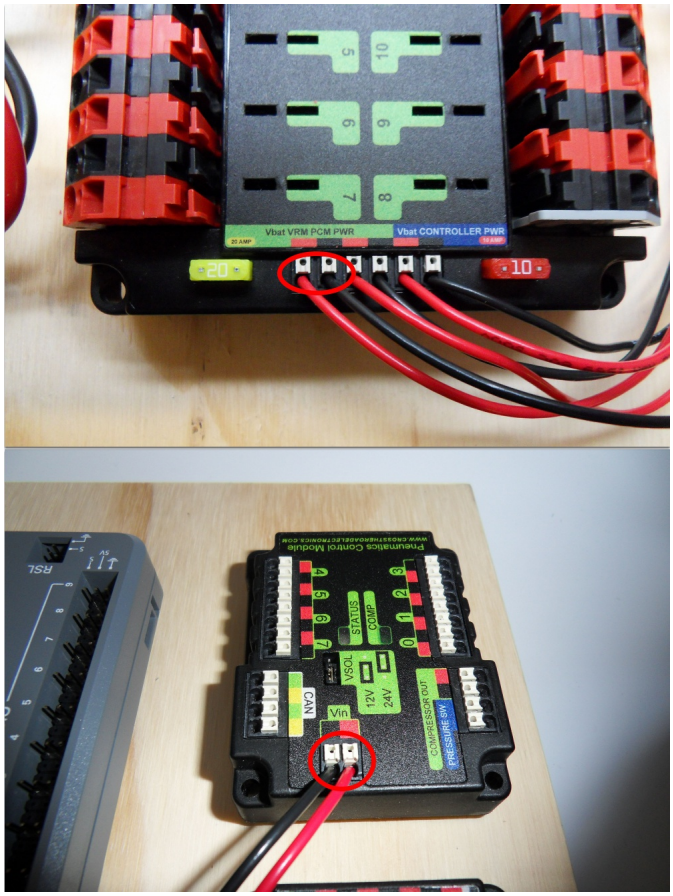
---

Nota: The Pneumatics Hub is an optional component used for controlling pneumatics on the robot.

The Pneumatics Hub can be wired to either a non-switchable fused port on the PDH with a 15A or smaller fuse or to a circuit breaker protected port with a breaker up to 20A.

1. Strip ~5/16» (~8 mm) on the end of the red and black 18 AWG (1 mm^2) wire.
2. Connect the wire to the PDH in one of the two ways described above
3. Measure the length required to reach the red terminals on the short end of the PH labeled +/-. Take care to leave enough length to route the wires around any other components such as the battery and to allow for any strain relief or cable management.
4. Cut and strip ~5/16» (~8 mm) from the other end of the wire.
5. Connect the wire to the PH input terminals.

CTR



Requires: Wire stripper, small flat screwdriver (optional), 18 AWG (1 mm^2) red and black wire

Nota: The PCM is an optional component used for controlling pneumatics on the robot.

1. Strip ~5/16» (~8 mm) on the end of the red and black 18 AWG (1 mm^2) wire.
2. Connect the wire to one of the two terminal pairs labeled «Vbat VRM PCM PWR» on the PDP.
3. Measure the length required to reach the «Vin» terminals on the PCM. Take care to leave enough length to route the wires around any other components such as the battery and to allow for any strain relief or cable management.
4. Cut and strip ~5/16» (~8 mm) from the end of the wire.
5. Connect the wire to the PCM 12Vin terminals.

2.1.14 Ethernet Cables

REV

Requires: 2x Ethernet cables

1. Connect an Ethernet cable from the RJ45 (Ethernet) socket of the roboRIO to the port on the Radio Power Module labeled roboRIO.
2. Connect an Ethernet cable from the RJ45 socket of the radio closest to the barrel connector socket (labeled 18-24v POE) to the socket labeled WiFi Radio on the RPM

CTR

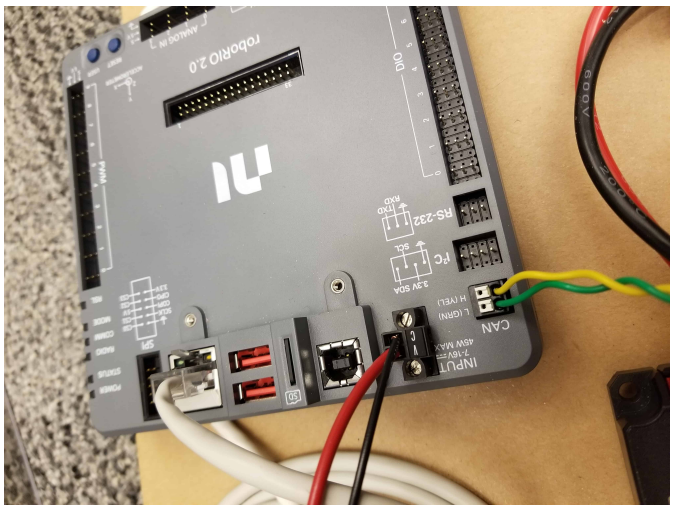
Requires: Ethernet cable

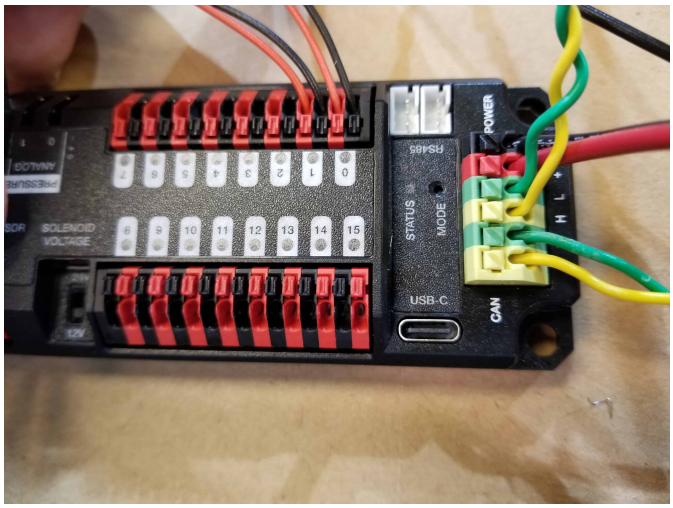
Connect an Ethernet cable from the RJ45 (Ethernet) socket of the Rev Passive POE cable to the RJ45 (Ethernet) port on the roboRIO.

2.1.15 CAN Devices

roboRIO to Pneumatics CAN

REV





Requires: Wire stripper, small flat screwdriver (optional), yellow/green twisted CAN cable

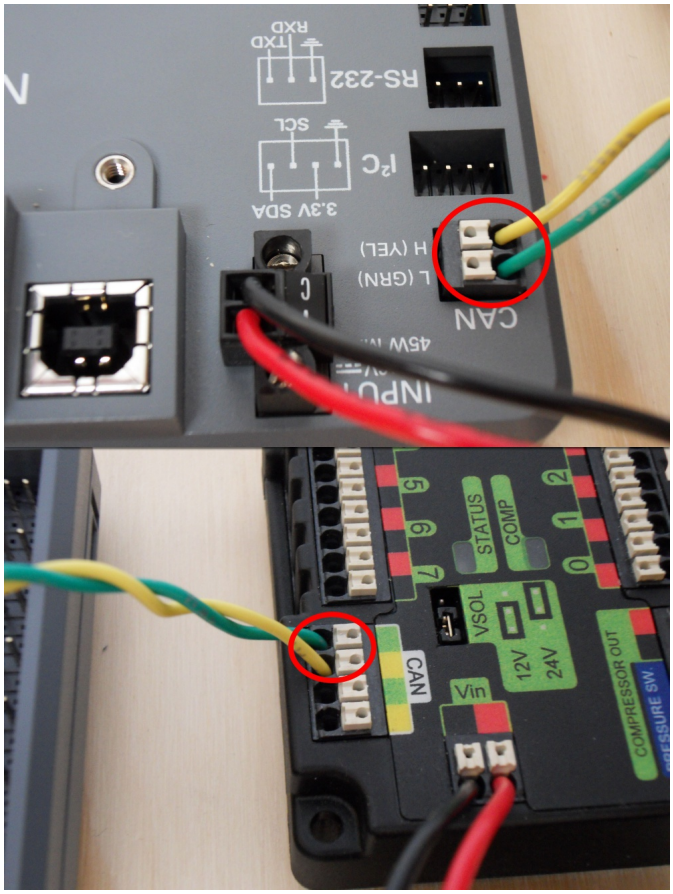
---

Nota: The PH is an optional component used for controlling pneumatics on the robot. If you are not using the PH, wire the CAN connection directly from the roboRIO (shown in this step) to the PDH (shown in the next step).

---

1. Strip ~5/16» (~8 mm) off of each of the CAN wires.
2. Insert the wires into the appropriate CAN terminals on the roboRIO (Yellow->YEL, Green->GRN).
3. Measure the length required to reach the CAN terminals of the PCM (either of the two available pairs). Cut and strip ~5/16» (~8 mm) off this end of the wires.
4. Insert the wires into the appropriate color coded CAN terminals on the PH. You may use either of the Yellow/Green terminal pairs on the PH, there is no defined in or out.

CTR



Requires: Wire stripper, small flat screwdriver (optional), yellow/green twisted CAN cable

---

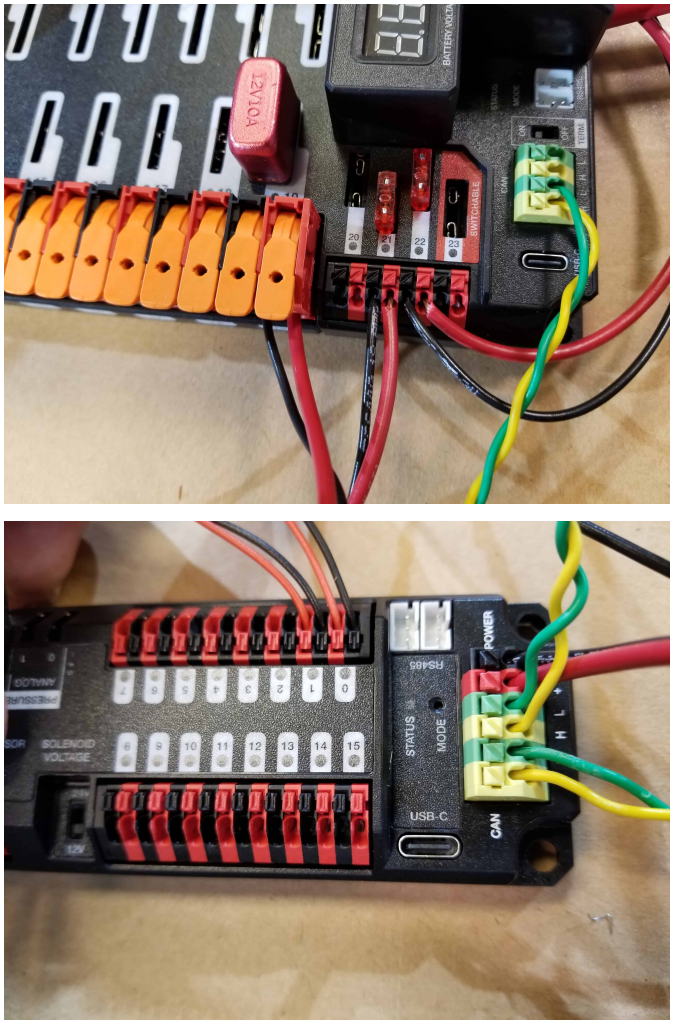
Nota: The PCM is an optional component used for controlling pneumatics on the robot. If you are not using the PCM, wire the CAN connection directly from the roboRIO (shown in this step) to the PDP (shown in the next step).

---

1. Strip ~5/16» (~8 mm) off of each of the CAN wires.
2. Insert the wires into the appropriate CAN terminals on the roboRIO (Yellow->YEL, Green->GRN).
3. Measure the length required to reach the CAN terminals of the PCM (either of the two available pairs). Cut and strip ~5/16» (~8 mm) off this end of the wires.
4. Insert the wires into the appropriate color coded CAN terminals on the PCM. You may use either of the Yellow/Green terminal pairs on the PCM, there is no defined in or out.

Pneumatics to PD CAN

REV



Requires: Wire stripper, small flat screwdriver (optional), yellow/green twisted CAN cable

---

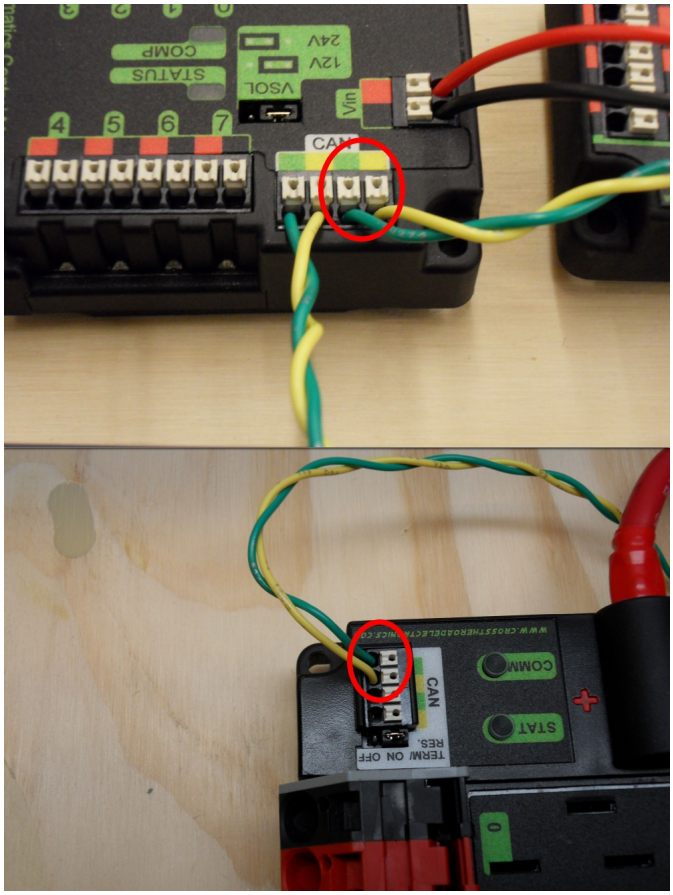
Nota: The PH is an optional component used for controlling pneumatics on the robot. If you are not using the PH, wire the CAN connection directly from the roboRIO (shown in the above step) to the PDH (shown in this step).

---

1. Strip ~5/16» (~8 mm) off of each of the CAN wires.
2. Insert the wires into the appropriate CAN terminals on the PH.
3. Measure the length required to reach the CAN terminals of the PDH (either of the two available pairs). Cut and strip ~5/16» (~8 mm) off this end of the wires.
4. Insert the wires into the appropriate color coded CAN terminals on the PDH. You may use either of the Yellow/Green terminal pairs on the PDH, there is no defined in or out.

Nota: See the *CAN Wiring Basics* if you need to terminate the CAN bus somewhere other than the PDP.

CTR



Requires: Wire stripper, small flat screwdriver (optional), yellow/green twisted CAN cable

Nota: The PCM is an optional component used for controlling pneumatics on the robot. If you are not using the PCM, wire the CAN connection directly from the roboRIO (shown in the above step) to the PDP (shown in this step).

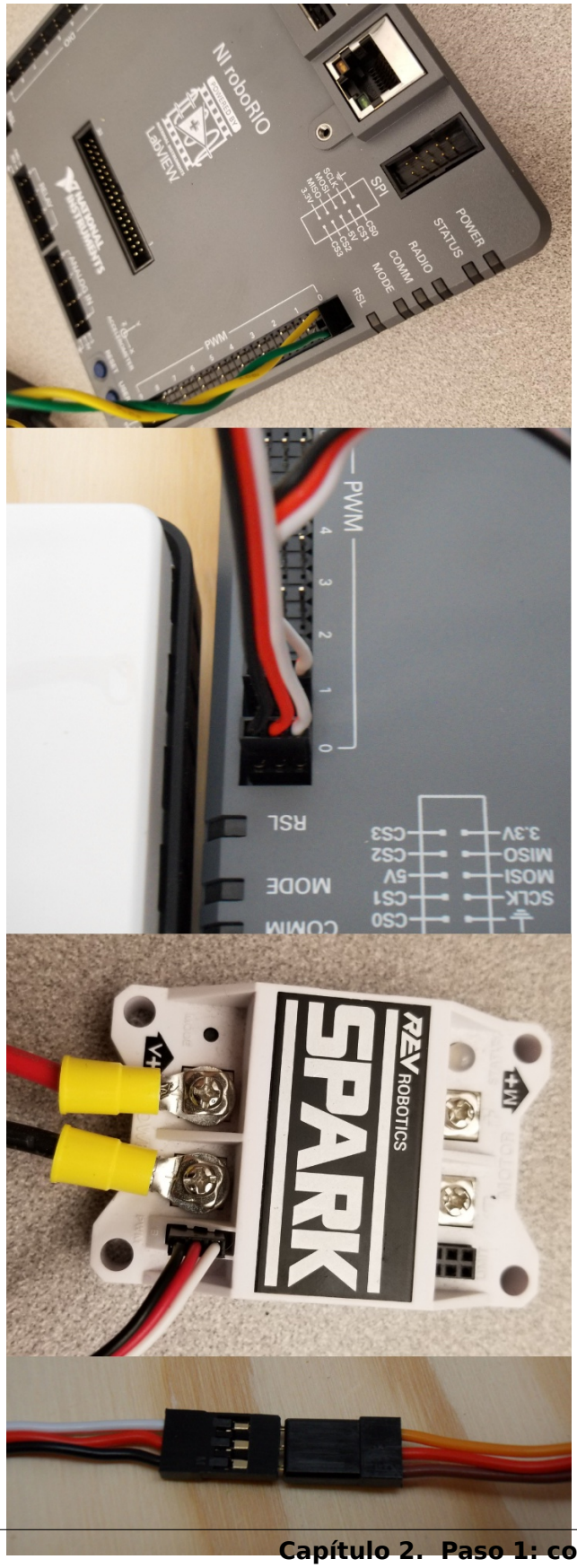
1. Strip ~5/16» (~8 mm) off of each of the CAN wires.
2. Insert the wires into the appropriate CAN terminals on the PCM.
3. Measure the length required to reach the CAN terminals of the PDP (either of the two available pairs). Cut and strip ~5/16» (~8 mm) off this end of the wires.
4. Insert the wires into the appropriate color coded CAN terminals on the PDP. You may use either of the Yellow/Green terminal pairs on the PDP, there is no defined in or out.

Nota: See the *CAN Wiring Basics* if you need to terminate the CAN bus somewhere other

than the PDP.

2.1.16 Motor Controller Signal Wires

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

This section details how to wire the SPARK MAX controllers using PWM signaling. This is a recommended starting point as it is less complex and easier to troubleshoot than CAN operation. The SPARK MAXs (and many other FRC motor controllers) can also be wired using *CAN* which unlocks easier configuration, advanced functionality, better diagnostic data and reduces the amount of wire needed.

Requires: 4x SPARK MAX PWM adapters (if using SPARK MAX), 4x PWM cables (if controllers without integrated wires or adapters, otherwise optional), 2x PWM Y-cable (Optional)

Option 1 (Direct connect):

1. If using SPARK MAX, attach the PWM adapter to the SPARK MAX (small adapter with a 3 pin connector with black/white wires).
2. If needed, attach PWM extension cables to the controller or adapter. On the controller side, match the colors or markings (some controllers may have green/yellow wiring, green should connect to black).
3. Attach the other end of the cable to the roboRIO with the black wire towards the outside of the roboRIO. It is recommended to connect the left side to PWM 0 and 1 and the right side to PWM 2 and 3 for the most straightforward programming experience, but any channel will work as long as you note which side goes to which channel and adjust the code accordingly.

Option 2 (Y-cable):

1. If using SPARK MAX, attach the PWM adapter to the SPARK MAX (small adapter with a 3 pin connector with black/white wires).
2. If needed, attach PWM extension cables between the controller or adapter and the PWM Y-cable. On the controller side, match the colors or markings (some controllers may have green/yellow wiring, green should connect to black).
3. Connect 1 PWM Y-cable to the 2 PWM cables for the controllers controlling each side of the robot. The brown wire on the Y-cable should match the black wire on the PWM cable.
4. Connect the PWM Y-cables to the PWM ports on the roboRIO. The brown wire should be towards the outside of the roboRIO. It is recommended to connect the left side to PWM 0 and the right side to PWM 1 for the most straightforward programming experience, but any channel will work as long as you note which side goes to which channel and adjust the code accordingly.

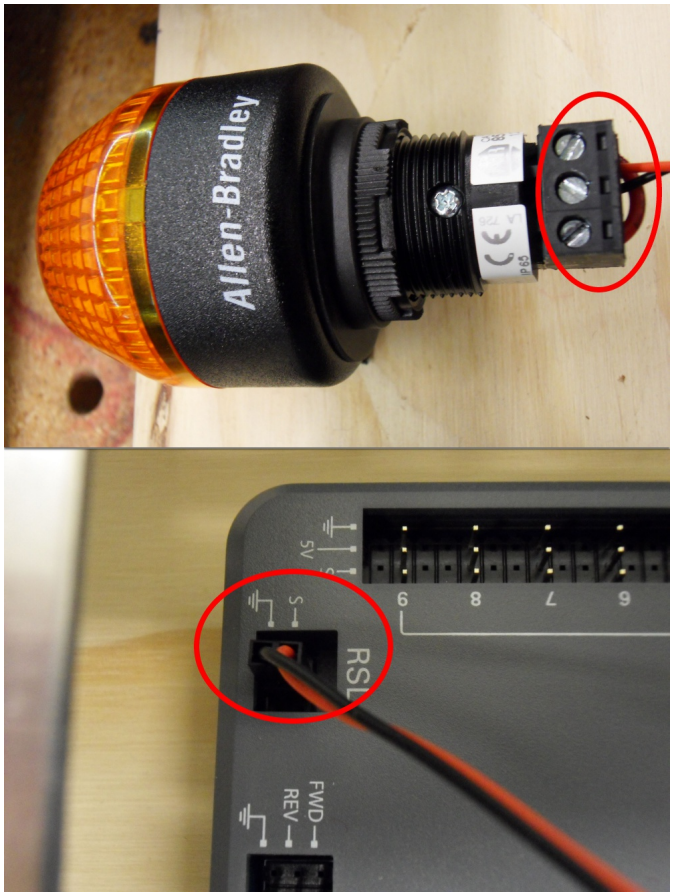
CAN

The Spark MAX controllers can also be wired using CAN. When wiring CAN the objective is to create a single complete bus running from the roboRIO on one end and running through all CAN devices on the robot. It is recommended to have either Power Distribution device at the other end of the bus because they have built-in termination. If you do not wish to locate one of these devices at the end of the bus see *CAN Wiring Basics* for info about terminating yourself.

The Spark MAX controllers come with CAN cables that are pre-terminated with connectors. You can chain these cables together directly, or buy or build extension cables to bridge larger gaps. To connect to other CAN devices such as pneumatics controllers, power distribution boards, or the roboRIO you will need to either cut off one of these pre-terminated connectors on the controller, cut off a connector on an extension, or build your own extension with just a single connector.

When chaining controllers together using the provided connectors, make sure to use the provided retaining clip. If unavailable, secure the connection with a small ziptie, electrical tape, or other similar method.

2.1.17 Robot Signal Light



Requires: Wire stripper, 2 pin cable, Robot Signal Light, 18 AWG (1 mm^2) red wire, very small flat screwdriver

1. Cut one end off of the 2 pin cable and strip both wires
2. Insert the black wire into the center, «N» terminal and tighten the terminal.
3. Strip the 18 AWG (1 mm^2) red wire and insert into the «La» terminal and tighten the terminal.
4. Cut and strip the other end of the 18 AWG (1 mm^2) wire to insert into the «Lb» terminal
5. Insert the red wire from the two pin cable into the «Lb» terminal with the 18 AWG (1 mm^2) red wire and tighten the terminal.
6. Connect the two-pin connector to the RSL port on the roboRIO. The black wire should be closest to the outside of the roboRIO.

Truco: You may wish to temporarily secure the RSL to the control board using cable ties or Dual Lock (it is recommended to move the RSL to a more visible location as the robot is being

constructed)

2.1.18 Circuit Breakers

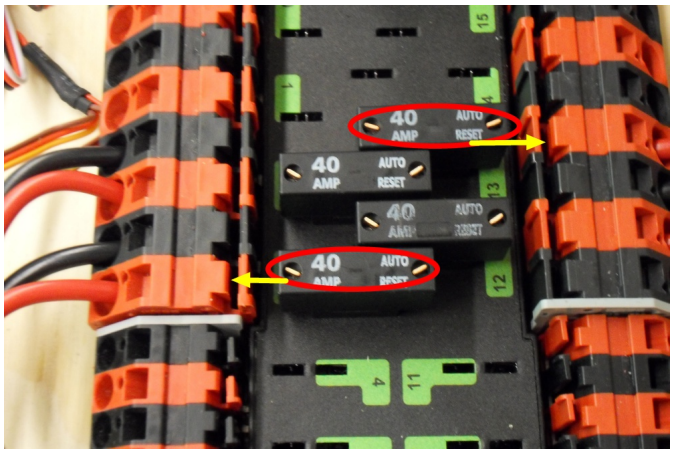
REV

Requires: 4x 40A circuit breakers

Insert 40-amp Circuit Breakers into the positions on the PDH corresponding with the Wago connectors the motor controllers are connected to. Note that the white graphic indicates which breakers are associated with which terminal pairs.

If working on a Robot Quick Build, stop here and insert the board into the robot chassis before continuing.

CTR

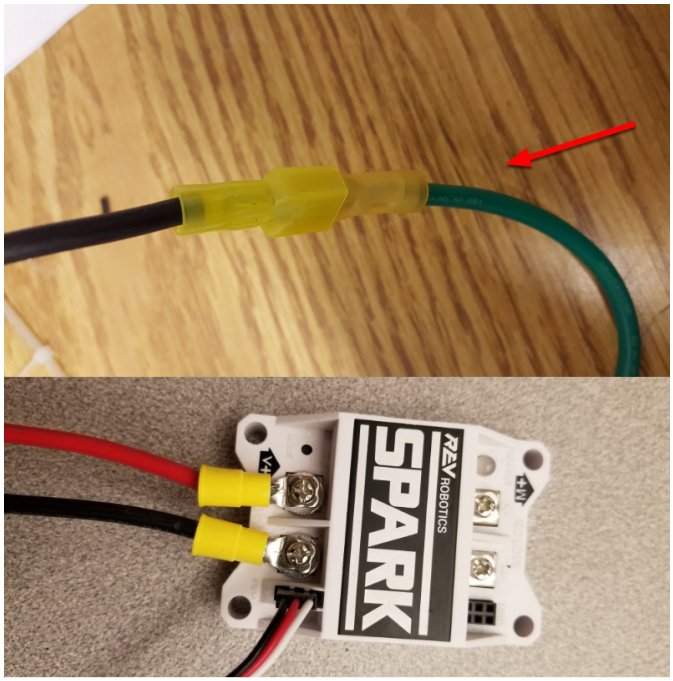


Requires: 4x 40A circuit breakers

Insert 40-amp Circuit Breakers into the positions on the PDP corresponding with the Wago connectors the motor controllers are connected to. Note that, for all breakers, the breaker corresponds with the nearest positive (red) terminal (see graphic above). All negative terminals on the board are directly connected internally.

If working on a Robot Quick Build, stop here and insert the board into the robot chassis before continuing.

2.1.19 Motor Power



Requires: Wire stripper, wire crimper, phillips head screwdriver, wire connecting hardware

For each *CIM* motor:

- Strip the ends of the red and black wires from the CIM

For integrated wire controllers including SPARK MAX (top image):

1. Strip the red and black wires (or white and green wires) from the controller (the SPARK MAX white wire is unused for brushed motors such as the CIM, it should be secured and the end should be insulated such with electrical tape or other insulation method).
2. Connect the motor wires to the matching controller output wires (for controllers with white/green, connect red to white and green to black). The images above show an example using quick disconnect terminals which are provided in the Rookie KOP.

For the SPARK or other non-integrated-wire controllers (bottom image):

1. Crimp a ring/fork terminal on each of the motor wires.
2. Attach the wires to the output side of the motor controller (red to +, black to -)

2.1.20 STOP

Peligro: Before plugging in the battery, make sure all connections have been made with the proper polarity. Ideally have someone that did not wire the robot check to make sure all connections are correct.

- Start with the battery and verify that the red wire is connected to the positive terminal
- Check that the red wire passes through the main breaker and to the + terminal of the PDP and that the black wire travels directly to the - terminal.
- For each motor controller, verify that the red wire goes from the red PDP terminal to the V+ terminal on the motor controller (not M+!!!!)
- For each non-motor controller device, verify that the red wire runs from a red terminal on the PD connects to a red terminal on the component.
- Make sure that the PoE cable is plugged directly into the radio NOT THE roboRIO!

Truco: It is also recommended to put the robot on blocks so the wheels are off the ground before proceeding. This will prevent any unexpected movement from becoming dangerous.

2.1.21 Manage Wires

Requires: Zip ties

Truco: Now may be a good time to add a few zip ties to manage some of the wires before proceeding. This will help keep the robot wiring neat.

2.1.22 Connect Battery

Connect the battery to the robot side of the Anderson connector. Power on the robot by moving the lever on the top of the 120A main breaker into the ridge on the top of the housing.

If stuff blinks, you probably did it right. If you hear any clicking, or see any smoke, power the system off immediately, clicking is likely the sound of circuit breakers tripping.

Before moving on, if using SPARK MAX controllers, there is one more configuration step to complete. The SPARK MAX motor controllers are configured to control a brushless motor by default. You can verify this by checking that the light on the controller is blinking either cyan or magenta (indicating brushless brake or brushless coast respectively). To change to brushed mode, press and hold the mode button for 3-4 seconds until the status LED changes color. The LED should change to either blue or yellow, indicating that the controller is in brushed mode (brake or coast respectively). To change the brake or coast mode, which controls how quickly the motor slows down when a neutral signal is applied, press the mode button briefly.

Truco: For more information on the SPARK MAX motor controllers, including how to test your motors/controllers without writing any code by using the REV Hardware Client, see the SPARK MAX Quickstart guide.

From here, you should connect to the roboRIO and try uploading your code!

3 Paso 2: instalación del software

Puede encontrar un resumen del software del sistema de control disponible *aquí*.

3.1 Preparación de Instalación Sin Conexión

Este artículo contiene instrucciones y enlaces a los componentes que querrá recopilar si necesita realizar una instalación sin conexión del software FRC® Control System.

Truco: Este documento recopila todos los enlaces de descarga de los siguientes documentos para facilitar la instalación en computadoras sin conexión o en múltiples computadoras. Si lo está instalando en un solo ordenador que está conectado a Internet, puede saltarse esta página.

Nota: El orden de instalación de estas herramientas no importa para los equipos Java y C++. LabVIEW debe instalarse antes que las Herramientas de Juego FRC u otra librería de 3eros.

3.1.1 Documentación

This documentation can be downloaded for offline viewing. The link to download the PDF can be found here.

3.1.2 Instaladores

Todos los equipos

- 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

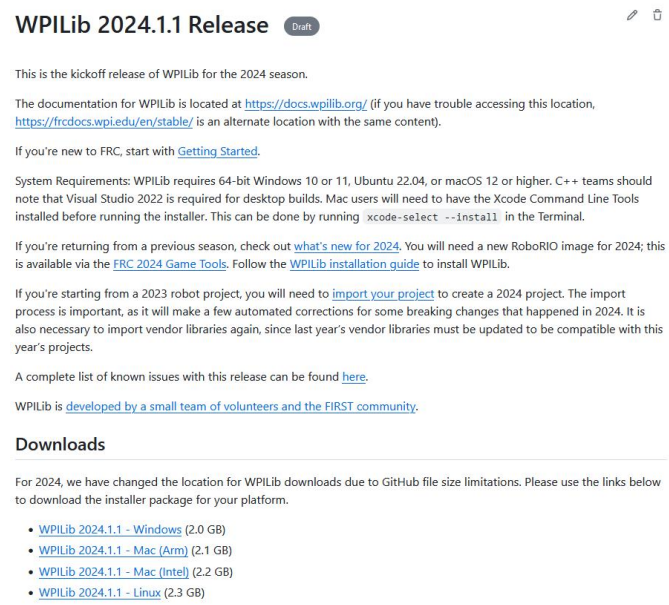
Equipos de LabVIEW

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

Equipos Java/C++

- Java/C++ WPILib Installer

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



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

Nota: After downloading the Java/C++ WPILib installer, run it once while connected to the internet and select *Install for this User* then *Create VS Code zip to share with other computers/OSes for offline install* and save the downloaded VS Code zip file for future offline installations.

3.1.3 Bibliotecas / Software de 3eros

Puede encontrar el directorio disponible de software de terceros que conecta con WPILib en *Bibliotecas de 3ros*.

3.2 Instalación LabVIEW para FRC (únicamente LabVIEW)

Nota: 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*.

Los tiempos de descarga e instalación variarán ampliamente según las especificaciones de conexión a Internet y computadora, sin embargo, tenga en cuenta que este proceso implica una descarga e instalación de archivos de gran tamaño y probablemente tardará al menos una hora en completarse.

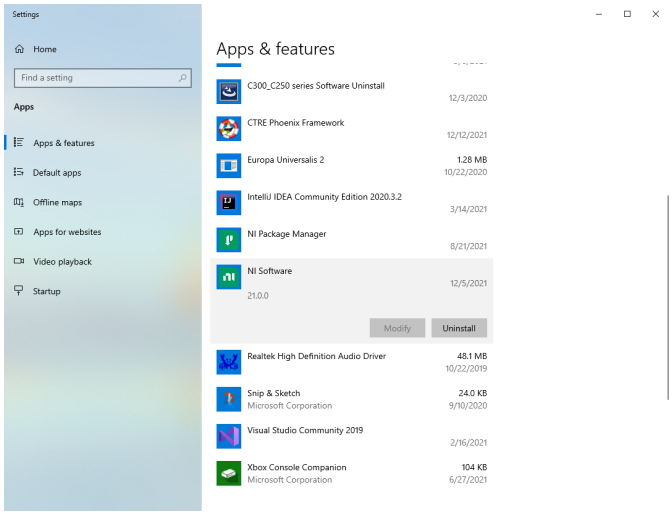
3.2.1 Requirements

- Windows 10 or higher (Windows 10, 11)

3.2.2 Desinstalar Versiones Anteriores (recomendado)

Nota: Si usted desea seguir programando cRIOs necesitará mantener una instalación de LabVIEW para FRC® 2014. La licencia de LabVIEW para FRC 2014 se ha extendido. Mientras estas versiones deberían poder co-existir en una sola computadora, esta no es configuración que ha sido extensivamente probada.

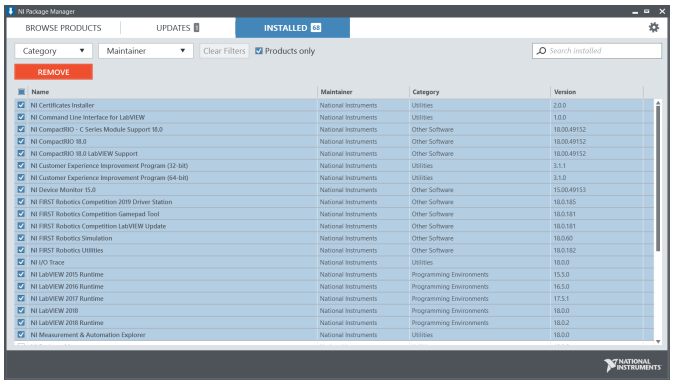
Before installing the new version of LabVIEW it is recommended to remove any old versions. The new version will likely co-exist with the old version, but all testing has been done with FRC 2024 only. Make sure to back up any team code located in the «User\LabVIEW Data» directory before un-installing.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



Seleccionar Componentes para Desinstalar

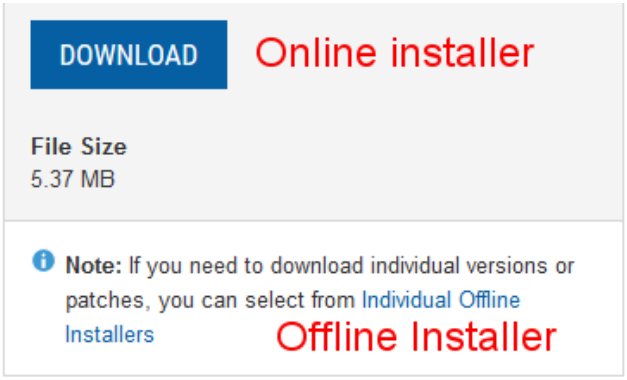
En la caja de diálogo que aparece, seleccione todas las entradas. La forma más sencilla de hacer esto es desmarcar la viñeta de verificación “Solo productos” y seleccionar la viñeta a la izquierda llamada “Nombre”. Haga clic en remover. Espere a que la desinstalación se complete y reinicie si se le solicita.

> **Advertencia:** Estas instrucciones asumen que ningún otro software National Instruments está instalado. Si tiene otro software National Instruments instalado, es necesario desmarcar el software que no debe desinstalarse.



3.2.3 Obtención del Instalador LabVIEW

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



Si desea instalarlo en otro ordenador sin conexión, no haga clic en el botón descargar, haga clic en **Instaladores Individuales Fuera de Línea** y después haga clic en descargar, para descargar el instalador completo.

Nota: 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 Instalando LabVIEW

National Instruments LabVIEW requiere una licencia. La licencia de cada temporada permanece activa hasta el 31 de enero del siguiente año (por ejemplo, la licencia de la temporada 2020 expira el 31 de enero del 2021)

Los equipos tienen permitido instalar el software en todos las computadoras que el equipo necesite, sujeto a las restricciones y términos de licencia que acompañan al software aplicable, y siempre y cuando sólo los miembros del equipo o mentores utilicen el software, y únicamente para FRC. Los derechos para usar LabVIEW se rigen únicamente por los términos de los acuerdos de licencia que se muestran durante la instalación del software aplicable.

Iniciando Instalación

Instalador En Línea

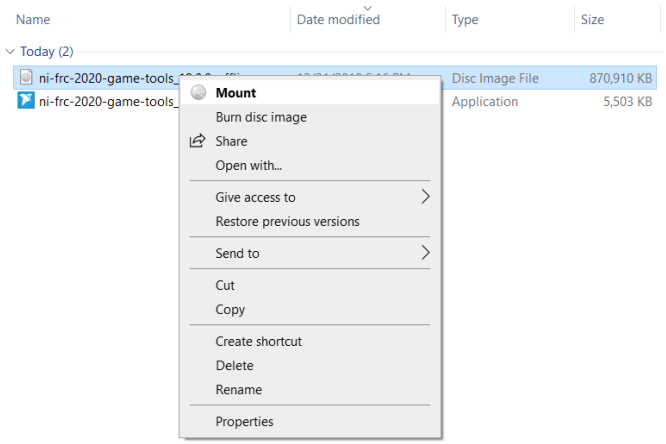
Ejecute el archivo exe descargado para iniciar la instalacion. Haga clic en *Si* si aparece un mensaje de seguridad de windows

Offline Installer (Windows 10+)

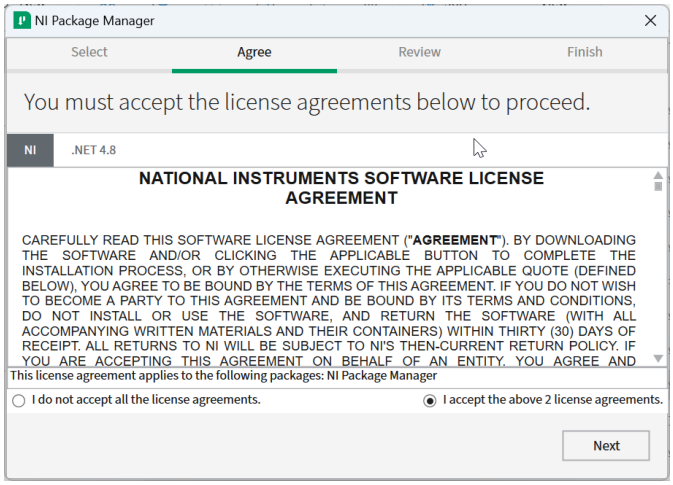
Haga clic derecho en el archivo iso descargado y seleccione montar. Ejecute install.exe desde la iso montada. Haga clic en «Sí» si aparece un mensaje de seguridad de Windows

---

Nota: 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 install 7-Zip and use that to extract the iso.

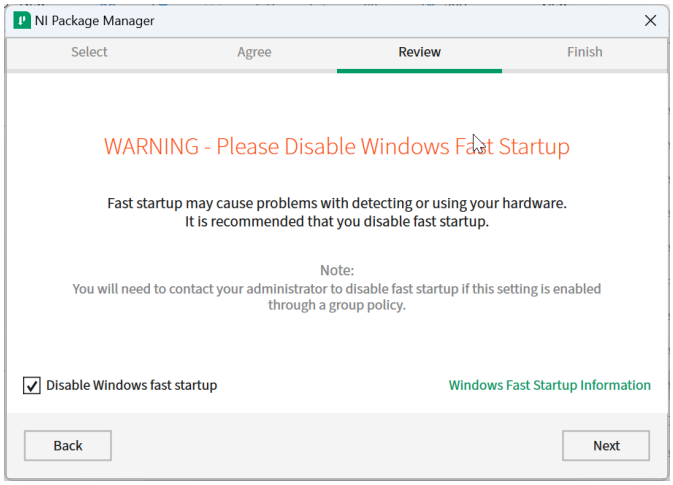


Licencia de Administrador de Paquetes de National Instruments



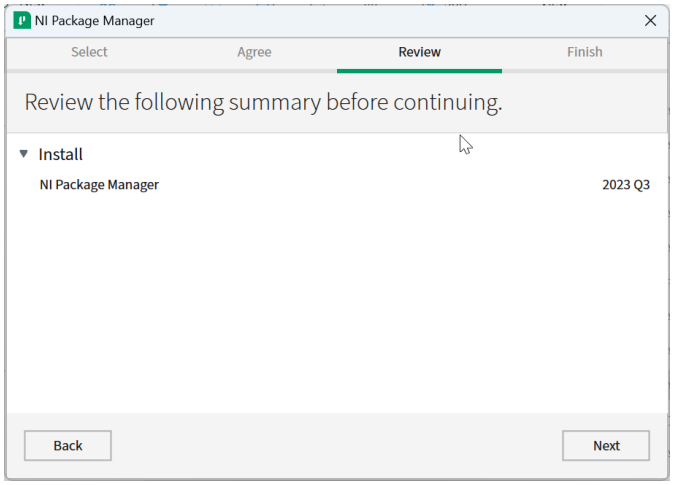
Si ve esta pantalla, haga clic en *Siguiente*

Deshabilitar el Inicio Rápido de Windows



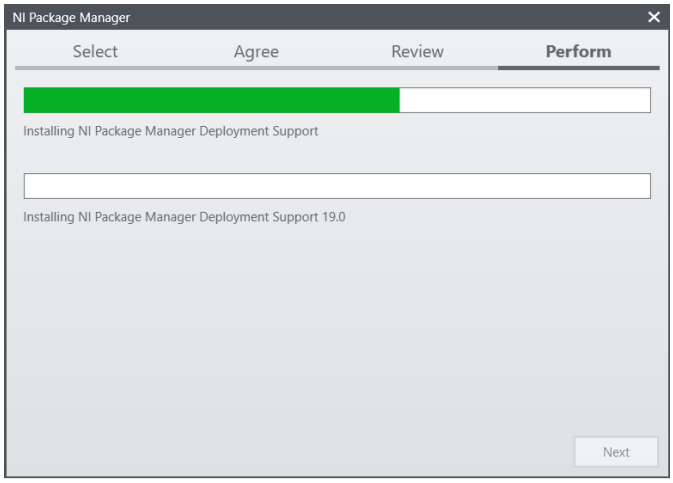
Si ve esta pantalla, haga clic en *Siguiente*

Revisión de Administrador de Paquetes de National Instruments



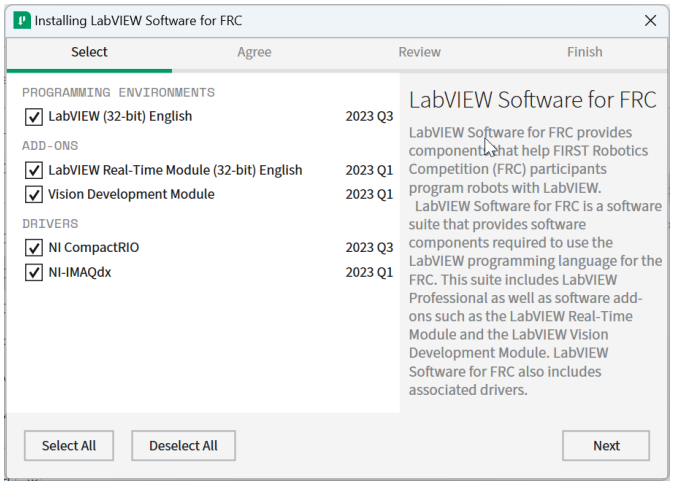
Si ve esta pantalla, haga clic en *Siguiente*

Instalación de Administrador de Paquetes de National Instruments



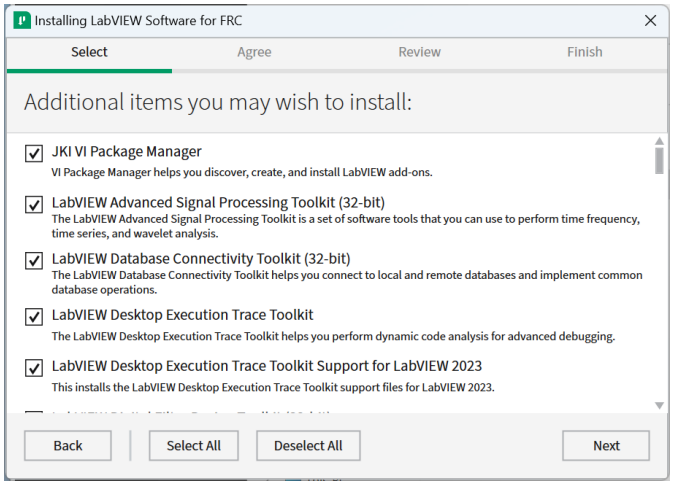
El proceso de instalación de Administrador de Paquetes de National Instruments se rastreará en esta ventana.

Lista de Productos



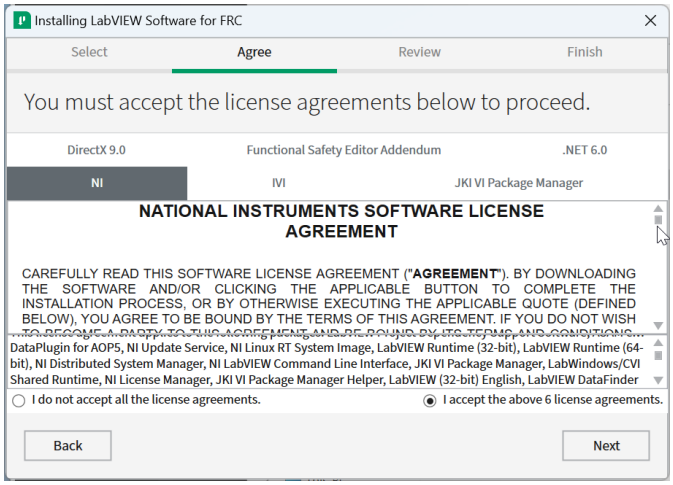
Haga clic en *Siguiente*

Paquetes Adicionales



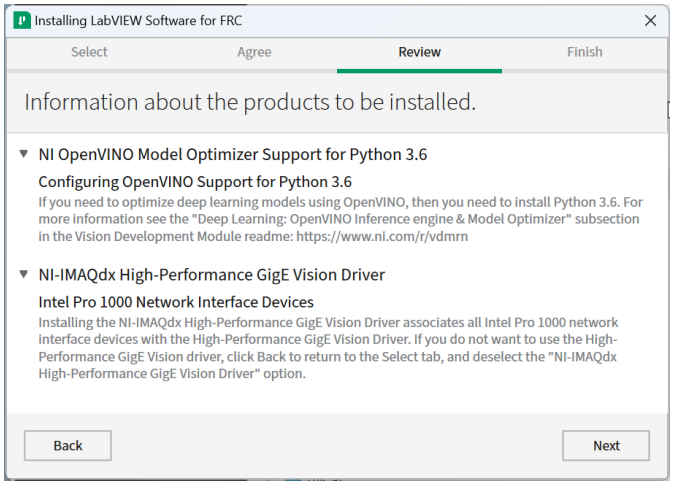
Haga clic en *Siguiente*

Acuerdos de Licencia



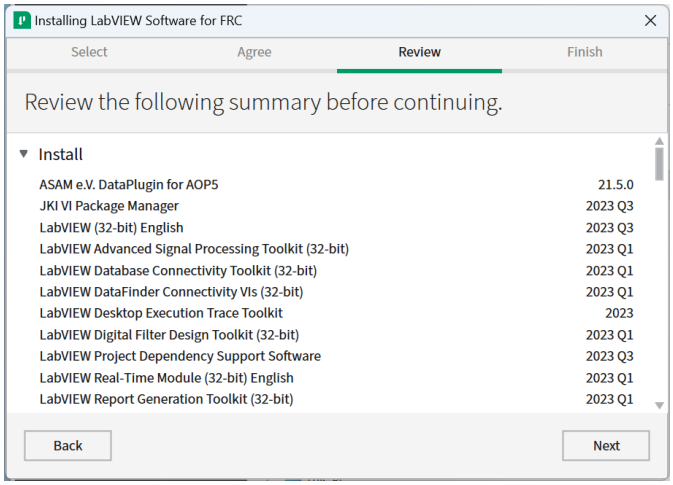
Marque “Acepto…” y después haga clic en *Siguiente*

Información del Producto



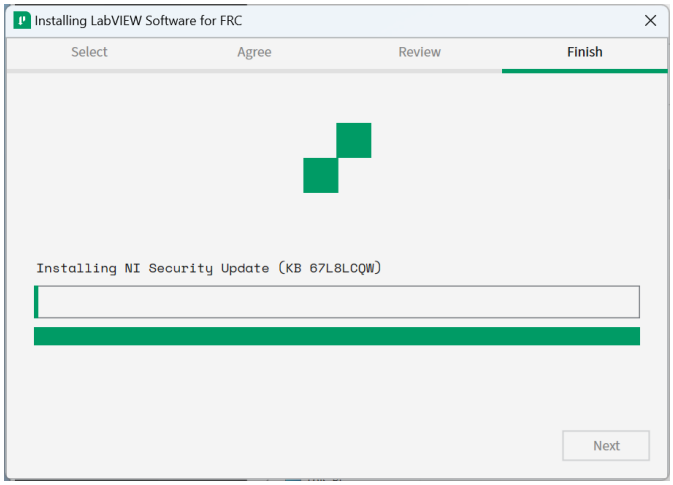
Haga clic en *Siguiente*

Comienzo de Instalación



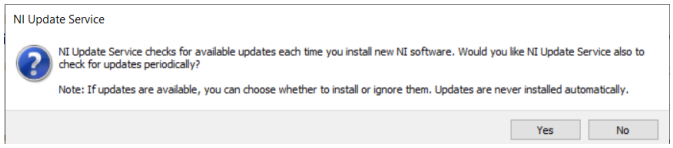
Haga clic en *Siguiente*

Progreso General



El progreso general de la instalación se realizará en esta ventana.

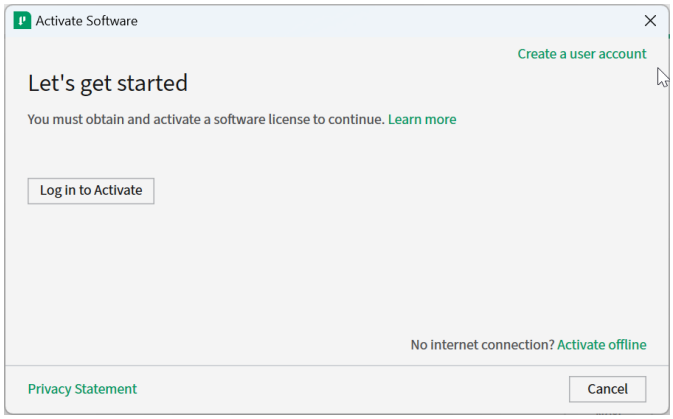
3.2.5 Servicio de Actualización National Instruments



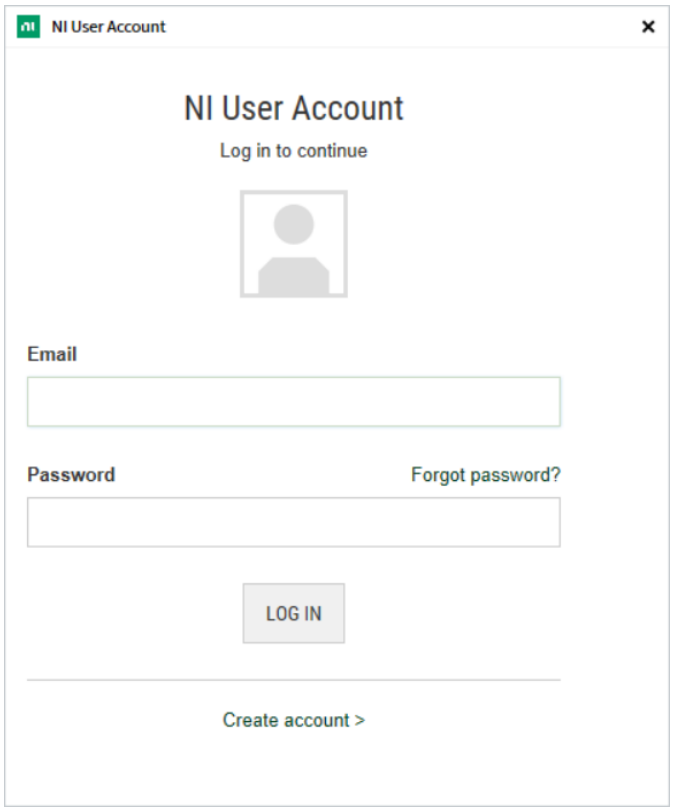
Se le preguntará si desea habilitar el servicio de actualización de National Instruments. Puede optar por no habilitar el servicio de actualización.

Advertencia: No es recomendado instalar estas actualizaciones a menos que FRC lo indique a través de nuestros canales de comunicación habituales (Blog de FRC, Actualizaciones de Equipo o Explosiones de Correo Electrónico).

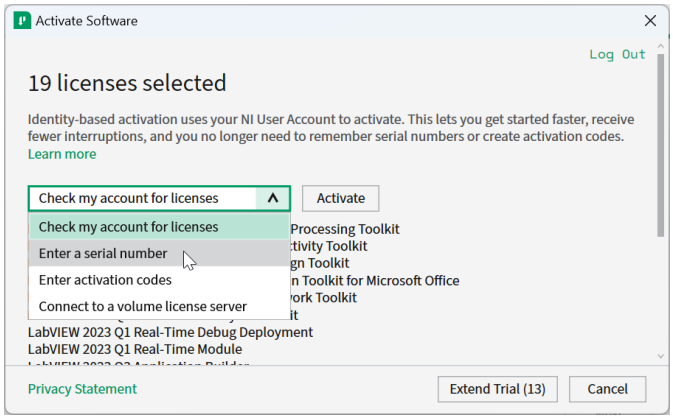
Asistente de Activación de National Instruments



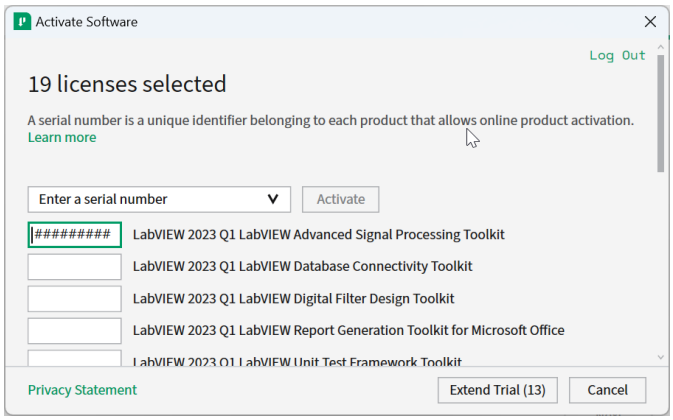
Click the *Log in to Activate* button.



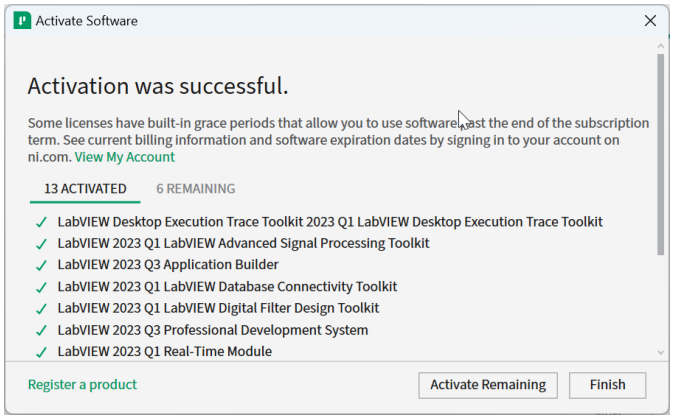
Ingrese en su cuenta ni.com. Si no tiene cuenta, seleccione *Crear cuenta* para crear una cuenta gratis.



From the drop-down, select enter a serial number

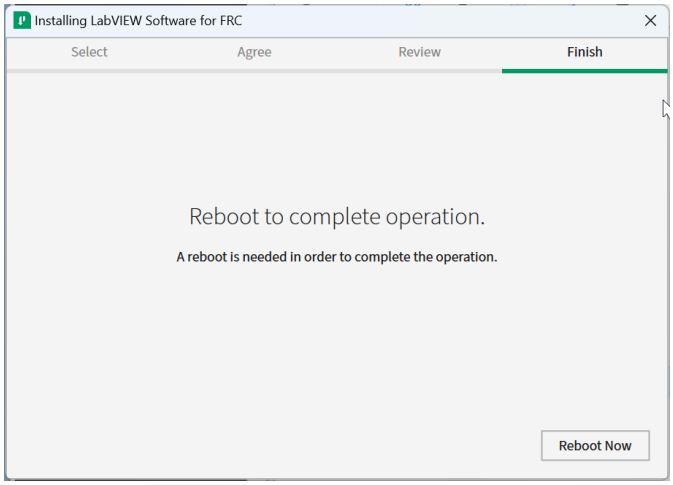


Enter the serial number in all the boxes. Click *Activate*.



If your products activate successfully, an «Activation Successful» message will appear. If the serial number was incorrect, it will give you a text box and you can re-enter the number and select *Try Again*. The items shown above are not expected to activate. If everything activated successfully, click *Finish*.

Reiniciar



Seleccione *Reinicie ahora* después de cerrar cualquier programa abierto.

3.3 Instalando FRC Game Tools

El FRC® Game Tools contiene los siguientes componentes de software:

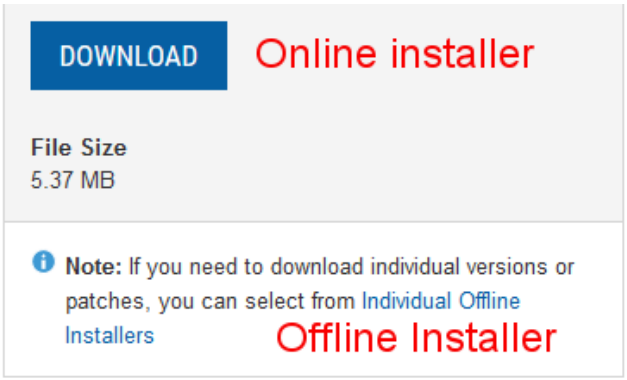
- Actualización de LabVIEW
- FRC Driver Station
- FRC roboRIO Imaging Tool and Images

The LabVIEW runtime components required for the Driver Station and Imaging Tool are included in this package.

Nota: No components from the LabVIEW Software for FRC package are required for running either the Driver Station or Imaging Tool.

3.3.1 Requerimientos:

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



Si desea instalarlo en otros aparatos de manera offline, haga click en *Individual Offline Installers* antes de pulsar *Download* para descargar el instalador completo.

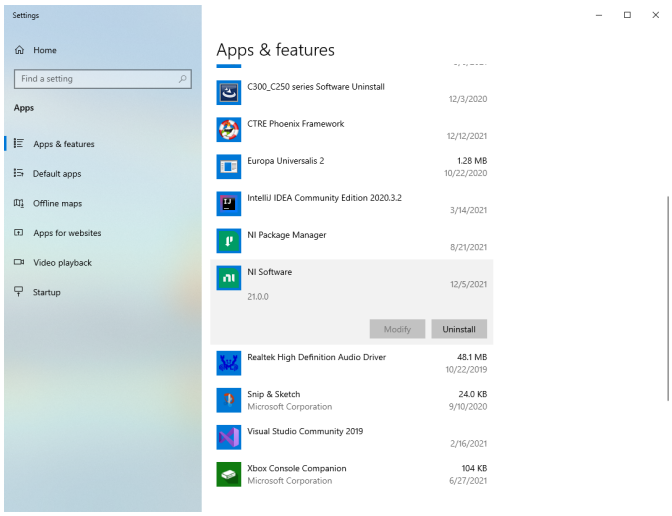
3.3.2 Desinstalar Versiones Anteriores (Recomendado)

Importante: Los equipos LabVIEW ya han completado este paso, no lo repitan. Los equipos LabVIEW deben saltar a la sección *Instalación*.

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*.

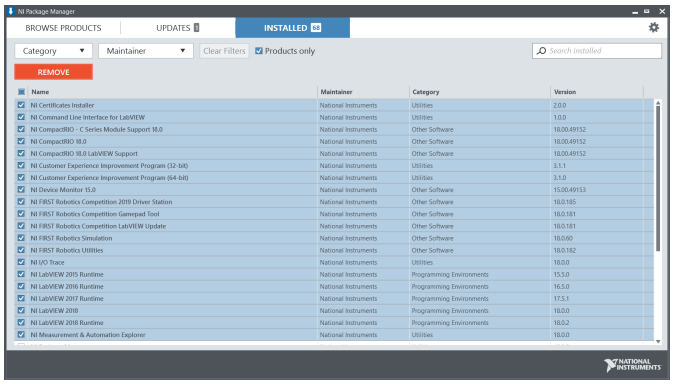
Nota: It is only necessary to uninstall previous versions when installing a new year's tools (or when a beta is installed). For example, uninstall the 2021 tools before installing the 2022

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



Seleccionar Componentes para Desinstalar

En la caja de dialogo que aparece, seleccione todas las entradas. La manera más fácil de hacer esto es de-seleccionar la opción de *Products Only* y seleccione la opción a la izquierda de «Name». De click en *Remove*. Espere a que el desinstalador complete y reinicie si así se desea.



3.3.3 Instalación

Importante: El Game Tools installer puede solicitar que .NET Framework 4.6.2 deba actualizarse o instalarse. Siga las instrucciones en pantalla para completar la instalación, incluido el reinicio si se requiere. Luego, reanude la instalación de FRC Game Tools, reinicie el instalador si es necesario.

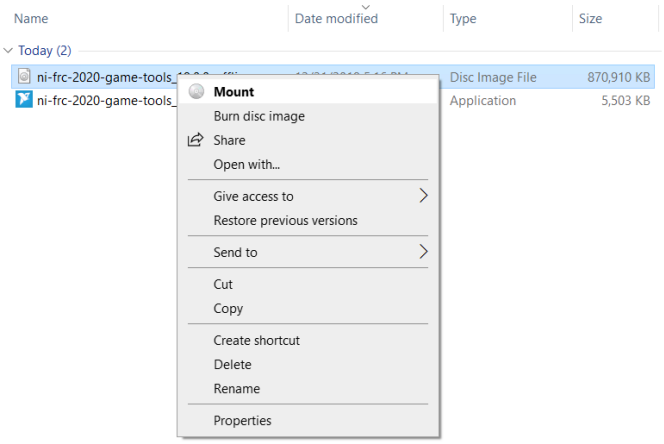
Extracción

Online

Ejecute el archivo ejecutable que descargó para empezar el proceso de instalación. Haga click en *Yes* si una ventana de Windows Security aparece.

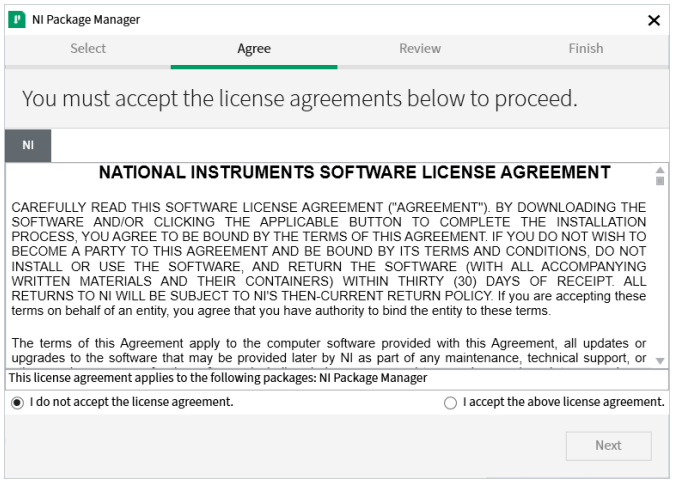
Offline (Windows 10+)

Haga clic derecho en el archivo iso descargado y seleccione *mount*. Ejecute `install.exe` desde el iso montado. Haga clic en *Yes* si una ventada de Windows Security aparece.



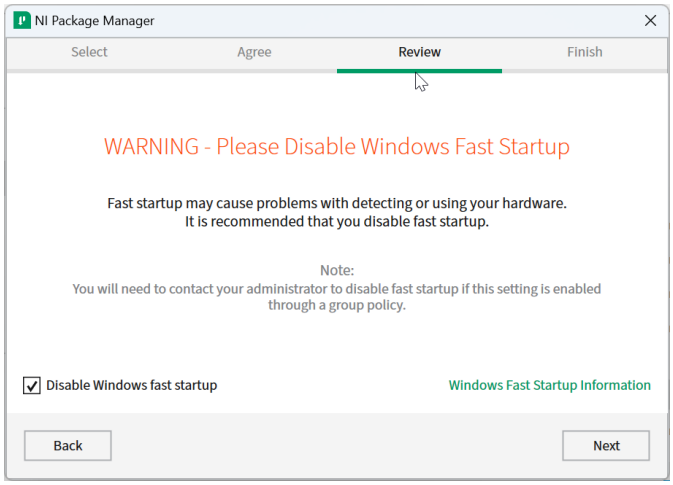
Nota: Other installed programs may associate with iso files and the *mount* option may not appear. If that software does not give the option to mount or extract the iso file, then install 7-Zip and use that to extract the iso.

Licencia de NI Package Manager



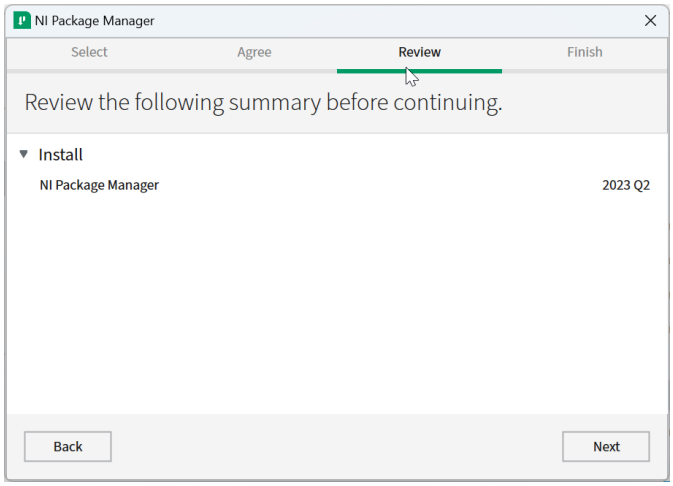
Si ve esto en su pantalla, haga clic en *Next*. Esta pantalla confirma que está de acuerdo con la licencia del NI Package Manager.

Desactivar el inicio Rápido de Windows



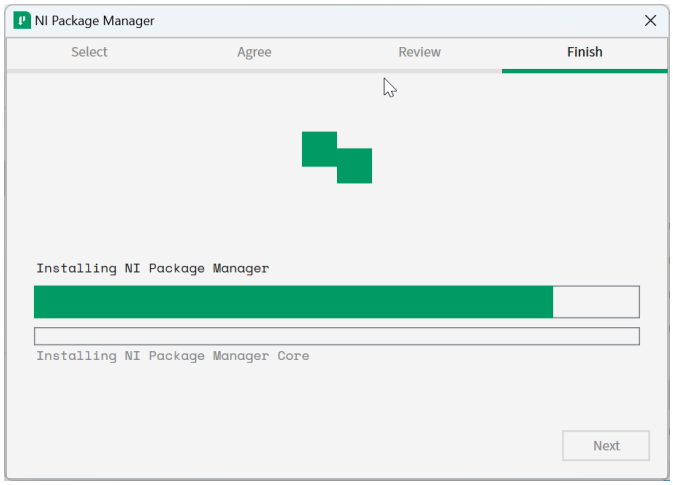
Es recomendable dejar la pantalla tal como está, ya que Windows Fast Startup puedae causar problemas con los drivers de NI necesarios para la transcripción de la roboRIO. Siga y de clic a *Next*.

Revisión de NI Package Manager



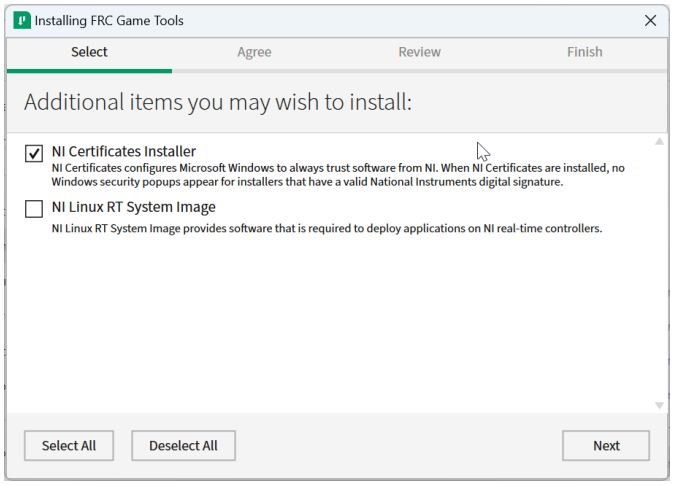
Si ve esta pantalla, de clic en *Next*.

Instalación de NI Package Manager



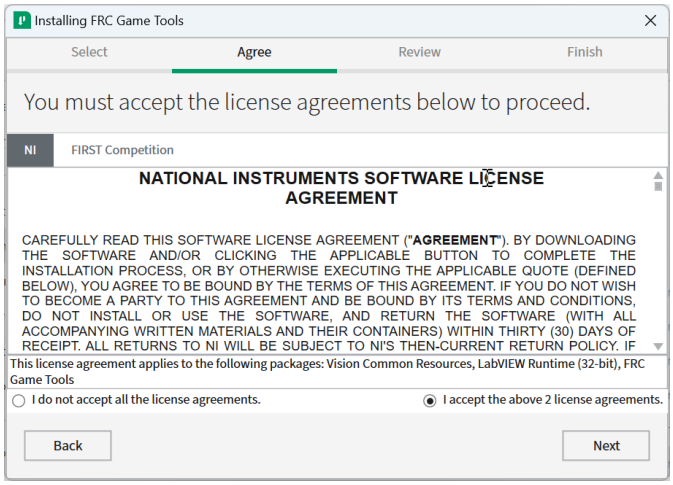
El proceso de instalación del NI Package Manager será rastreado a través de esta ventana.

Software Adicional



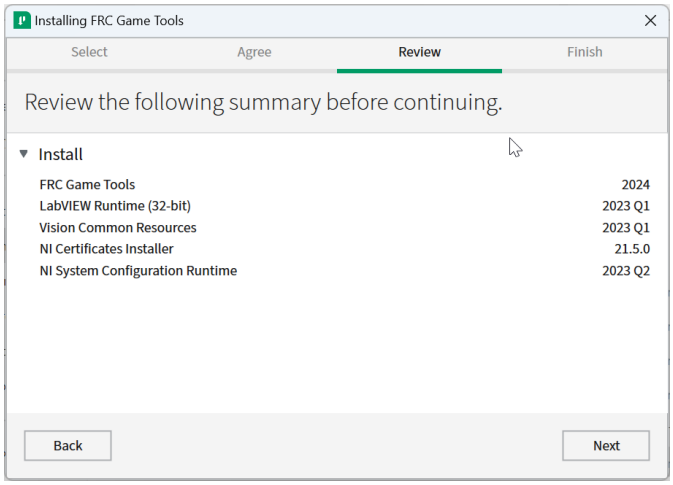
Si ve esta pantalla, de clic en *Next*.

Acuerdos de Licencia



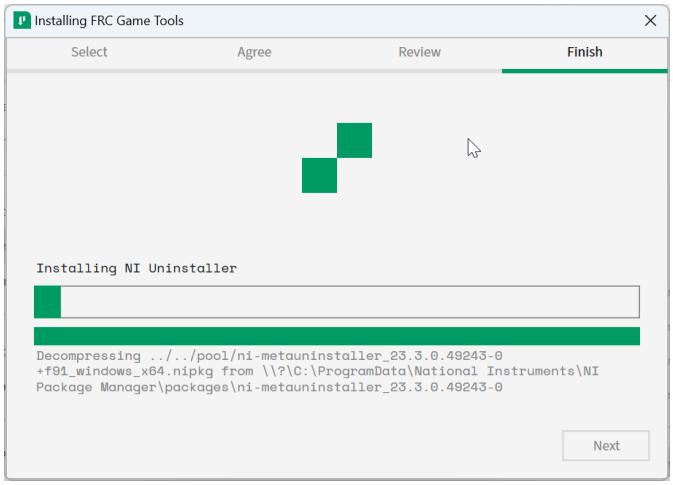
Seleccione *I accept...* y de clic en *Next*

Resumen de la Revisión



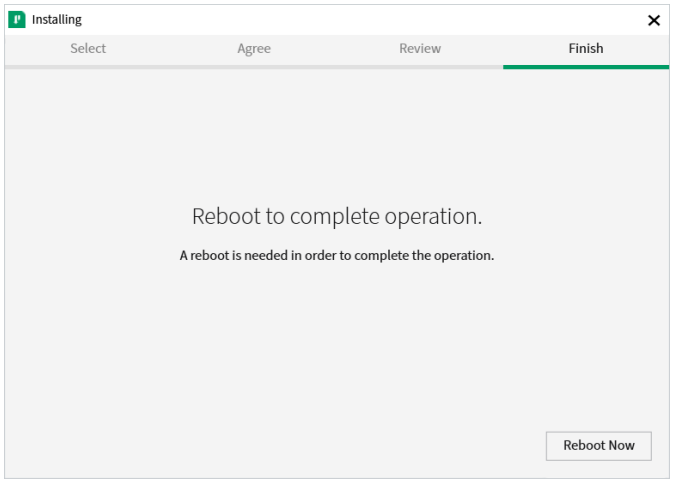
Clic *Next*.

Progreso Detallado



Esta pantalla muestra el proceso de instalación, continué y presione *Next* cuando esté terminado.

3.3.4 Reiniciar para Completar la Instalación



Si así se incita, seleccione *Reboot Now* después de cerrar cualquier programa abierto

3.4 Guía de Instalación WPILib

This guide is intended for Java and C++ teams. LabVIEW teams can skip to *Instalación LabVIEW para FRC (únicamente LabVIEW)*. Python teams can skip to *Python Installation Guide*. 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 Prerrequisitos

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.

> **Advertencia:** 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 is designed to install to different folders for different years, so that it is not necessary to uninstall a previous version before installing this year's WPILib.

3.4.2 Downloading

WPILib Installer

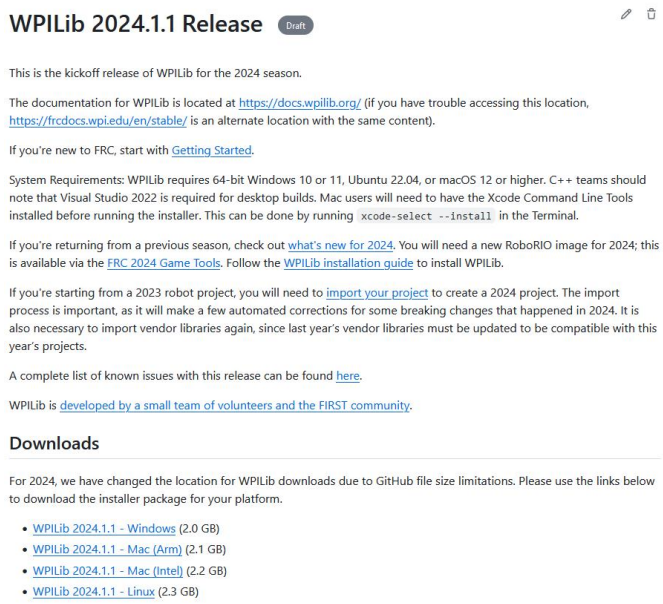
WPILib 2024.3.2 Release - March 14, 2024 Downloads

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)

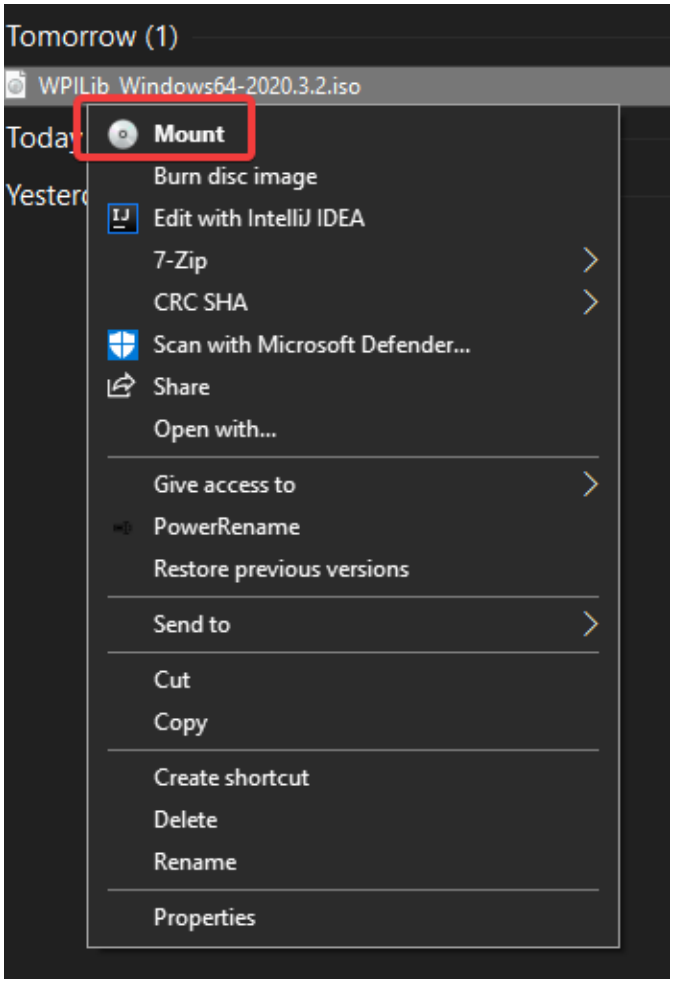
Then click on the correct binary for your OS and architecture to begin the download.

3.4.3 Extrayendo el Instalador

Cuando descarga el instalador de WPILib, este está distribuido como un archivo de imagen de disco `.iso` para Windows, `.tar.gz` para Linux, y distribuido como `DMG` para MacOS.

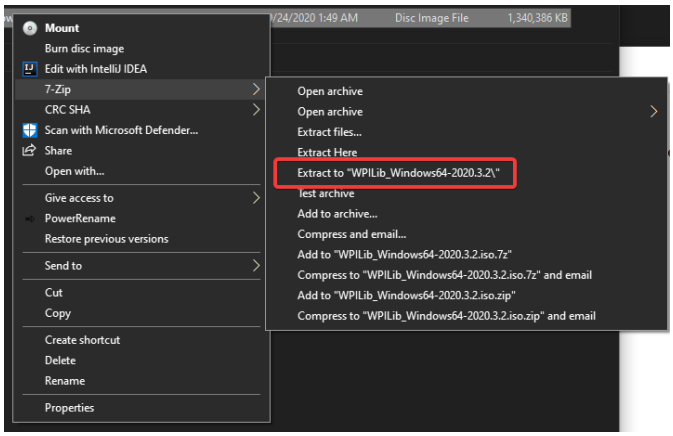
Windows 10+

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



Nota: 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`.

Nota: 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.

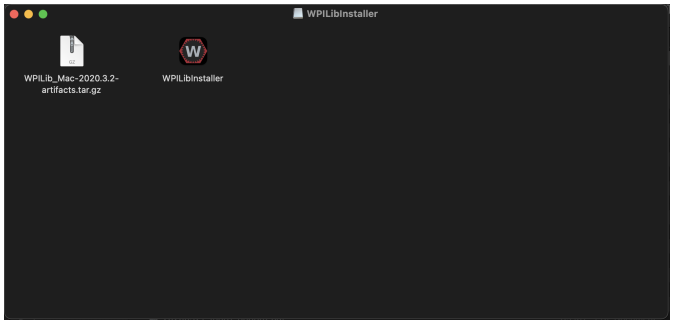
---

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.

Importante: 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.

Los usuarios de macOS pueden dar doble clic en el `DMG` descargado y seleccionar `WPILibInstaller` para abrir la aplicación.



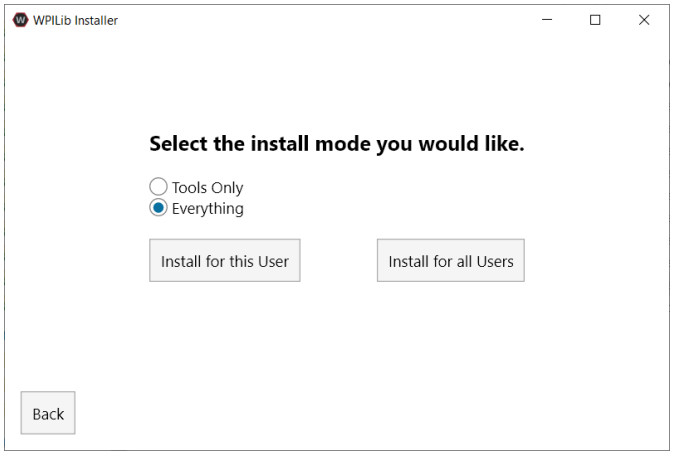
Linux

Los usuarios de Linux deben extraer el archivo `.tar.gz` descargado y luego ejecutar `WPILibInstaller`. Ubuntu trata los ejecutables en el explorador de archivos como bibliotecas compartidas, por lo que hacer doble clic no los ejecutará. Ejecute los siguientes comandos en una terminal en lugar de `<version>` reemplazado por la versión que está instalando.

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

3.4.4 Corriendo el Instalador

Al abrir el instalador, se le presentará la siguiente pantalla. Presione *Start*.



This showcases a list of options included with the WPILib installation.

- *Tools Only* installs just the WPILib tools (Pathweaver, Shuffleboard, RobotBuilder, SysId, Glass, and OutlineViewer) and JDK.
- *Everything* installs the full development environment (VS Code, extensions, all dependencies), WPILib tools, and JDK.

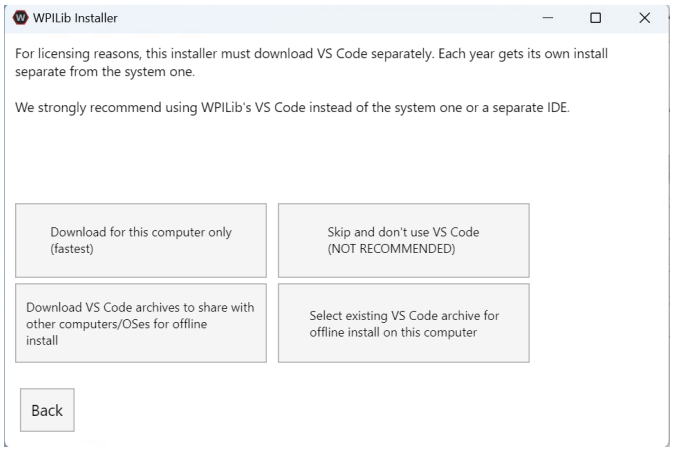
Notará dos botones, *Install for this User* y *Install for all Users*. *Install for this User* solo se instala en la cuenta de usuario actual, y no requiere privilegios de administrador. Sin embargo,

Install for all Users instala las herramientas para todas las cuentas del sistema y *requerirá* acceso de administrador. *Install for all Users* no es una opción para macOS y Linux.

Nota: If you select Install for all Users, Windows will prompt for administrator access through UAC during installation.

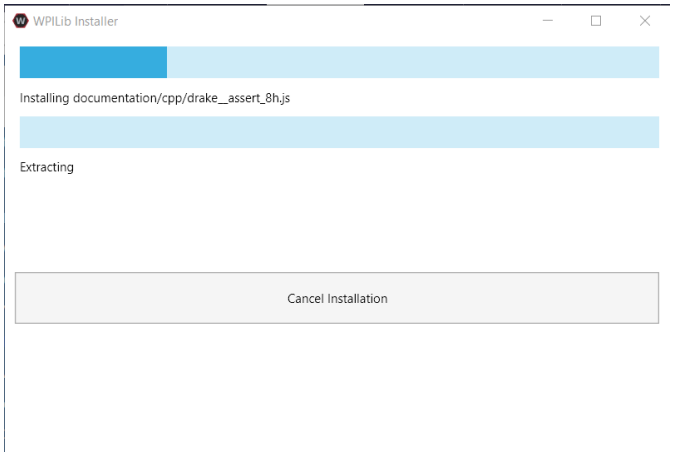
Seleccione la opción que sea adecuada para usted y se le presentará la siguiente pantalla de instalación.

La siguiente pantalla implica la descarga de VS Code. Desafortunadamente, debido a razones de licencia, VS Code no se puede incluir con el instalador.

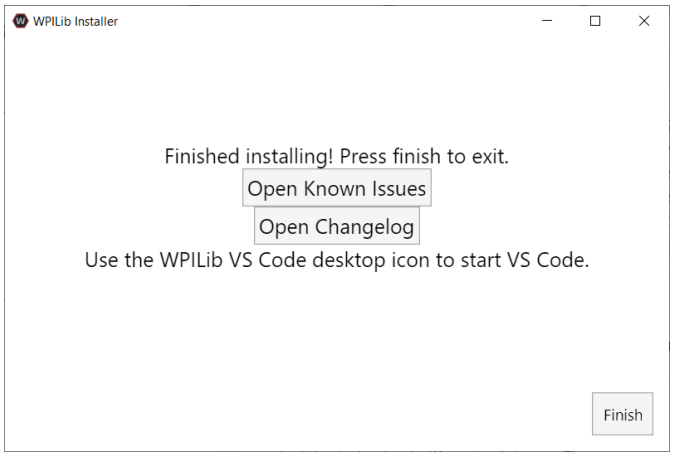


- Download for this computer only
  - Esto descarga VS Code solo para la plataforma actual, que es también la descarga más pequeña.
- Skip and don’t use VS Code
  - Omitir la instalación de VS Code. Útil para instalaciones o configuraciones avanzadas. Generalmente no es recomendado.
- Select existing VS Code archive for offline install on this computer
  - Al seleccionar esta opción, aparecerá un mensaje que le permitirá seleccionar un archivo zip preexistente de VS Code que ha sido descargado previamente por el instalador. Esta opción **no** le permite seleccionar una copia ya instalada de VS Code en su máquina.
- Create VS Code archives to share with other computers/OSes for offline install
  - Esta opción descarga y guarda una copia de VS Code para todas las plataformas, que es útil para compartir la copia del instalador.

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.



Una vez completada la instalación, se le presentará la pantalla finalizada.



Importante: WPILib installs a separate version of VS Code. It does not use an already existing installation. Each year has it's own copy of the tools appended with the year. IE: `WPILib VS Code 2022`. Please launch the WPILib VS Code and not a system installed copy!

¡Felicitaciones, el entorno de desarrollo y las herramientas de WPILib ya están instalados en su computadora! Presione Finalizar para salir del instalador.

3.4.5 Post-Instalación

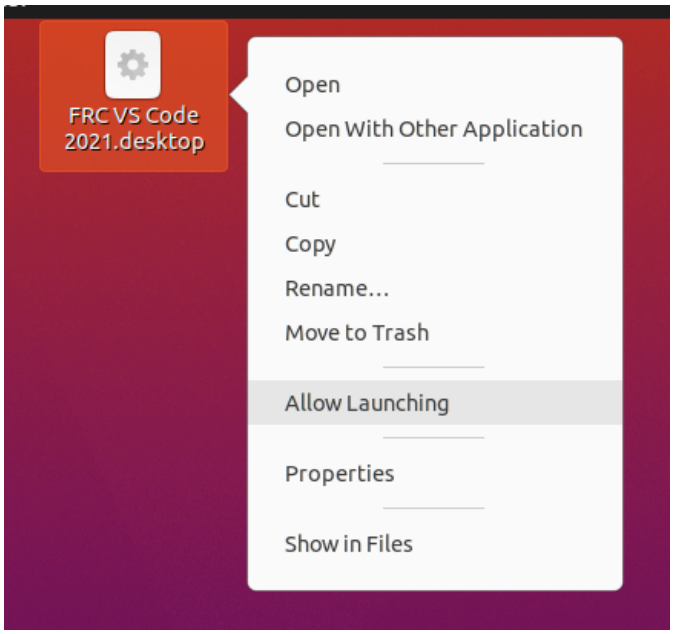
Algunos sistemas operativos requieren acciones finales para completar la instalación.

macOS

Después de la instalación, el instalador abre la carpeta WPILib VS Code. Arrastre la aplicación VS Code al dock. Expulsar la imagen de WPILibInstaller del escritorio.

Linux

Algunas versiones de Linux (por ejemplo, Ubuntu 20.04) requieren que le des al acceso directo del escritorio la capacidad de iniciarse. Haga clic derecho en el icono del escritorio y seleccione Permitir lanzamiento.

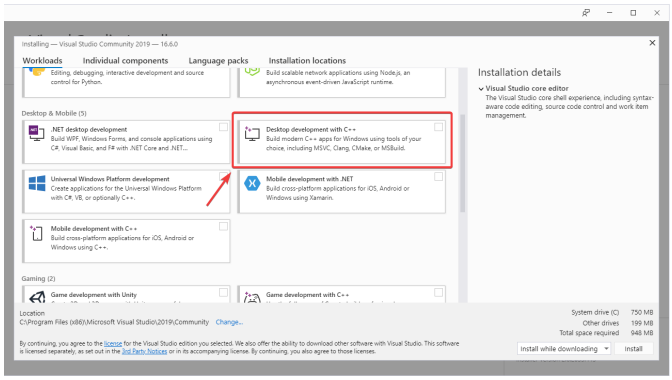


Nota: Installing desktop tools and rebooting will create a folder on the desktop called `YYYY WPILib Tools`, where `YYYY` is the current year. Desktop tool shortcuts are not available on Linux and macOS.

3.4.6 Additional C++ Installation for 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.

Ensure the *Desktop Development with C++* option is checked in the Visual Studio installer for simulation support.



3.4.7 ¿Qué se instaló?

El instalador sin conexión instala los siguientes componentes:

- **Visual Studio Code** - El IDE compatible para 2019 y el desarrollo de código de robot posterior. El instalador fuera de línea configura una copia separada de VS Code para el desarrollo de WPILib, incluso si ya tiene VS Code en su máquina. Esto se hace porque algunas de las configuraciones que hacen que la configuración de WPILib funcione pueden romper los flujos de trabajo existentes si usa VS Code para otros proyectos.
- **C++ Compiler** - Las cadenas de herramientas para construir código de C++ para la roboRIO
- **Gradle** - La versión específica de Gradle usada para construir/desplegar código de robot C++ o Java
- **Java JDK/JRE** - Una versión específica de Java JDK/JRE que se utiliza para crear código de robot Java y para ejecutar cualquiera de las herramientas basadas en Java (Dashboards, etc.). Esto existe al lado de cualquier instalación de JDK existente y no sobrescribe 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 Desinstalando

WPILib está diseñado para poderse instalar en diferentes carpetas durante diferentes años, así que no es necesario desinstalar una versión previa antes de instalar WPILib de este año. No obstante, las siguientes instrucciones pueden usarse para desinstalar WPILib si es que se desea.

Windows

1. Delete the appropriate wpilib folder (`c:\Users\Public\wpilib\YYYY` where YYYY is the year to uninstall)
2. Elimine los iconos del escritorio en `C:\Users\Public\Public Desktop`

macOS

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

Linux

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

3.4.9 Solución de problemas

En caso de que el instalador falle, abra un problema en el repositorio del instalador. Un enlace está disponible *aquí <https://github.com/wpilibsuite/wpilibinstaller-avalonia>* __. El instalador debe dar un mensaje sobre la causa del error, por favor inclúyalo en la descripción de su problema.

3.5 Python Installation Guide

This guide is intended for Python teams. Java and C++ teams can skip to *Guía de Instalación WPILib*. LabVIEW teams can skip to *Instalación LabVIEW para FRC (únicamente LabVIEW)*.

3.5.1 Prerequisites

You must install a supported version of Python on a supported operating system. We currently support Python 3.8/3.9/3.10/3.11/3.12, but only 3.12 is available for the roboRIO.

Supported Operating Systems and Architectures:

- Windows 10 & 11, 64 bit only. 32 bit and Arm are not supported
- macOS 12 or higher
- Ubuntu 22.04, 64 bit. Other Linux distributions with glibc >= 2.35 may work, but are unsupported

On Windows and macOS, we recommend using the official Python installers distributed by python.org.

- Python for Windows
- Python for macOS

3.5.2 Install RobotPy

Once you have installed Python, you can use pip to install RobotPy on your development computer.

Windows

Nota: If you previously installed a pre-2024 or 2024 beta version of RobotPy, you should first uninstall RobotPy via `py -m pip uninstall robotpy` before upgrading.

Advertencia: On Windows, the Visual Studio 2019 redistributable package is required to be installed.

Run the following command from cmd or Powershell to install the core RobotPy packages:

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

To upgrade, you can run this:

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

If you don't have administrative rights on your computer, either use virtualenv/virtualenvwrapper-win, or or you can install to the user site-packages directory:

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

macOS

Nota: If you previously installed a pre-2024 or 2024 beta version of RobotPy, you should first uninstall RobotPy via `python3 -m pip uninstall robotpy` before upgrading.

On a macOS system that has pip installed, just run the following command from the Terminal application (may require admin rights):

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

To upgrade, you can run this:

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

If you don't have administrative rights on your computer, either use virtualenv/virtualenvwrapper, or you can install to the user site-packages directory:

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

Linux

Nota: If you previously installed a pre-2024 or 2024 beta version of RobotPy, you should first uninstall RobotPy via `python3 -m pip uninstall robotpy` before upgrading.

RobotPy distributes manylinux binary wheels on PyPI. However, installing these requires a distro that has glibc 2.35 or newer, and an installer that implements **PEP 600**, such as pip 20.3 or newer. You can check your version of pip with the following command:

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

If you need to upgrade your version of pip, it is highly recommended to use a virtual environment.

If you have a compatible version of pip, you can simply run:

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

To upgrade, you can run this:

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

If you manage to install the packages and get the following error or something similar, your system is most likely not compatible with 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 ARM Coprocessor

We publish prebuilt wheels on artifactory, which can be downloaded by giving the `--extra-index-url` option to pip:

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

source install

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

Advertencia: It may take a very long time to install!

Advertencia: Mixing our pre-built wheels with source installs may cause runtime errors. This is due to internal ABI incompatibility between compiler versions.

Our ARM wheels are built for Debian 11 with GCC 10.

If you need to build with a specific compiler version, you can specify them using the `CC` and `CXX` environment variables:

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

3.5.3 Download RobotPy for roboRIO

After installing the `robotpy` project on your computer, there are a variety of commands available that can be ran from the command line via the `robotpy` module.

Ver también:

Documentation for robotpy subcommands

If you already have a RobotPy robot project, you can use that to download the pieces needed to run on the roboRIO. If you don't have a project, running this command in an empty directory will initialize a new robot project:

Windows

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

macOS

```
python3 -m robotpy init
```

Linux

```
python3 -m robotpy init
```

This will create a `robot.py` and `pyproject.toml` file. The `pyproject.toml` file should be customized and details the requirements needed to run your robot code, among other things.

Ver también:

The default `pyproject.toml` created for you only contains the version of RobotPy installed on your computer. If you want to enable vendor packages or install other python packages from PyPI, see our *pyproject.toml documentation*

Next run the `robotpy sync` subcommand, which will:

- Download Python compiled for roboRIO
- Download roboRIO compatible python packages as specified by your `pyproject.toml`
- Install the packages specified by your `pyproject.toml` into your local environment

---

Nota: If you aren't using a virtualenv and don't have administrative privileges, the `robotpy sync` command accepts a `--user` argument to install to the user-specific site-packages directory.

Windows

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

macOS

```
python3 -m robotpy sync
```

Linux

```
python3 -m robotpy sync
```

When you deploy your code to the roboRIO, *the deploy subcommand* will automatically install Python (if needed) and your robot project requirements on the roboRIO as part of the deploy process.

3.6 Próximos pasos

¡Felicidades! Ha completado el paso 2 y ahora debería de tener un entorno de desarollo de software funcional. El paso 3 de este tutorial cubre la actualización del hardware para que lo pueda programar, mientras que el paso 4 muestra la programación de un robot en el VS Code Integrated Development Environment (IDE). Para más información, puede leer *VS Code section* para familiarizarse con el IDE.

Los artículos específicos que se recomienda leer son:

- *Visual Studio Code Basics*
- *WPILib Commands in Visual Studio Code*
- *Creating a Robot Program*
- *Building and Deploying Robot Code*
- *Installing 3rd Party Libraries*

Además, es posible que deba realizar una configuración adicional que sea aplicable al robot de su equipo. Utilice la función de búsqueda para encontrar la documentación necesaria.

Nota: Es importante que los equipos que utilizan controladores de motor CAN de terceros consulten el artículo *Instalado librerías de Terceros* ya que se requieren pasos adicionales para codificar estos dispositivos.

---

4 Paso 3: Preparación de su Robot

4.1 Imaging your roboRIO 2

Nota: 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.

Importante: 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.

Advertencia: 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

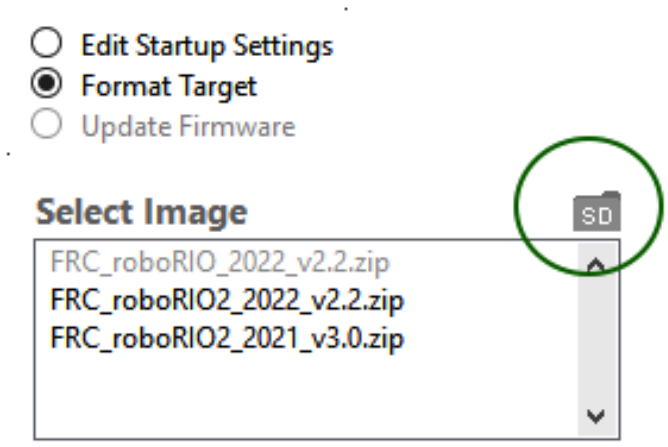
roboRIO are secure and that the connector is secure firmly to the roboRIO (4 total screws to check).

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.

Nota: 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*

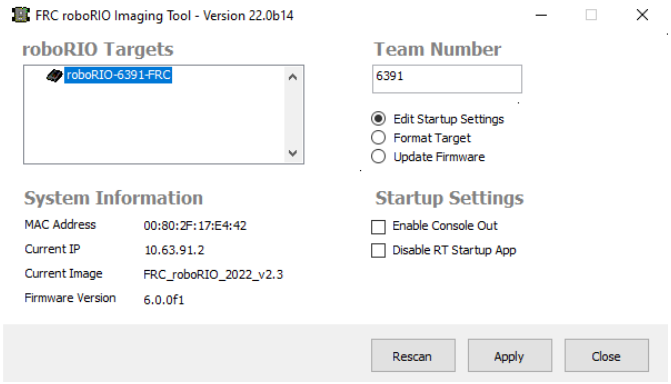
Writing the image with Raspberry Pi Imager

- 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*

Advertencia: 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

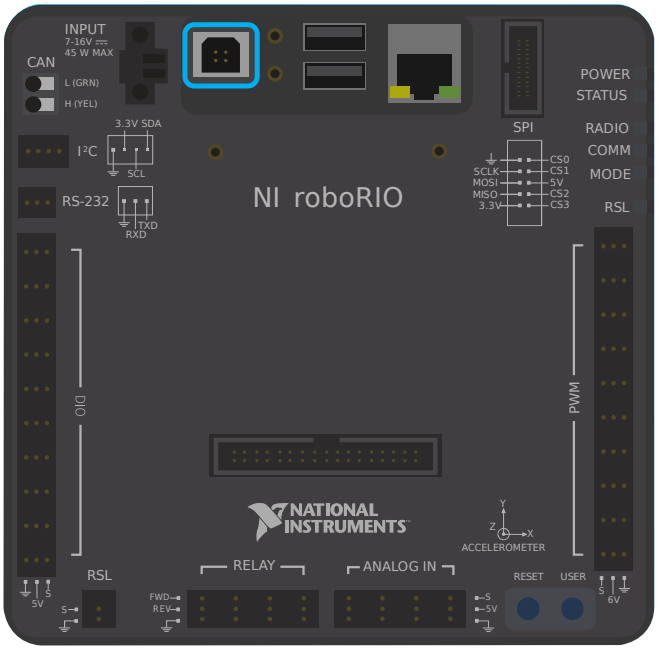
Advertencia: Antes de crear la imagen de su roboRIO, debe haber completado la instalación de *FRC Game Tools*. También debe tener la alimentación de energía de la roboRIO debidamente conectada al panel de distribución de energía o PDP. Asegúrese de que los cables de alimentación de la roboRIO estén seguros y de que el conector esté firmemente asegurado a la roboRIO (4 tornillos en total para verificar).

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

4.2.1 Configurar la roboRIO

La herramienta de imágenes roboRIO se utilizará para crear imágenes de su roboRIO con el software más reciente.

Conexión USB



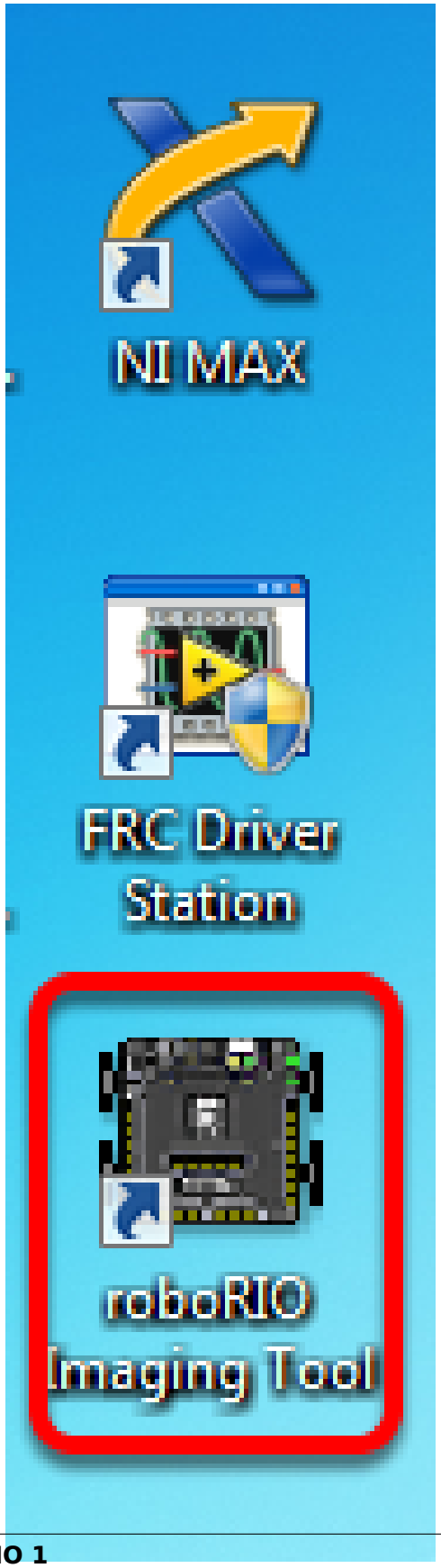
Conecte un cable USB desde el puerto del dispositivo USB roboRIO a la PC. Esto requiere un cable USB tipo A macho (extremo de PC estándar) a tipo B macho (cuadrado con 2 esquinas cortadas), que se encuentra más comúnmente como cable USB de impresora.

Nota: La imagen de la roboRIO solo debe realizarse a través de la conexión USB. No se recomienda intentar obtener imágenes utilizando la conexión Ethernet.

Instalación del controlador

El controlador del dispositivo debería instalarse automáticamente. Si ve la ventana emergente «Nuevo dispositivo» en la parte inferior derecha de la pantalla, espere a que se complete la instalación del controlador antes de continuar.

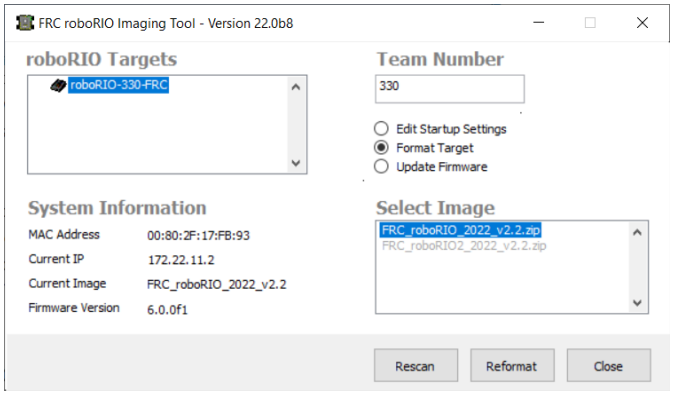
4.2.2 Iniciar la herramienta de imágenes



La herramienta de generación de imágenes roboRIO y la última imagen se instalan con NI FRC|reg| Game Tools. Inicie la herramienta de imágenes haciendo doble clic en el acceso directo del escritorio. Si tiene dificultades para crear imágenes de su roboRIO, es posible que deba intentar hacer clic con el botón derecho en el icono y seleccionar Ejecutar como administrador.

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

4.2.3 Herramienta de imágenes roboRIO

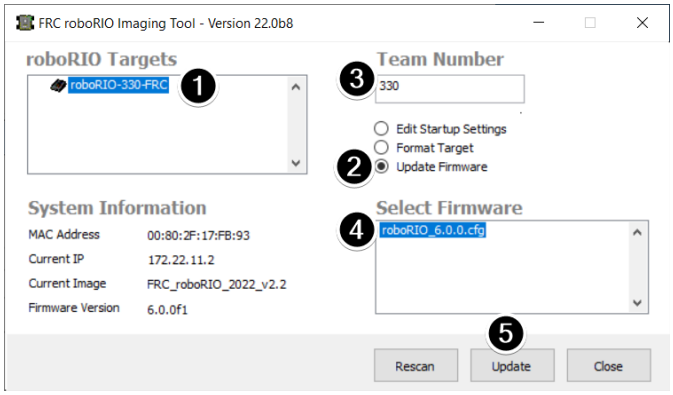


Despúes de inciarla, la roboRIO Imaging Tool buscará roboRIOs e indicará los que se encuentran disponibles en el cuadro superior izquierdo. El cuadro inferior izquierdo mostrará información y configuraciones para el roboRIO actualmente seleccionado. El panel derecho contiene controles para modificar la configuración de roboRIO:

- **Edit Startup Settings** - Esta opción se usa cuando quiere ajustar las configuraciones iniciales del roboRIO (los ajustes en el panel derecho), sin configurar en sí el roboRIO.
- **Format Target** - Esta opción es usada cuando se quiere subir una imagen en la roboRIO (o volver a cargar una imagen existente). Esta es la opción más común.
- **Update Firmware** - Esta opción es usada para actualizar el firmware del roboRIO. Para esta temporada, la herramienta de imagen requerirá que el firmware del roboRIO sean versión 5.0 o superior.

Actualizando el Firmware

Advertencia: 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.



El firmware de roboRIO debe ser al menos v5.0 para funcionar con la imagen de 2019 o posterior. Si su roboRIO es al menos la versión 5.0 (como se muestra en la parte inferior izquierda de la herramienta de imágenes), no necesita actualizar.

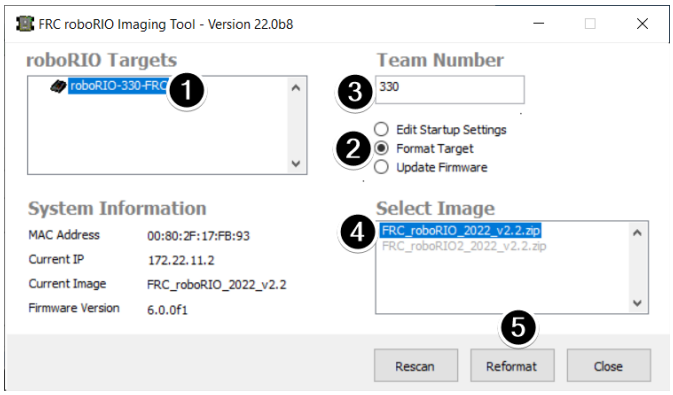
Nota: roboRIO firmware has had different version numbering schemes over the years. It isn't necessary to update the firmware if it has version 5, 6, 8, 22.5, 23.5 or variations of those version numbers (e.g. 8.8.0f0 is a variation of 8). The firmware is only utilized in *safe mode*, it is not used in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

Para actualizar el firmware de la roboRIO:

1. Asegúrese que su roboRIO esté seleccionada en el panel superior izquierdo.
2. Select *Update Firmware* in the top right pane
3. Enter a team number in the *Team Number* box
4. Seleccione el archivo de firmware más reciente en la parte inferior derecha
5. Click the *Update* button

4.2.4 Configurando la imagen de su roboRIO

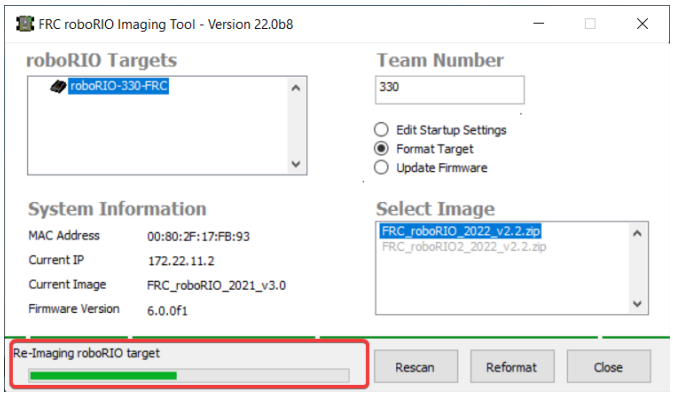
Advertencia: The roboRIO image is different then the firmware, and must be updated yearly.



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

1. Asegúrese que la roboRIO está seleccionado en el panel superior izquierdo
2. Select *Format Target* in the right pane
3. Enter a team number in the *Team Number* box
4. Seleccione la imagen con la versión más reciente en el cuadro.
5. Click *Reformat* to begin the imaging process.

4.2.5 Proceso de Configuración



El proceso de configuración tomará aproximadamente 3-10 minutos. Una barra de progreso en la parte inferior izquierda de la ventana indicará el progreso.

4.2.6 Configuración Completada

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 Solución de problemas

Si no puede configurar su roboRIO, los pasos para solución de problemas incluyen:

- Intente ejecutar la roboRIO Imaging Tool como Administrador haciendo clic derecho en el ícono del Escritorio para iniciarlo.
- Intente accesando la página web del roboRIO con un navegador web en `http://172.22.11.2/` y/o verifique que el adaptador de red NI aparezca en su lista de Adaptadores de Red en el Panel de Control. Si no, intente reinstalar las NI FRC Game Tools o pruebe en una PC diferente.
- *Deshabilite todos los demás adaptadores de red*
- Asegúrese que su firewall esté apagado.
- 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*
- Intente arrancar la roboRIO en Modo Seguro manteniendo presionado el botón de reinicio durante al menos 5 segundos.
- Try a different USB Cable
- Intente con una computadora diferente
- 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 Programando su Radio

Esta guía le enseñará cómo usar el software FRC® Radio Configuration Utility para configurar el puente inalámbrico de su robot para usarlo fuera de los eventos de FRC.

4.3.1 Requisitos previos

La FRC Radio Configuration Utility requiere de privilegios de Administrador para configurar ajustes de red en su máquina. El programa debería solicitar los privilegios automáticamente (podría requerir una contraseña si se ejecuta en una cuenta que no sea de Administrador), pero si esta teniendo problemas, intente ejecutarlo desde una cuenta de Administrador.

Descargue el último instalador de FRC Radio Configuration Utility de los siguientes links:

FRC Radio Configuration 24.0.1

FRC Radio Configuration 24.0.1 Israel Version

Nota: La versión _IL es para los equipos de Israel y contiene una versión del firmware OM5PAC con canales restringidos para uso en Israel.

Antes de empezar a usar el software:

1. *Deshabilite cualquier otro adaptador de red*
2. Plug directly from your computer into the wireless bridge ethernet port closest to the power jack. Make sure no other devices are connected to your computer via ethernet. If powering the radio via PoE, plug an Ethernet cable from the PC into the socket side of the PoE adapter (where the roboRIO would plug in). If you experience issues configuring through the PoE adapter, you may try connecting the PC to the alternate port on the radio.

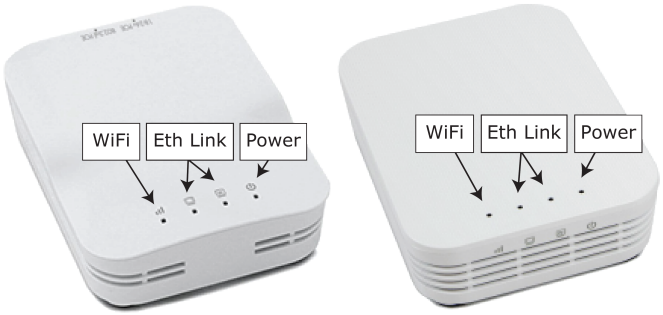
Advertencia: El OM5P-AN y AC usan el adaptador de corriente que el D-Link DAP 1522, sin embargo, son radios de 12V. Conecte la radio a los terminales de 12V 2A en el VRM (pin central positivo).

4.3.2 Notas de Aplicación

Por default, la Radio Configuration Utility programará para imponer el límite de ancho de banda de 4Mbps en el tráfico que sale de la radio a través de la interfaz inalámbrica. En la configuración de inicio (Modo AP) este es un límite total, no por cliente. Esto significa que no se recomienda transmitir video a múltiples clientes.

La Utility ha sido probada en Windows 7, 8 y 10. Puede que trabaje en otros sistemas operativos, pero no ha sido probada.

Configuración Programada



La FRC Radio Configuration Utility programa una serie de ajustes de configuración en la radio cuando se inicia. Esta configuración se aplica a la radio en todos los modos (incluso en eventos). Éstos incluyen:

- Establecer una IP estática de `10.TE.AM.1`
- Establecer una IP alternativa en el lado cableado de `192.168.1.1` para programación futura
- Puentee los puertos cableados para que puedan ser usados indistintamente
- Las configuraciones LED en el gráfico anterior.

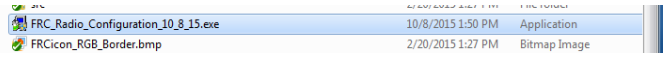
- Límite de ancho de banda de 4 Mb/s en el lado de salida de la interfaz inalámbrica (puede deshabilitarse para uso doméstico)
- Reglas QoS para priorización interna de paquetes (afecta el búfer interno y qué paquetes descartar si se alcanza el límite de ancho de banda). Estas reglas son:
  - Robot Control y Estatus (UDP `1110`, `1115`, `1150`)
  - Robot TCP & *NetworkTables* (TCP `1735`, `1740`)
  - Bulk (todo el resto del tráfico). (deshabilitado si el límite de ancho de banda está deshabilitado)
- *DHCP* server enabled. Serves out:
  - `10.TE.AM.11` - `10.TE.AM.111` en el lado cableado
  - `10.TE.AM.138` - `10.TE.AM.237` en el lado inalámbrico
  - Máscara de subred `255.255.255.0`
  - Dirección de transmisión `10.TE.AM.255`
- DNS server enabled. DNS server IP and domain suffix (`.lan`) are served as part of the DHCP.

Solo en casa:

- El SSID puede tener un «Nombre del robot» anexado al número de equipo para distinguir varias redes.
- La opción Firewall puede estar habilitada para imitar las reglas del firewall de campo (los puertos abiertos se pueden encontrar en el Manual del Juego)

Advertencia: Esta configuración no se puede modificar manualmente.

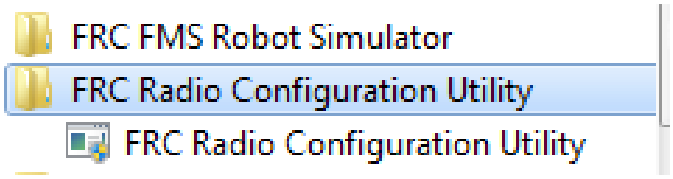
4.3.3 Instalar el software



Haga doble clic en `FRC_Radio_Configuration_VERSION.exe` para iniciar el instalador. Siga las indicaciones para completar la instalación.

Algunas de las indicaciones incluirán instalar Npcap si aún no está presente. El instalador de Npcap contiene un número de casillas de verificación para configurar la instalación. Debe dejar las opciones como default.

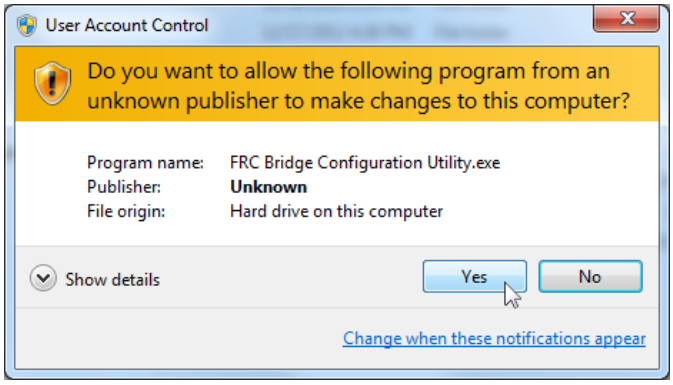
4.3.4 Iniciar el software



Use el menú de inicio o el atajo del escritorio para iniciar el programa.

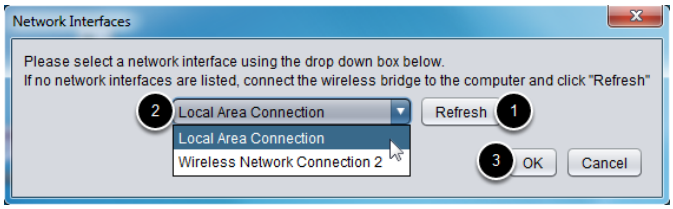
Nota: Si usted necesita localizar el programa, está instalado en `C:/Program Files (x86)/FRC Radio Configuration Utility`. Para computadoras de 32-bit la ubicación es `C:/Program Files/FRC Radio Configuration Utility/`

4.3.5 Permita que el programa realice cambios, si lo solicita



A prompt may appear about allowing the configuration utility to make changes to the computer. Click *Yes* if the prompt appears.

4.3.6 Seleccione la interfaz de red



Use la ventana desplegable para seleccionar cual interfaz ethernet usará la utilidad de configuración para comunicarse con el puente inalámbrico. En máquinas Windows, las interfaces ethernet son llamadas «Local Area Connection». La configuration utility no puede programar el puente sobre una conexión inalámbrica.

1. Si no hay interfaces ethernet enlistadas, haga clic en *Refresh* para reescanear por interfaces disponibles
2. Seleccione la interface que desee del menú desplegable.

3. Click *OK*.

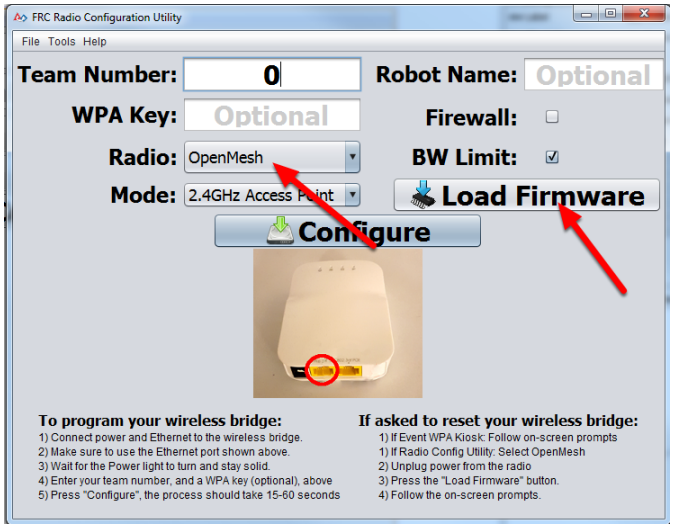
4.3.7 Abra Mesh Firmware Note

For the FRC Radio Configuration Utility to program the OM5P-AN and OM5P-AC radio, the radio must be running an FRC specific build of the OpenWRT firmware.

Si no necesita actualizar o recargar el firmware, salte el siguiente paso.

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

4.3.8 Cargando el Firmware de FRC para radio OpenMesh



Si necesita cargar el firmware de FRC (o reiniciar el radio), puede hacerlo usando la FRC Radio Configuration Utility.

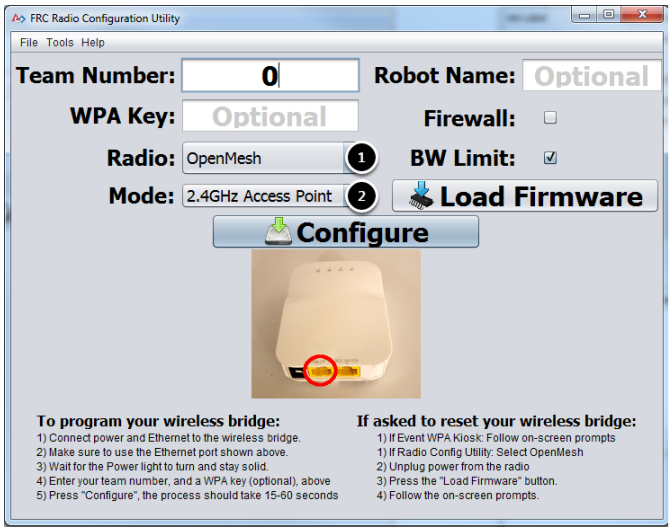
1. Siga las instrucciones anteriores para instalar el software, inicie el programa y seleccione la interfaz Ethernet.
2. Asegúrese que la radio OpenMesh esté seleccionada en el desplegable de Radio.
3. Asegúrese que la radio esté conectada a la PC vía Ethernet.
4. Desenchufe la energía del radio. (Si esta usando un cable PoE, también desenchufará el Ethernet de la PC, esto está bien)
5. Presione el botón Load Firmware
6. Cuando se le indique, enchufe energía al radio. El software debería detectar el radio, cargue el firmware e indique cuando termine.

Advertencia: Si ve un error sobre el nombre NPF, pruebe deshabilitando todos los adaptadores excepto el que esté siendo usado para programar el radio. Si solo un adaptador es

encontrado, la herramienta debería intentar usar ese. Vea los pasos en «**`Troubleshooting: Disabling Network Adapters`_**» para más información.

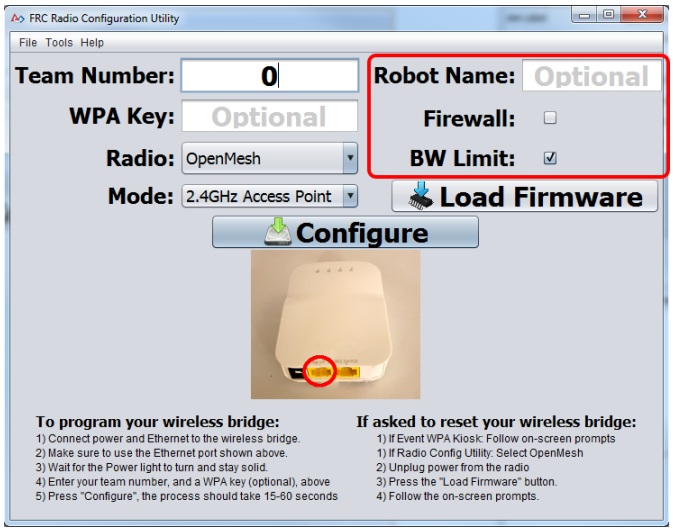
Teams may also see this error with Operating Systems configured for languages other than US English. If you experience issues loading firmware or programming on a foreign language OS, try using an English OS, such as on the KOP provided PC or setting the Locale setting to «en_us» as described on this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

4.3.9 Seleccione un modelo de puente y modo operativo



1. Seleccione cual radio va a configurar usando la lista desplegable.
2. Seleccione cual modo operativo desea configurar. Para la mayoría de los casos, la selección default de 2.4GHz Access Point será suficiente. Si sus computadoras lo soportan, el modo 5GHz AP es recomendado, debido a que el 5GHz está menos congestionado en muchos entornos.

4.3.10 Seleccione Opciones



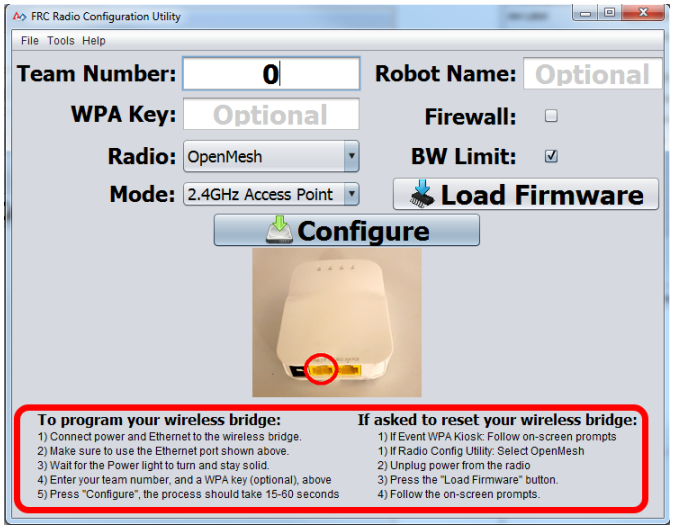
Los valores predeterminados de las opciones han sido seleccionados para corresponder el caso de uso de la mayoría de los equipos, de cualquier forma, podría desear personalizar estas opciones para su específico escenario:

1. **Robot Name:** Esta es una cadena que se agrega al SSID utilizado por la radio. Esto le permite tener múltiples redes con el mismo número de equipo y aún así poder distinguirlas.
2. **Firewall:** Si esta casilla está marcada, el firewall del radio estará configurado para intentar imitar el comportamiento de bloqueo del puerto del firewall presente en el campo de FRC. Para una lista de puertos abiertos, vea el FRC Game Manual.
3. **BW Limit:** If this box is checked, the radio enforces a 4 Mbps bandwidth limit like it does when programmed at events. Note that this is a total limit, not per client, so streaming video to multiple clients simultaneously may cause undesired behavior.

Nota: El Firewall y el Límite de Banda Ancha solo aplican para radios OpenMesh. Éstas opciones no tienen efecto en radios D-Link.

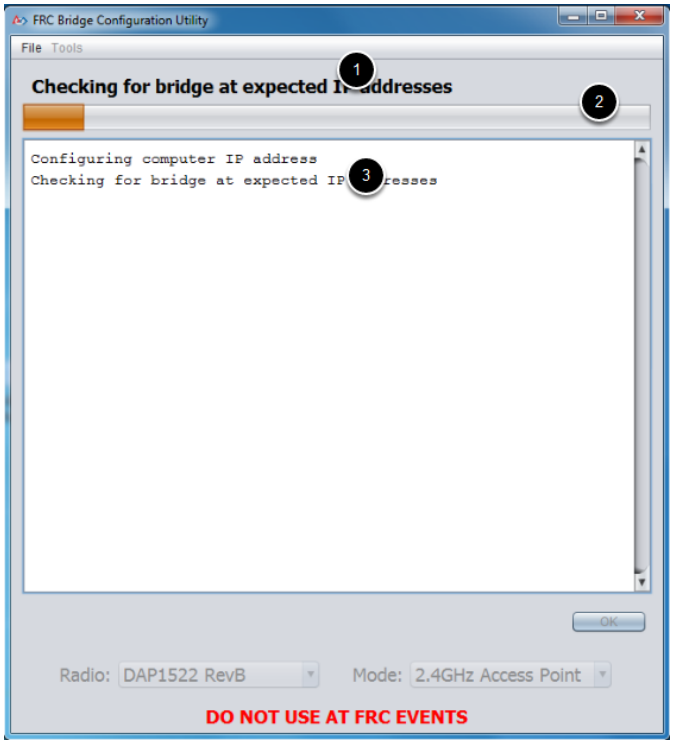
Advertencia: La opción “Firewall” configura la radio para emular el firewall de campo. Esto significa que no podrá implementar código de forma inalámbrica con esta opción habilitada.

4.3.11 Prepare e inicie el proceso de configuración



Siga las instrucciones en la pantalla para preparar su puente inalámbrico, ingresando los ajustes con los que el puente será configurado, e iniciando el proceso de configuración. Estas instrucciones en la pantalla se actualizan para corresponder al modelo del puente y el modo operativo elegido.

4.3.12 Progreso de Configuración

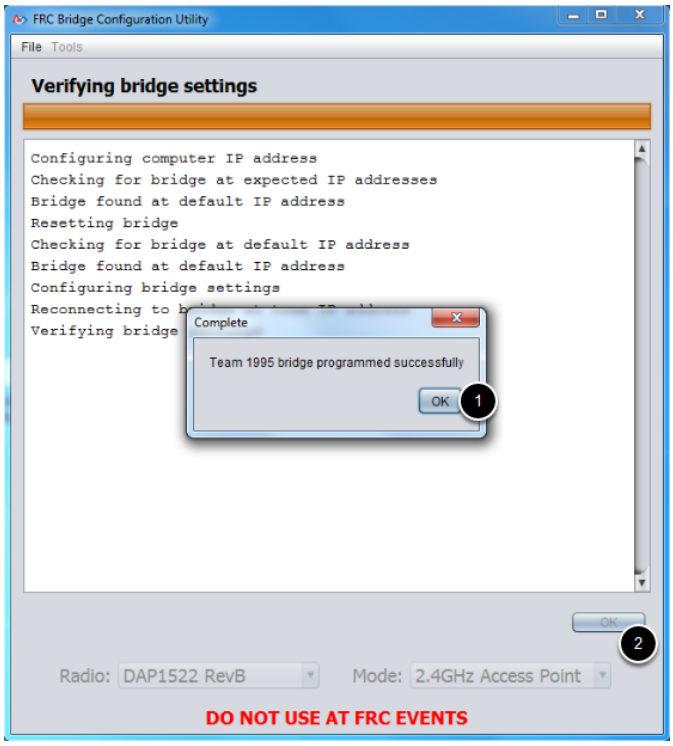


Durante el proceso de configuración, la ventana indicará:

1. El paso que esta siendo ejecutado en ese momento

2. El progreso general del proceso de configuración
3. Todos los pasos ejecutados hasta el momento

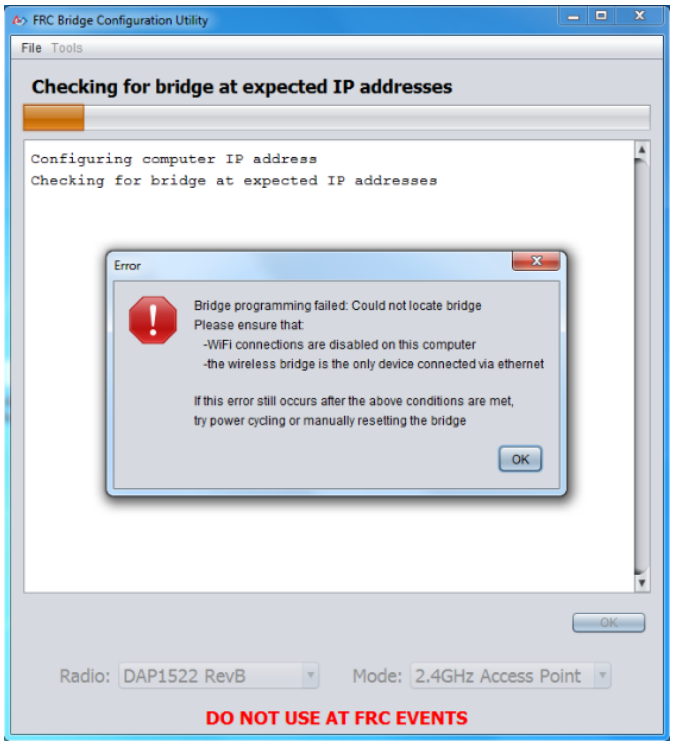
4.3.13 Configuración completada



Una vez que se complete la configuración:

1. Presione “OK” en la ventana de diálogo
2. Presione “OK” en la ventana principal para regresar a la pantalla de ajustes

4.3.14 Errores de configuración



Si ocurre un error durante el proceso de configuración, siga las instrucciones en el mensaje de error para corregir el problema.

4.3.15 Solución de problemas

- *Disable all other network adapters*.
- Asegúrese de esperar el tiempo suficiente para que la luz de encendido permanezca fija durante 10 segundos.
- Asegúrese de tener la interfaz de red correcta, y solo una interfaz aparece en el menú desplegable.
- Make sure your firewall is turned off.
- Conecte directamente desde su computadora el puente inalámbrico y asegúrese de que no hayan otros dispositivos conectados a su computadora vía ethernet
- Asegúrese de que el ethernet esté enchufado en el puerto más cercano al conector de alimentación en el puente inalámbrico.
- 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.

- Some users have reported success after installing npcap 1.60. If this doesn't resolve the issue, it's recommended to uninstall npcap and the radio tool and then reinstall the radio tool in order to get back to a known configuration.
- If all else fails, try a different computer.

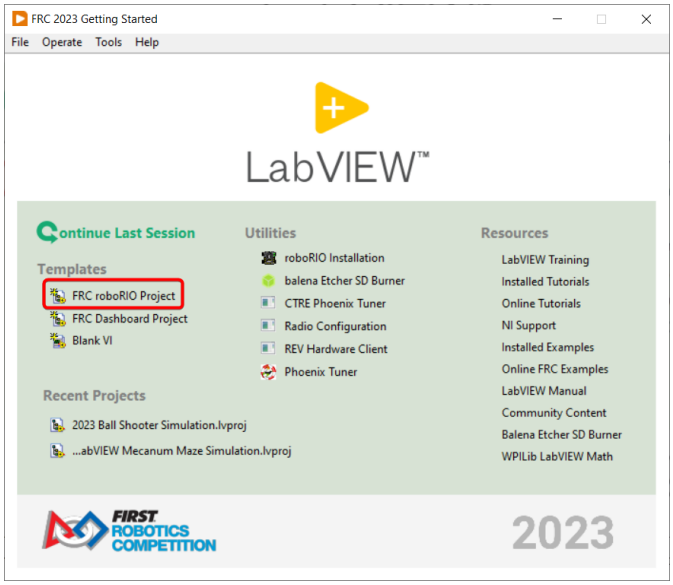
5

Paso 4: Programando su Robot

5.1 Creating your Test Drivetrain Program (LabVIEW)

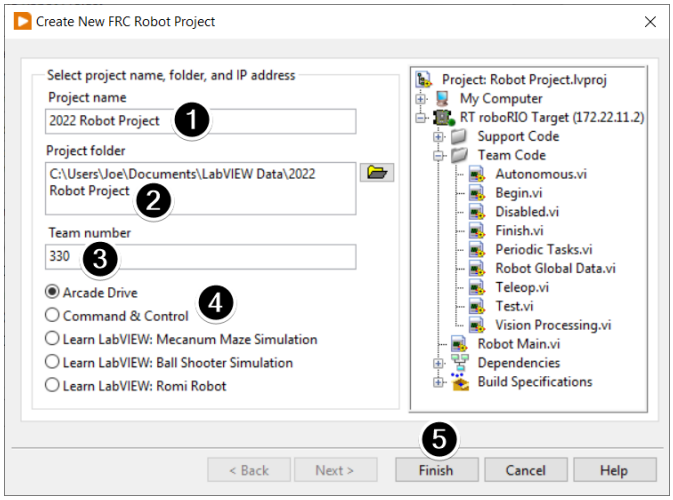
Nota: This document covers how to create, build and load a basic FRC® LabVIEW program for a drivetrain onto a roboRIO. Before beginning, make sure that you have installed LabVIEW for FRC and the FRC Game Tools and that you have configured and imaged your roboRIO as described in the *Zero-to-Robot tutorial*.

5.1.1 Creating a Project



Launch LabVIEW and click the FRC roboRIO Robot Project link to display the Create New FRC Robot Project dialog box.

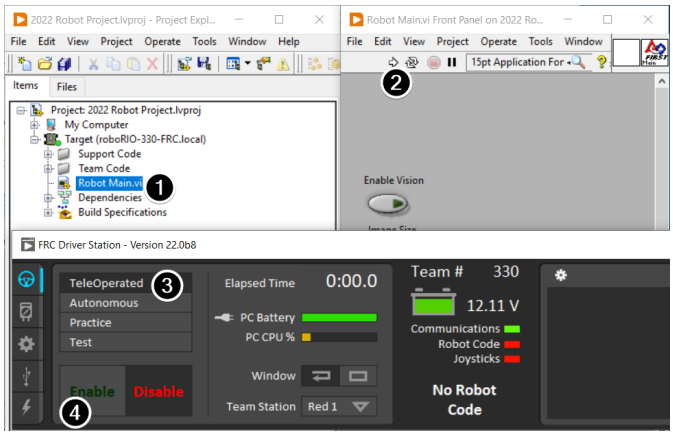
5.1.2 Configuring Project



Fill in the Create New FRC Project Dialog:

1. Pick a name for your project
2. Select a folder to place the project in.
3. Enter your team number
4. Select a project type. If unsure, select *Arcade Drive*.
5. Click *Finish*

5.1.3 Running the Program

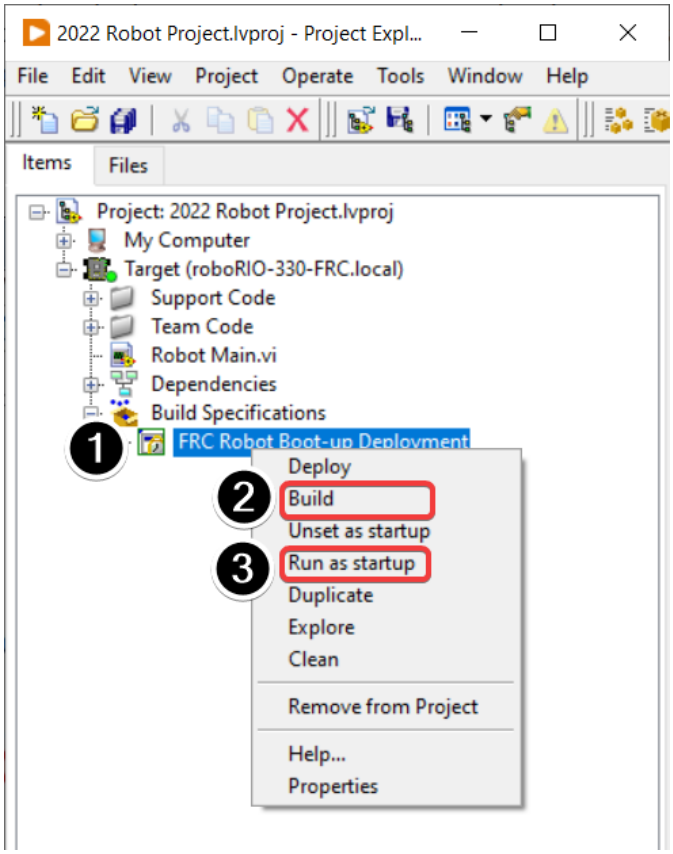


Nota: Note that a program deployed in this manner will not remain on the roboRIO after a power cycle. To deploy a program to run every time the roboRIO starts follow the next step, Deploying the program.

1. In the Project Explorer window, double-click the Robot Main.vi item to open the Robot Main VI.

2. Click the Run button (White Arrow on the top ribbon) of the Robot Main VI to deploy the VI to the roboRIO. LabVIEW deploys the VI, all items required by the VI, and the target settings to memory on the roboRIO. If prompted to save any VIs, click Save on all prompts.
3. Using the Driver Station software, put the robot in Teleop Mode. For more information on configuring and using the Driver Station software, see the FRC Driver Station Software article.
4. Click Enable.
5. Move the joysticks and observe how the robot responds.
6. Click the Abort button of the Robot Main VI. Notice that the VI stops. When you deploy a program with the Run button, the program runs on the roboRIO, but you can manipulate the front panel objects of the program from the host computer.

5.1.4 Deploying the Program



To run in the competition, you will need to deploy a program to your roboRIO. This allows the program to survive across reboots of the controller, but doesn't allow the same debugging features (front panel, probes, highlight execution) as running from the front panel. To deploy your program:

1. In the Project Explorer, click the + next to Build Specifications to expand it.
2. Right-click on FRC Robot Boot-up Deployment and select Build. Wait for the build to complete.

3. Right-click again on FRC Robot Boot-Up Deployment and select Run as Startup. If you receive a conflict dialog, click OK. This dialog simply indicates that there is currently a program on the roboRIO which will be terminated/replaced.
4. Either check the box to close the deployment window on successful completion or click the close button when the deployment completes.
5. The roboRIO will automatically start running the deployed code within a few seconds of the dialog closing.

5.2 Creating your Test Drivetrain Program (Java/C++/Python)

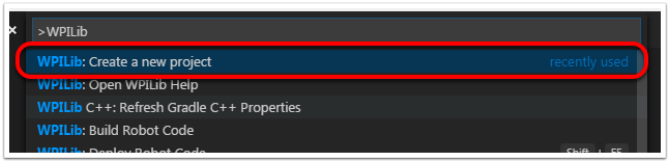
Once everything is installed, we're ready to create a robot program. WPILib comes with several templates for robot programs. Use of these templates is highly recommended for new users; however, advanced users are free to write their own robot code from scratch. This article walks through creating a project from one of the provided examples which has some code already written to drive a basic robot.

- *Creating a New WPILib Project (Java/C++)*
- *Creating a New WPILib Project (Python)*

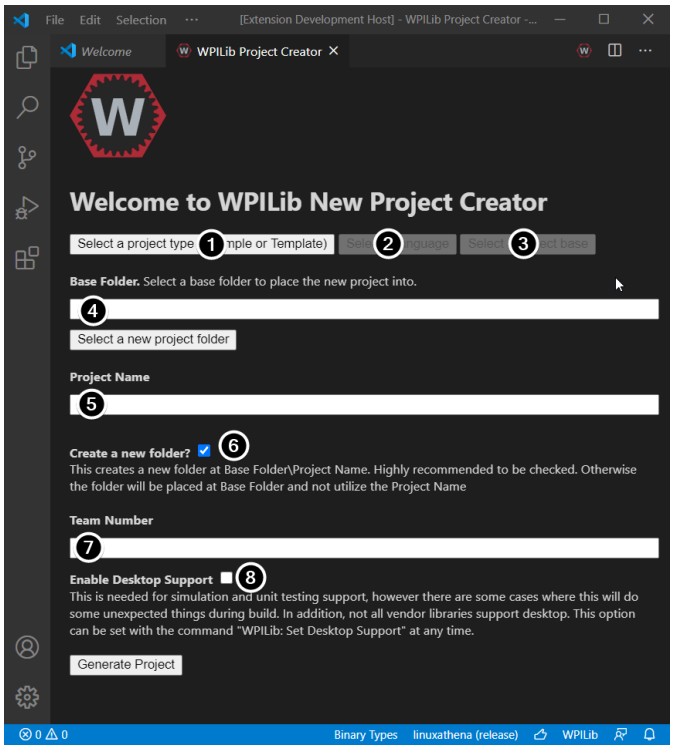
Importante: This guide includes code examples that involve vendor hardware for the convenience of the user. In this document, *PWM* refers to the motor controller included in the KOP. The CTRE tab references the Talon FX motor controller (Falcon 500 motor), but usage is similar for TalonSRX and VictorSPX. The REV tab references the CAN SPARK MAX controlling a brushless motor, but it's similar for brushed motor. There is an assumption that the user has already installed the required *vendordeps* and configured the device(s) (update firmware, assign CAN IDs, etc) according to the manufacturer documentation (CTRE REV).

5.2.1 Creating a New WPILib Project (Java/C++)

Bring up the Visual Studio Code command palette with `Ctrl+Shift+P`. Then, type «WPILib» into the prompt. Since all WPILib commands start with «WPILib», this will bring up the list of WPILib-specific VS Code commands. Now, select the «Create a new project» command:



This will bring up the «New Project Creator Window:»



The elements of the New Project Creator Window are explained below:

1. **Project Type**: The kind of project we wish to create. For this example, select **Example**
2. **Language**: This is the language (C++ or Java) that will be used for this project.
3. **Project Base**: This box is used to select the base class or example to generate the project from. For this example, select **Getting Started**
4. **Base Folder**: This determines the folder in which the robot project will be located.
5. **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.
6. **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. project folder will be given if the Create New Folder box is checked.
7. **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.
8. **Enable Desktop Support**: Enables unit test and simulation. While WPILib supports this, third party software libraries may not. If libraries do not support desktop, then your code may not compile or may crash. It should be left unchecked unless unit testing or simulation is needed and all libraries support it. For this example, do not check this box.

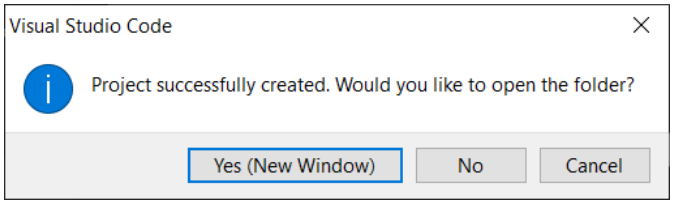
Once all the above have been configured, click «Generate Project» and the robot project will be created.

Nota: Any errors in project generation will appear in the bottom right-hand corner of the

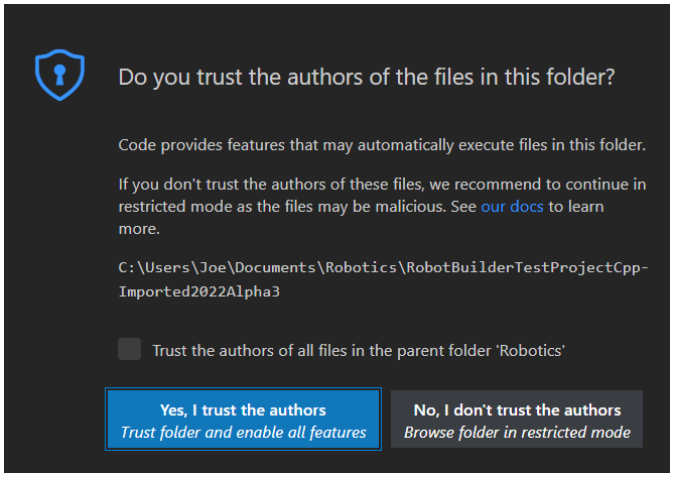
screen.

Advertencia: 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.

5.2.2 Opening The New Project

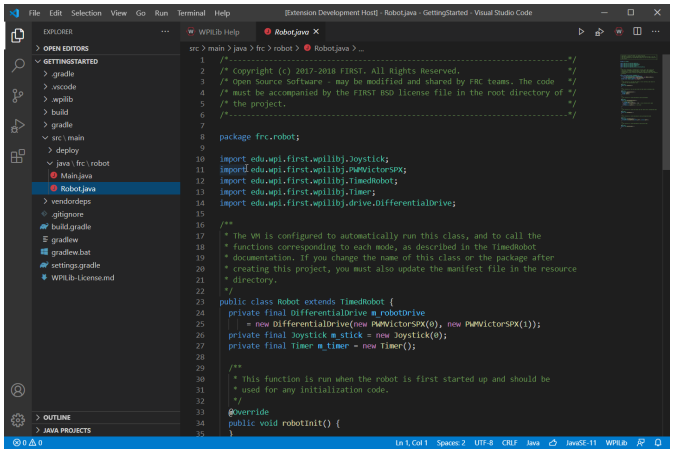


After successfully creating your project, VS Code will give the option of opening the project as shown above. We can choose to do that now or later by typing `Ctrl+K` then `Ctrl+O` (or just `Command+O` on macOS) and select the folder where we saved our project.



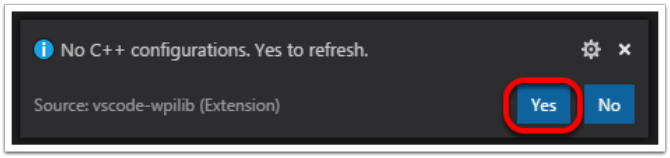
Click *Yes I trust the authors*.

Once opened we will see the project hierarchy on the left. Double clicking on the file will open that file in the editor.



5.2.3 C++ Configurations (C++ Only)

For C++ projects, there is one more step to set up IntelliSense. Whenever we open a project, we should get a pop-up in the bottom right corner asking to refresh C++ configurations. Click «Yes» to set up IntelliSense.



5.2.4 Creating a New WPILib Project (Python)

Running the `robotpy init` command will initialize a new robot project:

Windows

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

macOS

```
python3 -m robotpy init
```

Linux

```
python3 -m robotpy init
```

This will create a `robot.py` and `pyproject.toml` file, but will not overwrite an existing file.

- The `pyproject.toml` file contains the requirements for your project, which are downloaded and installed via the `robotpy sync` command.
- The `robot.py` file is where you will put the your Robot class.

Ver también:

Download RobotPy for roboRIO

5.2.5 Basic Drivetrain example

First, here is what a simple code can look like for a Drivetrain with PWM controlled motors (such as SparkMax).

Nota: the Python example below is from 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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  private final Timer m_timer = new Timer();

  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); },
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      [&](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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
            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)
```

Now let's look at various parts of the 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
```

Our code needs to reference the components of WPILib that are used. In C++ this is accomplished using `#include` statements; in Java it is done with `import` statements. The program references classes for `Joystick` (for driving), `PWMSparkMax` / `WPI_TalonFX` / `CANSparkMax (for controlling motors), ``TimedRobot` (the base class used for the example), `Timer` (used for autonomous), and `DifferentialDrive` (for connecting the joystick control to the motors).

5.2.7 Defining the variables for our sample 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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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)
```

The sample robot in our examples will have a joystick on USB port 0 for arcade drive and two motors on PWM ports 0 and 1 (Vendor examples use CAN with IDs 1 and 2). Here we create objects of type DifferentialDrive (m_robotDrive), Joystick (m_stick) and Timer (m_timer). This section of the code does three things:

1. Defines the variables as members of our Robot class.

2. Initializes the variables.

Nota: The variable initializations for C++ are in the `private` section at the bottom of the program. This means they are private to the class (`Robot`). The C++ code also sets the Motor Safety expiration to 0.1 seconds (the drive will shut off if we don't give it a command every .1 seconds) and starts the `Timer` used for autonomous.

5.2.8 Robot Initialization

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):
```

The `RobotInit` method is run when the robot program is starting up, but after the constructor. The `RobotInit` for our sample program inverts the right side of the drivetrain. Depending on your drive setup, you might need to invert the left side instead.

Nota: In C++, the drive inversion is handled in the `Robot()` constructor above.

5.2.9 Simple Autonomous Example

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   }
50
51   /** This function is called periodically during autonomous. */
```

(continúe en la próxima página)

(proviene de la página anterior)

```java
  @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
```

The `AutonomousInit` method is run once each time the robot transitions to autonomous from another mode. In this program, we restart the `Timer` in this method.

`AutonomousPeriodic` is run once every period while the robot is in autonomous mode. In the `TimedRobot` class the period is a fixed time, which defaults to 20ms. In this example, the periodic code checks if the timer is less than 2 seconds and if so, drives forward at half speed using the `ArcadeDrive` method of the `DifferentialDrive` class. If more than 2 seconds has elapsed, the code stops the robot drive.

5.2.10 Joystick Control for Teleoperation

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()
        )
```

Like in Autonomous, the Teleop mode has a `TeleopInit` and `TeleopPeriodic` function. In this example we don't have anything to do in `TeleopInit`, it is provided for illustration purposes only. In `TeleopPeriodic`, the code uses the `ArcadeDrive` method to map the Y-axis of the `Joystick` to forward/back motion of the drive motors and the X-axis to turning motion.

5.2.11 Test Mode

JAVA

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

```

(continúe en la próxima página)

(proviene de la página anterior)

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

C++

```
45  void TestInit() override {}
46
47  void TestPeriodic() override {}
```

PYTHON

```
54      def testInit(self):
55          """This function is called once each time the robot enters test mode."""
56
57      def testPeriodic(self):
58          """This function is called periodically during test mode."""
```

Test Mode is used for testing robot functionality. Similar to `TeleopInit`, the `TestInit` and `TestPeriodic` methods are provided here for illustrative purposes only.

5.2.12 Deploying the Project to a Robot

- *Deploy Java/C++ code*
- *Deploy Python code*

5.3 Running your Test Program

5.3.1 Overview

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

C++/Java/Python

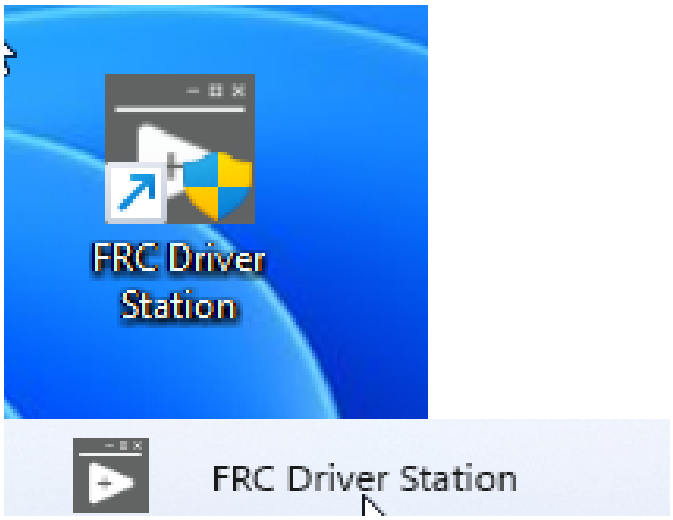
LabVIEW

5.3.2 Tethered Operation

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.

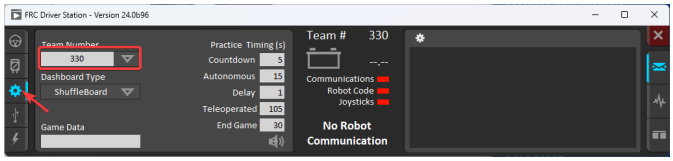
The roboRIO should be powered on and connected to the PC over Ethernet or USB.

5.3.3 Starting the FRC Driver Station



The FRC® Driver Station can be launched by double-clicking the icon on the Desktop or by selecting Start->All Programs->FRC Driver Station.

5.3.4 Setting Up the Driver Station



The DS must be set to your team number in order to connect to your robot. In order to do this click the Setup tab then enter your team number in the team number box. Press return or click outside the box for the setting to take effect.

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

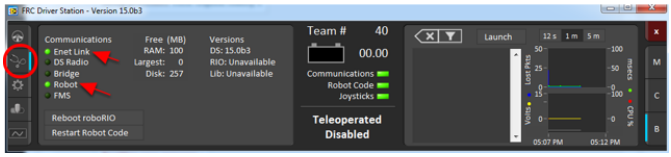


Figura 1: Tethered

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.

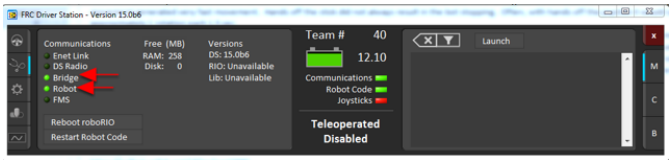
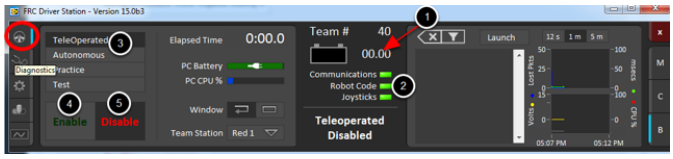


Figura 2: Wireless

5.3.6 Operate the Robot



Click the Operation Tab

1. Confirm that battery voltage is displayed
2. Communications, Robot Code, and Joysticks indicators are green.
3. Put the robot in Teleop Mode
4. Click Enable. Move the joysticks and observe how the robot responds.
5. Click Disable

5.3.7 Wireless Operation

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.

Configuring the Access Point

See the article *Programming your radio* for details on configuring the robot radio for use as an access point.

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 Descripción general de los componentes de Hardware

El objetivo de este documento es proporcionar una breve descripción de los componentes del hardware que componen el Sistema de Control FRC®. Cada componente contendrá una breve descripción de la función del componente y un enlace a más documentación.

Nota: For wiring instructions/diagrams, please see the *Wiring the FRC Control System* document.

6.1 Descripción general del Sistema de Control

REV

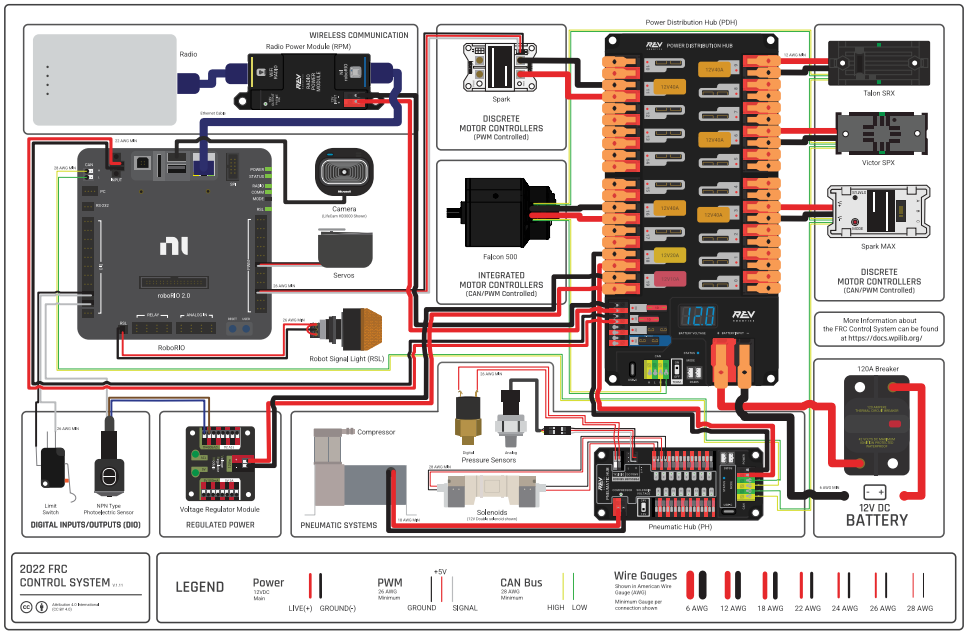


Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

CTRE

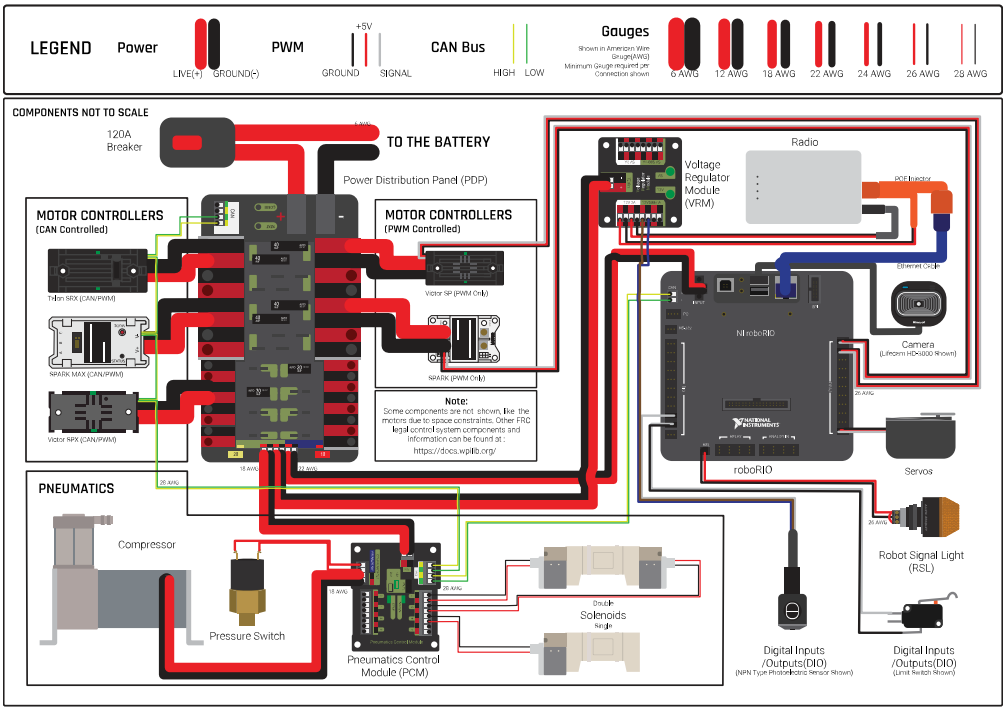
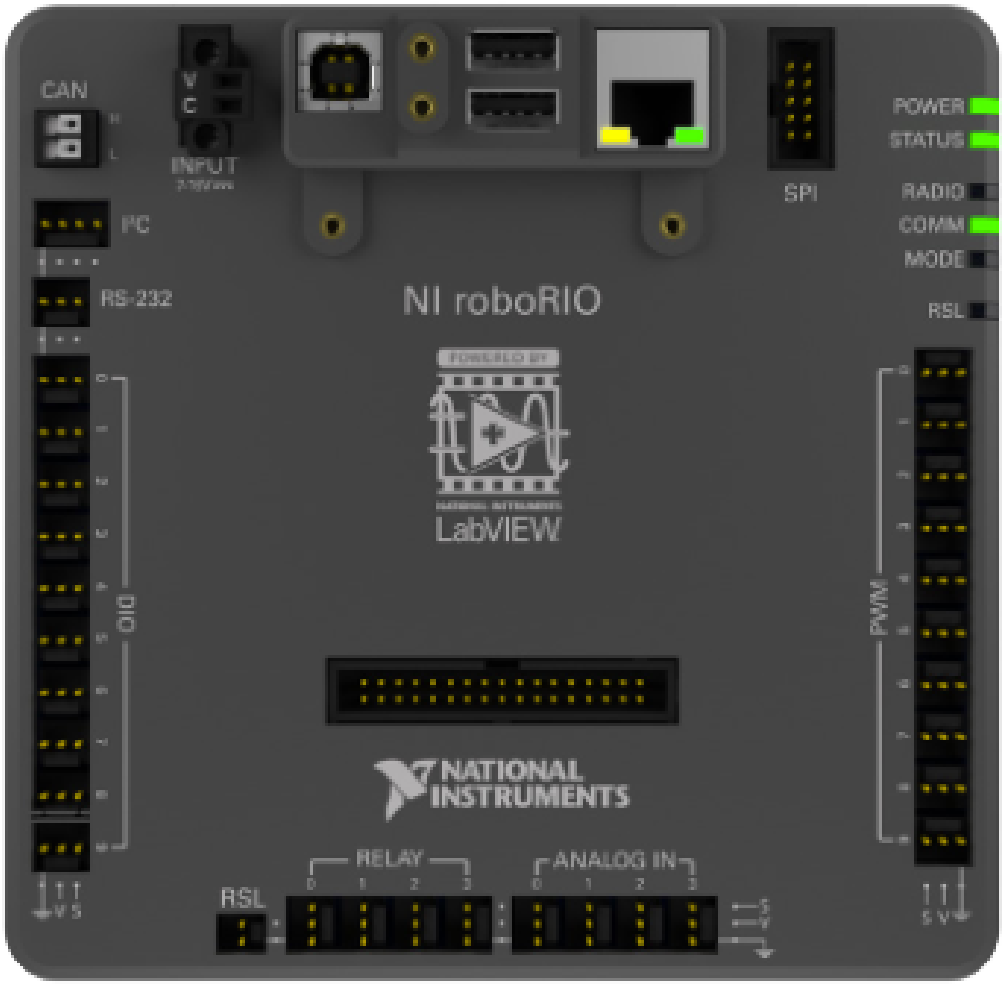


Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

6.2 NI roboRIO

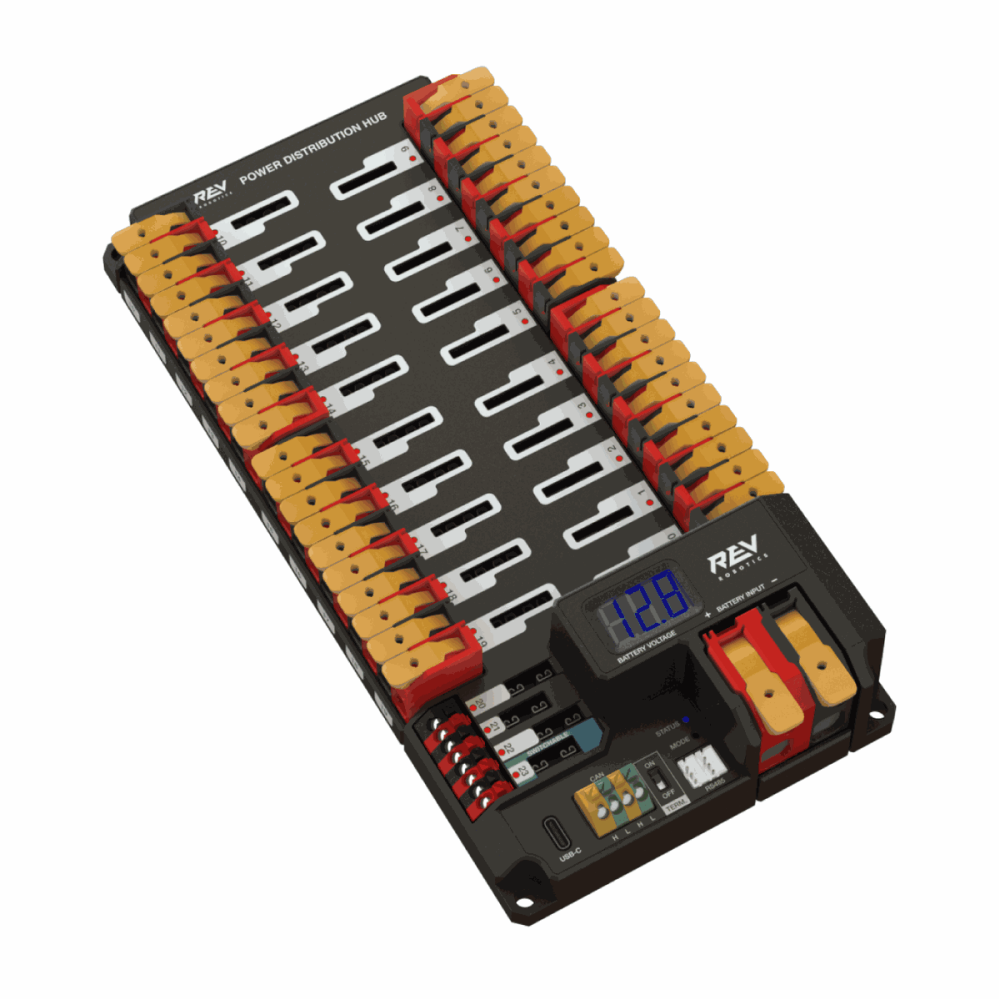


El :ref``NI-roboRIO <docs/software/roborio-info/roborio-introduction:roboRIO Introduction>` es el controlador de robot principal utilizado para FRC. El roboRIO sirve como el «cerebro» del robot que ejecuta un código generado por el equipo que controla todo el resto del hardware.

6.3 CTRE Power Distribution Panel

The *CTRE Power Distribution Panel* (PDP) is designed to distribute power from a 12VDC battery to various robot components through auto-resetting circuit breakers and a small number of special function fused connections. The PDP provides 8 output pairs rated for 40A continuous current and 8 pairs rated for 30A continuous current. The PDP provides dedicated 12V connectors for the roboRIO, as well as connectors for the Voltage Regulator Module and Pneumatics Control Module. It also includes a CAN interface for logging current, temperature, and battery voltage. For more detailed information, see the PDP User Manual.

6.4 REV Power Distribution Hub



The REV Power Distribution Hub (PDH) is designed to distribute power from a 12VDC battery to various robot components. The PDH features 20 high-current (40A max) channels, 3 low-current (15A max), and 1 switchable low-current channel. The Power Distribution Hub features toolless latching WAGO terminals, an LED voltage display, and the ability to connect over CAN or USB-C to the REV Hardware Client for real-time telemetry.

6.5 CTRE Voltage Regulator Module

The CTRE Voltage Regulator Module (VRM) is an independent module that is powered by 12 volts. The device is wired to a dedicated connector on the PDP. The module has multiple regulated 12V and 5V outputs. The purpose of the VRM is to provide regulated power for the robot radio, custom circuits, and IP vision cameras. For more information, see the VRM User Manual.

6.6 REV Radio Power Module

The REV Radio Power Module is designed to keep one of the most critical system components, the OpenMesh WiFi radio, powered in the toughest moments of the competition. The Radio Power Module eliminates the need for powering the radio through a traditional barrel power jack. Utilizing 18V Passive POE with two socketed RJ45 connectors, the Radio Power Module passes signal between the radio and roboRIO while providing power directly to the radio. After connecting the radio and roboRIO, easily add power to the Radio Power Module by wiring it to the low-current channels on the Power Distribution Hub utilizing the color coded push button WAGO terminals.

6.7 Radio OpenMesh OM5P-AN o OM5P-AC

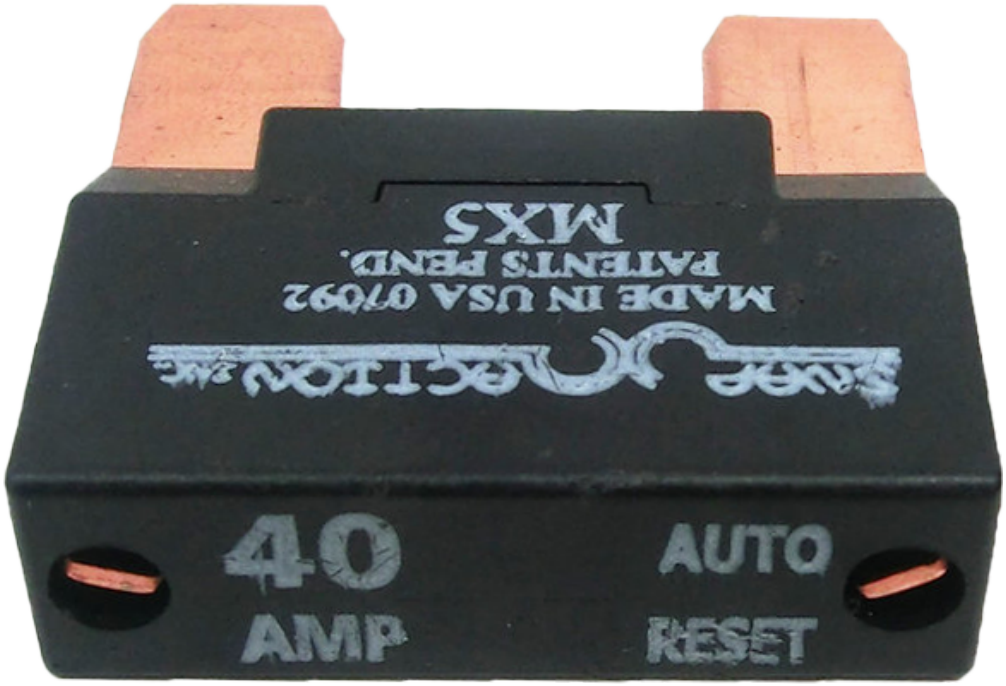
Either the OpenMesh OM5P-AN or OpenMesh OM5P-AC wireless radio is used as the robot radio to provide wireless communication functionality to the robot. The device can be configured as an Access Point for direct connection of a laptop for use at home. It can also be configured as a bridge for use on the field. The robot radio should be powered by one of the 12V/2A outputs on the VRM and connected to the roboRIO controller over Ethernet. For more information, see *Programming your Radio*.

La OM5P-AN no esta disponible para su compra. La OM5P-AC es un poco más pesada, tiene más rejillas de ventilación, y tiene una superficie más áspera en comparación a la OM5P-AN.

6.8 Interruptor de circuito 120A

El interruptor de circuito de 120A cumple dos funciones en el robot: el interruptor de alimentación principal del robot y un dispositivo de protección para el cableado y los componentes del robot después de este. El interruptor de 120A está conectado a los terminales positivos de la batería del robot y a los tableros de distribución de energía. Para obtener más información, consulte la Cooper Bussmann 18X Series Datasheet (PN: 185120F)

6.9 Interruptores de circuito de acción inmediata



The Snap Action circuit breakers, MX5 series and VB3 Series, are used with the Power Distribution Panel to limit current to branch circuits. The ratings on these circuit breakers are for continuous current, temporary peak values can be considerably higher.

6.10 Batería del robot

La fuente de alimentación de un robot FRC es una única batería de plomo ácido sellada (SLA) de 12V y 18Ah, capaz de satisfacer las altas demandas de corriente de un robot FRC. Para más información, consulte la página Batería del robot.

Nota: Pueden ser legales varios números de pieza de la batería, consulte el Manual FRC para obtener una lista completa.

6.11 Luz de Señal del Robot

Allen-Bradley

Figura 1: Allen-Bradley 855PB-B12ME522

AndyMark

The Robot Signal Light (RSL) is required to be either Allen-Bradley 855PB-B12ME522 or AndyMark am-3583. It is directly controlled by the roboRIO and will flash when enabled and stay solid while disabled.

Figura 2: AndyMark am-3583

6.12 CTRE Pneumatics Control Module

The *CTRE Pneumatics Control Module* (PCM) contains all of the inputs and outputs required to operate 12V or 24V pneumatic solenoids and the on board compressor. The PCM contains an input for the pressure sensor and will control the compressor automatically when the robot is enabled and a solenoid has been created in the code. For more information see the PCM User Manual.

6.13 REV Pneumatic Hub

The REV 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. The user selectable output voltage is fully regulated, allowing even 12V solenoids to stay active when the robot battery drops as low as 4.75V.

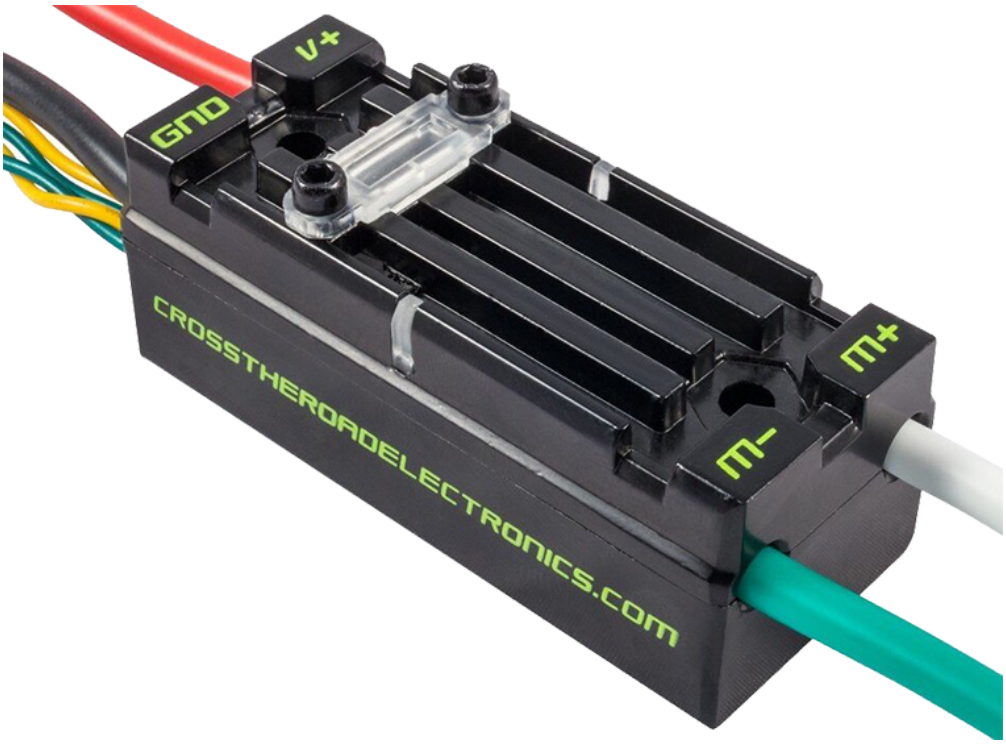
Digital and analog pressure sensor ports are built into the device, increasing the flexibility and feedback functionality of the pneumatic system. The USB-C connection on the Hub works with the REV Hardware Client, allowing users to test pneumatic systems without a need for an additional robot controller.

6.14 Controladores de motor

Hay una variedad de diferentes *controladores de motor* que funcionan con el Sistema de Control FRC y están aprobados para su uso. Estos dispositivos se utilizan para proporcionar un control de voltaje variable de los motores de CC con escobillas y sin escobillas utilizados en el FRC. Se enumeran aquí en orden de *uso <https://www.firstinspires.org/robotics/frc/blog/2021-beta-testing-usage-report>*.

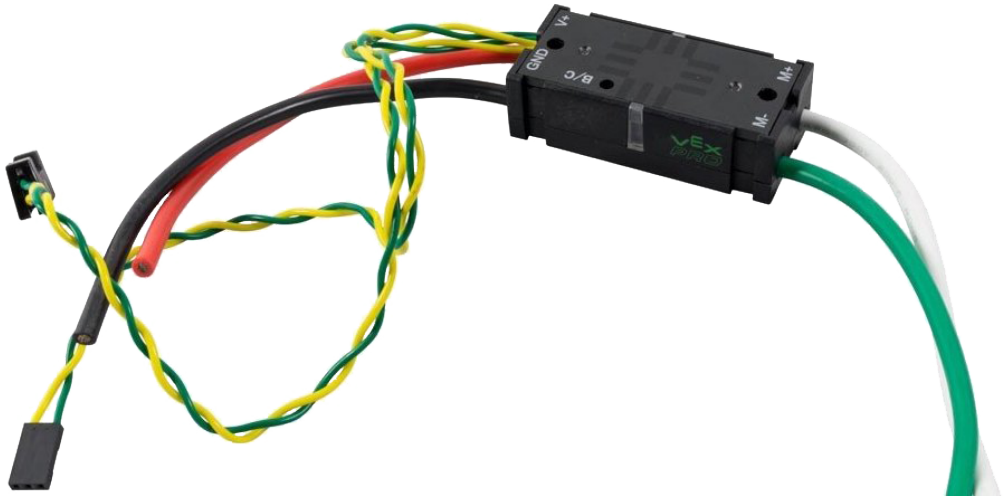
Nota: El control CAN de terceros no es compatible con WPILib. Consulte esta sección en *Dispositivos CAN de terceros* para obtener más información.

6.14.1 Talon SRX



The Talon SRX Motor Controller is a «smart motor controller» from Cross The Road Electronics/VEX Robotics. The Talon SRX can be controlled over the CAN bus or *PWM* interface. When using the CAN bus control, this device can take inputs from limit switches and potentiometers, encoders, or similar sensors in order to perform advanced control. For more information see the Talon SRX User's Guide.

6.14.2 Víctor SPX



The Victor SPX Motor Controller is a CAN or PWM controlled motor controller from Cross The Road Electronics/VEX Robotics. The device is connectorized to allow easy connection to the roboRIO PWM connectors or a CAN bus. The case is sealed to prevent debris from entering the controller. For more information, see the Victor SPX User Guide.

6.14.3 Controlador de motor SPARK MAX

El Controlador de Motor SPARK MAX es un avanzado controlador de motores DC con y sin escobillas de REV Robotics. Cuando se utiliza el bus CAN o el control USB, el SPARK MAX utiliza la entrada de los interruptores de límite, codificadores y otros sensores, incluyendo el codificador integrado del motor REV NEO Brushless, para realizar modos de control avanzados. El SPARK MAX puede ser controlado a través de PWM, CAN o USB (sólo para configuración/prueba). Para más información, consulte el Manual de Usuario del SPARK MAX.

6.14.4 Controlador de motor TalonFX

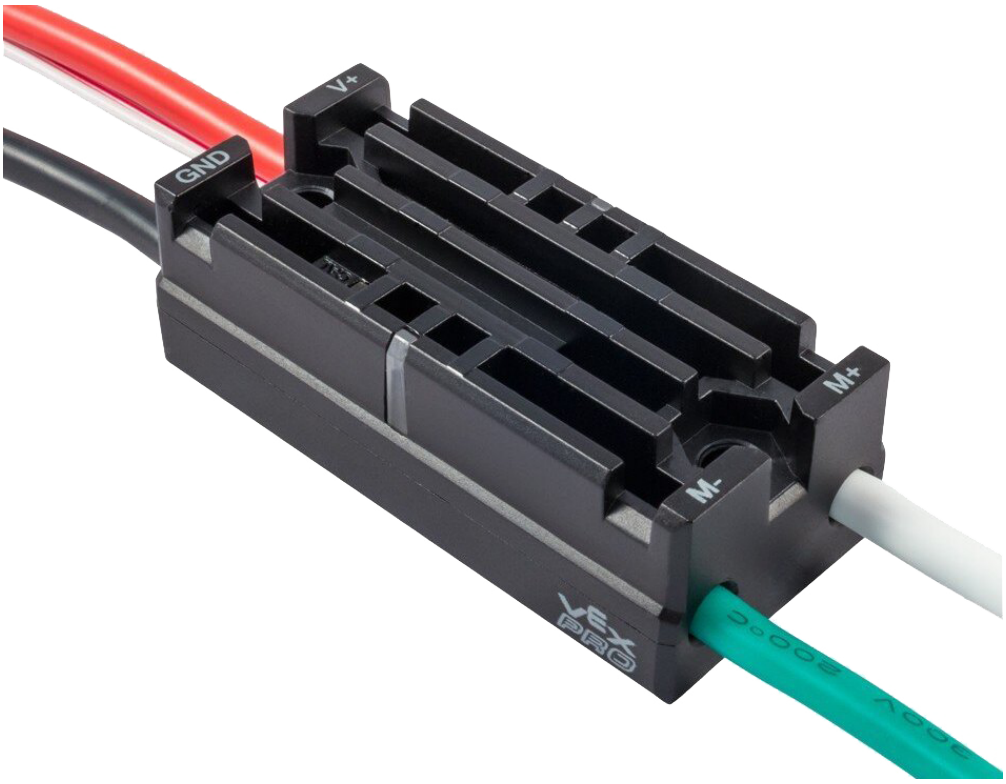
The TalonFX Motor Controller is integrated into the Falcon 500 brushless motor. It features an integrated encoder and all of the smart features of the Talon SRX and more! For more information see the Falcon 500 User Guide.

6.14.5 Controlador de motor SPARK

Advertencia: Aunque este controlador de motor es aún legal para el uso en FRC, el fabricante ha descontinuado el producto.

The SPARK Motor Controller from REV Robotics is an inexpensive brushed DC motor controller. The SPARK is controlled using the PWM interface. Limit switches may be wired directly to the SPARK to limit motor travel in one or both directions. For more information, see the SPARK User's Manual.

6.14.6 Víctor SP



Advertencia: Aunque este controlador de motor es aún legal para el uso en FRC, el fabricante ha descontinuado el producto.

The Victor SP Motor Controller is a PWM motor controller from Cross The Road Electronics/VEX Robotics. The Victor SP has an electrically isolated metal housing for heat dissipation, making the use of the fan optional. The case is sealed to prevent debris from entering the controller. The controller is approximately half the size of previous models.

6.14.7 Controlador de motor Talon

Advertencia: Aunque este controlador de motor es aún legal para el uso en FRC, el fabricante ha descontinuado el producto.

The Talon Motor Controller from Cross the Road Electronics is a PWM controlled brushed DC motor controller with passive cooling.

6.14.8 Controlador de motor Victor 888 / Controlador de motor Victor 884

Advertencia: Aunque este controlador de motor es aún legal para el uso en FRC, el fabricante ha descontinuado el producto.

Los controladores de motor Victor 884 y Victor 888 son controladores PWM de velocidad variable de VEX Robotics para ser usados en FRC. El Victor 888 reemplaza al Victor 884 que también era utilizable en FRC.

6.14.9 Controlador de motor Jaguar

Advertencia: Aunque este controlador de motor es aún legal para el uso en FRC, el fabricante ha descontinuado el producto.

El `Jaguar Motor Controller <https://www.ti.com/lit/an/spma033a/spma033a.pdf?ts=1607574399581>`__de VEX Robotics (anteriormente fabricado por Luminary Micro y Texas Instruments) es un controlador de motor de velocidad variable para su uso en FRC. Para FRC, el Jaguar solo se puede controlar mediante la interfaz PWM.

6.14.10 Controlador de motor DMC-60 y DMC-60C

Advertencia: Aunque este controlador de motor es aún legal para el uso en FRC, el fabricante ha descontinuado el producto.

El DMC-60 es un controlador de motor PWM de Digilent. El DMC-60 cuenta con detección y protección térmicas integradas que incluyen retroceso de corriente para evitar el sobrecalentamiento y daños, y cuatro LED multicolores para indicar la velocidad, la dirección y el estado para facilitar la depuración. Para obtener más información, consulte el Manual de referencia de DMC-60

El DMC-60C agrega capacidades de controlador inteligente CAN al controlador DMC-60. Debido a que el fabricante descontinuó este producto, el DMC-60C solo se puede utilizar con PWM. Para obtener más información, consulte la *Página del producto DMC-60C <https://reference.digilentinc.com/dmc-60c/start/>* __

6.14.11 Controlador de motor de veneno

The Venom Motor Controller from Playing With Fusion is integrated into a motor based on the original *CIM*. Speed, current, temperature, and position are all measured onboard, enabling advanced control modes without complicated sensing and wiring schemes.

6.14.12 Motor Nidec Dynamo BLDC con controlador

The Nidec Dynamo BLDC Motor with Controller is the first brushless motor and controller legal in FRC. This motor's controller is integrated into the back of the motor. The motor data sheet provides more device specifics.

6.14.13 Controladores de motor SD540B y SD540C

Los controladores de motor SD540B y SD540C de Mindsensors se controlan mediante PWM. El controlador CAN ya no esta disponible para el motor SD540C debido a la falta de soporte del fabricante. Los interruptores de límite pueden conectarse directamente al SD540 para limitar el recorrido del motor en una o ambas direcciones. Para obtener más información, consulte la *página FRC de Mindsensors <http://www.mindsensors.com/68-frc>* __

6.15 Spike H-Bridge Relay

Advertencia: Aunque este relé es aún legal para el uso en FRC, el fabricante ha descontinuado el producto.

El Spike H-Bridge Relay de VEX Robotics es un dispositivo que se utiliza para controlar la energía de los motores u otros dispositivos electrónicos de robot personalizados. Cuando se conecta a un motor, el Spike proporciona control de encendido/apagado en ambas direcciones hacia adelante y hacia atrás. Las salidas de Spike se controlan de forma independiente, por lo que también se pueden utilizar para proporcionar energía a hasta 2 circuitos electrónicos personalizados. El Spike H-Bridge Relay debe conectarse a una salida de relé del roboRIO y alimentarse desde el panel de distribución de energía. Para obtener más información, consulte la *Guía del usuario de Spike <https://content.vexrobotics.com/docs/spike-blue-guide-sep05.pdf>* __.

6.16 Servo Power Module

El Servo Power Module de Rev Robotics es capaz de expandir la potencia disponible para los servos más allá de lo que es capaz de hacer la fuente de alimentación integrada roboRIO. El módulo de alimentación servo proporciona hasta 90 W de potencia de 6 V en 6 canales. Todas las señales de control se transmiten directamente desde roboRIO. Para obtener más información, consulte la página web del Servo Power Module <https://www.revrobotics.com/rev-11-1144/> `__.

6.17 Microsoft Lifecam HD3000

The Microsoft Lifecam HD3000 is a USB webcam that can be plugged directly into the roboRIO. The camera is capable of capturing up to 1280x720 video at 30 FPS. For more information about the camera, see the Microsoft product page. For more information about using the camera with the roboRIO, see the *Vision Processing* section of this documentation.

6.18 Créditos de imagen

Image of roboRIO courtesy of National Instruments. Image of DMC-60 courtesy of Digilent. Image of SD540 courtesy of Mindsensors. Images of Jaguar Motor Controller, Talon SRX, Talon FX, Victor 888, Victor SP, Victor SPX, and Spike H-Bridge Relay courtesy of VEX Robotics, Inc. Image of SPARK MAX, Power Distribution Hub, Radio Power Module, and Pneumatic Hub courtesy of REV Robotics. Lifecam, PDP, PCM, SPARK, and VRM photos courtesy of *FIRST*®. All other photos courtesy of AndyMark Inc.

7 Visión general del componente de Software

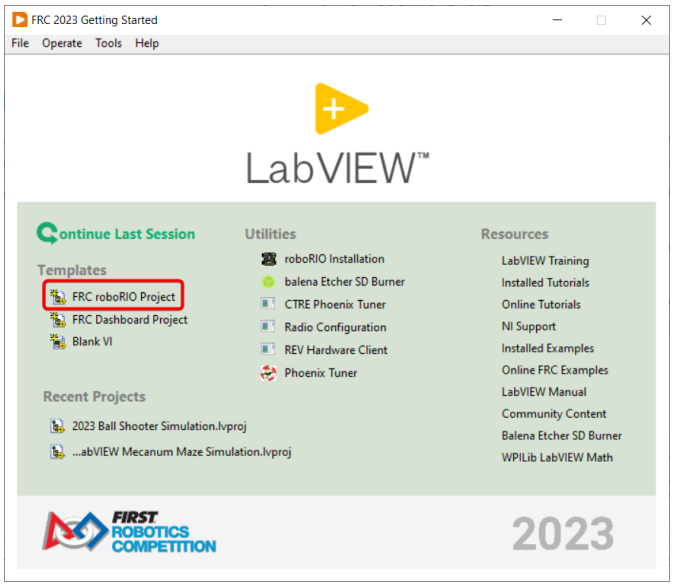
El software de FRC® consiste de una amplia variedad de componentes obligatorios y opcionales. Estos elementos están diseñados para ayudarle en el diseño, desarrollo y depuración de su código de robot así como también asistir con la operación de control del robot y proveer un feedback al momento de resolver preguntas. Para cada componente de software se dará una descripción breve y su propósito, un link al paquete de descarga si es apropiado y un link a más documentación donde esté disponible.

7.1 Compatibilidad del sistema operativo

The primary supported OS for FRC components is Windows. All required FRC software components have been tested on Windows 10 & 11.

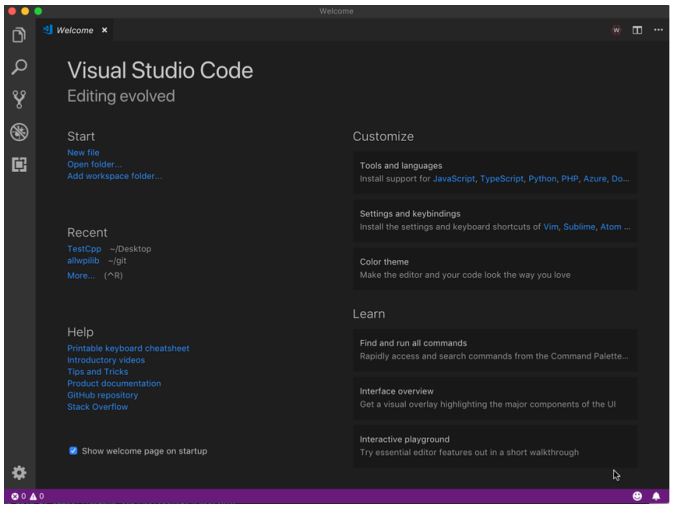
Many of the tools for C++/Java/Python programming are also supported and tested on macOS and Linux. Teams programming in C++/Java/Python should be able to develop using these systems, using a Windows system for the Windows-only operations such as the Driver Station, Radio Configuration Utility, and roboRIO Imaging Tool.

7.2 LabVIEW FRC (solo Windows)



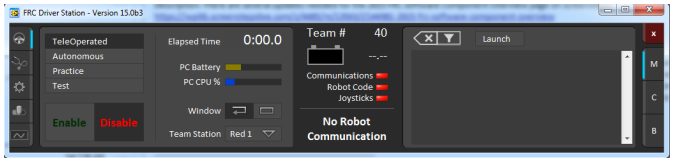
LabVIEW FRC, basado en una versión reciente de LabVIEW Professional, es uno de los tres lenguajes oficialmente soportados para programar un robot de FRC. LabVIEW es un lenguaje gráfico impulsado por flujo de datos. Los programas de LabVIEW consisten en una colección de iconos, llamados VIs, conectados entre sí con cables que pasan datos entre los VIs. El instalador de LabVIEW FRC se distribuye en un DVD que se encuentra en el Kickoff Kit of Parts y también está disponible para descargar. Puede encontrar una guía para comenzar con el software LabVIEW FRC, incluidas las instrucciones de instalación *here*.

7.3 Visual Studio Code



Visual Studio Code is the supported development environment for C++, Java. A guide to getting started with Java and C++ for FRC, including the installation and configuration of Visual Studio Code can be found *here*.

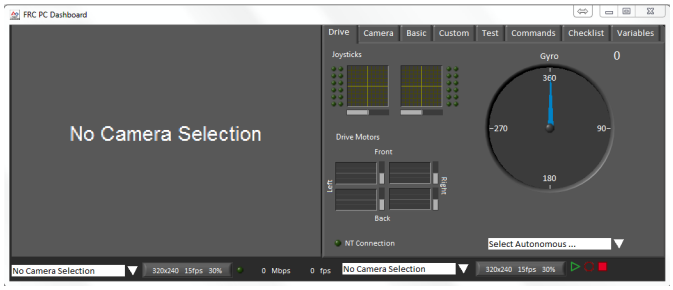
7.4 FRC Driver Station desarrollada por NI LabVIEW (Solo Windows)



Este es el único software que se puede utilizar con el fin de controlar el estado del robot durante la competición. Este software envía datos a su robot desde una variedad de dispositivos de entrada. También contiene una serie de herramientas que se utilizan para ayudar a solucionar problemas de robots. Puede encontrar más información sobre la FRC Driver Station impulsada por NI LabVIEW: ref: *here <docs/software/driverstation/driver-station:FRC Driver Station Powered by NI LabVIEW>*.

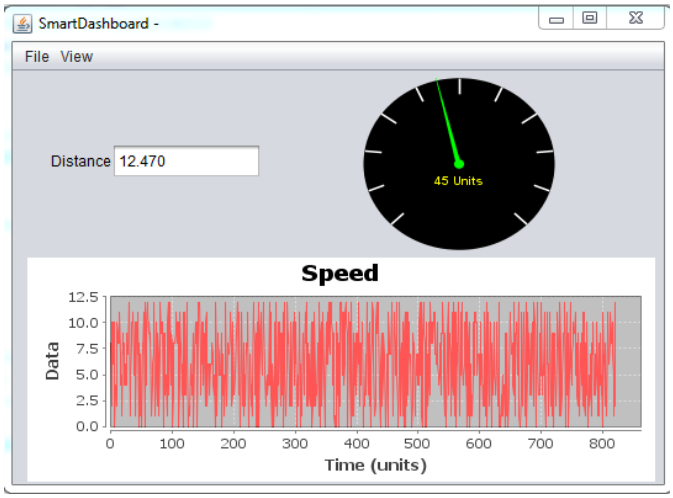
7.5 Opciones de la Dashboard

7.5.1 Dashboard de LabVIEW (solo Windows)



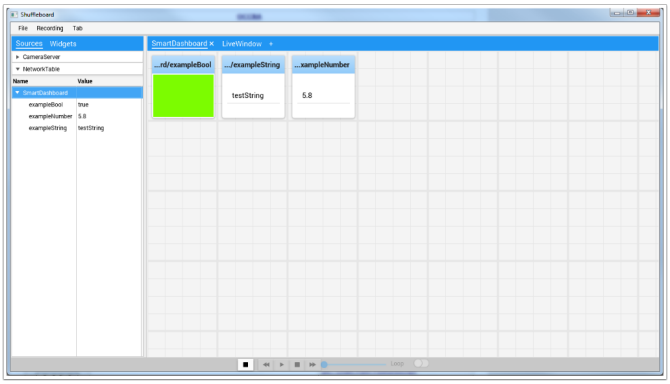
El Dashboard de LabVIEW es lanzado automáticamente por la FRC Driver Station por defecto. El propósito del Dashboard es proporcionar información sobre el funcionamiento del robot utilizando una pantalla con pestañas con una variedad de características incorporadas. Puede encontrar más información sobre el software FRC Default Dashboard *aquí*.

7.5.2 SmartDashboard



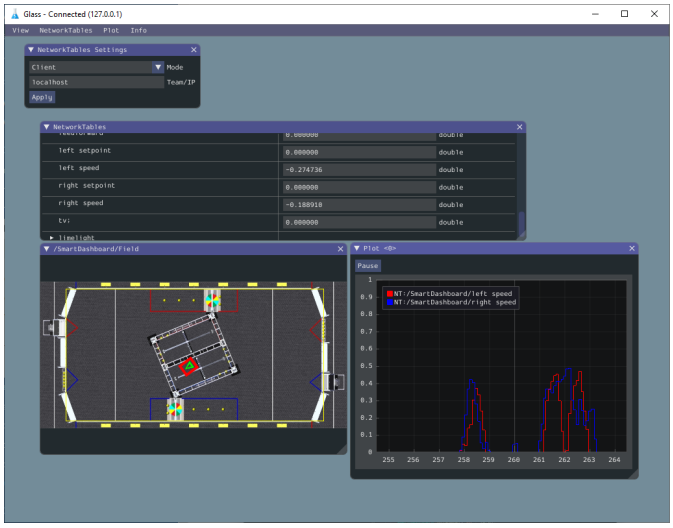
SmartDashboard le permite ver los datos de su robot creando automáticamente indicadores personalizables específicamente para cada dato enviado desde su robot. Puede encontrar documentación adicional sobre SmartDashboard *aquí*.

7.5.3 Shuffleboard



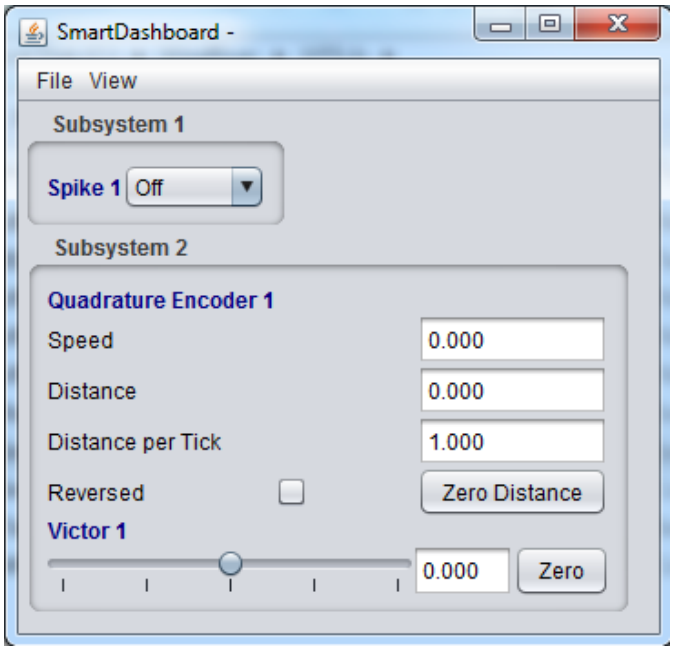
Shuffleboard tiene las mismas características que SmartDashboard. También mejora la configuración y visualización de sus datos con nuevas características y un diseño moderno a costa de ser menos eficiente en cuanto a recursos. Puede encontrar documentación adicional sobre Shuffleboard *aquí*.

7.5.4 Glass



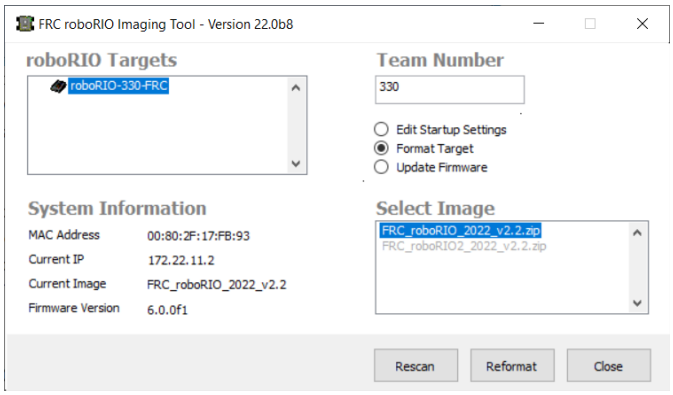
Glass es un Dashboard enfocado a ser una herramienta de depuración para el programador. Las principales ventajas son la vista de campo, la visualización de poses y las herramientas avanzadas de trazado de señales.

7.6 LiveWindow



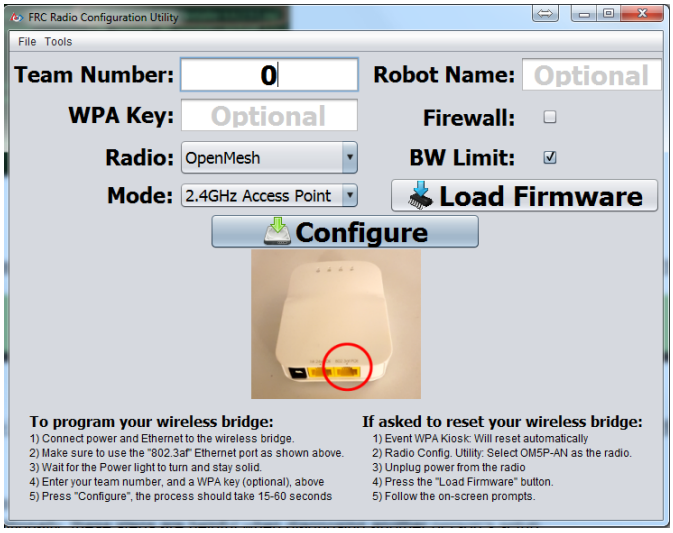
LiveWindow es una característica de SmartDashboard y Shuffleboard, diseñada para su uso con el modo de prueba de la Driver Station. LiveWindow permite al usuario ver la información de los sensores del robot y controlar los actuadores independientemente del código de usuario escrito. Se puede encontrar más información sobre LiveWindow *aquí*.

7.7 FRC roboRIO Imaging Tool (Solo Windows)



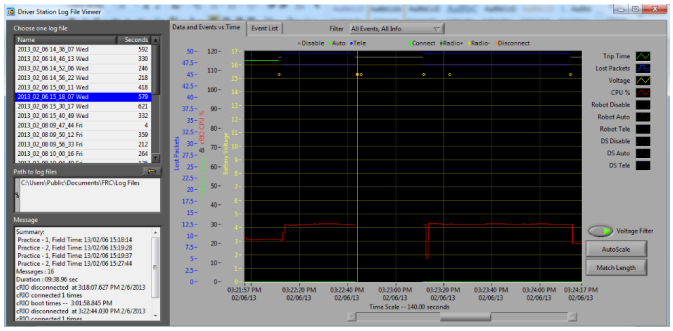
Esta herramienta se utiliza para formatear y configurar una roboRIO para su uso en FRC. Las instrucciones de instalación se pueden encontrar *aquí*. Se pueden encontrar instrucciones adicionales sobre cómo ´poner una imagen a su roboRIO usando esta herramienta *aquí*.

7.8 Utilidad de Configuración de Red para FRC (Sólo para Windows)



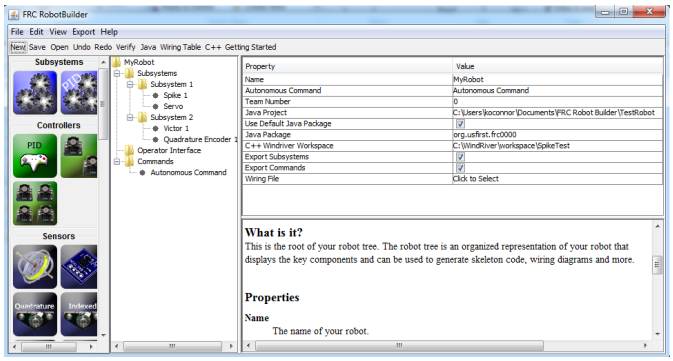
La utilidad de configuración de radio de FRC es una herramienta que se utiliza para configurar la radio estándar para uso práctico en el hogar. Esta herramienta establece la configuración de red adecuada para imitar la experiencia del campo de juego de FRC. La Utilidad de configuración de radio FRC es instalada por un instalador independiente que se puede encontrar *aquí*.

7.9 Visor de Registro de la Driver Station para FRC (Sólo para Windows)



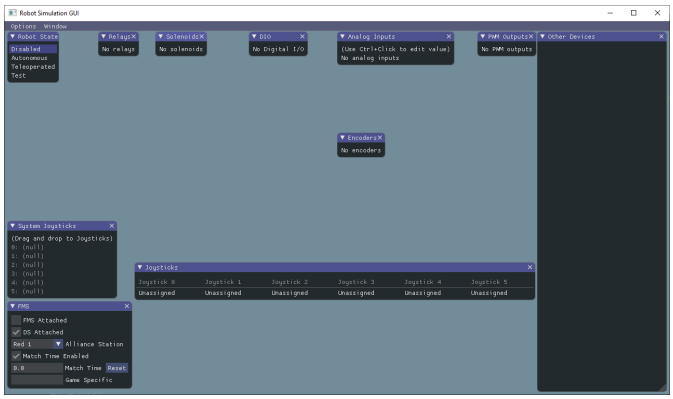
El Log Viewer de la FRC Driver Station es usada para ver los registros creados por la FRC Driver Station. Estos registros contienen una variedad de información importante para entender qué sucedió durante una sesión de practica o match de FRC. Más información acerca de la Log Viewer de la FRC Driver Station y entender los registros puede ser encontrada *aqui*

7.10 RobotBuilder



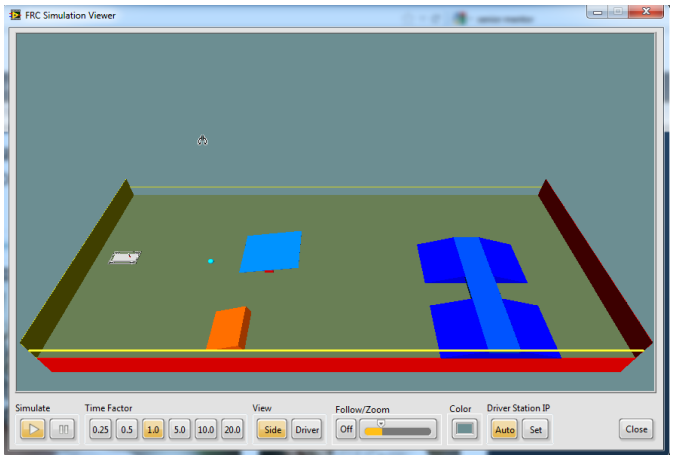
RobotBuilder is a tool designed to aid in setup and structuring of a Command Based robot project for C++ or Java (Python not currently supported). RobotBuilder allows you to enter in the various components of your robot subsystems and operator interface and define what your commands are in a graphical tree structure. RobotBuilder will then generate structural template code to get you started. More information about RobotBuilder can be found *here*. More information about the Command Based programming architecture can be found *here*.

7.11 Simulación del robot



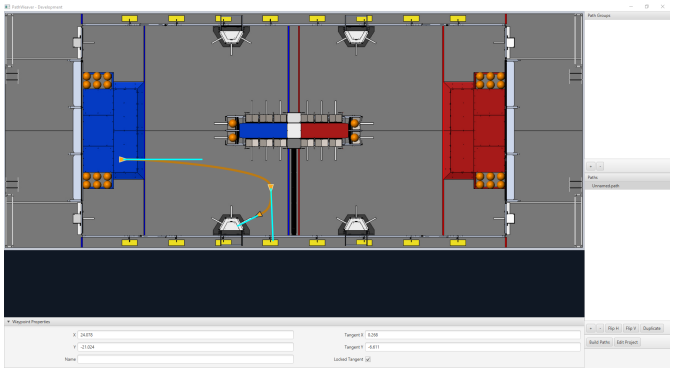
Robot Simulation offers a way for Java, C++, and Python teams to verify their actual robot code is working in a simulated environment. This simulation can be launched directly from VS Code and includes a 2D field that users can visualize their robot's movement on. For more information see the *Robot Simulation section*.

7.12 FRC LabVIEW Robot Simulator (Solo Windows)



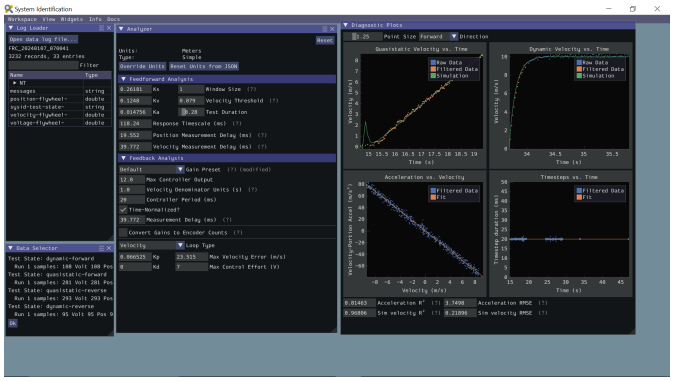
El FRC Robot Simulator es un componente del entorno de programación de LabVIEW que le permite operar un robot predefinido en un entorno simulado para probar código y/o funciones de la Driver Station. Puede encontrar información sobre el uso del FRC Robot Simulator *aquí <https://forums.ni.com/t5/FIRST-Robotics-Competition/LabVIEW-Tutorial-10-Robot-Simulation/ta-p/3739702?profile.language=en>* __ o abriendo el archivo de Robot Simulation Readme.html en el Explorador de proyectos de LabVIEW.

7.13 PathWeaver



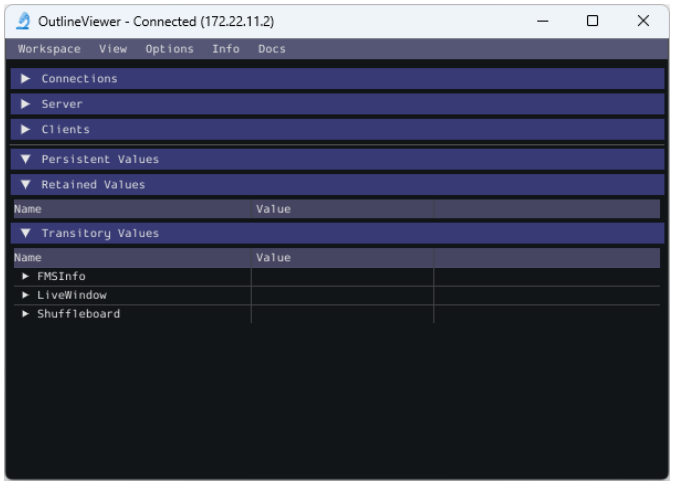
PathWeaver allows teams to quickly generate and configure paths for advanced autonomous routines. These paths have smooth curves allowing the team to quickly navigate their robot between points on the field. For more information see the *PathWeaver section*.

7.14 System Identification



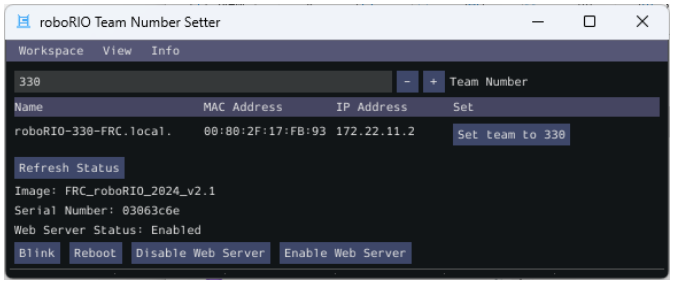
This tool helps teams automatically calculate constants that can be used to describe the physical properties of your robot for use in features like robot simulation, trajectory following, and PID control. For more information see the *System Identification section*.

7.15 OutlineViewer



OutlineViewer es una utilidad utilizada para ver, modificar y añadir a todo el contenido de las NetworkTables para propósitos de depuración. Los equipos de LabVIEW pueden utilizar la pestaña Variables del Dashboard de LabVIEW para realizar esta funcionalidad. Para más información vea la sección :ref:`Outline Viewer <docs/software/wpilib-tools/outlineviewer/index:OutlineViewer>

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 ¿Qué es 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 Idiomas admitidos

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 Documentación y código fuente

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:

El código fuente de Java y C ++ se puede encontrar en los directorios fuente de WPILibJ y WPILibC:

- Código fuente Java

- Código fuente 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 Overview de 2024

9.1 Problemas Conocidos

Este artículo detalla problemas conocidos (y soluciones) para el Software de Control de Sistema de FRC®

9.1.1 Problemas Abiertos

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.

Nota: 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 Wi-Fi 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.

Nota: 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.

Onboard I2C Causing System Lockups

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.

Updating Properties on roboRIO 2.0 may be slow or hang

Issue: Updating the properties on a roboRIO 2.0 without reformatting using the Imaging Tool (such as setting the team number) may be slow or hang.

Workaround: After a few minutes of the tool waiting the roboRIO should be able to be rebooted and the new properties should be set.

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.

Construcción inválida debido a la falta de GradleRIO

Problema: En raras ocasiones, el caché de Gradle de un usuario se rompe y se muestran errores similares a los siguientes:

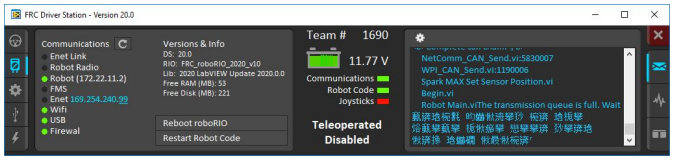
```
Could not apply requested plugin [id: 'edu.wpi.first.GradleRIO', version: '2020.3.2']␣
↪as it does not provide a plugin with id 'edu.wpi.first.GradleRIO'
```

Solución Alternativa:

Elimina tu caché de Gradle ubicado en `~$USER_HOME/.gradle`. Las máquinas Windows pueden necesitar habilitar la capacidad de ver archivos ocultos. Este problema sólo ha aparecido en Windows hasta ahora. Por favor, informe este problema si lo tiene en un sistema operativo alternativo.

Caracteres Chinos al iniciar la Driver Station

Problema: Rara vez, al iniciar la driver station se mostrarán caracteres Chinos en vez de texto en Inglés. Esto aparece solo cuando Windows esta configurado en otro idioma que no es Inglés.



Solución alternativa: Hay dos soluciones alternativas conocidas:

1. Copie y pegue los caracteres Chinos en un bloc de notas, y el texto en Inglés se mostrará.
2. Cambie temporalmente el lenguaje de Windows a Inglés.

C++ Intellisense - Los Archivos Abiertos al Iniciarse No Funcionan Correctamente

Problema: En C++, los archivos que se abren cuando se inicia VS Code tendrán problemas con Intellisense mostrando sugerencias de todas las opciones desde una unidad de compilación y no solo las apropiadas o no encontrando archivos de encabezado. Este es un error en VS Code.

Solución Alternativa:

1. Cierre todos los archivos en VS Code, pero deje VS Code abierto
2. Elimine el archivo c_cpp_properties.json en la carpeta .vscode, si existe
3. Run the «Refresh C++ Intellisense» command in VS Code.
4. En la esquina inferior derecha debe ver algo que se vea como una plataforma (linuxathena o windowsx86-64 etc). Si no es linuxathena de click y configure a linuxathena (lanzamiento)
5. Espere ~1 min
6. Abra el archivo main cpp (no el archivo de título). Intellisense debe de funcionar ahora

Issues with WPILib Dashboards and Simulation on Windows N Editions

Issue: WPILib code using CSCore (dashboards and simulated robot code) will have issues on Education N editions of Windows.

- Shuffleboard will run, but not load cameras
- Smartdashbard will crash on start-up
- Robot Simulation will crash on start-up

Solución: Intalar el Media Feature Pack

Python - CameraServer/cscore runs out of memory on roboRIO 1

Issue: When using CameraServer on a roboRIO 1, the image processing program will sometimes exit with a `SIGABRT` or «Error code 6» or a `MemoryError`.

Solution: You may be able to workaround this issue by disabling the NI webserver using the following robotpy-installer command:

```
python -m robotpy installer niweb disable
```

Ver también:

Github issue

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.

It's recommended to also review the list of *known issues*.

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 Cambios importantes (Java/C++)

Estos cambios contienen *algunos* de los principales cambios en la biblioteca que es importante que el usuario reconozca. Esto no incluye todos los cambios de ruptura, ver las otras secciones de este documento para más cambios.

- 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.

Advertencia: 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

Librería General

- 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); }};`

Cambios Importantes

- 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 Simulación

- 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

Importante: 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 Instalador de WPILib todo en uno

- 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 Visual Studio Code Extension

- 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 Descripción general de VS Code

10.1 Conceptos básicos de Visual Studio Code y la extensión WPILib

Microsoft's Visual Studio Code is the supported IDE for C++ and Java development in FRC. This article introduces some of the basics of using Visual Studio Code and the WPILib extension.

10.1.1 Pagina de bienvenida

|Pantalla de bienvenida|

Cuando se abre Visual Studio Code por primera vez, se le presenta una página de bienvenida. En esta página encontrará algunos enlaces rápidos que le permiten personalizar Visual Studio Code, así como una serie de enlaces a documentos y videos de ayuda que pueden ayudarlo a aprender sobre los conceptos básicos del IDE, así como algunos consejos y trucos.

También puede notar un pequeño logotipo de WPILib en la esquina superior derecha. Esta es una forma de acceder a las funciones proporcionadas por la extensión WPILib (que se explican más adelante).

10.1.2 Interfaz de usuario

El enlace más importante que se debe consultar es probablemente el documento básico de la interfaz de usuario. Este documento describe muchos de los conceptos básicos del uso de la interfaz de usuario y proporciona la mayor parte de la información que debe necesitar para comenzar a usar Visual Studio Code para FRC.

10.1.3 Paleta de comandos

La paleta de comandos se puede usar para acceder o ejecutar casi cualquier función o característica en Visual Studio Code (incluidas las de la extensión WPILib). Se puede acceder a la paleta de comandos desde el menú View o presionando `Ctrl+Shift+P` (`Cmd+Shift+P` en macOS). Escribir texto en la ventana reducirá dinámicamente la búsqueda a comandos relevantes y los mostrará en el menú desplegable.

En el siguiente ejemplo, se escribe «wpilib» en el cuadro de búsqueda después de activar la paleta de comandos y reduce la lista a funciones que contienen WPILib.

10.1.4 Extensión WPILib

|Comandos de WPILib|

La extensión WPILib proporciona la funcionalidad específica de FRC® relacionada con la creación de proyectos y componentes de proyectos, la construcción y descarga de código al roboRIO y más. Puede acceder a los comandos de WPILib de dos formas:

- Escribiendo «WPILib» en la paleta de comandos
- Haciendo clic en el icono de WPILib en la parte superior derecha de la mayoría de las ventanas. Esto abrirá la paleta de comandos con «WPILib» pre-ingresado

---

Nota: **No** se recomienda instalar el complemento Visual Studio IntelliCode con la instalación FRC de VS Code, ya que se sabe que rompe IntelliSense de formas extrañas.

Para obtener más información sobre los comandos específicos de la extensión WPILib, consulte los otros artículos de este capítulo.

10.2 Comandos WPILib en Visual Studio Code

Este documento contiene una lista completa de los comandos proporcionados por la extensión de código WPILib VS y lo que hacen.

Para acceder a estos comandos, presione Ctrl + Shift + P para abrir la paleta de comandos, luego comience a escribir el nombre del comando como se muestra aquí para filtrar la lista de comandos. Haga clic en el nombre del comando para ejecutarlo.

- **WPILib: Build Robot Code** - Construye un proyecto abierto usando GradleRIO
- **WPILib: Create a new project** - Crea un nuevo proyecto de robot
- **WPILib C++: Refresh C++ Intellisense**: fuerza una actualización de la configuración de 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** - cancela cualquier tarea que la extensión WPILib esté ejecutando actualmente

- **WPILib: Change Auto Save On Deploy Setting** - Cambia si los archivos se guardan automáticamente al realizar una implementación. Este valor predeterminado es Activado.
- **WPILib: Change Auto Start RioLog on Deploy Setting** - Cambie si RioLog se inicia automáticamente al implementar. Este valor predeterminado es Activado.
- **WPILib: Change Desktop Support Enabled Setting** - Cambie si está habilitada la creación de código de robot en el escritorio. Habilite esto para fines de prueba y simulación. De forma predeterminada, el soporte de escritorio está desactivado.
- **WPILib: Change Language Setting** - Cambia si el proyecto actualmente abierto es C++ o Java.
- **WPILib: Change Run Commands Except Deploy/Debug in Offline Mode Setting** - Cambie si GradleRIO se está ejecutando en modo en línea para comandos diferentes a deploy/debug (intentará extraer automáticamente las dependencias de la red). El valor predeterminado es deshabilitado (modo en línea).
- **WPILib: Change Run Deploy/Debug Command in Offline Mode Setting** - Cambia si GradleRIO se está ejecutando en modo en línea para deploy/debug (intentará extraer automáticamente las dependencias de la red). El valor predeterminado es deshabilitado (modo fuera de línea).
- **WPILib: Change Select Default Simulate Extension Setting** - Cambia si las extensiones de simulación son habilitadas predeterminadamente (todas las extensiones de simulación que sean definidas en `build.gradle` serán deshabilitadas)
- **WPILib: Change Skip Tests On Deploy Setting**: cambie si se deben omitir las pruebas en la implementación. El valor predeterminado es deshabilitado (las pruebas se ejecutan en la implementación)
- **WPILib: Change Stop Simulation on Entry Setting**: cambia si se debe detener el código del robot al ingresar al ejecutar la simulación. El valor predeterminado es deshabilitado (no se detiene al ingresar).
- **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**: compile e implemente código de robot en roboRIO en modo de depuración y comience a depurar
- **WPILib: Deploy Robot Code**: compile e implemente código de robot en 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** - Instale las herramientas de WPILib Java (por ejemplo, SmartDashboard, Shuffleboard, etc.). Tenga en cuenta que esto lo hace de forma predeterminada el instalador fuera de línea
- **WPILib: Manage Vendor Libraries** - Instalar / actualizar bibliotecas de 3ros

- **WPILib: Open API Documentation** - Abre ya sea WPILib Javadocs o la documenación C++ Doxygen
- **WPILib: Open Project Information**: abre un widget con información del proyecto (versión del proyecto, versión de extensión, etc.)
- **WPILib: Open WPILib Command Palette** - Este comando es usado para abrir un WPILib Command Palette (equivalente de presionar `Ctrl+Shift+P` y teclear `WPILib`)
- **WPILib: Open WPILib Help** - Esto abre una página simple que enlaza con la documentación de WPILib (este sitio)
- **WPILib: Reset Ask for WPILib Updates Flag**: esto borrará el indicador del proyecto actual, lo que le permitirá volver a solicitar la actualización de un proyecto a la última versión de WPILib si previamente eligió no actualizar.
- **WPILib: Run a command in Gradle**: esto le permite ejecutar un comando arbitrario en el entorno de comandos de GradleRIO
- **WPILib: Set Team Number**: se utiliza para modificar el número de equipo asociado con un proyecto. Esto solo es necesario si necesita cambiar el número de equipo del que se especificó inicialmente al crear el proyecto.
- **WPILib: Set VS Code Java Home to FRC Home** - Establezca la variable VS Code Java Home para que apunte al Java Home descubierto por la extensión FRC. Esto es necesario si no usa el instalador fuera de línea para asegurarse de que la configuración de intellisense esté sincronizada con la configuración de compilación de WPILib.
- **WPILib: Show Log Folder**: muestra la carpeta donde la extensión WPILib almacena los registros internos. Esto puede ser útil al depurar / informar un problema de extensión a los desarrolladores de 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**: esto inicia la pantalla RioLog utilizada para ver la salida de la consola desde un programa de robot
- **WPILib: Start Tool**: esto le permite iniciar herramientas WPILib (por ejemplo, SmartDashboard, Shuffleboard, etc.) desde dentro de VS Code
- **WPILib: Test Robot Code**: esto crea el proyecto de código de robot actual y ejecuta las pruebas creadas. Esto requiere que Desktop Support esté configurado en Enabled.

10.3 Creando un programa para el robot

Una vez que todo esté instalado, estamos listos para crear un programa para el robot. WPILib viene con una serie de platillas para los programas del robot. El uso de estas plantillas es altamente recomendado para los nuevos usuarios; sin embargo, usuarios avanzados son libres de hacer su código desde cero.

10.3.1 Escogiendo una Clase Base

Para iniciar un proyecto usando una de las plantillas de programa de robot de WPILib, los usuarios deben elegir una clase base para su robot. Los usuarios ponen como subclase estas clases base para crear su clase primaria para `Robot`, que controla el flujo principal del programa del robot. Existen tres opciones disponibles para la clase base:

TimedRobot

Documentation: Java - C++

Fuente: 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++)

Nota: Hay disponible una plantilla *TimedRobot Skeleton* que elimina algunos comentarios informativos y el ejemplo autónomo. Puede usar esto si ya está familiarizado con *TimedRobot*. El ejemplo que se muestra a continuación es 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
18    * initialization code.
19    */
20   @Override
21   public void robotInit() {}
22
23   @Override
24   public void robotPeriodic() {}
25
26   @Override
27   public void autonomousInit() {}
```

(continúe en la próxima página)

(proviene de la página anterior)

```

  @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() {}

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

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

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

(continúe en la próxima página)

(proviene de la página anterior)

```
}
#endif
```

Los métodos periódicos son llamados cada 20 ms por defecto. Esto puede cambiarse llamando la superclase constructor con la nueva actualización deseada.

Peligro: 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++

Fuente: Java - C++

La clase `RobotBase` es la clase base más mínima que se ofrece y, por lo general, no se recomienda para uso directo. No se maneja ningún flujo de control del robot para el usuario; todo debe escribirse desde cero dentro del método `startCompetition()`. La plantilla por defecto muestra cómo procesar los diferentes modos de operación (teleop, auto, etc.).

Nota: Hay disponible una plantilla `` RobotBase Skeleton`` que ofrece un método `startCompetition ()` en blanco.

Robot a base de comandos

The `Command Robot` framework adds to the basic functionality of a `Timed Robot` by automatically polling inputs and converting the raw input data into events. These events are tied to user code, which is executed when the event is triggered. For instance, when a button is pressed, code tied to the pressing of that button is automatically called and it is not necessary to poll or keep track of the state of that button directly. The `Command Robot` framework makes it easier to write compact easy-to-read code with complex behavior, but requires an additional up-front time investment from a programmer in order to understand how the Command Robot framework works.

Teams using `Command Robot` should see the *Command-Based Programming Tutorial*.

Romi

Los equipos que estén utilizando *Romi* deberían de utilizar la plantilla `Romi - Timed` o `Romi - Command Bot`.

Romi - Timed

La plantilla `Romi - Timed` proporciona la clase `RomiDrivetrain` que expone un método `arcadeDrive(double xaxisSpeed, double zaxisRotate)` . Depende del usuario alimentar esta función arcadeDrive.

Esta clase también provee funciones para recuperar y restablecer los encoders integrados de Romi.

Romi - Robot a base de comandos

La plantilla `Romi - Command Bot` proporciona un subsistema `RomiDrivetrain` que expone un método `arcadeDrive (double xaxisSpeed, double zaxisRotate)` Depende del usuario alimentar esta función arcadeDrive.

Este subsistema también proporciona funciones para recuperar y restablecer los codificadores integrados de Romi.

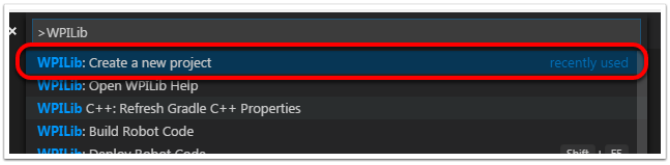
No usando una clase base

Si así lo desean, los usuarios pueden omitir una clase base entera y simplemente escribir el programa en un método `main()`, como sería para cualquier otro programa. Esto es *altamente* no recomendado - los usuarios no deberían «reinventar la rueda» cuando escriben el código de su robot - pero esto está soportado para quienes quieren tener control absoluto en el flujo de su programa.

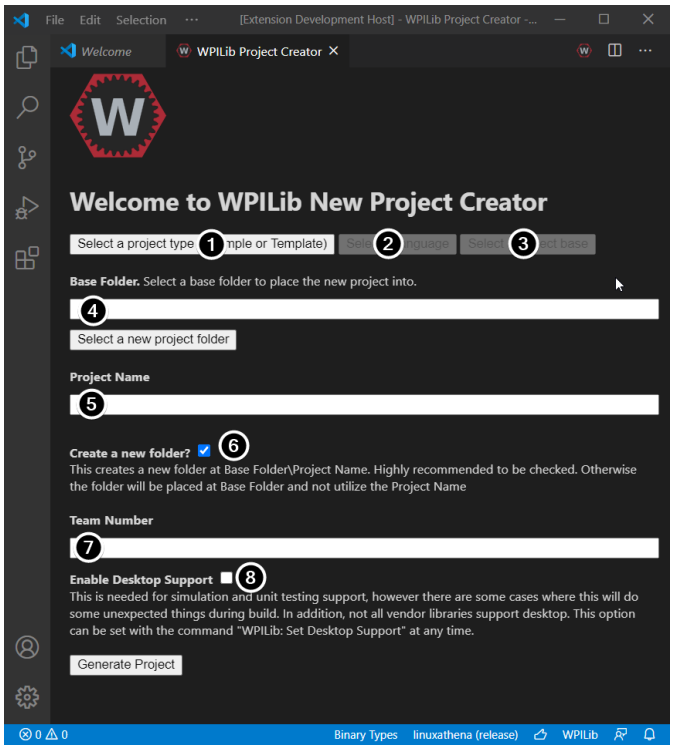
> **Advertencia:** Los usuarios *no* deberían modificar el método `main()` de un programa de robot a menos que estén absolutamente seguros de lo que están haciendo.

10.3.2 Creando un Proyecto Nuevo de WPILib

Once we've decided on a base class, we can create our new robot project. Bring up the Visual Studio Code command palette with `Ctrl+Shift+P`. Then, type «WPILib» into the prompt. Since all WPILib commands start with «WPILib», this will bring up the list of WPILib-specific VS Code commands. Now, select the *Create a new project* command:



Esto va a abrir la «New Project Creator Window:»



Los elementos de la New Project Creator Window o Ventana del Creador de Nuevo Proyecto está explicado abajo:

1. **Tipo de Proyecto**: El tipo de proyecto que queremos crear. Esto puede ser un proyecto de ejemplo, o uno de los proyectos modelo brindados por WPILib. Los modelos existen para cada clase de base del robot. Adicionalmente, un modelo existe para proyectos *Command-based*, que están hechos en la clase base `TimedRobot` pero incluye un número de características adicionales - este tipo de programa de robot es altamente recomendado para equipos nuevos.
2. **Lenguaje**: Este es el lenguaje (C++ o Java) que será usado en el proyecto.
3. **Carpeta base**: Si se trata de un proyecto de plantilla, esto especifica el tipo de plantilla que se utilizará.
4. **Localización del Proyecto**: Esto determina la carpeta en la que el proyecto del robot estará localizado.
5. **Nombre del Proyecto**: El nombre del proyecto del robot. Esto también especifica el nombre que la carpeta del proyecto tendrá si la caja de Crear Nueva Carpeta está marcada.
6. **Cree una Nueva Carpeta**: Si se marca esta casilla, se creará una nueva carpeta para mantener el proyecto dentro de la carpeta previamente especificada. Si se marca *no*, el proyecto se ubicará directamente en la carpeta previamente especificada. Se producirá un error si la carpeta no está vacía y no está marcada.
7. **Número del Equipo**: El número del equipo para la carpeta, que es el que se usará para empacar nombres con el proyecto y localizarlo en el robot cuando se corra el código.
8. **Habilitar soporte de escritorio *: Habilita la prueba unitaria y la simulación. Si bien WPILib admite esto, es posible que las bibliotecas de software de terceros no. Si las bibliotecas no son compatibles con el escritorio, es posible que su código no se compile

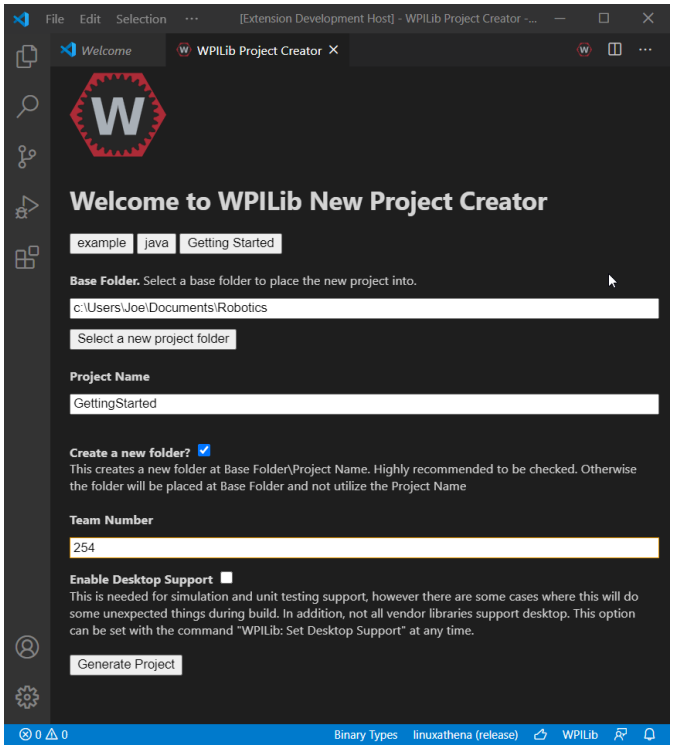
o se bloquee. Debe dejarse sin marcar a menos que se necesiten pruebas unitarias o simulación y todas las bibliotecas lo admitan.

Una vez que todo lo de arriba está configurado, de click en «Generar Proyecto» y el proyecto del robot será creado.

Nota: Ningún error en la generación de proyecto debe aparecer en la esquina inferior derecha de la pantalla.

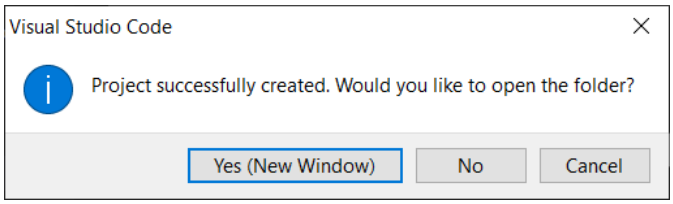
Advertencia: Creating projects on OneDrive is not supported as OneDrive's caching interferes with the build system. Some Windows installations put the Documents and Desktop folders on OneDrive by default.

Un ejemplo de todas las opciones seleccionadas se muestra abajo.

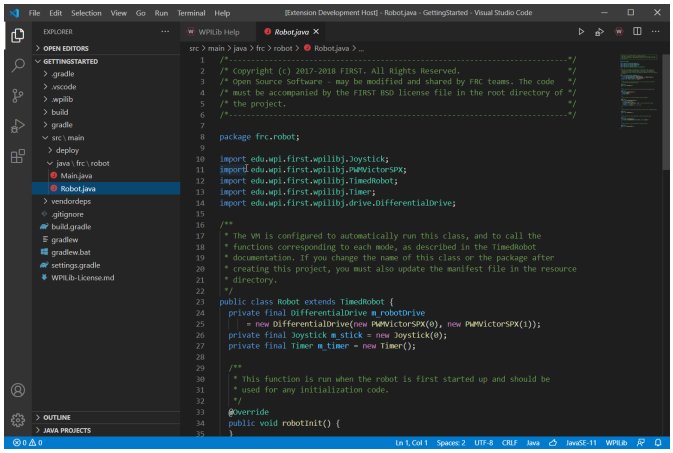


10.3.3 Abriendo El Nuevo Proyecto

After successfully creating your project, VS Code will give the option of opening the project as shown below. We can choose to do that now or later by typing `Ctrl+K` then `Ctrl+O` (or just `Command+O` on macOS) and select the folder where we saved our project.

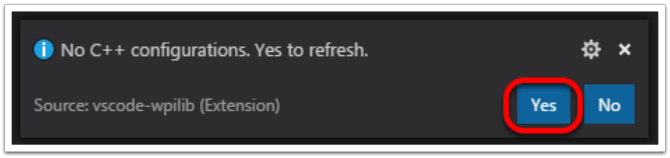


Una vez que abrimos, vamos a ver el proyecto jerarquizado a la izquierda. Dando doble click en el archivo abrirá ese archivo en el editor.



10.3.4 Configuraciones de C++ (Solo C++)

Para proyectos C++, hay un paso más para configurar IntelliSense. Cada vez que abrimos un proyecto, debería aparecernos una ventana emergente en la esquina inferior derecha pidiendo actualizar las configuraciones de C++. Haga clic en «Yes» para configurar IntelliSense



10.4 Bibliotecas de 3ros

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.

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

Nota: 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.

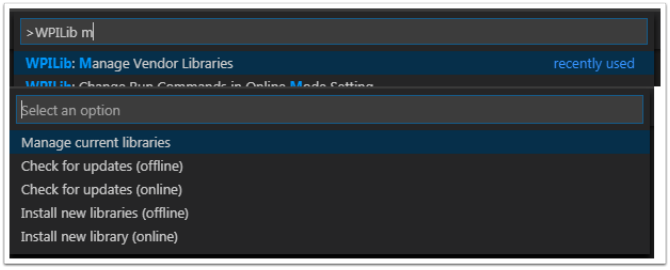
How Does It Work? - Python

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.

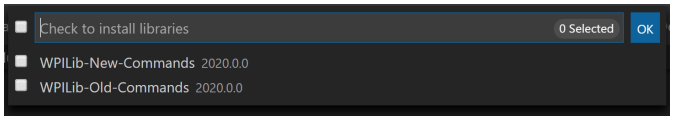
Installing Libraries

Java/C++

VS Code



Para añadir una librería vendor que se ha instalado por un instalador fuera de línea, presione `Ctrl+Shift+P` y escriba WPILib o haga clic en el icono de WPILib en la esquina superior derecha para abrir la Paleta de Comandos de WPILib y empiece a escribir *Manage Vendor Libraries*, después selecciónelo en el menú. Seleccione la opción de *Install new libraries (offline)*.



Seleccione las bibliotecas deseadas para agregarlas al proyecto marcando la casilla situada junto a cada una y, a continuación, haga clic en *OK*. El archivo JSON se copiará en la carpeta `vendordeps` del proyecto, agregando la biblioteca como una dependencia al proyecto.

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)

Parte del archivo JSON que los proveedores pueden completar opcionalmente es una ubicación de actualización en línea. Si una biblioteca tiene una ubicación especificada, ejecutar *Check for updates (online)* comprobará si hay una versión más nueva de la biblioteca disponible desde la ubicación remota.

Removing a Library Dependency

Para eliminar una dependencia de biblioteca de un proyecto, seleccione *Manage Current Libraries* del menú *Manage Vendor Libraries*, marque la casilla de las biblotecas que desee eliminar y haga clic en *OK*. Estas bibliotecas serán eliminadas como dependencias del proyecto.

Command-Line

También se puede agregar una dependencia de biblioteca de proveedor desde la URL del proveedor a través de la línea de comandos a través de una tarea gradle. Abra una instancia de la línea de comandos en la raíz del proyecto e ingrese `gradlew vendordep--url=<url>` donde``<url>`` es la URL JSON del proveedor. Esto agregará el archivo JSON de dependencia de la biblioteca del proveedor a la carpeta `vendordeps` del proyecto. Las bibliotecas de proveedores se pueden actualizar de la misma manera.

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`

Ver también:

pyproject.toml usage

10.4.3 Bibliotecas

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++

New Command Library

Python

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

Romi Library

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`

Nota: 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 <https://www.playingwithfusion.com/docview.php?docid=1205> __ - Biblioteca para todos los dispositivos PWF, incluido el motor / controlador 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 - Bibliotecas para NavX-MXP, NavX-Micro, y 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"]`

Bibliotecas comunitarias

PhotonVision - Biblioteca para el software PhotonVision CV

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

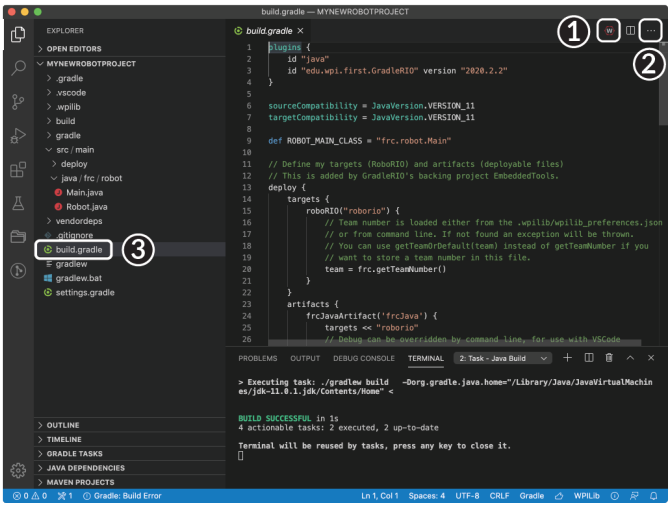
Not available

10.5 Creación e implementación del código del robot

Los proyectos del robot deben compilarse («construirse») e implementarse para que puedan ejecutarse en la roboRIO. Dado que el código no se compila de forma inicial en el controlador del robot, esto se conoce como «cross-compilation.»

Para construir e implementar un proyecto de robot, realice una de las siguientes acciones:

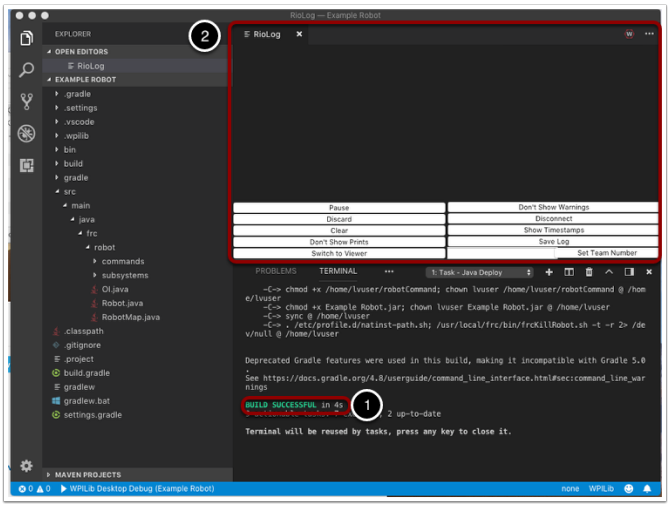
1. Abra la paleta de comandos e introduzca / seleccione «Build Robot Code»
2. Abra el menú de acceso directo indicado por las elipses en la esquina superior derecha de la ventana de VS Code y seleccione «Build Robot Code»
3. Haga clic con el botón derecho en el archivo build.gradle en la jerarquía del proyecto y seleccione» 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.

Advertencia: 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 Visualización de la salida de la consola

Para ver la salida de la consola de programas basados en texto, roboRIO implementa una NetConsole. Hay dos formas principales de ver la salida de NetConsole desde roboRIO: el visor de consola en FRC Driver Station y el complemento Riolog en VS Code.

Nota: En roboRIO, NetConsole es solo para salida de programa. Si desea interactuar con la consola del sistema, deberá utilizar SSH o la consola en serie.

10.6.1 Console Viewer

Abrir el Console Viewer

|Abrir el Console Viewer|

Para abrir Console Viewer, primero abra FRC® Driver Station. Luego, haga clic en el engrane en la parte superior de la ventana del visor de mensajes (1) y seleccione «Ver consola».

Ventana del Console Viewer

|Ventana del Console Viewer|

La ventana del Console Viewer muestra la salida de nuestro programa de robot en verde. El engranaje en la parte superior derecha puede limpiar la ventana y establecer el nivel de mensajes que se muestran.

10.6.2 Riolog VS Complemeto Code

El complemento Riolog es una vista de VS Code que se puede usar para ver la salida de NetConsole en VS Code (crédito por la versión original de Eclipse: Manuel Stoeckl, FRC1511).

Abrir la vista RioLog

|Abrir vista Riolog|

De forma predeterminada, la vista RioLog se abrirá automáticamente al final de cada implementación de roboRIO. Para iniciar la vista RioLog manualmente, presione:kbd:*Ctrl+Shift+P* para abrir la paleta de comandos y comience a escribir «RioLog», luego seleccione la opción WPILib: Iniciar RioLog.

Ventana Riolog

|Ventana Riolog|

La vista RioLog debería aparecer en el panel superior. El Riolog contiene una serie de controles para manipular la consola:

- **Pausar/reanudar la visualización** - esto pausará / reanudará la visualización. En segundo plano, los nuevos paquetes se seguirán recibiendo y se mostrarán cuando se haga clic en el botón Reanudar.
- **Descartar/Aceptar entrantes** - esto alternará entre aceptar nuevos paquetes. Cuando se descartan paquetes, la pantalla se detendrá y todos los paquetes recibidos se descartarán. Al hacer clic en el botón nuevamente, se reanudará la recepción de paquetes.
- **Clear** - esto borrará el contenido actual de la pantalla.
- **No mostrar/mostrar impresiones** - muestra u oculta mensajes categorizados como declaraciones impresas
- **Cambiar a Visualizador** - esto cambia a visualizador para los archivos de registro guardados
- **No mostrar/mostrar advertencias** - muestra u oculta los mensajes clasificados como advertencias
- **Desconectar/Volver a conectar** - esto desconecta o vuelve a conectar a la transmisión de la consola
- **Mostrar/No mostrar marcas de tiempo** - muestra u oculta las marcas de tiempo en los mensajes de la ventana
- **Guardar registro**: copia el contenido del registro en un archivo que puede guardar y ver o abrir más tarde con el visor RioLog (consulte Cambiar al visor más arriba)
- **Establecer número de equipo** - establece el número de equipo del roboRIO para conectarse a la transmisión de la consola, se establece automáticamente si el proceso de implementación inicia RioLog

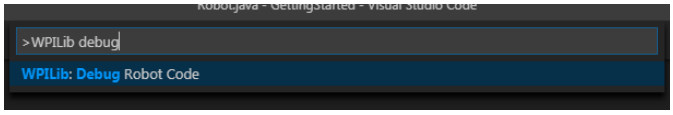
10.7 Depurar un Programa de Robot

Inevitablemente, un programa no se comportará de la manera que esperamos que se comporte. Cuando esto ocurre, es necesario averiguar por qué el programa está haciendo lo que está haciendo, para que podamos hacer que haga lo que queremos que haga, en su lugar. Este comportamiento de programa no deseado se denomina «error», y este proceso se denomina «depuración».

Un depurador es una herramienta que se utiliza para controlar el flujo del programa y monitorear las variables para ayudar a depurar un programa. Esta sección describirá cómo configurar una sesión de depuración para un FRC® programa de robot.

Nota: Para los usuarios principiantes que necesitan depurar sus programas pero no saben/tienen tiempo para aprender a usar un depurador, a menudo es posible depurar un programa simplemente imprimiendo el estado del programa relevante en la consola. Sin embargo, se recomienda encarecidamente que los alumnos finalmente aprendan a usar un depurador.

10.7.1 Ejecución del depurador



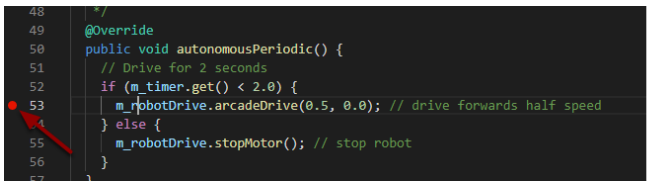
Presione `Ctrl+Mayús+P` y escriba `WPILib` o haga clic en el elemento *Menú WPILib* para abrir la paleta Comandos con WPILib rellenado previamente. Escriba Depurar y seleccione el elemento de menú Depurar código robot para iniciar la depuración. El código se descargará en la roboRIO y comenzará la depuración.

10.7.2 Breakpoints

Un «breakpoint» es una línea de código en la que el depurador detendrá la ejecución del programa para que el usuario pueda examinar el estado del programa. Esto es extremadamente útil durante la depuración, ya que permite al usuario pausar el programa en puntos específicos en código problemático para determinar dónde exactamente el programa se está desviando del comportamiento esperado.

El depurador se detendrá automáticamente en el primer breakpoint que encuentre.

Estableciendo un Breakpoint



Haga doble clic en el margen izquierdo de la ventana del código fuente (a la izquierda del numero de línea) para establecer un breakpoint en el programa: un pequeño círculo rojo indicará que el breakpoint se ha establecido en la línea correspondiente.

10.7.3 Depurar con declaraciones impresas

Otra forma de depurar su programa es usar declaraciones de impresión en su código y verlas usando RioLog en Visual Studio Code o Driver Station. Las declaraciones impresas deben agregarse con cuidado, ya que no son muy eficientes, especialmente cuando se usan en grandes cantidades. Deben eliminarse para la competencia, ya que pueden provocar desbordamientos de bucle.

JAVA

```
System.out.print("example");
```

C++

```
wpi::outs() << "example\n";
```

10.7.4 Debugging with NetworkTables

NetworkTables can be used to share robot information with your debugging computer. *NetworkTables* can be viewed with your favorite Dashboard or *OutlineViewer*. One advantage of NetworkTables is that tools like *Shuffleboard* can be used to graphically analyze the data. These same tools can then be used with same data to later provide an operator interface for your drivers.

10.7.5 Aprender más

- Para aprender más acerca de la depuración con VS Code visite este link.
- Algunas de las características mencionadas en este artículo de VS Code lo ayudarán a entender y diagnosticar problemas con su código. La función Quick Fix (Bombilla amarilla) puede ser muy útil con una variedad de problemas incluyendo qué importar.
- Una de las mejores maneras para prevenir el tener que depurar tantos problemas es hacer una prueba de la unidad.
- Verificar que su robot funcione en *Simulation* es también una excelente forma de evitar tener que hacer depuraciones complejas en el robot actual.

10.8 Importing Last Year's Robot Code

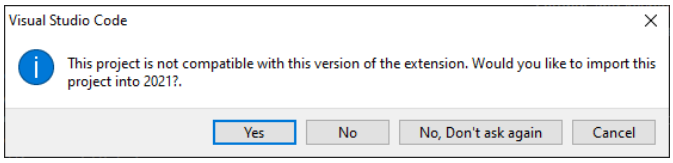
Due to changes in the project, it is necessary to update the build files for a previous years Gradle project. It is also necessary to import vendor libraries again, since last year's vendor libraries must be updated to be compatible with this year's projects.

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.

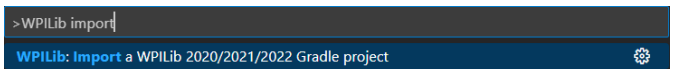
Importante: The import process copies your project source files from the current directory to a new directory and completely regenerates the gradle files. Additionally, it updates the code for the package changes made in 2023. If you made non-standard updates to the `build.gradle`, you will need to make 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.

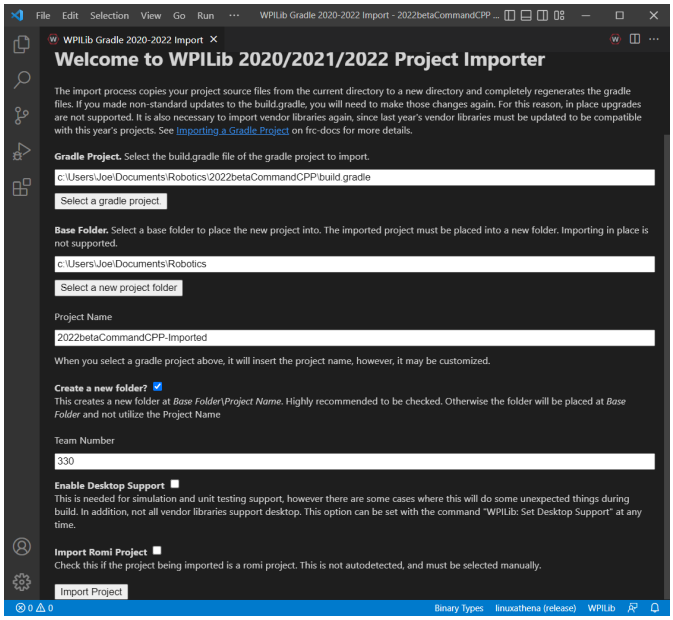
Launching the Import Wizard



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.





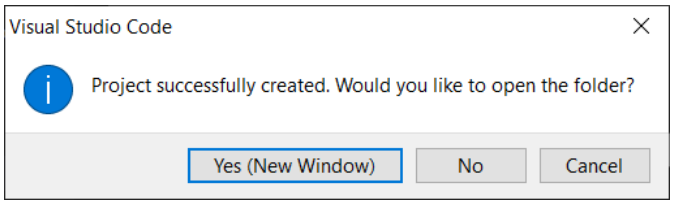
You'll be presented with the WPILib Project Importer window. This is similar to the process of creating a new project and the window and the steps are shown below. This window contains the following elements:

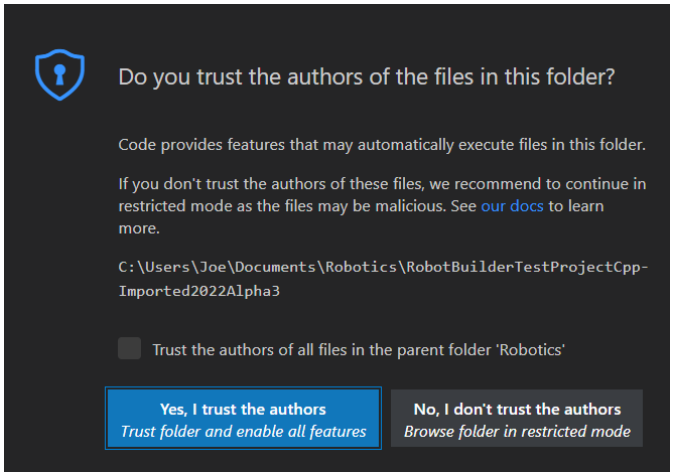
1. **Gradle Project**: Selects the project to be imported. Users should select the build.gradle file in the root directory of the gradle project.
2. **Project Location**: This determines the folder in which the robot project will be located.
3. **Project Name**: The name of the robot project. This also 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.

> **Advertencia:** 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.

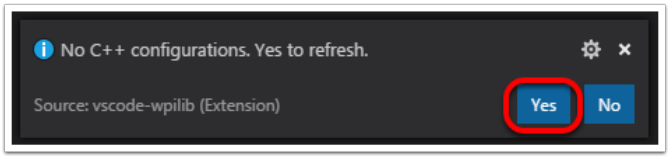




Click *Yes I trust the authors*.

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)

Nota: 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.

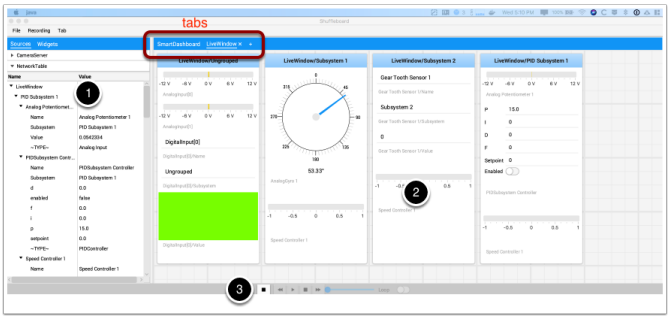
11.2.1 Shuffleboard- Empezando

Guía por 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:

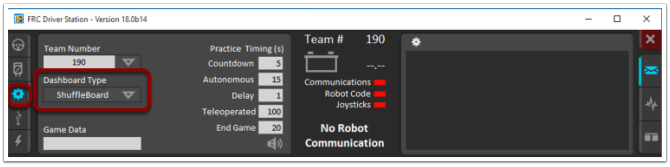
- Los gráficos están basados en **JavaFX**, el estándar de gráficas de Java. Cada uno de sus componentes tiene una hoja de estilos asociada, entonces se vuelve posible tener diferentes "diseños" o "temas" para la Shuffleboard. Nosotros proporcionamos los temas claro y oscuro de manera predeterminada.
- La Shuffleboard soporta **múltiples páginas para la mostrar tus datos**. De hecho, usted puede crear una nueva página (mostrada como una pestaña en la ventana de la Shuffleboard) e indicar si y cuales datos deben auto completar en esta. De manera predeterminada hay una pestaña de Test y una pestaña de SmartDashboard que se auto completan a medida que los datos llegan. Otras ventanas pueden ser para la depuración del robot vs. driving.
- Los **elementos de visualización gráfica (widgets) son mostrados en una cuadrícula** para mantener la interfaz ordenada y que sea fácil de leer. Puede cambiar el tamaño de la cuadrícula para que tenga una mayor o menor resolución en sus diseños y se proporcionan señales visuales para ayudarle a cambiar el diseño usando arrastrar y soltar. O usted puede elegir quitar las líneas de la cuadrícula, sin embargo el diseño de esta se mantendrá.
- Los diseños son guardados y el diseño previo es instanciado de forma predeterminada cuando vuelva a iniciar la Shuffleboard.
- Existe la función de **record and playback**, grabar y repetir, que le permite revisar los datos enviados por el programa de su robot después que termina. De esta manera usted puede revisar detalladamente las acciones del robot si algo va mal.
- **Los widgets gráficos están disponibles para datos numéricos** y usted puede arrastrar los datos dentro de una gráfica para ver varios puntos al mismo tiempo y en la misa escala.
- 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. **Área de orígenes:** Aquí están las fuentes de datos, usted puede elegir valores de Network – Tables o de otras fuentes para mostrar al arrastrar un valor dentro uno de las pestañas.



2. **Paneles de pestañas:** Aquí es donde sus datos son mostrados desde el robot u otras fuentes. En este ejemplo se muestran los subsistemas de Test-mode en la pestaña de LiveWindow. Este área puede mostrar cualquier número de ventanas, y cada ventana tiene su propio conjunto de propiedades, como el tamaño de la cuadrícula y la función auto completar.
3. **Controles de Record /Playback:** conjunto de controles donde usted puede repetir la sesión actual para ver el historial de datos.

Iniciar la Shuffleboard



Usted puede iniciar la Shuffleboard con una de cuatro maneras:

1. Puede iniciarla automáticamente cuando la Driver Station inicia al configurar el “Dashboard Type” seleccionando la opción Shuffleboard en la pestaña de configuraciones, como se muestra en la foto de arriba.
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. Puede ejecutarlo haciendo doble clic en el archivo shuffleboard.XXX (donde XXX es `.vbs` en Windows y `.py` en Linux o macOS) en `~/WPILib/YYYY/tools/` (donde YYYY es el año y `~` es `C:\Users\Public` en Windows). Esto es útil en un sistema de desarrollo que no tiene la Driver Station instalada, como un sistema macOS o 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.

Nota: Los scripts `.vbs` (Windows) y `.py` (macOS/Linux) ayudan a iniciar las herramientas utilizando el JDK correcto.

Ingresar datos del robot en la Dashboard.

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());
```

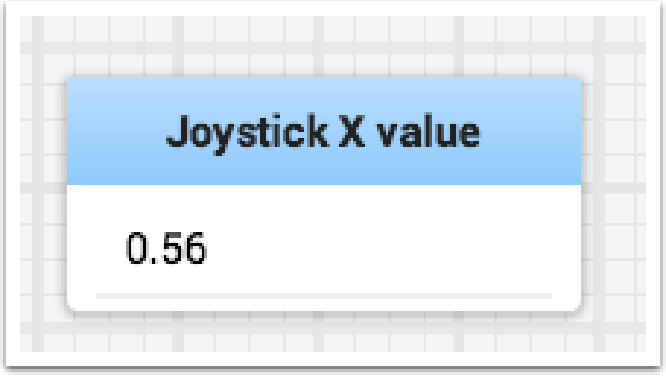
C++

```
frc::SmartDashboard::PutNumber("Joystick X value", joystick1.getX());
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putNumber("Joystick X value", joystick1.getX())
```

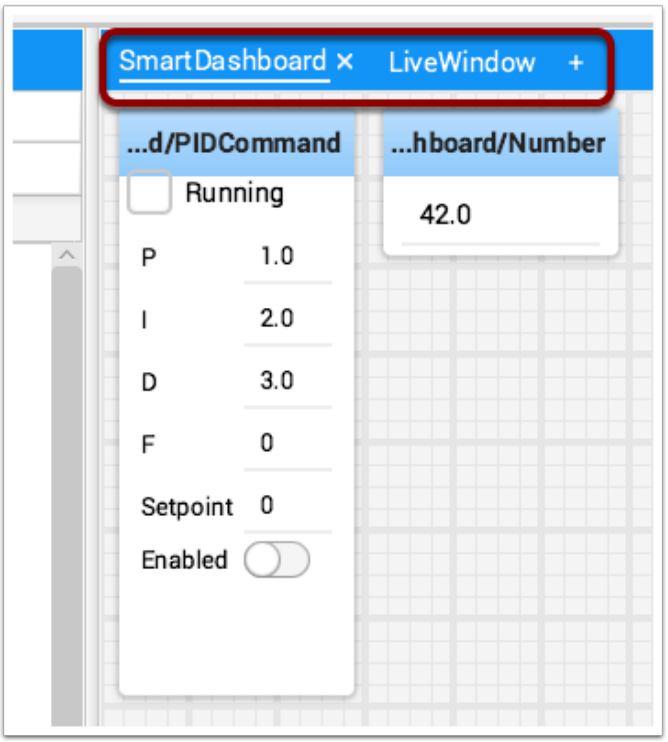
para ver un campo desplegado con la etiqueta " Joystick X value" y un valor del valor X del joystick. Cada vez que esta línea de código es ejecutada, un nuevo valor del Joystick será enviado a la Shuffleboard. Recuerde: debe escribir el valor del joystick cuando quiera ver un valor actualizado. Al ejecutar esta línea una vez al inicio del programa solo mostrará el valor una vez al momento en que la línea de código fue ejecutada.



Mostrando los datos de su robot

El robot puede mostrar datos en los modos de funcionamiento regulares como el Teleop y los modos autónomos, pero también puede mostrar el estado y operar todos los subsistemas del robot cuando éste se cambia al modo de prueba. De forma predeterminada, al iniciar Shuffleboard, verá dos pestañas, una para Teleop/Autónomo y otra para el modo Test. La pestaña actualmente seleccionada está subrayada, como se puede ver en la imagen siguiente.

A menudo la depuración o la supervisión del estado de un robot implica escribir una serie de valores en la consola y verlos pasar. Con Shuffleboard puedes poner valores a un GUI que se



construye automáticamente basado en tu programa. A medida que los valores se actualizan, el elemento correspondiente de la GUI cambia de valor - no hay necesidad de tratar de captar los números que pasan por la pantalla.

Visualización de los valores en el modo de funcionamiento normal (autónomo o teleoperado)

JAVA

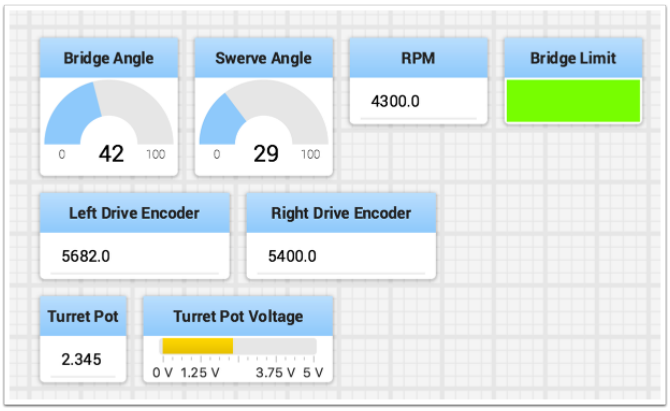
```
protected void execute() {
   SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge());
   SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition());
   SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle());
   SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder());
   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())
```

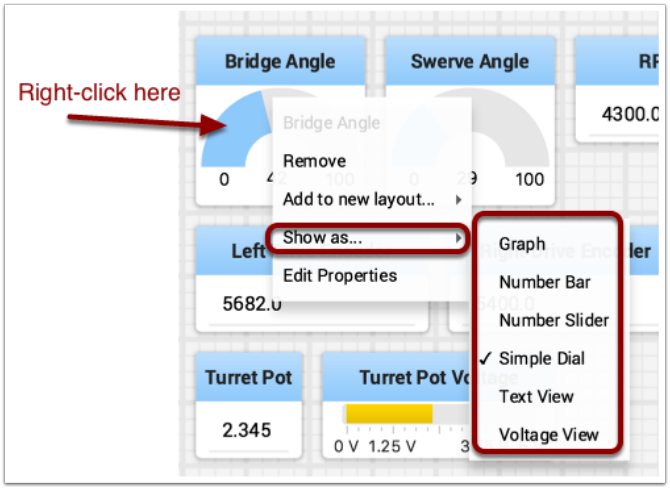


Se pueden escribir valores booleanos, numéricos o de cadena en el Shuffleboard simplemente llamando al método correcto para el tipo e incluyendo el nombre y el valor de los datos, no se requiere ningún código adicional.

- Tipos numéricos como char, int, long, float o doble llama SmartDashboard.putNumber(«nombre de Dashboard», valor).
- Tipos de cadenas llama a SmartDashboard.putString(«nombre de Dashboard», valor)
- Tipos booleanos llama a SmartDashboard.putBoolean(«nombre de Dashboard», valor)

Cambiar el tipo de datos de la pantalla

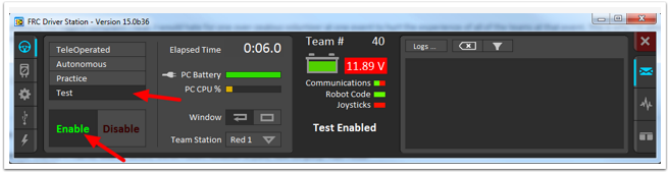
Dependiendo del tipo de datos de los valores que se envían a la Shuffleboard, a menudo se puede cambiar el formato de visualización. En el ejemplo anterior se puede ver que los valores numéricos se mostraron como números decimales, una manera para representar mejor los ángulos y como una vista de voltaje para el potenciómetro de la torreta. Para establecer el tipo de visualización, haga clic con el botón derecho del ratón en la ficha y seleccione «Mostrar como...». Puede elegir los tipos de visualización de la lista del menú emergente.



Visualización de datos en el modo de prueba

Puede añadir código a su programa para mostrar los valores de sus sensores y actuadores mientras el robot está en modo de prueba. Esto puede seleccionarse desde la estación del conductor siempre que el robot no esté en el campo. El código para mostrar estos valores es generado automáticamente por RobotBuilder o añadido manualmente a su programa y se describe en el siguiente artículo. El modo de prueba está diseñado para verificar el correcto funcionamiento de los sensores y actuadores de un robot. Además, puede utilizarse para obtener puntos de ajuste de los sensores, como los potenciómetros, y para ajustar los bucles PID en su código.

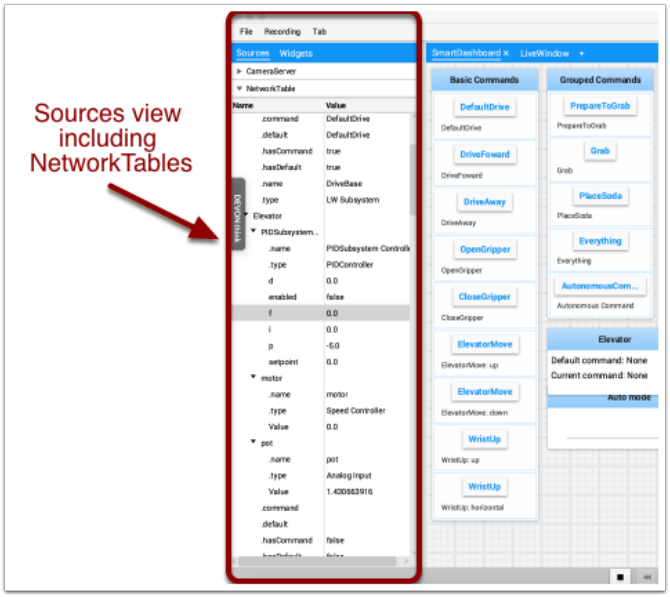
Ajustar el modo de prueba



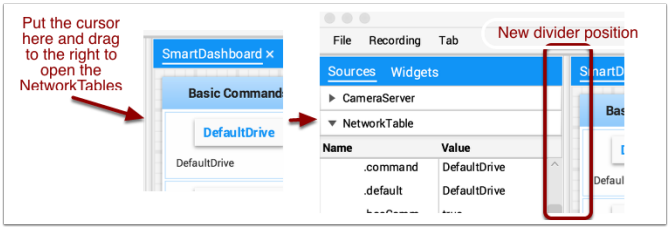
Habilite el modo de prueba en la estación del conductor haciendo clic en el botón «Test» y configurando «Habilitar» en el robot. Al hacer esto, Shuffleboard mostrará el estado de los actuadores y sensores utilizados por su programa organizado por subsistema.

Obtención de datos de la vista de Fuentes

Normalmente los datos de *NetworkTables* aparecen automáticamente en una de las pestañas y usted sólo tiene que reorganizar y utilizar esos datos. A veces puede querer recuperar un valor que se ha borrado accidentalmente de la pestaña o mostrar un valor que no forma parte de la clave SmartDashboard / NetworkTables. Para estos casos los valores pueden ser arrastrados a un panel de la vista de NetworkTables bajo Fuentes en el lado izquierdo de la ventana. Elija el valor que desea mostrar y simplemente arrástrelo al panel y se creará automáticamente con el tipo de widget por defecto para el tipo de datos.



Nota: A veces la vista de las Fuentes no es visible a la izquierda - es posible arrastrar el divisor entre los paneles con pestañas y las Fuentes para que las fuentes no sean visibles. Si esto sucede, mueva el cursor sobre el borde izquierdo y busque un cursor de cambio de tamaño del divisor, luego haga clic con el botón izquierdo y arrastre la vista. En las dos imágenes siguientes puede ver dónde hacer clic y arrastrar, y cuando termine el divisor es como se muestra en la segunda imagen.

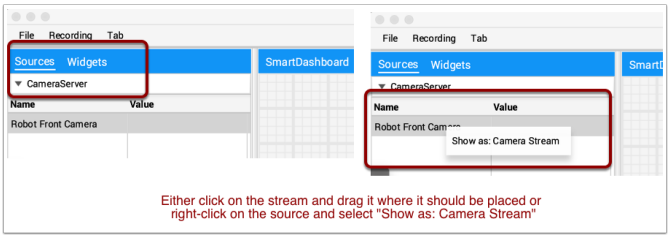


Visualización de la transmisión de la cámara

La transmisión de la cámara del robot puede verse en una pestaña en Shuffleboard. Esto es útil para ver lo que el robot está viendo para dar una vista menos obstruida a los operadores o ayudar a visualizar la salida de un algoritmo de visión que se ejecuta en la computadora de la Driver Station o un coprocesador en el robot. Cualquier transmisión que se esté ejecutando usando la API de CameraServer puede ser visto en un widget de transmisión de la cámara.

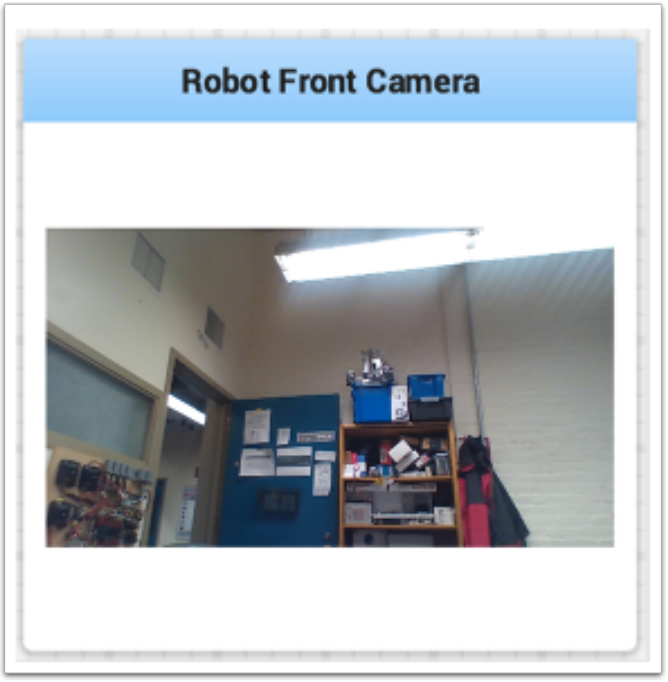
Añadir una transmisión de cámara

Para añadir una cámara al Dashboard, seleccione «Sources» y vea «CameraServer» en el panel lateral izquierdo de la ventana del tablero de mandos, como se muestra en el ejemplo siguiente. Se mostrará una lista de transmisión de cámaras, en este caso sólo hay una cámara llamada «Robot Front Camera». Arrástrala a la pestaña donde debe mostrarse. Alternativamente, la secuencia también puede ser colocada en el tablero de mandos haciendo clic con el botón derecho del ratón sobre la secuencia en la lista de Sources y seleccionando «Show as: Camera Stream».



Una vez que se agregue la transmisión de la cámara, se mostrará en la ventana. Puede ser redimensionado y movido a donde quieras.

Nota: Tengan en cuenta que enviar demasiados datos desde una resolución o velocidad de fotogramas demasiado alta causará un alto uso de la CPU tanto en el roboRIO como en la laptop.

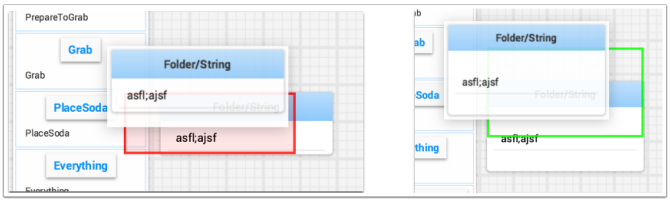


Trabajar con Widgets

Las unidades de visualización que usted manipula en la pantalla de la Shuffleboard son llamados widgets. Los Widgets son general y automáticamente desplegados de valores que el programa del robot publica con NetworkTables.

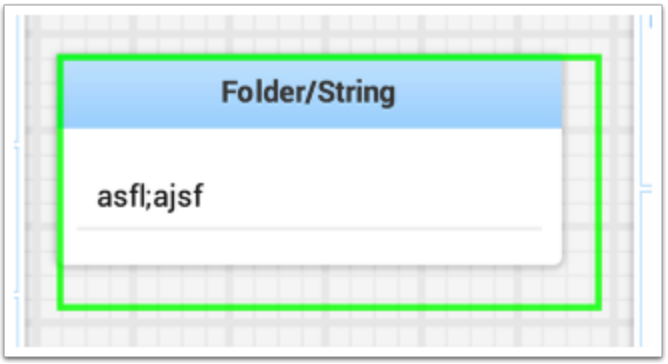
Mover widgets

Los widgets pueden moverse simplemente al arrastrar y soltar. Solamente mueva el cursor sobre el widget, dé clic izquierdo y arrástrelo a su nueva posición. Cuando lo esté arrastrando solamente lo puede acomodar en recuadrados dentro de la cuadrícula y el tamaño de la cuadricula tendrá efecto en la resolución de tu visualización. Al arrastrarlos aparecerá una margen rojo o verde. El color verde generalmente significa que existe espacio suficiente en esa ubicación para soltar el widget y el rojo significa que el widget es muy grande para ese espacio. En el ejemplo de abajo, un widget está siendo movido a una ubicación donde no cabe.



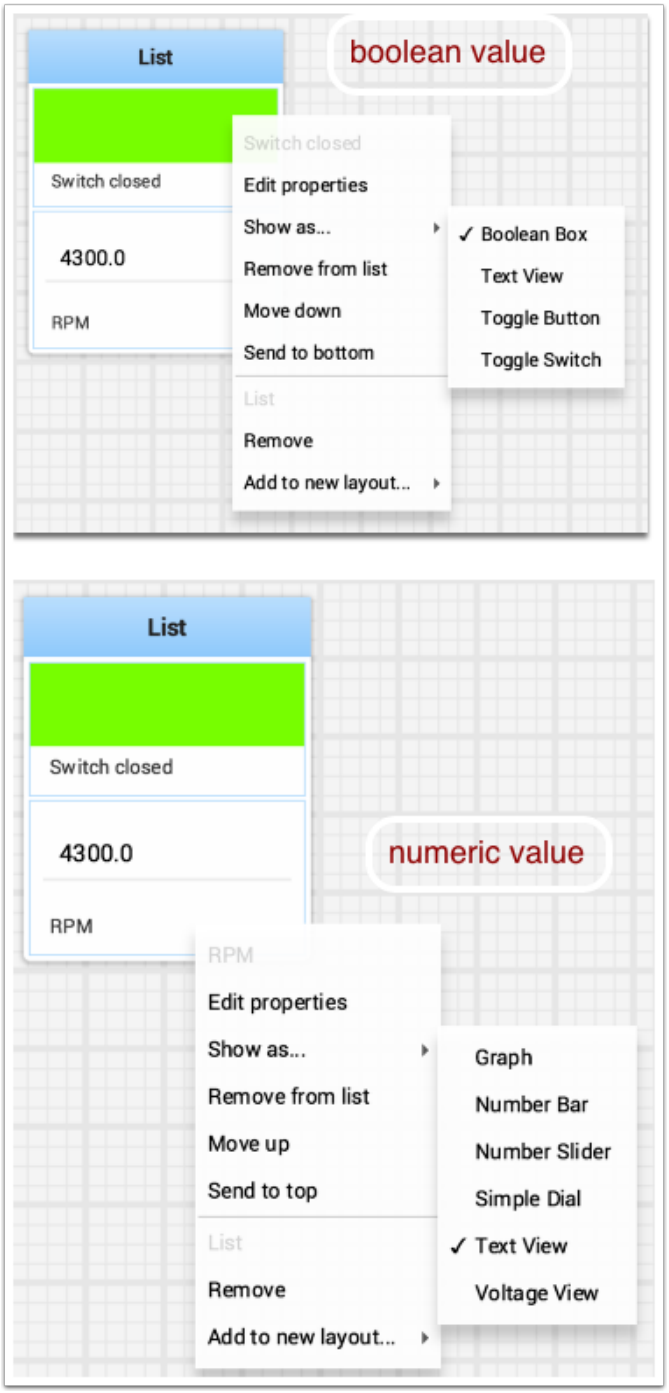
Cambiar el tamaño de los widgets

Los widgets pueden cambiar su tamaño al dar clic y arrastrar la esquina o borde de este. El cursor cambiará a cursor-cambia-tamaños cuando esté en la posición correcta para cambiar el tamaño del widget. Al igual que al mover el widget, un borde verde o rojo aparecerá indicando si el widget puede cambiar su tamaño o no. El ejemplo de abajo muestra un widget siendo cambiado de tamaño a un área más grande con el borde verde indicando que no se sobrepone con los widgets de alrededor.



Cambiar el tipo de visualización de los widgets

La Shuffleboard tiene varios tipos de visualización dependiendo de los datos publicados del robot. Esta eligirá el tipo de visualización predeterminada, pero puede que usted quiera cambiarlo dependiendo de la aplicación. Para ver qué tipos de diseños están disponibles para cada widget, dé clic derecho en el widget y seleccione “Show as...”, y del menú emergenete seleccione el tipo que usted desee. En el ejemplo debajo hay dos tipos de valores de datos, uno numérico y otro booleano. Usted puede ver las diferentes opciones de tipos de visualización que están disponibles para cada uno. El valor booleano solo tiene dos posibles valores (verdadero / falso), puede mostrarse como un recuadro booleano (de color verde/ rojo), o texto, un botón o switch de activación. El valor numérico puede mostrarse como una gráfica, una tabla de números, una barra de números, dial, texto o una vista de voltaje dependiendo de su contexto.

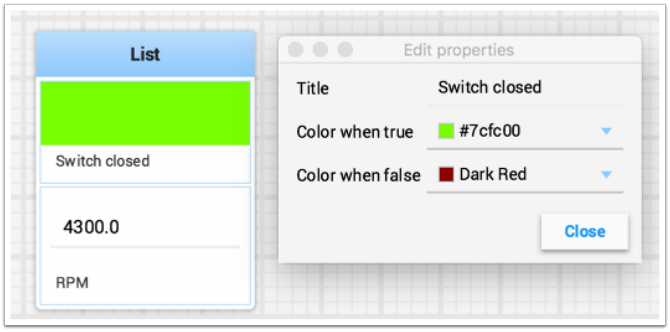

List
boolean value
Switch closed
4300.0
RPM
Switch closed
Edit properties
Show as...
Remove from list
Move down
Send to bottom
List
Remove
Add to new layout...
Boolean Box
Text View
Toggle Button
Toggle Switch
List
Switch closed
4300.0
RPM
numeric value
RPM
Edit properties
Show as...
Remove from list
Move up
Send to top
List
Remove
Add to new layout...
Graph
Number Bar
Number Slider
Simple Dial
Text View
Voltage View

Cambiar el nombre de los widgets

Usted puede cambiar el nombre del widget dando doble clic en la barra de título y editandolo a un nuevo valor. Si el widget está contenido en un diseño, entonces dé clic derecho en el widget y seleccione properties. Desde ahí usted puede cambiar el título del widget que se muestra.

Cambiar las propiedades de los widgets

Usted puede cambiar la apariencia de un widget, como el rango de los valores representados, el color u otro elemento visual. En casos donde esto es posible dé clic derecho en el widget y seleccione “Edit properties” del menú. En este valor tipo boolean mostrado abajo, el nombre del widget, el color de verdadero y el color de falso pueden ser editados.



Trabajando con Listas

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:

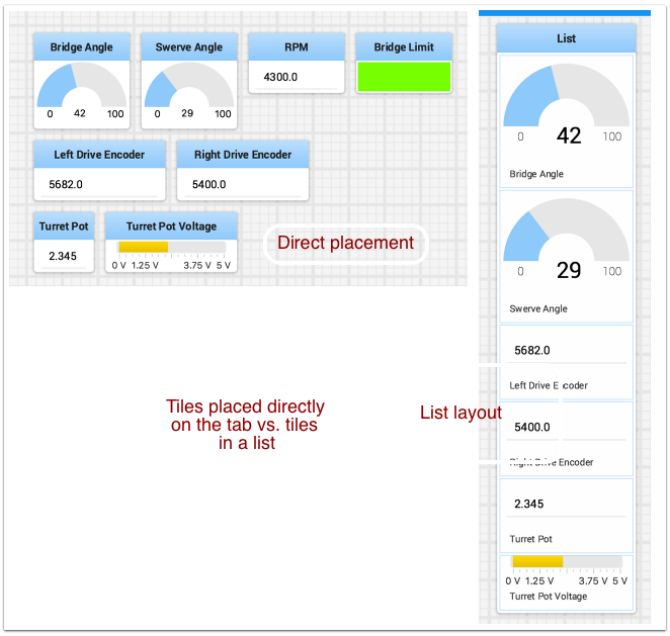
- Tiles in lists don’t have individual header labels; they instead have smaller labels within their list entries.
- Individual tiles placed together create gaps between one another; lists have smaller gaps between tiles.

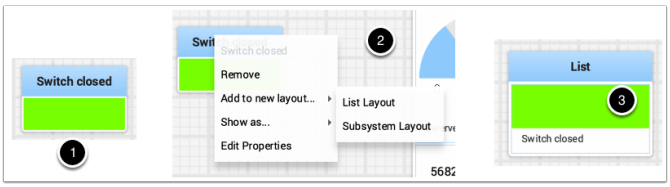
Creando una lista

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.





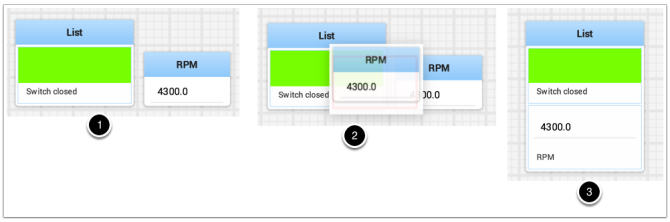
Adding tiles to/removing tiles from a list

A tile can be **added** to an existing list as follows:

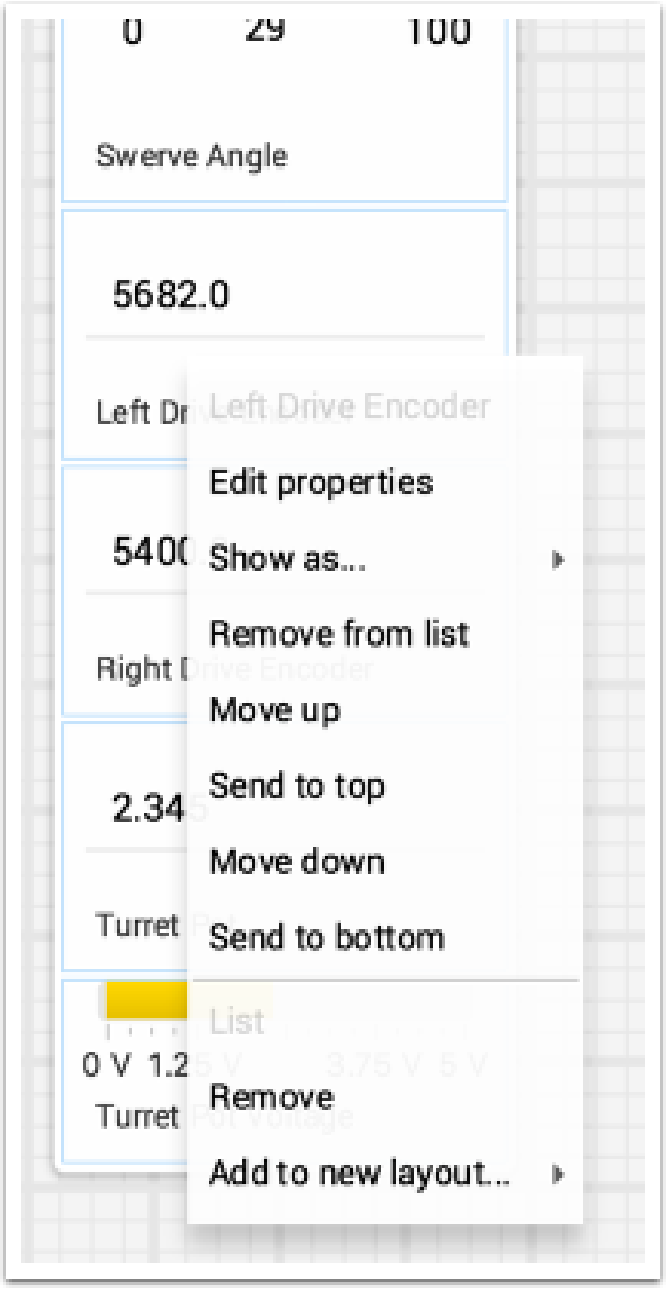
1. Identify the list and the tile to be added.
2. Drag the new tile onto the list.
3. The tile will be added to the list. If the current list size is too small to show it, the tile will be added to the list off-screen and a vertical scrollbar will be added if not already present.

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.



Reorganizar tiles en una lista



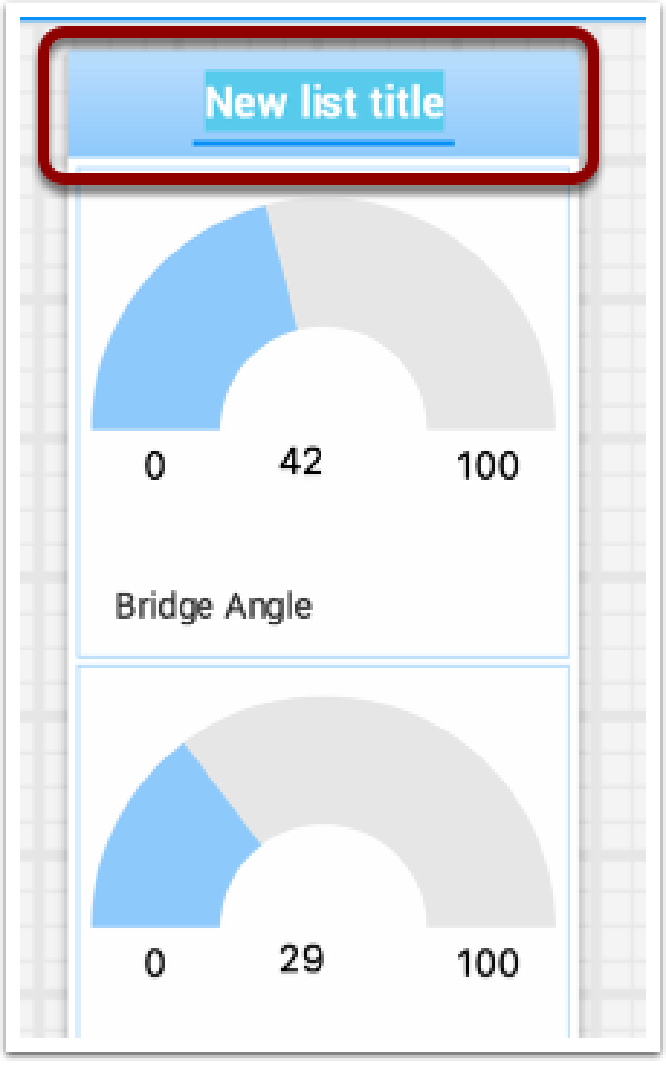
Tiles in a list can be rearranged by right-clicking on the tile and selecting:

- *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*.

cambiando el nombre de una lista

You can rename a list by double-clicking on the list label and changing the name. Click outside the label to save changes.

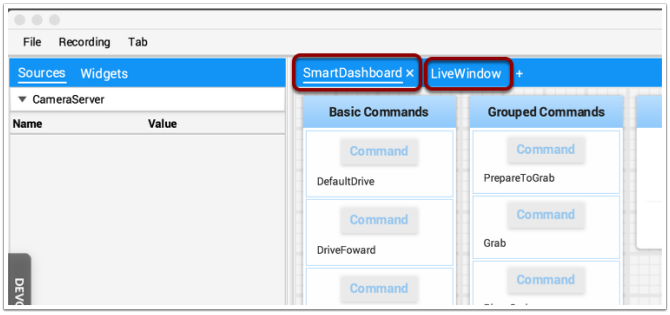


Crear y manipular pestañas

El diseño de pestañas que la Shuffleboard usa ayuda a separar diferentes «vistas» de los datos de su robot y hacer las visualizaciones más útiles. Puede que usted tenga una pestaña que ayude a depurar el programa del robot y una pestaña diferente para usar en competencias. Existen numerosas opciones que hacen a las pestañas muy útiles. Usted puede controlar que datos de la NetworkTable u otras fuentes aparecen en cada una de tus pestañas usando la opción de auto-populate descrita más adelante en este artículo.

Pestañas default

Cuando usted abre la Shuffleboard por primera vez hay dos pestañas, etiquetadas como SmartDashboard y LiveWindow. Estas corresponden a las dos vistas que tiene la SmartDashboard dependiendo en si tu robot está en modo Autónomo/Teleop o Test mode. En la Shuffleboard ambas vistas son disponibles cuando sea.



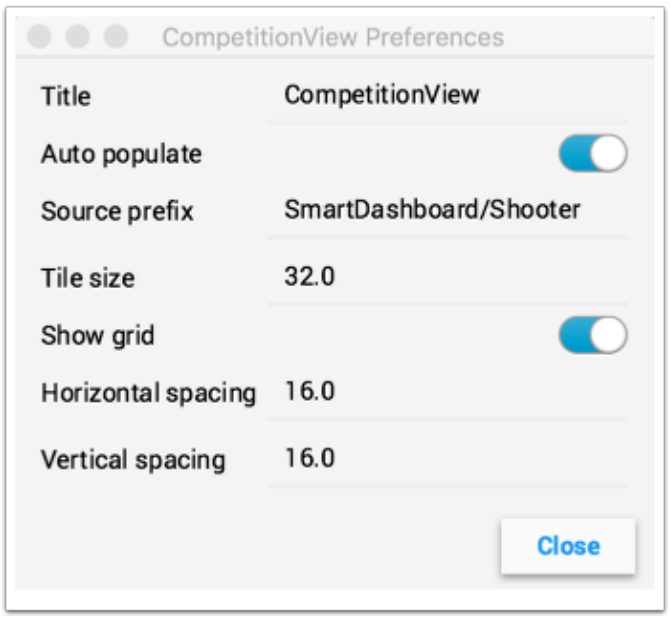
En la pestaña SmartDashboard todos los valores que son escritos usan el conjunto de métodos SmartDashboard.putType(). En la pestaña LiveWindow todos los valores autogenerados de la depuración son mostrados.

Cambiar entre pestañas

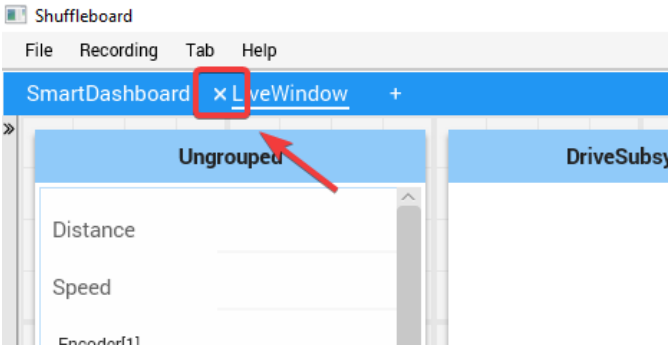
Usted puede cambiar entre pestañas al dar click en la pestaña en la parte superior de la ventana. En el caso de arriba, simplemente dé click en SmartDashboard o LiveWindow para ver los valores asociados con cada pestaña.

Adding and Hiding Tabs

Usted puede agregar pestañas adicionales al dar click en símbolo de más (+) justo al lado de la última pestaña. Una vez que usted ha creado una nueva pestaña puede establecer la etiqueta al dar doble click en la etiqueta de la pestaña y editarla. También puede dar click derecho en la pestaña o usar el Tab menú/menú de pestaña para sacar las preferencias de la pestaña y desde esa ventana usted puede cambiar el nombre al editar el campo de Title/título.



You can hide tabs by clicking the minus(-) symbol to the left of the selected tab name. Since tabs are generated based on the relevant *NetworkTable*, it is not possible to permanently delete them without deleting the table.



Configurar la pestaña para llenar automáticamente

Una de las funciones más útiles en las pestañas es hacer que llenen automáticamente nuevos valores basados en un prefijo de fuente que es proporcionado en el Preferences pane de la pestaña. En el ejemplo de abajo el Preferences pane tiene un prefijo de fuente de "SmartDashboard/Shooter" y Autopopulate está habilitado. Cualquier valor que es escrito usando la clase SmartDashboard que especifica una subllave de Shooter automáticamente aparecerá en la pestaña. Nota: Las llaves que queden con más de un prefijo de Fuente aparecerán en ambas pestañas. Porque esas llaves también inician con SmartDashboard/ y esa es el prefijo de fuente para la pestaña SmartDashboard predeterminada, esos widgets aparecerán en ambas ventanas. Para solo tener valores que aparezcan en una ventana usted puede utilizar NetworkTables para escribir etiquetas y valores y usar un diferente camino que no esté bajo la SmartDashboard. De otra manera usted puede dejar que todo aparezca en la misma SmartDashboard haciéndolo muy desordenado, pero teniendo pestañas específicas para sus necesidades que serán filtradas.

Utilizar la cuadrícula de pestañas y el espaciado

Cada pestaña tiene su propio tamaño rectangular (número de pixeles por largo al cuadrado). Entonces algunas pestañas puede que tengan una resolución más pesada para una vista más fácil y otras pueden tener una cuadrícula fina. El tamaño de los rectángulos en Tab preferences anulan cualquier configuración global en las preferencias de la Shuffleboard. Además, usted puede especificar el vacío entre el dibujo en el widget y el borde del widget. Si su programa usa interfaces estos parámetros son referidos usualmente como espacio horizontal y vertical (hgap, vgap).

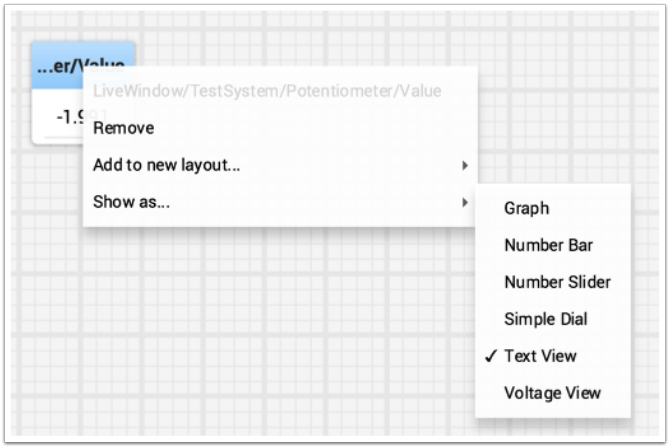
Mover widgets entre pestañas

Actualmente no existe una manera sencilla para mover widgets entre pestañas sin tener que borrarlo de una pestaña y arrástralo al área de la jerarquía de las fuentes en la izquierda dentro de una nueva ventana. Esperamos tener esta capacidad en una actualización futura.

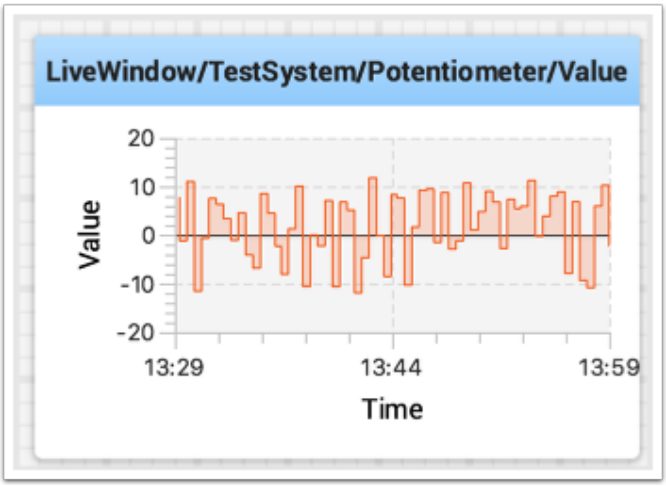
Trabajando con gráficos

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.

Para crear un gráfico, elija un valor numérico y haga clic con el botón derecho del ratón en el encabezado y seleccione «Mostrar como…» y luego elija el gráfico

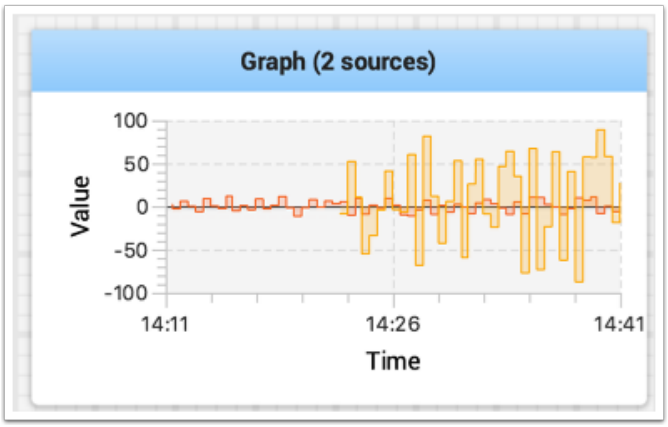


El widget de gráficos muestra gráficos lineales del valor que has seleccionado. Establecerá automáticamente la escala y el intervalo de tiempo por defecto que mostrará el gráfico será de 30 segundos. Puedes cambiar esto en la configuración del gráfico (ver más abajo).

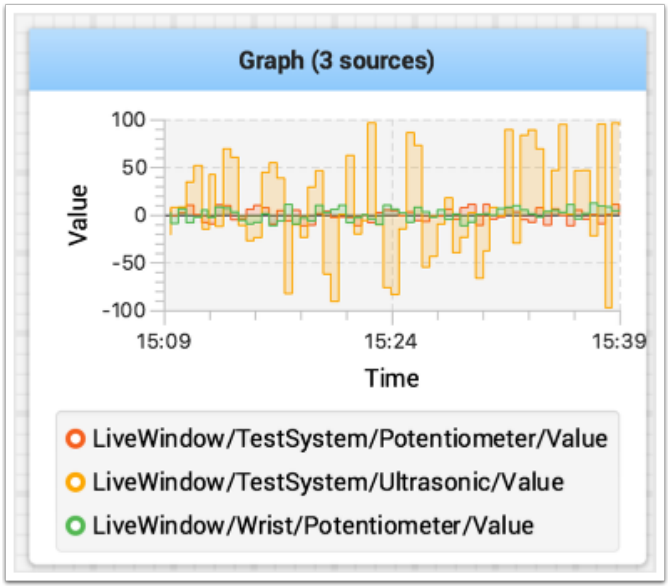


Adición de valores de datos adicionales

Para los valores relacionados, a menudo es deseable mostrar múltiples valores en el mismo gráfico. Para ello, simplemente arrastre valores adicionales desde la vista de origen de NetworkTables (lado izquierdo de la ventana de Shuffleboard) y suéltelos en el gráfico y ese valor se añadirá como se muestra a continuación. Puede continuar arrastrando valores adicionales al gráfico.



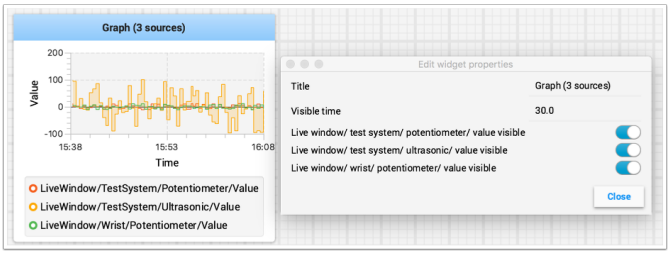
Puede cambiar el tamaño del gráfico verticalmente para ver la leyenda si no se muestra como se muestra en la imagen de abajo. La leyenda muestra todas las fuentes que se utilizan en el gráfico.



Configuración de las propiedades de los gráficos

Puedes establecer el número de segundos que se muestran en el gráfico cambiando el «Tiempo visible» en las propiedades del widget del gráfico. Para acceder a las propiedades, haz clic con el botón derecho del ratón en el gráfico y selecciona «Editar propiedades».

Además de establecer el tiempo visible, el gráfico puede encender y apagar selectivamente las fuentes al encender y apagar el interruptor para cada una de las fuentes que se muestran en la ventana de propiedades (véase más abajo).

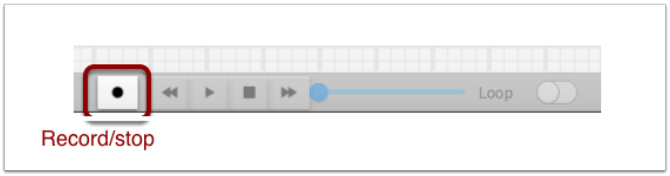


Grabaciones y Playback

La Shuffleboard puede cargar todas las actualizaciones de los widgets durante una sesión. Después el archivo cargar puede repetirse para ver que sucedió durante un partido o una práctica. Esto es especialmente útil si algo no funciona como se desea durante un partido y usted quiere ver que sucedió. Cada grabación es guardada en un archivo de grabación.

Crear una Grabación

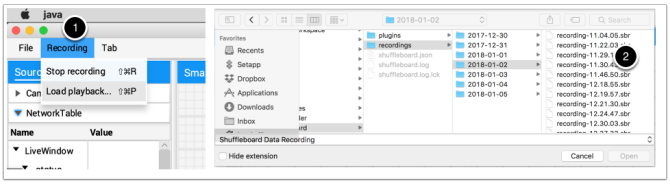
Cuando Shuffleboard comienza a grabar a todos los valores de NetworkTables se registran y continúa hasta que se detiene pulsando el botón de grabación / parada en los controles de la grabadora como se muestra a continuación. Si se desea realizar una nueva grabación, como cuando se está probando una nueva pieza de código o sistema mecánico, detenga la grabación actual si se está ejecutando, y haga clic en el botón de grabación. Vuelva a hacer clic en el botón para detener la grabación y cerrar el archivo de grabación. Si el botón es redondo (como se muestra), haga clic en él para iniciar una grabación. Si el botón es un cuadrado, entonces se está ejecutando una grabación, así que haga clic en él para detener la grabación.



Reproducir una Grabación

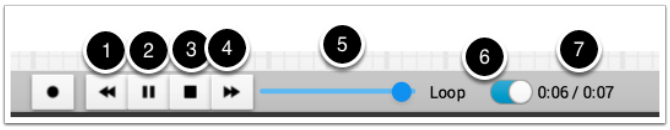
Grabaciones previas pueden volver a reproducirse haciendo:

1. Seleccionar el menú «Recording» después dé click en «Load playback».
2. Elija una grabación del directorio mostrado. Las grabaciones son agrupadas por fecha y los nombres de los archivos son la fecha en la cual se grabó, para ayudar a identificar la correcta. Seleccione la grabación correcta de la lista.



Controlar el Playback

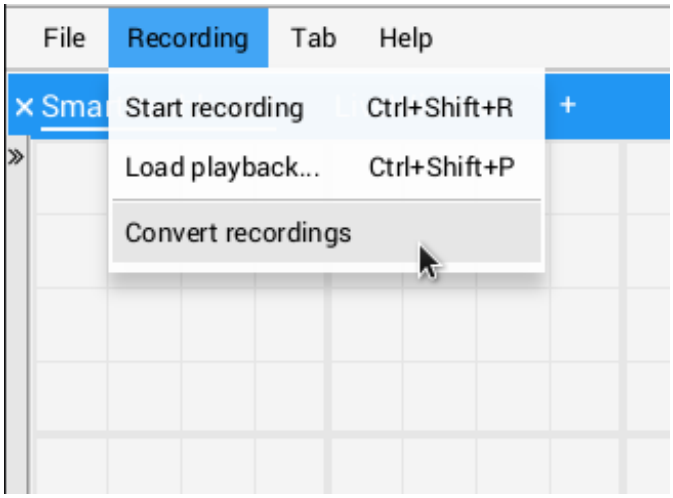
Al seleccionar el archivo empezará la repetición de ese. Mientras el archivo se reproduce los controles de grabación mostrarán el tiempo actual dentro de la grabación también la opción de loop the recording while watching it/repetir la grabación mientras se escucha. Cuando la grabación se está reproduciendo de los controles de “transport” permitirá que el playback sea controlado.



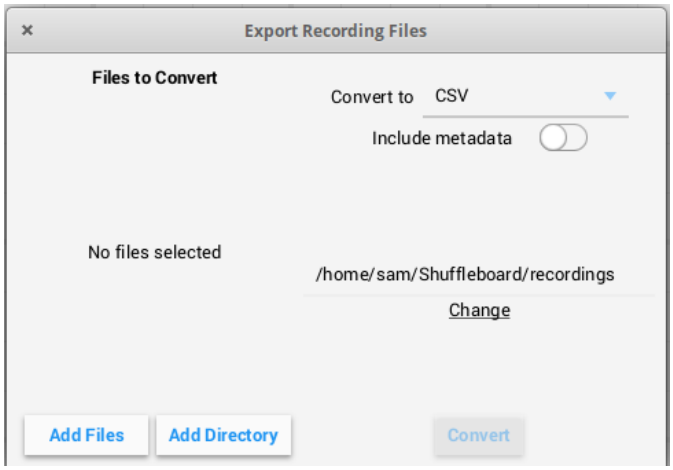
Los controles trabajan de la siguiente manera:

1. El botón izquierdo con una doble flecha respalda el playback al último punto de datos
2. Los controles play/pause empiezan y paran el playback
3. El botón cuadrado de stop detiene el playback y continúa mostrando los valores del robot en ese momento
4. El botón derecho con una doble flecha se salta al siguiente valor de datos que cambia
5. La barra de desplazamiento permite posicionarse directamente en cualquier punto para ver diferentes partes de la grabación
6. El switch de repetición habilita que siga funcionando playback, eso significa, que el playback se repetirá una y otra vez hasta que lo detenga
7. El tiempo muestra el punto actual de la grabación y la duración total de esta

Convertir a Diferentes Tipos de Archivos



Las grabaciones de la Shuffleboard están en un formato predeterminado por eficiencia. Para analizar datos grabados sin tener que repetirlo a través de la app. La Shuffleboard proporciona convertidores de datos para convertir grabaciones a cualquier formato. Solamente un simple convertidor CSV es enviado con la app. Pero los equipos pueden desarrollar convertidores e incluirlos en los plugins de la Shuffleboard.



Varias grabaciones se pueden convertirse a la vez. Los archivos individuales pueden ser seleccionados con el botón “Add files”, o todos los archivos de grabaciones en un directorio pueden seleccionarse al mismo tiempo con el botón “Add directory”.

Las grabaciones convertidas se generarán en el directorio `~/Shuffleboard/recordings`, pero pueden ser seleccionados manualmente con el botón “Change”.

Los diferentes convertidores pueden seleccionarse con la herramienta dropdown en la esquina derecha superior. Por default, solamente el convertidor CSV está disponible. Los convertidores personalizados de plugins aparecerán como opciones en el drop dowm.

Notas Adicionales

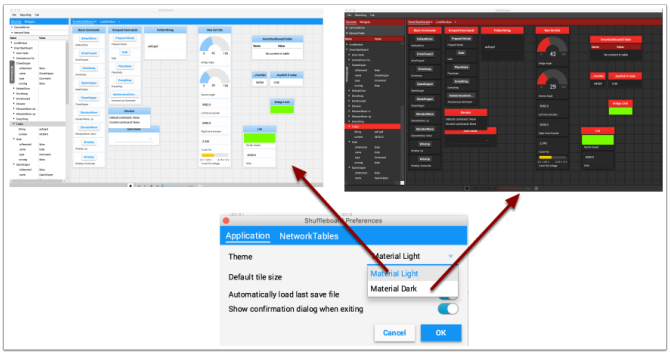
Las gráficas no se mostraran de manera correcta mientras se esté depurando la línea del tiempo pero si se está reproduciendo a través de la gráfica puede capturar su historia por la gráfica entonces se mostrara en la presentación original.

Establecer preferencias globales para el Shuffleboard

Hay una serie de ajustes que establecen la forma en que Shuffleboard se ve y se comporta. Están en el panel de Preferencias de Shuffleboard al que se puede acceder desde el menú archivo.

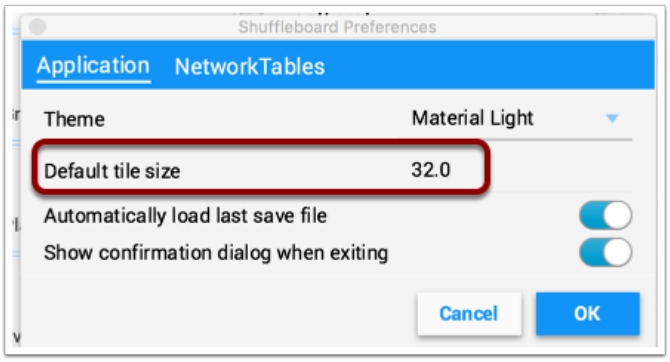
Estableciendo el tema

Los Shuffleboard soportan dos temas, Material Dark y Material Light y el ajuste depende de tus preferencias. Esto utiliza estilos css que se aplican a toda la aplicación y que se pueden cambiar en cualquier momento.



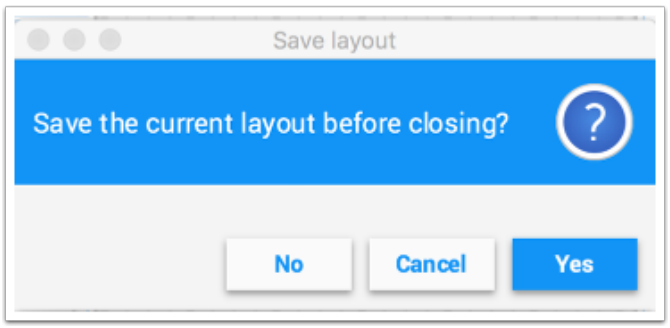
Estableciendo el tamaño de los tiles por defecto

Los Shuffleboards colocan los tiles/teselas en una cuadrícula cuando los agrega o mueve usted mismo o cuando se auto-poblan. Puede establecer el tamaño de los tiles por defecto para cada ficha o puede establecerse globalmente para todas las fichas creadas después de cambiar la configuración por defecto. Una resolución más fina en la cuadrícula da como resultado un control más fino sobre la colocación de los tiles. Esto se puede establecer en la ventana de Preferencias de Shuffleboard como se muestra a continuación.

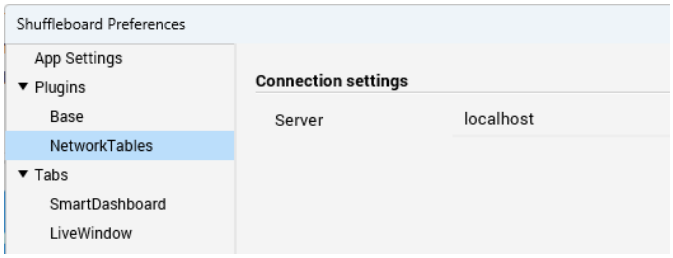


Trabajando con los archivos de guardado del diseño

You can save your layout using the File / Save and File / Save as... menu options. The preferences window has options to cause the previous layout to be automatically applied when Shuffleboard starts. In addition, Shuffleboard will display a «Save layout» window to remind you to save the layout on exit, if the layout has changed. You can choose to turn off the automatic prompt on exit, but be sure to save the layout manually in this case so you don’t loose your changes.



Fijando el número de equipo



Para que Shuffleboard pueda encontrar su servidor de NetworkTables en su robot, especifique su número de equipo en la pestaña «NetworkTables» del panel de Preferencias. Si está ejecutando Shuffleboard con una estación de controladora en funcionamiento, el campo Servidor se rellenará automáticamente con la información correcta. Si está ejecutando en un ordenador sin la Estación del controlador, puede introducir manualmente su número de equipo o la dirección de red del robotRIO.

Preguntas frecuentes, problemas y errores de Shuffleboard

Advertencia: El Shuffleboard así como la mayoría de los otros componentes del sistema de control fueron desarrollados con Java 11 y no funcionarán con Java 8. Asegúrate antes de reportar problemas que tu computadora tiene Java 11 instalado y esté configurado con Java Environment que tiene por defecto.

Preguntas frecuentes

¿Cómo puedo informar de problemas, errores o solicitudes de funciones con 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.

¿Cómo puedo añadir mis propios widgets u otras extensiones a 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.

¿Cómo puedo construir el Shuffleboard desde el código fuente?

Puedes obtener el código fuente descargándolo, clonándolo o bifurcando el depósito en el sitio de GitHub. Para construir y ejecutar Shuffleboard desde el código fuente, asegúrate de que el directorio actual es el código fuente de nivel superior y utiliza uno de estos comandos:

Aplicación	Comando (para los sistemas de Windows ejecute el archivo gradlew.bat)
Correr Shuffleboard	./gradlew :app:run
Construir las API y las clases de utilidad para la creación de plugins	./gradlew :api:shadowJar
Construyendo el archivo completo de la aplicación	./gradlew :app:shadowJar

11.2.2 Shuffleboard - Diseños con código

Usar pestañas

La Shuffleboard es una interfaz con pestañas. Cada pestaña organiza los widgets agrupándolos de manera lógica. De manera predeterminada la Shuffleboard tiene pestañas por herencia de SmartDashboard y LiveWindows, pero nuevas pestañas pueden crearse directamente en la Shuffleboard desde el programa del robot para una mejor organización.

Crear una nueva pestaña

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")
```

Crear una nueva pestaña es tan fácil como llamar un solo método en la clase Shuffleboard, lo cual creará una nueva pestaña en la Shuffleboard y devolverá una herramienta para añadir sus datos a la pestaña. Llamar al método getTab varias veces con el mismo título de pestaña devolverá la misma herramienta cada vez.

Seleccionar una pestaña

JAVA

```
Shuffleboard.selectTab("Tab Title");
```

C++

```
Shuffleboard::SelectTab("Tab Title");
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

Shuffleboard.selectTab("Tab Title")
```

Este método permite seleccionar pestañas por su título. Este método distingue entre mayúsculas y minúsculas (entonces “Tab Title” y “Tab title” son dos diferentes pestañas), y solamente funciona si una pestaña con ese título existe cuando el método es invocado, entonces al invocar `selectTab("Example")`solamente funcionará si una pestaña llamada “Example” ha sido definida previamente.

Este método puede usarse para seleccionar cualquier pestaña en la Shuffleboard, no solamente las que fueron creadas por el programa del robot.

Caveats

Las pestañas creadas por el robot difieren en algunas maneras importantes de las pestañas creadas desde la dashboard:

- Sin guardar en el archivo de guardar de la Shuffleboard.
- No contiene la función de auto completar.
- Los usuarios deben especificar el contenido de las pestañas en su programa.
- Tener un color especial para diferenciarlas de pestañas normales.

Enviando datos

A diferencia de SmartDashboard, los datos no se pueden enviar directamente a Shuffleboard sin antes especificar en qué pestaña se deben colocar los datos.

Envío de datos simples

El envío de datos simples (números, cadenas, valores booleanos y matrices de estos) se realiza llamando `add` en una pestaña. Este método establecerá el valor si aún no está presente, pero no sobrescribirá un valor existente.

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 los datos deben actualizarse (por ejemplo, la salida de algún cálculo realizado en el robot), llame a `getEntry()` después de definir el valor, luego actualice cuando sea necesario o en una función `periódica`

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)
```

Hacer que las elecciones persistan entre reinicios

Al configurar un robot desde el tablero, es posible que algunos ajustes persistan entre los reinicios del robot o de la estación del conductor en lugar de que los conductores recuerden (u olviden) configurar los ajustes antes de cada partido.

Simplemente usando *addPersistent* en lugar de` add` hará que el valor se guarde en el roboRIO y se cargue cuando se inicie el programa del robot.

Nota: Esto no se aplica a los datos que se pueden enviar como selectores o controladores de motor.

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))
```

Envío de sensores, motores, etc.

De manera análoga a `SmartDashboard.putData`, cualquier objeto `Sendable` (la mayoría de los sensores, controladores de motor y SendableChoosers) se pueden agregar a cualquier pestaña

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))
```

Recuperar datos

A diferencia de la función `SmartDashboard.getNumber` y parecidos, recuperar datos de la Shuffleboard también se puede hacer a través de las NetworkTableEntries, de las cuales hablamos en el artículo anterior.

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 = ...

   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)
```

Este ejemplo básico tiene una falla muy evidente: La máxima velocidad puede configurarse en la dashboard a un valor fuera [0, 1] - lo cual puede causar que las entradas se saturen (siempre a la máxima velocidad), ¡o incluso al revés! Afortunadamente, existe una manera para evitar este problema - cubierto en el siguiente artículo.

Configurar widgets

Los programas de robot pueden especificar exactamente qué widget usar para mostrar un punto de datos, así como debe configurarse ese widget. Como hay demasiados widgets para enumerarlos aquí, consulte los documentos para obtener más detalles.

Especificando 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();
```

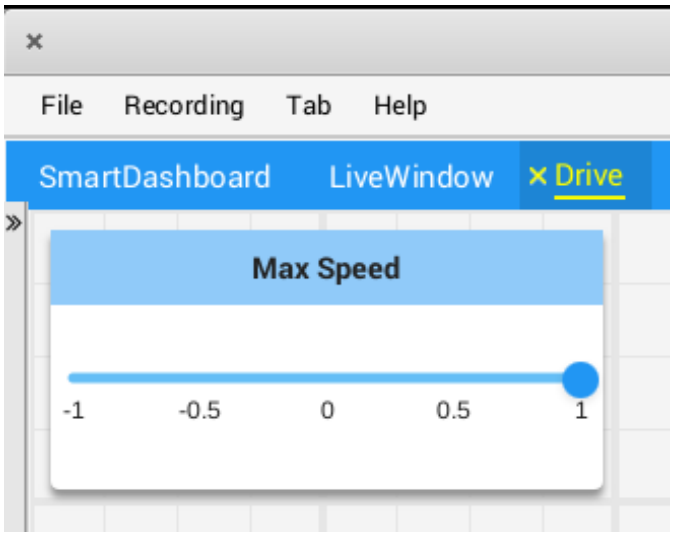
C++

```
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())
```

En este ejemplo, configuramos el widget «Velocidad máxima» para usar un control deslizante para modificar los valores en lugar de un campo de texto básico.



Configurar las propiedades del widget

Dado que la velocidad máxima solo tiene sentido ser un valor de 0 a 1 (punto a punto a velocidad máxima), un control deslizante de -1 a 1 puede causar problemas si el valor cae por debajo de cero. Afortunadamente, podemos modificar eso usando el método «withProperties “”

JAVA

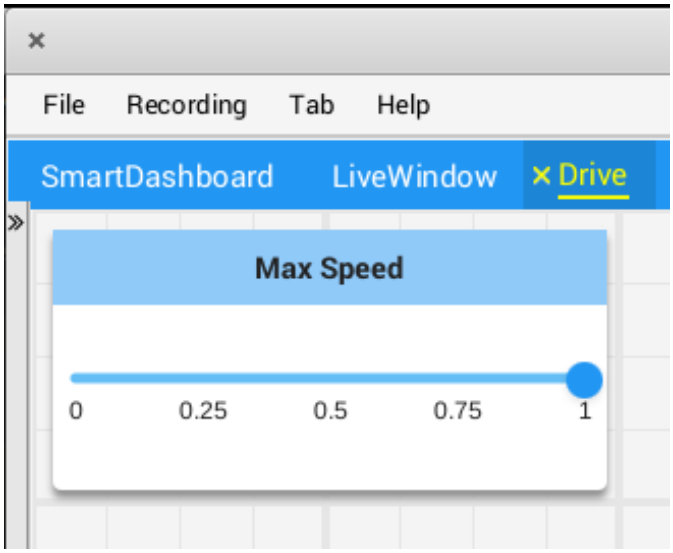
```
Shuffleboard.getTab("Drive")
   .add("Max Speed", 1)
   .withWidget(BuiltInWidgets.kNumberSlider)
   .withProperties(Map.of("min", 0, "max", 1)) // specify widget properties here
   .getEntry();
```

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())
```



Notas

Los widgets se pueden especificar por nombre; sin embargo, los nombres distinguen entre mayúsculas y minúsculas y espacios en blanco («Number Slider» es diferente de «Number Slider» y «NumberSlider»). Por esta razón, se recomienda utilizar la clase de widgets incorporada para especificar el widget en lugar de por nombre sin formato. Sin embargo, un widget personalizado solo se puede especificar por nombre o creando un «WidgetType “” personalizado para ese widget.

Los nombres de las propiedades de los widgets no distinguen entre mayúsculas y minúsculas ni espacios en blanco («Max» y «max» son lo mismo). Consulte la documentación sobre el widget en la clase BuiltInWidgets para obtener detalles sobre las propiedades de ese widget.

Organizando Widgets

Ajuste del tamaño y la posición del Widget

Llame «withSize» y «withPosition» para fijar el tamaño y la posición del Widget en la pestaña.

`withSize` establece el número de columnas de ancho y las filas de alto que el Widget debe ser. Por ejemplo, llamar «withSize» (1, 1) hace que el Widget ocupe una sola celda en la red. Tenga en cuenta que algunos Widgets tienen un tamaño mínimo que puede ser mayor que el tamaño especificado, en cuyo caso el Widget usará el tamaño más pequeño soportado.

`withPosition` establece la fila y la columna de la esquina superior izquierda del Widget. Tanto las filas como las columnas están indexadas con el 0, por lo que la fila superior es el número 0 y la columna más a la izquierda también es el número 0. Si se especifica la posición de cualquier Widget en una pestaña, cada Widget también debería tener su posición establecida para evitar que se superpongan.

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
```

Adición de Widgets a los diseños

Si hay muchos widgets en una pestaña con datos relacionados, puede ser útil colocarlos en subgrupos más pequeños en lugar de perderlos en la pestaña. De forma muy parecida a como se recupera el manejo de una pestaña con `Shuffleboard.getTab`, un diseño dentro de una pestaña (o incluso en otro diseño) puede ser recuperado con `ShuffleboardTab.getLayout`.

JAVA

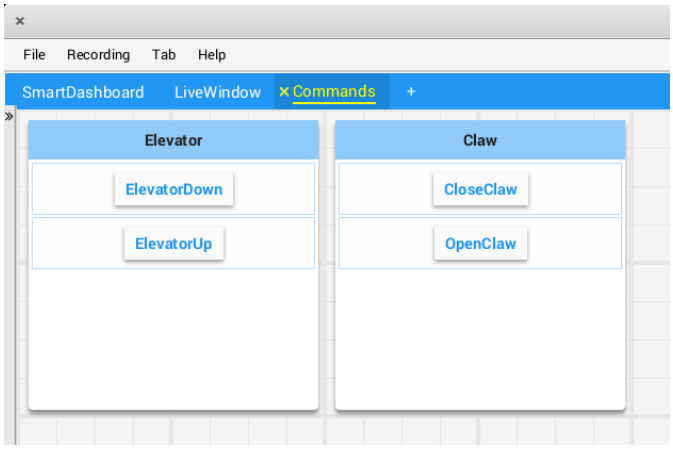
```
ShuffleboardLayout elevatorCommands = Shuffleboard.getTab("Commands")
  .getLayout("Elevator", BuiltInLayouts.kList)
  .withSize(2, 2)
  .withProperties(Map.of("Label position", "HIDDEN")); // hide labels for commands

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 - Uso avanzado

Comandos y Subsistemas

Al utilizar un framework basado en comandos, Shuffleboard hace más sencillo entender qué es lo que el robot hace pues muestra el estado de varios comandos y subsistemas en tiempo real.

Visualización de subsistemas

Para ver el estado de un subsistema mientras el robot está operando, ya sea en modo autónomo o teleoperado, eso es lo que sus comandos predeterminados son y qué comando está utilizando el subsistema en ese momento, envía un `instance` del subsistema a la Shuffleboard:

JAVA

```
SmartDashboard.putData(subsystem-reference);
```

C++

```
SmartDashboard::PutData(subsystem-pointer);
```

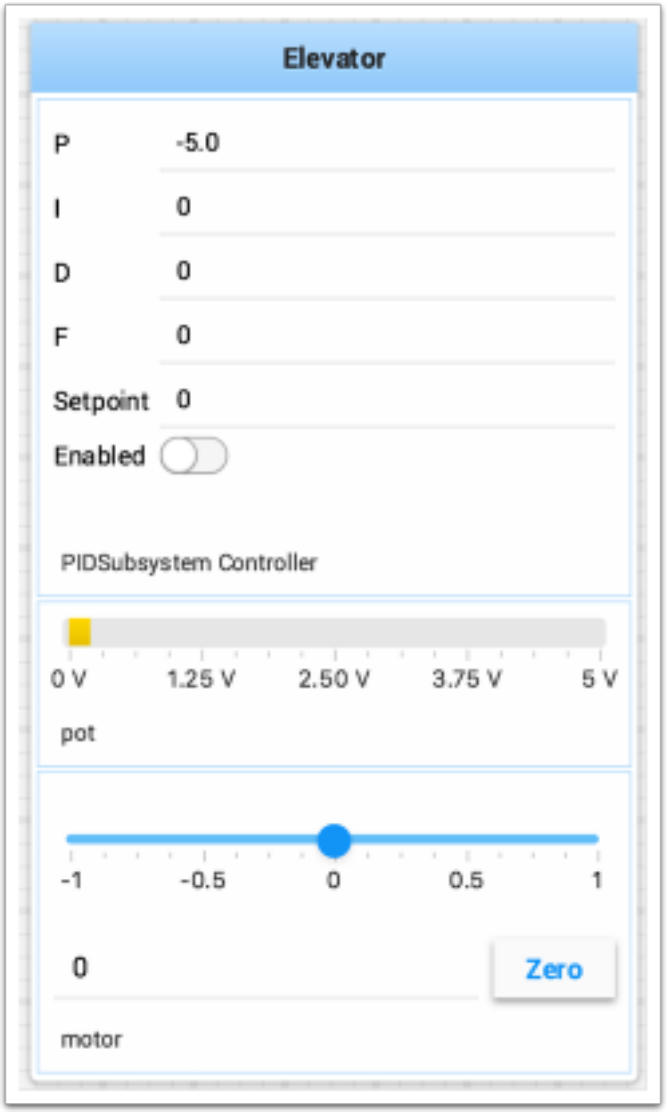
PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData(subsystem-reference)
```

Suffleboard mostrará el nombre del subsistema, el comando predeterminado asociado al subsistema y el comando utilizado en ese momento. En este ejemplo el comando predeterminado del subsistema elevador se llama `AutonomousCommand` y también es el comando que se está utilizando en el subsistema Elevador en ese momento.

Subsistemas en Modo de prueba

En el Modo de prueba (Prueba/Habilitado en la driver station) los subsistemas pueden mostrarse en la pestaña de LiveWindow con los sensores y actuadores del subsistema. Esto es ideal para verificar que los sensores están funcionando correctamente al confirmar que se están devolviendo los valores. Además, los actuadores pueden ser operados. Por ejemplo, los motores pueden operarse utilizando intensificadores para configurar su velocidad y dirección. Para subsistemas PID, los valores P, I, D y F son constantes que se muestran junto a los puntos de control y un control habilitado. Esto puede ser de utilidad para sintonizar los subsistemas PID al ajustar dichas constantes, configurar un punto de control, y habilitar el controlador PID embebido. Así, puede observarse la respuesta del mecanismo. Este ciclo (modificar parámetros, habilitar y observar) puede repetirse hasta que se encuentre un conjunto razonable de parámetros.



Para mayor información sobre cómo sintonizar subsistemas PID *aquí*. Utilizar RobotBuilder automáticamente generará el código necesario para obtener el subsistema mostrado en el Modo de prueba. El código necesario para mostrar los subsistemas se encuentra aquí abajo, donde el nombre del subsistema es una cadena de caracteres.

```
setName(subsystem-name);
```

Visualizar comandos

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));
```

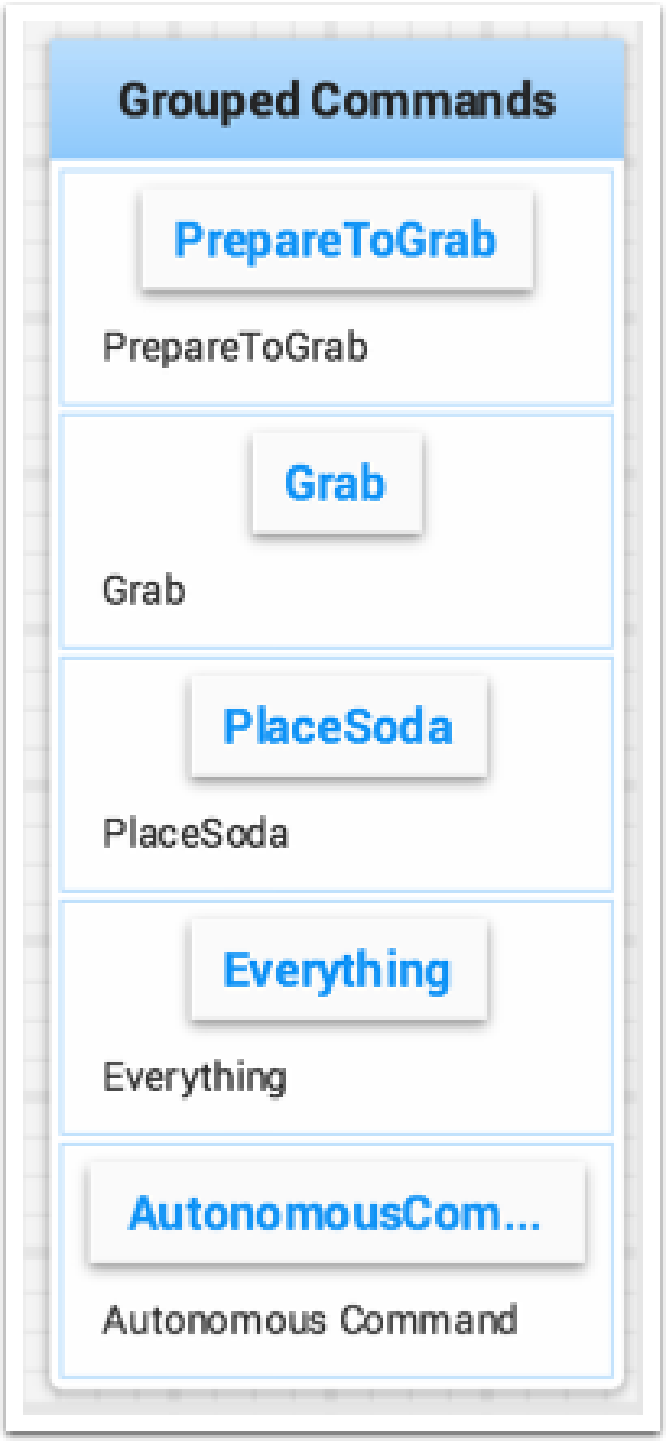
C++

```
SmartDashboard::PutData("ElevatorMove: up", new ElevatorMove(2.7));
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData("ElevatorMove: up", ElevatorMove(2.7))
```

Shuffleboard mostrará el nombre del comando y un botón para ejecutarlo. De esta forma, comandos individuales y en grupo pueden probarse fácilmente sin necesidad de un código de prueba especial en el programa del robot. En la siguiente imagen se encuentran algunos comandos contenidos en una lista de Shuffleboard. Al presionar el botón una vez se corre el comando y al presionarlo de nuevo, se detiene. Para utilizar esta característica el robot debe tener habilitado el modo teleoperado.


Grouped Commands
PrepareToGrab
PrepareToGrab
Grab
Grab
PlaceSoda
PlaceSoda
Everything
Everything
AutonomousCom...
Autonomous Command

Probar y sintonizar ciclos 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:

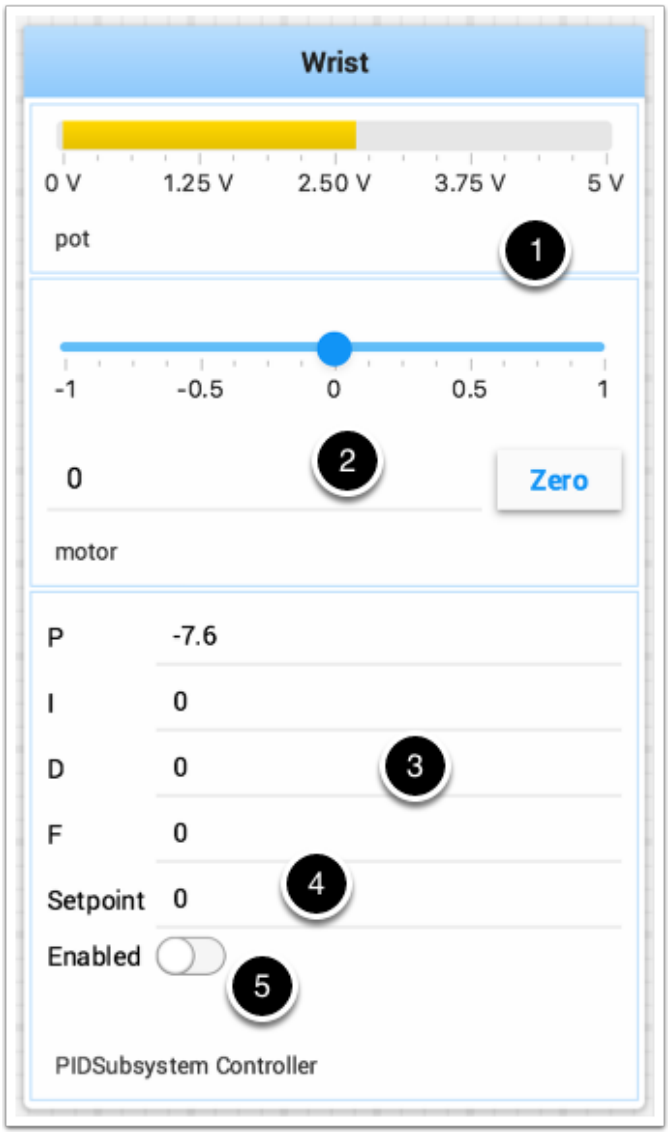
1. Al leer los valores del sensor se determina la distancia entre el robot o el mecanismo y el punto de control deseado. El punto de control es el valor del sensor que corresponde con el objetivo. Por ejemplo, un brazo robótico con una articulación tipo muñeca debería moverse a un ángulo específico rápidamente y detenerse cuando el sensor indica dicho ángulo. Un potenciómetro es un sensor que puede medir el ángulo rotacional. Al conectarlo a una entrada análoga, el programa puede determinar un voltaje que sea directamente proporcional al ángulo.
2. Al calcular un error (la diferencia entre los valores del sensor y los esperados). La señal de error en los valores indica en qué lado del punto de control se encuentra la muñeca. Por ejemplo, valores negativos pueden indicar que el ángulo medido es mayor al deseado. La magnitud del error determina la distancia entre el valor actual y el valor esperado del ángulo de la muñeca. Si el error es cero, esto quiere decir que el valor medido es exactamente igual al valor esperado. El error puede utilizarse como un dato de entrada para que el algoritmo PID calcule la velocidad del motor.
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 cuenta con una clase PIDController que implementa el algoritmo PID y acepta constantes que corresponden a los valores Kp, Ki y Kd. El algoritmo PID tiene tres componentes que contribuyen al cálculo de la velocidad del motor a través del error.

1. P (proporcional) – Es el término que, al multiplicarse por una constante (Kp) generará una velocidad del motor que ayudará a mover al motor en la dirección correcta con la velocidad adecuada.
2. I (integral) - Es el término de la suma de errores sucesivos. Mientras más tiempo exista el error más grande será la contribución integral. Simplemente es una suma de todos los errores a través del tiempo. Si la muñeca no se acerca lo suficiente al punto de control debido a la gran carga que intenta mover, el término integral continuará aumentando (suma de errores) hasta que contribuya lo suficiente para que la velocidad del motor pueda alcanzar el punto de control. La suma de los errores se multiplica por una constante (Ki) para escalar el término integral para el sistema
3. D (diferencial) – Este valor es la tasa de cambio de los errores. Se utiliza para alentar la velocidad del motor cuando se mueve demasiado rápido. Se calcula de la diferencia entre el valor actual de error y el valor del error anterior. También se multiplica por una constante (Kd) para escalarlo y homologarlo con el resto del sistema.

Sintonizar el controlador 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).



Esta es la imagen del modo de prueba de un subsistema de muñeca que tiene un potenciómetro como sensor (pot) y un controlador de motor conectado al motor. Tiene una serie de áreas que corresponden al PIDSubsystem.

1. El valor de la entrada análoga del voltaje del potenciómetro. Este es el valor de entrada del sensor.
2. Un intensificador que mueve el motor de la muñeca del brazo en cualquier dirección, con 0 como detenido. Los valores positivos y negativos corresponden a arriba o abajo.
3. Las constantes PID descritas arriba (F es para un valor preliminar que se utiliza en los lazos PID).

4. El punto de control que corresponde al valor de pot cuando la muñeca ha alcanzado el valor esperado.
5. Habilita el controlador PID - Ya no funciona, vea abajo.

Pruebe varias varias veces el controlador PID para obtener el rendimiento de motor deseado. Puede mirar el vídeo enlazado al principio de este artículo o en otras fuentes de Internet para obtener el rendimiento deseado.

Importante: 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.

Habilitar la Funcionalidad en el Nuevo Controlador PID

El siguiente ejemplo demuestra cómo crear un botón en la dashboard que habilite/deshabilite el controlador PID.

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));
}
```

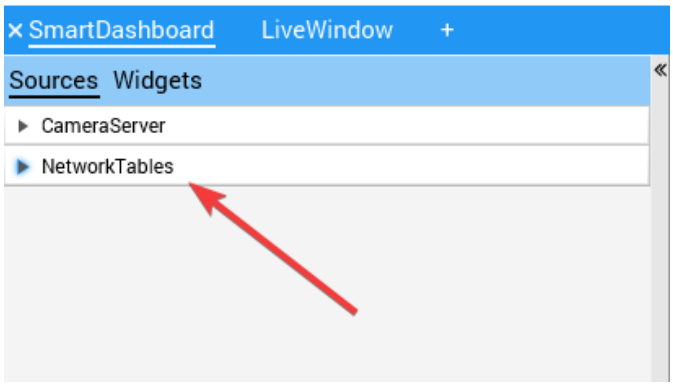
PYTHON

```
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))
```

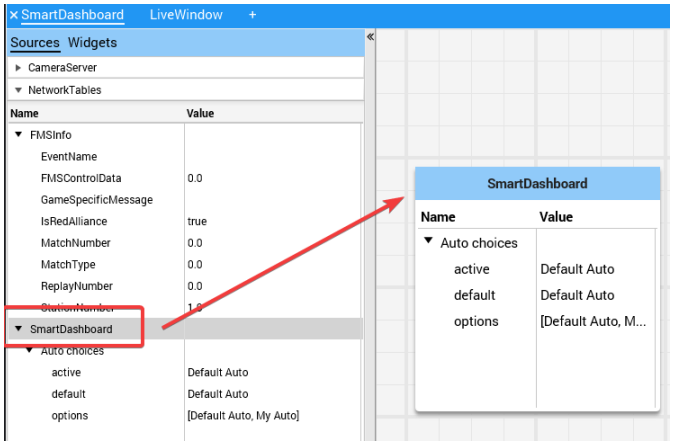
Ver jerarquías de datos

Arrastrar una clave con otras claves debajo de ella (más profundo en la jerarquía) muestra la jerarquía en un árbol, similar a las fuentes de NetworkTables a la izquierda.

Seleccione la fuente de datos:



Haga clic y arrastre la clave NetworkTables a la pestaña preferida.



11.2.4 Shuffleboard - Widgets personalizados

Built-in Plugins

Shuffleboard proporciona un número de plugins incorporados que manejan tareas comunes para el uso de FRC ®, como flujos de cámaras, todos los widgets y conexiones *NetworkTables*.

Base Plugin

La base plugin define todos los tipos de datos, widgets y diseños necesarios para el uso de FRC. *No* define ninguno de los tipos de fuente, ni ningún tipo de datos o widgets especiales para esos tipos de fuente. Estos son manejados por el *NetworkTables Plugin* y el *CameraServer Plugin*. Esta separación de preocupaciones facilita a los equipos la creación de plugins para tipos de fuentes o protocolos personalizados (por ejemplo, HTTP, ZeroMQ) para los tipos de datos FRC sin necesidad de un cliente NetworkTables.

CameraServer Plugin

El plugin del servidor de la cámara proporciona fuentes y widgets para ver las transmisiones de la cámara de la clase WPILib `CameraServer`.

Este plugin depende del *NetworkTables Plugin* para descubrir las transmisiones de cámara disponibles.

Detección de transmisiones

Las fuentes de CameraServer se descubren automáticamente mirando la NetworkTable `/CameraPublisher`.

```
/CameraPublisher
  /<camera name>
    streams=["url1", "url2", ...]
```

Por ejemplo, una cámara llamada «Camera» con un servidor en `roborio-0000-frc.local` tendría este diseño de tabla:

```
/CameraPublisher
  /Camera
    streams=["mjpeg:http://roborio-0000-frc.local:1181/?action=stream"]
```

Esta configuración detectará automáticamente todas las transmisiones de cámara alojadas en un roboRIO por la clase CameraServer en WPILib. Cualquier proyecto que no sea WPILib que quiera que aparezcan transmisiones de cámara en shuffleboard tendrá que configurar la entrada de transmisiones para el servidor de la cámara.

NetworkTables Plugin

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.

Este plugin maneja la conexión y reconexión a NetworkTables automáticamente, los usuarios de shuffleboard y los escritores de plugins personalizados no tendrán que preocuparse por las complejidades del protocolo de NetworkTables.

Crear un Plugin

Vista general

Plugins provide the ability to create custom widgets, layouts, data sources/types, and custom themes. Shuffleboard provides the following *built-in plugins*.

- NetworkTables Plugin: conecta los datos publicados en NetworkTables
- Base Plugin: Muestra tipos personalizados de datos FRC® en widgets personalizados
- CameraServer Plugin: Muestra transmisiones desde la CamaraServer.

Truco: Puede encontrar un ejemplo de complemento de Shuffleboard personalizado que crea un tipo de datos personalizado y un widget simple para mostrarlo aquí.

Crear un Plugin Personalizado

Para definir un complemento, la clase de complemento debe ser una subclase de edu.wpi.first.shuffleboard.api.Plugin o una de sus subclases. Un ejemplo de una clase de complemento sería el siguiente.

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 {

}
```

Explicaciones adicionales sobre cómo utilizar estos atributos, incluyendo versiones numéricas pueden encontrarse aquí.

Note que la anotación `@Description` es necesaria para notificar al cargador de plugin sobre las propiedades de la clase plugin personalizada. Las clases plugin pueden tener un constructor predeterminado, pero no pueden tener ningún argumento.

Construir 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"`

Los plugin pueden tener dependencias con otros plugin y bibliotecas, sin embargo, deben estar incluidos correctamente en el maven o en el archivo gradle build. Cuando un plugin depende de otro es una buena práctica definir esas dependencias para que el plugin no cargue cuando las dependencias no lo hagan. Esto puede realizarse utilizando la anotación `@Requires` como se muestra a continuación:

```
@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` especifica la versión mínima permitida del plugin que puede cargarse. Por ejemplo, si la `minVersion` es 1.4.5 y el plugin con la versión 1.4.7 se carga, se permitirá hacerlo. Sin embargo, si la versión fuese 1.2.4, entonces no se permitirá desde que es menos que la `minVersion`.

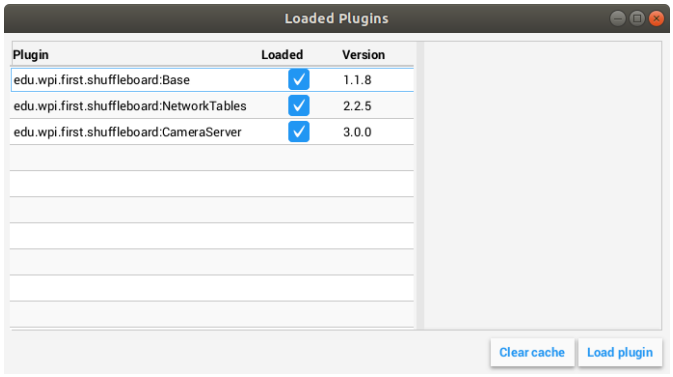
Desplegar un Plugin a Shuffleboard

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`

Después de desplegar, Shuffleboard almacenará la dirección del plugin para que cargue automáticamente la siguiente vez que se cargue Shuffleboard. Tal vez sea necesario seleccionar `Clear Cache` en el menú de plugin para remover o volver a cargar un plugin en Shuffleboard.

Agregar Manualmente 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.



Desde la ventana de plugin, escoja el botón “Cargar plugin” en la esquina inferior derecha, y seleccione su archivo jar.

Crear tipos de datos personalizados

Los widgets le permiten controlar y visualizar diferentes tipos de datos. Estos datos podrían ser integers o doubles o incluso Objetos de Java. Para visualizar estos tipos de datos utilizando widgets es útil crear una clase contenedora para estos. No es necesario crear su propia clase de Datos si el widget manejará tipos de datos con un solo campo, tales como doubles, arrays o strings.

Crear Clase de Datos

En este ejemplo, crearemos un tipo de datos personalizado para un punto 2D y sus coordenadas x e y. Para crear una clase de tipo de datos personalizada, debe extender la clase abstracta ComplexData. Su clase de datos personalizados también debe implementar el método `asMap()` que devuelve los datos representados como un mapa simple como se indica a continuación con la anotación `@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);
   }
}
```

También es una buena práctica anular los métodos por defecto `equals` y `hashcode` para asegurar que diferentes objetos se consideren equivalentes cuando sus campos sean iguales. El método `asMap()` debe devolver los datos representados en un simple objeto Map, ya que se mapeará a la entrada de NetworkTables a la que corresponde. En este caso, podemos representar el punto como sus coordenadas X e Y y devolver un `Map` que las contenga.

```
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);
   }
 }
```

Otros métodos pueden añadirse para recuperar o editar campos y variables de la instancia, sin embargo, es buena práctica hacer estas clases inmutables para prevenir cambios en la fuente de datos. En cambio, puede realizar una copia del objeto en lugar de manipular el objeto existente. Por ejemplo, si quisiera cambiar la coordenada y de su punto, podría definir el siguiente método:

```
public MyPoint2D withY(double newY) {
   return new MyPoint2D(this.x, newY);
}
```

Esto crea un nuevo objeto `MyPoint2D` y lo regresa con una nueva coordenada Y. Puede hacerse lo mismo para la coordenada X.

Crear un Tipo de Dato

Se pueden crear dos tipos de datos: Tipos de datos simples que solo contienen un campo (por ejemplo, un solo número o un string) y tipos de datos complejos que contienen varios campos (por ejemplo, múltiples strings o números).

Para definir un tipo de dato simple, la clase debe extender la clase `SimpleDataType<DataType>` con el tipo de dato necesario e implementar el método `getDefaultValue()`. En este ejemplo, usaremos un double como su tipo de dato simple.

```
public final class MyDoubleDataType extends SimpleDataType<Double> {

   private static final String NAME = "Double";
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   private MyDataType() {
      super(NAME, Double.class);
   }

   @Override
   public Double getDefaultValue() {
      return 0.0;
   }

}
```

El constructor de la clase es privado para asegurar que una sola instancia del tipo de dato exista

Para definir un tipo de dato complejo, la clase debe extender la clase `ComplexDataType` y sobrescribir los métodos `fromMap()` y `getDefaultValue()`. Se continuará usando como ejemplo la clase MyPoint2D para demostrar cómo se ve una clase de tipos de datos complejos.

```
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);
   }

}
```

El código de arriba funciona de la siguiente forma:

El método `fromMap()` crea un nuevo MyPoint2D utilizando los valores de la entrada de NetworkTables a la que está vinculado. El método `getOrDefault` devolverá 0.0 si no puede obtener los valores de la entrada. El método `getDefaultValue` devolverá un nuevo objeto `MyPoint2D` si no hay ninguna fuente presente.

Exportar un tipo de dato al plugin

Para que Shuffleboard reconozca el tipo de dato, el plugin debe exportarlos al sobrescribir el método `getDataTypes`. Por ejemplo,

```
public class MyPlugin extends Plugin {

   @Override
   public List<DataType> getDataTypes() {
      return List.of(PointDataType.Instance);
   }

}
```

Crear un widget

Los widget permiten visualizar, cambiar e interactuar con la información publicada a través de distintas fuentes de información. Los plugin CameraServer, NetworkTables y Base proveen de widgets para controlar tipos de datos básicos (incluidos tipos de datos específicos de FRC). Sin embargo, los widgets personalizados permiten controlar sus tipos de datos personalizados creados en las secciones anteriores u Objetos de Java.

La interface básica `Widget` es hereda de las interfaces `Component` y `Sourced`. `Component` es el bloque de construcción más básico disponible en Shuffleboard. `Sourced` es una interface para objetos que puedan manejar e interactuar con fuentes de datos para mostrar o visualizar datos. Los widgets que no admiten enlaces de datos, pero tienen nodos hijos no utilizarían la interfaz `Sourced`, sino únicamente la interfaz `Component`. Ambas son bloques de construcción básicos para crear widgets y permiten visualizar y modificar datos.

Un buen widget permite al usuario final personalizar al widget de la forma que más le convenga. Un ejemplo podría ser permitir al usuario controlar el rango de un intensificador numérico, esto es, su mínimo y máximo o la orientación del mismo intensificador. La vista del widget o el cómo se ve se define utilizando FXML. `FXML` es un lenguaje basado en XML muy útil para definir layouts estáticos para los widgets («Panes», Etiquetas y Controles).

Más información sobre FMXL puede encontrarse aquí.

Definir el XML de un widget

En este ejemplo, creará dos intensificadores para ayudarle a controlar las coordenadas X,Y del objeto Point2D creado en secciones anteriores. Es útil colocar el archivo FXML en el mismo paquete que la clase Java.

Para crear una ventana vacía para el widget, es necesario crear un `Pane`. Un Pane es un nodo padre que contiene otros nodos hijos, en este caso, 2 intensificadores. Existen muchos tipos de Pane, como se muestra:

- Stack Pane
  - Permiten sobreponer elementos. Además, StackPanes centran, de forma predeterminada, a los nodos hijos.
- Grid Pane
  - Son extremadamente útiles para definir elementos hijos al utilizar un sistema coordinado al crear una cuadrícula flexible de filas y columnas en el plano.

- Flow Pane
  - Envuelven a todos los nodos hijos en el conjunto de límites. Los nodos hijos pueden fluir verticalmente (contenidos en el límite de altura del panel) u horizontalmente (contenidos por el límite de anchura del panel).
- Anchor Pane
  - Permiten a los elementos hijos ser posicionados encima, debajo, a la izquierda o derecha, o al centro del panel.

Los layout panes son extremadamente útiles para colocar nodos hijos en una fila horizontal al utilizar HBox o en una columa vertical usando VBox.

La sintaxis básica para definir un Pane usando FXML sería la siguiente:

```
<?import javafx.scene.layout.*?>
<StackPane xmlns:fx="http://javafx.com/fxml/1" fx:controller="/path/to/widget/class"␣
↪fx:id="root">
   ...
</StackPane>
```

El atributo `fx:controller` contiene el nombre de la clase del widget. Una instancia de esta clase se crea cuando se carga un archivo FXML. Para este trabajo, la clase del controlador debe tener un constructor vacío.

Crear una clase widget

Ahora que tiene el Pane, puede añadir elementos hijos al panel. Por ejemplo, puede agregar dos objetos intensificadores. Recuerde añadir `fx:id` a cada elemento para que puedan ser referenciados en la clase Java que hará más adelante. Utilizará *VBox*` para colocar su intensificador uno encima de otro.

```
<?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>
```

Ahora que ha terminado de crear el archivo FXML, puede crear la clase del widget. La clase del widget debe incluir la anotación `@Description` que establece que los tipos de datos permitidos por el widget y el nombre del widget. Si la anotación`` @Description`` no está presente, la clase plugin debe implementar el método `get()` para regresar sus widgets.

También debe incluir una anotación `@ParametrizedController` que apunte al archivo FXML que contiene el diseño del widget. Si la clase que solo admite una fuente de datos, debe extender la clase `SimpleAnnotatedWidget`. Si la clase admite múltiples fuentes de datos, debe extender la clase `ComplexAnnotatedWidget`. Para obtener más información, consulte *Tipos de Widget*.

```
import edu.wpi.first.shuffleboard.api.widget.Description;
import edu.wpi.first.shuffleboard.api.widget.ParametrizedController;
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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 no está utilizando un tipo de dato personalizado, puede referenciar a cualquier tipo de dato Java (como `Double.class`) o si el widget no necesita enlaces de datos, puede pasar `NoneType.class`.

Ahora que ha creado la clase, puede crear campos para los widgets declarados en el archivo FXML usando la anotación `@FXML`. Para los dos intensificadores, un ejemplo sería:

```
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;
}
```

Para visualizar el panel en el widget personalizado es necesario sobreescribir el método `getView()` y regresar el `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> {

   @FXML
   private StackPane root;
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   @FXML
   private Slider xSlider;

   @FXML
   private Slider ySlider;

   @Override
   public Pane getView() {
      return root;
   }

}
```

Enlazar Elementos y añadir Escuchadores

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 ejemplo, en este caso, sería cambiar las coordenadas X,Y del objeto 2DPoint al cambiar los valores de xSlider y ySlider, respectivamente.

Una buena práctica es enlazar en el método `initialize()` con la etiqueta `@FXML` que requiere llamar al método de FXML si el método no es `público`.

```
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() {
      return root;
   }
```

(continúe en la próxima página)

(proviene de la página anterior)

```
 }
```

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.

Usar un escuchador es otra manera de cambiar los valores cuando el deslizador o la fuente de datos cambia. Un ejemplo de un escuchador para nuestro deslizador puede ser:

```
xSlider.valueProperty().addListener((observable, oldValue, newValue) ->␣
↪setData(getData().withX(newValue));
```

En este caso, el método `setData()` establece el valor en la fuente de datos del widget al `newValue`.

Explorar componentes personalizados

Los widgets no se descubren automáticamente cuando se cargan plugins, el plugin se debe exportar explícitamente para poder ser utilizado. Este acercamiento permite que múltiples plugins sean definidos en el mismo JAR.

```
@Override
public List<ComponentType> getComponents() {
  return List.of(WidgetType.forAnnotatedWidget(Point2DWidget.class));
}
```

Configurar un widget predeterminado para tipos de datos

Para configurar su widget como la predeterminada para su tipo de dato personalizado, puede sobreescribir el método `getDefaultComponents()` en la clase de su plugin que almacena un Mapa para todos los widgets predeterminados, como se muestra a continuación:

```
@Override
public Map<DataType, ComponentType> getDefaultComponents() {
   return Map.of(Point2DType.Instance, WidgetType.forAnnotatedWidget(Point2DWidget.
↪class));
}
```

Temas Personalizados

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).

La shuffleboard viene con tres temas por defecto: Material Light, Material Dark y Midnight. Estas son variaciones de color en la misma hoja de estilo de diseño de material. Además, heredan de una hoja de estilo``base.css`` que define estilos para los componentes personalizados,

definidos en shuffleboard o bibliotecas que utiliza; la hoja de estilo de diseño de material base solo se aplica a los componentes de la interfaz de usuario integrados en JavaFX.

Hay dos maneras de definir un tema personalizado: colocar las hojas de estilo en un directorio con el nombre del tema en `~/Shuffleboard/themes`; por ejemplo, un tema teórico «Amarillo» podría colocarse ahí

```
~/Shuffleboard/themes/Yellow/yellowtheme.css
```

Todas las hojas de estilo del directorio se tratarán como parte del tema.

Cargar Temas a través de Complementos

Los temas personalizados también se pueden definir mediante complementos/plugins. Esto hace que sea más fácil compartirlos y agruparlos con widgets personalizados, pero son un poco más difíciles de definir. El objeto de tema necesitará una referencia a una clase definida en el complemento para que el cargador del complemento pueda determinar dónde se encuentran las hojas de estilo. Si se pasa una clase que *no* está presente en el JAR en el que se encuentra el complemento, el tema no podrá utilizarse.

```
@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);
  }

}
```

Modificar o Ampliar los Temas Predeterminados de la Shuffleboard

Los temas Material Light y Material Dark de la Shuffleboard proporcionan una gran parte del marco para los temas claros y oscuros, respectivamente, así como muchos estilos específicos de los componentes de la interfaz de usuario de la Shuffleboard, ControlsFX y Medusa para adaptarse al diseño de estilo material.

Los temas que quieran modificar estos temas deben agregar declaraciones de `import` para estas hojas de estilo:

```
@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 */
```

Tenga en cuenta que `base.css` importa internamente `material.css`, y `light.css`, `dark.css` y `midnight.css` todos importan `base.css`, por lo que la importación `light.css` también importará implícitamente tanto `base.css` como `material.css`.

Código Fuente para los Archivos 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

Muestras de Color del Diseño del Material

El diseño CSS del material utiliza variables de muestra de color para casi todo. Estas variables se pueden configurar desde los archivos CSS personalizados, lo que reduce la cantidad de código personalizado necesario.

Las variables `-swatch- <100|200|300|400|500>` definen tonos progresivamente más oscuros del mismo color primario. El tema claro usa los tonos de azul predeterminados establecidos en `material.css`, pero el tema oscuro los reemplaza con tonos de rojo. `-swatch-<|light|dark>-gray` define tres niveles de gris para usar con varios colores de fondo o texto.

Anulación de los Colores de Muestra

Reemplazo de azul con rojo (claro)

```
@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%);
}
```

Reemplazo de rojo con azul (oscuro)

```
@import "/edu/wpi/first/shuffleboard/app/dark.css"

.root {
    -swatch-100: #BBDEFB;
    -swatch-200: #90CAF9;
    -swatch-300: #64BEF6;
    -swatch-400: #42A5F5;
    -swatch-500: #2196F3;
}
```

Tipos de Widget

Mientras `Widget` es bastante sencillo en lo que respecta a la interfaz, hay varias implementaciones intermedias para facilitar definir el widget.

Clase	Descripción
`AbstractWidget`	Implementos `getProperties()`, `getSources()`, y `titleProperty()`
`SingleTypeWidget <T>`	Agrega propiedades para widgets que solo admiten un único tipo de datos
`AnnotatedWidget`	Agrega implementaciones predeterminadas para `getName()` y `getDataTypes()` para widgets con una anotación `@Description`
`SingleSourceWidget`	Para widgets con una sola fuente (de forma predeterminada, los widgets admiten múltiples fuentes)
`SimpleAnnotatedWidget<T>`	Combina `SingleTypeWidget<T>`, `AnnotatedWidget`, y `SingleSourceWidget`

También hay dos anotaciones para ayudar a definir los widgets:

Nombre	Descripción
`@ParametrizedController`	Permite que los widgets sean controladores FXML para vistas JavaFX definidas a través de FXML
`@Description`	Permite definir el nombre y los tipos de datos admitidos en una sola línea.

AbstractWidget

Esta clase implementa `getProperties()`, `getSources()`, `addSource()`, y `titleProperty()`. También define un método `exportProperties(Property<?>...)` para que las subclases puedan añadir fácilmente propiedades personalizadas al widget, o propiedades para los componentes JavaFX en el widget. La mayoría de los widgets en la base plugin usan esto.

SingleTypeWidget

Un tipo de widget que solo admite un tipo de datos. Esta interfaz está parametrizada y tiene métodos para configurar u obtener los datos, así como un método para obtener el tipo de datos (único) del 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 tipo de widget que solo utiliza una fuente.

SimpleAnnotatedWidget

Una combinación de `` SingleTypeWidget <T>``, `` AnnotatedWidget “” y `` SingleSource-Widget “”. La mayoría de los widgets del complemento base se extienden desde esta clase. Esto también tiene un campo `protegido'' llamado ``dataOrDefault` que permite que las subclases usen un valor de datos predeterminado si el widget no tiene una fuente, o si la fuente proporciona `nulo`.

@ParametrizedController

Esta anotación se puede colocar en una clase de widget para que permita a shuffleboard saber que es un controlador FXML para una vista JavaFX definida a través de FXML. La anotación toma un único parámetro que define dónde se encuentra el archivo FXML *en relación con la clase en la que se coloca*. Por ejemplo, un widget en el directorio `src/main/java/com/acme` que es un controlador FXML para un archivo FXML en `src/main/resources/com/acme` puede usar la anotación como

```
@ParametrizedController("MyWidget.fxml")
```

o como

```
@ParametrizedController("/com/acme/MyWidget.fxml")
```

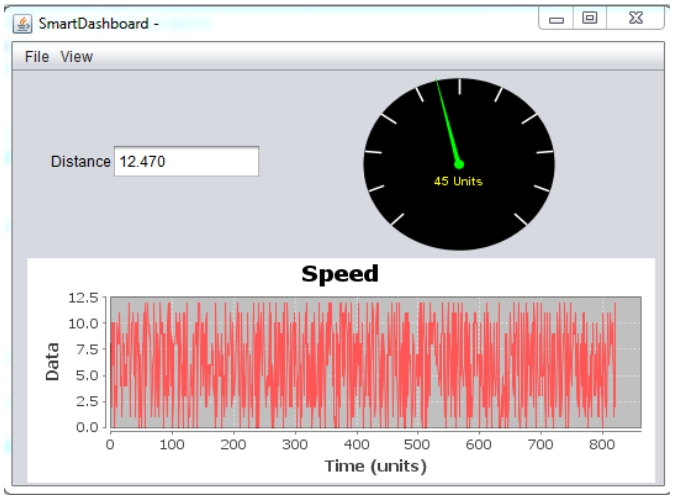
@Description

Esto permite que los widgets tengan su nombre y los tipos de datos admitidos definidos por una sola anotación, cuando se utilizan junto con *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 Introducción a SmartDashboard

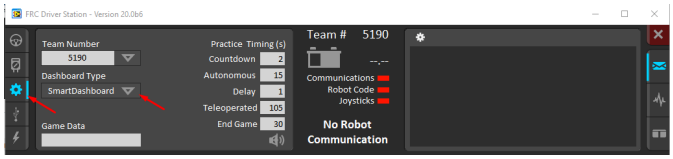


La SmartDashboard es un programa de Java que proporciona datos del robot en tiempo real. La SmartDashboard te ayuda con las siguientes cosas:

- Proporciona los datos de su elección del robot mientras el código está ejecutándose. Pueden mostrase como simples recuadros de texto o en un tipo de visualización más elaborado como gráficas, diales, etc.
- Muestra el estado del robot, los comandos que se están ejecutando al momento y el estado de cualquier subsistema.
- Muestra botones que puedes presionar para establecer variables en tu robot.
- Permite elegir opciones de inicio en la dashboard que pueden leerse en el programa del robot.

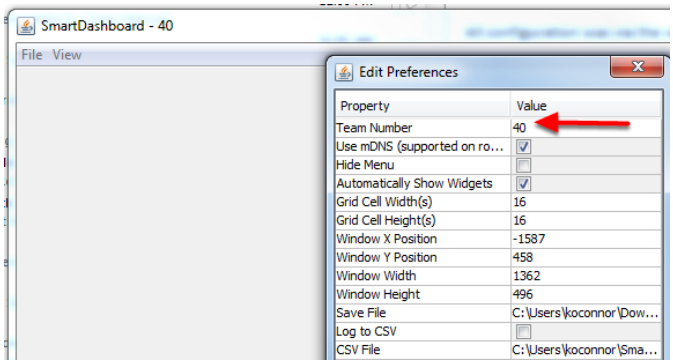
Los datos mostrados son automáticamente formateados en tiempo-real a medida que los datos son enviados desde el robot, pero usted puede cambiar el formato o la visualización de los widgets y después guardar los diseños nuevos de la pantalla para ser usados de nuevo más tarde. Con todas estas opciones, sigue siendo extremadamente fácil de usar. Para mostrar algunos datos en la dashboard, simplemente llama a uno de los métodos SmartDashboard con los datos y su nombre y el valor aparecerá automáticamente la pantalla de la dashboard.

Instalar la SmartDashboard



La SmartDashboard está empaquetada con el WPILib Installer y puede ser iniciada directamente desde la Driver Station al seleccionar el botón **SmartDashboard** en la pestaña Setup.

Configurar el número de equipo



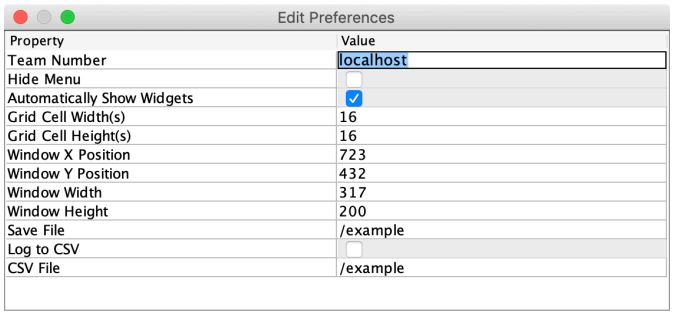
La primera vez que inicie la SmartDashboard se le solicitará su número de equipo. Para cambiar el número de equipo después de esto: clic en **File > Preferences**, para abrir el cuadro de Preferencias. Doble clic en el cuadro a la derecha de **Team Number** e ingrese su número de equipo de FRC®, después, dé clic afuera del cuadro para guardar.

Nota: La SmartDashboard tardará un momento en configurarse por el número de equipo, no se alarme.

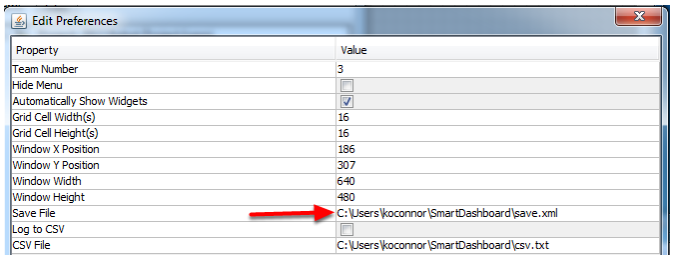
Configuración de una ubicación personalizada de NetworkTables

De forma predeterminada, SmartDashboard buscará instancias de NetworkTables que se ejecuten en un RoboRIO conectado, pero a veces es útil buscar NetworkTables en una dirección IP diferente. Para conectarse a SmartDashboard desde un host que no sea roboRIO, abra las preferencias de SmartDashboard en el menú `File` y en el campo `Team Number`, ingrese la dirección IP o el nombre del host de NetworkTables.

Esta opción es increíblemente útil para usar SmartDashboard con *WPILib simulation*. Simplemente agregue `localhost` al campo `Team Number` y SmartDashboard detectará su robot alojado localmente.

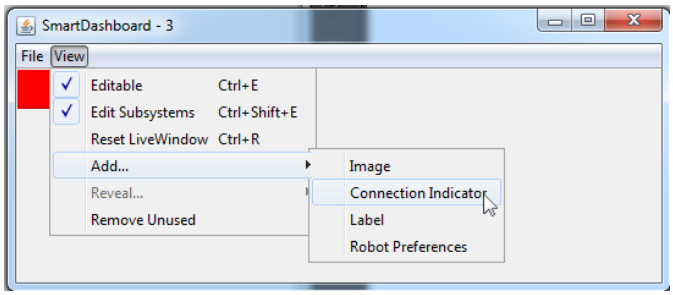


Ubicar Save File



Puede que desee personalizar la dirección de guardado de la SmartDashboard. Para hacer esto de click en el recuadro continuo a **Save File**, después seleccione la carpeta en la que le gustaría guardar la configuración. Los archivos guardados en los directorios de instalación de los componentes de WPILib, probablemente se sobrescribirán en actualizaciones a las herramientas.

Añadir un indicador de conexión



Muchas veces es útil revisar si la SmartDashboard está conectada al robot. Para agregar un indicador de conexión, seleccione **View > Add > Connection Indicator**. El indicador será rojo cuando esté desconectado y verde cuando esté conectado. Para mover o modificar el tamaño del indicador, seleccione **View > Editable** para cambiar la SmartDashboard a modo editable, después arrastra el centro del indicador para moverlo, o las esquinas para cambiar su tamaño. Seleccione el objeto **Editable** de nuevo para ajustarlo en su lugar.

Añadir widgets a la SmartDashboard

Los Widgets se agregan automáticamente al SmartDashboard por cada "clave" enviada por el código del robot. Para instrucciones sobre como hacer que el código del robot escriba al SmartDashboard vea Displaying Expressions from Within the Robot Program 1.

11.3.2 Mostrar expresiones desde el programa del robot

Nota: A menudo la depuración o el monitoreo del estado del robot involucra ingresar una serie de valores a la consola y verlos fluir. Con la SmartDashboard puede ingresar valores a un GUI que está automáticamente construido basado en su programa. A medida que los valores se actualizan, los elementos correspondientes GUI cambian su valor – no hay necesidad de intentar capturar números que aparecen en la pantalla.

Escribir valores a la 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())
```

Puede escribir valores tipo Boolean, Numeric o String a la SmartDashboard simplemente llamando el método correcto de acuerdo al tipo e incluyendo el nombre del valor de los datos, no es necesario un código adicional. Cada vez que escriba otro valor con el mismo nombre en su programa, aparece en el mismo elemento UI en la pantalla en la driver station o la computadora en uso. Como puede imaginar este es una gran manera de depurar y obtener el estado de su robot mientras esté funcionando.

Crear widgets en la SmartDashboard

Los widgets se completan en SmartDashboard automáticamente, no se requiere la intervención del usuario. Tenga en cuenta que los widgets solo se completan cuando se escribe el valor por primera vez, es posible que deba habilitar su robot en un modo particular o activar una rutina de código particular para que aparezca un elemento. Para modificar la apariencia del widget, consulte las dos secciones siguientes: *Changing the Display Properties of a Value* y *Changing the Display Widget Type for a Value*.

Datos desactualizados

SmartDashboard utiliza *NetworkTables* para comunicar valores entre el robot y la Driver Station. NetworkTables actúa como una tabla distribuida de pares de nombres y valores. Si se añade un par nombre/valor al cliente (portátil) o al servidor (robot), se replica al otro. Si se elimina un par nombre/valor de, por ejemplo, el robot pero el SmartDashboard o el OutlineViewer siguen funcionando, cuando el robot se reinicie, los valores antiguos seguirán apareciendo en el SmartDashboard y el OutlineViewer porque nunca dejaron de funcionar y siguen teniendo esos valores en sus tablas. Cuando el robot se reinicie, esos valores antiguos se replicarán en el robot.

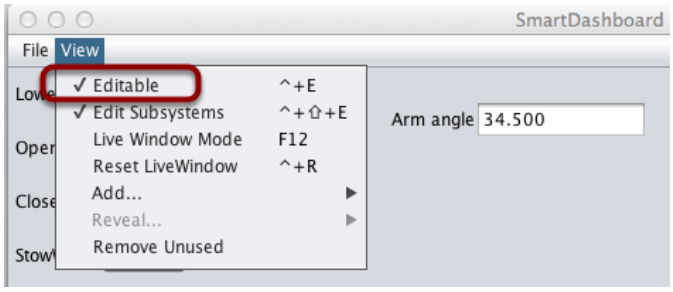
Para garantizar que el SmartDashboard y el OutlineViewer muestren los valores actuales, es necesario reiniciar los clientes de NetworkTables y el robot al mismo tiempo. De esta forma, los valores antiguos que uno de ellos mantiene no se replicarán a los otros.

Normalmente esto no es un problema si el programa no está cambiando constantemente, pero si el programa está en desarrollo y el conjunto de claves que está siendo añadido a los NetworkTables está cambiando constantemente, entonces puede ser necesario hacer el reinicio de todo para ver en ese momento las actualizaciones.

11.3.3 Cambiar las propiedades de visualización de un valor

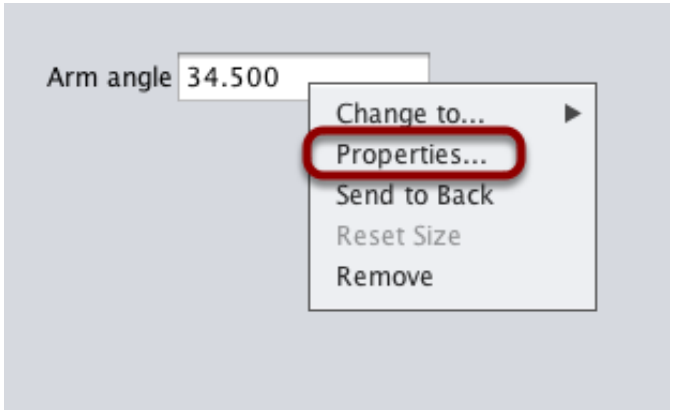
Each value displayed with SmartDashboard has a set of properties that effect the way it's displayed.

Configurar el modo de edición en el visualizador de la SmartDashboard



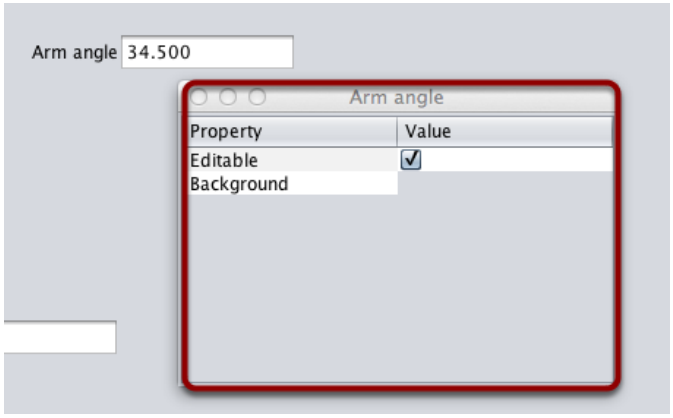
La SmartDashboard tiene dos modos de operación: el modo de visualización y el modo de edición. En el segundo, usted puede cambiar la posición de los widgets en la pantalla y editar sus propiedades. Para cambiar al modo de edición en la SmartDashboard, debe presionar el menú "View", luego seleccione "Editable" para acceder al modo de edición.

Acceder al editor de propiedades de un widget



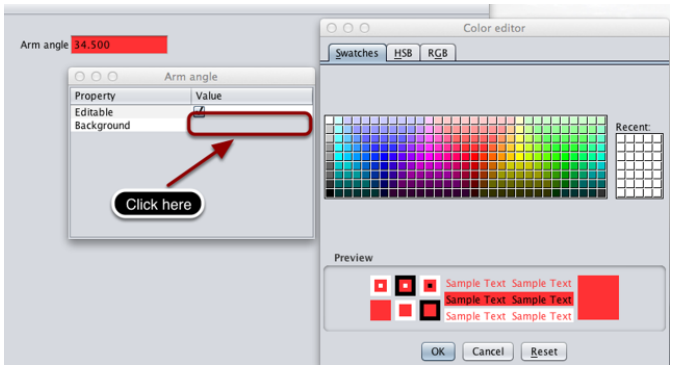
Una vez que ha cambiado al modo de edición, podrá visualizar las propiedades de los widgets al presionar el botón derecho del ratón sobre el widget y seleccionar "Properties...".

Configurar las propiedades en un campo



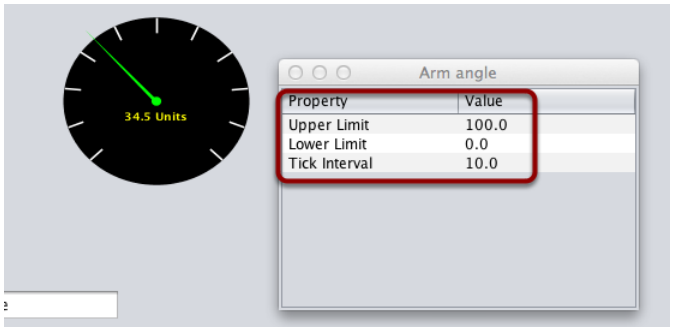
Se mostrará una ventana de diálogo en respuesta del menú de «Properties…» en el menú del click derecho.

Editar el color de fondo de los widgets



Para cambiar el valor de una propiedad, por ejemplo, el color de fondo, haga click en el color de fondo mostrado (gris, en este caso) y escoja un color en el editor que aparecerá en pantalla. Este será el nuevo color de fondo de los widgets.

Editar las propiedades de otros widgets



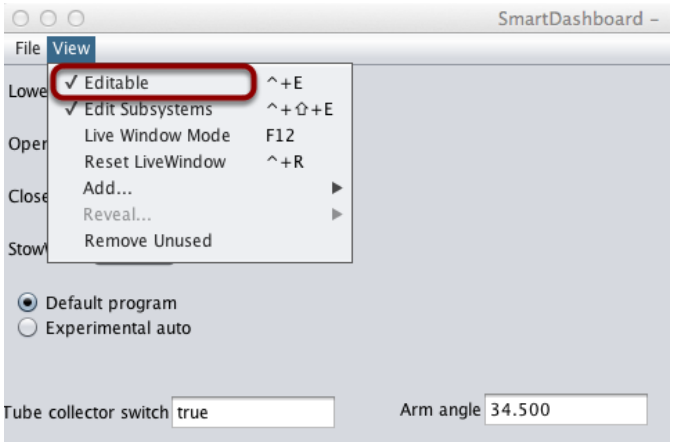
Cada tipo de widget tiene diferentes conjuntos de propiedades editables para cambiar su apariencia. En este ejemplo, los límites superior e inferior, y el intervalo entre tics son parámetros

configurables.

11.3.4 Cambiar el tipo de widget de pantalla por un valor

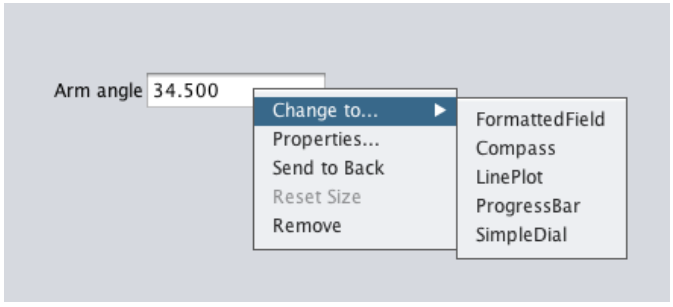
Se puede cambiar el tipo de widget que muestra los valores con el SmartDashboard. Los widgets permitidos dependen del tipo de valor que se muestra.

Ajuste del modo de edición



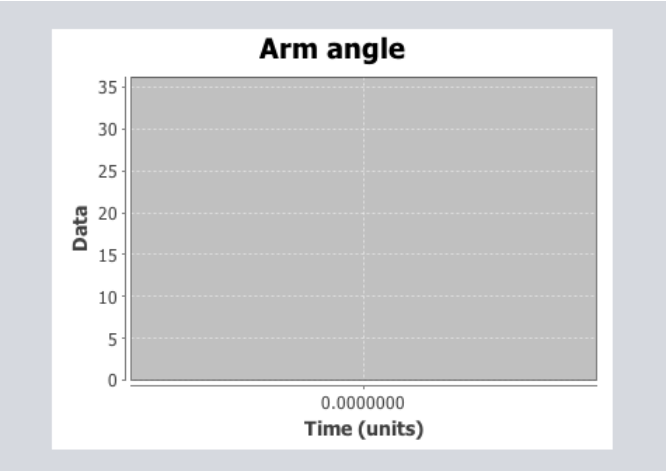
Asegúrate de que el SmartDashboard esté en modo de edición. Esto se hace seleccionando `Editable` de la `vista del menú`.

Elección del tipo de widget



Haga clic con el botón derecho del ratón en el widget y selecciona `Cambiar a...`. Luego, elija el tipo de widget a usar para el valor en particular. En este caso elegimos `LinePlot`.

Mostrando el nuevo tipo de widget



Se muestra el nuevo tipo de widget. En este caso, un gráfico de líneas, mostrará los valores del ángulo del brazo a lo largo del tiempo. Puede configurar las propiedades del gráfico para que se ajuste mejor a sus datos haciendo clic con el botón derecho del ratón y seleccionando `Propiedades...`. Ver: :doc:`Cambiar las propiedades de visualización de un valor <changing-display-properties> `.

11.3.5 Elección de un programa autónomo

A menudo, los equipos tienen más de un programa autónomo, ya sea por razones competitivas o para probar software nuevo. Los programas a menudo varían agregando cosas como retrasos de tiempo, diferentes estrategias, etc. Los métodos para elegir la estrategia a ejecutar generalmente involucran interruptores, botones de joystick, perillas u otras entradas basadas en hardware.

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

Nota: 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;
    }
  }
```

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

Nota: The code snippets shown below are part of the HatchbotTraditional example project (Java, C++, Python):

Creación del objeto SendableChooser

En `RobotContainer`, crea una variable para mantener una referencia a un objeto `SendableChooser`. Se pueden crear dos o más comandos y almacenarlos en nuevas variables. Usando el `SendableChooser`, se puede elegir entre ellos. En este ejemplo, se muestran como opciones `SimpleAuto` y `ComplexAuto`.

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()
```

Configuración de SendableChooser

Imagine que tiene dos programas autónomos para elegir y que están encapsulados en los comandos `SimpleAuto` y `ComplexAuto`. Para elegir entre ellos:

En `RobotContainer`, cree un objeto `SendableChooser` y agregue instancias de los dos comandos. Puede haber cualquier número de comandos, y el agregado como predeterminado (`setDefaultOption`), se convierte en el que se selecciona inicialmente. Tenga en cuenta que cada comando se incluye en una llamada al método `setDefaultOption()` o `addOption()` en la instancia de `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)
```

Iniciar un comando autónomo

En `Robot.java`, cuando comienza el período autónomo, el objeto `SendableChooser` es sondeado para obtener el comando seleccionado y ese comando debe ser programado.

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()
```

Ejecución del planificador durante el modo autónomo

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.

Nota: La ejecución del planificador puede ocurrir en la función `autonomPeriodic()` o `robotPeriodic()`, ambas funcionarán de manera similar en modo autónomo.

Java

```
40  @Override
41  public void robotPeriodic() {
42    CommandScheduler.getInstance().run();
43  }
```

C++ (Source)

```
29 void Robot::RobotPeriodic() {
30   frc2::CommandScheduler::GetInstance().Run();
31 }
```

Cancelación del mando autónomo

En `Robot.java`, cuando comience el período de teleoperado, se cancelará el comando autónomo.

Java

```
78   @Override
79   public void teleopInit() {
80     // This makes sure that the autonomous stops running when
81     // teleop starts running. If you want the autonomous to
82     // continue until interrupted by another command, remove
83     // this line or comment it out.
84     if (m_autonomousCommand != null) {
85       m_autonomousCommand.cancel();
86     }
87   }
```

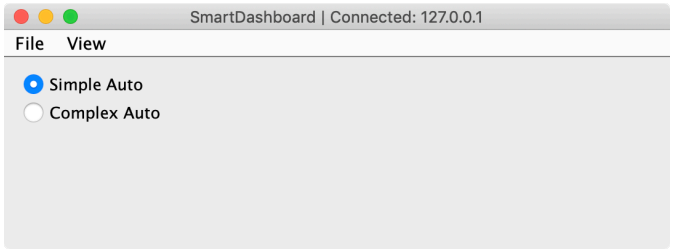
C++ (Source)

```
56 void Robot::TeleopInit() {
57   // This makes sure that the autonomous stops running when
58   // teleop starts running. If you want the autonomous to
59   // continue until interrupted by another command, remove
60   // this line or comment it out.
61   if (m_autonomousCommand != nullptr) {
62     m_autonomousCommand->Cancel();
63     m_autonomousCommand = nullptr;
64   }
65 }
```

Python

```
51     def teleopInit(self) -> None:
52         # This makes sure that the autonomous stops running when
53         # teleop starts running. If you want the autonomous to
54         # continue until interrupted by another command, remove
55         # this line or comment it out.
56         if self.autonomousCommand:
57             self.autonomousCommand.cancel()
```

Pantalla SmartDashboard

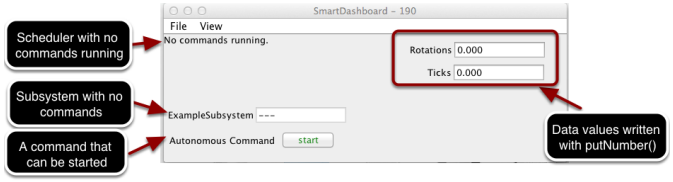


Cuando se ejecuta SmartDashboard, las opciones del `SendableChooser` se muestran automáticamente. Simplemente puede elegir una opción antes de que comience el período autónomo y se ejecutará el comando correspondiente.

11.3.6 Visualizar el estado de comandos y subsistemas

Si está utilizando las funciones de programación basadas en comandos de WPILib, encontrará que están muy bien integradas con SmartDashboard. Puede ayudar a diagnosticar lo que está haciendo el robot en cualquier momento y le brinda control y una vista de lo que se está ejecutando actualmente.

Descripción general de las pantallas de comando y subsistema



Con la SmartDashboard puede visualizar el estado de los comandos y subsistemas en su programa de robot de varias formas. Los resultados deberían reducir significativamente el tiempo de depuración de sus programas. En esta imagen puede ver una serie de pantallas posibles. Aquí se muestran:

- El Programador actualmente con `No hay comandos en ejecución`. En el siguiente ejemplo, puede ver cómo se ve con algunos comandos en ejecución que muestran el estado del robot.
- Un subsistema, `ExampleSubsystem` que indica que actualmente no hay comandos en ejecución que lo «requieran». Cuando los comandos se estén ejecutando, indicará el nombre de los comandos que están usando el subsistema.
- Un comando escrito en SmartDashboard que muestra un botón de `` inicio “” que se puede presionar para ejecutar el comando. Esta es una excelente manera de probar sus comandos uno a la vez.
- Y algunos valores de datos escritos en la dashboard para ayudar a depurar el código que se está ejecutando.

En los siguientes ejemplos, verá cómo se vería la pantalla cuando se ejecutan comandos y el código que produce esta pantalla.

Visualización del estado del programador de comandos

JAVA

```
SmartDashboard.putData(CommandScheduler.getInstance());
```

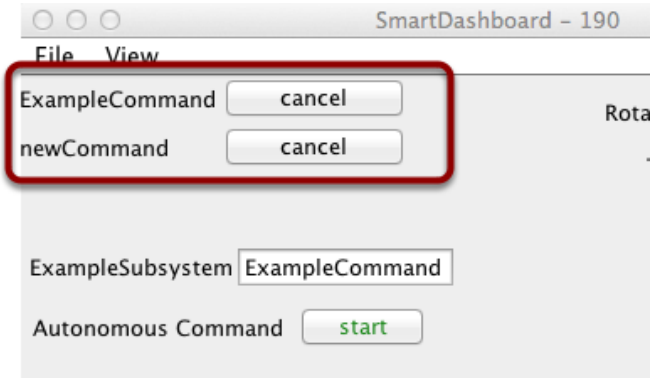
C++

```
frc::SmartDashboard::PutData(frc2::CommandScheduler::GetInstance());
```

PYTHON

```
from wpilib import SmartDashboard
from commands2 import CommandScheduler

SmartDashboard.putData(CommandScheduler.getInstance())
```

Puede mostrar el estado del Programador (el código que programa la ejecución de sus comandos). Esto se hace fácilmente agregando una sola línea al método `RobotInit` en su RobotProgram como se muestra aquí. En este ejemplo, la instancia del Programador se escribe utilizando el método `putData` en SmartDashboard. Esta línea de código produce la visualización de la imagen anterior.



Este es el estado del programador cuando hay dos comandos en ejecución, `ExampleCommand` y `newCommand`. Esto reemplaza el mensaje `No hay comandos en ejecución` de la imagen de pantalla anterior. Puede ver los comandos que se muestran en la dashboard a medida que se ejecuta el programa y se activan varios comandos.

Visualizar el estado del subsistema

JAVA

```
SmartDashboard.putData(exampleSubsystem);
```

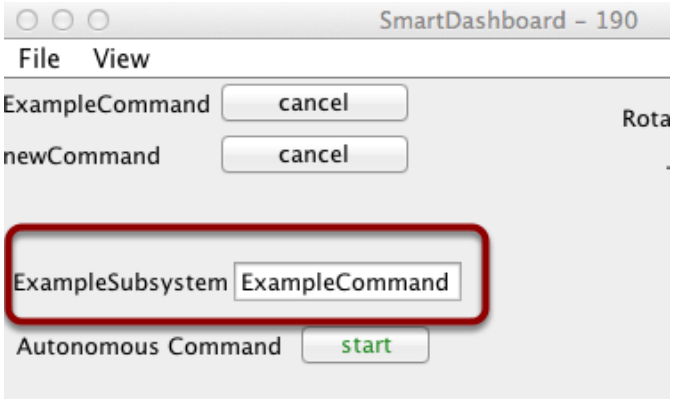
C++

```
frc::SmartDashboard::PutData(&exampleSubsystem);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData(exampleSubsystem)
```

En este ejemplo, estamos escribiendo la instancia de comando, `exampleSubsystem` y la instancia de la clase `ExampleSubsystem` en el SmartDashboard. Esto provoca la visualización mostrada en la imagen anterior. El campo de texto contendrá algunos guiones, `---` indicando que ningún comando está usando este subsistema, o el nombre del comando que usa actualmente este subsistema.



Los comandos en ejecución «requerirán» subsistemas. Ese es el comando que reserva el subsistema para su uso exclusivo. Si muestra un subsistema en la SmartDashboard, mostrará qué comando lo está usando actualmente. En este ejemplo, `ExampleSubsystem` está siendo utilizado por `ExampleCommand`.

Activar comandos con un botón

JAVA

```
SmartDashboard.putData("Autonomous Command", exampleCommand);
```

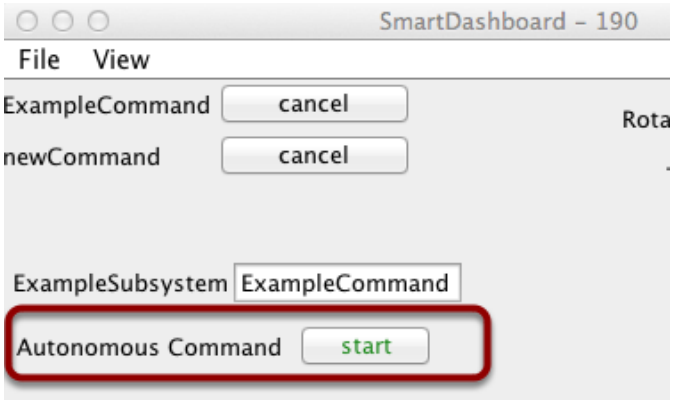
C++

```
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.

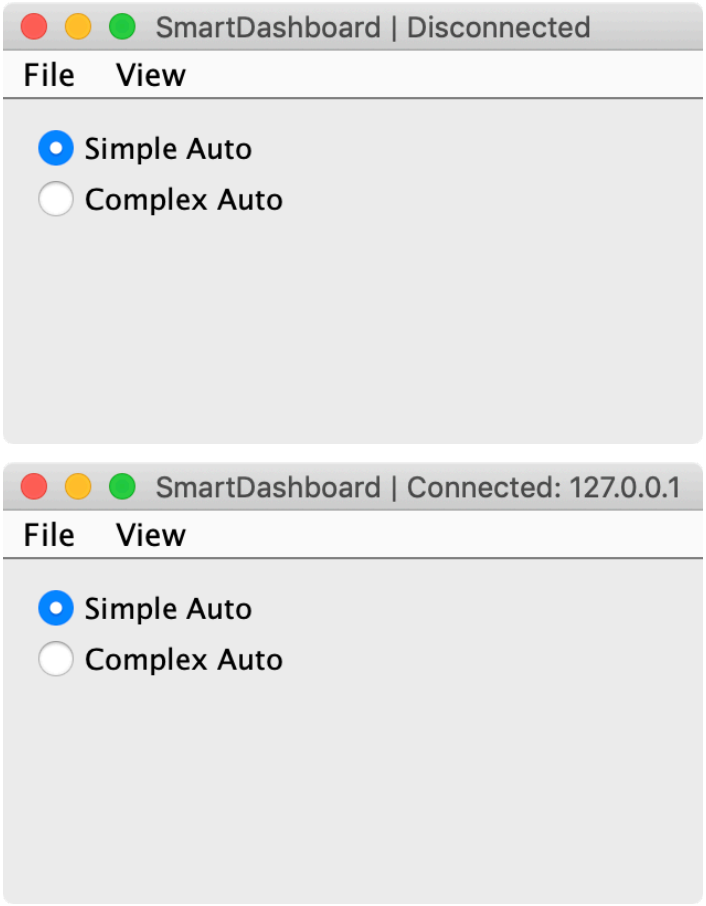


En este ejemplo, puede ver un botón etiquetado como `Autonomous Command`. Al presionar este botón se ejecutará el comando asociado y es una excelente manera de probar los comandos uno a la vez sin tener que agregar código de prueba desechable a su programa de robot. Agregar botones para cada comando simplifica la prueba del programa, un comando a la vez.

11.3.7 Verificando que SmartDashboard está funcionando

Indicador de conexión

SmartDashboard incluirá automáticamente el estado de la conexión y la dirección IP de la fuente NetworkTables en el título de la ventana.



Widget de indicador de conexión

SmartDashboard includes a connection indicator widget which will turn red or green depending on the connection to NetworkTables, usually provided by the roboRIO. For instructions to add this widget, look at *Adding a Connection Indicator* in the SmartDashboard Intro.

Ejemplo de programa de robot

JAVA

```
public class Robot extends TimedRobot {
   double counter = 0.0;

   public void teleopPeriodic() {
      SmartDashboard.putNumber("Counter", counter++);
   }
}
```

C++

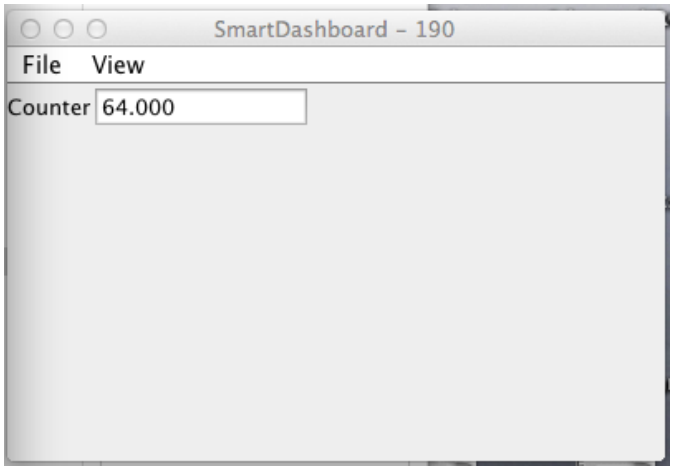
```
#include "Robot.h"
float counter = 0.0;

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)
```

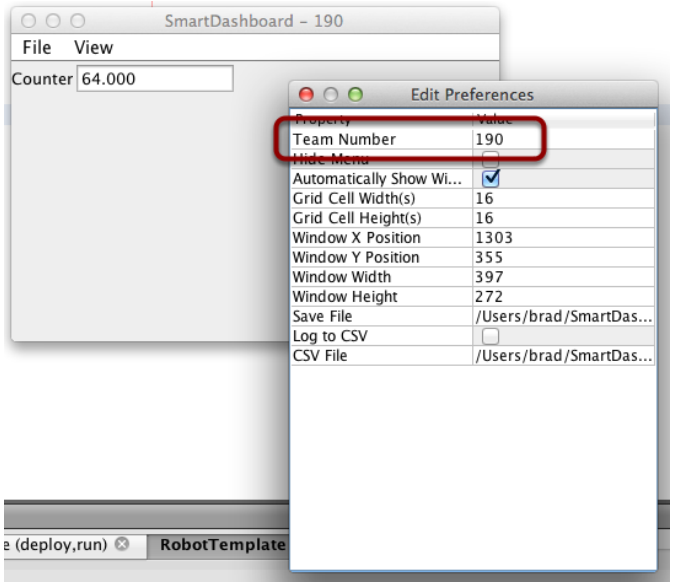
Este es un programa de robot mínimo que escribe un valor en el SmartDashboard. Simplemente incrementa un contador 50 veces por segundo para verificar que la conexión está funcionando. Sin embargo, para minimizar el uso de ancho de banda, NetworkTables de forma predeterminada acelerará las actualizaciones a 10 veces por segundo.

Salida de SmartDashboard para el programa de muestra



La pantalla SmartDashboard debería verse así después de aproximadamente 6 segundos de que el robot se haya habilitado en el modo Teleop. Si no es así, debe verificar que la conexión esté configurada correctamente.

Verificación de la dirección IP en SmartDashboard

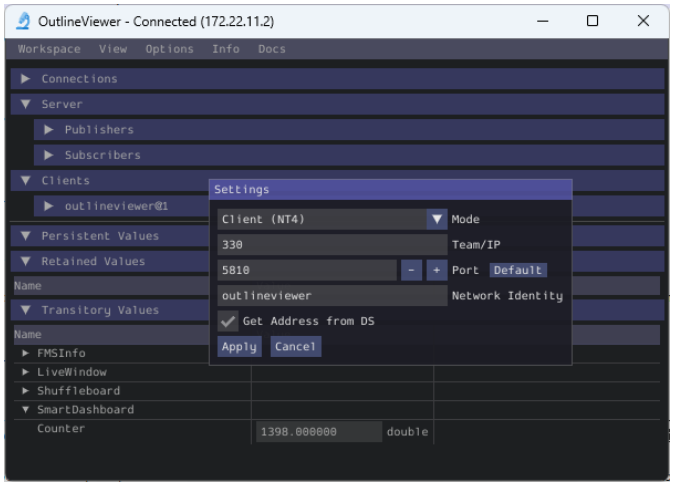


Si la pantalla del valor no aparece, verifique que el número de equipo esté configurado correctamente como se muestra en esta imagen. El cuadro de diálogo de preferencias se puede ver seleccionando `Archivo` y luego `` Preferencias``.

Verificación del programa mediante 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.



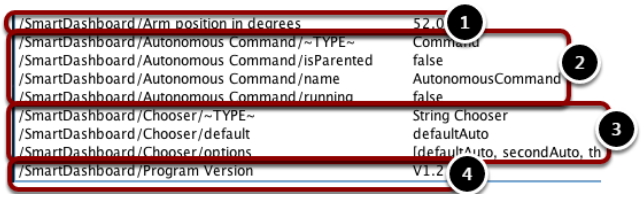
11.3.8 Espacio de nombres para SmartDashboard

SmartDashboard usa NetworkTables para enviar datos entre el robot y la computadora Dashboard (Driver Station). NetworkTables envía datos como nombre, pares de valores, como una matriz asociativa/tabla hash distribuida entre el robot y la computadora. Cuando se cambia un valor en un lugar, su valor se actualiza automáticamente en el otro lugar. Este mecanismo y un conjunto estándar de nombres (claves) es cómo se muestran los datos en el SmartDashboard.

Existe una estructura jerárquica en el espacio de nombres que crea un conjunto de tablas y subtablas. Los datos de SmartDashboard están en la subtabla SmartDashboard y los datos de LiveWindow están en la subtabla de LiveWindow como se muestra a continuación.

A efectos informativos, los nombres y valores pueden mostrarse utilizando la aplicación OutlineViewer que se instala en la misma ubicación que el SmartDashboard. Esta aplicación mostrará todas las claves y valores de NetworkTables a medida que se vayan actualizando.

Valores de datos de SmartDashboard

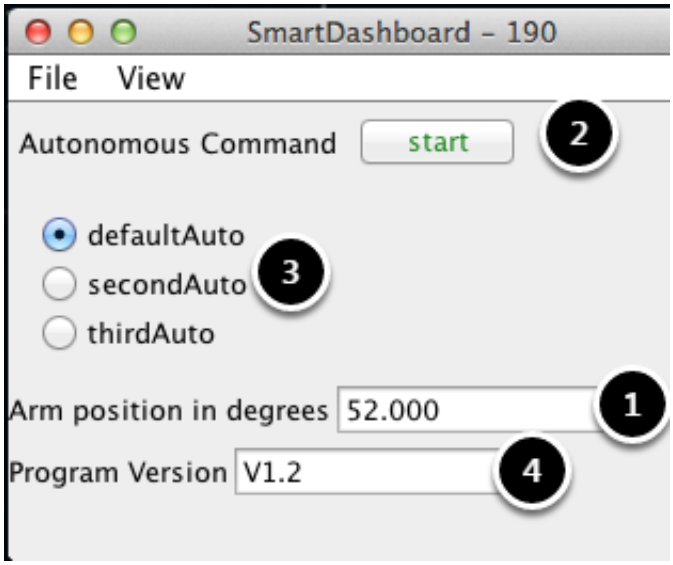


Los valores de SmartDashboard se crean con nombres clave que comienzan con `SmartDashboard/`. Los valores anteriores vistos con OutlineViewer corresponden a los datos colocados en el SmartDashboard con las siguientes declaraciones:

```
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");
```

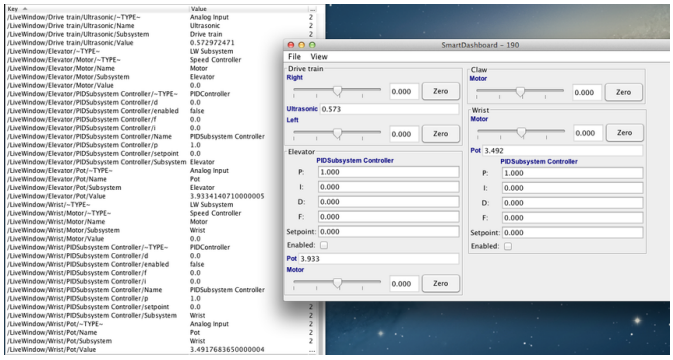
La `Posición del brazo` se crea con la llamada `putNumber()`. El `AutonomousCommand` está escrito con un `putData("Autonomous Command", command)` que no se muestra en el fragmento de código anterior. El selector se crea como un objeto `SendableChooser` y el valor de cadena, `Versión del programa` se crea con la llamada `putString()`.

Vista de SmartDashboard



El código del paso anterior genera los valores de la tabla como se muestra y la pantalla del SmartDashboard como se muestra aquí. Los números corresponden a las variables de NetworkTables mostradas en el paso anterior.

Valores de datos de LiveWindow



Los datos de LiveWindow se agrupan automáticamente por subsistema. Los datos se pueden ver en el SmartDashboard cuando el robot está en modo de prueba (configurado en la Driver Station). Si no está escribiendo un programa basado en comandos, aún puede hacer que los sensores y actuadores se agrupen para facilitar la visualización especificando el nombre del subsistema. En la pantalla anterior, puede ver los nombres de las teclas y la salida resultante en el modo de prueba en el SmartDashboard. Todas las cadenas comienzan con ``/LiveWindow ``, luego el nombre del subsistema, luego un grupo de valores que se utilizan para mostrar cada elemento. El código que genera esta pantalla LiveWindow se muestra a continuación:

```
drivetrainLeft = new PWMVictorSPX(1);
drivetrainLeft.setName("Drive train", "Left");

drivetrainRight = new PWMVictorSPX(1);
drivetrainRight.setName("Drive train", "Right";
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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");
```

Los valores que corresponden a los actuadores no solo se muestran, sino que se pueden configurar mediante los controles deslizantes creados en el SmartDashboard en el modo de prueba.

11.3.9 SmartDashboard: Modo Test y Live Window

Mostrar valores de LiveWindow

LiveWindow will automatically add your sensors and actuators for you. There is no need to do it manually. LiveWindow values may also be displayed by writing the code yourself and adding it to your robot program. This allows you to customize the names and group them in subsystems. This is a convenient method of displaying whether they are actual command based program subsystems or just a grouping that you decide to use in your program.

Agregar el código necesario a su programa

For each sensor or actuator that is created, set the subsystem name and display name by calling `setName` (`SetName` in C++). When the SmartDashboard is put into LiveWindow mode, it will display the sensors and actuators.

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);
```

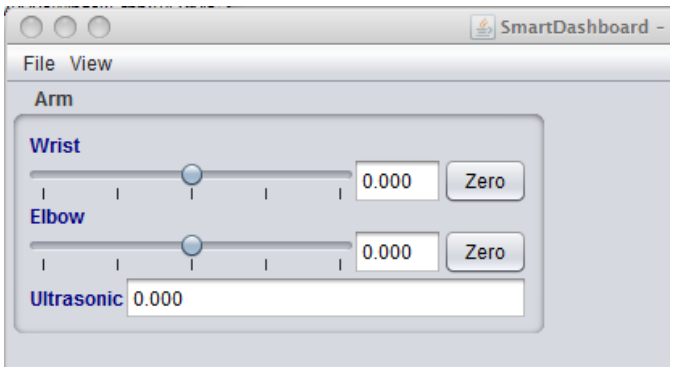
C++

```
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)
```

Visualizar el display en la SmartDashboard



Los sensores y actuadores añadidos a LiveWindow se mostrarán agrupados por subsistema. El nombre del subsistema es sólo una agrupación arbitraria para organizar la visualización de los sensores. Los actuadores también pueden operarse, operando el intensificador para los dos controladores del motor.

Habilitar el modo de prueba (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 *Enabling Test Mode* 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.

Importante: 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

```
@Override
public void robotInit() {
   enableLiveWindowInTest(true);
}
```

C++

```
void Robot::RobotInit() {
   EnableLiveWindowInTest(true);
}
```

PYTHON

```
def robotInit(self) -> None:
   enableLiveWindowInTest(true)
```

Modo de prueba explícito vs implícito

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++

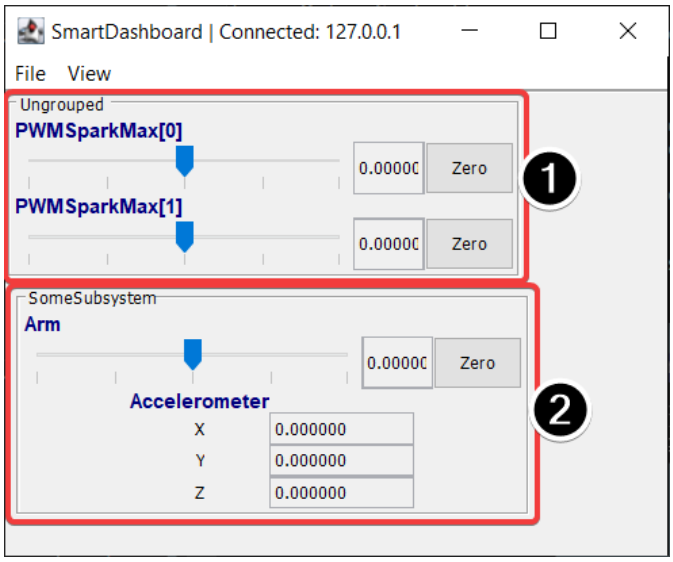
```
frc::PWMSparkMax leftDrive{0};
frc::PWMSparkMax rigthDrive{1};
frc::BuiltInAccelerometer accel{};
frc::PWMVictorSPX arm{3};

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")
```

All sensors and actuators will automatically be displayed on the SmartDashboard in test mode and will be named using the object type (such as PWMSparkMax, PWMVictorSPX, BuiltInAccelerometer, etc.) with channel number with which the object was created. In addition, the program can explicitly add sensors and actuators to the test mode display, in which case programmer-defined subsystem and object names can be specified making the program clearer. This example illustrates explicitly defining those sensors and actuators.

Entender qué se muestra en el modo de prueba



This is the output in the SmartDashboard display when the robot is placed into test mode. In the display shown above the objects listed as Ungrouped were implicitly created by WPILib when the corresponding objects were created. These objects are contained in a subsystem group called «Ungrouped» **(1)** and are named with the device type (PWMSparkMax in this case), and the channel numbers. The objects shown in the «SomeSubsystem» **(2)** group are explicitly created by the programmer from the code example in the previous section. These are named in the calls to `SendableRegistry.setName()`. Explicitly created sensors and actuators will be grouped by the specified subsystem.

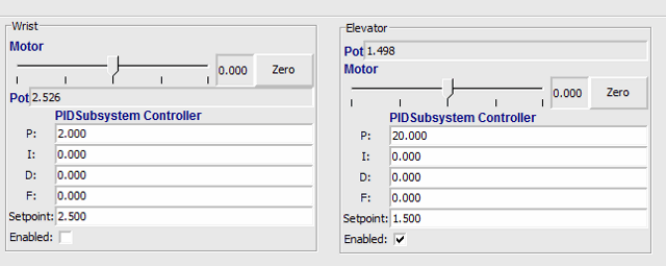
Sintonizar un PID con SmartDashboard

Un PID (Proporcional, Integral, Diferencial) es un algoritmo para determinar la velocidad de un motor basado en la retroalimentación de sensores, para alcanzar el punto de control lo más rápido posible. Por ejemplo, un robot con un elevador que se mueve a una determinada posición debería moverse lo más rápido posible y luego detenerse sin excederse, para no generar oscilaciones. Conseguir que el controlador PID se comporte de esta forma se llama “sintonizar”. La idea es calcular un valor de error que sea la diferencia entre el valor actual del elemento del mecanismo de retroalimentación y el valor deseado (punto de control). En caso de ser un brazo, puede que exista un potenciómetro conectado al canal análogo que provea de un voltaje proporcional a la posición del brazo. El valor deseado es el voltaje predeterminado a la posición a la que se moverá el brazo, y el valor actual es el voltaje para la posición actual del brazo.

Encontrar los valores de los puntos de control con LiveWindow

Cree un subsistema PID por cada mecanismo con retroalimentación. Los subsistemas PID contienen el actuador (motor) y el sensor de retroalimentación (potenciómetro, en este caso). Puede usar el modo de prueba para visualizar los sensores y actuadores del subsistema. Utilice manualmente el intensificador para ajustar el actuador a la posición que desee. Note que los valores del sensor (2) para cada posición deseada. Estos valores se convertirán en los puntos de control de los controladores PID.

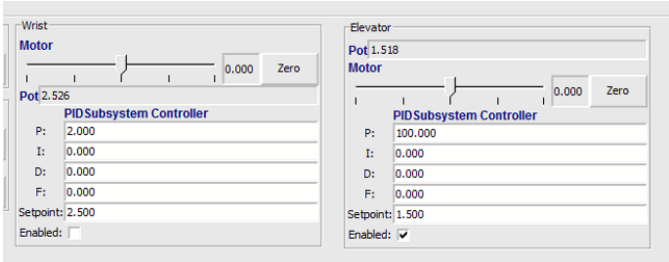
Visualizar el controlador PID en LiveWindow



En el modo de prueba, los subsistemas PID se muestran con sus parámetros P, I, y D por separado. Los valores P, I y D son los pesos aplicados al error calculado (P), la suma de los errores en razón al tiempo (I), y la tasa de cambio de los errores (D). Cada uno de estos términos es multiplicado por los pesos y se añaden juntos para formar el valor del motor. Escoger los valores óptimos para P, I y D puede ser una tarea difícil y requiere cierta experiencia. El modo de prueba permite modificar estos valores, y observar el mecanismo de respuesta.

Importante: 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.

Sintonizar el controlador 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.

Una vez que ha determinado los valores de P, I y D estos pueden añadirse a su código. Puede encontrarlos en las propiedades de sus subsistemas PID en RobotBuilder o en el constructor del subsistema PID dentro de su código.

El término F (avance) es utilizado para controlar la velocidad con el controlador PID.

Para más información, consulte *Control PID en WPILib*.

11.4 Glass

Glass es una nueva dashboard de visualización de datos de robots y cuadros de mando. Su interfaz gráfica de usuario es muy similar a la de *Simulation GUI*. En su estado actual, está pensada para ser utilizada como una herramienta del programador más que como una dashboard en un entorno de competición.

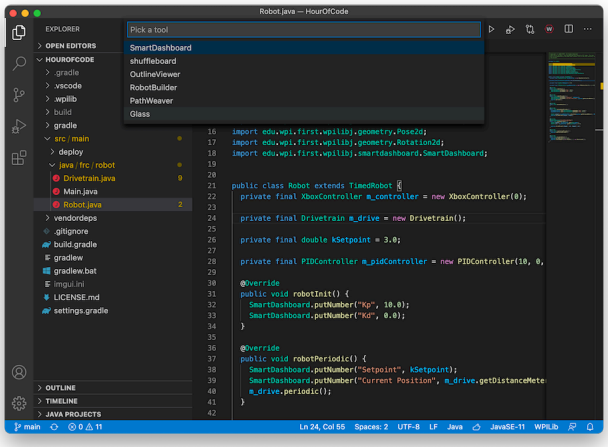
Nota: Glass will not be available within the list of dashboards in the NI Driver Station.

11.4.1 Introducción a Glass

Glass es un nuevo dashboard y herramienta de visualización de datos del robot. Es compatible con muchos de los mismos *widgets* que admite la Simulación GUI, incluyendo visualización de pose de robot y trazado avanzado. En su estado actual, está destinado a ser utilizado como una herramienta de programación para la depuración y no como un dashboard para uso en competencia.

Apertura a Glass

Glass se puede iniciar seleccionando el menú de puntos suspensivos (...) en el código Visual Studio, haciendo clic en *Start Tool* y luego seleccionando *Glass*.

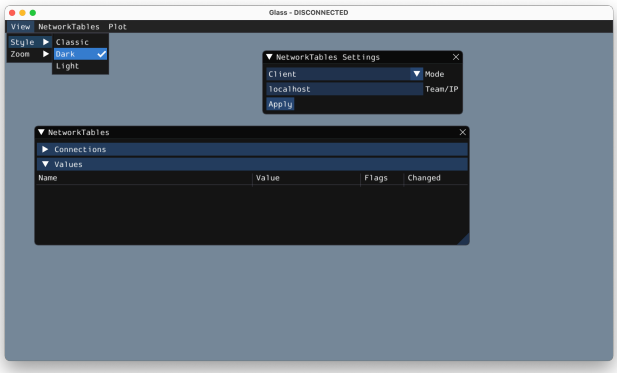


Nota: You can also launch Glass directly by navigating to `~/wpilib/YYYY/tools` and running `Glass.py` (Linux and macOS) or by using the shortcut inside the WPILib Tools desktop folder (Windows).

Cambiar la configuración de la vista

El elemento *View* del menú contiene configuraciones de *Zoom* y *Style* que se pueden personalizar. La opción de *Zoom* dicta el tamaño del texto en la aplicación mientras que la opción de *Style* le permite seleccionar entre los modos `Clásico`, `Claro`, y `Oscuro`.

Un ejemplo sobre la configuración en estilo `Oscuro` esta abajo:



Borrar datos de la aplicación

Los datos de la aplicación para Glass, incluidos los tamaños y posiciones de los widgets, así como otra información personalizada para los widgets, se almacenan en un archivo `glass.ini`. La ubicación de este archivo varía según su sistema operativo:

- En Windows, la configuración del archivo se encuentra en `%APPDATA%`.
- En macOS, la configuración del archivo se encuentra en `~/Library/Preferences`.
- En Linux, la configuración del archivo se encuentra en `$XDG_CONFIG_HOME` o `~/.config` si la primera no existe.

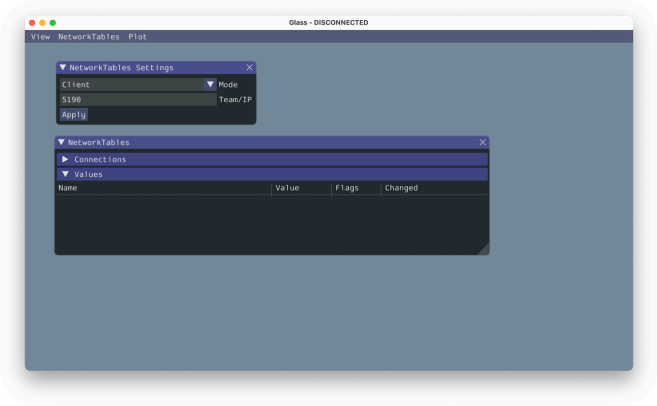
El archivo de configuración `` glass.ini`` simplemente se puede eliminar para restaurar Glass a una «pizarra limpia».

11.4.2 Establecimiento de conexiones de tablas de red

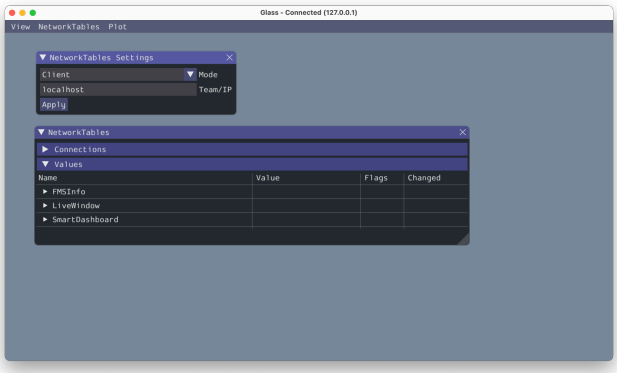
Glass usa *NetworkTables* protocolo para establecer una conexión con su programa de robot. También se utiliza para transmitir y recibir datos desde y hacia el robot.

Conexión a un robot

Cuando se inicie Glass por primera vez, verá dos widgets *NetworkTables Settings* y *NetworkTables*. Para conectarse a un robot, seleccione *Cliente* en *Modo* en el widget *Configuración de tablas de red*, ingrese el número de su equipo y haga clic en *Aplicar*.



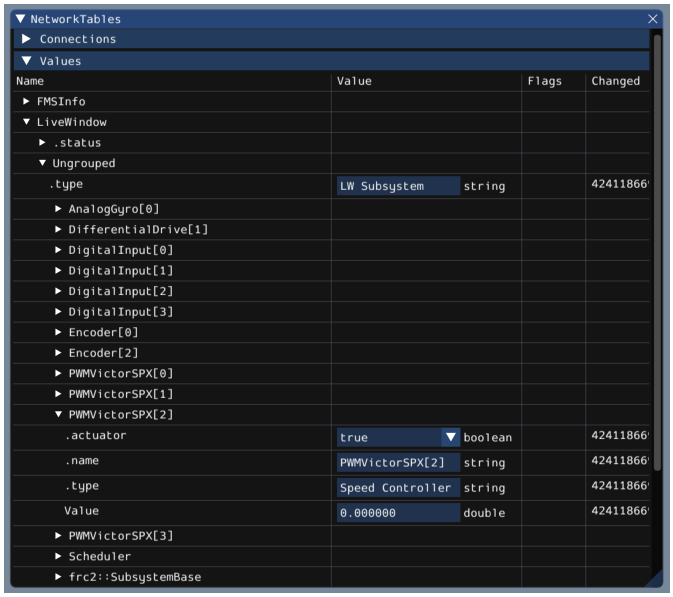
También puede conectarse a un robot que se está ejecutando en simulación en su computadora (incluidos los robots Romi) escribiendo `localhost` en el cuadro *Equipo / IP*.



Importante: El estado de conexión de NetworkTables siempre está visible en la barra de título de la aplicación Glass.

Visualización de entradas de NetworkTables

El widget *NetworkTables* se puede utilizar para ver todas las entradas que se envían a través de NetworkTables. Estas entradas están ordenadas jerárquicamente por tabla principal, subtabla, etc.



Además, puede ver todos los clientes NetworkTables conectados en el panel *Conexiones* del widget.

11.4.3 Widgets de Glass

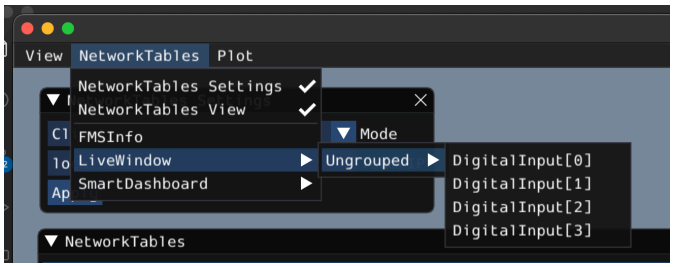
Specialized widgets are available for certain types that exist in robot code. These include objects that are manually sent over NetworkTables such as `SendableChooser` instances, or hardware that is automatically sent over *LiveWindow*.

Nota: El soporte de widgets en Glass está todavía en su infancia, por lo que sólo hay un puñado de widgets disponibles. Esta lista crecerá a medida que el trabajo de desarrollo continúe.

Nota: Un widget puede ser renombrado haciendo clic con el botón derecho del ratón en su cabecera y especificando un nuevo nombre.

Widgets del hardware

Widgets de hardware en especifico (como controladores de motor) están disponibles usualmente por medio de LiveWindow. Se puede acceder a estos seleccionando la opción *NetworkTables* en el menú, haciendo clic en *LiveWindow* y escogiendo el widget deseado.



La lista de hardware (enviada por LiveWindow automáticamente) que tiene widgets está en la parte de abajo:

- «Entrada digital»
- «Salida digital»
- `SpeedController`
- `Gyro`

Aquí está un ejemplo del widget para giroscopios:

Widget de elección enviable

El widget *Sendable Chooser*` representa una instancia `SendableChooser` del código del robot. A menudo se utiliza para seleccionar modos autónomos. Como otros cuadros de mando, tu instancia «SendableChooser» simplemente necesita ser enviada usando una API de NetworkTables. Lo más simple es usar algo como `SmartDashboard`:

JAVA

```
SmartDashboard.putData("Auto Selector", m_selector);
```

C++

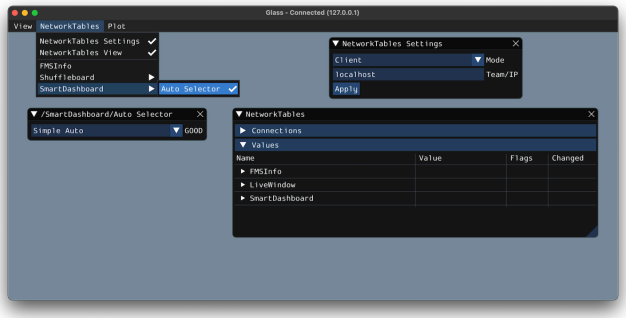
```
frc::SmartDashboard::PutData("Auto Selector", &m_selector);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData("Auto Selector", selector)
```

Nota: For more information on creating a `SendableChooser`, please see *this document*.

El widget *Seleccionador de envío* aparecerá en el menú *Tablas de red* y bajo el nombre de la tabla principal a la que se envió la instancia. En el ejemplo anterior, el nombre de la tabla principal sería *SmartDashboard*.



Widget de controlador PID

El widget *PID Controller* le permite ajustar rápidamente los valores del PID para un determinado controlador. Una instancia de `PIDController` debe ser enviada utilizando una API de Tablas de Red. Lo más sencillo es utilizar algo como `SmartDashboard`:

JAVA

```
SmartDashboard.putData("Elevator PID Controller", m_elevatorPIDController);
```

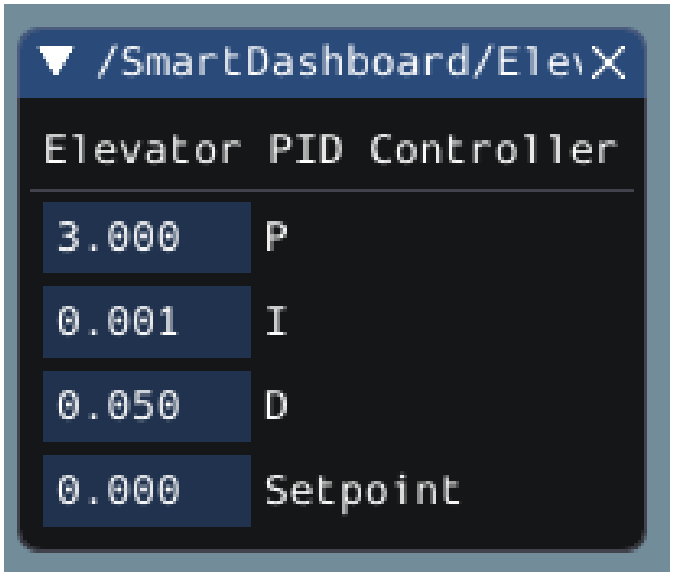
C++

```
frc::SmartDashboard::PutData("Elevator PID Controller", &m_elevatorPIDController);
```

PYTHON

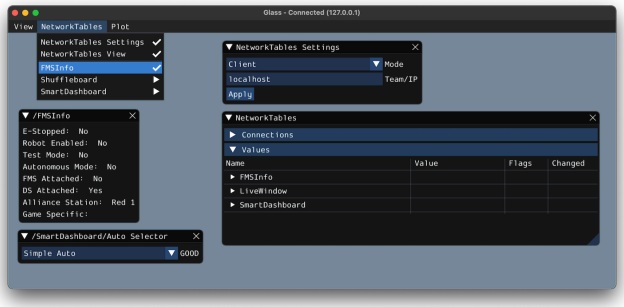
```
from wpilib import SmartDashboard

SmartDashboard.putData("Elevator PID Controller", elevatorPIDController)
```

Esto le permite ajustar rápidamente los valores P, I y D para varios puntos de ajuste.



FMSInfo Widget

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 para el marco de trabajo basado en comandos

Glass también tiene varios widgets que son específicos para el *command-based framework*. Estos incluyen widgets para programar comandos, ver los comandos que se están ejecutando activamente en un subsistema específico, o ver el estado del *command scheduler*.

Widget de selección de comandos

La *Command Selector* le permite iniciar y cancelar una instancia específica de un comando (enviado a través de NetworkTables) desde Glass. Por ejemplo, puede crear una instancia de `MyCommand` y enviarla a SmartDashboard:

JAVA

```
MyCommand command = new MyCommand(...);
SmartDashboard.putData("My Command", command);
```

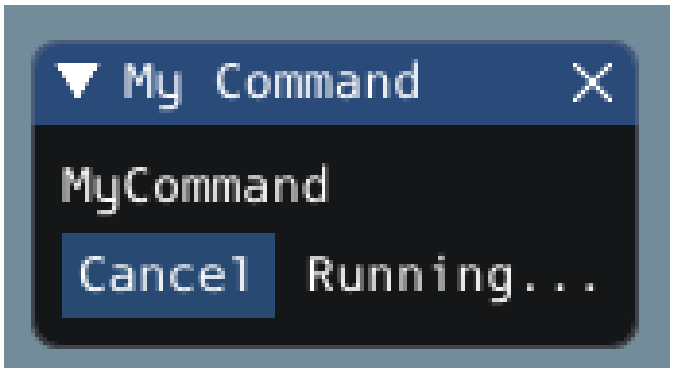
C++

```
#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)
```

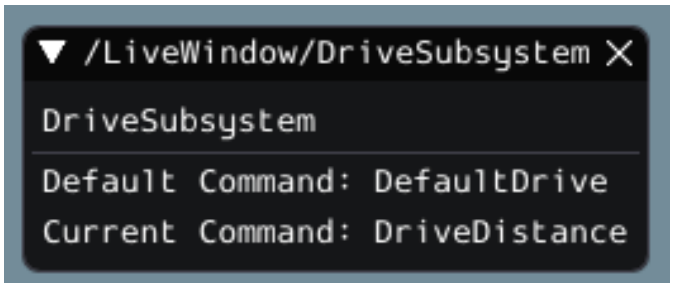
Nota: The `MyCommand` instance can also be sent using a lower-level NetworkTables API or using the *Shuffleboard API*. In this case, the `SmartDashboard` API was used, meaning that the *Command Selector* widget will appear under the `SmartDashboard` table name.



El widget tiene dos estados. Cuando el comando no se está ejecutando, aparecerá el botón: guilabel: *Ejecutar*; al hacer clic en él, se programará el comando. Cuando el comando se está ejecutando, aparecerá un botón: guilabel: *Cancelar*, acompañado del texto: guilabel:` Ejecutando ... `(como se muestra arriba). Esto cancelará el comando.

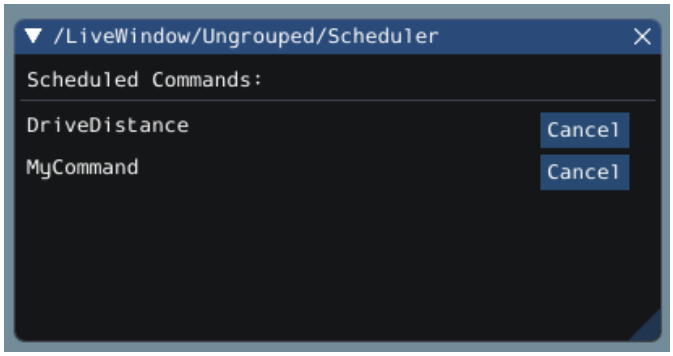
Widget de subsistema

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 del programador de comandos

El widget *Programador de comandos* permite ver todos los comandos programados actualmente. Además, cualquiera de estos comandos puede ser cancelado desde la GUI.



La instancia de `CommandScheduler` se envía automáticamente a NetworkTables a través de LiveWindow. Para ver este widget, busque debajo del nombre de la tabla principal *LiveWindow*

en el menú *NetworkTables*.

11.4.5 El widget Field2d

Glass permite mostrar la posición de su robot en el campo usando el widget *Field2d*. Debe crearse una instancia de la clase `Field2d`, enviarse a través de NetworkTables y actualizarse periódicamente con la última pose de robot en su código de robot.

Enviando la pose del robot desde el código de usuario

Para enviar la posición de tu robot (normalmente obtenida por *odometría* o un estimador de poses), se debe crear una instancia de `Campo2d` en código de robot y enviarla a través de NetworkTables. La instancia debe ser actualizada periódicamente con la última pose del robot.

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++

```
#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())
```

Nota: The `Field2d` instance can also be sent using a lower-level NetworkTables API or using the *Shuffleboard API*. In this case, the `SmartDashboard` API was used, meaning that the *Field2d* widget will appear under the `SmartDashboard` table name.

Sending Trajectories to Field2d

Visualizing your trajectory is a great debugging step for verifying that your trajectories are created as intended. Trajectories can be easily visualized in *Field2d* using the `setTrajectory()`/`SetTrajectory()` functions.

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++

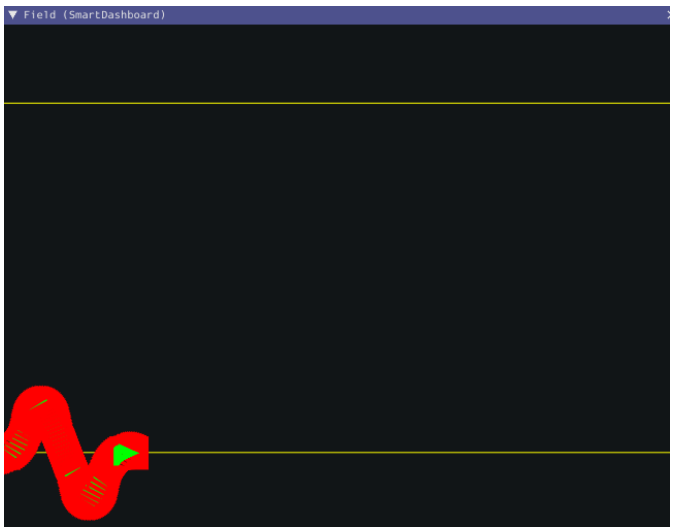
```
18  void AutonomousInit() override {
19    // Start the timer.
20    m_timer.Start();
21
22    // Send Field2d to SmartDashboard.
23    frc::SmartDashboard::PutData(&m_field);
24
25    // Reset the drivetrain's odometry to the starting pose of the trajectory.
26    m_drive.ResetOdometry(m_trajectory.InitialPose());
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)
```

Viewing Trajectories with Glass

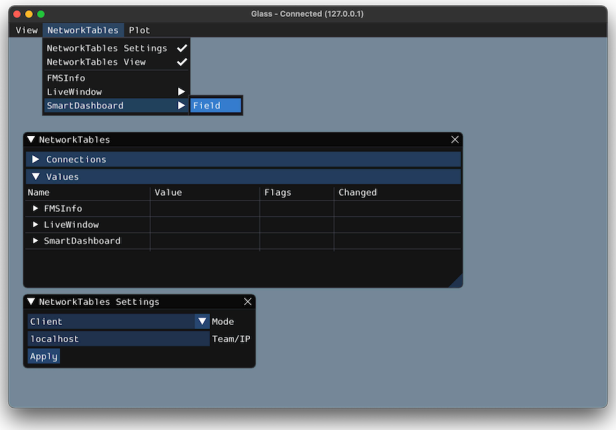
The sent trajectory can be viewed with *Glass* through the dropdown *NetworkTables -> SmartDashboard -> Field2d*.



Nota: The above example which uses the RamseteController (Java / C++ / Python) will not show the sent trajectory until autonomous is enabled at least once.

Ver la postura del robot en glass

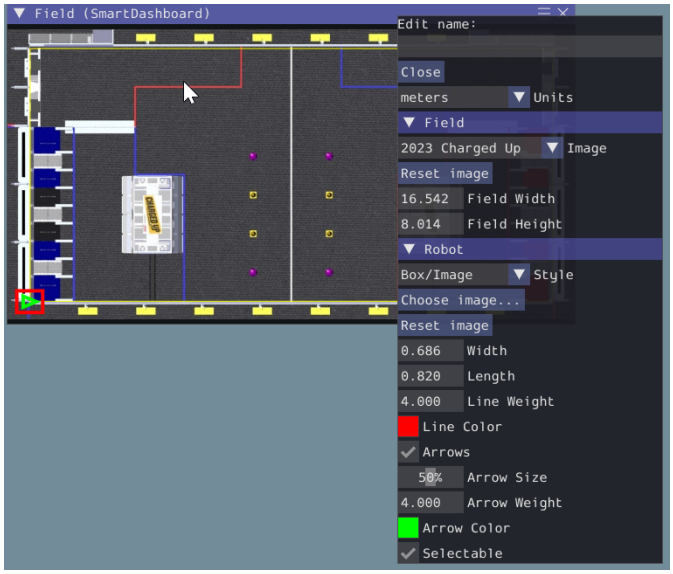
Después de enviar la instancia `Field2d` a través de NetworkTables, el widget *Field2d* se puede agregar a Glass seleccionando *NetworkTables* en la barra de menú, eligiendo el nombre de la tabla a la que se envió la instancia, y luego haciendo clic en el botón *Campo*.



Una vez que aparece el widget, puede cambiar su tamaño y colocarlo en el espacio de trabajo de Glass como desee. Al hacer clic con el botón derecho en la parte superior del widget, podrá personalizar el nombre del widget, seleccionar una imagen de campo personalizada, seleccionar una imagen de robot personalizada y elegir las dimensiones del campo y el robot.

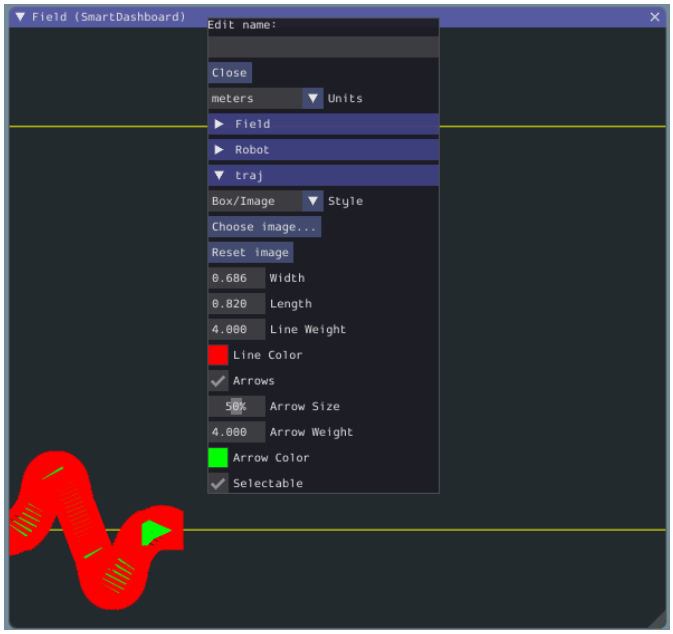
You can choose from an existing field layout using the *Image* drop-down. Or you can select a custom file by setting the *Image* to `Custom` and selecting *Choose image...*. You can choose to either select an image file or a PathWeaver JSON file as long as the image file is in the same directory. Choosing the JSON file will automatically import the correct location of the field in the image and the correct size of the field.

Nota: 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*.

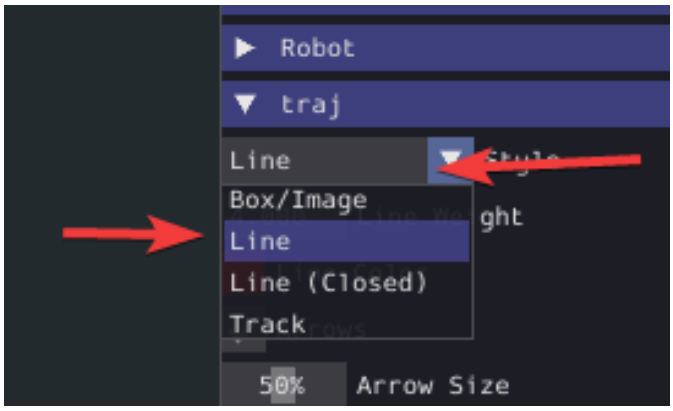


Modifying Pose Style

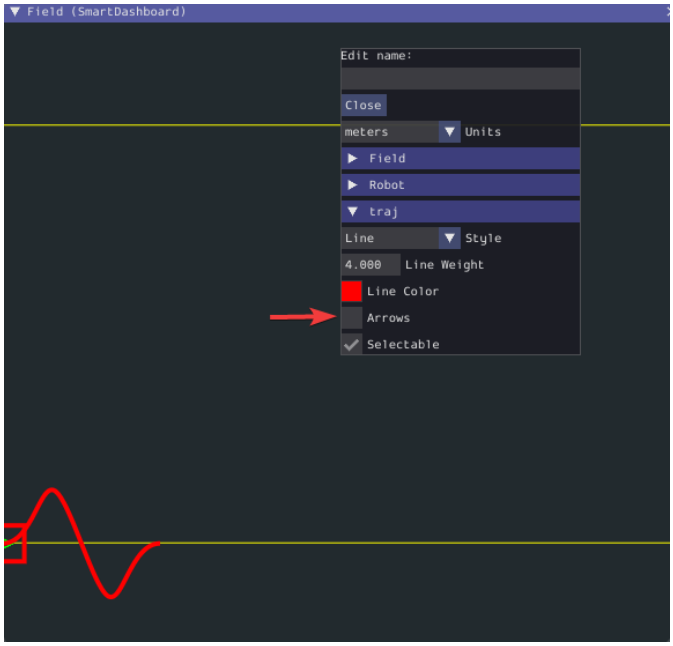
Poses can be customized in a plethora of ways by right clicking on the Field2d menu bar. Examples of customization are: line width, line weight, style, arrow width, arrow weight, color, etc.



One usage of customizing the pose style is converting the previously shown `traj` pose object to a line, rather than a list of poses. Click on the *Style* dropdown box and select *Line*. You should notice an immediate change in how the trajectory looks.

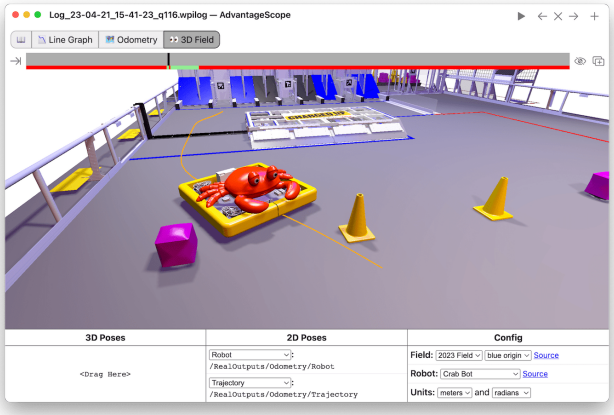


Now, uncheck the *Arrows* checkbox. This will cause our trajectory to look like a nice and fluid line!



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 The Mechanism2d Widget

Glass supports displaying stick-figure representations of your robot's mechanisms using the *Mechanism2d* widget. It supports combinations of ligaments that can rotate and / or extend or retract, such as arms and elevators and they can be combined for more complicated mechanisms. An instance of the `Mechanism2d` class should be created and populated, sent over NetworkTables, and updated periodically with the latest mechanism states in your robot code. It can also be used with the *Physics Simulation* to visualize and program your robot's mechanisms before the robot is built.

Creating and Configuring the Mechanism2d Instance

The `Mechanism2d` object is the «canvas» where the mechanism is drawn. The root node is where the mechanism is anchored to `Mechanism2d`. For a single jointed arm this would the pivot point. For an elevator, this would be where it's attached to the robot's base. To get a root node (represented by a `MechanismRoot2d` object), call `getRoot(name, x, y)` on the container `Mechanism2d` object. The name is used to name the root within NetworkTables, and should be unique, but otherwise isn't important. The `x / y` coordinate system follows the same orientation as `Field2d` - `(0,0)` is bottom left.

In the examples below, an elevator is drawn, with a rotational wrist on top of the elevator. The full Mechanism2d example is available in Java / C++

JAVA

```
43    // the main mechanism object
44    Mechanism2d mech = new Mechanism2d(3, 3);
45    // the mechanism root node
46    MechanismRoot2d root = mech.getRoot("climber", 2, 0);
```

C++

```
59  // the main mechanism object
60  frc::Mechanism2d m_mech{3, 3};
61  // the mechanism root node
62  frc::MechanismRoot2d* m_root = m_mech.GetRoot("climber", 2, 0);
```

PYTHON

```
32        # the main mechanism object
33        self.mech = wpilib.Mechanism2d(3, 3)
34        # the mechanism root node
35        self.root = self.mech.getRoot("climber", 2, 0)
```

Each `MechanismLigament2d` object represents a stage of the mechanism. It has a three required parameters, a name, an initial length to draw (relative to the size of the `Mechanism2d` object), and an initial angle to draw the ligament in degrees. Ligament angles are relative to the parent ligament, and follow math notation - the same as *Rotation2d* (counterclockwise-positive). A ligament based on the root with an angle of zero will point right. Two optional

parameters let you change the width (also relative to the size of the Mechanism2d object) and the color. Call `append()`/`Append()` on a root node or ligament node to add another node to the figure. In Java, pass a constructed `MechanismLigament2d` object to add it. In C++, pass the construction parameters in order to construct and add a 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)
        )
```

Then, publish the `Mechanism2d` object to 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)
```

Nota: The `Mechanism2d` instance can also be sent using a lower-level NetworkTables API or using the *Shuffleboard API*. In this case, the `SmartDashboard` API was used, meaning that the *Mechanism2d* widget will appear under the `SmartDashboard` table name.

To manipulate a ligament angle or length, call `setLength()` or `setAngle()` on the `MechanismLigament2d` object. When manipulating ligament length based off of sensor measurements, make sure to add the minimum length to prevent 0-length (and therefore invisible) ligaments.

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++

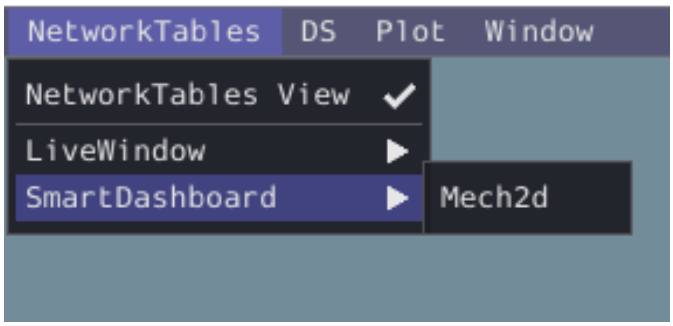
```
  void RobotPeriodic() override {
    // update the dashboard mechanism's state
    m_elevator->SetLength(kElevatorMinimumLength +
                          m_elevatorEncoder.GetDistance());
    m_wrist->SetAngle(units::degree_t{m_wristPotentiometer.Get()});
  }
```

PYTHON

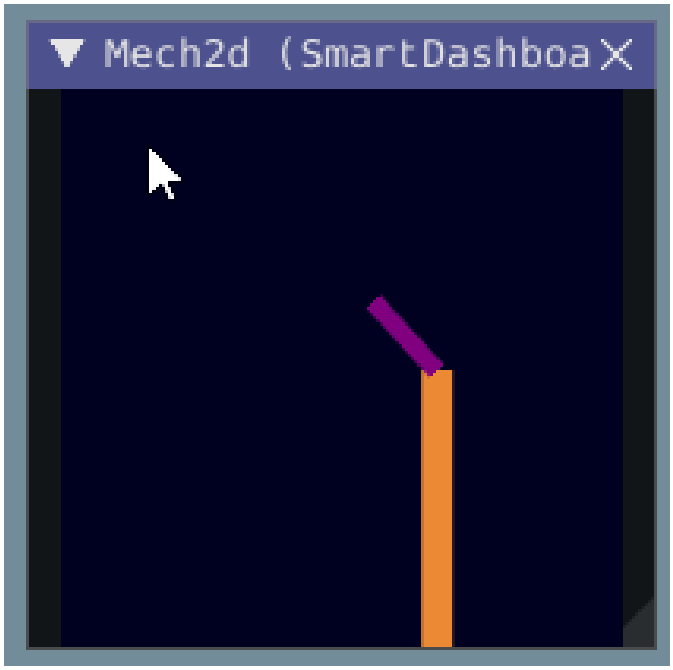
```
49      def robotPeriodic(self):
50          # update the dashboard mechanism's state
51          self.elevator.setLength(
52              self.kElevatorMinimumLength + self.elevatorEncoder.getDistance()
53          )
54          self.wrist.setAngle(self.wristPot.get())
```

Viewing the Mechanism2d in Glass

After sending the `Mechanism2d` instance over NetworkTables, the *Mechanism2d* widget can be added to Glass by selecting *NetworkTables* in the menu bar, choosing the table name that the instance was sent over, and then clicking on the *Field* button.

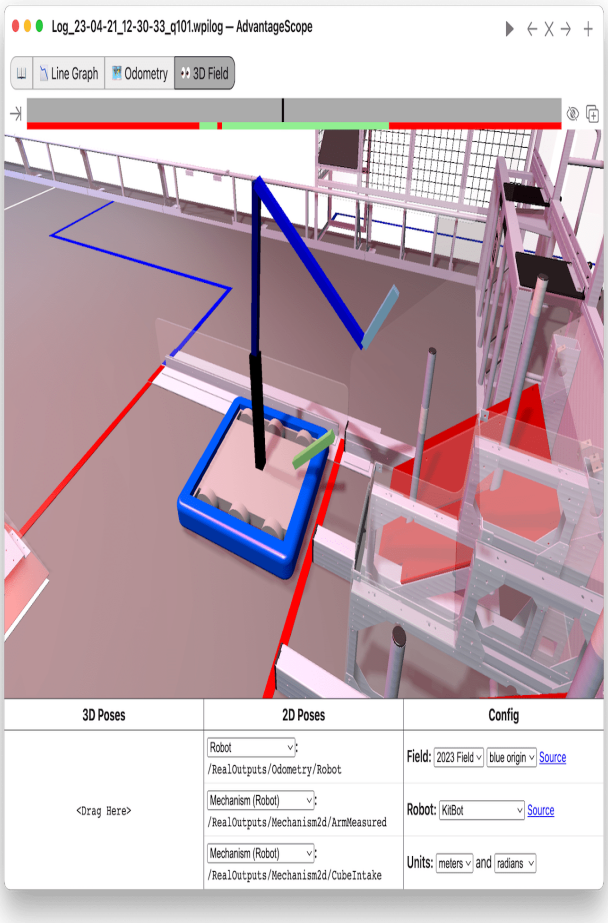


Once the widget appears as shown below, you can resize and place it on the Glass workspace as you desire. Right-clicking the top of the widget will allow you to customize the name of the widget. As the wrist potentiometer and elevator encoder changes, the mechanism will update in the widget.



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.



Próximos pasos

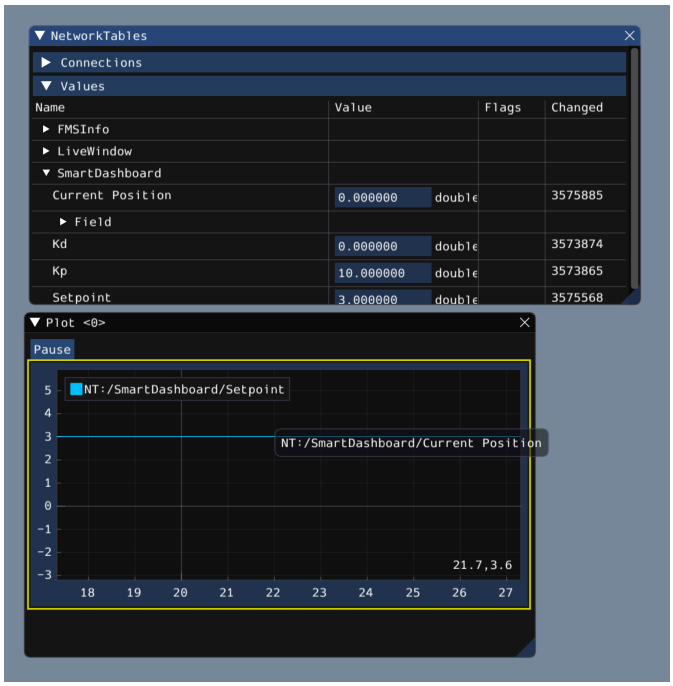
As mentioned above, the Mechanism2d visualization can be combined with *Physics Simulation* to help you program mechanisms before your robot is built. The ArmSimulation (Java / C++ / Python) and ElevatorSimulation (Java / C++ / Python) examples combine physics simulation and Mechanism2d visualization so that you can practice programming a single jointed arm and elevator without a robot.

11.4.7 Parcelas

El vidrio sobresale en el trazado de datos de alto rendimiento y exhaustivos de las Tablas de Red. Algunas características incluyen gráficos de tamaño variable, gráficos con múltiples ejes Y y la capacidad de pausar, examinar y reanudar los gráficos.

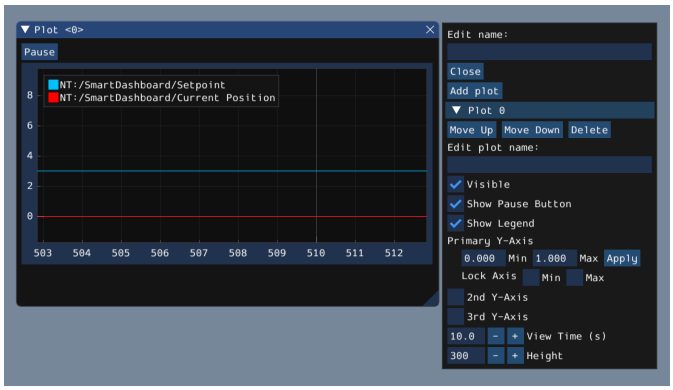
Crear una parcela

Se puede crear un nuevo widget de la parcela seleccionando el botón *Plot* en la barra del menú principal y luego haciendo clic en *New Plot Window*. Se pueden añadir varias parcelas individuales a cada ventana de la trama. Para añadir un gráfico dentro de una ventana de gráficos, haga clic en el botón *Add plot* dentro del widget. Luego puedes arrastrar varias fuentes del widget *Tablas de Red* dentro del gráfico:

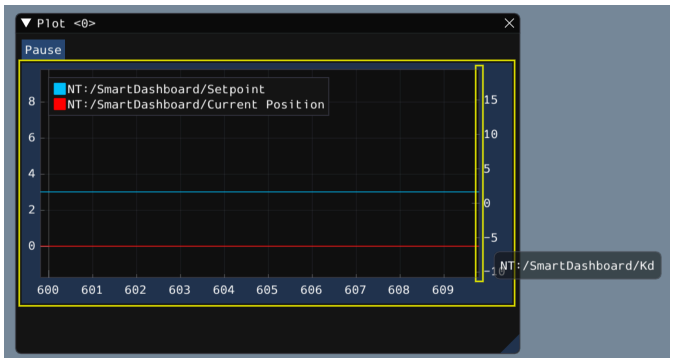


Manipulación de parcelas

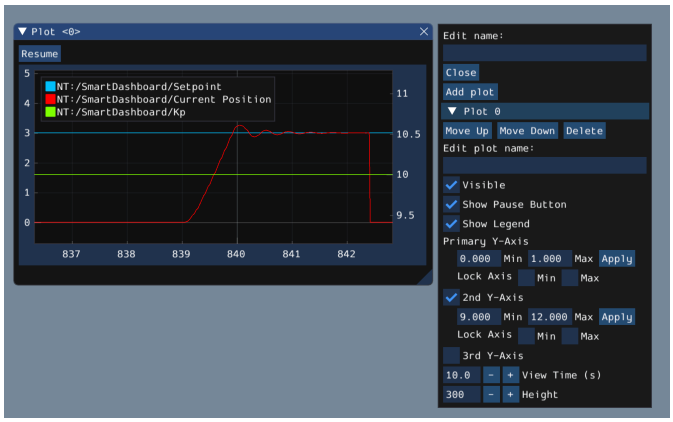
Puedes hacer clic y arrastrar en el gráfico para moverte y desplazarte por encima del gráfico para acercar y alejar los ejes y. Haciendo doble clic en el gráfico para que los límites de zoom y de los ejes se ajusten a todos los datos que se están trazando. Además, al hacer clic con el botón derecho del ratón en el gráfico, se le presentarán una gran cantidad de opciones, incluyendo si desea mostrar los ejes y secundarios y terciarios, si desea bloquear ciertos ejes, etc.



Si elige poner a disposición ejes y secundarios y terciarios, puede arrastrar las fuentes de datos a esos ejes para hacer que sus líneas se correspondan con el eje deseado:

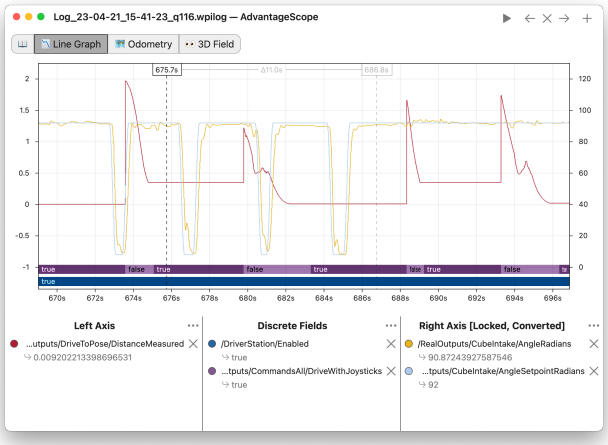


Luego, puede bloquear ciertos ejes para que su rango siempre permanezca constante, independientemente de la panorámica. En este ejemplo, el rango del eje secundario (con la entrada `/SmartDashboard/Kp`) se bloqueó entre 9 y 12.



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

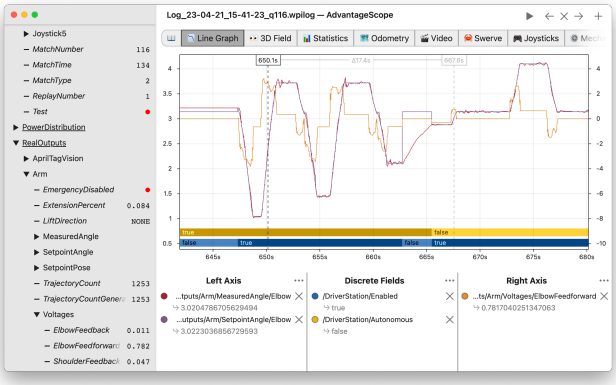
AdvantageScope is a data visualization tool for *NetworkTables*, *WPILib data logs*, and *Driver Station logs*. It is a programmer's tool (rather than a competition dashboard) and can be used to debug real or simulated robot code from a log file or live over the network.

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.

Nota: 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



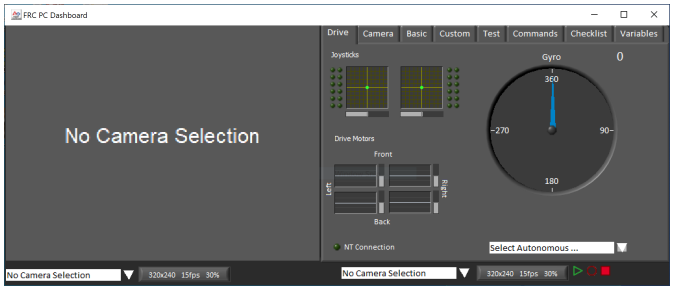
11.6 Dashboard de LabVIEW

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 de LabVIEW de FRC

La aplicación Dashboard instalada y lanzada por la Driver Station de FRC® es un programa de LabVIEW diseñado para proporcionar a los equipos información básica de su robot, con la posibilidad de ampliar y personalizar la información para adaptarla a sus necesidades. Esta aplicación Dashboard utiliza *NetworkTables* y contiene una variedad de herramientas que los equipos pueden encontrar útiles.

Dashboard de LabVIEW



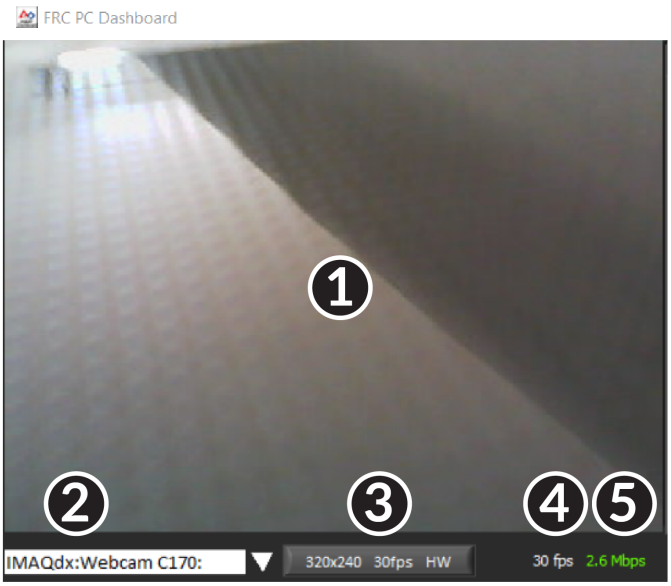
La Dashboard se divide en dos secciones principales. El panel izquierdo es para mostrar una imagen de cámara. El panel derecho contiene:

- Pestaña Drive que contiene indicadores para joystick y valores de motor de accionamiento (conectados por defecto cuando se usa con código de robot de LabVIEW), un indicador de giro, un cuadro de texto de selección autónoma, un indicador de conexión y algunos controles e indicadores para la cámara

- Pestaña básica que contiene algunos controles e indicadores predeterminados
- Pestaña de Camera que contiene una vista secundaria de la cámara, parecida a la del panel izquierdo.
- Pestaña Custom para personalizar la dashboard usando LabVIEW.
- Pestaña Test para usar con el modo prueba dentro de la estructura de LabVIEW.
- Pestaña Commands para usar con la nueva estructura LabVIEW C&C .
- Pestaña Checklist que se puede usar para crear listas de tareas para completar antes y/o entre partidos.
- Pestaña de variables que muestra las variables brutas de NetworkTables en un formato de vista de árbol

La Dashboard de LabVIEW también incluye las funciones Record/Playback, ubicadas al fondo a la derecha. Más detalles sobre esta función se incluyen más abajo en *Record/Playback*.

Imagen de la Cámara y controles



The left pane is used to display a video feed from acamera located on the robot. There are also some controls and indicators related to the camera below the tab area:

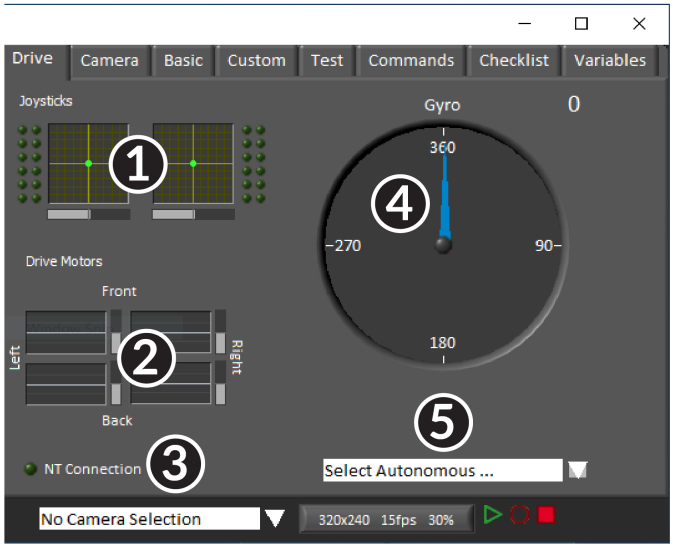
1. Transmisión de imagen de la cámara.
2. Mode Selector - This drop-down allows you to select the type of camera display to use. The choices are Camera Off, USB Camera SW (software compression), USB Camera HW (hardware compression) and IP Camera. Note that the IP Camera setting will not work when your PC is connected to the roboRIO over USB.
3. Ajustes de la cámara – Este control le permite cambiar la resolución, cuadros por segundo y compresión de la transmisión de imagen a la dashboard, de click en control para mostrar la configuración.
4. Indicador de ancho de banda – Indica aproximadamente el uso del ancho de banda de la imagen. El indicador estará verde cuando el ancho de banda sea seguro, amarillo cuando

el equipo debe tener precaución y rojo cuando el ancho de banda de la transmisión sobrepasa los niveles que funcionan en la cancha durante la competencia.

5. Framerate – Indica aproximadamente los cuadros por segundo que se reciben de la transmisión.

Truco: El indicador de ancho de banda indica el ancho de banda combinado para todas las transmisiones de cámara abiertas.

Unidad



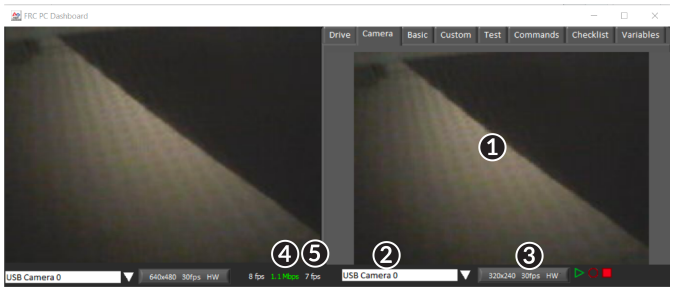
El panel del centro contiene una sección que proporciona retroalimentación de los comandos de joysticks y drive cuando se usa dentro de la estructura de LabVIEW, y una sección que muestra el estado de los Network Tables y el selector de autónomo:

1. Muestra los ejes X, Y, información del acelerador y valores de no más de 2 joysticks cuando se usa al estructura de LabVIEW.
2. Muestra valores siendo enviados a los controladores de motor al usar la estructura de LabVIEW.
3. Muestra un indicador de conexión para los datos de NetworkTables del robot
4. Muestra un valor Gyro.
5. Muestra un recuadro de texto que puede usarse para seleccionar un modo de autónomo. Cada lenguaje de un código tiene ejemplos sobre usar este recuadro para seleccionar de múltiples programas autónomos.

Estos indicadores (además de Gyro) están llenados con valores adecuados de manera predeterminada al tener la estructura de LabVIEW. Para más información en cómo usarlos con código en C++/Java vea *Using the LabVIEW Dashboard with C++, Java, or Python Code*.

Camara

Truco: El panel izquierdo solo puede mostrar una sola transmisión de cámara, entonces use la pestaña camera del panel derecho para mostrar una segunda transmisión si necesita.

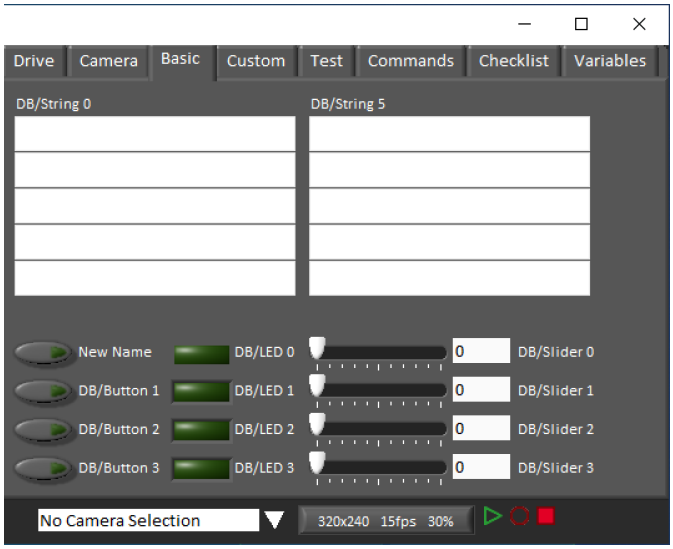


The camera tab is used to display a video feed from a camera located on the robot. There are also some controls and indicators related to the camera below the tab area:

1. Transmisión de imagen de la cámara.
2. Mode Selector - This drop-down allows you to select the type of camera display to use. The choices are Camera Off, USB Camera SW (software compression), USB Camera HW (hardware compression) and IP Camera. Note that the IP Camera setting will not work when your PC is connected to the roboRIO over USB.
3. Ajustes de la cámara – Este control le permite cambiar la resolución, cuadros por segundo y compresión de la transmisión de imagen a la dashboard, de click en control para mostrar la configuración.
4. Indicador de ancho de banda – Indica aproximadamente el uso del ancho de banda de la imagen. El indicador estará verde cuando el ancho de banda sea seguro, amarillo cuando el equipo debe tener precaución y rojo cuando el ancho de banda de la transmisión sobrepasa los niveles que funcionan en la cancha durante la competencia.
5. Framerate – Indica aproximadamente los cuadros por segundo que se reciben de la transmisión.

Truco: El indicador de ancho de banda indica el ancho de banda combinado para todas las transmisiones de cámara abiertas.

Basic

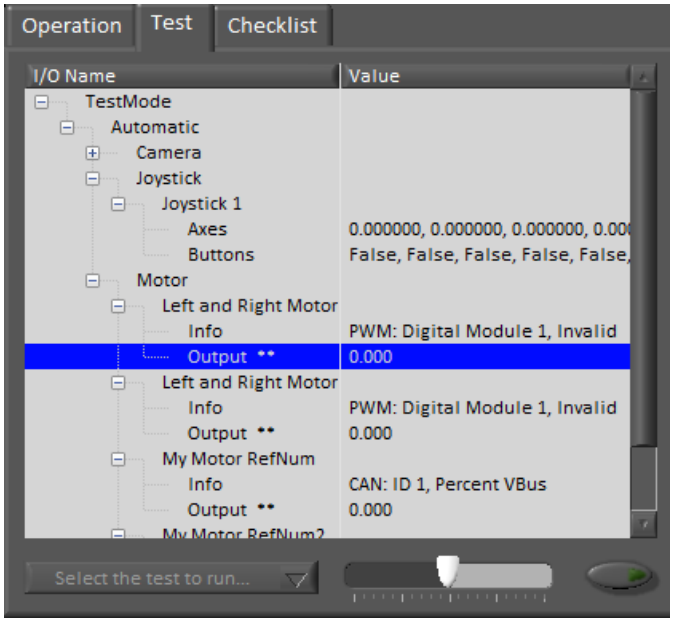


La pestaña Basic contiene una variedad de controles/indicadores pre- llenados y bidireccionales que pueden usarse para controlar el robot o mostrar información de este. Los nombres clave de la SmartDashboard asociados con cada objeto son etiquetados junto al indicador, excepto los strings que siguen el mismo patrón de nombrado e incrementan de DB/String 0 a DB/String 4 en la izquierda y DB/String 5 a DB/String 9 en la derecha. La estructura de LabVIEW contiene un ejemplo de leer desde los botones y deslizadores en modo Teleop. También contiene un ejemplo sobre personalizar las etiquetas en Begin. Para mas detalles sobre como usar esta pestaña con código en C++/Java, vea *Using the LabVIEW Dashboard with C++, Java, or Python Code*.

Personalizados

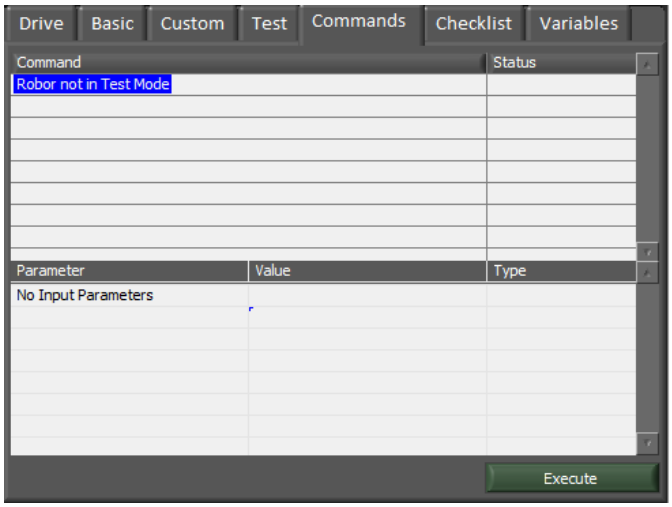
La pestaña Personalizada le permite añadir controles/indicadores adicionales al dashboard de mandos usando LabVIEW sin eliminar ninguna funcionalidad existente. Para personalizar esta pestaña necesitarás crear un proyecto de dashboard en LabVIEW.

Prueba



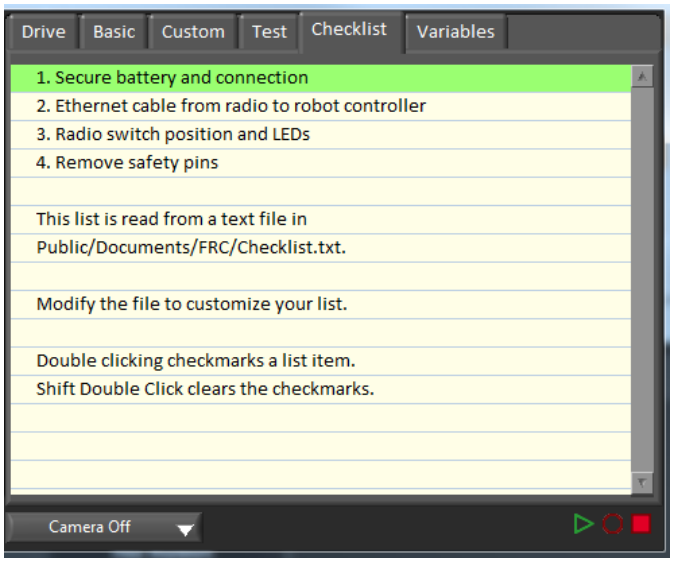
La pestaña de prueba es para usar con el modo de prueba para los equipos que usan LabVIEW (los equipos de Java y C++ deben usar SmartDashboard o Shuffleboard cuando usan el modo de prueba). Para muchos elementos de las bibliotecas, la información de entrada/salida se completará aquí automáticamente. Todos los elementos que tienen ** junto a ellos son salidas que pueden ser controladas por la dashboard. Para controlar una salida, haga clic en ella para seleccionarla, arrastre el control deslizante para fijar el valor y luego mantenga pulsado el botón verde para activar la salida. Tan pronto como se suelte el botón verde, la salida se desactivará. Esta pestaña también se puede utilizar para ejecutar y supervisar las pruebas del robot. Un ejemplo de prueba se proporciona en el marco de LabVIEW. Al seleccionar esta prueba en el cuadro desplegable se mostrará el estado de la prueba en lugar del control deslizante y se habilitarán los controles.

Comandos



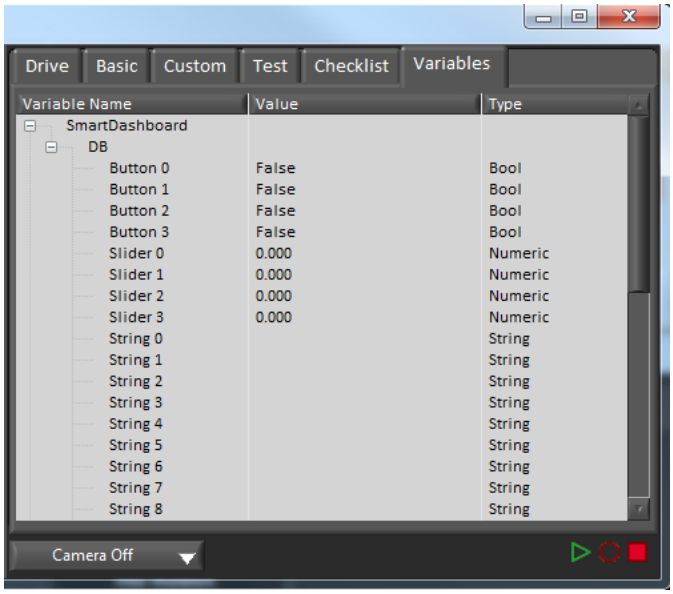
La pestaña de comandos puede utilizarse con el robot en el modo de prueba para ver qué comandos se están ejecutando y para ejecutar manualmente los comandos con fines de prueba.

Lista de control



La pestaña «Checklist» puede ser utilizada por los equipos para crear una lista de tareas a realizar antes o entre los partidos. Las instrucciones para utilizar la pestaña de la lista de comprobación están preinstaladas en el archivo de la lista de comprobación predeterminada.

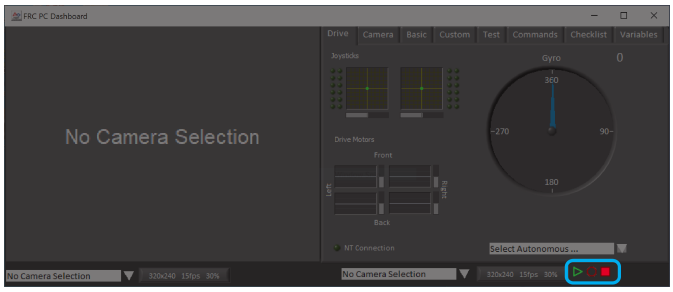
Variables



La pestaña Variables del panel izquierdo muestra todas las variables de NetworkTables en forma de árbol. El nombre de la variable (clave), el valor y el tipo de datos se muestran para cada variable. La información sobre el uso del ancho de banda de NetworkTables también

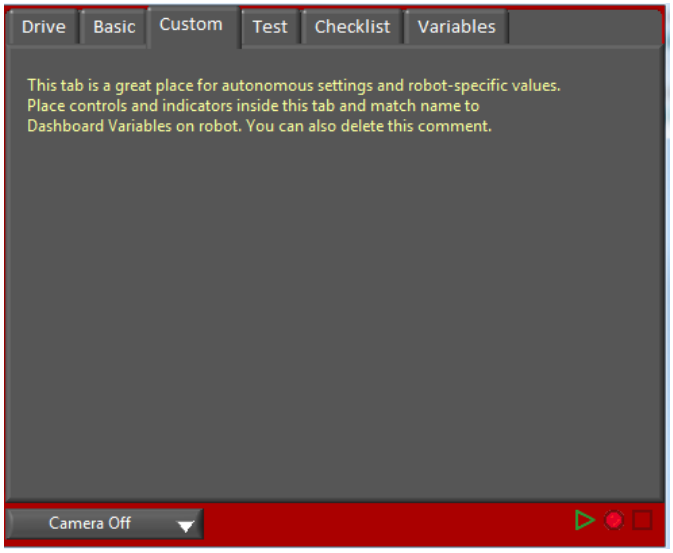
se muestra en esta pestaña. Las entradas se mostrarán con diamantes negros si no están actualmente sincronizadas con el robot.

Grabar/Reproducir



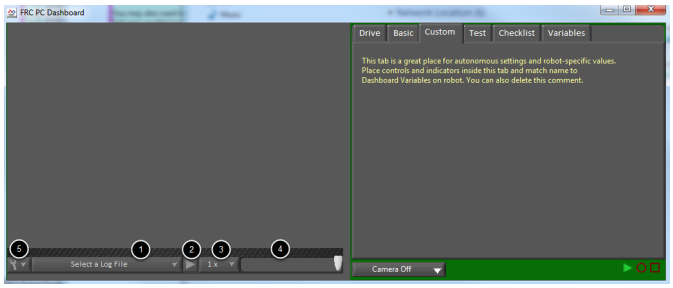
El Dashboard de LabVIEW incluye una función de grabación/reproducción que le permite grabar vídeo y datos de NetworkTables (como el estado de sus indicadores del Dashboard) y reproducirlos más tarde.

Grabación



Para iniciar a grabar, de click en el botón circular rojo. El fondo del panel derecho se volverá roja para indicar que usted esta grabando. Para parar la grabación, presione el botón cuadrado rojo.

Playback



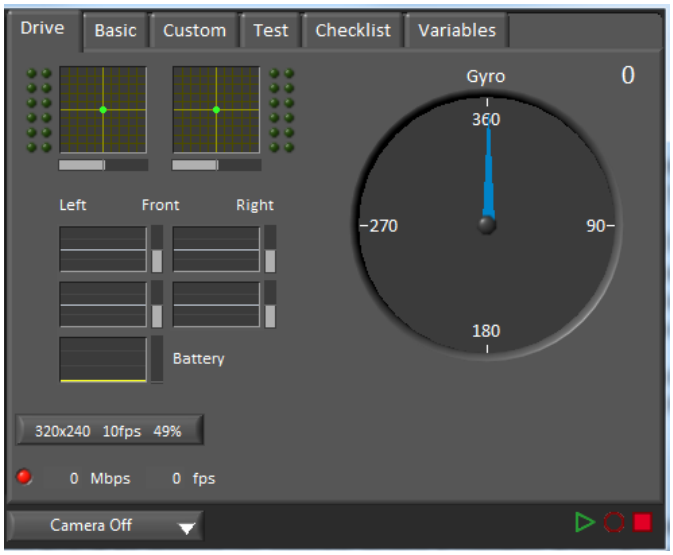
Para reproducir una grabación de nuevo, de click en el triangulo verde - botón Play. El fondo de la sección derecha empezara a mostrarse verde y los controles de la función playback aparecerán al fondo de la ventana de la cámara.

1. Selector de archivos – El menu le permite seleccionar un archivo para reproducir de nuevo. Los archivos de datos son nombrados usando la fecha y el menú también indicará la longitud del archivo. Al seleccionar un archivo este se empezará a reproducir inmediatamente.
2. Botón de pausa/reanudar - Este botón le permite pausar o reanudar la reproducción del archivo de registro.
3. Velocidad de reproducción -Este menu le permite ajustar la velocidad de la reproducción de 1/10 a 10x, la velocidad predeterminada es a tiempo real (1x).
4. Barra de tiempo – Esta barra le permite repetir o avanzar/retroceder rápidamente a través del archivo al dar click en la posición que desea o arrastrar el indicador por la línea del tiempo.
5. Configuración - Con un archivo de registro seleccionado, este desplegable permite renombrar o borrar un archivo o abrir la carpeta que contiene los registros en el Explorador de Windows (Típicamente `C:\Usuarios\Documentos Públicos\FRC\Ficheros de registro\N-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.

Pestaña Drive



Se puede utilizar el menú desplegable *Select Autonomous...* para mostrar las rutinas autónomas disponibles y elegir una para ejecutar en el partido.

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":
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   // 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
```

El envío a la entrada «Gyro» de NetworkTables rellenará el giroscopio aquí.

JAVA

```
SmartDashboard.putNumber("Gyro", drivetrain.getHeading());
```

C++

```
frc::SmartDashboard::PutNumber("Gyro", Drivetrain.GetHeading());
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putNumber("Gyro", self.drivetrain.getHeading())
```

Hay cuatro salidas que muestran la potencia del motor a la transmisión. Está configurado para 2 motores por lado y una transmisión estilo tanque. Esto se hace configurando «RobotDrive Motors» como en el ejemplo siguiente.

JAVA

```
SmartDashboard.putNumberArray("RobotDrive Motors", {drivetrain.getLeftFront(),␣
↪drivetrain.getRightFront(), drivetrain.getLeftBack(), drivetrain.getRightBack()});
```

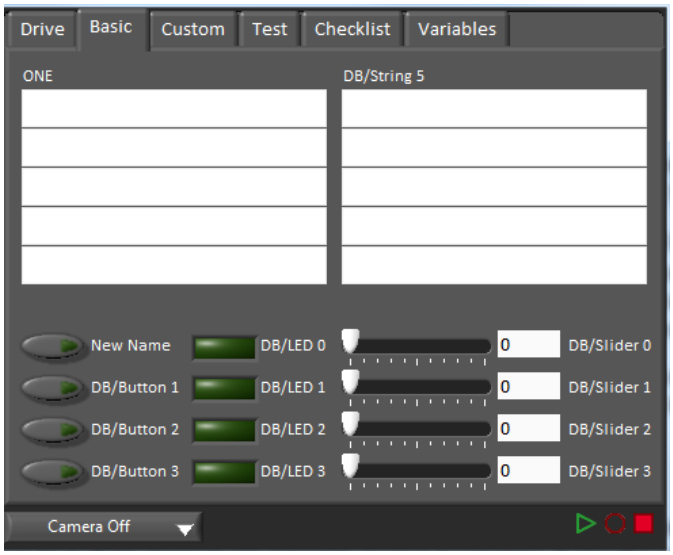
C++

```
frc::SmartDashboard::PutNumberArray("Gyro", {drivetrain.GetLeftFront(), drivetrain.
↪GetRightFront(), drivetrain.GetLeftBack(), drivetrain.GetRightBack()});
```

PYTHON

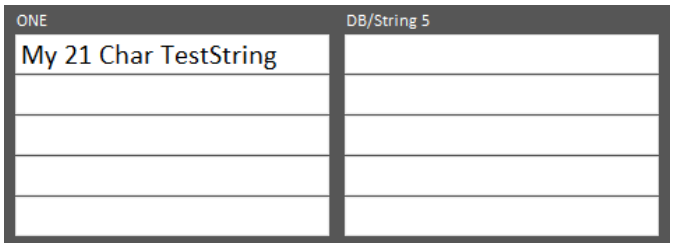
```
from wpilib import SmartDashboard

SmartDashboard.putNumberArray("RobotDrive Motors", [self.drivetrain.getLeftFront(),␣
↪self.drivetrain.getRightFront(), self.drivetrain.getLeftBack(), self.drivetrain.
↪getRightBack()])
```

Pestaña basica



La pestaña Basic utiliza un número de teclas en la sub-tabla «DB» para enviar/recibir datos de la Dashboard. Los LED´s son sólo de salida, los otros campos son todos bidireccionales (enviar o recibir).

Strings



Los strings están etiquetadas de arriba a abajo, de izquierda a derecha de «DB/String 0» a «DB/String 9». Cada campo de un string puede mostrar al menos 21 caracteres (el número exacto depende de qué caracteres). Para escribir a estas cadenas:

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")
```

Para leer los datos tipo string ingresados en la Dashboard:

JAVA

```
String dashData = SmartDashboard.getString("DB/String 0", "myDefaultData");
```

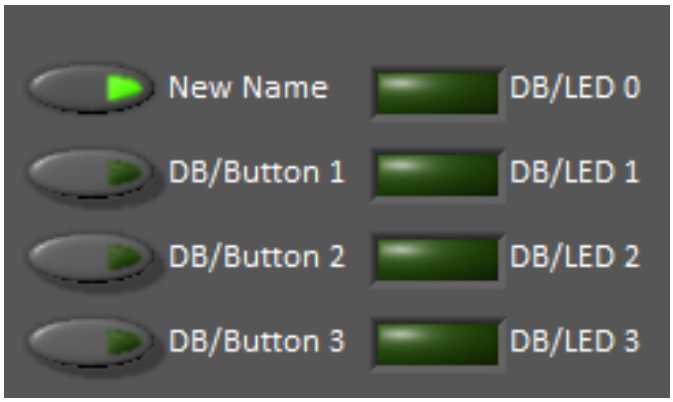
C++

```
std::string dashData = frc::SmartDashboard::GetString("DB/String 0", "myDefaultData");
```

PYTHON

```
from wpilib import SmartDashboard

dashData = SmartDashboard.getString("DB/String 0", "myDefaultData")
```

Botones y LEDs



Los botones y LEDs son valores booleanos y están etiquetados de arriba a abajo de «DB/Button 0» a «DB/Button 3» y «DB/LED 0» a «DB/LED 3». Los botones son bidireccionales, los LEDs sólo son capaces de ser escritos desde el robot y leidos en la Dashboard. Para escribir a los botones o LEDs:

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)
```

Para leer desde los botones: (el valor predeterminado es falso)

JAVA

```
boolean buttonValue = SmartDashboard.getBoolean("DB/Button 0", false);
```

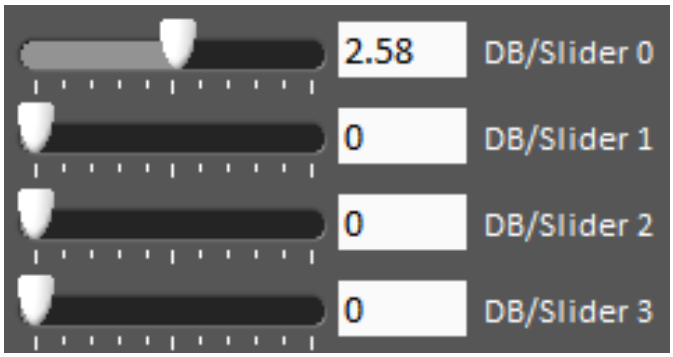
C++

```
bool buttonValue = frc::SmartDashboard::GetBoolean("DB/Button 0", false);
```

PYTHON

```
from wpilib import SmartDashboard

buttonValue = SmartDashboard.getBoolean("DB/Button 0", false)
```

Sliders



Los sliders o controles deslizantes son controles/indicadores analógicos bidireccionales (dobles) con un rango de 0 a 5. Para escribir a estos indicadores:

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)
```

Para leer valores de una Dashboard dentro del robot: (valor predeterminado de 0.0)

JAVA

```
double dashData = SmartDashboard.getNumber("DB/Slider 0", 0.0);
```

C++

```
double dashData = frc::SmartDashboard::GetNumber("DB/Slider 0", 0.0);
```

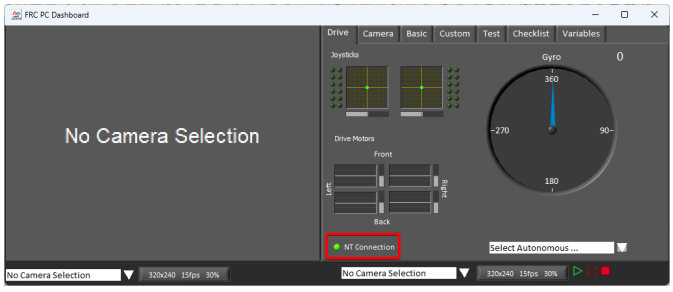
PYTHON

```
from wpilib import SmartDashboard

dashData = SmartDashboard.getNumber("DB/Slider 0", 0.0)
```

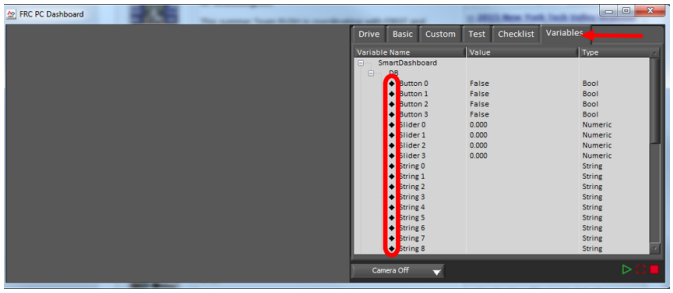
11.6.3 Troubleshooting Dashboard Connectivity

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

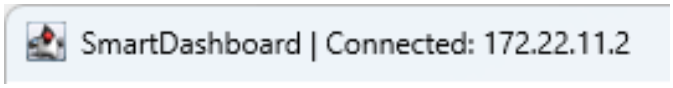


The LabVIEW Dashboard has a NetworkTables Connection indicator on the front panel.

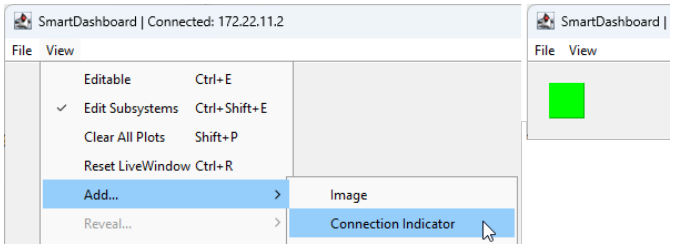


On the Variables tab of the Dashboard, the variables are shown with a black diamond when they are not synced with the robot. Once the Dashboard connects to the robot and these variables are synced, the diamond will disappear.

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.

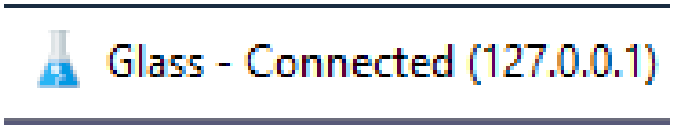


For more visibility, you can also add a Connection Indicator widget. The connection indicator can be moved or re-sized if the Editable checkbox is checked.

Recognizing Shuffleboard Connectivity

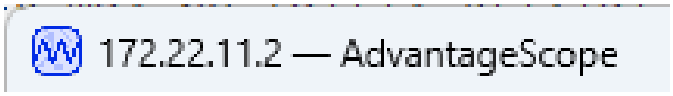
Shuffleboard indicates if it is connected or not in the bottom right corner of the application as shown in the image above. See *page* for more on configuring the connection.

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 Telemetry

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.

Nota: 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.

At the most basic level, the *Riolog* provides support for viewing print statements from robot code. This is useful for on-the-fly debugging of problematic code, but does not scale as console interfaces are not suitable for rich data streams.

WPILib supports several *dashboards* that allow users to more easily send rich telemetry data to the driver-station computer. All WPILib dashboards communicate with the *NetworkTables* protocol, and so they are *to some degree* interoperable (telemetry logged with one dashboard will be visible on the others, but the specific widgets/formatting will generally not be compatible). NetworkTables (and thus WPILib all dashboards) currently support the following data types:

- `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 Robot Telemetry with Sendable

While the WPILib dashboard APIs allow users to easily send small pieces of data from their robot code to the dashboard, it is often tedious to manually write code for publishing telemetry values from the robot code's operational logic.

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 provides this functionality with the `Sendable` interface. Classes that implement `Sendable` are able to register value listeners that automatically send data to the dashboard - and, in some cases, receive values back. These classes can be declaratively sent to any of the WPILib dashboards (as one would an ordinary data field), removing the need for teams to write their own code to send/poll for updates.

12.2.1 What is 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.

Many WPILib classes (such as *Commands*) already implement `Sendable`, and so can be sent to the dashboard without any user modification. Users are also able to easily extend their own classes to implement `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 Sending a Sendable to the Dashboard

Nota: Unlike simple data types, Sendables are automatically kept up-to-date on the dashboard by WPILib, without any further user code - «set it and forget it». Accordingly, they should usually be sent to the dashboard in an initialization block or constructor, *not* in a periodic function.

To send a `Sendable` object to the dashboard, simply use the dashboard's `putData` method. For example, an «arm» class that uses a *PID Controller* can automatically log telemetry from the controller by calling the following in its constructor:

JAVA

```
SmartDashboard.putData("Arm PID", armPIDController);
```

C++

```
frc::SmartDashboard::PutData("Arm PID", &armPIDController);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData("Arm PID", armPIDController)
```

Additionally, some `Sendable` classes bind setters to the data values sent *from the dashboard to the robot*, allowing remote tuning of robot parameters.

12.3 Registro de telemetría en robot en registros de datos

De forma predeterminada, no se registra (guarda) ningún dato de telemetría en el robot. La clase «DataLogManager» proporciona un envoltorio conveniente alrededor de la clase «DataLog» una clase de nivel inferior para el registro en el robot de datos de telemetría. Los registros de datos de WPILib son binarios por razones de tamaño y velocidad. En general, las facilidades de registro de datos proporcionadas por WPILib tienen un sobrecosto mínimo para el código del robot, ya que toda la E/S de archivos se realiza en un hilo separado: la operación de registro consiste principalmente en la adquisición de un mutex y la copia de los datos.

12.3.1 Estructura de los registros de datos

Similar a NetworkTables, los registros de datos tienen el concepto de entradas con identificadores de cadena (keys) con un tipo de datos especificado. A diferencia de NetworkTables, el tipo de datos no puede cambiarse después de que se crea la entrada, y las entradas también tienen metadatos: una cadena arbitraria (típicamente JSON) que se puede utilizar para transmitir información adicional sobre la entrada, como la fuente de datos o el esquema de datos. Además, a diferencia de NetworkTables, la operación de registro de datos es unidireccional: la clase `DataLog` solo puede escribir registros de datos (no admite la lectura de valores escritos) y la clase `DataLogReader` solo puede leer registros de datos (no admite el cambio de valores en el registro de datos).

Los registros de datos consisten en una serie de registros con marca de tiempo. Los registros de control permiten iniciar, finalizar o cambiar los metadatos de las entradas, y los registros de datos registran cambios en los valores de los datos. Las marcas de tiempo se almacenan en microsegundos enteros; cuando se ejecuta en el RoboRIO, se utiliza la marca de tiempo del FPGA (la misma marca de tiempo devuelta por `Timer.getFPGATimestamp()`).

Nota: For more information on the details of the data log file format, see the WPILib Data Log File Format Specification.

12.3.2 Registro de datos estándar utilizando 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.

Nota: 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`.

Al iniciar, se eliminarán todos los archivos de registro existentes en los que no se haya conectado un DS. Si hay menos de 50 MB de espacio libre en el almacenamiento objetivo, se eliminarán los archivos de registro `FRC_` (del más antiguo al más nuevo) hasta que haya 50 MB libres O queden 10 archivos restantes.

El uso más básico de DataLogManager solo requiere una línea de código (normalmente esto se llamaría desde `robotInit`). Esto registrará todos los cambios de NetworkTables en el registro de datos.

JAVA

```
import edu.wpi.first.wpilibj.DataLogManager;

// Starts recording to data log
DataLogManager.start();
```

C++

```
#include "frc/DataLogManager.h"

// Starts recording to data log
frc::DataLogManager::Start();
```

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

```
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++

```
#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

```
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.

Nota: 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

```
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;
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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

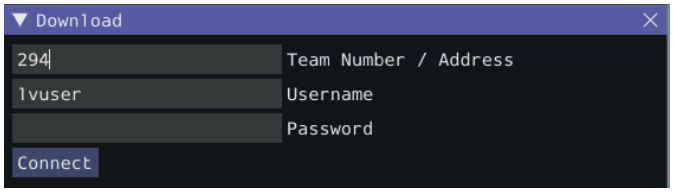
The DataLogTool desktop application integrates a SFTP client for downloading data log files from a network device (e.g. roboRIO or coprocessor) to the local computer.

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

Nota: 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.

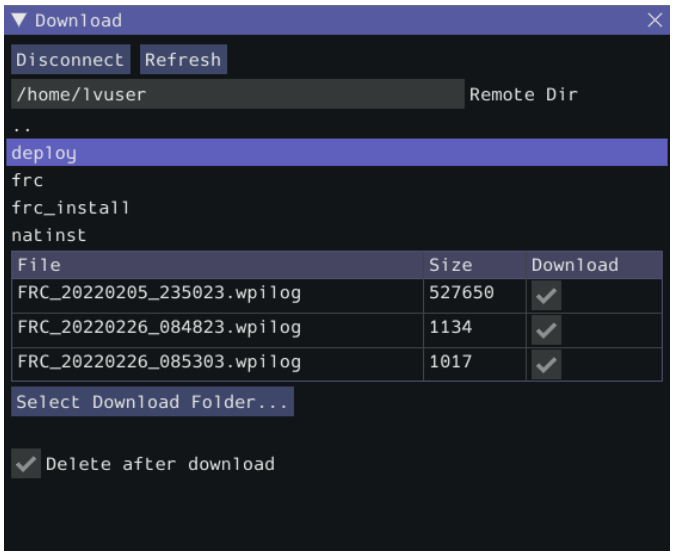
The tool also supports connecting to network devices other than the roboRIO, such as coprocessors, as long as the device supports SFTP password-based authentication.

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.

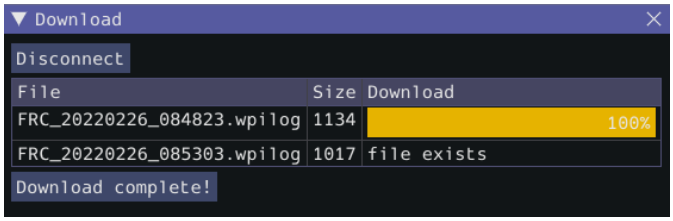
Nota: On the roboRIO, log files are typically saved to either `/home/lvuser/logs` or `/u/logs` (USB stick location).



Click *Select Download Folder...* to bring up a file browser for the local computer.

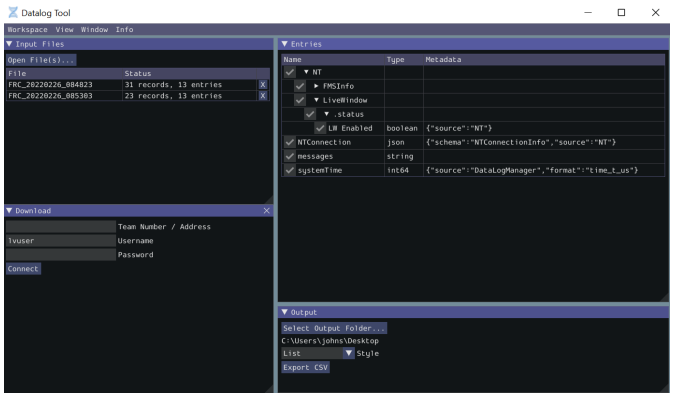
If you want to delete the files from the remote device after they are downloaded, check the *Delete after download* checkbox.

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.

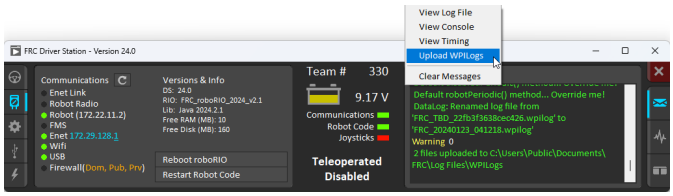


The conversion process is started by opening data log files in the «Input Files» window. Files are opened by clicking *Open File(s)...*. Summary status on each file (e.g. number of records and entries) is displayed. Clicking *X* in the table row closes the file.

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 Writing Your Own Sendable Classes

Desde que la interfaz «Sendable» solo tiene un método, escribir tus propias clases que implementen «Sendable» (y por lo tanto registren automáticamente valores en el panel de control y/o consuman valores del panel de control) es extremadamente fácil: simplemente proporciona una implementación para el método «initSendable» que se puede sobrescribir, en el cual los setters y getters de los campos de tu clase se enlazan declarativamente a los valores clave (sus nombres de visualización en el panel de control).

For example, here is the implementation of `initSendable` from WPILib's `BangBangController`:

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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  builder.AddBooleanProperty(
      "atSetpoint", [this] { return AtSetpoint(); }, nullptr);
}
```

To enable the automatic updating of values by WPILib «in the background», `Sendable` data names are bound to getter and setter functions rather than specific data values. If a field that you wish to log has no defined setters and getters, they can be defined inline with a lambda expression.

12.5.1 The SendableBuilder Class

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.

Databinding with addProperty Methods

Como todo el código del panel de control de WPILib, los campos «Sendable» se transmiten en última instancia a través de *NetworkTables*, y por lo tanto, los métodos de enlace de datos proporcionados por «SendableBuilder» coinciden con los tipos de datos admitidos por NetworkTables.

- `boolean`: `addBooleanProperty`
- `boolean[]`: `addBooleanArrayProperty`
- `double`: `addDoubleProperty`
- `double[]`: `addDoubleArrayProperty`
- `string`: `addStringProperty`
- `string[]`: `addStringArrayProperty`
- `byte[]`: `addRawProperty`

Ensuring Safety with setSafeState and setActuator

Since `Sendable` allows users to consume arbitrary values from the dashboard, it is possible for users to pipe dashboard controls directly to robot actuations. This is extremely unsafe if not done with care; dashboards are not a particularly good interface for controlling robot movement, and users generally do not expect the robot to move in response to a change on the dashboard.

To help users ensure safety when interfacing with dashboard values, `SendableBuilder` exposes a `setSafeState` method, which is called to place any `Sendable` mechanism that actuates based on dashboard input into a safe state. Any potentially hazardous user-written `Sendable` implementation should call `setSafeState` with a suitable safe state implementation. For example, here is the implementation from the WPILib `PWMMotorController` class:

JAVA

```java
  @Override
  public void initSendable(SendableBuilder builder) {
    builder.setSmartDashboardType("Motor Controller");
    builder.setActuator(true);
    builder.setSafeState(this::disable);
    builder.addDoubleProperty("Value", this::get, this::set);
  }
```

C++

```cpp
void PWMMotorController::InitSendable(wpi::SendableBuilder& builder) {
  builder.SetSmartDashboardType("Motor Controller");
  builder.SetActuator(true);
  builder.SetSafeState([=, this] { Disable(); });
  builder.AddDoubleProperty(
      "Value", [=, this] { return Get(); },
      [=, this](double value) { Set(value); });
```

Additionally, users may call `builder.setActuator(true)` to mark any mechanism that might move as a result of `Sendable` input as an actuator. Currently, this is used by *Shuffleboard* to disable actuator widgets when not in *LiveWindow* mode.

12.6 Third-Party Telemetry Libraries

Truco: Is your library not listed here when it should be? Open a pull request to add it!

Several third-party logging utilities and frameworks exist that provide functionality beyond what is currently provided by WPILib:

- AdvantageKit (Java only): «Log everything»-based logging framework with hooks for replaying logged data in *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

Programación de LabVIEW de FRC

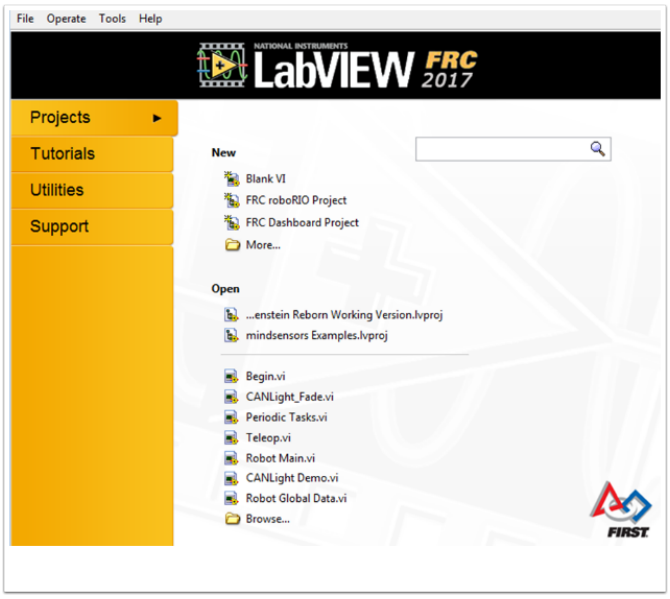
13.1 Crear Programas de Robot

13.1.1 Tutorial de conducción de tanques

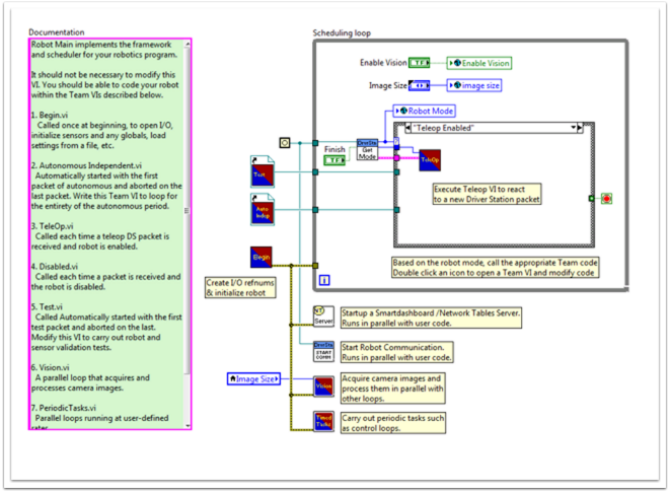
Pregunta: ¿Cómo consigo que mi robot se conduzca con dos joysticks usando el accionamiento por tanque?

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. Abra LabVIEW y haga doble clic en `Proyecto FRC roboRIO`.



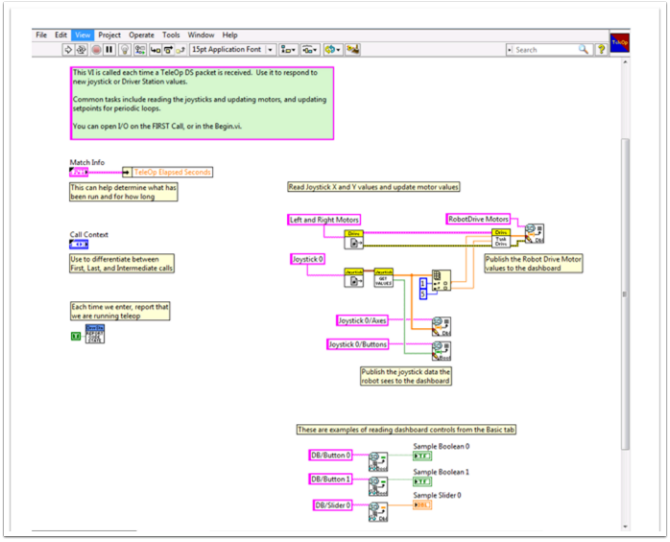
2. Dé un nombre a su proyecto, agregue el número de su equipo y seleccione `Arcade Drive Robot roboRIO`. Puede seleccionar otra opción, sin embargo, este tutorial discutirá cómo configurar la unidad de tanque para este proyecto.
3. En la ventana del Explorador de proyectos, abra el `Robot Main.vi`.
4. Presione:kbd:*Ctrl+E* para ver el diagrama de bloque. Debería verse como la siguiente imagen:



5. Haga doble clic en «Teleop» vi dentro de la estructura del caso Teleop Enabled. Mire su diagrama de bloques. Querrá hacer dos cambios aquí:

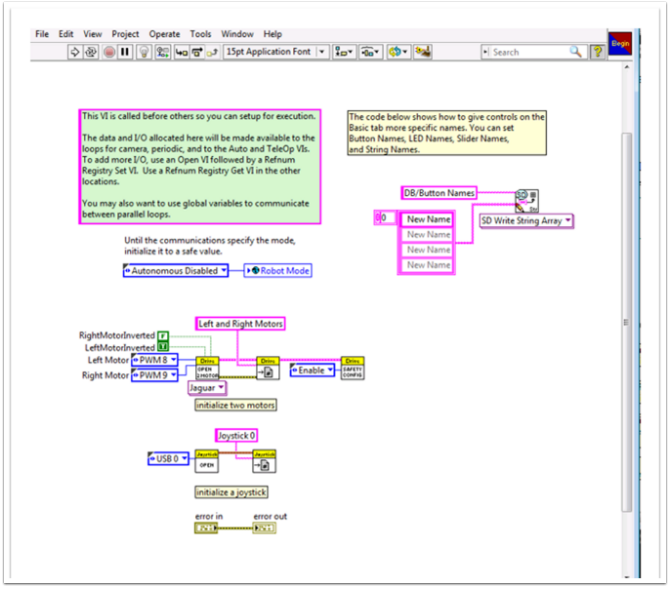
- Reemplace Arcade Drive con el tanque drive.vi. Esto se puede encontrar haciendo clic derecho en el diagrama de bloques >> WPI Robotics Library >> Robot Drive >> y haciendo clic en Tank Drive VI.
- Encuentre la función Array que está en el Índice después de Get Values.vi. Necesitará crear dos constantes numéricas y conectar cada una a una de las entradas de índice. Puede determinar cuáles son los valores de cada índice mirando la pestaña de Dispositivos USB en el FRC®.driver station. Mueva los dos joysticks para determinar a qué número (índice) están vinculados. Probablemente querrás usar el índice del eje Y para

cada joystick. Esto se debe a que es intuitivo presionar el joystick hacia arriba cuando quieres que los motores vayan hacia adelante, y hacia abajo cuando quieres que vayan en reversa. Si seleccionas el índice del eje X para cada uno, entonces tendrás que mover el joystick a la izquierda o a la derecha (direcciones del eje x) para que los motores del robot se muevan. En mi configuración, he seleccionado el índice 1 para el control del eje Y de mis motores izquierdos y el índice 5 como control del eje Y de los motores derechos. Puedes ver los ajustes en LabVIEW en la siguiente imagen:



6. A continuación, querrá volver a su «Robot Main.vi» y haga doble clic en «Begin.vi».
7. Lo primero que debe confirmar en este VI es que sus motores izquierdo y derecho están conectados a las mismas líneas PWM en LabVIEW que en su PDP (Panel de distribución de energía).
8. Lo segundo a confirmar en este VI es que el «Open 2 Motor.vi» tiene seleccionado el controlador de motor correcto (Talon, Jaguar, Victor, etc.).

   Por ejemplo, estoy usando controladores de motor Jaguar y mis motores están conectados a PWM 8 y 9. La siguiente imagen muestra los cambios que necesito hacer:



9. ¡Guarde todos los Vis en los que ha realizado ajustes y ahora puede conducir un robot con un tanque de impulsión!

13.1.2 Tutorial de comando y control

Introducción

Command and Control es una nueva plantilla de LabVIEW agregada para la temporada 2016 que organiza el código del robot en comandos y controladores para una colección de subsistemas específicos del robot. Cada subsistema tiene un bucle de control independiente o una máquina de estado que se ejecuta a la velocidad adecuada para el mecanismo y los comandos de alto nivel que actualizan las operaciones y los puntos de ajuste deseados. Esto hace que sea muy fácil para el código autónomo construir secuencias síncronas de comandos. Mientras tanto, TeleOp se beneficia porque puede usar los mismos comandos sin necesidad de esperar a que se completen, lo que permite una fácil cancelación e inicio de nuevos comandos de acuerdo con la entrada del equipo de manejo. Cada subsistema tiene un panel que muestra sus valores de sensor y control a lo largo del tiempo, y seguimiento de comandos para ayudar en la depuración.

¿Qué es Command and Control?

Command and Control reconoce que los robots FRC® tienden a estar formados por mecanismos relativamente independientes como Drive, Shooter, Arm, etc. Cada uno de estos se conoce como un subsistema y necesita un código que coordine los diversos sensores y actuadores del subsistema para completar los comandos solicitados. o acciones, como «Cerrar pinza» o «Brazo inferior». Uno de los principios clave de este marco es que los subsistemas tendrán cada uno un circuito de controlador independiente que es el único responsable de actualizar los motores y otros actuadores. El código fuera del controlador del subsistema puede emitir comandos que pueden cambiar la salida del robot, pero no deben cambiar directamente ninguna salida. La diferencia es muy sutil, pero esto significa que los resultados solo pueden actualizarse desde una ubicación en el proyecto. Esto acelera la depuración de un robot que se comporta de manera inesperada al brindarle la capacidad de revisar una lista de comandos enviados al subsistema en lugar de buscar en su proyecto donde se haya modificado una salida. También se vuelve más fácil agregar un sensor adicional, cambiar de marcha o deshabilitar un mecanismo sin necesidad de modificar el código fuera del controlador.

El código del juego, principalmente consiste en el Autónomo y Teleoperado, va a necesitar actualizar los puntos de ajuste y reaccionar al estado de ciertos mecanismos. Para el Autónomo, es muy común definir la operación del robot como una secuencia de operaciones – maneja aquí, recoge eso, llévalo allá, dispara, etc. Los comandos se pueden conectar secuencialmente con lógica adicional para construir rápido rutinas complejas. Para Teleoperado, los mismos comandos pueden operar de manera síncrona, dejando que el robot siempre procese las últimas entradas del driver, y si lo implementa apropiadamente, nuevos comandos van a interrumpir, dejando que el drive team responda rápido en las condiciones de la cancha mientras también toma ventaja de comandos automatizados y secuencias de comandos.

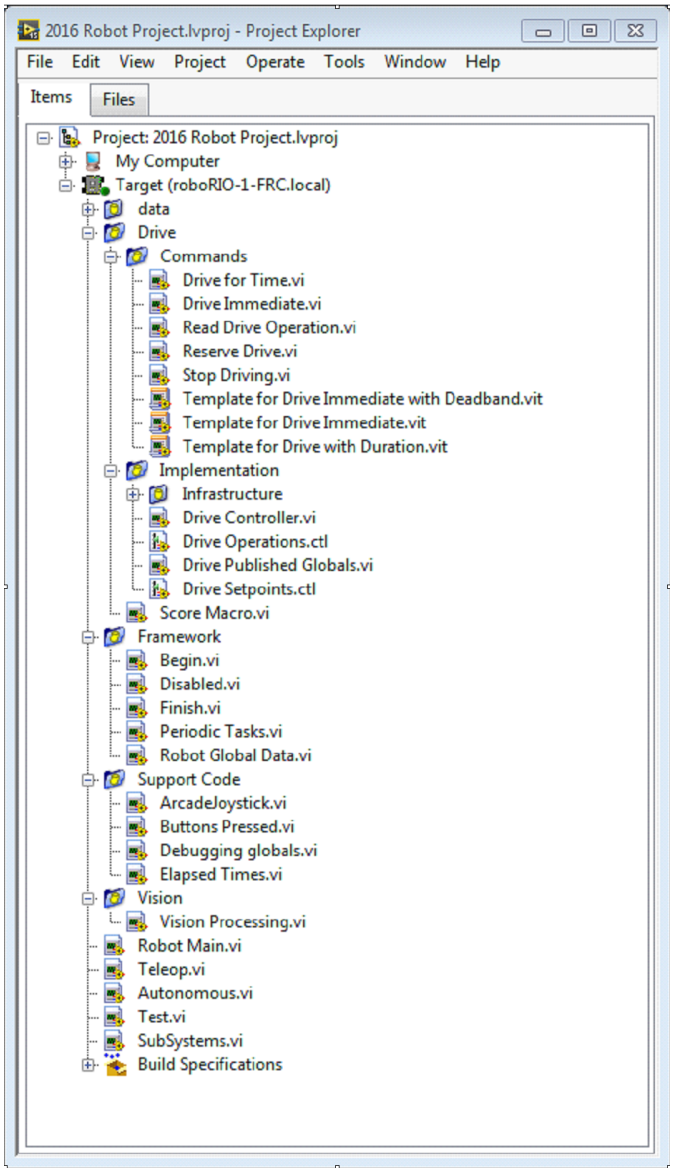
¿Por qué debemos usar Command and Control?

Command and Control añade funcionalidad a las plantillas de proyecto existentes en LabVIEW, permitiendo que el código se escale mejor con robots y códigos de robot mas sofisticados. Se usan subsistemas para abstraer los detalles de la implementación, y el código de juego es creado a partir de secuencias de VIs de comando de alto nivel. Los comandos por si mismos son VIS que pueden actualizar puntos fijos, realizar mapeo/escalado numérico entre unidades de ingeniería y unidades del mecanismo y pueden ofrecer opciones de sincronización. Si se realizan cambios físicos al robot, como modificar un a relaciona de engranes, se pueden realizar los cambios a solo un poco de VIs de comando para reflejar este cambio a través de todo el código fuente.

La encapsulación I/O ayuda a tener una operación mas predecible y una depuración mas rápida en caso de que ocurran conflictos entre los recursos. Debido a que cada comando es un VI, puede realizar un solo paso a través de los comandos o utilizar la funcionalidad Trace incorporada para ver una lista de todos los comandos enviados a cada subsistema. de comandos o agregue una lógica simple para determinar el comando correcto para ejecutar.

Parte 1: Explorador de proyectos

El Explorador de proyectos proporciona organización para todos los Vis y archivos que usted usará para su sistema de robot. A continuación se muestra una descripción de los componentes principales del Explorador de proyectos para ayudar con la expansión de nuestro sistema. Los elementos utilizados con más frecuencia se han marcado en negrita.



Mi computadora
: Los elementos que definen la operación en la computadora en la que se cargó el proyecto. Para un proyecto de robot, esto se utiliza como un objetivo de simulación y se completa con archivos de simulación.

Archivos de soporte de Sim
: The folder containing 3D *CAD* models and description files for the simulated robot.

Simulación del 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.

Posesiones
: Muestra los archivos usados en la simulación del código del robot. Esto se completará cuando designe el código para el objetivo del robot simulado.

Especificaciones de construcción
: Esto contendrá los archivos que definen cómo construir y desplegar el código para el objetivo de robot simulado.

Objetivo (roboRIO-TEAM-FRC.local)
Los elementos que definen la operación en el roboRIO ubicado en (dirección).

Unidad
La implementación del subsistema y los comandos para la base de accionamiento del robot. Esto sirve como un reemplazo personalizado para los VIs de WPILib RobotDrive.

Marco de trabajo
VI utilizados para código de robot que no forma parte de un subsistema que no se utilizan con mucha frecuencia.

Comienzo
Se manda a llamar una vez cuando el código del robot comienza. Esto es útil para códigos de iniciación que no pertenecen a un subsistema en particular.

Deshabilitado
Se llama una vez por cada paquete deshabilitado y se puede usar para depurar sensores cuando no desea que el robot se mueva.

Terminar
Durante el desarrollo, esto puede ser llamado cuando el código del robot termine. No se llama «abortar» o cuando se apaga la energía.

Tareas periódicas
Un buen lugar para los bucles periódicos ad hoc para la depuración o la vigilancia

Datos globales del robot
Útil para compartir información de los robots que no pertenece a un subsistema.

Código de apoyo
Ayudas para la depuración y el desarrollo de códigos.

Visión
Subsistema y comandos para la cámara y el procesamiento de imágenes.

Robot Main.vi
El máximo nivel VI que se ejecutará mientras se desarrolla el código.

Autonomous.vi
VI que se desarrolla durante el período autónomo.

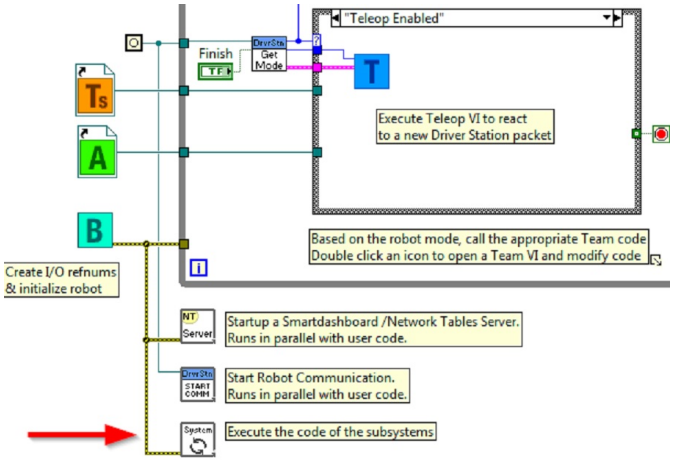
Teleop.vi
VI que se llama para cada paquete de TeleOp.

Test.vi
VI que se ejecuta cuando la «driver station» está en modo de prueba.

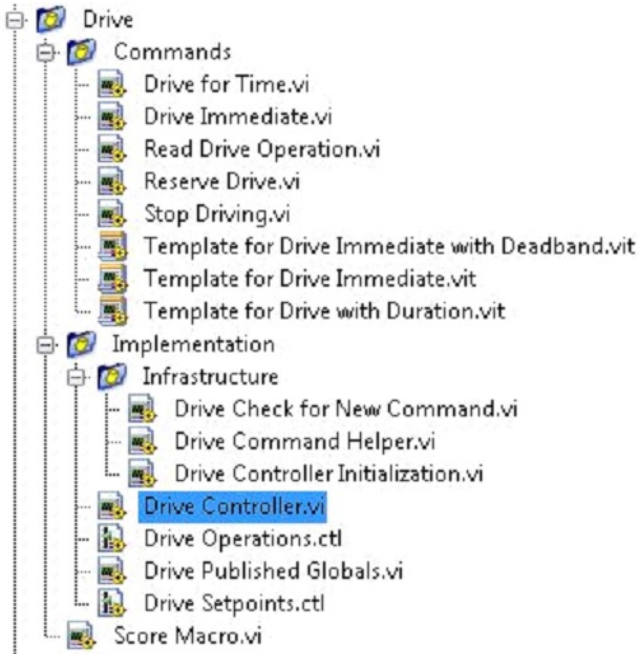
SubSystems.vi
VI que contiene y pone en marcha todos los subsistemas.

Posesiones
Muestra los archivos usados por el código del robot.

Especificaciones de construcción
Se usa para construir y ejecutar el código como una aplicación de inicio una vez que el código funciona correctamente.



Explorador del proyecto del subsistema de conducción



Comandos:

Esta carpeta contiene el comando VIs que solicita al controlador realizar una operación. Además contiene plantillas para crear comandos de unidad adicionales.

Nota: Después de crear un nuevo comando, podría necesitar editar `Drive Setpoints.ctl` para añadir o actualizar campos que el controlador usa para definir la nueva operación. También podría necesitar ir dentro del Drive Controller.vi y modificar la estructura del caso para añadir un caso para cada valor.

Implementación

Estos son los VIs y Controles usados para construir el subsistema.

Infraestructura VIs

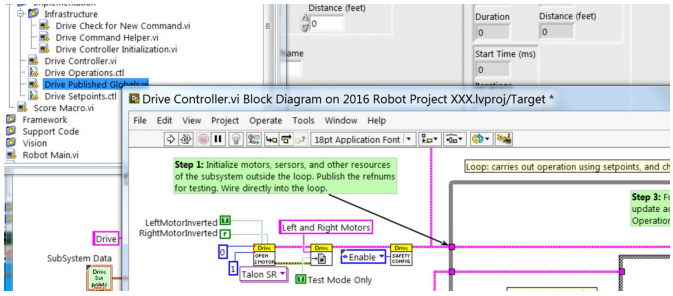
- Comprobación de conducción para el Nuevo Comando: Se llama cada iteración del bucle controlador. Comprueba si hay nuevos comandos, actualiza los datos de tiempo, y al terminar notifica un comando de espera.
- Conduzca el Ayudante de Comando.vi: Los comandos llaman a este VI para notificar al controlador que se ha emitido un nuevo comando.
- Drive Controller Initialization.vi: Asigna el notificador y combina el tiempo, el comando predeterminado y otra información en un solo cable de datos.

- Drive Controller.vi: Este VI contiene el bucle de la máquina de control/estado. El panel también puede contener pantallas útiles para la depuración.
- Drive Operation.ctl: Este tipedef define los modos de funcionamiento del controlador. Muchos comandos pueden compartir una operación.
- Drive Setpoint.ctl: Contiene los archivos de datos usados por todos los modos de funcionamiento del subsistema Drive.
- Drive Published Globals.vi:: Un lugar útil para publicar información global sobre el subsitema drive.

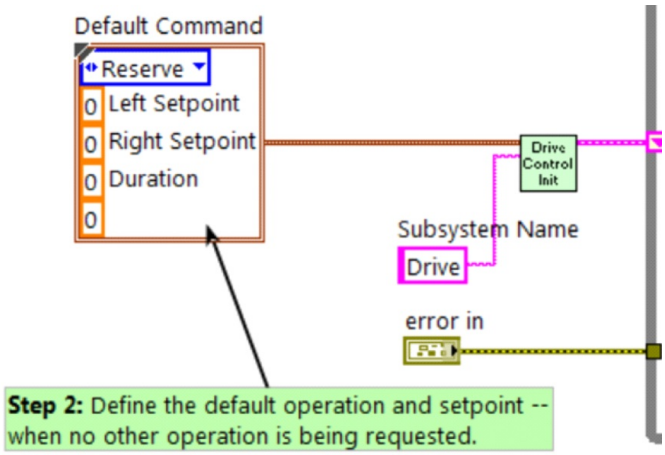
Parte 2: Iniciando el subsistema de accionamiento

Hay comentarios en verde en el diagrama de bloques del controlador que señalan áreas clave que querrás saber cómo editar.

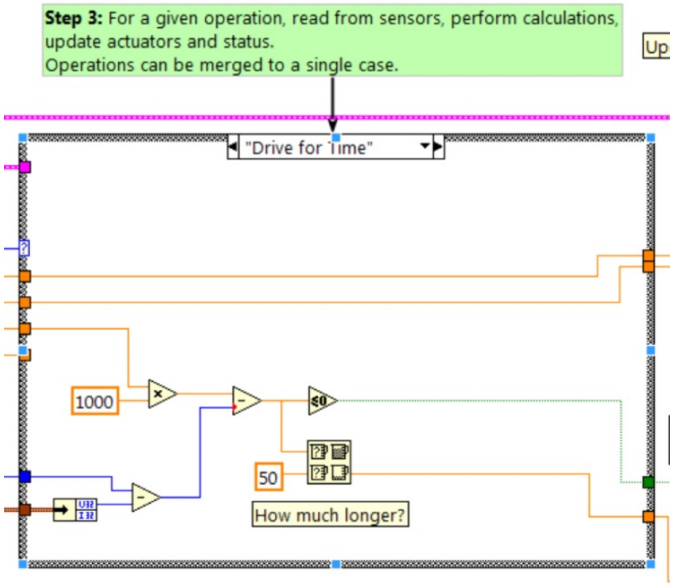
El área a la izquierda del bucle de control se ejecutará una vez cuando el subsistema se ponga en marcha. Aquí es donde normalmente se asignarán e inicializarán todos los datos de E/S y de estado. Puedes publicar los refnums de E/S, o puedes registrarlos en el modo de prueba sólo para mantenerlos privados de modo que otros códigos no puedan actualizar los motores sin usar un comando.



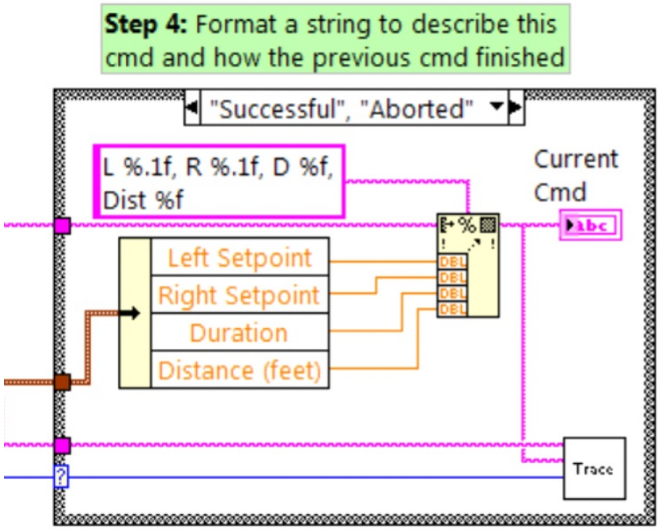
Nota: La inicialización de los recursos de cada subsistema en su respectivo Controller.vi en lugar de en Begin.vi mejora la encapsulación de E/S, reduciendo los posibles conflictos de recursos y simplifica la depuración.



Parte de la inicialización consiste en seleccionar la operación predeterminada y los valores de punto de ajuste cuando no se está procesando ninguna otra operación.

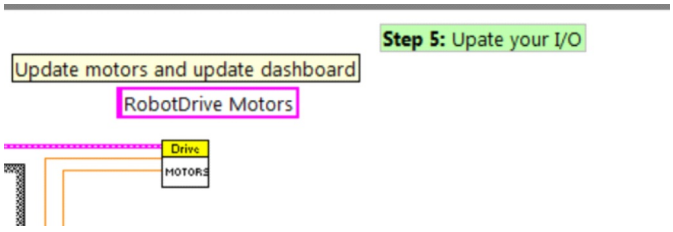


Dentro del bucle de control hay una declaración de caso donde las operaciones se llevan a cabo realmente. Los valores del punto de ajuste, el retardo de iteración, el recuento de iteración y los sensores pueden influir en el funcionamiento del subsistema. Esta estructura de caso tiene un valor para cada estado de operación del subsistema.



Cada iteración del bucle controlador actualizará opcionalmente el Trace VI. El marco ya incorpora el nombre del subsistema, la operación y la descripción, y puede resultar útil dar formato a valores de puntos de ajuste adicionales en la información de la traza. Abra el Trace VI y haga clic en Activar mientras el código del robot se ejecuta a los puntos de ajuste actuales y a los comandos enviados a cada subsistema.

El objetivo principal del controlador es actualizar los actuadores del subsistema. Esto puede ocurrir dentro de la estructura de la caja, pero muchas veces, es beneficioso hacerlo aguas abajo de la estructura para asegurar que los valores se actualizan siempre con el valor correcto y en una sola ubicación en el código.



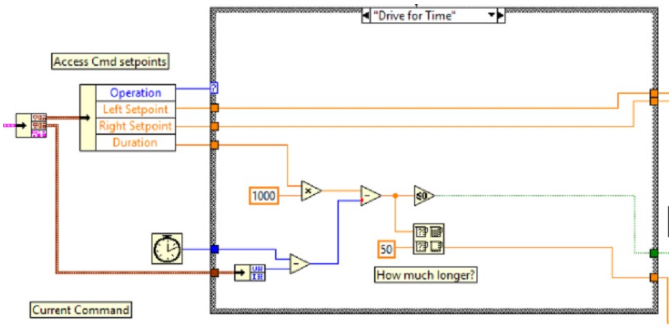
Parte 3: Comandos enviados por el subsistema de conducción

Hay 3 comandos de ejemplo enviados para cada nuevo subsistema:

Conduzca por el tiempo.vi

Este VI pone los motores en marcha durante un número determinado de segundos. Opcionalmente se sincroniza con la finalización del comando.

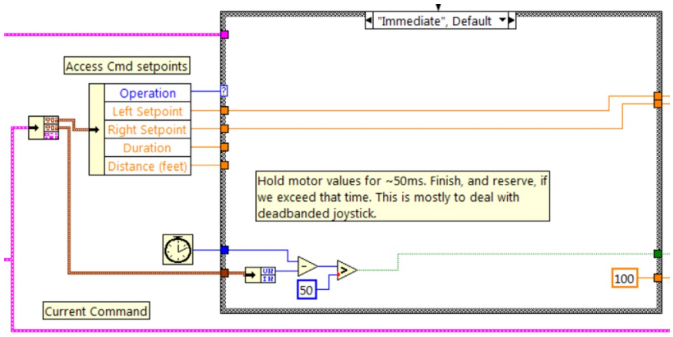
El caso Drive for Time hará funcionar los motores en el punto de ajuste hasta que el temporizador transcurra o se emita un nuevo comando. Si los motores tienen el tiempo de seguridad habilitado, es necesario actualizar los motores al menos una vez cada 100ms. Por eso el código espera el menor de los tiempos restantes y 50ms.



Conduzca inmediatamente.vi

Consigue las velocidades izquierda y derecha deseadas para los motores y ajustará los motores inmediatamente a esos puntos de ajuste.

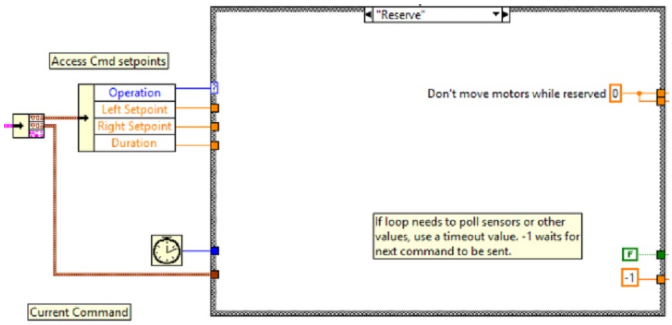
El caso inmediato actualiza los motores al punto de ajuste definido por el comando. El comando no se considera terminado ya que se quiere que los motores mantengan este valor hasta que llegue un nuevo comando o hasta un valor de tiempo de espera. El tiempo de espera es útil siempre que un comando incluye una banda muerta. No se solicitarán valores pequeños si son menores que la banda muerta, y resultarán en gruñidos o arrastramientos a menos que el comando se agote.



Stop Driving.vl

Detiene los motores, haciendo que el robot permanezca estacionario.

El comando de reserva apaga los motores y espera un nuevo comando. Cuando se utiliza con una secuencia de comandos con nombre, reserva identifica que el subsistema de accionamiento forma parte de una secuencia, incluso si no está moviendo el robot en ese momento. Esto ayuda a arbitrar el recurso del subsistema entre los comandos que se ejecutan simultáneamente.



Parte 4: Creando Comandos Nuevos

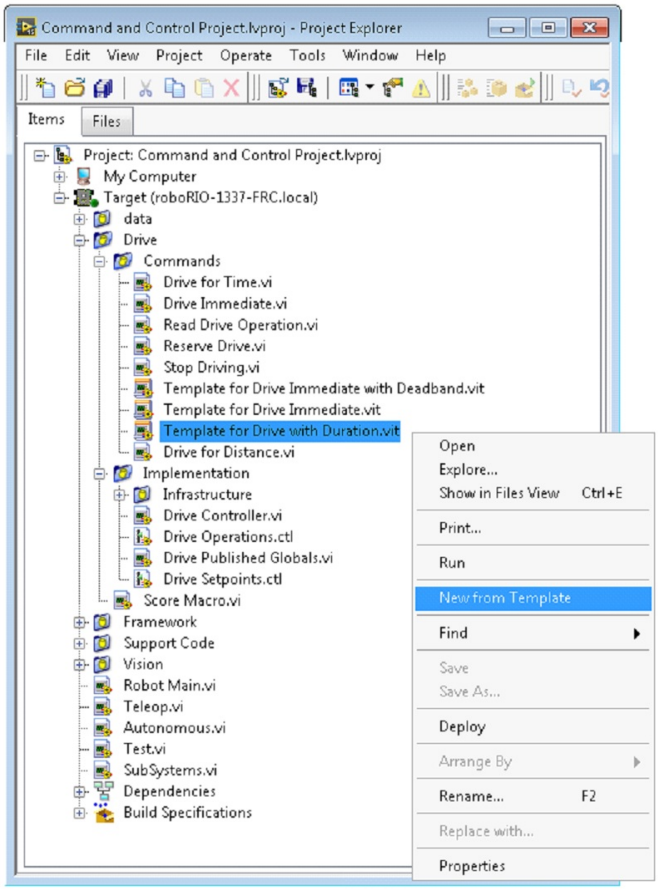
El comando y el Control framework permite a los usuarios crear fácilmente nuevos comandos para un subsistema. Para crear un nuevo comando abra la carpeta del subsistema/Comandos en la ventana del explorador de proyectos, seleccione una de las plantillas de VI para usar como partida inicial de su nuevo comando, clic derecho, y seleccione Nuevo Desde Plantilla.

- **Inmediato**: Este VI notifica al subsistema sobre el nuevo punto de ajuste.
- **Inmediatamente con la banda muerta**: Este VI compara el valor de entrada con la banda muerta y opcionalmente notifica al subsistema sobre el nuevo punto de ajuste.

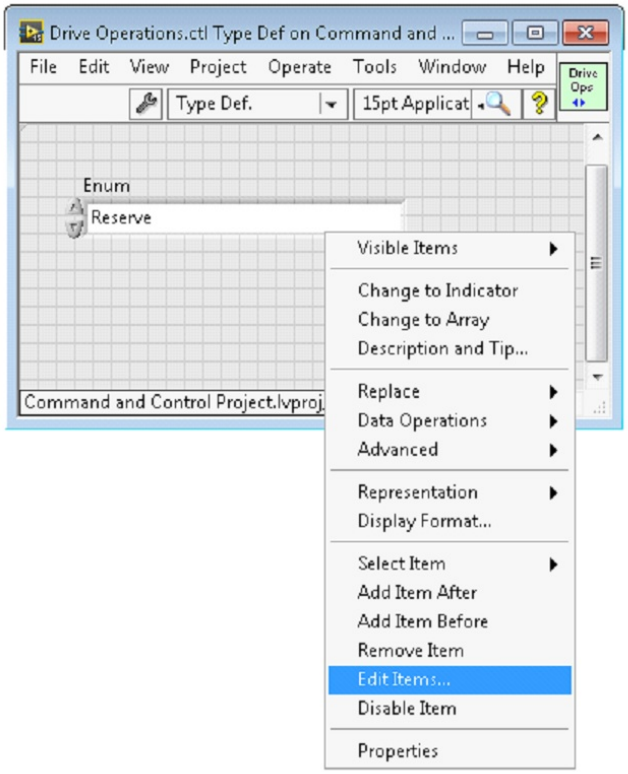
Esto es muy útil cuando se utilizan valores continuos de joystick.

- **Con la duración**: Este VI notifica al subsistema para que realice este comando durante la duración dada, y luego volver al estado por defecto. La sincronización determina si este VI inicia la operación y regresa inmediatamente, o espera a que la operación se complete. La primera opción se usa comúnmente para TeleOp, y la segunda para Secuenciación Autónoma.

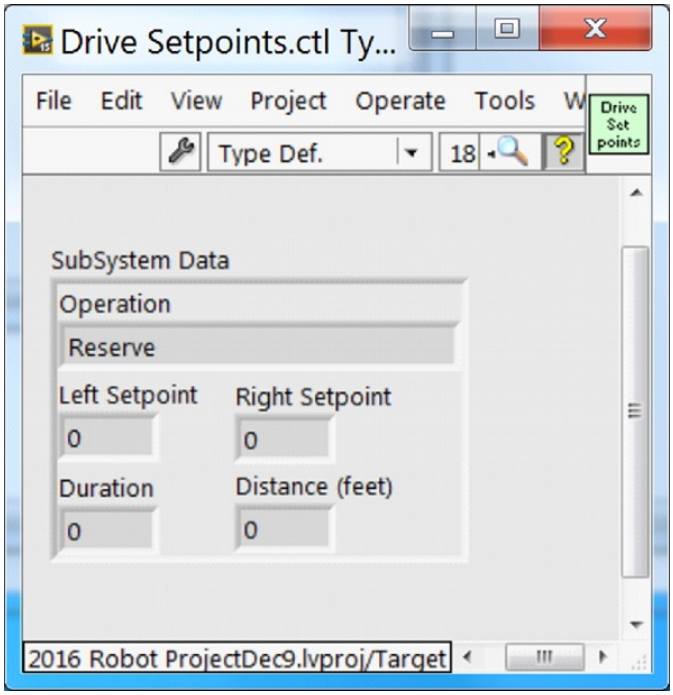
En este ejemplo, agregaremos el nuevo comando «Conducir por distancia».



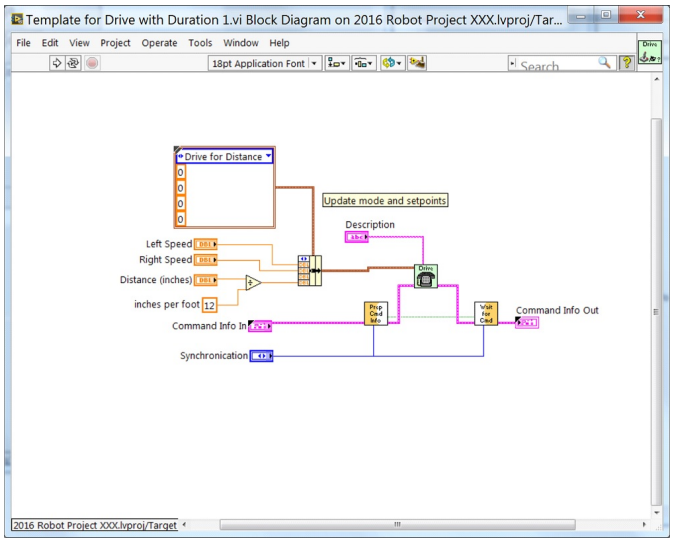
Primero, guarde el nuevo VI con un nombre descriptivo como «Drive for Distance». A continuación, determine si el nuevo comando necesita un nuevo valor agregado en Drive Operations enum typedef. El código del proyecto inicial ya tiene un valor de enumeración de Drive for Distance, pero la siguiente imagen muestra cómo agregaría uno si fuera necesario.



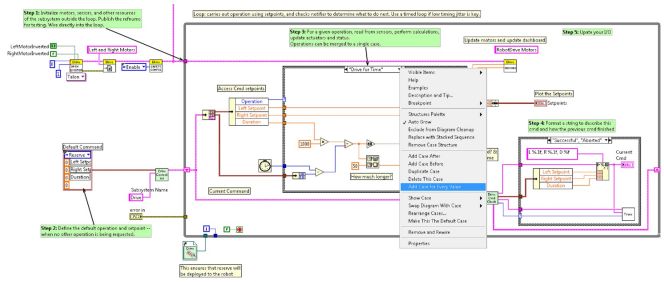
Si un comando necesita información adicional para ejecutarse, añádalo al control de puntos de ajuste. Por defecto, el subsistema Drive tiene campos para el Left Setpoint, Right Setpoint y Duration junto con la operación que se va a ejecutar. El comando Drive for Distance podría reutilizar la duración como distancia, pero sigamos adelante y agreguemos un control numérico a Drive Setpoints.ctl llamdo Distance (pies).



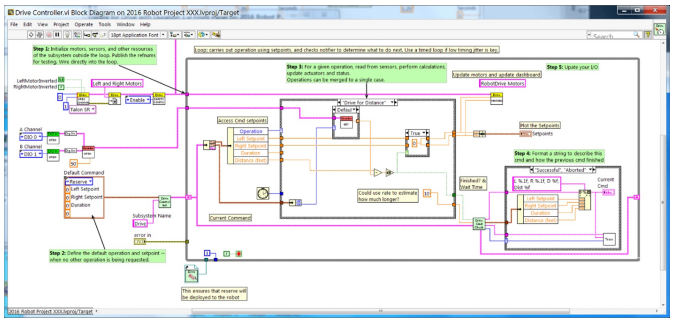
Una vez que tenemos todos los campos necesarios para especificar nuestro comando, podemos modificar el recién creado Drive for Distance.vi. Como se muestra a continuación, seleccione Drive for Distance en el menú desplegable del enum y añada un parámetro VI para especificar la distancia, las velocidades, etc. Si las unidades no coinciden, el comando VI es un gran lugar para mapear entre unidades.



A continuación, agregue código al controlador de unidad para definir qué sucede cuando se ejecuta el comando de unidad para distancia. Haga clic derecho en la estructura del caso y duplique o agregue caso para cada valor. Esto creará un nuevo caso «Drive for Distance».

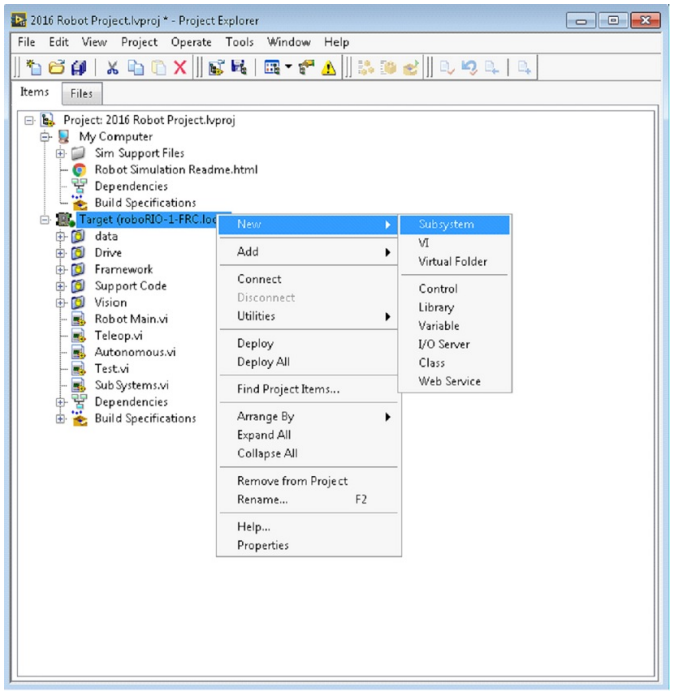


Para acceder a nuevos campos de puntos de ajuste, haga crecer el nodo de deshacer «Puntos de ajuste de Cmd de acceso». Abra su codificador (o codificadores) en el exterior, a la izquierda del bucle. En el nuevo diagrama de la estructura del caso, agregamos una llamada para restablecer el codificador en la primera iteración del bucle y leerlo de otra manera. También hay un código simple que compara los valores del codificador y actualiza la potencia del motor. Si se agregan nuevos controles al grupo de puntos de ajuste, también debe considerar agregarlos al Trace. Los cambios necesarios se muestran en la imagen a continuación.

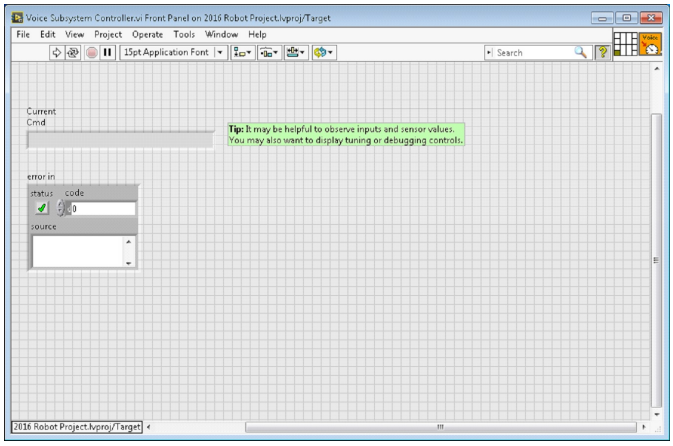


Parte 5: Creando un Subsistema

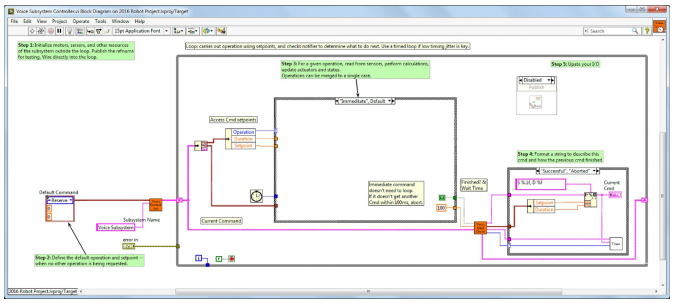
Para crear un nuevo subsistema, haga clic con el botón derecho del ratón en el objetivo de roboRIO y seleccione New» Subsystem. En el cuadro de diálogo emergente, introduzca el nombre del subsistema, enumere los modos de funcionamiento y especifique el color del icono.



Cuando haga clic en OK, la carpeta del subsistema se generará y se añadirá a la carpeta y al árbol del disco del proyecto. Contendrá una implementación base de los VIs y controles que componen un subsistema. Una llamada al nuevo controlador se insertará en los VI de los subsistemas. El controlador VI se abrirá, listo para que usted añada I/O e implemente el código de estado de la máquina o del control. Los iconos de los VI generados utilizarán el color y el nombre proporcionados en el diálogo. El código generado usará typedefs para los campos de puntos de ajuste y operaciones.



A continuación se muestra el diagrama de bloques del subsistema recién creado. Este código se generará automáticamente cuando se cree el subsistema.



13.2 Recursos LabVIEW

13.2.1 Recursos LabVIEW

Nota: Para aprender más sobre la programación en LabVIEW y específicamente la programación de robots FRC® en LabVIEW, revise los siguientes recursos.

Fundamentos 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.

Tutoriales de NI FRC

NI también hospeda algunos Tutoriales y presentaciones específicos de FRC de básico a avanzado. Para un recurso único y profundo, revisa las Clases de Entrenamiento Básico y Avanzado del FRC enlazadas cerca de la parte inferior de la página.

Tutoriales Instalados y Ejemplos

También hay tutoriales y ejemplos para todo tipo de tareas y componentes proporcionados como parte de su instalación de LabVIEW. Para acceder a los tutoriales, desde la pantalla de LabVIEW Splash (la pantalla que aparece cuando se lanza el programa por primera vez) haga clic en la pestaña `Tutoriales` en el lado izquierdo. Tenga en cuenta que los tutoriales están todos en un solo documento, así que una vez que esté abierto podrá navegar a otros tutoriales sin volver a la pantalla de inicio.

Para acceder a los ejemplos, haga clic en la pestaña `Support`, luego en `Find FRC Examples` o cada vez que trabajes en un programa abre el menú `Help`, seleccione `Find Examples` y abra la carpeta `FRC Robotics`.

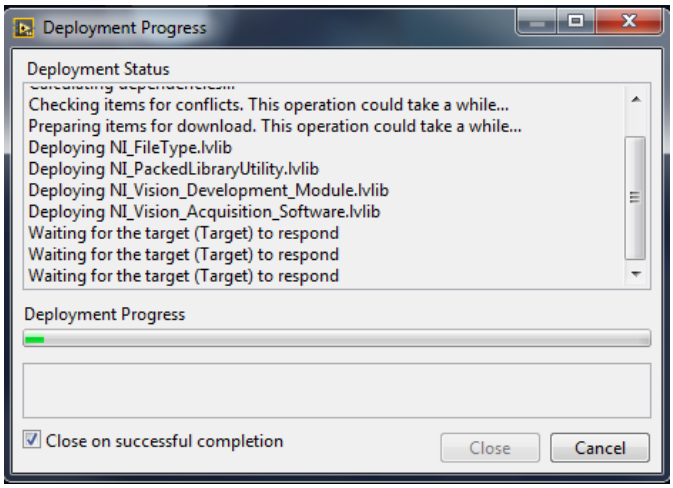
Third Party Resources

- FRC Control and Trajectory Library
- Secret Book Of FRC LabVIEW 2

13.2.2 Esperando que el objetivo responda - Recuperándose de bucles defectuosos

Nota: If you download LabVIEW code which contains an unconstrained loop (a loop with no delay) it is possible to get the roboRIO into a state where LabVIEW is unable to connect to download new code. This document explains the process required to load new, fixed, code to recover from this state.

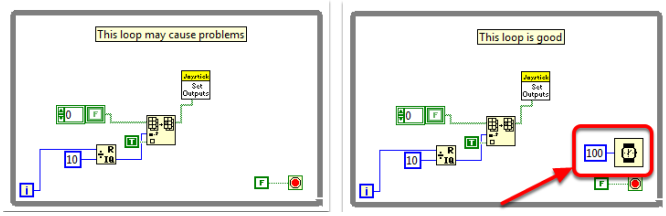
El síntoma



El síntoma principal de este problema es que los intentos de descargar un nuevo código de robot se cuelgan en el paso «Esperando que el objetivo (Target) responda» como se muestra arriba. Tenga en cuenta que hay otras causas posibles de este síntoma (como cambiar de un programa C++Java a un programa LabVIEW) pero los pasos descritos aquí deberían resolver la mayoría o todos ellos.

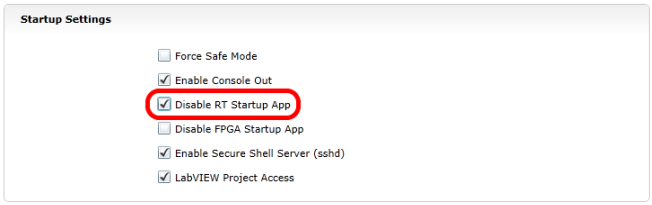
Haga clic en `Cancelar` para cerrar el cuadro de diálogo de descarga.

El problema



Una fuente común de este problema son los bucles sin restricciones en su código de LabVIEW. Un bucle sin restricciones es un bucle que no contiene ningún elemento de retardo (como el de la izquierda). Si no está seguro de dónde empezar a buscar, Disabled.VI, Periodic Tasks.VI y Vision Processing.VI son las ubicaciones comunes para este tipo de bucle. Para solucionar el problema con el código, agregue un elemento de retardo como Wait (ms) VI de la paleta Timing, que se encuentra en el bucle derecho.

Establecer ninguna aplicación

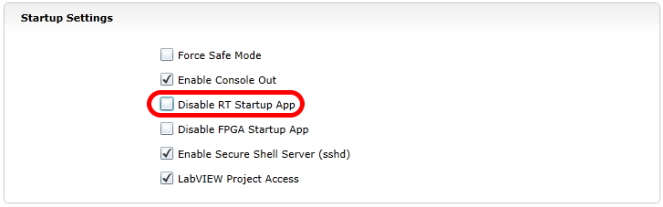


Usando el servidor web roboRIO (consulte el artículo sobre *roboRIO Web Dashboard Startup Settings* para obtener más detalles) **Marque** la casilla para «Desactivar la aplicación de inicio RT».

Reiniciar

Reinicia el roboRIO, ya sea usando el botón Restablecer en el dispositivo o haciendo clic en Reiniciar en la esquina superior derecha de la página web.

Borrar ninguna aplicación



Usando el servidor web roboRIO (consulte el artículo sobre: *roboRIO Web Dashboard Startup Settings* para obtener más detalles) **Desmarque** la casilla para «Desactivar la aplicación de inicio RT».

Cargar código de LabVIEW

Cargue el código de LabVIEW (ya sea usando el botón Ejecutar o Ejecutar como inicio). Asegúrese de configurar el código de LabVIEW en Ejecutar como inicio antes de reiniciar el roboRIO o deberá seguir las instrucciones anteriores nuevamente.

13.2.3 Cómo Alternar Entre Dos Modos de Cámara

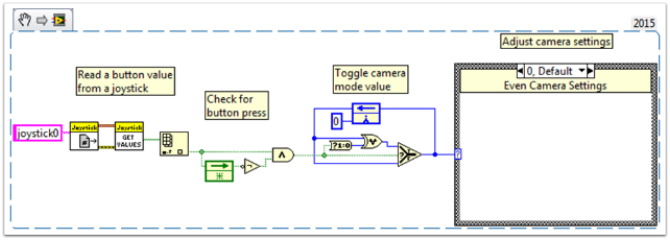
Este código muestra cómo usar un botón para alternar entre dos modos de cámara distintos. El código consta de cuatro etapas.

En la primera etapa, se lee el valor de un botón del joystick.

A continuación, la lectura actual se compara con la lectura anterior usando un **Feedback Node** y algo de aritmética booleana. Juntos, garantizan que el modo de cámara solo se cambie cuando se presiona el botón inicialmente en lugar de alternar hacia adelante y hacia atrás varias veces mientras se mantiene presionado el botón.

Después de eso, el modo de cámara se cambia enmascarando el resultado de la segunda etapa sobre el valor del modo de cámara actual. Esto se llama enmascaramiento de bits y al hacerlo con la función **XOR**, el código cambiará el modo de cámara cuando la segunda etapa regrese a verdadero y no hará nada de otra manera.

Finalmente, puede insertar el código para cada modo de cámara en la estructura del caso al final. Cada vez que se ejecuta el código, esta sección ejecutará el código para el modo de cámara actual.



13.2.4 Ejemplos y Tutoriales de WPILib

Tutoriales Populares

Tutorial de Movimiento Autónomo Temporal

- Mueva su robot autónomamente en base a diferentes intervalos de tiempo
- Vea más en Movimiento Autónomo

Tutorial de Control de Motor Básico

- Configure el hardware y software del motor roboRIO
- Aprenda a configurar el Sistema de Control de FRC® y el Proyecto de Robot de FRC
- Vea más en Control del Motor

Tutorial de Procesamiento de Imagen

- Aprenda técnicas básicas de Procesamiento de Imagen y cómo usar la Asistencia de Visión NI

- Vea más en Procesamiento de Cámaras e Imagen

Tutorial de Control PID

- ¿Qué es el Control PID y cómo se puede implementar?

Tutorial de Comandos y Control

- ¿Qué es un Comando y Control?
- ¿Cómo lo puedo implementar?

Tutorial de la Driver Station

- Conozca la Driver Station de FRC

Tutorial del Modo Prueba

- Aprenda a configurar el Modo Prueba

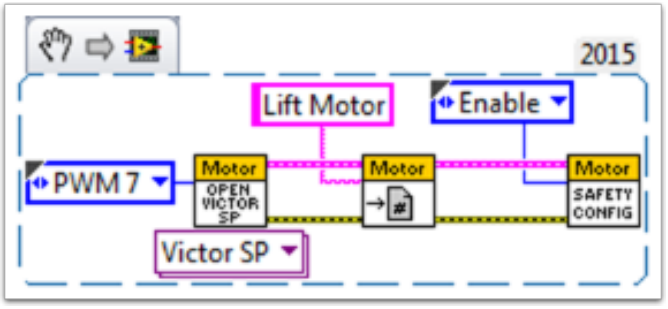
¿Está buscando más ejemplos y discusiones? Busque en más documentos o publique su propia duda, ejemplo de código, o tutorial al hacer clic aquí! ¡No olvide marcar sus mensajes con una etiqueta!

13.2.5 Agregar un Motor Independiente a un Proyecto

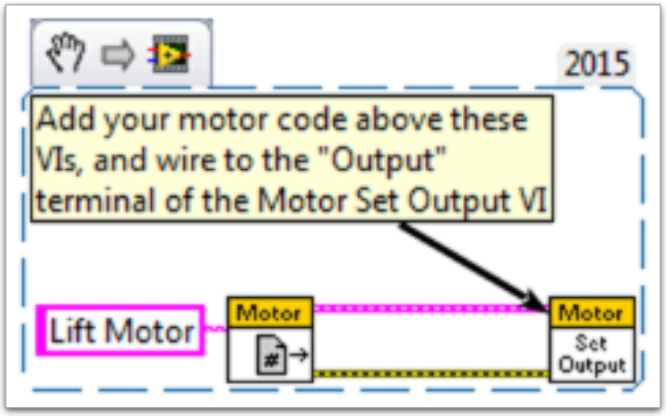
Una vez que la unidad que controla las ruedas esté configurada, es posible que deba agregar un motor adicional para controlar algo completamente independiente de las ruedas, como un brazo. Dado que este motor no será parte de su tanque, arcade o unidad mecanum, definitivamente querrá un control independiente del mismo.

Estos VI Snippets muestran cómo configurar un solo motor en un proyecto que ya puede contener un drive de múltiples motores. Si ve el símbolo HAND>ARROW>LABVIEW, simplemente arrastre la imagen a su diagrama de bloques y listo: ¡código! Ok, así es como lo haces.

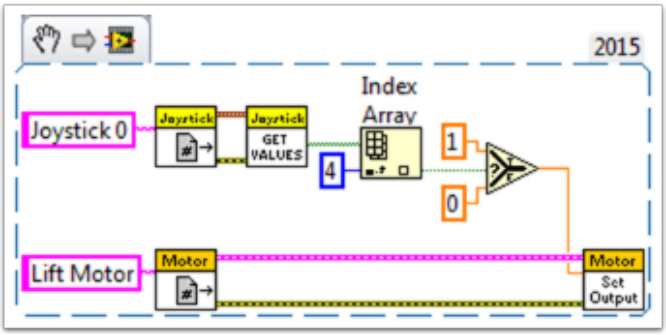
FIRST, create a motor reference in the **Begin.vi**, using the **Motor Control Open VI** and **Motor Control Refnum Registry Set VI**. These can be found by right-clicking in the block diagram and going to **WPI Robotics Library>>RobotDrive>>Motor Control**. Choose your *PWM* line and name your motor. 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.)



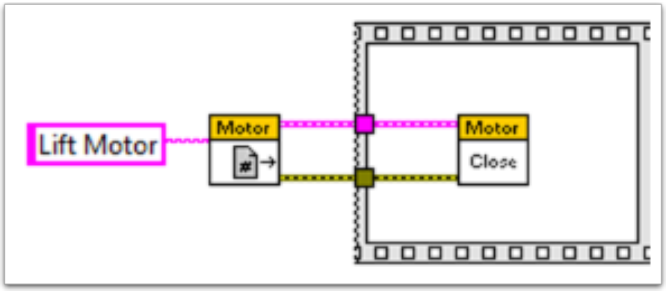
Ahora, haga referencia a su motor (el nombre debe ser exacto) en **Teleop.vi** usando el **Motor Control Refnum Registry Get VI** y dígale qué hacer con el **Motor Control Set Output VI**. Estos se encuentran en el mismo lugar que los VI anteriores.



Por ejemplo, el siguiente snippet le dice al motor de elevación que se mueva hacia adelante si se presiona el botón 4 en el joystick 0 y que permanezca inmóvil en caso contrario. Para mí, el botón 4 es el bumper izquierdo de mi controlador estilo Xbox («Joystick 0»). Para opciones de botones de joystick mucho más detalladas, consulte *Cómo usar los botones de joystick para controlar motores o solenoides*.



Finalmente, necesitamos cerrar las referencias en **Finish.vi** (al igual que hacemos con el drive y el joystick), usando **Motor Control Refnum Registry Get VI** y **Motor Control Close VI** . Si bien esta imagen muestra el Close VI en una estructura de secuencia plana por sí mismo, realmente queremos todos los Close VI en el mismo cuadro. Puede colocar estos dos VIs debajo de los otros Get VIs y Close VIs (para el joystick y el drive).

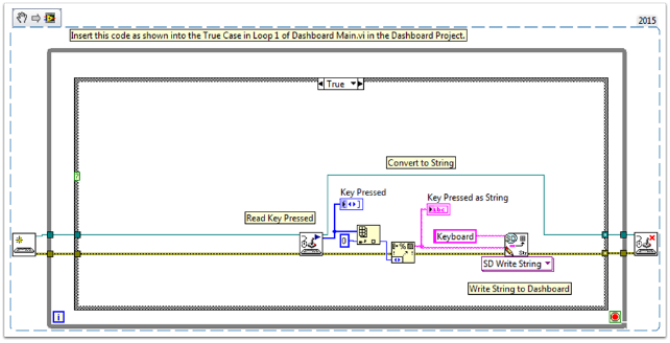


¡Espero que esto te ayude a programar el mejor robot de todos los tiempos! ¡Buena suerte!

13.2.6 Navegación por teclado con la roboRIO

Este ejemplo proporciona algunas sugerencias para controlar el robot mediante la navegación por teclado en lugar de un joystick u otro controlador. En este caso, usamos las teclas A, W, S y D para controlar dos motores en una configuración de tipo tanque.

El primer VI Snippet es el código que deberá incluirse en el Dashboard Main VI. Puede insertar este código en el caso Verdadero del Loop 1. El código abre una conexión con el teclado antes de que comience el bucle, y en cada iteración lee la tecla presionada. Esta información se convierte en una cadena, que luego se pasa al VI Teleop en el proyecto del robot. Cuando el Loop 1 deja de funcionar, la conexión al teclado se cierra.

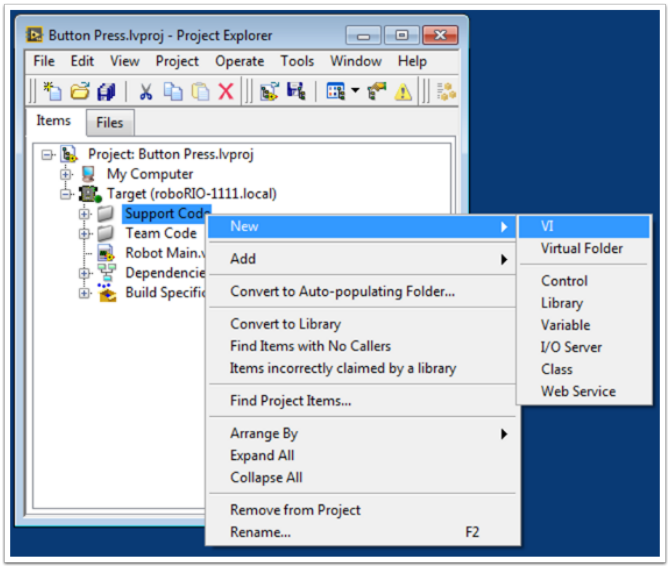


El segundo VI Snippet es un código que debe incluirse en Teleop VI. Esto lee el valor de la cadena del Dashboard que indica qué tecla se presionó. A continuación, una estructura Case determina qué valores deben escribirse en los motores izquierdo y derecho, según la tecla. En este caso, W es hacia adelante, A es hacia la izquierda, D es hacia la derecha y S es hacia atrás. Cada caso en este ejemplo hace funcionar los motores a la mitad de la velocidad. Puede mantener esto igual en su código, cambiar los valores o agregar código adicional para permitir que el driver ajuste la velocidad, para que pueda conducir rápido o lento según sea necesario. Una vez que se seleccionan los valores del motor, se escriben en los drive motors, y los valores del motor se publican en el dashboard.

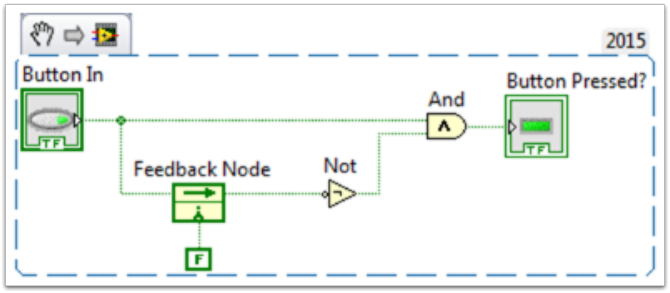
13.2.7 Hacer una pulsación de botón de un disparo

Cuando utilice la función Obtener valores del joystick, al presionar un botón del joystick, el botón se leerá VERDADERO hasta que se suelte el botón. Esto significa que lo más probable es que lea varios valores VERDADEROS para cada pulsación. ¿Qué sucede si desea leer solo un valor VERDADERO cada vez que se presiona el botón? A esto se le suele llamar «One-Shot Button». El siguiente tutorial le mostrará cómo crear un subVI que puede colocar en su Teleop.vi para hacer esto.

Primero, cree un nuevo VI en la carpeta Support Code de su proyecto.



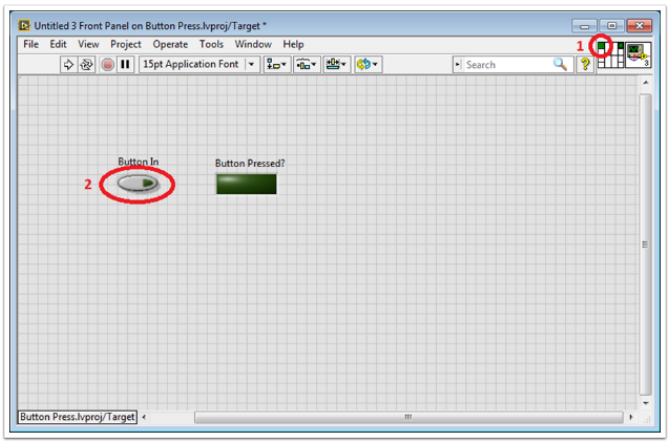
Ahora, en el diagrama de bloques del nuevo VI, coloque el siguiente fragmento de código.



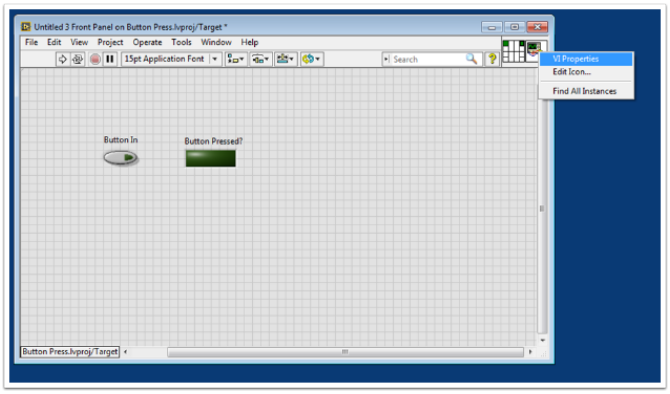
Este código utiliza una función llamada Nodo de retroalimentación. Hemos cableado el valor actual del botón en el lado izquierdo del nodo de retroalimentación. El cable que sale de la flecha del nodo de retroalimentación representa el valor anterior del botón. Si la flecha en su nodo de retroalimentación va en la dirección opuesta como se muestra aquí, haga clic derecho para encontrar la opción para invertir la dirección.

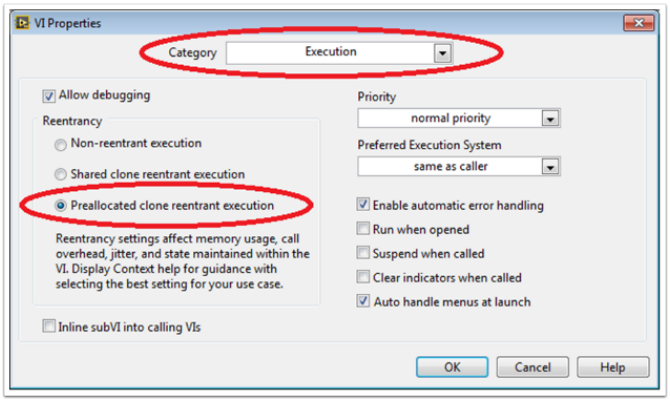
Cuando se presiona un botón, el valor del botón pasa de FALSE a TRUE. Queremos que la salida de este VI sea VERDADERA solo cuando el valor actual del botón sea VERDADERO y el valor anterior del botón sea FALSO.

A continuación, necesitamos conectar el control booleano y el indicador a las entradas y salidas del VI. Para hacer esto, primero haga clic en el bloque en el panel del conector, luego haga clic en el botón para conectar los dos (vea el diagrama a continuación). Repita esto para el indicador.

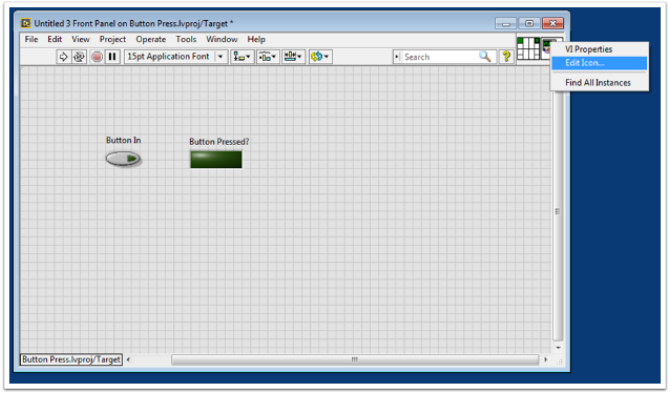


A continuación, necesitamos cambiar las propiedades de este VI para que podamos usar múltiplos de este VI en nuestro TeleOp.vi. Haga clic derecho en el icono VI y vaya a Propiedades VI. Luego seleccione la categoría «Execution» y seleccione «Preallocated clone reentrant execution».



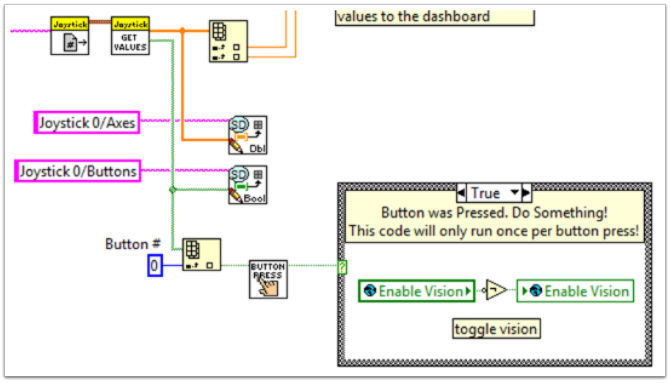


Por último, deberíamos cambiar el icono VI para que sea más descriptivo de la función del VI. Haga clic derecho en el icono y vaya a Editar icono. Crea un nuevo icono.



Finalmente, guarde el VI con un nombre descriptivo. Ahora puede arrastrar y soltar este VI desde la carpeta Support Files a su TeleOp.vi. Aquí hay una copia del VI completo: `Button_Press.vi`

Aquí hay un ejemplo de cómo podría usar este VI.



13.2.8 Agregar funciones de seguridad a su código de robot

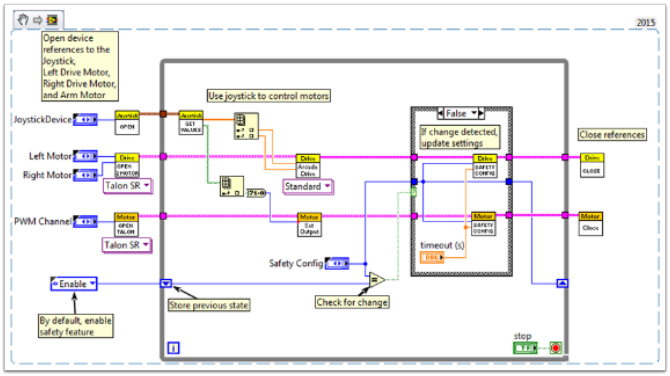
Un problema común con los proyectos complejos es asegurarse de que todo su código se esté ejecutando cuando lo espera. Pueden surgir problemas cuando las tareas con alta prioridad, tiempos de ejecución prolongados o llamadas frecuentes acaparan la potencia de procesamiento del roboRIO. Esto conduce a lo que se conoce como «starvation» para las tareas que no se pueden ejecutar debido a que el procesador está ocupado. En la mayoría de los casos, esto simplemente ralentizará el tiempo de reacción a su entrada de los joysticks y otros dispositivos. Sin embargo, esto también puede hacer que los motores de accionamiento de su robot permanezcan encendidos mucho tiempo después de que intente detenerlos. Para evitar cualquier catástrofe robótica a causa de esto, puede implementar funciones de seguridad que verifiquen la falta de entrada de tareas y apague automáticamente las operaciones potencialmente dañinas.

Hay funciones integradas para los motores que permiten una fácil implementación de los controles de seguridad. Estas funciones son:

- Configuración de seguridad del accionamiento del robot
- Configuración de seguridad del accionamiento del motor
- Configuración de seguridad de la retransmisión
- *PWM* Safety Configuration

- Configuración de seguridad del solenoide
- Seguridad de actualización y retardo del accionamiento del robot

En todas las funciones de configuración de seguridad, puede habilitar y deshabilitar las comprobaciones de seguridad mientras se ejecuta la programación y configurar el tiempo de espera que considere apropiado. Las funciones mantienen un caché de todos los dispositivos que tienen la seguridad habilitada y comprobarán si alguno de ellos ha superado su límite de tiempo. Si alguno lo ha hecho, todos los dispositivos en la caché se desactivarán y el robot se detendrá inmediatamente o se apagarán sus salidas de retransmisión / PWM / solenoide. El código a continuación demuestra cómo usar las funciones de configuración de seguridad del variador para establecer un límite de tiempo máximo en el que los motores no recibirán ninguna entrada antes de apagarse.



Para probar el apagado de seguridad, intente agregar una función de espera al ciclo que sea más largo que su tiempo de espera.

La función final que se relaciona con la implementación de verificaciones de seguridad, el retardo de la unidad de robot y la actualización de seguridad, le permite poner el roboRIO en modo autónomo sin exceder el límite de tiempo. Mantendrá la salida actual del motor sin realizar costosas llamadas a las funciones de salida del variador y también se asegurará de que las comprobaciones de seguridad se actualicen periódicamente para que los motores no se detengan repentinamente.

En general, es muy recomendable que se implemente algún tipo de control de seguridad en su proyecto para asegurarse de que su robot no quede involuntariamente en un estado peligroso.

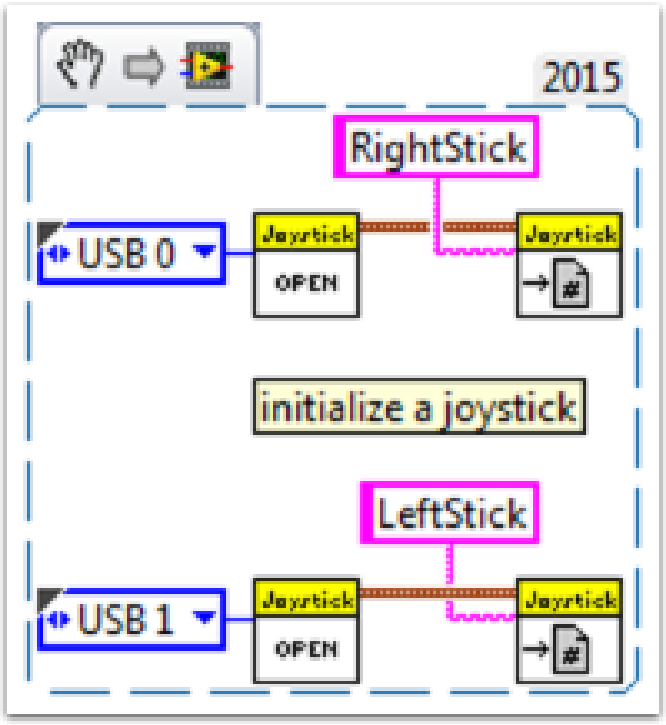
13.2.9 Cómo utilizar los botones del Joystick para controlar motores o solenoides

A medida que todos ponemos en funcionamiento nuestros sistemas, pasaremos a conectar nuestros dispositivos auxiliares, como motores y solenoides. Con esto, generalmente usaremos botones de joystick para controlar estos dispositivos. Para comenzar con esto, veremos varias formas de controlar dispositivos con botones del joystick.

¿Sabía que puede hacer clic y arrastrar un fragmento de VI desde un documento como este directamente a su código de LabVIEW? Pruébelo con los fragmentos de este documento.

Preparación:

Independientemente de la configuración, deberá agregar uno, dos o más (si está realmente emocionado) joysticks al «Begin.vi». El primer ejemplo usa 2 joysticks y los otros solo usan uno. Dé a cada uno un nombre único para que podamos usarlo en otros lugares, como el fragmento de abajo. Los llamé «LeftStick» y «RightStick» porque están en los lados izquierdo y derecho de mi escritorio. Si sus joysticks ya están configurados, ¡genial! Puede omitir este paso.

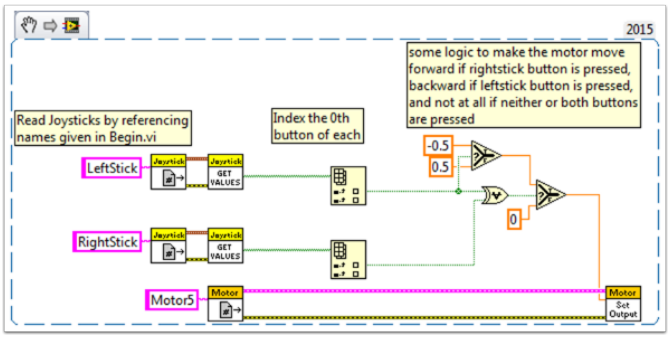


El resto del código en este documento se colocará en el «Teleop.VI» Aquí es donde estaremos programando nuestros botones de joystick para controlar diferentes aspectos de nuestros motores o solenoides.

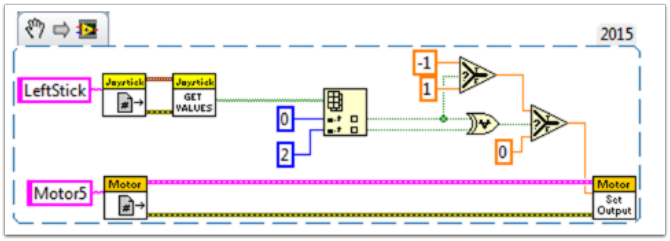
escenario 1

«Quiero que un motor se mueva en una dirección cuando presione un botón y en la otra cuando presione un botón diferente».

Este código usa el botón 0 en dos joysticks diferentes para controlar el mismo motor. Si se presiona el botón 0 en LeftStick, el motor se mueve hacia atrás, y si se presiona el botón 0 en RightStick, el motor avanza. Si se presionan ambos botones o no se presiona ninguno, el motor no se mueve. Aquí llamé a mi motor de referencia «Motor5», pero puedes nombrar tu motor como quieras en el «Begin.vi»



Es posible que desee utilizar varios botones del mismo joystick para su control. Para ver un ejemplo de esto, mire el siguiente fragmento de VI o el fragmento de VI en el Escenario 2.



Aquí usé los botones de joystick 0 y 2, pero siéntase libre de usar los botones que necesite.

Escenario 2

«Quiero que los diferentes botones del joystick se muevan a distintas velocidades».

Este ejemplo podría ser útil si necesita que un motor haga diferentes cosas según los botones que presione. Por ejemplo, digamos que mi joystick tiene un gatillo (botón 0) y 4 botones en la parte superior (botones 1 a 4). En este caso, los siguientes botones deben tener las siguientes funciones:

- botón 1 - retroceder a la mitad de la velocidad
- botón 2 - avanzar a la mitad de la velocidad
- botón 3 - retroceder a 1/4 de velocidad
- botón 4 - avanzar a 1/4 de velocidad
- gatillo - ¡adelante a toda velocidad! (avanzar a toda velocidad)

Luego tomaríamos la matriz booleana de «JoystickGetValues.vi» y la conectaríamos a un nodo «Boolean Array to Number» (Paleta numérica de conversión de paleta). Esto convierte la matriz booleana en un número que podemos usar. Conecte este número a una estructura case.

Cada caso corresponde a una representación binaria de los valores de la matriz. En este ejemplo, cada caso corresponde a una combinación de un botón. Agregamos seis casos: 0 (todos los botones apagados), 1 (botón 0 encendido), 2 (botón 1 encendido), 4 (botón 2 encendido), 8 (botón 3 encendido) y 16 (botón 4 encendido). Observe que omitimos el valor 3. 3 correspondería a los botones 0 y 1 presionados al mismo tiempo. No definimos esto en nuestros requisitos, por lo que dejaremos que el caso predeterminado lo maneje.

Puede ser útil revisar el documento de ayuda de estructura Case de LabVIEW 2014 aquí:

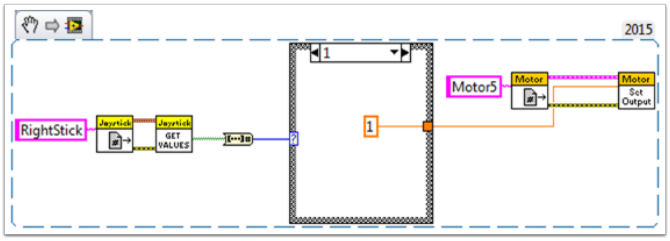
https://zone.ni.com/reference/en-XX/help/371361L-01/glang/case_structure/

También hay 3 tutoriales comunitarias sobre estructuras de casos aquí:

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

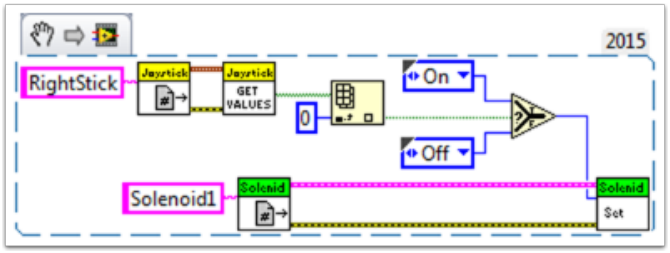


Dado que nuestros requisitos eran simples, solo necesitamos una única constante en cada caso. Para el caso 1 (completo adelante) usamos un 1, para el caso 2 (medio reverso) usamos -0.5, etc. Podemos usar cualquier valor constante entre 1 y -1. Dejé el caso 0 como predeterminado, por lo que si se presionan varios botones (se alcanzó cualquier estado indefinido) el motor se detendrá. Por supuesto, usted puede personalizar estos estados como quiera.

Escenario 3

«Quiero controlar un solenoide con los botones de mi joystick».

A estas alturas, estamos familiarizados con cómo el joystick genera los botones en una matriz de valores booleanos. Necesitamos indexar esta matriz para obtener el botón que nos interesa y conectar este booleano a un nodo seleccionado. Dado que el «Solenoide Set.vi» requiere un Enum como entrada, la forma más fácil de obtener el enum es hacer clic con el botón derecho en la entrada «Valor» del «Solenoide Set.vi» y seleccionar «Crear constante». Duplique esta constante y conecte una copia al terminal Verdadero y otra al terminal Falso del nodo seleccionado. Luego conecte la salida del nodo de selección a la entrada «Valor» del solenoide VI.



¡Feliz Roboteo!

13.2.10 Variables locales y globales en LabVIEW para FRC

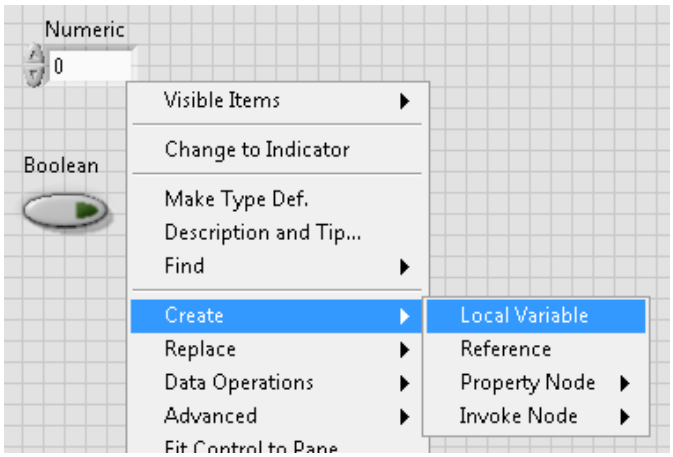
Este ejemplo sirve como una introducción a las variables locales y globales, cómo se utilizan en el proyecto de robot por defecto de LabVIEW para FRC® y cómo podría querer utilizarlas en su proyecto.

Las variables locales y las variables globales se pueden usar para transferir datos entre ubicaciones dentro del mismo VI (variables locales) o dentro de diferentes VI (variables globales), rompiendo el convencional ` Paradigma de Data Flow <https://www.ni.com/getting-started/labview-basics/dataflow>`__ por el cual LabVIEW es famoso. Por lo tanto, pueden ser útiles cuando, por cualquier motivo, no puede conectar el valor directamente el nodo a otro.

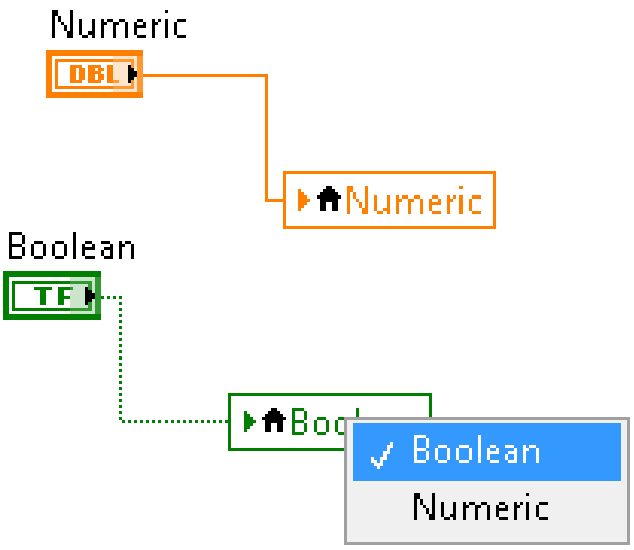
Nota: Una posible razón puede ser que necesite pasar datos entre iteraciones de bucle consecutivas; Miro_T cubrió esto en éste post. También debe tenerse en cuenta que el feedback node en LabVIEW puede usarse como un equivalente al registro de desplazamiento, ¡aunque ese puede ser un tema para otro día!

Introducción a las variables locales y globales

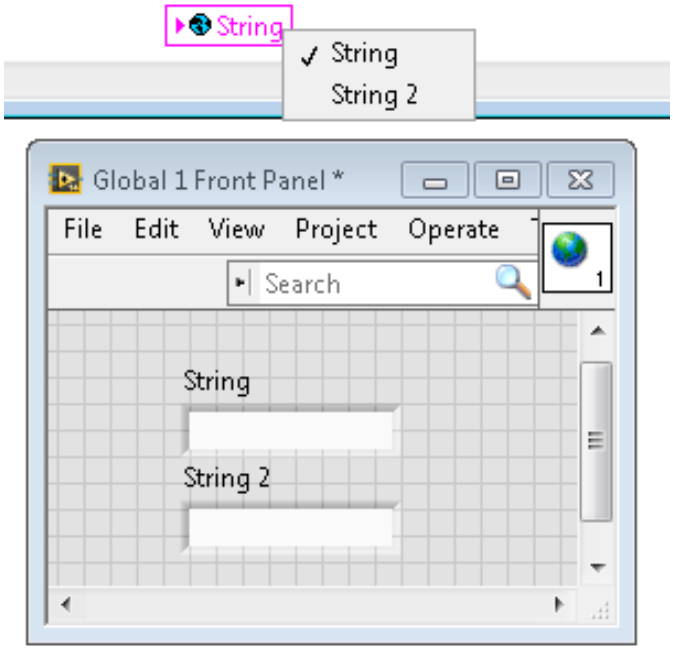
Las variables locales se pueden usar dentro del mismo VI. Cree una variable local haciendo clic con el botón derecho en un control o indicador en su panel frontal:



También puede crear una variable local desde la paleta Estructuras en el diagrama de bloques. Cuando tiene múltiples variables locales en un VI, puede hacer clic izquierdo para elegir qué variable es:



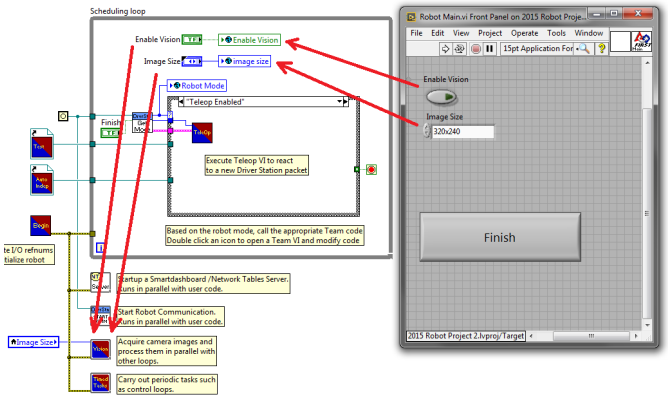
Las variables globales se crean de forma ligeramente diferente. Agregue uno al diagrama de bloques de la paleta Estructuras y observe que cuando hace doble clic en él, se abre un panel frontal separado. Este panel frontal no tiene un diagrama de bloques, pero agrega tantas entidades al panel frontal como desee y lo guarda como un archivo *.Vi:



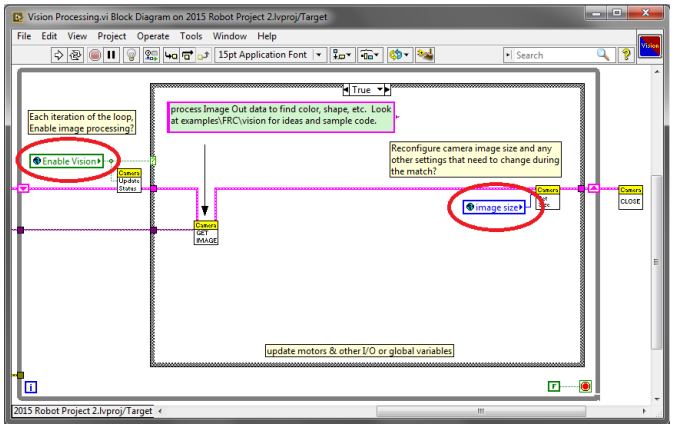
Nota: ¡Tenga mucho cuidado de evitar condiciones de carrera al usar variables locales y globales! Básicamente, asegúrese de no escribir accidentalmente en la misma variable en varias ubicaciones sin una forma de saber en qué ubicación se escribió por última vez. Para obtener una explicación más detallada, consulte este documento

Cómo se utilizan en el LabVIEW predeterminado para el proyecto de robot FRC

Las variables globales para «Enable Vision» e «Image Size» se escriben durante cada iteración del Robot Main VI ...



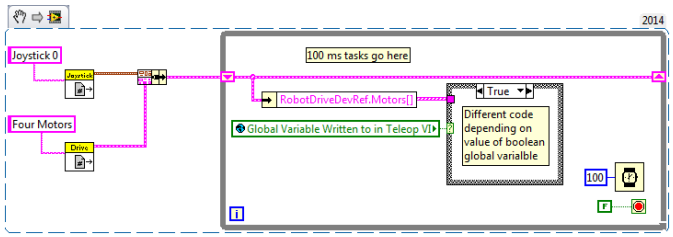
... Y luego lea en cada iteración del Vision Processing VI:



Esto permite al usuario, al implementar en Robot Main VI desde el entorno de desarrollo de LabVIEW, habilitar / deshabilitar la visión y cambiar el tamaño de la imagen desde el panel frontal de Robot Main.

¿Cómo puede utilizarlos en su proyecto?

Consulte el diagrama de bloques del VI Periodic Tasks. Quizás haya algún valor, como un booleano, que se pueda escribir en una variable global en el VI Teleop y luego leer en el VI Periodic Tasks. Luego puede decidir qué código o valores usar en el VI Periodic Tasks, dependiendo de la variable global booleana:



13.2.11 Usando el Compresor en LabVIEW

Este fragmento muestra cómo configurar su proyecto roboRIO para usar el Módulo de control neumático (PCM). El PCM arranca y detiene automáticamente el compresor cuando se miden presiones específicas en el tanque. En su programa roboRIO, deberá agregar los siguientes VI.

Para obtener más información, consulte los siguientes enlaces:

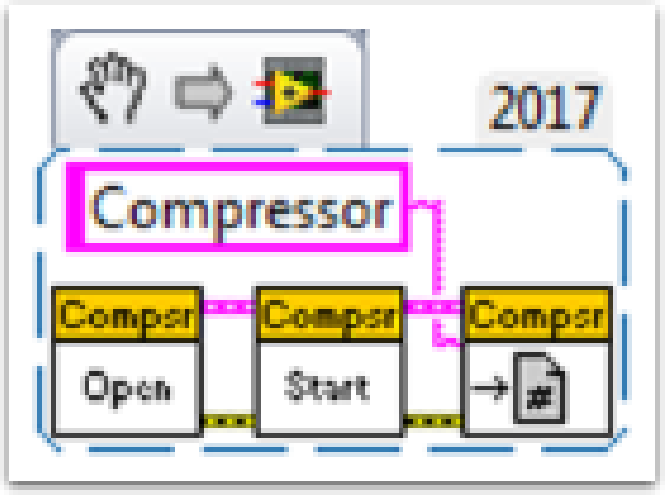
`Manual de neumática para FRC`

PCM User's Guide

Neumática paso a paso para la roboRIO

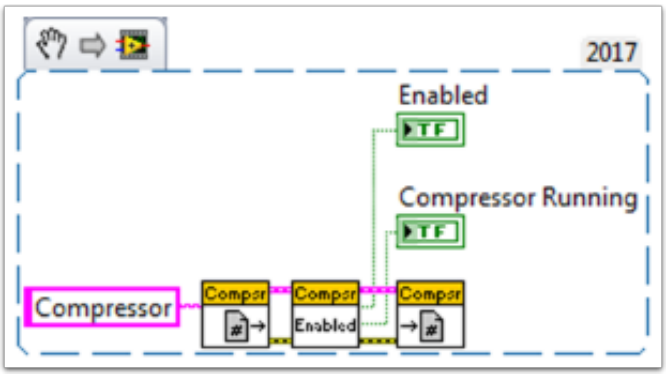
Begin VI

Coloque este fragmento en Begin.vi.



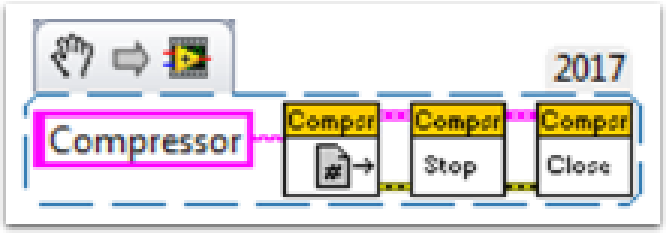
Teleop VI

Coloque este fragmento en Teleop.vi. Esta parte solo es necesaria si utiliza las salidas para otros procesos.



Finish VI

Coloque este fragmento en Close Refs, guarde datos, etc. marco del Finish.vi.



14

FRC Python Programming

14.1 pyproject.toml usage

Nota: 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"
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    # "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
```

Nota: 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

Nota: 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

Advertencia: 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.

Ver también:

Visualización de la salida de la consola

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
```

Nota: 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 Hardware APIs

Esta sección habla del control de motores y neumáticos a través de controladores de velocidad, solenoides y neumáticos, y su interfaz con Java y C++ WPILib.

15.1 Motors APIs

Programming your motors are absolutely essential to a moving robot! This section showcases some helpful classes and examples for getting your robot up and moving!

15.1.1 Uso de controladores de motor en código

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 *Uso de las clases de WPILib para conducir su robot*.

Uso de controladores de motor 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 *Controladores de motor PWM en profundidad*. 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.

Nota: 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
```

Controladores de motor CAN

A handful of CAN motor controllers are available through vendors such as CTR Electronics, REV Robotics, and Playing with Fusion. See *Dispositivos CAN de terceros*, *Bibliotecas de 3ros*, and *Third Party Example Projects* for more information.

15.1.2 Controladores de motor PWM en profundidad

Consejo: WPLib tiene un amplio soporte para el control de motores. Hay una serie de clases que representan diferentes tipos de controladores para motor y servos. Actualmente existen dos clases de controladores de motor, los basados en PWM y los basados en CAN. WPLib también contiene clases compuestas (como DifferentialDrive) que permite el controlar varios objetos con un solo objeto. Este artículo cubrirá los detalles de los controladores de motor PWM; los controladores CAN y las clases compuestas se tratarán artículos separados.

Controladores PWM, breve teoría de operación

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).

La señal PWM que usan los controladores para una entrada es un poco diferente. Incluso en los límites del rango de señal (avance máximo o retroceso máximo) la señal nunca se acerca a un ciclo de trabajo de 0% o 100%. En cambio, los controladores usan una señal con un período de 5 ms o 10 ms y un ancho del pulso del punto medio de 1.5ms. Muchos de los controladores usan el típico controlador RC de 1ms a 2ms.

Valores de salida Sin Procesar vs Escalados

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.

Calibración de controladores de motor

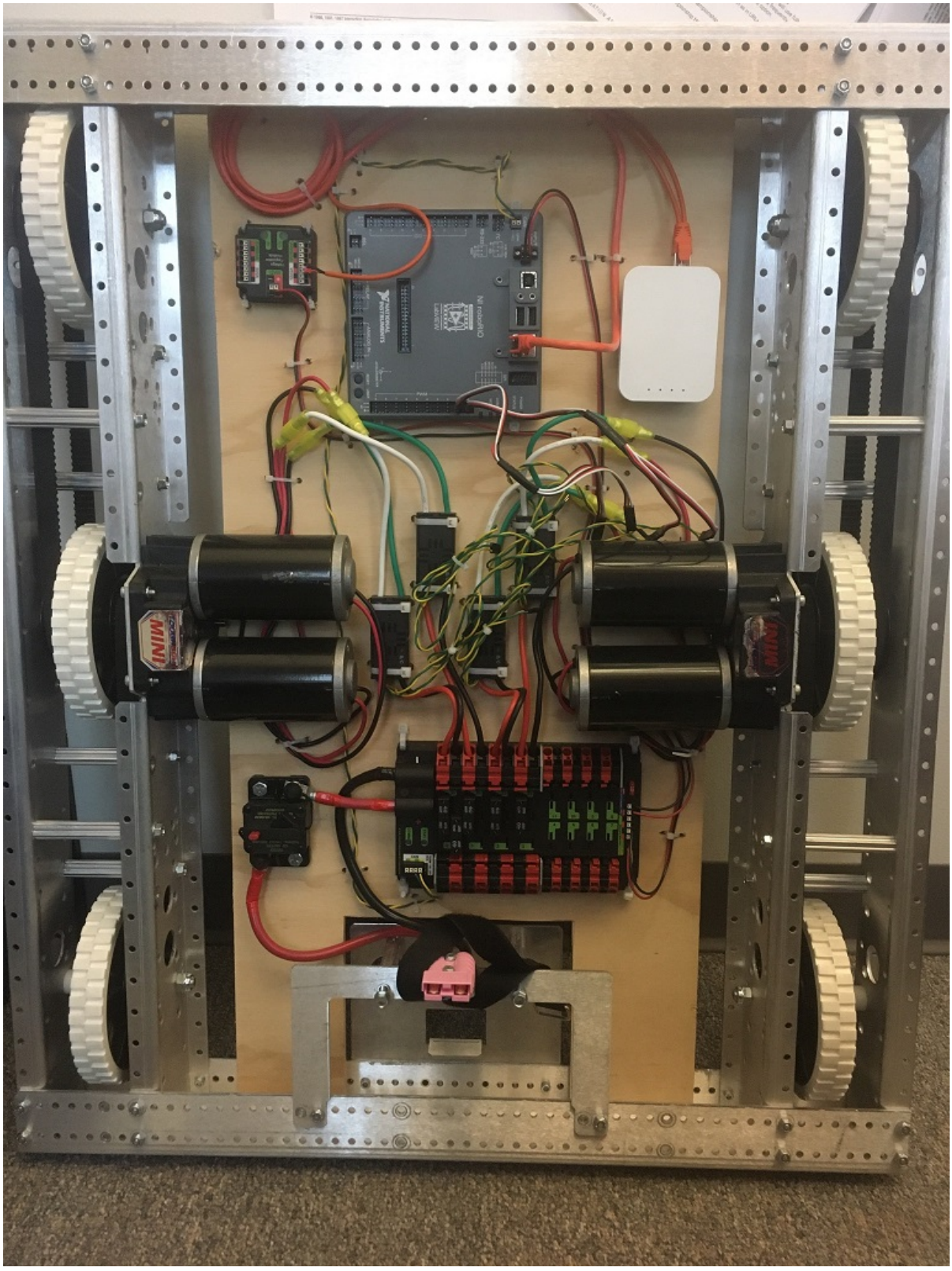
Entonces, sí WPLib se encarga de todo este escalado, ¿Por qué tendrías que calibrar el controlador de motor? Los valores que usa WPLib para el escalado son aproximados, basados en la medición de un número de muestras de cada tipo de controlador. Debido a una variedad de factores, la sincronización de un controlador de motor individual puede variar ligeramente. Para eliminar definitivamente el «zumbido» (señal de punto medio interpretada como un ligero movimiento en una dirección) y conducir el controlador hasta cada extremo, se recomienda calibrar los controladores. En general, el procedimiento de calibración para cada controlador consiste en poner el controlador en modo en modo de calibración y luego conducir la señal de entrada a cada extremo, y luego volver al punto medio. Para ver ejemplos de cómo utilizar estos controladores de motor en su código, consulte *Using Motor Controllers in Code/Using PWM Motor Controllers*

15.1.3 Uso de las clases de WPILib para conducir su robot

WPILib incluye muchas clases para ayudar a que su robot conduzca más rápido.

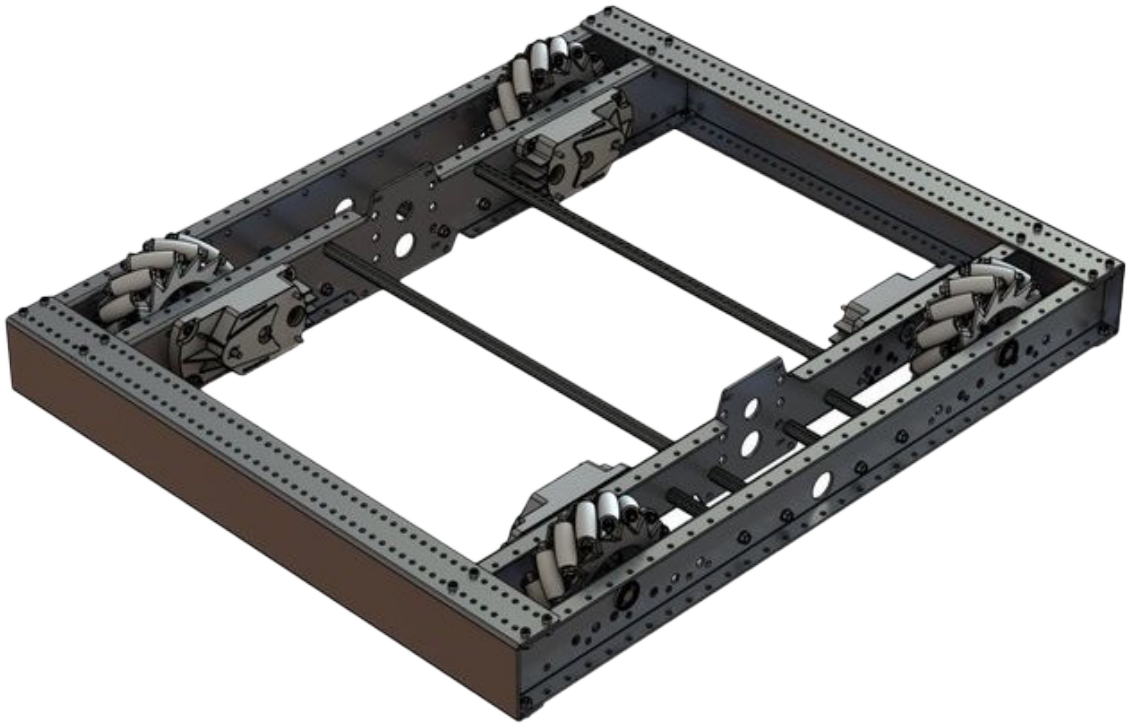
Chasis estándar

Robots con chasis diferencial



Estas bases de transmisión suelen tener dos o más ruedas en línea u omnidireccionales por lado (p. Ej., 6WD u 8WD) y también pueden conocerse como «skid-steer», «tank drive» o «West Coast Drive». La transmisión del kit de piezas es un ejemplo de transmisión diferencial. Estas transmisiones son capaces de conducir hacia adelante / hacia atrás y puede girar conduciendo los dos lados en direcciones opuestas causando que las ruedas patinen de lado. Estas transmisiones no son capaces de moverse de manera lateral.

Chasis mecanum



Chasis Mecanum es un método de conducción que utiliza ruedas especialmente diseñadas que permiten al robot conducir en cualquier dirección sin cambiar la orientación del robot. Un robot con una transmisión convencional (todas las ruedas apuntando en la misma dirección) debe girar en la dirección que se necesita conducir. Un robot mecanum puede moverse en cualquier dirección sin girar primero y se llama impulso holonómico. Las ruedas (que se muestran en este robot) tienen rodillos que causan las fuerzas de conducir para ser aplicado en un ángulo de 45 grados en lugar de en línea recta como en el caso de un accionamiento convencional.

Cuando se ve desde la parte superior, los rodillos de una transmisión mecanum deben formar un patrón de «X». Esto da como resultado los vectores de fuerza (al conducir la rueda hacia adelante) en las dos ruedas delanteras apuntando hacia adelante y hacia adentro y las dos ruedas traseras apuntando hacia adelante y hacia afuera. Girando las ruedas en diferentes direcciones, varios componentes de los vectores de fuerza se cancelan, lo que resulta en el movimiento deseado del robot. Un gráfico rápido de diferentes movimientos se proporciona a continuación, extraer los vectores de fuerza para cada uno de estos movimientos puede ayudar en entender cómo funcionan estas transmisiones. Variando las velocidades de las ruedas, además a la dirección, los movimientos se pueden combinar dando como resultado la traslación en cualquier dirección y rotación, simultáneamente.

Convenciones de clase de conducción

Inversión motora

As of 2022, the right side of the drivetrain is **no longer** inverted by default. It is the responsibility of the user to manage proper inversions for their drivetrain. Users can invert motors by calling `setInverted()`/`SetInverted()` on their motor objects.

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)
```

Cuadrar las entradas

Al conducir robots, a menudo es deseable manipular las entradas del joystick de modo que el robot tiene un control más fino a bajas velocidades mientras sigue usando el rango de salida completo. Una forma de lograr esto es al cuadrar la entrada del joystick y luego volver a aplicar el signo. Por defecto la Differential Drive class cuadrará las entradas. Si esto no se desea (por ejemplo, si pasa valores desde un PID Controller), utilice uno de los métodos de accionamiento con el parámetro squaredInputs y configúrelo en falso.

Deadband de entrada

Por defecto, la clase de Differential Drive aplica una banda muerta/deadband de entrada de 0.02. Esto significa que la entrada los valores con una magnitud inferior a 0.02 (después de cualquier cuadrado como se describe anteriormente) se establecerán en 0. En la mayoría de los casos, estas entradas pequeñas resultan del centrado imperfecto del joystick y no son suficientes para provocar el movimiento de la transmisión, la banda muerta ayuda a reducir el calentamiento innecesario del motor que puede resultar de aplicar estos pequeños valores a la transmisión. Para cambiar la banda muerta, use el método setDeadband().

Salida máxima

A veces, los drivers sienten que su tren motriz conduce demasiado rápido y quieren limitar la salida. Esto se puede lograr con el método *setMaxOutput()*. Esta salida máxima se multiplica por el resultado de las funciones de accionamiento anteriores, como entradas cuadradas y de banda muerta.

Motor Safety

Motor Safety es un mecanismo en WPILib que toma el concepto de un perro guardián/watchdog y lo rompe a un perro guardián/watchdog (temporizador de seguridad del motor) para cada actuador individual. Tenga en cuenta que este mecanismo de protección es adicional al System Watchdog que es controlado por la red Código de comunicaciones y FPGA y deshabilitará todas las salidas del actuador si no recibe un paquete de datos válido para 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 característica de Seguridad del Motor opera manteniendo un temporizador que rastrea cuánto tiempo ha pasado desde que el método feed() ha sido llamado para ese actuador. El código en la clase Driver Station inicia una comparación de estos temporizadores con los valores de tiempo de espera para cualquier actuador con la seguridad activada cada 5 paquetes recibidos (100ms nominal). Los métodos set() de cada clase de controlador de motor y los métodos set() y setAngle() de la clase servo llaman a feed() para indicar que la salida del actuador ha sido actualizada.

The Motor Safety interface of motor controllers can be interacted with by the user using the following methods:

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()
```

By default all Drive objects enable Motor Safety. Depending on the mechanism and the structure of your program, you may wish to configure the timeout length of the motor safety (in seconds). The timeout length is configured on a per actuator basis and is not a global setting. The default (and minimum useful) value is 100ms.

Convenciones del Eje

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.

Nota: See the *Coordinate System* section for more detail about the axis conventions and coordinate systems.

Uso de la clase DifferentialDrive para controlar robots de accionamiento diferencial

---

Nota: 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 es un método proporcionado para el control de transmisiones “skid-steer” o “West Coast”, como el chasis del Kit de piezas. Crear una instancia de una unidad diferencial es tan simple como esto:

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++ (Encabezado)

```
  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++ (Fuente)

```
  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++ (Encabezado)

```
 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++ (Fuente)

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

Nota: 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

    frontRight = wpilib.Spark(3)
    rearRight = wpilib.Spark(4)
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    right = wpilib.MotorControllerGroup(frontLeft, rearLeft)

    self.drive = wpilib.drive.DifferentialDrive(left, right)
```

Modos de manejo

Nota: La clase DifferentialDrive contiene tres modos predeterminados diferentes para conducir sus motores del robot.

- Tank Drive, que controla los lados izquierdo y derecho de forma independiente
- Arcade Drive, que controla la velocidad de avance y giro
- Curvature Drive, un subconjunto de Arcade Drive, que hace que su robot se maneje como un automóvil con giros de curvatura constante.

La clase DifferentialDrive contiene tres métodos predeterminados para controlar los robots de dirección deslizante o WCD. Tenga en cuenta que puede crear sus propios métodos para controlar la conducción del robot y hacer que llamen a tankDrive() con las entradas derivadas para los motores izquierdo y derecho.

El modo Tank Drive se usa para controlar cada lado del drivetrain de forma independiente (generalmente con un eje de joystick individual que controla cada uno). Este ejemplo muestra cómo usar el eje Y de dos palancas de mando separadas para ejecutar la transmisión en modo Tank. La construcción de los objetos ha sido omitida, para arriba drivetrain y aquí para Joystick.

El modo Arcade Drive se usa para controlar el drivetrain mediante la velocidad / aceleración y la velocidad de la rotación. Esto generalmente se usa con dos ejes de un solo joystick, o dividido en joysticks (a menudo en un solo gamepad) con el acelerador proveniente de una palanca y la rotación de otra. Este ejemplo muestra cómo usar un solo joystick con el modo Arcade. La construcción de los objetos ha sido omitida, para arriba para la construcción de drivetrain y aquí para la construcción de Joystick.

Al igual que Arcade Drive, el modo Curvature Drive se usa para controlar el drivetrain usando velocidad/aceleración y tasa de rotación. La diferencia es que la entrada de control de rotación controla el radio de curvatura en lugar de la tasa de cambio de rumbo, al igual que el volante de un automóvil. Este modo también admite girar en su lugar, que se habilita cuando el tercer parámetro `boolean` es verdadero.

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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    myDrive.curvatureDrive(-driveStick.getY(), -driveStick.getX(), driveStick.
↪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))
```

Uso de la clase MecanumDrive para controlar los robots Mecanum Drive

MecanumDrive es un método proporcionado para el control de transmisiones holonómicas con ruedas Mecanum, como el chasis del kit de piezas con el kit de actualización de la transmisión mecanum, como se muestra arriba. Crear una instancia de MecanumDrive es tan simple como esto:

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)
```

Modos de manejo mecanum

Nota: Las convenciones del eje de accionamiento son diferentes de las convenciones comunes del eje del joystick. Ver las convenciones del eje anteriores para más información.

La clase MecanumDrive contiene dos modos predeterminados diferentes para conducir los motores de su robot.

- driveCartesian: los ángulos se miden en sentido horario desde el eje X positivo. La velocidad de los robots es independiente de su ángulo o velocidad de rotación.
- drivePolar: los ángulos se miden en sentido antihorario desde el frente. La velocidad a que el robot conduce (traslación) es independiente de su ángulo o velocidad de rotación.

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)
```

Conducción con orientación del campo

Se puede suministrar un 4to parámetro al método `driveCartesian(doble ySpeed, doble xSpeed, doble zRotation, doble gyroAngle)`, el ángulo devuelto por un sensor Gyro. Esto ajustará el valor de rotación suministrado. Esto es particularmente útil con mecanum drive dado que, a los efectos de la dirección, el robot realmente no tiene frente, parte posterior o laterales. Puede entrar cualquier dirección. Agregar el ángulo en grados desde un objeto giroscópico hará que el robot se mueva lejos de los conductores cuando el joystick se empuja hacia adelante, y hacia los conductores cuando se tira hacia ellos, independientemente de la dirección que esté mirando el robot.

El uso de la conducción orientada al campo a menudo hace que el robot sea mucho más fácil de conducir, especialmente en comparación con un sistema de accionamiento «orientado al robot» donde los controles se invierten cuando el robot se enfrenta a los conductores.

Solo recuerde obtener el ángulo giroscópico cada vez que se llame a `driveCartesian()`.

Nota: A muchos equipos también les gusta ramificar las entradas de los mandos en el tiempo para promover una aceleración suave y reducir las sacudidas. Esto se puede conseguir con un *Limitador de velocidad de giro*.

15.1.4 Movimiento de baja potencia repetible - Controlador de servos con WPILib

Servo motors are a type of motor which integrates positional feedback into the motor in order to allow a single motor to perform repeatable, controllable movement, taking position as the input signal. WPILib provides the capability to control servos which match the common hobby input specification (Pulse Width Modulation (PWM) signal, 0.6 ms - 2.4 ms pulse width)

Construcción de un objeto servo

JAVA

```
Servo exampleServo = new Servo(1);
```

C++

```
frc::Servo exampleServo {1};
```

PYTHON

```
exampleServo = wpilib.Servo(1)
```

Un objeto servo se construye pasando un canal.

Establecer valores de servo

JAVA

```
exampleServo.set(.5);
exampleServo.setAngle(75);
```

C++

```
exampleServo.Set(.5);
exampleServo.SetAngle(75);
```

PYTHON

```
exampleServo.set(.5)
exampleServo.setAngle(75)
```

Hay dos métodos para establecer valores de servo en WPILib:

- Valor escalado - establece la posición del servo usando un valor escalado de 0 a 1.0. 0 corresponde a uno extremo del servo y 1.0 corresponde al otro.
- Angle - Set the servo position by specifying the angle, in degrees from 0 to 180. This method will work for servos with the same range as the Hitec HS-322HD servo . Any values passed to this method outside the specified range will be coerced to the boundary.

15.2 Neúmatica APIs

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.

Nota: 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++):

Nota: 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 Using the FRC Control System to Control Pneumatics

There are two options for operating solenoids to control pneumatic cylinders, the CTRE Pneumatics Control Module and the REV Robotics Pneumatics Hub.

The CTRE Pneumatics Control Module (PCM) is a CAN-based device that provides control over the compressor and up to 8 solenoids per module.

The REV Pneumatic Hub (PH) is a CAN-based device that provides control over the compressor and up to 16 solenoids per module.

These devices are integrated into WPILib through a series of classes that make them simple to use. The closed loop control of the Compressor and Pressure switch is handled by the PCM

hardware and the Solenoids are handled by the `Solenoid` class that controls the solenoid channels.

These modules are responsible for regulating the robot's pressure using a pressure switch and a compressor and switching solenoids on and off. They communicate with the roboRIO over CAN. For more information, see *Descripción general de los componentes de Hardware*.

15.2.4 Module Numbers

CAN Devices are identified by their CAN ID. The default CAN ID for PCMs is 0. The default CAN ID for PHs is 1. If using a single module on the bus it is recommended to leave it at the default CAN ID. Additional modules can be used where the modules corresponding solenoids are differentiated by the module number in the constructors of the `Solenoid`, `DoubleSolenoid` and `Compressor` classes.

15.3 Sensors

Sensors are an integral way of having your robot hardware and software communicate with each other. This section highlights interfacing with those sensors at a software level.

15.3.1 Descripción general del sensor- software

Nota: Esta sección cubre el uso de sensores en el software. Para obtener una guía sobre el hardware del sensor, vea *Descripción general del sensor - Hardware*.

Nota: Si bien las cámaras definitivamente pueden considerarse «sensores», el procesamiento de la visión es un tema suficientemente complicado que está cubierto *en su propia sección*, en vez de aquí.

Para ser efectivo, a menudo es vital que los robots puedan recopilar información sobre sus alrededores. Los dispositivos que proporcionan información al robot sobre el estado de su entorno son llamados «sensores». WPILib admite de forma innata una gran variedad de sensores a través de clases incluidas en la biblioteca. Esta sección proporcionará una guía para usar ambos tipos de sensores comunes a través de WPILib, así como escribir código para sensores sin soporte oficial.

¿Qué sensores admite WPILIB?

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 proporciona soporte nativo para:

- *Accelerometers*
- *Gyroscopes*
- *Ultrasonic rangefinders*

- *Potentiometers*
- *Counters*
- *Quadrature encoders*
- *Limit switches*

Además, WPILib incluye clases de nivel inferior para interactuar directamente con las entradas y salidas digitales y analógicas de FPGA.

15.3.2 Acelerómetros-Software

Nota: Esta sección cubre los acelerómetros en el software. Para obtener una guía de hardware para acelerómetros, consulte *Acelerómetros - Hardware*.

Un acelerómetro es un dispositivo que mide la aceleración.

Los acelerómetros generalmente vienen en dos tipos: eje único y eje 3. Un acelerómetro de un solo eje mide la aceleración a lo largo de una dimensión espacial; un acelerómetro de 3 ejes mide aceleración a lo largo de las tres dimensiones espaciales a la vez.

WPILib admite acelerómetros de eje único a través de la clase *AnalogAccelerometer*.

Los acelerómetros de tres ejes a menudo requieren protocolos de comunicación más complicados (como SPI o I2C) para enviar datos multidimensionales. WPILib tiene soporte nativo para los siguientes acelerómetros de 3 ejes:

- *ADXL345_I2C*
- *ADXL345_SPI*
- *ADXL362*
- *BuiltInAccelerometer*

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 los usuarios cuentan cuentan con un acelerómetro de 3 ejes, se pueden usar tres instancias de esta clase, una para cada eje.

There are getters for the acceleration along each cardinal direction (x, y, and z), as well as a setter for the range of accelerations the accelerometer will measure.

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);
```

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};
```

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};
```

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};
```

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;
```

Acelerómetros de terceros

Así como WPILib provee soporte nativo para un numero de acelerómetros disponibles en el kit of parts o por medio de FIRST Choice, hay también dispositivos menos populares AHRS (Attitude and Heading Reference System) los cuales son comúnmente usados en FRC e incluyen acelerómetros. Estos son controlados generalmente mediante un vendor de libraries, aunque si tienen una entrada análoga sencilla, se pueden usar con la clase *AnalogAccelerometer*.

Usar acelerómetros en código

Nota: Los acelerómetros, como su nombre lo dice, miden la aceleración. Acelerómetros precisos pueden ser útiles para determinar posiciones a través de la doble integración (desde que la aceleración es la segunda derivada de la posición), de manera similar que los giroscopios son usados para determinar la dirección. Sin embargo, los acelerómetros disponibles para usar en FRC no son de la calidad suficiente para usarse de este modo.

Se recomienda utilizar acelerómetros en FRC® para cualquier aplicación que necesite una medición aproximada de la aceleración actual. Esto puede incluir la detección de colisiones con otros robots o elementos de campo, de modo que los mecanismos vulnerables se puedan retraer automáticamente. También se pueden usar para determinar cuándo el robot pasa por un terreno accidentado para una rutina autónoma (como atravesar las defensas en FIRST Stronghold).

Para detectar colisiones, a menudo es más robusto medir la sobreaceleración que la aceleración. La sobreaceleración es la derivada (o la tasa de cambio) de la aceleración e indica la rapidez con la que cambian las fuerzas sobre el robot: el impulso repentino de una colisión provoca un pico agudo en la sacudida. La sobreaceleración se puede determinar simplemente tomando la diferencia de las mediciones de aceleración posteriores y dividiéndolas por el tiempo entre ellas:

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 Giroscopios - Software

Nota: Esta sección cubre la parte de software de los giroscopios. Para una guía mecánica de estos, véase docs/hardware/sensors/gyros-hardware:Giroscopios - Hardware.

A gyroscope, or «gyro,» is an angular rate sensor typically used in robotics to measure and/or stabilize robot headings. WPILib natively provides specific support for the ADXRS450 gyro available in the kit of parts, as well as more general support for a wider variety of analog gyros through the *AnalogGyro* class.

There are getters the current angular rate and heading and functions for zeroing the current heading and calibrating the gyro.

Nota: Es crucial que el robot permanezca inmóvil al calibrar el giroscopio.

ADIS16448

The ADIS16448 uses the `ADIS16448_IMU` class (Java, C++, Python). See the Analog Devices ADIS16448 documentation for additional information and examples.

Advertencia: The Analog Devices documentation linked above contains outdated instructions for software installation as the ADIS16448 is now built into 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.

Advertencia: The Analog Devices documentation linked above contains outdated instructions for software installation as the ADIS16470 is now built into 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.

Nota: El uso de varios ADXRS450 Gyro se maneja a través de circuitos especiales en la FPGA, así que solo se debería usar una instancia `ADXRS450_Gyro` .

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.

Nota: El uso de varios ADXRS450 Gyro se maneja a través de circuitos especiales en la FPGA, de acuerdo a esto solo se debería usar `AnalogGyro`s en los puertos analógicos 0 y 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

The navX uses the AHRS class. See the navX documentation for additional connection types.

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

The Pigeon should use the `WPI_PigeonIMU` class. The Pigeon can either be connected with CAN or by data cable to a TalonSRX. The Pigeon IMU User's Guide contains full details on using the 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
```

Usando giroscopios en código

Nota: Como los gyros miden la velocidad en vez de la posición, la posición se infiere integrando (sumando) la señal de la velocidad para obtener el cambio total en el ángulo. Así, todas las mediciones del giroscopio son relativas a un punto cero arbitrario (determinado por el ángulo del giroscopio ya sea cuando se encendió el robot o restableciendo este punto), sufren una acumulación de errores (llamada desvío) que incrementa a medida que el gyro se usa. La cantidad de desvió varía de acuerdo al tipo de gyro.

Los gyros son extremadamente útiles en el FRC tanto para medir como para controlar el rumbo de los robots. Dado que las coincidencias de FRC son generalmente cortas, la deriva total del giróscopo en el curso de un partido de FRC tiende a ser manejablemente pequeña (del orden de un par de grados para un giróscopo de buena calidad). Además, no todas las aplicaciones útiles de los giroscopios requieren que la medición absoluta del rumbo sea precisa durante todo el partido.

Mostrando el rumbo del robot en la dashboard

Shuffleboard includes a widget for displaying heading data from a gyro in the form of a compass. This can be helpful for viewing the robot heading when sight lines to the robot are obscured:

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)
```

Estabilizando el rumbo mientras se maneja

Un uso muy común de un gyro es estabilizar el rumbo del robot mientras conduce, para que el robot conduzca recto. Esto es especialmente importante para los impulsos holonómicos como el mecanismo y el desvío, pero también es muy útil para los impulsos de los tanques.

Esto se logra típicamente cerrando un controlador PID en la velocidad de giro o en el rumbo, y canalizando la salida del lazo hacia el control de giro de uno (en el caso del accionamiento de un tanque, esto sería una diferencia de velocidad entre los dos lados de la unidad).

Advertencia: Como con todos los bucles de control, los usuarios deben tener cuidado de asegurarse de que la dirección del sensor y la dirección de giro sean consistentes. Si no lo son, el bucle será inestable y el robot girará salvajemente.

Ejemplo: Estabilización de la conducción del tanque usando la velocidad de giro

El siguiente ejemplo muestra cómo estabilizar el rumbo utilizando un simple bucle P cerrado en la velocidad de giro. Dado que un robot que no está girando debería tener una tasa de giro de cero, el punto de ajuste del bucle es implícitamente cero, lo que hace que este método sea muy 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
    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

```
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)
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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)
```

Nota: MotorControllerGroup is *deprecated* in 2024. Can you help update the Python example?

Las implementaciones más avanzadas pueden utilizar un bucle de control más complicado. Al cerrar el bucle en la tasa de giro para la estabilización del rumbo, los bucles PI son particularmente efectivos.

Ejemplo: Estabilización de la conducción del tanque usando el rumbo

El siguiente ejemplo muestra cómo estabilizar el rumbo utilizando un simple bucle P cerrado en el rumbo. A diferencia del ejemplo de la velocidad de giro, necesitaremos establecer el punto de ajuste del rumbo actual antes de iniciar el movimiento, lo que hace que este método sea un poco más complicado.

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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)

    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)
```

Las implementaciones más avanzadas pueden utilizar un bucle de control más complicado. Cuando se cierra el bucle en el rumbo para la estabilización del rumbo, los bucles PD son particularmente efectivos.

Pasando a un rumbo fijo

Otra aplicación común y muy útil para un gyro es hacer girar a un robot para que se dirija a una dirección específica. Esto puede ser un componente de una rutina de conducción autónoma, o puede utilizarse durante el control teleoperado para ayudar a alinear un robot con los elementos de campo.

Al igual que con la estabilización del rumbo, esto se logra a menudo con un bucle PID - a diferencia de la estabilización, sin embargo, el bucle sólo puede cerrarse en el rumbo. El siguiente código de ejemplo hará que el robot se enfrente a 90 grados con un simple bucle 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);

@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++

```
// 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};
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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)
```

Al igual que antes, las implementaciones más avanzadas pueden utilizar bucles de control

más complicados.

Nota: Los bucles de giro-a-ángulo pueden ser difíciles de afinar correctamente debido a la fricción estática en la transmisión, especialmente si se utiliza un simple bucle P. Hay varias maneras de explicar esto; una de las más comunes/efectivas es añadir una «salida mínima» a la salida del bucle de control. Otra estrategia efectiva es la de conectar en cascada a controladores de velocidad bien ajustados a cada lado de la unidad.

15.3.4 Ultrasónicos - Software

Nota: Esta sección cubre los ultrasónicos en software. Para obtener una guía de hardware para ultrasónicos, consulte *Ultrasónicos - Hardware*.

Un sensor ultrasónico se usa comúnmente para medir la distancia a un objeto usando sonido de alta frecuencia. Generalmente, los ultrasónicos miden la distancia al objeto más cercano dentro de su «campo de visión».

Hay dos tipos principales de ultrasónicos compatibles de forma nativa con WPILib:

- *Ping-response ultrasonics*
- *Analog ultrasonics*

Ultrasónicos de respuesta de 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;
```

Ultrasónicos analógicos

Algunos sensores ultrasónicos simplemente devuelven un voltaje analógico correspondiente a la distancia medida. Estos sensores se pueden utilizar simplemente con la clase *AnalogPotentiometer* .

Ultrasónicos de terceros

Otros sensores ultrasónicos ofrecidos por terceros pueden utilizar protocolos de comunicación más complicados (como I2C o SPI). WPILib no proporciona soporte nativo para tales ultrasónicos; normalmente se controlarán con bibliotecas de proveedores.

Usando ultrasónicos en código

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;
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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);

  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;
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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};
  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);
}
```

Additionally, ping-response ultrasonics can be sent to *Shuffleboard*, where they will be displayed with their own widgets:

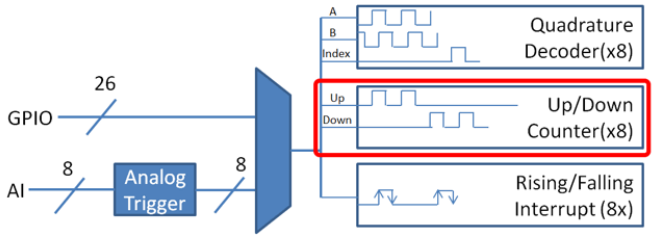
Java

```
// 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 Counters



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.

Nota: Hay un total de 8 unidades de contador en la FPGA roboRIO, lo que significa que no más de 8 objetos `Counter` pueden instanciar en cualquier momento, incluidos aquellos contenidos como recursos en otros objetos WPILib. Para obtener información detallada sobre cuándo un `Counter` puede ser utilizado por otro objeto, consulte la documentación oficial de la API.

Configurar un counter

La clase `Counter` se puede configurar de varias formas para proporcionar diferentes funcionalidades.

Modos del counter

El objeto `Counter` puede configurarse para operar en uno de cuatro modos diferentes:

1. *Two-pulse mode*: cuenta hacia arriba y hacia abajo según los bordes de dos canales diferentes.
2. *Semi-period mode*:Mide la duración de un pulso en un solo canal.
3. *Pulse-length mode*: cuenta hacia arriba y hacia abajo según los bordes de un canal, con la dirección determinada por la duración del pulso en ese canal.

4. *External direction mode*: cuenta hacia arriba y hacia abajo según los bordes de un canal, con un canal separado que especifica la dirección.

Nota: 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.

Modo de dos pulsos

En el modo de dos pulsos, el `Counter` contará hacia arriba por cada flanco / pulso en el «canal ascendente» especificado, y hacia abajo por cada flanco / pulso en el «canal descendente» especificado. Un contador se puede inicializar en dos pulsos con el siguiente código:

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);
```

Modo de semiperíodo

En el modo de semiperíodo, el `Counter` contará la duración de los pulsos en un canal, ya sea desde un borde ascendente hasta el siguiente borde descendente, o desde un borde descendente hasta el siguiente borde ascendente. Un contador se puede inicializar en modo semiperíodo con el siguiente código:

JAVA

```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++

```cpp
// 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);
```

Para obtener el ancho de pulso, llame al método `getPeriod()`.

JAVA

```java
// Return the measured pulse width in seconds
counter.getPeriod();
```

C++

```cpp
// Return the measured pulse width in seconds
counter.GetPeriod();
```

Modo de duración de pulso

En el modo de longitud de pulso, el contador contará hacia arriba o hacia abajo según la longitud del pulso. Un pulso por debajo del umbral de tiempo especificado se interpretará como un conteo hacia adelante y un pulso por encima del umbral es un conteo hacia atrás. Esto es útil para algunos sensores de dientes de engranajes que codifican la dirección de esta manera. Un contador se puede inicializar en este modo de la siguiente manera:

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)
```

Modo de dirección externa

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

```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
    counter.setUpSourceEdge(true, true);
}
```

C++

```cpp
// 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);
```

Configurar los parámetros del contador

Nota: La clase `Counter` no hace ninguna suposición sobre las unidades de distancia; devolverá valores en las unidades que se hayan utilizado para calcular el valor de distancia por pulso. De este modo, los usuarios tienen un control completo sobre las unidades de distancia utilizadas. Sin embargo, las unidades de tiempo son *siempre* en segundos.

Nota: El número de pulsos utilizados en el cálculo de la distancia por pulso *no* depende del tipo de decodificación; cada «pulso» siempre debe considerarse como un ciclo completo (ascendente y descendente).

Aparte de las configuraciones específicas del modo, la clase `Counter` ofrece varios métodos de configuración adicionales:

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
// 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);
```

Leer información de contadores

Independientemente del modo, hay cierta información que la clase `Counter` siempre expone a los usuarios:

Count

Los usuarios pueden obtener el recuento actual con el método `get()`.

JAVA

```
// returns the current count
counter.get();
```

C++

```
// returns the current count
counter.Get();
```

Distancia

Nota: Counters measure *relative* distance, not absolute; the distance value returned will depend on the position of the encoder when the robot was turned on or the encoder value was last *reset*.

If the *distance per pulse* has been configured, users can obtain the total distance traveled by the counted sensor with the `getDistance()` method:

JAVA

```
// returns the current distance
counter.getDistance();
```

C++

```
// returns the current distance
counter.GetDistance();
```

Velocidad

Nota: Las unidades de tiempo para la clase `Counter` son *siempre* en segundos.

Los usuarios pueden obtener la tasa de cambio actual del contador con el método `getRate()`:

JAVA

```
// Gets the current rate of the counter
counter.getRate();
```

C++

```
// Gets the current rate of the counter
counter.GetRate();
```

Detenido

Los usuarios pueden obtener si el contador está parado con el método: code:*getStopped()*:

JAVA

```
// Gets whether the counter is stopped
counter.getStopped();
```

C++

```
// Gets whether the counter is stopped
counter.GetStopped();
```

Dirección

Los usuarios pueden obtener la dirección en la que el contador se movió por última vez con el método `getDirection()` :

JAVA

```
// Gets the last direction in which the counter moved
counter.getDirection();
```

C++

```
// Gets the last direction in which the counter moved
counter.GetDirection();
```

Período

Nota: In *semi-period mode*, this method returns the duration of the pulse, not of the period.

Los usuarios pueden obtener la duración (en segundos) del período más reciente con el método `getPeriod()` :

JAVA

```
// returns the current period in seconds
counter.getPeriod();
```

C++

```
// returns the current period in seconds
counter.GetPeriod();
```

Reiniciar un counter

Para restablecer un contador a una lectura de distancia de cero, llame al método `reset()`. Esto es útil para asegurar que la distancia medida corresponde a la medida física deseada real.

JAVA

```
// Resets the encoder to read a distance of zero
counter.reset();
```

C++

```
// Resets the encoder to read a distance of zero
counter.Reset();
```

Usando conters en código

Counters are useful for a wide variety of robot applications - but since the `Counter` class is so varied, it is difficult to provide a good summary of them here. Many of these applications overlap with the `Encoder` class - a simple counter is often a cheaper alternative to a quadrature encoder. For a summary of potential uses for encoders in code, see *Codificadores - Software*.

15.3.6 Codificadores - Software

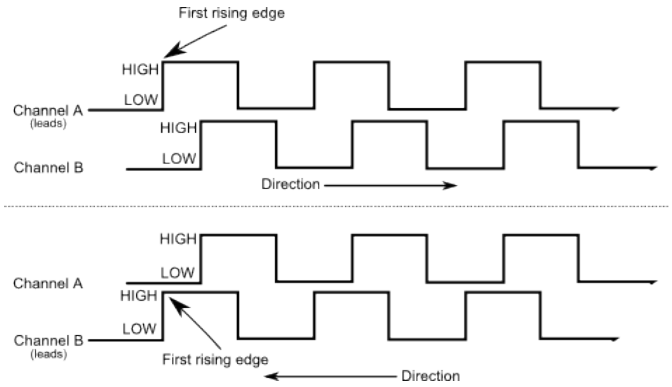
Nota: Esta sección cubre codificadores en software. Para obtener una guía de hardware para codificadores, consulte *Codificadores - Hardware*.

Encoders are devices used to measure motion (usually, the rotation of a shaft).

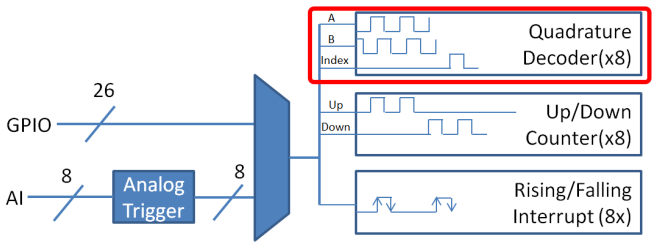
Importante: The classes in this document are only used for encoders that are plugged directly into the roboRIO! Please reference the appropriate vendors" documentation for using encoders plugged into motor controllers.

Quadrature Encoders - The `Encoder` Class

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.



These encoders produce square-wave signals on two channels that are a quarter-period out-of-phase (hence the term, «quadrature»). The pulses are used to measure the rotation, and the direction of motion can be determined from which channel «leads» the other.



The FPGA handles quadrature encoders either through a counter module or an encoder module, depending on the *decoding type* - the choice is handled automatically by WPILib. The FPGA contains 8 encoder modules.

Examples of quadrature encoders:

- AMT103-V available through FIRST Choice
- CIMcoder
- CTRE Mag Encoder
- Grayhill 63r
- REV Through Bore Encoder
- US Digital E4T

Initializing a Quadrature Encoder

A quadrature encoder can be instantiated as follows:

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};
```

Decoding Type

La clase WPILib `Encoder` puede decodificar señales de codificador en tres modos diferentes:

- **Decodificación 1X**: aumenta la distancia por cada período completo de la señal del codificador (una vez cada cuatro bordes).
- **Decodificación 2X**: Incrementa la distancia por cada medio período de la señal del codificador (una vez por dos bordes).

- **Decodificación 4X**: aumenta la distancia para cada borde de la señal del codificador (cuatro veces por período).

La decodificación 4X ofrece la mayor precisión, pero a costa de un aumento de la «fluctuación» en las mediciones de velocidad. Para usar un tipo de decodificación diferente, use el siguiente constructor:

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};
```

Configuring Quadrature Encoder Parameters

Nota: La clase `Encoder` no hace ninguna suposición sobre las unidades de distancia; devolverá valores en las unidades que se hayan utilizado para calcular el valor de distancia por pulso. De este modo, los usuarios tienen un control completo sobre las unidades de distancia utilizadas. Sin embargo, las unidades de tiempo son *siempre* en segundos.

Nota: The number of pulses used in the distance-per-pulse calculation does *not* depend on the *decoding type* - each «pulse» should always be considered to be a full cycle (four edges).

La clase `Encoder` ofrece varios métodos de configuración:

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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
// 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);
```

Reading information from Quadrature Encoders

La clase `Encoder` proporciona una gran cantidad de información al usuario sobre el movimiento del codificador.

Distancia

Nota: Quadrature encoders measure *relative* distance, not absolute; the distance value returned will depend on the position of the encoder when the robot was turned on or the encoder value was last *reset*.

Los usuarios pueden obtener la distancia total recorrida por el codificador con el método `getDistance()`:

JAVA

```
// Gets the distance traveled
encoder.getDistance();
```

C++

```
// Gets the distance traveled
encoder.GetDistance();
```

Velocidad

Nota: Las unidades de tiempo para la clase `Encoder` son *siempre* en segundos.

Los usuarios pueden obtener la tasa actual de cambio del codificador con el método `getRate()`:

JAVA

```
// Gets the current rate of the encoder
encoder.getRate();
```

C++

```
// Gets the current rate of the encoder
encoder.GetRate();
```

Detenido

Los usuarios pueden obtener si el codificador está estacionario con el método `getStopped()`:

JAVA

```
// Gets whether the encoder is stopped
encoder.getStopped();
```

C++

```
// Gets whether the encoder is stopped
encoder.GetStopped();
```

Dirección

Los usuarios pueden obtener la dirección en la que el codificador se movió por última vez con el método `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();
```

Período

Los usuarios pueden obtener el período de los pulsos del codificador (en segundos) con el método `getPeriod()`:

JAVA

```
// Gets the current period of the encoder
encoder.getPeriod();
```

C++

```
// Gets the current period of the encoder
encoder.GetPeriod();
```

Resetting a Quadrature Encoder

To reset a quadrature encoder to a distance reading of zero, call the `reset()` method. This is useful for ensuring that the measured distance corresponds to the actual desired physical measurement, and is often called during a *homing* routine:

JAVA

```
// Resets the encoder to read a distance of zero
encoder.reset();
```

C++

```
// Resets the encoder to read a distance of zero
encoder.Reset();
```

Duty Cycle Encoders - The `DutyCycleEncoder` class

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.

The roboRIO's FPGA handles duty cycle encoders automatically.

Examples of duty cycle encoders:

- AndyMark Mag Encoder
- CTRE Mag Encoder
- REV Through Bore Encoder
- Team 221 Lamprey2
- US Digital MA3

Initializing a Duty Cycle Encoder

A duty cycle encoder can be instantiated as follows:

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};
```

Configuring Duty Cycle Encoder Parameters

Nota: The `DutyCycleEncoder` class does not make any assumptions about units of distance; it will return values in whatever units were used to calculate the distance-per-rotation value. Users thus have complete control over the distance units used.

The `DutyCycleEncoder` class offers a number of configuration methods:

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);
```

Reading Distance from Duty Cycle Encoders

Nota: Duty Cycle encoders measure absolute distance. It does not depend on the starting position of the encoder.

Users can obtain the distance measured by the encoder with the `getDistance()` method:

JAVA

```
// Gets the distance traveled
encoder.getDistance();
```

C++

```
// Gets the distance traveled
encoder.GetDistance();
```

Detecting a Duty Cycle Encoder is Connected

As duty cycle encoders output a continuous set of pulses, it is possible to detect that the encoder has been unplugged.

JAVA

```
// Gets if the encoder is connected
encoder.isConnected();
```

C++

```
// Gets if the encoder is connected
encoder.IsConnected();
```

Resetting a Duty Cycle Encoder

To reset an encoder so the current distance is 0, call the `reset()` method. This is useful for ensuring that the measured distance corresponds to the actual desired physical measurement. Unlike quadrature encoders, duty cycle encoders don't need to be homed. However, after reset, the position offset can be stored to be set when the program starts so that the reset doesn't have to be performed again. The *Preferences class* provides a method to save and retrieve the values on the 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);
```

Analog Encoders - The `AnalogEncoder` Class

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.

Examples of analog encoders:

- Team 221 Lamprey2
- Thrifty Absolute Magnetic Encoder
- US Digital MA3

Initializing an Analog Encoder

An analog encoder can be instantiated as follows:

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};
```

Configuring Analog Encoder Parameters

Nota: The `AnalogEncoder` class does not make any assumptions about units of distance; it will return values in whatever units were used to calculate the distance-per-rotation value. Users thus have complete control over the distance units used.

The `AnalogEncoder` class offers a number of configuration methods:

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);
```

Reading Distance from Analog Encoders

Nota: Analog encoders measure absolute distance. It does not depend on the starting position of the encoder.

Users can obtain the distance measured by the encoder with the `getDistance()` method:

JAVA

```
// Gets the distance measured
encoder.getDistance();
```

C++

```
// Gets the distance measured
encoder.GetDistance();
```

Resetting an Analog Encoder

To reset an analog encoder so the current distance is 0, call the `reset()` method. This is useful for ensuring that the measured distance corresponds to the actual desired physical measurement. Unlike quadrature encoders, duty cycle encoders don't need to be homed. However, after reset, the position offset can be stored to be set when the program starts so that the reset doesn't have to be performed again. The *Preferences class* provides a method to save and retrieve the values on the 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);
```

Using Encoders in Code

Encoders are some of the most useful sensors in FRC®; they are very nearly a requirement to make a robot capable of nontrivially-automated actuations and movement. The potential applications of encoders in robot code are too numerous to summarize fully here, but an example is provided below:

Driving to a Distance

Encoders can be used on a robot drive to create a simple «drive to distance» routine. This is useful in autonomous mode, but has the disadvantage that the robot's momentum will cause it to overshoot the intended distance. Better methods include using a *PID Controller* or using *Path Planning*

Nota: The following example uses the *Encoder* class, but is similar if other *DutyCycleEncoder* or *AnalogEncoder* is used. However, quadrature encoders are typically better suited for drivetrains since they roll over many times and don't have an absolute position.

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.);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    // 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); }};

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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    } else {
        drive.TankDrive(0, 0);
    }
}
```

Homing a Mechanism

Since quadrature encoders measure *relative* distance, it is often important to ensure that their «zero-point» is in the right place. A typical way to do this is a «homing routine,» in which a mechanism is moved until it hits a known position (usually accomplished with a limit switch), or «home,» and then the encoder is reset. The following code provides a basic example:

Nota: Homing is not necessary for absolute encoders like duty cycle encoders and analog encoders.

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 {
        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()) {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        spark.Set(-0.5);
    } else {
        spark.Set(0);
        encoder.Reset();
    }
}
```

15.3.7 Entradas análogas - Software

Nota: Esta sección cubre las entradas análogas en software. Para una guía de entradas análogas en Hardware, lea *Entradas analógicas - Hardware*.

El FPGA de roboRIO admite hasta 8 canales de entrada analógica que se pueden usar para leer el valor de un voltaje analógico de un sensor. Las entradas analógicas se pueden usar para cualquier sensor que emite un voltaje simple.

Las entradas analógicas del FPGA por defecto devuelven un número entero de 12 bits proporcional al voltaje, de 0 a 5 voltios.

Clase AnalogInput

Nota: Normalmente es más conveniente usar la clase *Analog Potentiometers<analog-potentiometers-software>* que usar la *AnalogInput* directamente, ya que permite escalar a unidades más significativas.

Support for reading the voltages on the FPGA analog inputs is provided through the `AnalogInput` class (Java, C++).

Inicializando una AnalogInput

Una AnalogInput tiene que inicializarse de esta forma:

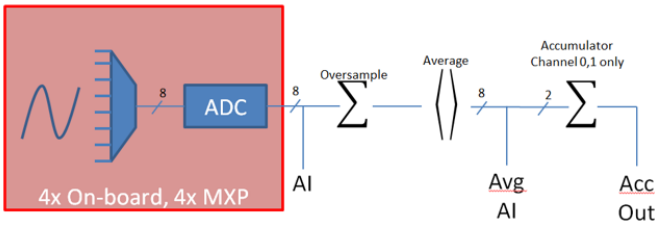
JAVA

```
// Initializes an AnalogInput on port 0
AnalogInput analog = new AnalogInput(0);
```

C++

```
// Initializes an AnalogInput on port 0
frc::AnalogInput analog{0};
```

Sobremuestreo y Promedio



Los módulos de entrada analógica de FPGA admiten sobremuestreo y promedio. Estos comportamientos son muy similares, pero difieren en algunas formas importantes. Ambos pueden usarse al mismo tiempo.

Sobremuestreo

When oversampling is enabled, the FPGA will add multiple consecutive samples together, and return the accumulated value. Users may specify the number of *bits* of oversampling - for n bits of oversampling, the number of samples added together is 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);
```

Promedio

El promedio se comporta como un sobremuestreo, excepto que los valores acumulados se dividen por el número de muestras para que la escala de los valores devueltos no cambie. Esto suele ser más-conveniente, pero ocasionalmente el error de redondeo adicional introducido por el redondeo es indeseable.

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);
```

Nota: Cuando se utilizan sobremuestreo y promedio al mismo tiempo, primero se aplica el sobremuestreo y luego se promedian los valores sobremuestreados. Por lo tanto, el sobremuestreo de 2 bits y el promedio de 2 bits utilizados al mismo tiempo aumentará la escala de los valores devueltos en aproximadamente un factor de 2 y disminuirá la velocidad de actualización en aproximadamente un factor de 4.

Leyendo valores de la clase AnalogInput

Los valores se pueden leer desde AnalogInput con uno de cuatro métodos diferentes:

getValue

El método getValue devuelve el valor bruto instantáneo medido de la entrada analógica, sin aplicar ninguna calibración e ignorar las configuraciones de sobremuestreo y promedio. El valor devuelto es un entero.

JAVA

```
analog.getValue();
```

C++

```
analog.GetValue();
```

getVoltage

El método getVoltage devuelve el voltaje medido instantáneo desde la entrada analógica. Las configuraciones de sobremuestreo y promedio se ignoran, pero el valor se vuelve a escalar para representar un voltaje. El valor devuelto es un doble.

JAVA

```
analog.getVoltage();
```

C++

```
analog.GetVoltage();
```

getAverageValue

El método getAverageValue devuelve el valor promedio de la entrada analógica. El valor no se vuelve a escalar, pero se aplican sobremuestreo y promedio. El valor devuelto es un entero.

JAVA

```
analog.getAverageValue();
```

C++

```
analog.GetAverageValue();
```

getAverageVoltage

El método getAverageVoltage devuelve el voltaje promedio de la entrada analógica. El reescalado, el sobremuestreo y el promedio se aplican. El valor devuelto es un doble.

JAVA

```
analog.getAverageVoltage();
```

C++

```
analog.GetAverageVoltage();
```

Acumulador

Nota: Los métodos de acumulación no admiten actualmente la devolución de un valor en unidades de voltios; el valor devuelto siempre será un entero (específicamente, uno largo).

Los canales de entrada analógica 0 y 1 soportan adicionalmente un acumulador, que integra (suma) la señal indefinidamente, de modo que el valor devuelto es la suma de todos los valores medidos pasados. El sobremuestreo y el promedio se aplican antes de la acumulación.

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);

// 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);

```

(continúe en la próxima página)

(proviene de la página anterior)

```
// 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();
```

Obtención de recuento y valor sincronizados

A veces, es necesario obtener medidas coincidentes de la cuenta y el valor. Esto se puede hacer usando el método 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);
```

Usar entradas analógicas en código

La clase AnalogInput se puede usar para escribir código para una amplia variedad de sensores (incluidos potenciómetros, acelerómetros, giroscopios, ultrasonidos y más) que devuelven sus datos como un voltaje analógico. Sin embargo, si es posible, casi siempre es más conveniente utilizar una de las otras clases WPILib existentes que manejan el código de nivel inferior (leer los voltajes analógicos y convertirlos en unidades significativas) para usted. Los usuarios solo deben usar AnalogInput directamente como «último recurso».

En consecuencia, para obtener ejemplos de cómo usar efectivamente sensores analógicos en el código, los usuarios deben consultar las otras páginas de este capítulo que tratan sobre clases más específicas.

15.3.8 Potenciómetros analógicos - Software

Nota: Esta sección cubre el software del analizador de potenciómetros. Para una guía de hardware a potenciómetros analógicos, ver *Potenciómetros análogos - Hardware*.

Los potenciómetros son resistencias variables que permiten que la información sobre la posición se convierta en una señal de voltaje analógica. La roboRIO puede leer esta señal para controlar cualquier dispositivo conectado al potenciómetro.

While it is possible to read information from a potentiometer directly with an *Entradas análogas - Software*, 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.

De hecho, el nombre *AnalogPotentiometer* es un nombre poco apropiado: esta clase se debe utilizar para la gran mayoría de los sensores que devuelven su señal como un voltaje analógico simple y de escala lineal.

La clase AnalogPotentiometer

Nota: Los parámetros de «rango completo» o «escala» en el constructor AnalogPotentiometer son factores de escala de un rango de 0-1 al rango real, no de 0-5. Es decir, representan una escala fraccional nativa, en lugar de una escala de voltaje.

Para inicializar la AnalogPotentiometer puede utilizar lo siguiente:

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};
```

Personalizando la entrada analógica subyacente

Nota: If the user changes the scaling of the `AnalogInput` with oversampling, this must be reflected in the scale setting passed to the `AnalogPotentiometer`.

Si el usuario desea aplicar configuraciones personalizadas a la entrada *AnalogInput* utilizada por el *AnalogPotentiometer*, se puede usar un constructor alternativo en el que se introduzca la entrada analógica:

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};
```

Leyendo valores de AnalogPotentiometer

El valor escalado se puede leer simplemente llamando al método `get` :

JAVA

```
pot.get();
```

C++

```
pot.Get();
```

Cómo usar potenciómetros analógicos en el código

Los sensores analógicos se pueden usar en código de la misma manera que otros sensores que miden lo mismo. Si el sensor analógico es un potenciómetro que mide el ángulo de un brazo, puede usarse de manera similar a un codificador.<encoders-software> Si es un sensor ultrasónico, puede usarse de manera similar a otros ultrasonidos.<ultrasonics-software>

Es muy importante tener en cuenta que los potenciómetros físicos reales generalmente tienen un rango de movimiento limitado. Las salvaguardas deben estar presentes tanto en el mecanismo físico como en el código para garantizar que el mecanismo no rompa el sensor al pasar su alcance máximo.

15.3.9 Entradas digitales - Software

Nota: Esta sección cubre entradas digitales en software. Para obtener una guía de hardware para entradas digitales, consulte *Entradas digitales - Hardware*.

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 *Entradas digitales - Hardware*). 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 clase DigitalInput

Una entrada digital `DigitalInput` se puede inicializar de la siguiente manera:

JAVA

```
// Initializes a DigitalInput on DIO 0
DigitalInput input = new DigitalInput(0);
```

C++

```
// Initializes a DigitalInput on DIO 0
frc::DigitalInput input{0};
```

Leyendo el valor de DigitalInput

El estado de `DigitalInput` se puede consultar con el método `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();
```

Crear una DigitalInput a partir de un AnalogInput.

Nota: Un `AnalogTrigger` construido con un argumento de número de puerto puede compartir ese puerto analógico con un `AnalogInput` separado, pero dos objetos` AnalogInput `pueden no compartir el mismo puerto.

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` disparador análogo puede iniciarse como sigue. Al igual que con el `AnalogPotentiometer`, un `AnalogInput` se puede pasar una Entrada Analógica explícitamente si el usuario desea personalizar la configuración de muestreo:

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};
```

Estableciendo los puntos del gatillo

Nota: Para obtener detalles sobre la escala de los valores de `AnalogInput` «sin procesar», consulte *Entradas análogas - Software*.

Para convertir la señal analógica a digital, es necesario especificar a qué valores habilitará y deshabilitará el disparador. Estos valores pueden ser diferentes para evitar «dithering» alrededor del punto de transición:

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);
```

Uso de entradas digitales en el código

As almost all switches on the robot will be used through a `DigitalInput`. This class is extremely important for effective robot control.

Limitar el rango de movimiento de un mecanismo

Los mecanismos motorizados (como brazos y elevadores) utilizados en FRC® deben recibir algún tipo de «limit switch» para evitar que estos se dañen al final de su rango de movimiento. A continuación, se muestra un breve ejemplo:

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 {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        spark.Set(0);
    }
}
```

Dirigiendo un mecanismo

Limit switches are very important for being able to «home» a mechanism with an encoder. For an example of this, see *Homing a Mechanism*.

15.3.10 Programación de interruptores de límite

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.

Los interruptores de límite pueden tener salidas «normalmente abiertas» o «normalmente cerradas». Esto controlará si una señal alta significa que el interruptor está abierto o cerrado. Para obtener más información sobre el hardware del interruptor de límite, consulte este *artículo*.

Control de un motor con dos interruptores de límite

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
            motor.set(0);
        } else {
            // We are going down but bottom limit is not tripped so go at commanded␣
↪speed
            motor.set(speed);
        }
    }
}
```

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 Miscellaneous Hardware APIs

This section highlights miscellaneous hardware APIs that are standalone.

15.4.1 LED direccionables

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*.

Crear instancias del objeto LED direccionable

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.

> **Advertencia:** Es importante tener en cuenta que configurar la longitud del encabezado del LED es una tarea costosa y no se recomienda ejecutar esto periódicamente.

Después de que se haya establecido la longitud de la tira, tendrá que crear un objeto `AddressableLEDBuffer` que toma la cantidad de LED como entrada. Entonces llamarás a `myAddressableLed.setData(myAddressableLEDBuffer)` para establecer los datos de salida del led. Finalmente puedes llamar `myAddressableLed.start()` para escribir la salida continuamente. A continuación, se muestra un ejemplo completo del proceso de inicialización

---

Nota: C++ no tiene un AddressableLEDBuffer, y en su lugar usa una matriz.

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++

```
class Robot : public frc::TimedRobot {
 private:
  static constexpr int kLength = 60;

  // PWM port 9
  // Must be a PWM header, not MXP or DIO
  frc::AddressableLED m_led{9};
  std::array<frc::AddressableLED::LEDData, kLength>
      m_ledBuffer;  // Reuse the buffer
  // Store what the last hue of the first pixel is
  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();
}
```

Nota: 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.

Configuración de toda la tira a un color

El color se puede ajustar a un led individual en la tira usando dos métodos. `setRGB()` el cual toma los valores RGB como entrada y `setHSV()` el cual toma los valores HSV como entrada.

Usando valores RGB

RGB significa rojo, verde y azul. Este es un modelo de color bastante común, ya que es bastante fácil comprender. Los LED se pueden configurar con el método `setRGB` que toma 4 argumentos: índice del LED, cantidad de rojo, cantidad de verde, cantidad de azul. La cantidad de rojo, verde y azul son valores enteros entre 0-255.

Java

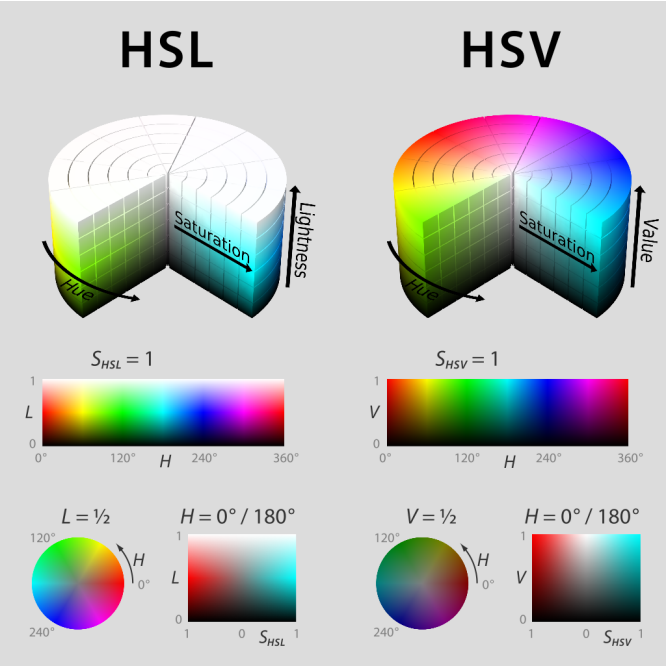
```
for (var i = 0; i < m_ledBuffer.getLength(); i++) {
   // Sets the specified LED to the RGB values for red
   m_ledBuffer.setRGB(i, 255, 0, 0);
}

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);
```

Usando valores HSV

HSV significa Hue, Saturación y Valor. El tono describe el color o tinte, siendo la saturación la cantidad de gris y el valor es el brillo. En WPILib, Hue es un número entero de 0 a 180. La saturación y el valor son enteros del 0 al 255. Si observa un selector de color como el de Google's <https://www.google.com/search?q=color+picker>`_, el tono será 0 - 360 y la saturación y el valor son del 0% al 100%. Esta es la misma manera que OpenCV maneja los colores HSV. Asegúrese de que los valores de HSV ingresados en WPILib sean correctos, o el color producido podría no ser el mismo que se esperaba.



Los LED se pueden configurar con el método setHSV que toma 4 argumentos: índice del LED, tono, saturación y valor. A continuación, se muestra un ejemplo para configurar el color de una tira de LED en rojo (matiz 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);
```

Crea un efecto arcoíris

El siguiente método hace un par de cosas importantes. Dentro del bucle *for*, distribuye equitativamente el tono en toda la longitud del hilo y almacena el tono LED individual en un variable llamada `tonalidad`. Luego, el bucle for establece el valor HSV de ese píxel especificado utilizando el tono valor.

Moviéndose fuera del bucle for, `m_rainbowFirstPixelHue` luego itera el píxel que contiene el tono «inicial» que crea el efecto arcoíris. `m_rainbowFirstPixelHue` luego verifica para asegurarse de que el tono está dentro de los límites de tono de 180. Esto se debe a que el tono HSV es un valor de 0-180.

Nota: Es una buena práctica de robots mantener el método `robotPeriodic()` lo más limpio posible, así que crearemos un método para manejar la configuración de nuestros datos LED. Llamaremos a este método `rainbow()` y llamarlo desde `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;
}
```

Ahora que hemos creado nuestro método de arco iris, tenemos que llamar al método y establecer los datos del 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 Controladores de motor

A motor controller is responsible on your robot for making motors move. For brushed DC motors such as the *CIM* or 775, the motor controller regulates the voltage that the motor receives, much like a light bulb. For brushless motor controllers such as the Spark MAX, the controller regulates the power delivered to each «phase» of the motor.

Nota: Otro nombre para un controlador de motor es un controlador de velocidad.

Consejo: Se puede hacer un controlador de motor rápido, no legal para la competición, quitando el motor de un taladro inalámbrico BRUSHED y conectando PowerPoles o equivalentes a los cables del motor. Asegúrese de que el voltaje suministrado por el taladro no dañe el motor, pero tenga en cuenta que el 775 está bien hasta 24 voltios.

Advertencia: Advertencia: Conectar el controlador de un motor SIN ESCOBILLAS directo a la corriente, como un controlador de motor de escobilla, ¡destruirá el motor!

15.5.1 Controladores de motores legales 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 Controlador de Motor (P/N: 410-334-1, 410-334-2)
- Jaguar Motor Controller (P/N: MDL-BDC, MDL-BDC24, and 217-3367) connected to *PWM* only
- Motor Nidec Dynamo BLDC con controlador solo a control de solenoide integral (P/N 840205-000, am-3740)
- SD540 Controlador de Motor(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,
- Controlador de motor Talon (P/N: CTRE_Talon, CTRE_Talon_SR, and am-2195)
- Controlador de motor Talon SRX (P/N: 217-8080, am-2854, 14-838288)
- Motor Venom con controlador (P/N BDC-10001) solo para controlado integral de motor.
- Controlador de motor Victor 884 (P/N: VICTOR-884-12/12)
- Controlador de motor Victor 888 (P/N: 217-2769)
- Controlador de motor Victor SP (P/N: 217-9090, am-2855, 14-868380)
- Controlador de motor Victor SPX (P/N: 217-9191, 17-868388, am-3748)

15.6 Neumática

Pneumatics are a quick and easy way to make something that's in one state or another using compressed air. For information on operating pneumatics, see *Neúmatica APIs*.

15.6.1 Controladores de neumáticos legales FRC

- Módulo de control de neumáticos (P/N: am-2858, 217-4243)
- Pneumatic Hub (P/N REV-11-1852)

15.7 Relés

Un relé controla la energía a un motor o electrónica personalizada en forma de encendido / apagado.

15.7.1 Modulo relevador legales FRC

- Relevador Spike H-Bridge (P/N: 217-0220 and SPIKE-RELAY-H)
- Relevador Directo de Automatizacion (P/N: AD-SSR6M12-DC200D, AD-SSR6M25-DC200D, ADSSR6M40-DC200D)
- Power Distribution Hub (PDH) switched channel (P/N REV-11-1850)

16 Dispositivos CAN

16.1 Usando dispositivos CAN

CAN has many advantages over other methods of connection between the robot controller and peripheral devices.

- Las conexiones CAN están conectadas en cadena de un dispositivo a otro, lo que a menudo resulta en tendidos de cables mucho más cortos que tener que conectar cada dispositivo al propio RIO.
- 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.
- CAN es bidireccional, por lo que los controladores de motor CAN pueden enviar datos de vuelta al RIO, lo que nuevamente facilita un conjunto de características más expansivo que el que ofrecen los controladores PWM.

For instructions on wiring CAN devices, see the relevant section of the *robot wiring guide*.

Los dispositivos CAN generalmente tienen sus propias clases WPILib. Las siguientes secciones describirán el uso de varias de estas clases.

16.2 Módulo de control neumático

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 Pneumatic Hub

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 Módulo de distribución de energía

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 Creating a Power Distribution Object

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)
```

Note: it is not necessary to create a PowerDistribution object unless you need to read values from it. The board will work and supply power on all the channels even if the object is never created.

> **Advertencia:** To enable voltage and current logging in the Driver Station, the CAN ID for the CTRE Power Distribution Panel *must* be 0, and for the REV Power Distribution Hub it *must* be 1.

16.4.2 Lectura del voltaje del bus

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)
```

Monitorear el voltaje del bus puede ser útil para (entre otras cosas) detectar cuando el robot está cerca de un apagón, de modo que se pueden tomar medidas para evitar el apagón de manera controlada. Consulte el documento *roboRIO Brownouts document* para obtener más información.

16.4.3 Leyendo la temperatura

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)
```

El monitoreo de la temperatura puede ser útil para detectar si el robot ha consumido demasiada energía y necesita apagarse por un tiempo, o si hay un problema de cableado corto u otro.

16.4.4 Reading the Total Current, Power, and Energy

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)
```

Monitoring the total current, power and energy can be useful for controlling how much power is being drawn from the battery, both for preventing brownouts and ensuring that mechanisms have sufficient power available to perform the actions required. Power is the bus voltage

multiplied by the current with the units Watts. Energy is the power summed over time with units Joules.

16.4.5 Lectura de corrientes de canales individuales

The PDP/PDH also allows users to monitor the current drawn by the individual device power channels. You can read the current on any of the 16 PDP channels (0-15) or 24 PDH channels (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)
```

El monitoreo de las corrientes de corriente de dispositivos individuales puede ser útil para detectar cortocircuitos o motores estancados.

16.4.6 Using the Switchable Channel (PDH)

The REV PDH has one channel that can be switched on or off to control custom circuits.

JAVA

```
examplePD.setSwitchableChannel(true);
examplePD.setSwitchableChannel(false);
```

C++

```
examplePD.SetSwitchableChannel(true);
examplePD.SetSwitchableChannel(false);
```

PYTHON

```
examplePD.setSwitchableChannel(True)
examplePD.setSwitchableChannel(False)
```

16.5 Dispositivos CAN de terceros

A number of FRC® vendors offer their own *CAN* peripherals. As CAN devices offer expansive feature-sets, vendor CAN devices require similarly expansive code libraries to operate. As a result, these libraries are not maintained as an official part of WPILib, but are instead maintained by the vendors themselves. For a guide to installing third-party libraries, see *3rd Party Libraries*

A continuación se proporciona una lista de dispositivos CAN comunes de terceros de varios proveedores, junto con enlaces a la documentación externa correspondiente:

16.5.1 CTR Electronics

CTR Electronics (CTRE) offers several CAN peripherals with external libraries. General resources for all CTRE devices include:

> Phoenix Device Software Documentation

Controladores de motor CTRE

- **Talon FX (with Falcon 500 Motor)**
  - API Documentation (v5: Java, C++ | Pro: Java, C++)
  - Hardware User's Manual
  - Software Documentation (v5, Pro)
- **Talon SRX**
  - API Documentation (Java, C++)
  - Hardware User's Manual
  - Software Documentation

- **Victor SPX**
  - API Documentation (Java, C++)
  - Hardware User's Manual
  - Software Documentation

Sensores CTRE

- **CANcoder**
  - API Documentation (v5: Java, C++ | Pro: Java, C++)
  - Hardware User's Manual
  - Software Documentation (v5, Pro)
- **Pigeon 2.0**
  - API Documentation (v5: Java, C++ | Pro: Java, C++)
  - Hardware User's Manual
  - Software Documentation (v5, Pro)
- **Pigeon IMU**
  - API Documentation (Java, C++)
  - Hardware User's Manual
  - Software Documentation
- **CANifier**
  - API Documentation (Java, C++)
  - Hardware User's Manual
  - Software Documentation

CTRE Other CAN Devices

- **CANdle LED Controller**
  - API Documentation (Java, C++)
  - Hardware User's Manual
  - Software Documentation

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.

Controladores de motor 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) ofrece el motor / controlador integrado Venom, así como un sensor de distancia de tiempo de vuelo:

Controladores de motor PWF

- **Venom**
  - Documentación API (Java, C++)
  - *Manual técnico <https://www.playingwithfusion.com/include/getfile.php?fileid=7086>* —

Sensores PWF

- ** Sensor de tiempo de vuelo **
  - Documentación API (Java, C++)
  - Manual técnico

16.5.4 Redux Robotics

Redux Robotics currently offers the Canandcoder *CAN* + *PWM* magnetic encoder and the Canandcolor *CAN*-enabled color sensor.

Redux Sensors

- **Canandcoder**
  - API Documentation (Java, C++)
  - Technical Manual
- **Canandcolor**
  - API Documentation (Java, C++)
  - Technical Manual

16.6 Especificaciones del dispositivo 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 Direccionamiento

Los nodos FRC CAN asignan ID de arbitraje basados en un esquema predefinido que divide la ID en 5 componentes:

Tipo de dispositivo

Este es un valor de 5 bits que describe el tipo de dispositivo al que se dirige. Una tabla de los tipos de dispositivos asignados actualmente se puede encontrar a continuación. Si desea que se asigne un nuevo tipo de dispositivo del grupo `Reservado`, envíe una solicitud a FIRST.

Tipos de dispositivo	
Mensajes de difusión	0
Controlador del robot	1
Controlador del motor	2
Controlador de relé	3
Sensor giroscópico	4
Acelerómetro	5
Sensor ultrasónico	6
Sensor de dientes de engranaje	7
Módulo de distribución de energía	8
Controlador Neumático	9
Miscellaneous	10
IO Breakout	11
Reservado	12-30
Actualización de firmware	31

Fabricante

Este es un valor de 8 bits que indica el fabricante del dispositivo CAN. Los valores asignados actualmente se pueden encontrar en la tabla a continuación. Si desea que se asigne un ID de fabricante del grupo `Reservado`, envíe una solicitud a FIRST.

Fabricante	
Emisión	0
NI	1
Luminary Micro	2
DEKA	3
CTR Electronics	4
REV Robotics	5
Grapple	6
MindSensors	7
Uso del equipo	8
Kauai Labs	9
Copperforge	10
Playing With Fusion	11
Studica	12
The Thrifty Bot	13
Redux Robotics	14
AndyMark	15
Vivid Hosting	16
Reservado	17-255

API / Identificador de mensaje

El identificador de la API o del mensaje es un valor de 10 bits que identifica un comando o un tipo de mensaje particular. Estos identificadores son únicos para cada combinación de Fabricante + Tipo de Dispositivo (por lo que un identificador de API que puede ser un «Voltage Set» para un Luminary Micro Motor Controller puede ser un «Status Get» para un CTR Electronics Motor Controller o `Current Get` para un CTR Power Distribution Module).

El identificador del mensaje se divide en 2 subcampos: la clase API de 6 bits y el índice API de 4 bits.

Clase API

La clase API es un identificador de 6 bits para una agrupación API. Los mensajes similares se agrupan en una sola clase de API. En la tabla a continuación se muestra un ejemplo de las clases API para el controlador de motor Jaguar.

Clase API	
Modo de control de voltaje	0
Modo de control de velocidad	1
Modo de compensación de voltaje	2
Modo de control de posición	3
Modo de control actual	4
Estado	5
Estado periódico	6
Configuración	7
Ack	8

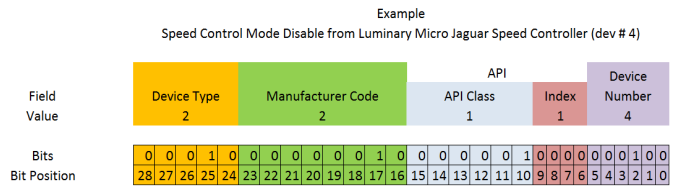
Índice API

El índice API es un identificador de 4 bits para un mensaje en particular dentro de una clase API. En la tabla a continuación se muestra un ejemplo de los valores del índice API para la clase API de control de velocidad del controlador de motor Jaguar.

Índice API	
Control Enable	0
Control Disable	1
Establecer punto de ajuste	2
Constante P	3
Constante I	4
Constante D	5
Establecer referencia	6
Trusted Enable	7
Confianza sin reconocimiento	8
Punto de ajuste de confianza sin reconocimiento	10
Establecer punto de ajuste sin reconocimiento	11

Número del dispositivo

Número de dispositivo es una cantidad de 6 bits que indica el número del dispositivo de un tipo particular. Los dispositivos deben tener el ID de dispositivo 0 para que coincida con otros componentes del sistema de control FRC. El dispositivo 0x3F puede reservarse para mensajes de difusión específicos del dispositivo.



16.6.2 Marcos Protegidos

Los nodos FRC CAN que implementan la capacidad de control del actuador (controladores de motor, relés, controladores neumáticos, etc.) deben implementar una forma de verificar que el robot esté habilitado y que los comandos se originen con el controlador principal del robot (es decir, el roboRIO).

16.6.3 Mensajes de difusión

Los mensajes de difusión son mensajes enviados a todos los nodos estableciendo el tipo de dispositivo y los campos del fabricante en 0. La clase de API para mensajes de difusión es 0. Los mensajes de difusión definidos actualmente se muestran en la tabla a continuación:

Descripción	
Inhabilitar	0
Sistema detenido	1
Reinicio de sistema	2
Asignar dispositivo	3
Consulta de dispositivo	4
Heartbeat	5
Sincronizar	6
Actualizar	7
Versión de firmware	8
Enumerar	9
Reanudar sistema	10

Devices should disable immediately when receiving the Disable message (arbID 0). Implementation of other broadcast messages is optional.

16.6.4 Requisitos para los nodos FRC CAN

Para que los nodos CAN sean aceptados para su uso en el sistema FRC, deben:

- Comuníquese utilizando ID de arbitraje que coincidan con el formato FRC prescrito:
  - Un tipo de dispositivo CAN emitido válido (según la Tabla 1 - Tipos de dispositivo CAN)
  - Una ID de fabricante válida y emitida (según la Tabla 2 - Códigos de fabricante de CAN)
  - Clase(s) e índice(s) API asignados y documentados por el fabricante del dispositivo
  - Un número de dispositivo seleccionable por el usuario si se pretende que varias unidades del tipo de dispositivo coexistan en la misma red.
- Admite los requisitos mínimos de mensajes de transmisión como se detalla en la sección Mensajes de transmisión.
- 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 Universal Heartbeat

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.

Descripción	Byte	Width (bits)
Match time (seconds)	8	8
Match number	6-7	10
Replay number	6	6
Alianza roja	5	1
Habilitado	5	1
Modo autónomo	5	1
Modo de prueba	5	1
System watchdog	5	1
Tipo de torneo	5	3
Tiempo del día (año)	4	6
Tiempo del día (mes)	3-4	4
Tiempo del día (día)	3	5
Tiempo del día (segundos)	2-3	6
Tiempo del día (minutos)	1-2	6
Tiempo del día (horas)	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;
  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.

Tenga en cuenta que todos los campos excepto «Enabled», «Autonomous mode», «Test mode» y «System watchdog» contendrán valores no válidos hasta un tiempo arbitrario después de que se conecte la Driver Station.

17 Programación básica

17.1 Introducción del Control de la Versión Git

Importante: Una guía a mayor profundidad se encuentra disponible en el sitio Web Git.

Git es un sistema de control de versiones distribuido (VCS) creado por Linus Torvalds, también conocido por crear y mantener el kernel de Linux. Version Control es un sistema de seguimiento de cambios de código para desarrolladores. Las ventajas de Git Version Control son:

- Separation of testing environments into *branches*
- Habilidad de navegar a un *commit* particular sin remover el historial
- Habilidad para administrar commits de distintas formas, incluso combinándolos
- Para ver más características, da click aquí

17.1.1 Prerrequisitos

Importante: En el tutorial se utiliza el sistema operativo Windows.

Para descargar e instalar Git utilize los siguientes links:

- Windows
- macOS
- Linux

Nota: Es probable que necesite añadir Git a su ruta

17.1.2 Vocabulario Git

Git revolves around several core data structures and commands:

- **Repositorio:** la estructura de datos de su código, incluyendo la carpeta `.git` en el directorio principal
- **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:** actualiza el repositorio remoto con sus cambios locales
- **Pull:** actualiza el repositorio local con los cambios remotos
- **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 Repositorio

Un repositorio de Git es una estructura de datos que contiene la estructura, historial, y archivos de un proyecto.

Los repositorios de Git usualmente están conformados por:

- Una carpeta `.git`. Esta carpeta contiene diversa información sobre el repositorio.
- Un archivo `.gitignore`. Este archivo contiene los archivos o directorios que *no* quiera incluir cuando actualice el repositorio (commit).
- Archivos y carpetas. Este es el contenido principal del repositorio.

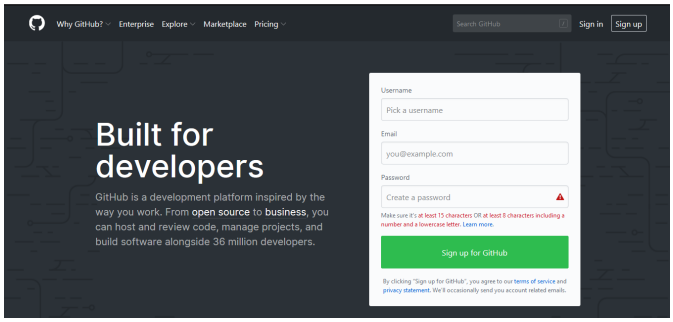
Crear un repositorio

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.

Nota: Hay varios proveedores que pueden ofrecer repositorios. Gitlab y Bitbucket son algunas alternativas a Github.

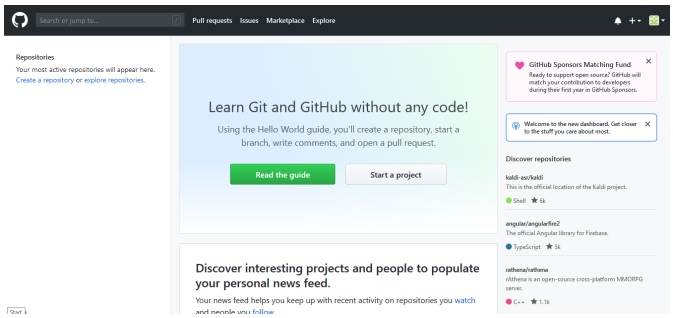
Crear una cuenta de GitHub

Go ahead and create a GitHub account by visiting the website and following the on-screen prompts.

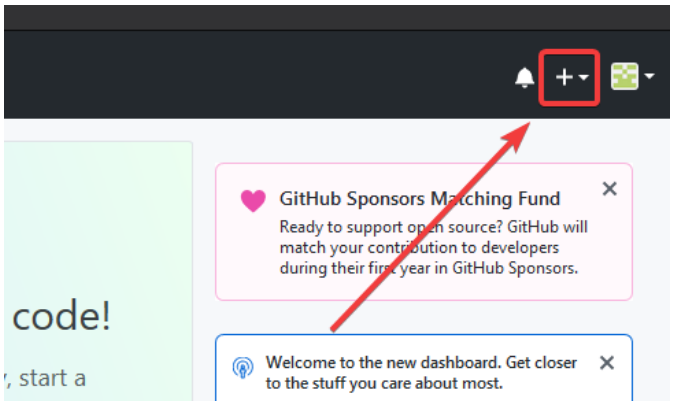


Creación Local

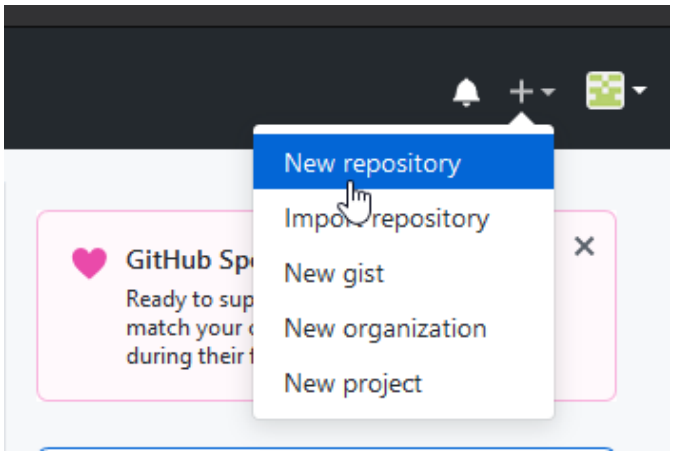
Después de crear y verificar su cuenta, querrá visitar la página de inicio. Se verá similar a lo que muestra la imagen.



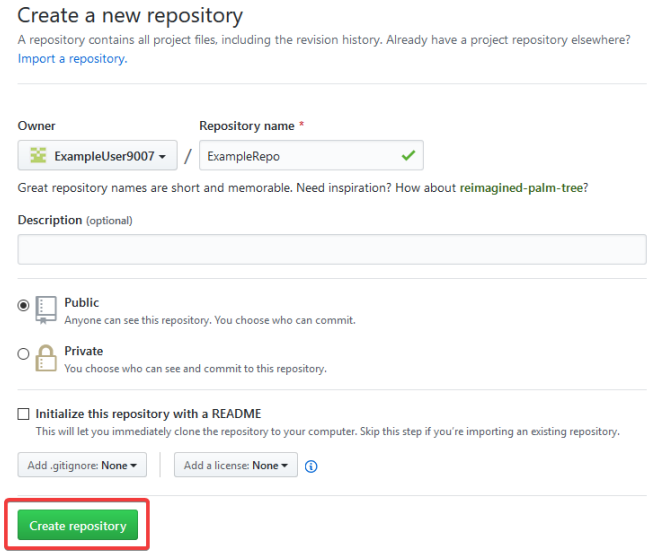
De clic en el ícono de `más` (+) de la esquina superior derecha.



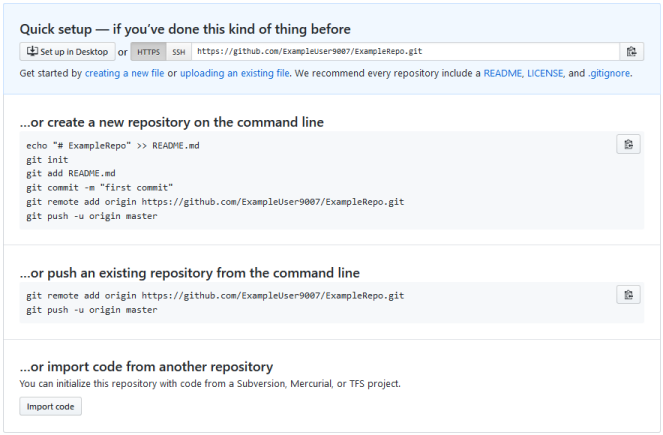
Después de click en *"New Repository"*



Llene con la información correspondiente, y después de clic en *"Create repository"*



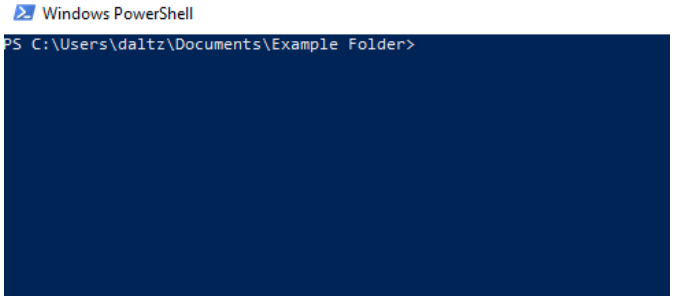
Debe ver una pantalla similar a esta



Nota: The keyboard shortcut `Ctrl+~` can be used to open a terminal in Visual Studio Code

for Windows.

Ahora querrá abrir una ventana de PowerShell y navegar al directorio de su proyecto. Se puede encontrar un excelente tutorial sobre PowerShell aquí.. Consulte su motor de búsqueda sobre cómo abrir un terminal en sistemas operativos alternativos.



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.

Nota: Reemplace la ruta de archivo `"C:\Users\ExampleUser9007\Documents\Example Folder"` con la que desea crear el repositorio y reemplace la URL remota `https://github.com/ExampleUser9007/ExampleRepo.git` con la URL del repositorio que creó en los pasos anteriores.

Empty Directory

```
> 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
```

Existing Project

```
> 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 Commits

Los repositorios están conformados principalmente por commits. Los commits son estados guardados o versiones de código.

En el ejemplo anterior, creamos un archivo llamado README.md. Abra ese archivo en su editor de texto favorito y edite algunas líneas. Después de jugar un poco con el archivo, simplemente guárdelo y ciérrelo. Navegue hasta PowerShell y escriba los siguientes comandos.

```
> 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
```

Nota: Writing good commit messages is a key part of a maintainable project. A guide on writing commit messages can be found here.

Git Pull

Nota: `git fetch` puede ser utilizado cuando el usuario no desea que automáticamente se fusione con la rama en la que se está trabajando

---

Este commando recupera el historial o los commits del repositorio remoto. Cuando el remoto contiene el trabajo que usted no tiene, éste intentará fusionarlo automáticamente. Vea *Fusionar*.

Corra: `git pull`

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.

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.

Corra: `git commit -m "type message here"`

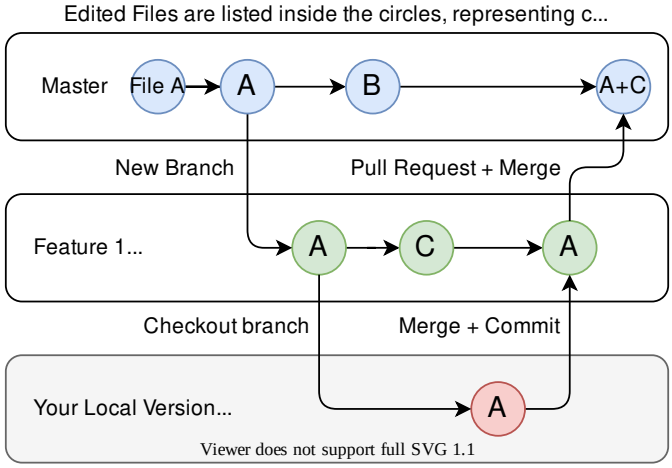
Git Push

Suba (Push) sus cambios locales al repositorio remoto (Cloud)

Corra: `git push`

17.1.5 Ramas

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.



En el ejemplo anterior, main se ramificó (o duplicó) en la rama Característica 1 y alguien revisó la rama, creando una copia local. Luego, alguien confirmó (o subió) sus cambios, fusionándolos en la rama Característica 1. Está «fusionando» los cambios de una rama con otra.

Crear una rama

Corra: `git branch branch-name` donde branch-name es el nombre de la rama a crear. El nuevo historial de la rama se creará de la rama activa actual.

Acceder a una rama

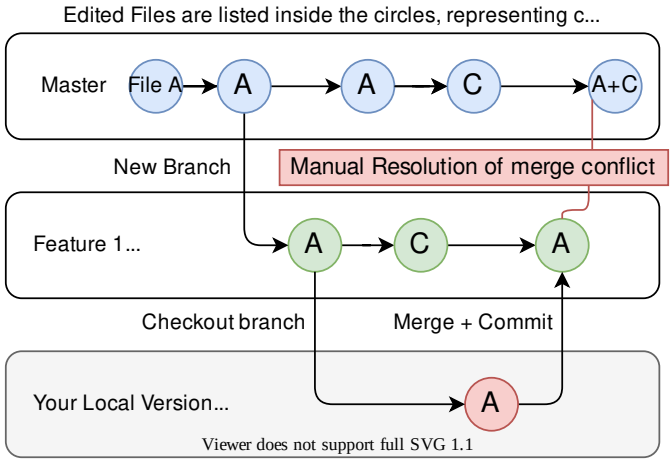
Una vez que la rama está creada, tienes que entrar a la rama.

Corra: `git checkout branch-name` donde branch-name es la rama que fue previamente creada.

17.1.6 Fusionar

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 Resets

Algunas veces, el historial necesita ser reiniciado, o un commit necesita ser deshecho. Esto puede ser realizado de diferentes maneras.

Convirtiendo el Commit

Nota: No puede revertir una fusión, pues git no sabrá que rama u origen deberá escoger.

Para revertir el historial que lleva hacia la ejecución de un commit, corra `git revert commit-id`. Los IDs de commits pueden ser mostrados usando el comando `git log`.

Restaurar el Inicio

Advertencia: Restablecer el inicio a la fuerza, es un comando peligroso. Borra permanentemente todo el historial pasado el objetivo. ¡Ha sido advertido!

Corra: `git reset --hard commit-id`.

17.1.8 Forks

Los forks pueden ser utilizados de manera similar a las ramas. Puede fusionar el upstream (repositorio original) con el origen (forked repositoy).

Clonación de un Repositorio Existente

En el caso de que un repositorio ya esté creado y almacenado en un control remoto, puede clonarlo usando

```
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*.

Actualizar un Fork

1. Agregue el upstream: *git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPO*
2. Confirme que se ha agregado a través de `git remote -v`
3. Obtenga los cambios del upstream: `git fetch upstream`
4. Fusione los cambios con el inicio: `git merge upstream/upstream-branch-name`

17.1.9 Gitignore

Importante: Es extremadamente importante que los equipos **no** modifiquen el archivo `.gitignore` que se incluye con su proyecto de robot. Esto puede provocar que la implementación sin conexión no funcione.

Un archivo `.gitignore` se usa comúnmente como una lista de archivos para no confirmar automáticamente con `git add`. Cualquier archivo o directorio listado en este archivo **no** se confirmará. Tampoco aparecerán con git status.

Additional Information can be found here.

Ocultar una Carpeta

Simplemente añada una nueva línea que contenga la carpeta a ocultar, con una diagonal al final

EJ: `directory-to-exclude/`

Ocultar un Archivo

Añada una nueva línea con el nombre del archivo a ocultar, incluyendo algún directorio prependiente relativo a la raíz del repositorio.

EJ: `directory/file-to-hide.txt`

EJ: `file-to-hide2.txt`

17.1.10 Información adicional

Un tutorial mucho más a fondo puede ser encontrado en el sitio web official de git website.

Una guía para corregir los errores comunes puede ser encontrada en el repositorio de **`reglas de vuelo<https://github.com/k88hudson/git-flight-rules/blob/master/README.md>`_** repository.

17.2 Biblioteca de unidades de 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 Usar la Biblioteca de Unidades

La biblioteca de las unidades es una biblioteca de sólo cabecera. Debe incluir la cabecera correspondiente en sus archivos fuente para las unidades que desea utilizar. Aquí tienes una lista de las unidades 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>
#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>
```

El encabezado `units/math.h` proporciona funciones de reconocimiento de unidades como `units::math::abs()`.

Tipos de Unidades y Tipos de Contenedores

Las librerías de unidades de C++ están basadas en dos clases de definiciones: tipos de unidades y tipos de contenedores.

Tipos de Unidades

Los tipos de unidades corresponden al concepto abstracto de una unidad, sin que tenga un valor actual guardado. Unit types son el "bloque de construcción" fundamental de las librerías de unidad - todos los tipos de unidades son definidos constructivamente (usando la plantilla `compound_unit`) de un número pequeño de unit types «básicas» (como `meters`, `seconds`, 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.

Tipos de contenedores

Los tipos de contenedores corresponden a una cantidad actual dimensionada de acuerdo con cierta unidad – que es, ellos son los que contienen realmente el valor numérico. Container types son hechos de unit types con el modelo `unit_t`. La mayoría de los unit types tienen un container type que les correponde que tiene el mismo sufijo de nombre por `_t` - por ejemplo, el unit type `units::meter` corresponde al container type `units::meter_t`.

Cuando una cantidad específica de unidad es utilizada (como variable o un parámetro de método), va a ser una instancia del tipo contenedor/container type. Por defecto, los tipos contenedores pueden almacenar el valor actual como `double` - usuarios avanzados pueden cambiar esto llamando al modelo `unit_t` manualmente.

Una lista completa de tipos unitarios y contenedores puede ser encontrada en la documentación.

Creando instancias de unidades

Para crear una instancia de una unidad específica, creamos una instancia con su tipo contenedor:

```
// The variable speed has a value of 5 meters per second.
units::meter_per_second_t speed{5.0};
```

Una alternativa es que la librería de unidades tiene tipos literales definidos por algunos de los tipos contenedores más comunes. Éstos pueden ser usados en conjunción con tipo de inferencia vía `auto` para definir una unidad más sucintamente:

```
// The variable speed has a value of 5 meters per second.
auto speed = 5_mps;
```

Las unidades también se pueden inicializar usando un valor de otro tipo contenedor, mientras que los tipos puedan ser convertidos en medio de los otros. Por ejemplo, un valor `meter_t` puede ser creado desde un valor `foot_t`.

```
auto feet = 6_ft;
units::meter_t meters{feet};
```

DE hecho, todos los tipos contenedor representando tipo unidad convertibles son *implícitamente convertibles*. Así, lo siguiente es perfectamente legal:

```
units::meter_t distance = 6_ft;
```

En breve, usamos *ninguna* unidad de medida en lugar de *otra* unidad de medida, en ningun lugar del codigo; las librerias de unidades pueden automáticamente realizar la conversión por nosotros.

Realizando aritmética con unidades

Los tipo contenedores soportan todas las operaciones aritméticas ordinarias de su tipo de datos subyacente, con la condición añadida que la operación debe estar *dimensionalmente* sólido. Así, las adiciones siempre tienen que estar realizadas en dos tipos de contenedores 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 multiplicación debe de ser realizada en cualquier par de tipo contenedor, y dar al tipo contenedor de una unidad compuesta:

Nota: Cuando un calculo da un tipo de unidad compuesta, éste tipo será solo revisado para validar en el punto de operación si el tipo resultado es especificado explícitamente. Si `auto` es utilizado, esta verificación no ocurrirá. Por ejemplo, cuando nosotros dividimos distancia sobre tiempo, deberíamos querer asegurar que el resultado, en efecto, es velocidad (por ejemplo, `units::meter_per_second_t`). Si el tipo resultante es declarado como `auto`, esta verificación no se hará.

```
// 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
```

<cmath> Funciones

Algunas funciones `std` (como `clamp`) son plantillas que aceptan cualquier tipo que realice las operaciones aritméticas. Las cantidades almacenadas como tipo contenedores pueden funcionar con estas funciones sin ningún problema.

De cualquier manera, otras funciones `std` funcionan solo con tipos numéricos ordinarios (por ejemplo, `double`). Los espacios para el nombre de las librerías unitarias `units::math``contienen envolvedores para algunas de estas funciones que aceptan unidades. Ejemplos de estas funciones incluyen ``sqrt`, `pow`, etc.

```
auto area = 36_sq_m;
units::meter_t sideLength = units::math::sqrt(area);
```

Quitar el envoltorio de la unidad

To convert a container type to its underlying value, use the `value()` method. This serves as an escape hatch from the units type system, which should be used only when necessary.

```
units::meter_t distance = 6.5_m;
double distanceMeters = distance.value();
```

17.2.2 Ejemplo de librería de unidades en el código WPILib

Algunos argumentos de métodos en nuevas funciones de WPILib (ex. *cinemáticas*) usan las librerías de unidad. Aquí hay un ejemplo de *probando una trayectoria*.

```
// 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);
```

Algunas clases de WPILib representan objetos que pueden trabajar naturalmente con múltiples opciones de tipos de unidades - por ejemplo, un perfil de movimiento puede operar en cualquier distancia lineal (por ejemplo, metros) o distancia angular (por ejemplo, radianes). Para esas clases, el tipo unitario es requerido como un parámetro modelo:

```
// 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};
```

Para documentación más detallada, por favor visite laágina oficial de GitHub para las librerías de unidad.

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.

Unidad
: Las unidades son realizaciones específicas de dimensiones. Son la forma de expresar cantidades físicas. Cada dimensión tiene una unidad base, como el metro para la longitud, el segundo para el tiempo, el kilogramo para la masa. Las unidades derivadas se forman combinando unidades base, como metros por segundo para la velocidad.

Medida
: Las medidas son la magnitud específica de las cantidades físicas, expresadas en una unidad particular. Por ejemplo, 5 metros es una medida de distancia.

Estos conceptos se utilizan dentro de la Biblioteca de Unidades. Por ejemplo, la **medida** *10 segundos* tiene una magnitud de 10, la **dimensión** es tiempo y la **unidad** es segundos.

17.3.2 Usar la Biblioteca de Unidades

La biblioteca de unidades de Java está disponible en el paquete «edu.wpi.first.units». Las clases más relevantes son:

- 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.

Nota: Se recomienda importar estáticamente `edu.wpi.first.units.Units.*` para tener acceso completo a todas las unidades predefinidas.

Genéricos de java

Las unidades de medida pueden ser expresiones complejas que involucran varias dimensiones, como distancia, tiempo y velocidad. Los parámetros de tipo genérico,<https://docs.oracle.com/javase/tutorial/java/generics/index.html>`__ anidados permiten la definición de unidades que pueden representar dichas expresiones complejas. Los genéricos se utilizan para mantener la biblioteca concisa, reutilizable y extensible, pero tienden a ser verbosos debido a la sintaxis de los genéricos de Java.

Por ejemplo, considera el tipo `Measure<Velocity<Distance>>`. Este tipo representa una medida para la velocidad, donde la velocidad misma se expresa como una unidad de distancia por unidad de tiempo. Esta estructura anidada permite la representación de unidades como metros por segundo o pies por minuto. De manera similar, el tipo `Measure<Per<Voltage, Velocity<Distance>>>` representa una medida para una relación de voltaje a velocidad. Este tipo es útil para representar cantidades como voltios por metro por segundo, la unidad de medida para algunos ganancias de *feedforward*.

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);
```

> **Advertencia:** 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 {
  // 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(
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      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;
  }
}
```

Advertencia: `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 {
    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)));
}
```

Nota: 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.

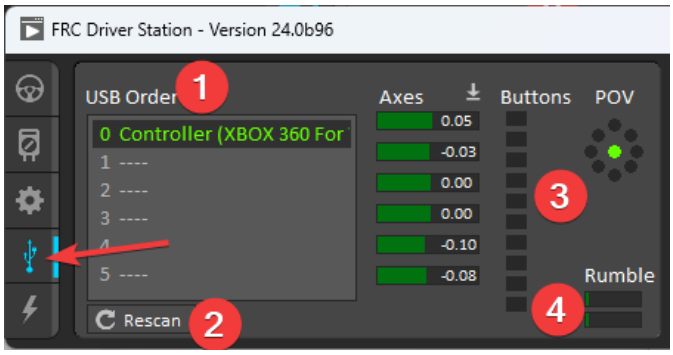
```
public class ElectricCharge extends Unit<ElectricCharge> {
  public ElectricCharge(double baseUnitEquivalent, String name, String symbol) {
    super(ElectricCharge.class, baseUnitEquivalent, name, symbol);
  }

  // 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 Joysticks

A joystick can be used with the Driver Station program to control the robot. Almost any «controller» that can be recognized by Windows can be used as a joystick. Joysticks are accessed using the `GenericHID` class. This class has three relevant subclasses for preconfigured joysticks. You may also implement your own for other controllers by extending `GenericHID`. The first is `Joystick` which is useful for standard flight joysticks. The second is `XboxController` which works for the Xbox 360, Xbox One, or Logitech F310 (in XInput mode). Finally, the `PS4Controller` class is ideal for using that controller. Each axis of the controller ranges from -1 to 1.

The command based way to use the these classes is detailed in the section: *Enlazando comandos a Triggers*.

17.4.1 Driver Station Joysticks



The *USB Devices Tab* of the Driver Station is used to setup and configure the joystick for use with the robot. Pressing a button on a joystick will cause its entry in the table to light up green. Selecting the joystick will show the values of axes, buttons and the POV that can be used to determine the mapping between physical joystick features and axis or button numbers.

The USB Devices Tab also assigns a joystick index to each joystick. To reorder the joysticks simply click and drag. The Driver Station software will try to preserve the ordering of devices between runs. It is a good idea to note what order your devices should be in and check each time you start the Driver Station software that they are correct.

When the Driver Station is in disabled mode, it is routinely looking for status changes on the joystick devices. Unplugged devices are removed from the list and new devices are opened and added. When not connected to the FMS, unplugging a joystick will force the Driver Station into disabled mode. To start using the joystick again: plug the joystick in, check that it shows up in the right spot, then re-enable the robot. While the Driver Station is in enabled mode, it will not scan for new devices. This is a time consuming operation and timely update of signals from attached devices takes priority.

Nota: For some joysticks the startup routine will read whatever position the joysticks are in as the center position, therefore, when the computer is turned on (or when the joystick is plugged in) the joysticks should be at their center position.

When the robot is connected to the Field Management System at competition, the Driver Station mode is dictated by the *FMS*. This means that you cannot disable your robot and the DS cannot disable itself in order to detect joystick changes. A manual complete refresh of the joysticks can be initiated by pressing the F1 key on the keyboard. Note that this will close and re-open all devices, so all devices should be in their center position as noted above.

17.4.2 `Joystick` Class

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
```

The `Joystick` class is designed to make using a flight joystick to operate the robot significantly easier. Depending on the flight joystick, the user may need to set the specific X, Y, Z, and Throttle channels that your flight joystick uses. This class offers special methods for accessing the angle and magnitude of the flight joystick.

Importante: 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 XboxController Class

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.

Importante: 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 PS4Controller Class

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
```

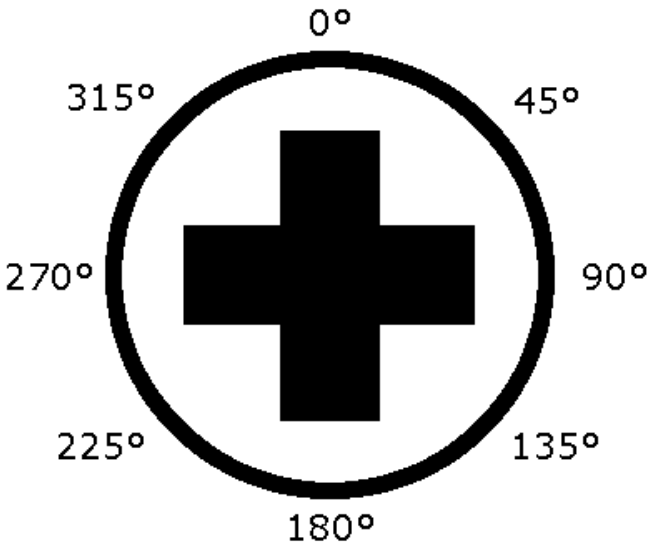
PYTHON

```
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)`.

Importante: 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 GenericHID Usage

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 Button Usage

Nota: Usage such as the following is for code not using the command-based framework. For button usage in the command-based framework, see *Enlazando comandos a Triggers*.

Unlike an axis, you will usually want to use the `pressed` and `released` methods to respond to button input. These will return true if the button has been activated since the last check. This is helpful for taking an action once when the event occurs but not having to continuously do it while the button is held down.

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()
```

A common request is to toggle something on and off with the press of a button. Toggles should be used with caution, as they require the user to keep track of the robot state.

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++

```
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

```
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.

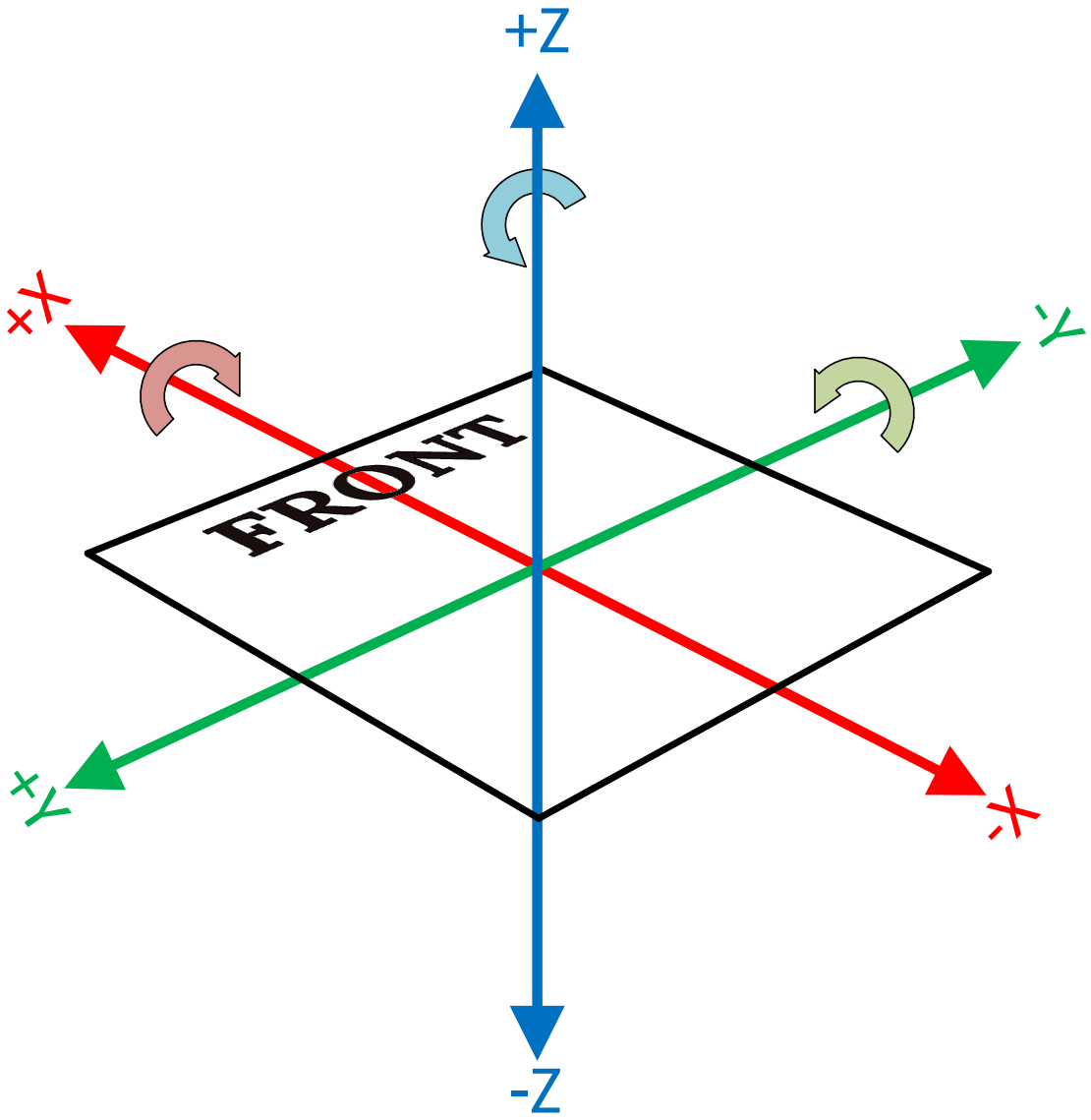


Figura 1: Robot coordinate system in three dimensions

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.

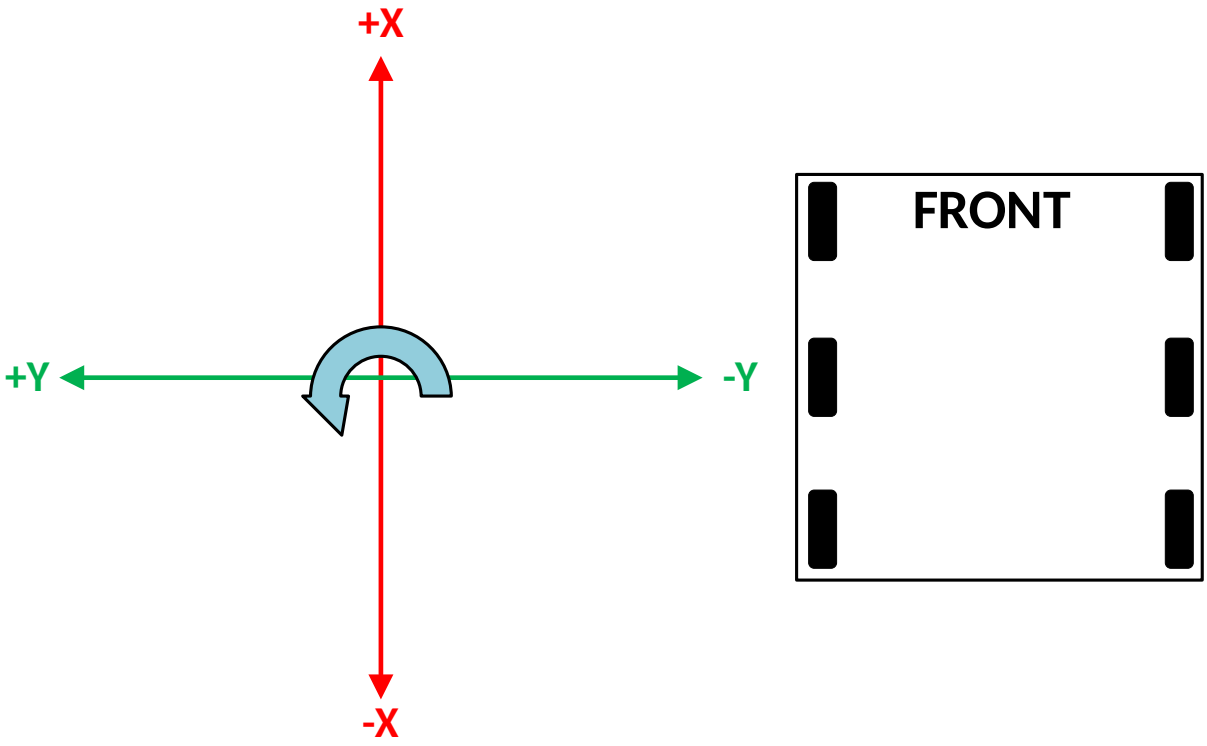


Figura 2: Robot coordinate system in two dimensions

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).

Nota: 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.

Advertencia: 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.

Advertencia: 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 rota-

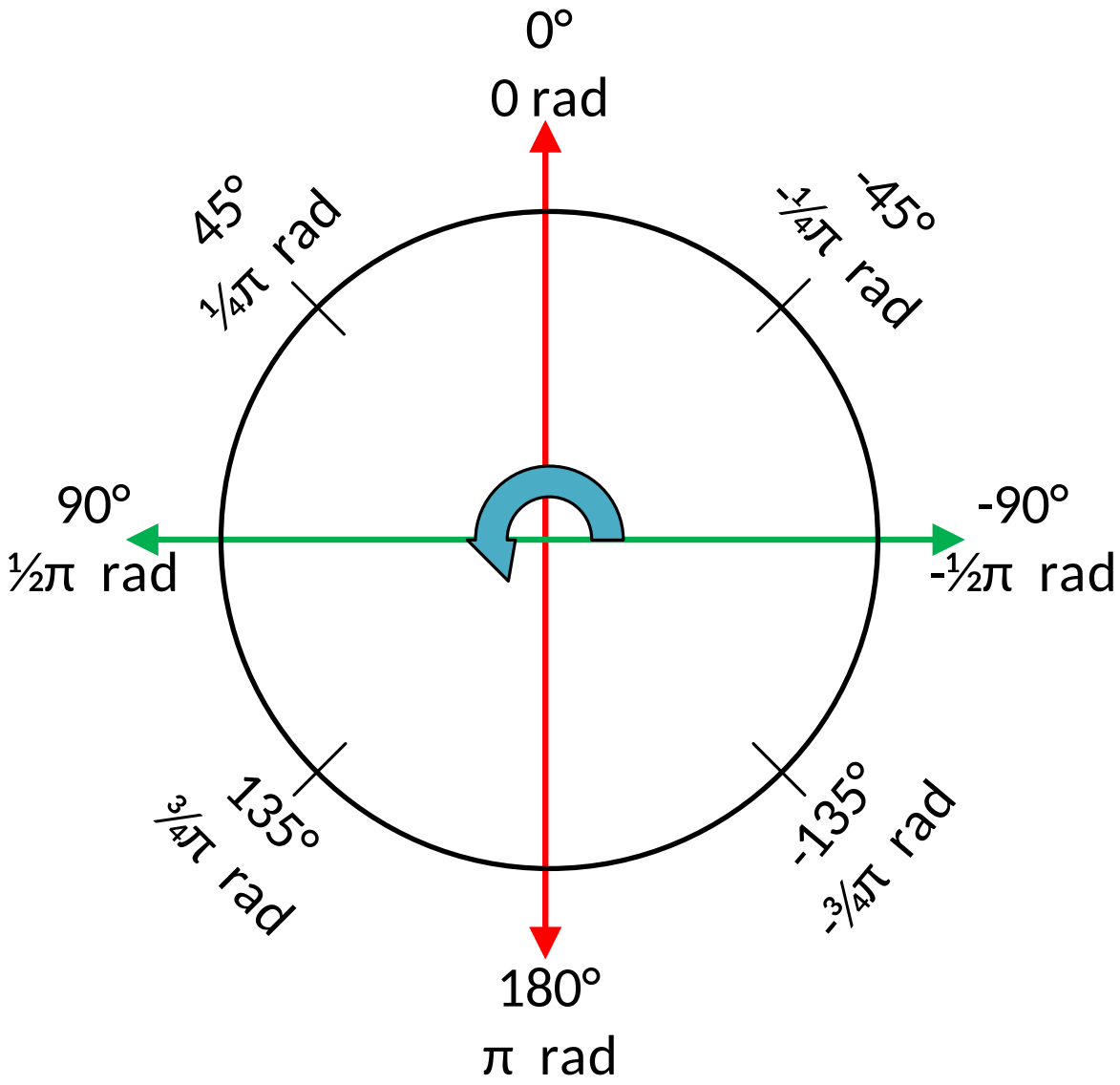


Figura 3: Unit circle with common angles

ting 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.

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.

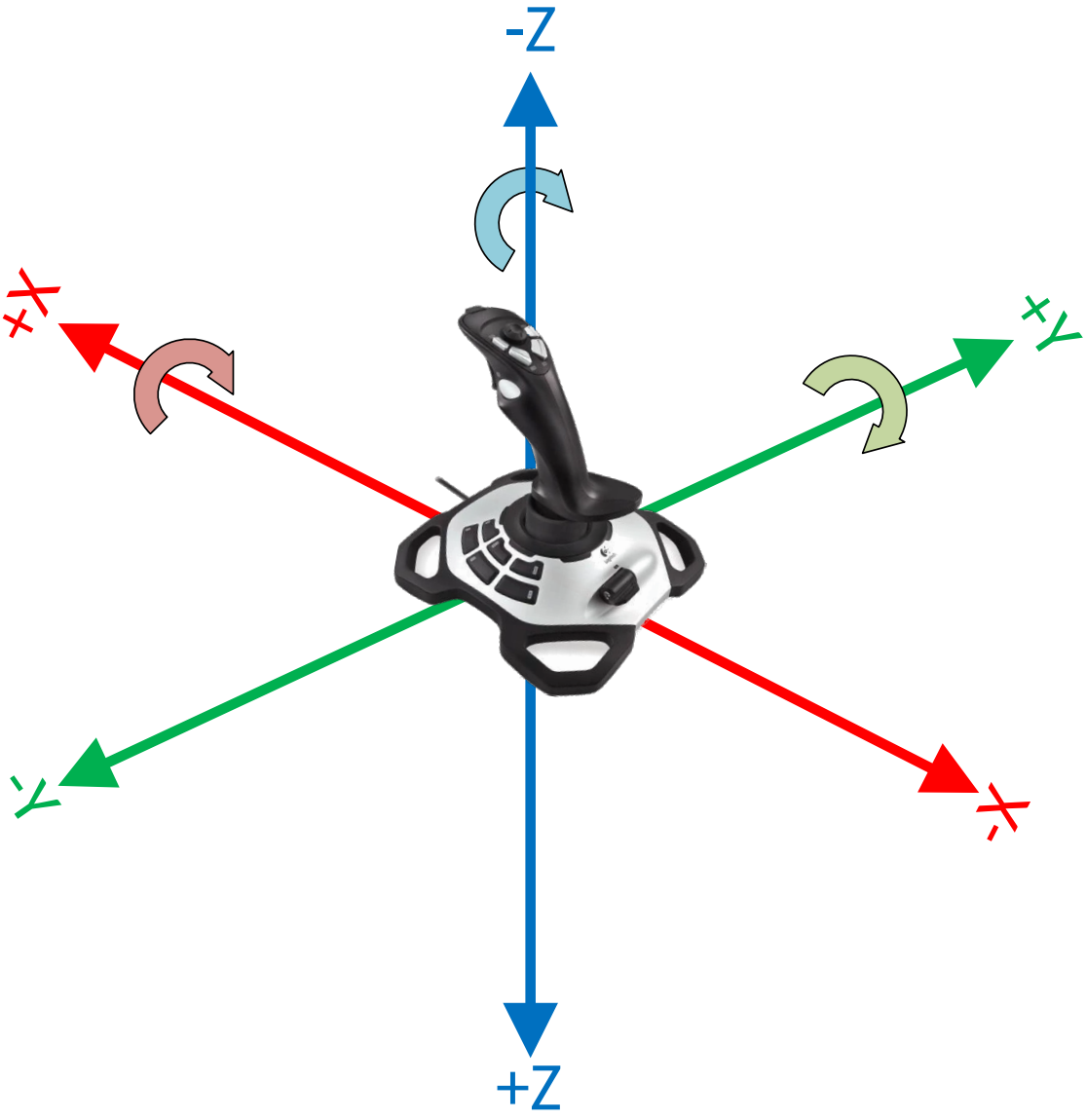


Figura 4: Joystick coordinate system

JAVA

```java
public void teleopPeriodic() {
    // Arcade drive with a given forward and turn rate
    myDrive.arcadeDrive(-driveStick.getY(), -driveStick.getX());
}
```

C++

```cpp
void TeleopPeriodic() override {
    // Arcade drive with a given forward and turn rate
    myDrive.ArcadeDrive(-driveStick.GetY(), -driveStick.GetX());
}
```

PYTHON

```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.

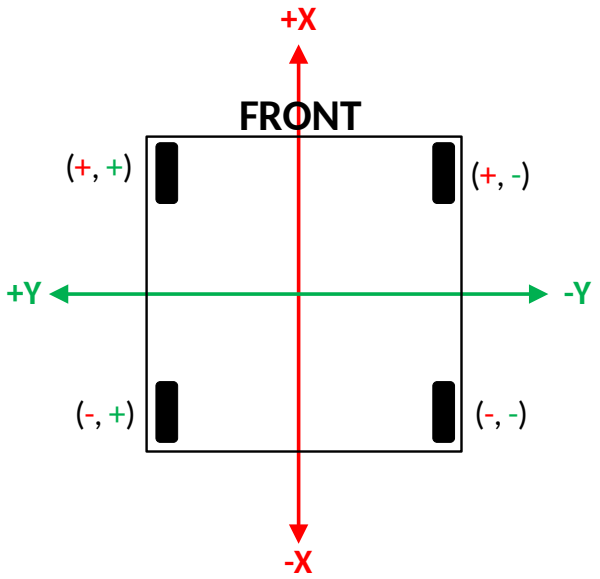


Figura 5: Kinematics with translation signs

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")

Advertencia: 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.

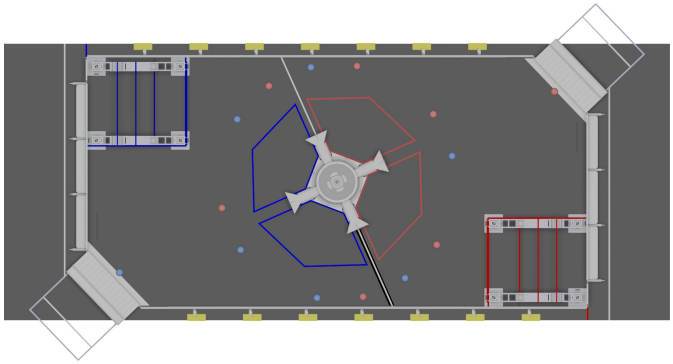


Figura 6: Rotated field from RAPID REACT in 2022[1]

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

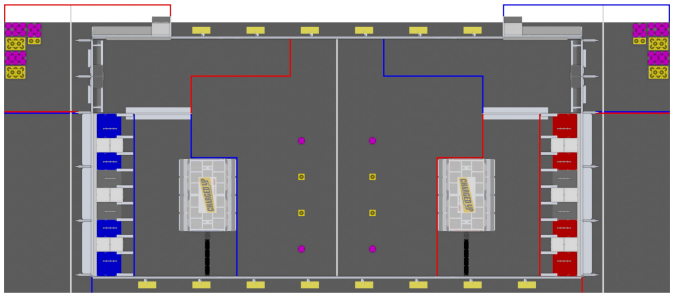


Figura 7: Mirrored field from CHARGED UP in 2023[2]

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.

Advertencia: 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.

Nota: 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 *Operation Tab*.

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.

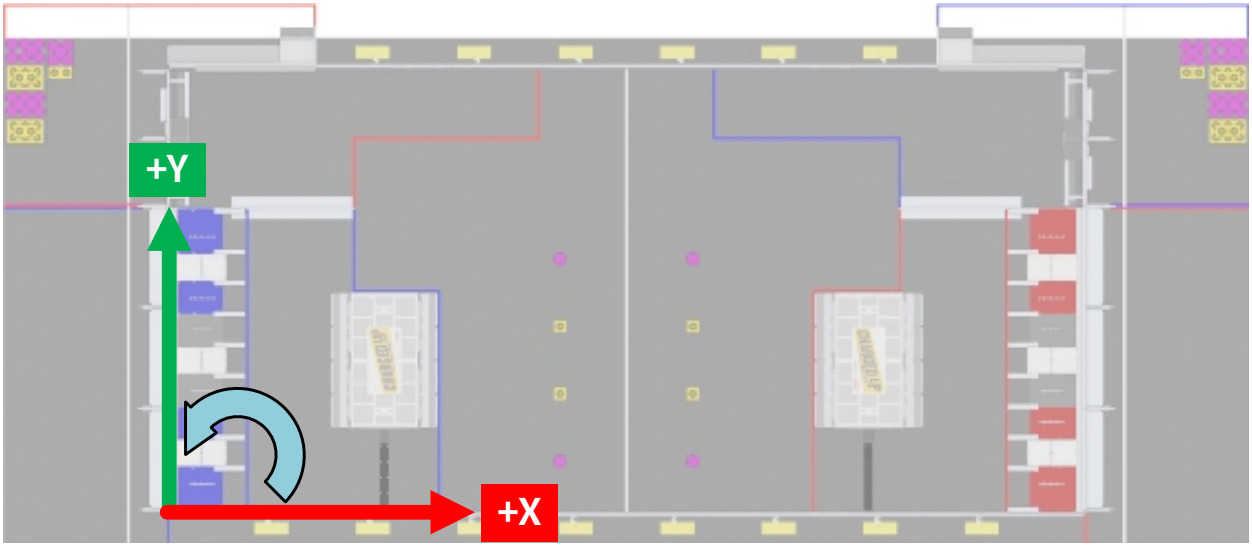


Figura 8: CHARGED UP with blue origin

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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:

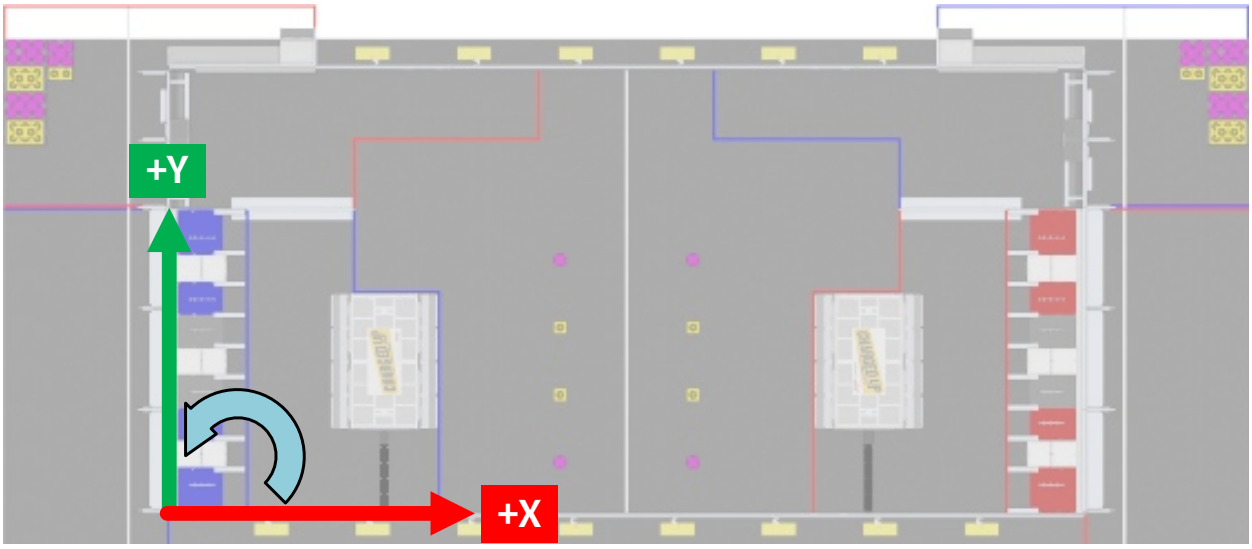


Figura 9: CHARGED UP field with blue alliance as origin

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.

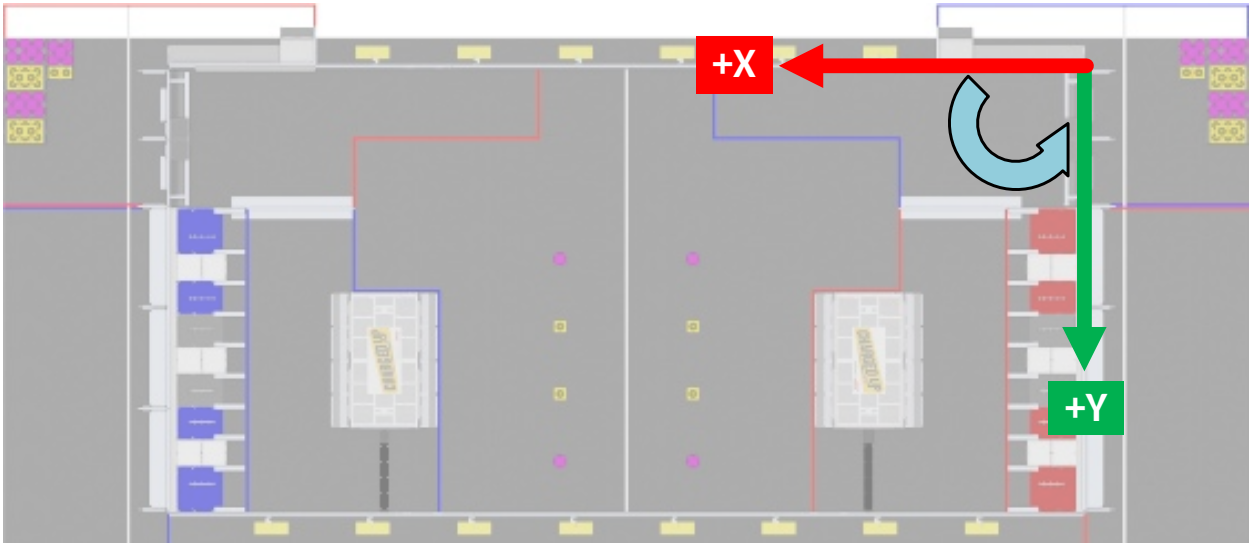


Figura 10: CHARGED UP field with red alliance as origin

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 Setting Robot Preferences

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.

This example shows how to utilize Preferences to change the setpoint of a PID controller and the P constant. The code examples are adapted from the Arm Simulation example (Java, C++). You can run the Arm Simulation example in the Robot Simulator to see how to use the preference class and interact with it using the dashboards without needing a robot.

17.6.1 Initializing Preferences

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
↪'t already exist
        wpilib.Preferences.initDouble(
            Constants.kArmPositionKey, self.armSetpointDegrees
        )
        wpilib.Preferences.initDouble(Constants.kArmPKey, self.armKp)
```

Preferences are stored using a name, the key. It's helpful to store the key in a constant, like `kArmPositionKey` and `kArmPKey` in the code above to avoid typing it multiple times and avoid typos. We also declare variables, `kArmKp` and `armPositionDeg` to hold the data retrieved from preferences.

In `robotInit`, each key is checked to see if it already exists in the Preferences database. The `containsKey` method takes one parameter, the key to check if data for that key already exists in the preferences database. If it doesn't exist, a default value is written. The `setDouble` method takes two parameters, the key to write and the data to write. There are similar methods for other data types like booleans, ints, and strings.

If using the Command Framework, this type of code could be placed in the constructor of a Subsystem or Command.

17.6.2 Reading Preferences

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.

Depending on the data that is stored in preferences, you can use it when you read it, such as the proportional constant above. Or you can store it in a variable and use it later, such as the setpoint, which is used in `telopPeriodic` below.

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();
  }
}
```

```
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});
}
```

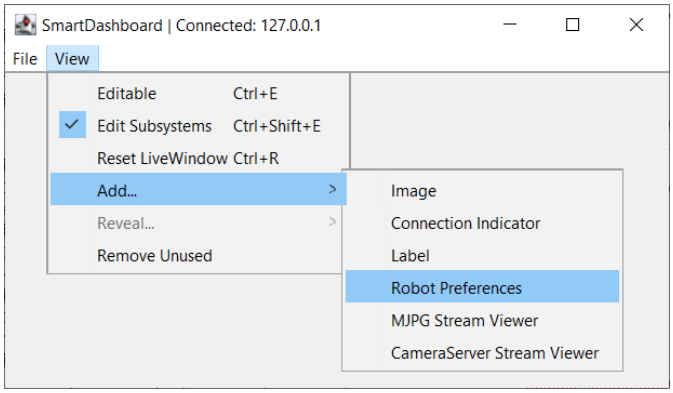
Python

```
    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()
```

```
    def reachSetpoint(self):
        pidOutput = self.controller.calculate(
            self.encoder.getDistance(),
            units.degreesToRadians(self.armSetpointDegrees),
        )
        self.motor.setVoltage(pidOutput)
```

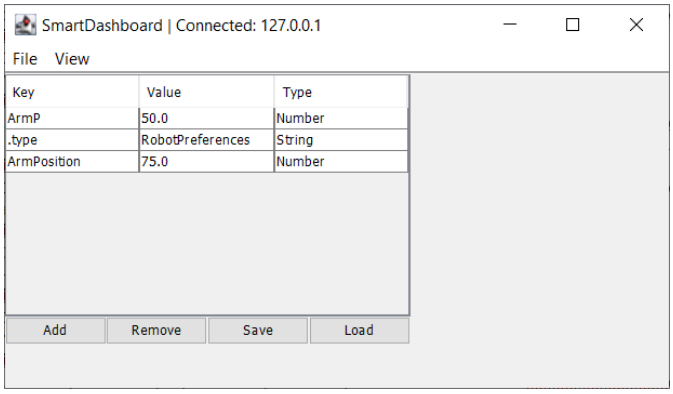
17.6.3 Using Preferences in SmartDashboard

Displaying Preferences in SmartDashboard



In the SmartDashboard, the Preferences display can be added to the display by selecting *View* then *Add...* then *Robot Preferences*. This reveals the contents of the preferences file stored in the roboRIO flash memory.

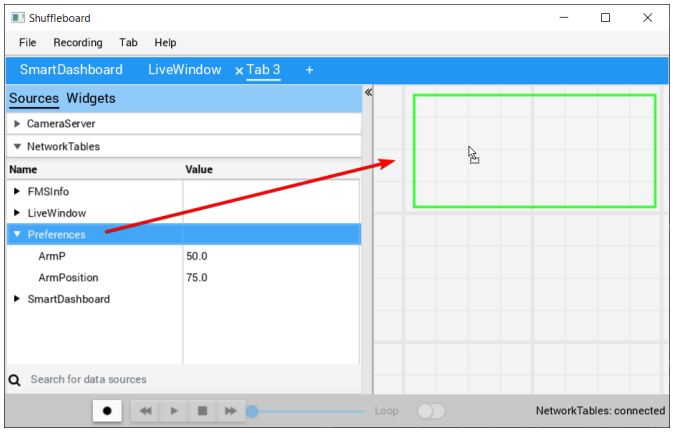
Editing Preferences in SmartDashboard



The values are shown here with the default values from the code. If the values need to be adjusted they can be edited here and saved.

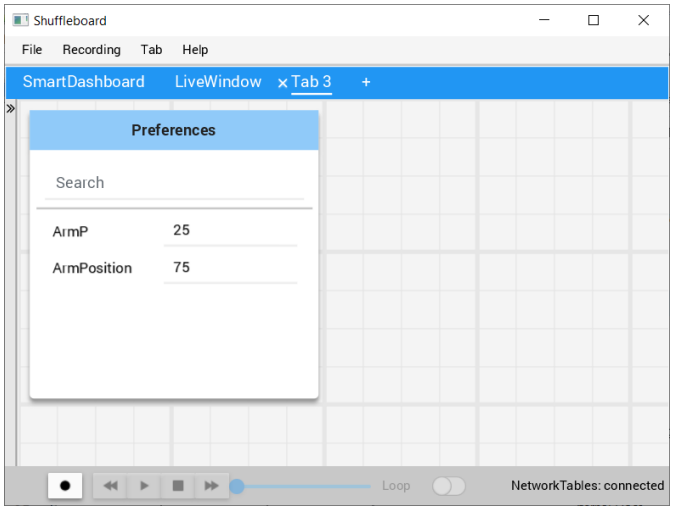
17.6.4 Using Preferences in Shuffleboard

Displaying Preferences in Shuffleboard



In Shuffleboard, the Preferences display can be added to the display by dragging the preferences field from the sources window. This reveals the contents of the preferences file stored in the roboRIO flash memory.

Editing Preferences in Shuffleboard

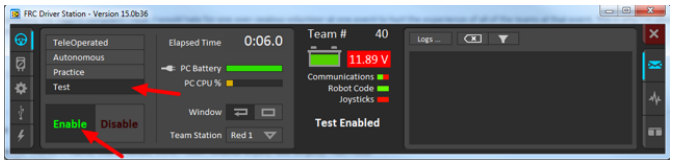


The values are shown here with the default values from the code. If the values need to be adjusted they can be edited here.

17.7 Using Test Mode

Test mode is designed to enable programmers to have a place to put code to verify that all systems on the robot are functioning. In each of the robot program templates there is a place to add test code to the robot.

17.7.1 Enabling Test Mode



Test mode on the robot can be enabled from the Driver Station just like autonomous or teleop. To enable test mode in the Driver Station, select the «Test» button and enable the robot. The test mode code will then run.

17.7.2 Adding Test mode code to your robot code

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.

Adding test mode can be as painless as calling your already written Teleop methods from Test. Or you can write special code to try out a new feature that is only run in Test mode, before integrating it into your teleop or autonomous code. You could even write code to move all motors and check all sensors to help the pit crew!

17.7.3 LiveWindow in Test Mode

Importante: 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: Modo Test y Live Window* for more details.

17.8 Leyendo Stacktraces

```
Ha ocurrido un error inesperado.
```

Cuando el código de su robot encuentra un error inesperado, verá este mensaje aparecer en alguna salida de la consola (Driver Station o RioLog). Probablemente también notará que su robot se detiene abruptamente, o posiblemente nunca se mueva. Estos errores inesperados se denominan *unhandled exceptions*.

Cuando se produce una excepción no controlada, significa que su código tiene uno o más errores que deben corregirse.

Este artículo explorará algunas de las herramientas y técnicas involucradas en encontrar y corregir esos errores.

17.8.1 ¿Qué es un «Stack Trace»?

El mensaje `unexpected error has occurred` es una señal de que se ha impreso un *stack trace*.

In Java and C++, the *call stack* data structure is used to store information about which function or method is currently being executed.

Un *stack trace* imprime información acerca de lo que estaba sucediendo en el programa cuando la excepción sin manejar ocurrió. Esto apunto a las lineas de código que estaban ejecutándose justo antes de que el problema pasara. Mientras que no siempre apunta a los puntos exactos que son la *causa principal* de su problema, normalmente es el mejor lugar para empezar a buscar.

17.8.2 ¿Qué es una «Unhandled Exception»?

Un error irrecuperable es cualquier condición que se manifieste donde el procesador no puede continuar ejecutando código. Casi siempre implica que, a pesar de que el código haya sido compilado y se está ejecutando, no hace sentido que la ejecución continúe.

En casi todos los casos, la raíz de la excepción no manejada es que el código no está correctamente implementado. Casi nunca implica que cualquier parte del hardware no esté funcionando.

17.8.3 ¿Cómo resuelvo mi problema?

Leyendo el Stack Trace

Para empezar, busque arriba de `unexpected error has occurred` para el stack trace.

Java

En Java, debería verse como algo así:

```
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)
```

Hay algunas cosas importantes que entender de aquí:

- Hubo un `Error`
- El error se provocó por una `Excepción no manejada`
- La excepción fue un `java.lang.NullPointerException`
- El error ocurrió mientras se ejecutaba la línea 24 adentro de `Robot.java`
  - `robotInit` era el nombre de el método que se estaba ejecutando cuando el error sucedió.
- `robotInit` es una función en el paquete de `frc.robot.Robot` (AKA, el código de su equipo)
- `robotInit` was called from a number of functions from the `edu.wpi.first.wpilibj` package (AKA, the WPILib libraries)

The list of indented lines starting with the word `at` represent the state of the *stack* at the time the error happened. Each line represents one method, which was *called by* the method right below it.

For example, If the error happened deep inside your codebase, you might see more entries on the stack:

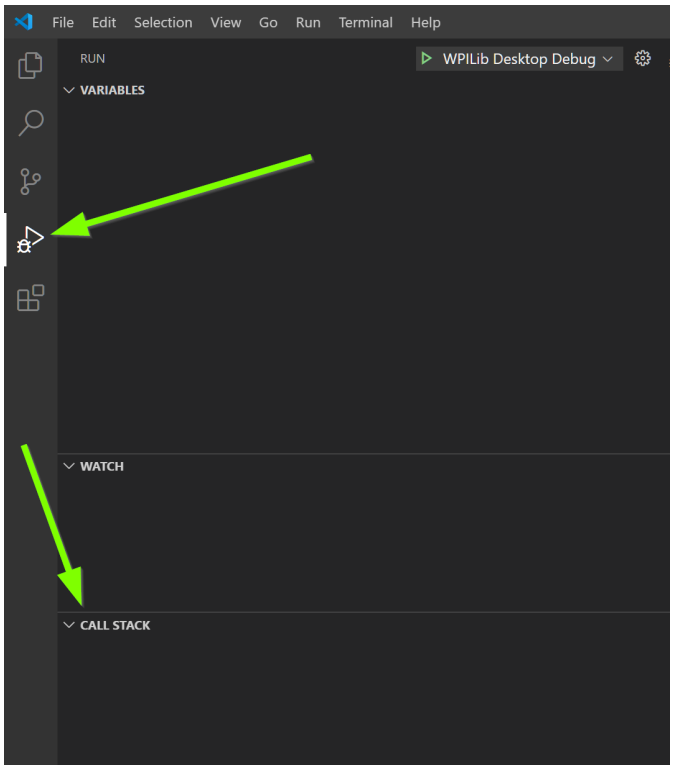
```
Error at frc.robot.Robot.buggyMethod(TooManyBugs.java:1138): Unhandled exception:␣
↪java.lang.NullPointerException
        at frc.robot.Robot.buggyMethod(TooManyBugs.java:1138)
        at frc.robot.Robot.barInit(Bar.java:21)
        at frc.robot.Robot.fooInit(Foo.java:34)
        at frc.robot.Robot.robotInit(Robot.java:24)
        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)
```

In this case: `robotInit` called `fooInit`, which in turn called `barInit`, which in turn called `buggyMethod`. Then, during the execution of `buggyMethod`, the `NullPointerException` occurred.

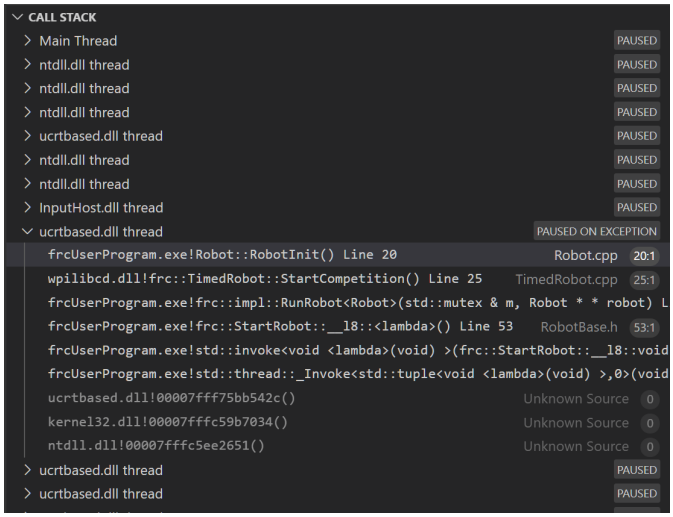
C++

Java will usually produce stack traces automatically when programs run into issues. C++ will require more digging to extract the same info. Usually, a single-step debugger will need to be hooked up to the executing robot program.

Stack traces can be found in the debugger tab of VS Code:



Stack traces in C++ will generally look similar to this:



Hay algunas cosas importantes que entender de aquí:

- The code execution is currently paused.
- The reason it paused was one thread having an `exception`

- The error happened while running line `20` inside of `Robot.cpp`
  - `RobotInit` was the name of the method executing when the error happened.
- `RobotInit` is a function in the `Robot::` namespace (AKA, your team's code)
- `RobotInit` was called from a number of functions from the `frc::` namespace (AKA, the WPILib libraries)

This «call stack» window represents the state of the *stack* at the time the error happened. Each line represents one method, which was *called by* the method right below it.

The examples in this page assume you are running code examples in simulation, with the debugger connected and watching for unexpected errors. Similar techniques should apply while running on a real robot.

Perform Code Analysis

Once you've found the stack trace, and found the lines of code which are triggering the unhandled exception, you can start the process of determining root cause.

Often, just looking in (or near) the problematic location in code will be fruitful. You may notice things you forgot, or lines which don't match an example you're referencing.

Nota: Developers who have lots of experience working with code will often have more luck looking at code than newer folks. That's ok, don't be discouraged! The experience will come with time.

A key strategy for analyzing code is to ask the following questions:

- When was the last time the code «worked» (I.e., didn't have this particular error)?
- What has changed in the code between the last working version, and now?

Frequent testing and careful code changes help make this particular strategy more effective.

Run the Single Step Debugger

Sometimes, just looking at code isn't enough to spot the issue. The *single-step debugger* is a great option in this case - it allows you to inspect the series of events leading up to the unhandled exception.

Search for More Information

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.

Seeking Outside Help

If all else fails, you can seek out advice and help from others (both in-person and online). When working with folks who aren't familiar with your codebase, it's very important to provide the following information:

- Access to your source code, (EX: *on github.com*)
- The **full text** of the error, including the full stack trace.

17.8.4 Common Examples & Patterns

There are a number of common issues which result in runtime exceptions.

Null Pointers and References

Both C++ and Java have the concept of «null» - they use it to indicate something which has not yet been initialized, and does not refer to anything meaningful.

Manipulating a «null» reference will produce a runtime error.

For example, consider the following code:

JAVA

```
PWMSparkMax armMotorCtrl;

@Override
public void robotInit() {
      armMotorCtrl.setInverted(true);
}
```

C++

```
class Robot : public frc::TimedRobot {
   public:
      void RobotInit() override {
         motorRef->SetInverted(false);
      }

   private:
      frc::PWMVictorSPX m_armMotor{0};
      frc::PWMVictorSPX* motorRef;
};
```

When run, you'll see output that looks like this:

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.
```

Reading the stack trace, you can see that the issue happened inside of the `robotInit()` function, on line 23, and the exception involved «Null Pointer».

By going to line 23, you can see there is only one thing which could be null - `armMotorCtrl`. Looking further up, you can see that the `armMotorCtrl` object is declared, but never instantiated.

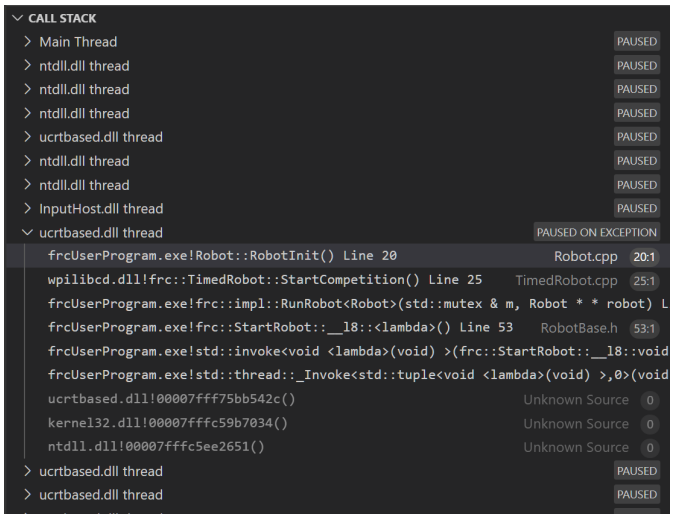
Alternatively, you can step through lines of code with the single step debugger, and stop when you hit line 23. Inspecting the `armMotorCtrl` object at that point would show that it is null.

C++

```
Exception has occurred: W32/0xc0000005
Unhandled exception thrown: read access violation.
this->motorRef was nullptr.
```

In Simulation, this will show up in a debugger window that points to line 20 in the above buggy code.

You can view the full stack trace by clicking the debugger tab in VS Code:



The error is specific - our member variable `motorRef` was declared, but never assigned a value. Therefore, when we attempt to use it to call a method using the `->` operator, the exception occurs.

The exception states its type was `nullptr`.

Fixing Null Object Issues

Generally, you will want to ensure each reference has been initialized before using it. In this case, there is a missing line of code to instantiate the `armMotorCtrl` before calling the `setInverted()` method.

A functional implementation could look like this:

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:
25       frc::PWMVictorSPX m_armMotor{0};
26       frc::PWMVictorSPX* motorRef;
27 };
```

Divide by Zero

It is not generally possible to divide an integer by zero, and expect reasonable results. Most processors (including the roboRIO) will raise an Unhandled Exception.

For example, consider the following code:

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

};
```

When run, you'll see output that looks like this:

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.
```

Looking at the stack trace, we can see a `java.lang.ArithmeticException: / by zero` exception has occurred on line 24. If you look at the two variables which are used on the right-hand side of the = operator, you might notice one of them has been initialized to zero. Looks like someone forgot to update it! Furthermore, the zero-value variable is used in the denominator of a division operation. Hence, the divide by zero error happens.

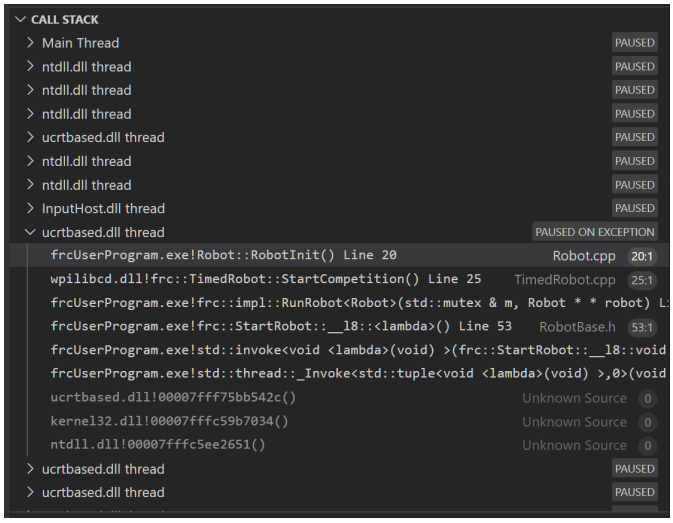
Alternatively, by running the single-step debugger and stopping on line 24, you could inspect the value of all variables to discover `shoulderToElbow_in` has a value of `0`.

C++

```
Exception has occurred: W32/0xc0000094
Unhandled exception at 0x00007FF71B223CD6 in frcUserProgram.exe: 0xC0000094: Integer␣
↪division by zero.
```

In Simulation, this will show up in a debugger window that points to line 20 in the above buggy code.

You can view the full stack trace by clicking the debugger tab in VS Code:



Looking at the message, we see the error is described as `Integer division by zero`. If you look at the two variables which are used on the right-hand side of the = operator on line 20, you might notice one of them has been initialized to zero. Looks like someone forgot to update it! Furthermore, the zero-value variable is used in the denominator of a division operation. Hence, the divide by zero error happens.

Note that the error messages might look slightly different on the roboRIO, or on an operating system other than windows.

Fixing Divide By Zero Issues

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?

Sometimes, you just need to use a different number other than 0.

A functional implementation could look like this:

JAVA

```
int armLengthRatio;
int elbowToWrist_in = 39;
int shoulderToElbow_in = 3;

@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 = 3

};
```

Alternatively, if zero *is* a valid value, adding `if/else` statements around the calculation can help you define alternate behavior to avoid making the processor perform a division by zero.

Finally, changing variable types to be `float` or `double` can help you get around the issue - floating-point numbers have special values like `NaN` to represent the results of a divide-by-zero operation. However, you may still have to handle this in code which consumes that calculation's value.

HAL Resource Already Allocated

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).

For example, consider the following code:

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:
      frc::PWMVictorSPX m_frontLeftMotor{0};
      frc::PWMVictorSPX m_rearLeftMotor{0};

   };
```

When run, you'll see output that looks like this:

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>
```

(continúe en la próxima página)

(proviene de la página anterior)

```
↪(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.
```

This stack trace shows that a `edu.wpi.first.hal.util.AllocationException` has occurred. It also gives the helpful message: `PWM or DIO 0 previously allocated.`.

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++

In C++, you won't specifically see a stacktrace from this issue. Instead, you'll get messages which look like the following:

```
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]
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        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
```

The key thing to notice here is the string, `PWM or DIO 0 previously allocated.`. That string is your primary clue that something in code has incorrectly «doubled up» on pin 0 usage.

The message example above was generated on a roboRIO. If you are running in simulation, it might look different.

Fixing HAL Resource Already Allocated Issues

`HAL: Resource already allocated` are some of the most straightforward errors to fix. Just spend a bit of time looking at the electrical wiring on the robot, and compare that to what's in code.

In the example, the left motor controllers are plugged into *PWM* ports `0` and `1`. Therefore, corrected code would look like this:

JAVA

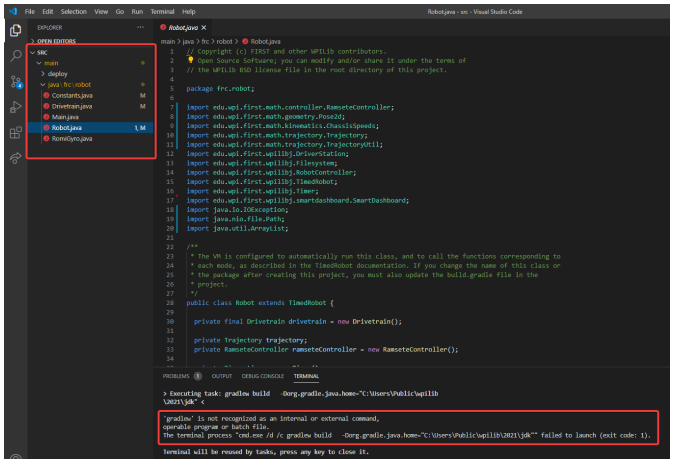
```
19 PWMSparkMax leftFrontMotor;
20 PWMSparkMax leftRearMotor;
21
22 @Override
23 public void robotInit() {
24
25    leftFrontMotor = new PWMSparkMax(0);
26    leftRearMotor = new PWMSparkMax(1);
27
28 }
```

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 is not recognized...

`gradlew is not recognized as an internal or external command` is a common error that can occur when the project or directory that you are currently in does not contain a `gradlew` file. This usually occurs when you open the wrong directory.



In the above screenshot, you can see that the left-hand sidebar does not contain many files. At a minimum, VS Code needs a couple of files to properly build and deploy your project.

- `gradlew`
- `build.gradle`
- `gradlew.bat`

If you do not see any one of the above files in your project directory, then you have two possible causes.

- A corrupt or bad project.
- You are in the wrong directory.

Fixing gradlew is not recognized...

`gradlew is not recognized...` is a fairly easy problem to fix. First identify the problem source:

Are you in the wrong directory? - Verify that the project directory is the correct directory and open this.

Is your project missing essential files? - This issue is more complex to solve. The recommended solution is to *recreate your project* and manually copy necessary code in.

17.9 Treating Functions as Data

Regardless of programming language, one of the first things anyone learns to do when programming a computer is to write a function (also known as a «method» or a «subroutine»). Functions are a fundamental part of organized code - writing functions lets us avoid duplicating the same piece of code over and over again. Instead of writing duplicated sections of code, we call a single function that contains the code we want to execute from multiple places (provided we named the function well, the function name is also easier to read than the code itself!). If the section of code needs some additional information about its surrounding context to run, we pass those to the function as «parameters», and if it needs to yield something back to the rest of the code once it finishes, we call that a «return value» (together, the parameters and return value are called the function's «signature»);

Sometimes, we need to pass functions from one part of the code to another part of the code. This might seem like a strange concept, if we're used to thinking of functions as part of a class definition rather than objects in their own right. But at a basic level, functions are just data - in the same way we can store an `integer` or a `double` as a variable and pass it around our program, we can do the same thing with a function. A variable whose value is a function is called a «functional interface» in Java, and a «function pointer» or «functor» in C++.

17.9.1 Why Would We Want to Treat Functions as Data?

Typically, code that calls a function is coupled to (depends on) the definition of the function. While this occurs all the time, it becomes problematic when the code *calling* the function (for example, WPILib) is developed independently and without direct knowledge of the code that *defines* the function (for example, code from an FRC team). Sometimes we solve this challenge through the use of class interfaces, which define collections of data and functions that are meant to be used together. However, often we really only have a dependency on a *single function*, rather than on an *entire class*.

For example, WPILib offers several ways for users to execute certain code whenever a joystick button is pressed - one of the easiest and cleanest ways to do this is to allow the user to *pass a function* to one of the WPILib joystick methods. This way, the user only has to write the code that deals with the interesting and team-specific things (e.g., «move my robot arm») and not the boring, error-prone, and universal thing («properly read button inputs from a standard joystick»).

For another example, the *Command-based framework* is built on `Command` objects that refer to methods defined on various `Subsystem` classes. Many of the included `Command` types (such as `InstantCommand` and `RunCommand`) work with *any* function - not just functions associated with a single `Subsystem`. To support building commands generically, we need to support passing

functions from a `Subsystem` (which interacts with the hardware) to a `Command` (which interacts with the scheduler).

In these cases, we want to be able to pass a single function as a piece of data, as if it were a variable - it doesn't make sense to ask the user to provide an entire class, when we really just want them to give us a single appropriately-shaped function.

It's important that *passing* a function is not the same as *calling* a function. When we call a function, we execute the code inside of it and either receive a return value, cause some side-effects elsewhere in the code, or both. When we *pass* a function, nothing in particular happens *immediately*. Instead, by passing the function we are allowing some *other* code to call the function *in the future.* Seeing the name of a function in code does not always mean that the code in the function is being run!

Inside of code that passes a function, we will see some syntax that either refers to the name of an existing function in a special way, or else defines a new function to be passed inside of the call expression. The specific syntax needed (and the rules around it) depends on which programming language we are using.

17.9.2 Treating Functions as Data in Java

Java represents functions-as-data as instances of functional interfaces. A «functional interface» is a special kind of class that has only a single method - since Java was originally designed strictly for object-oriented programming, it has no way of representing a single function detached from a class. Instead, it defines a particular group of classes that *only* represent single functions. Each type of function signature has its own functional interface, which is an interface with a single function definition of that signature.

This might sound complicated, but in the context of WPILib we don't really need to worry much about using the functional interfaces themselves - the code that does that is internal to WPILib. Instead, all we need to know is how to pass a function that we've written to a method that takes a functional interface as a parameter. For a simple example, consider the signature of `Commands.runOnce` (which creates an `InstantCommand` that, when scheduled, runs the given function once and then terminates):

Nota: The `requirements` parameter is explained in the *Command-based documentation*, and will not be discussed here.

```
public static Command runOnce(Runnable action, Subsystem... requirements)
```

`runOnce` expects us to give it a `Runnable` parameter (named `action`). A `Runnable` is the Java term for a function that takes no parameters and returns no value. When we call `runOnce`, we need to give it a function with no parameters and no return value. There are two ways to do this: we can refer to some existing function using a «method reference», or we can define the function we want inline using a «lambda expression».

Method References

A method reference lets us pass an already-existing function as our `Runnable`:

```
// Create an InstantCommand that runs the `resetEncoders` method of the `drivetrain`␣
↪object
Command disableCommand = runOnce(drivetrain::resetEncoders, drivetrain);
```

The expression `drivetrain::resetEncoders` is a reference to the `resetEncoders` method of the `drivetrain` object. It is not a method *call* - this line of code does not *itself* reset the encoders of the drivetrain. Instead, it returns a `Command` that will do so *when it is scheduled*.

Remember that in order for this to work, `resetEncoders` must be a `Runnable` - that is, it must take no parameters and return no value. So, its signature must look like this:

```
// void because it returns no parameters, and has an empty parameter list
public void resetEncoders()
```

If the function signature does not match this, Java will not be able to interpret the method reference as a `Runnable` and the code will not compile. Note that all we need to do is make sure that the signature matches the signature of the single method in the `Runnable` functional interface - we don't need to *explicitly* name it as a `Runnable`.

Lambda Expressions in Java

If we do not already have a named function that does what we want, we can define a function «inline» - that means, right inside of the call to `runOnce`! We do this by writing our function with a special syntax that uses an «arrow» symbol to link the argument list to the function body:

```
// Create an InstantCommand that runs the drive forward at half speed
Command driveHalfSpeed = runOnce(() -> { drivetrain.arcadeDrive(0.5, 0.0); },␣
↪drivetrain);
```

Java calls `() -> { drivetrain.arcadeDrive(0.5, 0.0); }` a «lambda expression»; it may be less-confusingly called an «arrow function», «inline function», or «anonymous function» (because it has no name). While this may look a bit funky, it is just another way of writing a function - the parentheses before the arrow are the function's argument list, and the code contained in the brackets is the function body. The «lambda expression» here represents a function that calls `drivetrain.arcadeDrive` with a specific set of parameters - note again that this does not *call* the function, but merely defines it and passes it to the `Command` to be run later when the `Command` is scheduled.

As with method references, we do not need to *explicitly* name the lambda expression as a `Runnable` - Java can infer that our lambda expression is a `Runnable` so long as its signature matches that of the single method in the `Runnable` interface. Accordingly, our lambda takes no arguments and has no return statement - if it did not match the `Runnable` contract, our code would fail to compile.

Capturing State in Java Lambda Expressions

In the above example, our function body references an object that lives outside of the function itself (namely, the `drivetrain` object). This is called a «capture» of a variable from the surrounding code (which is sometimes called the «outer scope» or «enclosing scope»). Usually the captured variables are either local variables from the enclosing method body in which the lambda expression is defined, or else fields of an enclosing class definition in which that method is defined.

In Java capturing state is a fairly safe thing to do in general, with one major caveat: we can only capture state that is «effectively final». That means it is only legal to capture a variable from the enclosing scope if that variable is never reassigned after initialization. Note that this does not mean that the captured state cannot change: Remember that Java objects are references, so the object that the reference *points to* may change after capture - but the reference itself cannot be made to point to another object.

This means we can only capture primitive types (like `int`, `double`, and `boolean`) if they're constants. If we want to capture a state variable that can change, it *must be wrapped in a mutable object*.

Syntactic Sugar for Java Lambda Expressions

The full lambda expression syntax can be needlessly verbose in some cases. To help with this, Java lets us take some shortcuts (called «syntactic sugar») in cases where some of the notation is redundant.

Omitting Function Body Brackets for One-Line Lambdas

If the function body of our lambda expression is only one line, Java lets us omit the brackets around the function body. When omitting function brackets, we also omit trailing semicolons And the *return* keyword.

So, our `Runnable` lambda above could instead be written:

```
// Create an InstantCommand that runs the drive forward at half speed
Command driveHalfSpeed = runOnce(() -> drivetrain.arcadeDrive(0.5, 0.0), drivetrain);
```

Omitting Parentheses around Single Lambda Parameters

If the lambda expression is for a functional interface that takes only a single argument, we can omit the parenthesis around the parameter list:

```
// We can write this lambda with no parenthesis around its single argument
IntConsumer exampleLambda = (a -> System.out.println(a));
```

17.9.3 Treating Functions as Data in C++

C++ has a number of ways to treat functions as data. For the sake of this article, we'll only talk about the parts that are relevant to using WPILibC.

In WPILibC, function types are represented with the `std::function` class (https://en.cppreference.com/w/cpp/utility/functional/function). This standard library class is templated on the function's signature - that means we have to provide it a function type as a template parameter to specify the signature of the function (compare this to *Java* above, where we have a separate interface type for each kind of signature).

This sounds a lot more complicated than it is to use in practice. Let's look at the call signature of `cmd::RunOnce` (which creates an `InstantCommand` that, when scheduled, runs the given function once and then terminates):

Nota: The `requirements` parameter is explained in the *Command-based documentation*, and will not be discussed here.

```
CommandPtr RunOnce(
 std::function<void()> action,
 Requirements requirements);
```

`runOnce` expects us to give it a `std::function<void()>` parameter (named `action`). A `std::function<void()>` is the C++ type for a `std::function` that takes no parameters and returns no value (the template parameter, `void()`, is a function type with no parameters and no return value). When we call `runOnce`, we need to give it a function with no parameters and no return value. C++ lacks a clean way to refer to existing class methods in a way that can automatically be converted to a `std::function`, so the typical way to do this is to define a new function inline with a «lambda expression».

Lambda Expressions in C++

To pass a function to `runOnce`, we need to write a short inline function expression using a special syntax that resembles ordinary C++ function declarations, but varies in a few important ways:

```
// Create an InstantCommand that runs the drive forward at half speed
CommandPtr driveHalfSpeed = cmd::RunOnce([this] { drivetrain.ArcadeDrive(0.5, 0.0); },
↪ {drivetrain});
```

C++ calls `[captures] (params) { body; }` a «lambda expression». It has three parts: a *capture list* (square brackets), an optional *parameter list* (parentheses), and a *function body* (curly brackets). It may look a little strange, but the only real difference between a lambda expression and an ordinary function (apart from the lack of a function name) is the addition of the capture list.

Since `RunOnce` wants a function with no parameters and no return value, our lambda expression has no parameter list and no return statement. The «lambda expression» here represents a function that calls `drivetrain.ArcadeDrive` with a specific set of parameters - note again that the above code does not *call* the function, but merely defines it and passes it to the `Command` to be run later when the `Command` is scheduled.

Capturing State in C++ Lambda Expressions

In the above example, our function body references an object that lives outside of the function itself (namely, the `drivetrain` object). This is called a «capture» of a variable from the surrounding code (which is sometimes called the «outer scope» or «enclosing scope»). Usually the captured variables are either local variables from the enclosing method body in which the lambda expression is defined, or else fields of an enclosing class definition in which that method is defined.

C++ has somewhat more-powerful semantics than Java. One cost of this is that we generally need to give the C++ compiler some help to figure out *how exactly* we want it to capture state from the enclosing scope. This is the purpose of the *capture list*. For the purposes of using the WPILibC Command-based framework, it is usually sufficient to use a capture list of `[this]`, which gives access to members of the enclosing class by capturing the enclosing class's `this` pointer by value.

Method locals cannot be captured with the `this` pointer, and must be captured explicitly either by reference or by value by including them in the capture list (or by implicitly by instead specifying a default capture semantics). It is typically safer to capture locals by-value, since a lambda can outlive the lifespan of an object it captures by reference. For more details, consult the C++ standard library documentation on capture semantics.

17.10 Obtener Color de Alianza

The `DriverStation` class (Java, C++, Python) has many useful features for getting data from the Driver Station computer. One of the most important features is `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 Obteniendo tu Alliance Color y Realizando una Acción

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.

Nota: *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.

Consideraciones de diseño

Si anticipas que tu aplicación creará una gran cantidad de objetos de corta duración, es importante considerar estrategias de diseño para mitigar los posibles problemas de memoria y rendimiento. Aquí hay algunas estrategias a considerar:

- 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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
            // 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*.

Advertencia: 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.

> **Prudencia:** 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() {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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.

Prudencia: 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 Recursos de Apoyo

Además de esta documentación, existe una gran variedad de recursos disponibles para apoyar a los equipos FRC® a entender el Sistema de Control y software.

18.1 Otros Recursos

Además de este sitio, existen algunos otros sitios a los que los equipos pueden recurrir para más documentación:

- Documentos para la comunidad NI FRC
- Página con recursos técnicos de FIRST Inspires
- CTRE Software & Resources Page
- REV Robotics Documentation

18.2 Foros

¿Tiene dudas? ¿Tiene una pregunta que no ha podido responder con la documentación? En los siguientes foro puede encontrar Apoyo Oficial:

- NI FRC Support Forum (roboRIO, LabVIEW and Driver Station software questions, as well as roboRIO repairs)
- Foro del control de sistema de FIRST Inspires (preguntas sobre cableado, hardware y Driver Station)
- Foro de programación de FIRST Inspires (preguntas de programación de C++, Java, o LabVIEW)

18.3 Apoyo de 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 Soporte de 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 Other Vendors

Support for vendors outside of the KOP can be found below.

- Copperforge
- Kauai Labs (NavX)
- Limelight
- PhotonVision (Discord)
- Playing with Fusion

18.6 Unofficial Support

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 Reporte de errores

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 Glosario de FRC

acelerómetro
Un sensor común es usado para medir la aceleración en uno o más ejes.

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.

deprecated
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
El mecanismo que impulsa la implementación del código del robot en la roboRIO.

giroscopio
Un dispositivo que mide la velocidad de rotación. Puede sumar las medidas de rotación para determinar *rumbo* del robot. ("gyro", para abreviar)

rumbo
La dirección en la que el robot apunta, usualmente expresado como un ángulo en grados.

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).

chasis 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.

simulación
: Una forma para que los equipos prueben su código sin tener un robot real disponible.

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.

teleop
La segunda fase de cada partido se llama Periodo Teleoperado (teleop) y consiste en conductores controlando sus robots.

topic
In *NetworkTables*, a named data channel.

torque
A force applied at a distance from some axis of rotation

trayectoria
Una trayectoria es una curva suave, con velocidades y aceleraciones en cada punto a lo largo de la curva, que conecta dos puntos finales en el campo.

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.

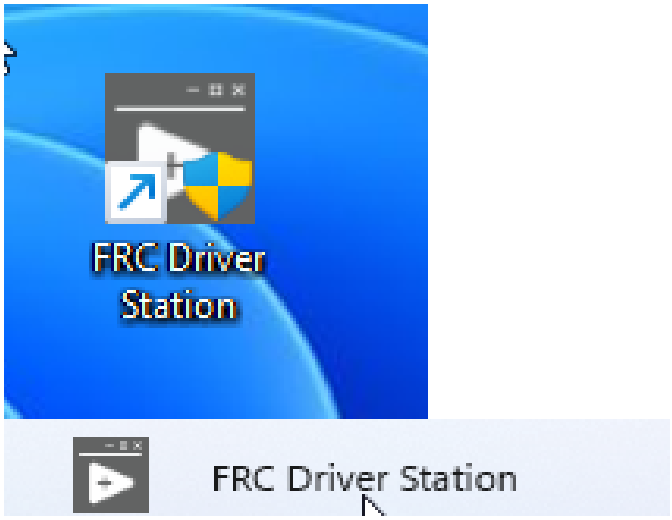
20 Driver Station

20.1 FRC Driver Station Powered by NI LabVIEW

Este artículo describe el uso y las características de FRC® Driver Station desarrollado por NI LabVIEW.

Para información sobre la instalación del software de la Driver Station vea *este documento*.

20.1.1 Empezar la FRC Driver Station



The FRC Driver Station can be launched by double-clicking the icon on the Desktop or by selecting Start->All Apps->FRC Driver Station.

Nota: By default the FRC Driver Station launches the *LabVIEW Dashboard*. It can also be configured on *Setup Tab* to launch the other Dashboards: *SmartDashboard* and *Shuffleboard*. WPILib must be *installed* to use SmartDashboard and Shuffleboard.

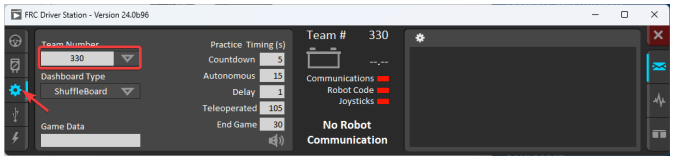
20.1.2 Atajos de teclado de la Driver Station

- `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.

Nota: La barra espaciadora detendrá al robot independientemente de si la ventana de la Driver Station está enfocado o no.

Advertencia: 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.

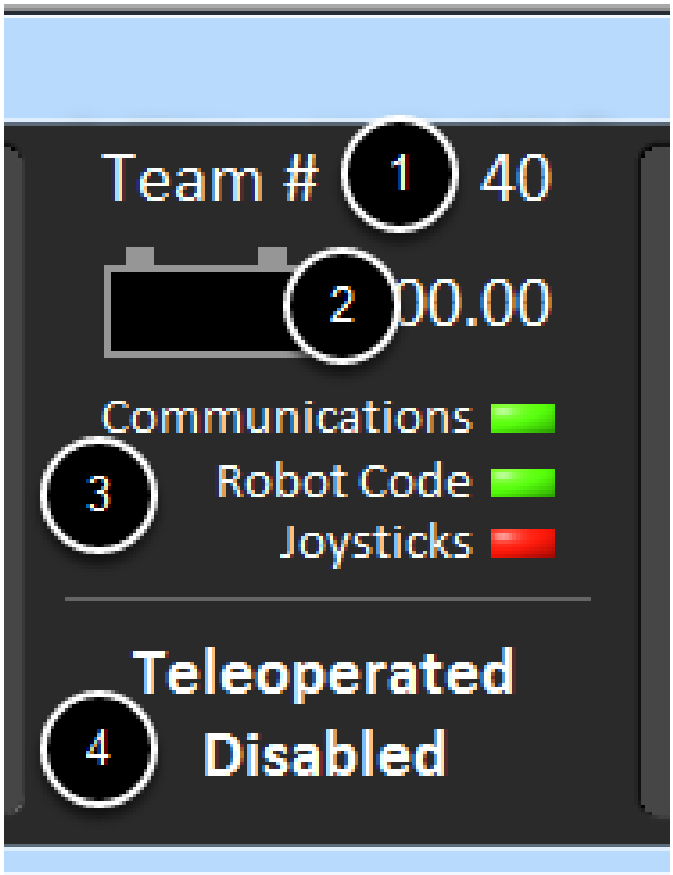
20.1.3 Configurar la FRC Driver Station



La DS debe configurarse con el número de su equipo para conectarse a su robot. Para hacer esto haga clic en la pestaña de Setup y luego ingrese su número de equipo en el cuadro de número de equipo. Presione return o haga clic fuera del cuadro para que la configuración tenga efecto.

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 Status Pane

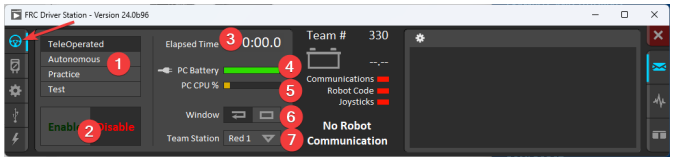


El Status Pane de la Driver Station está localizado en el centro del monitor y siempre está visible independientemente de la pestaña que esté seleccionada. Muestra una selección de información crítica sobre el estado de la DS y el robot:

1. Team # – El número del equipo para el cual el DS está configurado actualmente. Esto debería de coincidir con su número de equipo de FRC. Para cambiar el número de equipo vea la pestaña de Setup.
2. Battery Voltage – Si la DS está conectada y comunicándose con la roboRIO, esta muestra el voltaje de la batería actual como un número con una pequeña tabla del voltaje a lo largo del tiempo en el ícono de la batería. El fondo del indicador numérico se volverá rojo cuando se desencadene un apagón de la roboRIO. Vea *Caída de voltaje en roboRIO y Comprender el Consumo de Corriente* para más información.
3. Major Status Indicator – Estos tres indicadores muestran los principales elementos de estado para la DS. “Communications” indica cuando la DS está actualmente comunicándose con FRC Communications Task en la roboRIO (está dividido por la mitad para la comunicación de TCP y UDP). El indicador del código del robot muestra cuando el código del robot se está ejecutando actualmente (determinado cuando la Driver Station Task del código del robot está o no actualizando el voltaje de la batería), el indicador de los “joysticks” muestra si por lo menos alguno de los joysticks está conectado y reconocido por la 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», «Emer-

gency Stopped», and «Teleoperated Enabled». When the roboRIO brownout is triggered this will display «Voltage Brownout».

20.1.5 Operation Tab

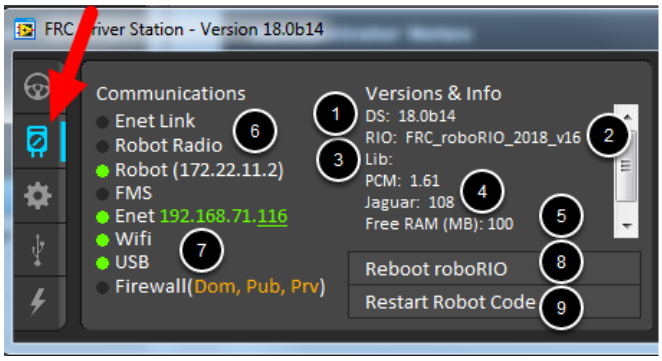


La Operation Tab es usada para controlar el modo del robot y proporcionar indicadores clave de estado adicionales mientras el robot está funcionando.

1. Robot Mode - This section controls the Robot Mode.
   - Teleoperated Mode causes the robot to run the code in the Teleoperated portion of the match.
   - Autonomous Mode causes the robot to run the code in the Autonomous portion of the match.
   - Practice Mode causes the robot to cycle through the same transitions as an FRC match after the Enable button is pressed (timing for practice mode can be found on the setup tab).
   - *Test Mode* is an additional mode where test code that doesn't run in a regular match can be tested.
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 – Cuando no esté conectada a FMS, configure la team station para transmitir al robot.

Nota: 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.

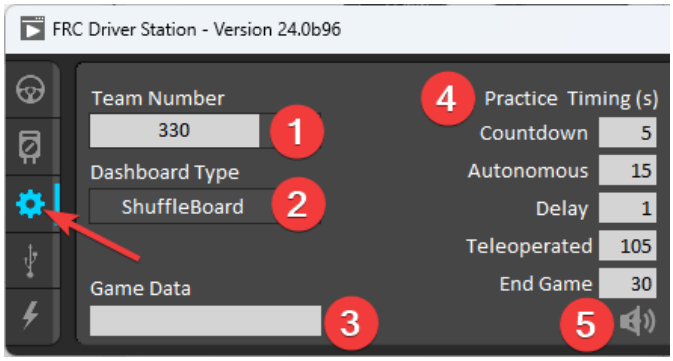
20.1.6 Diagnostics Tab



La Diagnostics Tab contiene indicadores de estado adicionales que los equipos pueden usar para diagnosticar los problemas con su 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 mitad superior de estos indicadores muestra el estado de conexión a varios componentes.
   - “Enet Link” indica que la computadora tiene algo conectado en el puerto de Ethernet.
   - “Robot Radio” indica el estado de ping al puente inalámbrico del robot en 10.XX.YY.1.
   - “Robot” indica el estado de Ping a la roboRIO utilizando mDNS (con un respaldo de dirección estática 10.TE.AM.2).
   - “FMS” indica si la DS está recibiendo paquetes de FMS (esto NO es un indicador 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” indica la dirección IP del adaptador de Ethernet detectado.
   - “WiFi” indica si un adaptador inalámbrico ha sido detectado como habilitado.
   - “USB” indica si una conexión USB de la roboRIO ha sido detectada.
   - «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).

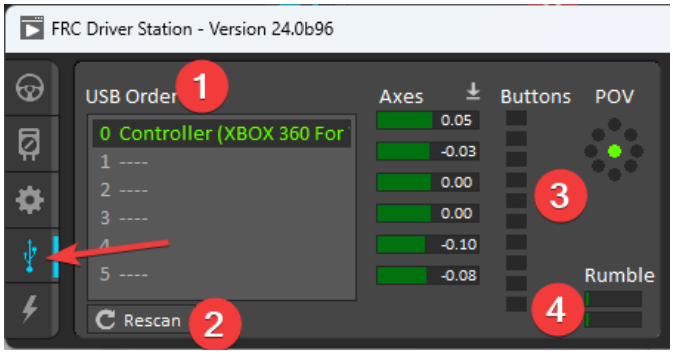
20.1.7 Setup Tab



La Setup Tab contiene un número de botones los cuales el equipo puede utilizar para controlar la operación de la 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 – Estas cajas controlan el tiempo de cada porción de la secuencia del modo de práctica. Cuando el robot está habilitado en modo de práctica la DS automáticamente avanza a través de los modos de indicados desde arriba hacia abajo.
5. Audio Control – Este botón controla si los tonos de audio suenan cuando se utiliza el modo de práctica.

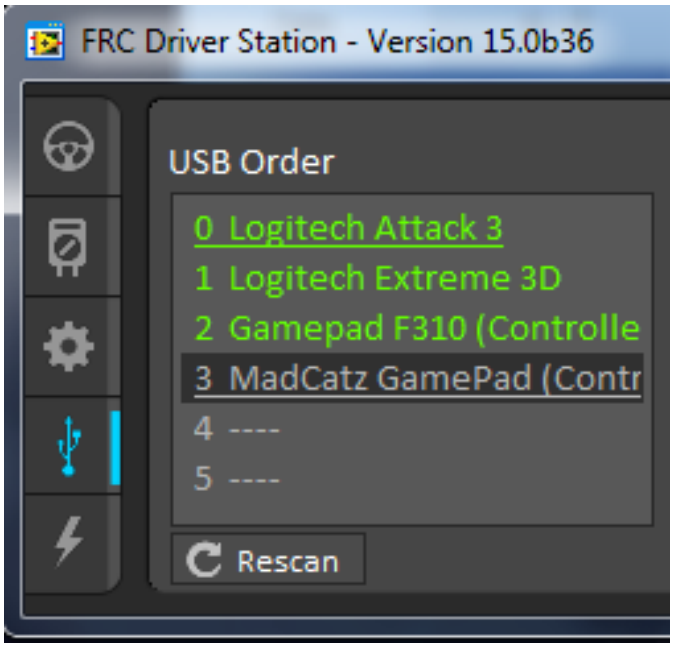
20.1.8 USB Devices Tab



La USB Devices tab incluye la información sobre los dispositivos USB conectados a la DS

1. USB Setup List – Esta contiene una lista de todos los dispositivos USB compatibles conectados a la DS. Si presiona un botón en un dispositivo, resaltará el nombre del dispositivo en verde y pondrá 2 *s antes del nombre del dispositivo.
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. Device Indicators – Estos indicadores muestran el estado actual de Axes, botones y POV del joystick.
4. Rumble – Para dispositivos XInput (como controladores de X-Box) el control Rumble aparecerá. Este puede ser usado para probar la funcionalidad rumble del dispositivo. La barra superior es “Right Rumble” y la barra inferior es “Left Rumble”. Haciendo clic y manteniendo en cualquier lugar a lo largo de la barra, activará la proporcionalidad de Rumble (izquierda es no rumble = 0 , derecha es rumble lleno = 1). Este es un control único y no indicará el valor de Rumble establecido en el código del robot.

Dispositivos de reordenamiento y bloqueo



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.

Nota: 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

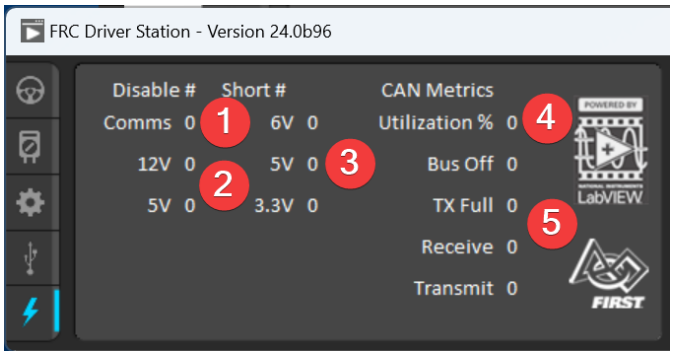
re-arrange the ports (take one device and plug it into a new port instead of swapping) the behavior is not determinate (the devices may swap slots). If you unplug one or more of the set of devices, the positions of the others may move; they should return to the proper locked slots when all devices are reconnected.

Ejemplo: la imagen de arriba muestra 4 dispositivos:

- Un joystick “Logitech Attack 3” bloqueado. Este dispositivo se quedará en esta posición a menos de que sea arrastrada a otro lugar o desbloqueado.
- Un joystick “Logitech Extreme 3D” desbloqueado.
- Un “Gamepad F310 (Controller)” desbloqueado, el cual es un gamepad Logitech F310.
- Un “MadCatz GamePad (Controller)” que está bloqueado, pero desconectado, el cual es un MadCatz Xbox 360 Controller.

En este ejemplo, si desconecta el joystick Logitech Extreme 3D resultará que el F310 Gamepad se moverá a la ranura 1. Conectar el MadCatz Gamepad (aunque los dispositivos en las ranuras 1 y 2 sean removidos y esas ranuras estén vacías) resultará en que este ocupe la ranura 3.

20.1.9 CAN/Power Tab

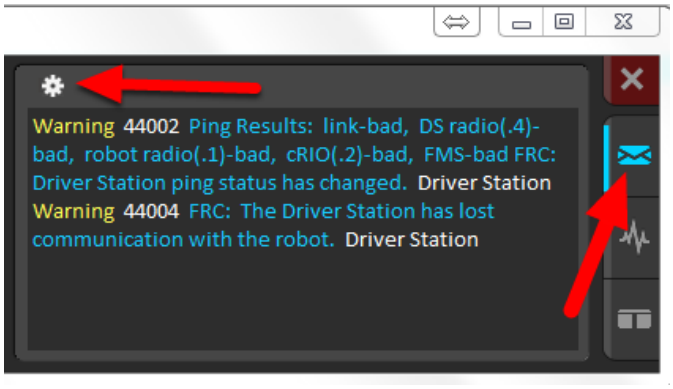


La última pestaña del lado izquierdo de la DS es la CAN/Robot Power Tab. Esta pestaña contiene información sobre el estado de energía de la roboRIO y el estado del CAN bus:

1. Comms Faults – Indica el número de falla de comunicaciones desde que la DS ha estado conectada.
2. 12V Faults – Indica el número de fallas de alimentación de entrada (Brownouts) que han ocurrido desde que la DS ha estado conectada.
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. CAN Bus Utilization – Indica el porcentaje de utilización del CAN bus.
5. CAN faults – Indica los recuentos de cada uno de los 4 tipos de fallas de CAN desde que la DS ha estado conectada.

Si una falla es detectada, el indicador de esta pestaña (mostrado en azul en la imagen de arriba) se volverá roja.

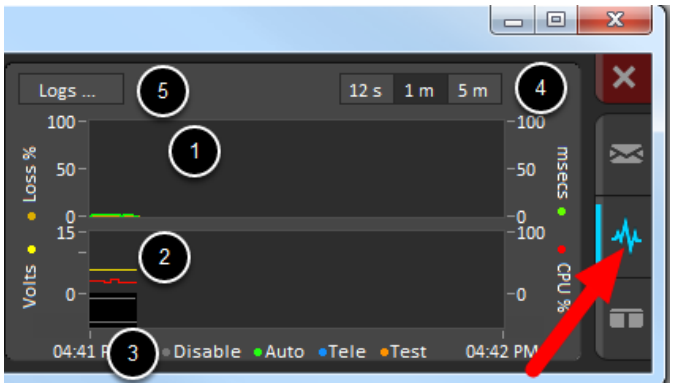
20.1.10 Messages Tab



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 Charts Tab



La Charts tab traza y muestra indicadores avanzados del estado del robot para ayudar a los equipos a diagnosticar los problemas del robot:

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. La gráfica inferior traza el voltaje de la batería en amarillo (contra el eje a la izquierda), roboRIO CPU en rojo (contra el eje de la derecha), el DS Requested Mode como una línea continua en el fondo de las gráficas y el robot mode como una línea discontinua arriba de esta.
3. Esta clave muestra los colores usados para la DS Requested y Robot Reported modes en la tabla inferior.
4. Chart scale - These controls change the time scale of the DS Charts.
5. This button launches the *DS Log File Viewer*.

El DS Requested mode es el modo en el que la Driver Station le ordena al robot que esté dentro. El Robot Reported mode es que código es el que se está ejecutando actualmente, basado en los métodos de informes contenidos en los marcos de codificación para cada lenguaje.

20.1.12 Both Tab

The last tab on the right side is the Both tab which displays Messages and Charts side by side.

20.2 Driver Station Best Practices

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.

¿Quiere asegurarse de que la driver station no sea un obstáculo para su equipo en el campo de FIRST Robotics Competition (FRC)? Construir y configurar una laptop sólida para la driver station es un proyecto fácil gracias a el tiempo entre el stop build day y el día de su competencia. Siga leyendo para encontrar lecciones aprendidas por muchos equipos en más de miles de partidos.

20.2.1 Antes De Partir A La Competencia

1. Dedique una laptop para ser usada solamente como una driver station. Varios equipos lo hacen. Una máquina dedicada permite que maneje la configuración para una meta - estar listos para competir en la cancha. Dedicada significa ningún otro software excepto el FRC - provided Driver Station software y Dashboard asociado instalado o en ejecución.
2. Use una laptop bussines-class para su driver station. ¿Por qué? Son mucho más duraderas que las de $300 USD en precio especial en Black Friday en Best Buy. Sobrevivirán los golpes de la competencia. Las laptops de bussines-class, tienen controladores de dispositivos de una mejor calidad y los controladores se mantienen por un periodo más largo que las laptops de consumo. Esto hace que su inversión dure más. Lenovo ThinkPad T series y Dell Latitude son dos marcas bussines-class populares que verá en las competencias. Hay miles de ofertas diariamente en eBay. Las laptops proporcionadas por los rookie kits recientes, es una buena máquina de nivel de entrada. Los equipos con frecuencia se gradúan en pantallas más grandes a medida que hacen más con la visión y dashboards.
3. Considere las laptops usadas en lugar de las nuevas. El FRC ® El software de la Driver Station y la dashboard utiliza muy pocos recursos del sistema, por lo que no necesita comprar una computadora portátil nueva; en su lugar, compre una usada barata de 4-5 años. Incluso puede obtener una donada por una tienda de computadoras usadas en su área.

Nota: 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. Características recomendadas para una laptop
    a. RAM – 8GB of RAM or greater

b. Un monitor de 13” o mayor, con un mínimo de resolución de 1440x1050.

c. Puertos

i. Un puerto incorporado de Ethernet es altamente preferido. Asegúrese de que sea un puerto de tamaño completo. Los puertos de Ethernet con bisagras no soportan el uso repetido.

ii. Use un protector de puerto de Ethernet para hacer su conexión Ethernet. Esto extiende la vida del puerto Ethernet de su laptop. Esto es particularmente importante si tiene una laptop de grado de consumidor con un puerto de Ethernet con bisagra.

iii. Si el puerto de Ethernet de su laptop es dudoso, ya sea que reemplace su laptop (recomendado) o compre un Ethernet USB dongle de alguna marca reconocida. Varios equipos encuentran que el Ethernet USB es menos confiable que el Ethernet incorporado, principalmente, debido al hardware barato y malos controladores. Los dongles que se les dan a los rookies en su KOP tienen una reputación de trabajar bien.

iv. 2 puertos USB mínimo

d. Un teclado. Es difícil solucionar rápidamente los problemas de las computadoras táctiles en la cancha.

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. Instale todas las actualizaciones de Windows una semana antes de la competencia. Esto le permite asegurarse con tiempo que las actualizaciones no interferirán en las funciones de la driver station. Para hacerlo, abra la página de ajustes de las actualizaciones de Windows y compruebe que esté actualizado. Instale las actualizaciones pendientes si es que tiene. Reinicie y revise de nuevo para asegurarse de que esté actualizado.

6. Cambie “Active hours” de las actualizaciones de Windows para qprevenir que se instalen las actualizaciones durante las horas de la competencia. Navegue a Start -> Settings -> Update & Security -> Windows Update, después seleccione Change active hours. Si está viajando para una competencia tome en Cuenta las diferencias de la zona horaria. Esto le ayuda a asegurarse a que su driver station no se reinicie o falle debido a la instalación de una actualización en la cancha.

7. Remove any 3rd party antivirus or antimalware software. Instead, use Windows Defender on Windows 10 or 11. Since you’re only connecting to the internet for Windows and FRC software updating, the risk is low. Only install software on your driver station that’s needed for driving. Your goal here is to eliminate variables that might interfere with proper operation. Remove any unneeded preinstalled software («bloatware») that came with the machine. Don’t use the laptop as your Steam machine for gaming back at the hotel the night before the event. Many teams go as far as having a separate programming laptop.

8. Avoid managed Windows 10 or 11 installations from the school’s IT department. These deployments are built for the school environment and often come with unwanted software that interferes with your robot’s operation.

9. Batería/energía de la laptop

a. Apague Ponga la computadora en suspender en su plan de energía para la batería y la alimentación de la operación.

b. Apague el USB Selective Suspend:

   i. Dé clic derecho en el icono de batería/carga en la bandeja, después seleccione Power Options.

   ii. Edite el plan de ajustes de su plan de energía.

   iii. Dé clic en el enlace de Cambiar la configuración avanzada de energía.

   iv. Desplácese hacia abajo en las configuraciones avanzadas y deshabilite la suspensión selectiva de configuración de USB para batería y enchufe.

c. Asegúrese de que la batería de la laptop pueda mantener una carga de al menos una hora después de hacer los cambios de arriba. Esto permite bastante tiempo para que el robot y el drive team puedan pasar por el queue y llegar a la estación de la alianza sin red eléctrica.

10. Traiga una USB y un cable de Ethernet confiable para su uso conectando a la roboRIO.
11. Agregue retención / alivio de tensión para prevenir que sus joystick / controladores de gamepad de caer al suelo y/o tirar de los puertos USB. Esta ayuda previene problemas con las conexiones intermitentes a los controladores.
12. La cuenta de usuario de Windows que se utiliza para manejar, debe de ser un miembro del grupo administrador.

20.2.2 En La Competencia

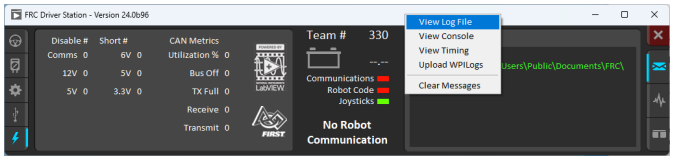
1. Turn off Windows firewall using *these instructions*.
2. Apague el adaptador del Wi-Fi, ya sea utilizando el conmutador de Wi-Fi de hardware dedicado o deshabilitándolo en el panel de control de Adapter Settings.
3. Cargue la driver station cuando esté en el pit.
4. Remueva todas las contraseñas para iniciar sesión o asegúrese de que todo el drive team conozca la contraseña. Se sorprendería al saber que tan seguido los drivers llegan a la cancha sin saber la contraseña de la laptop.
5. Asegúrese que su código de LabView se implemente permanentemente y se establece en “run as startup”, usando las instrucciones del Tutorial de LabView. Si debe de implementar el código cada vez que encienda de nuevo el robot, lo está haciendo mal.
6. Limítese a buscar en la red en sitios web relacionados con FRC. Esto minimiza la oportunidad de que se obtenga malware durante la competencia.
7. No planee usar acceso a internet para hacer actualizaciones del software. Es probable que no haya ninguno en el lugar y el Wi-Fi del hotel varíe ampliamente en la calidad. Si necesita hacer actualizaciones contacte al Control System Advisor en el pit.

20.2.3 Antes De Cada Partido

1. Asegúrese que la laptop esté encendida y conectada antes del término del partido anterior al suyo.
2. Cierre los programas que no sean necesarios durante el partido – p. ej., Visual Studio Code o Lab View – cuando esté compitiendo.
3. Lleve el cargador de su laptop a la cancha. Se proporciona energía en cada player station.
4. Sujete su computadora con cinta hook-and-loop al estante de la player station. Nunca sabrá cuando su equipo de alianza tenga algún problema en la programación en autónomo y golpee la pared.
5. Asegúrese que sus joysticks y controladores estén asignados a los puertos USB correctos.
    a. En la USB tab en el software de la FRC Driver Station, arrastre y suelte para asignar joysticks como sea necesario.
    b. Use el botón de reescaneo (F1) si los joysticks / controladores no aparecen el verde.
    c. Use el botón de reescaneo (F1) durante la competencia si los joysticks o controladores son desconectados y conectados de vuelta o si se vuelve gris durante la competencia.
6. Ensure your *Dashboard is connected to the robot* after your driver station connects to the robot.

20.3 Driver Station Log File Viewer

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.



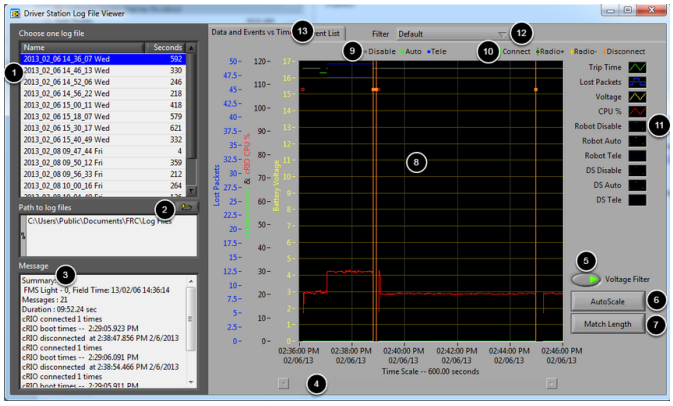
Nota: Several alternative tools exist that provide similar functionality to the FRC Driver Station Log Viewer. *AdvantageScope* is an option included in WPILib, and DSLOG Reader is a third-party option. Note that WPILib offers no support for third-party projects.

20.3.1 Archivos De Eventos

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 Log Viewer UI



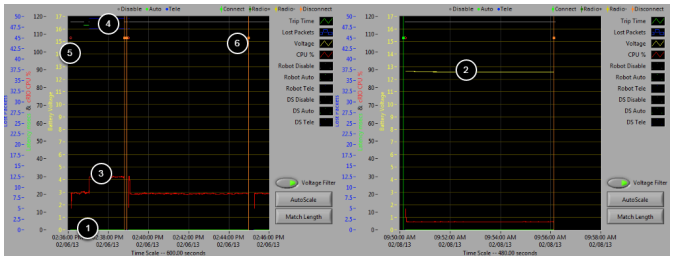
El Log Viewer contiene varios controles y visualizadores para ayudar en el análisis de los archivos de la Driver Station:

1. File Selection Box – Esta ventana muestra todos los archivos de datos disponibles en la carpeta seleccionada. De clic en un archivo de la lista para seleccionarlo.
2. Path to Log Files – Este recuadro muestra la carpeta en la que el viewer está buscando archivos. Esto predetermina a la carpeta donde la driver station guarda los archivos. De clic en el icono de la carpeta para buscar en una ubicación diferente.
3. Message Box – Este recuadro muestra un resumen de todos los mensajes de los archivos de eventos. Al flotar sobre un evento en la gráfica este recuadro cambia, mostrando información para ese evento.
4. Scroll Bar – Cuando la gráfica esta ampliada, esta barra de desplazamiento permite desplazarse horizontalmente por la gráfica.
5. Voltage Filter – Este control prende y apaga el filtro de voltaje (predeterminados también). El filtro de voltaje filtra datos como CPU%, modo del robot y tiempo de viaje cuando no se recibe voltaje de batería (indicando que la DS no está en comunicación con la roboRIO).
6. AutoScale – Este botón cambia de tamaño la gráfica para mostrar todos los datos en el archivo.
7. Match Length – Este botón cambia el tamaño de la gráfica a aproximadamente la duración de un partido de FRC (2 minutos y 30 segundos). No localiza automáticamente el inicio del partido, usted tendrá que desplazarse usando el la barra desplazatoria para localizar el inicio del modo autónomo.
8. Graph – Esto muestra gráficos con datos de la DS Log File (voltaje, tiempo de viaje, roboRIO CPU%, paquetes perdidos y modo del robot) así como datos de eventos sobrepuestos

(mostrados como puntos en la gráfica con eventos seleccionados mostrados como líneas verticales a través de toda la gráfica). Al pasar el cursor sobre los indicadores de eventos en la gráfica muestra información sobre el evento en la ventana de mensajes en la parte inferior izquierda de la pantalla.

9. Robot Mode Key – Clave para que el modo del robot se muestre en la parte superior de la pantalla.
10. Major event key – Clave para los eventos más grandes, se muestra como líneas verticales en la gráfica
11. Graph key – Clave para los datos gráficos.
12. Filter Control – Despliegue para seleccionar el modo del filtro (modo de filtro se explican debajo).
13. Tab Control – Control para cambiar entre gráficos (Data and Events vs. Time) y muestras de listas de eventos.

20.3.3 Uso de la pantalla gráfica

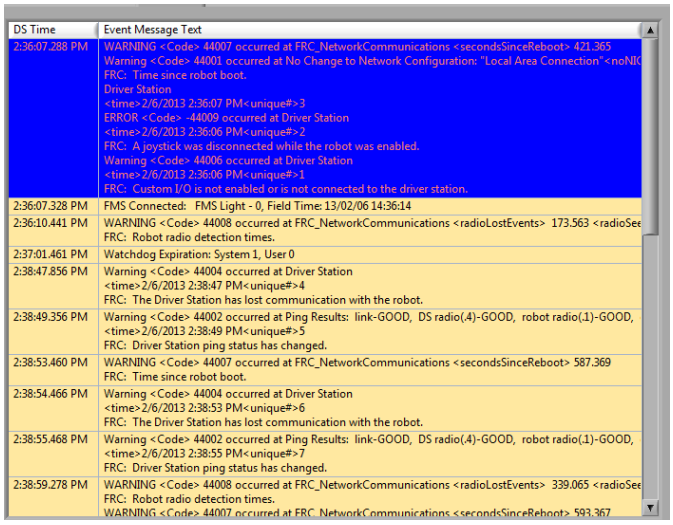


La visualización de la gráfica contiene la siguiente información:

1. Gráficas del tiempo de viaje en ms (línea verde) y paquetes perdidos por segundo (mostrados como líneas azules verticales). En estas imágenes de ejemplo el tiempo de viaje es una linea plana verde al fondo de la gráfica y no hay paquetes perdidos.
2. Gráfica de voltaje de la batería mostrada como una línea amarilla.
3. Gráfica del roboRIO CPU % como una línea roja
4. Gráfica del modo del robot y de la DS. El conjunto superior de la visualización muestra el modo ordenado por la Driver Station. El conjunto inferior muestra el modo reportado por el código del robot. En este ejemplo el robot no está reportando su modo durante los modos disabled y autónomo, pero lo reporta durante Teleop.
5. Marcadores de evento se mostrarán en la gráfica indicando el cuándo sucedió el evento. Los errores se mostraran en rojo; advertencias en amarillo. Pasar sobre un marcador de evento mostrará información sobre el evento en la caja de mensajes en la parte inferior izquierda de la pantalla.
6. Eventos grandes se muestran como líneas verticales a través de la gráfica.

Para amplificar una parte de la gráfica, dé clic y arrastre alrededor del área deseada. Usted solo puede amplificar el eje del tiempo, no puede amplificar verticalmente.

20.3.4 Event List



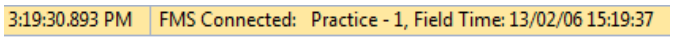
La Event List tab muestra una lista de eventos (advertencias y errores) grabados por la Driver Station. Estos eventos y detalles son determinados por el filtro activo en ese momento (la imagen muestra el filtro “All Events, All Info” activado).

20.3.5 Filtros

Estos tres filtros están disponibles actualmente en el Log Viewer:

1. Default: Este filtro remueve muchos de los errores y advertencias producidas por la Driver Station. Este filtro es útil para identificar los errores hechos por el mismo código del Robot.
2. All Events and Time: Este filtro muestra todos los eventos y el tiempo en el que ocurrieron.
3. All Events, All Info: Este filtro muestra todos los eventos y toda la información guardada. En este momento la principal diferencia entre este filtro y el “All Events and Time” es que esta opción muestra un designante único para la primera aparición de un mensaje en específico.

20.3.6 Identificar Archivos De Datos De Partidos.



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.

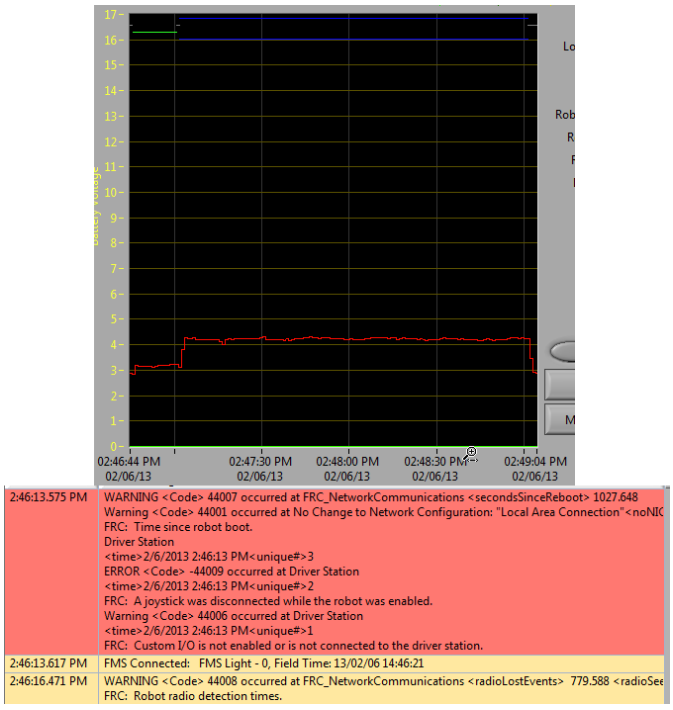
20.3.7 Identificar Fallas De Conexión Comunes Con El Low Viewer

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.

Nota: Note que todos los log files en esta sección han sido redimensionados para tener la misma longitud usando el botón Match Length y después desplazarse al principio del modo autónomo. También muchos de los archivos no contienen información sobre voltaje de la batería, la plataforma usada para la captura de archivos no se conectó de manera correcta para reportar el voltaje de la batería.

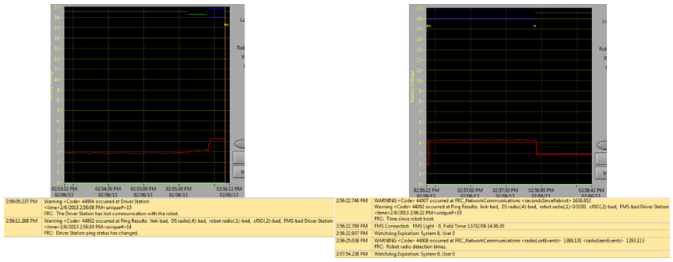
Truco: Algunos mensajes de error que se encuentran en el Log Viewer se muestran debajo y más son detallados en la sección *Errores/Advertencias de la Driver Station*

Log «Normal»



Este es un ejemplo de un archivo de datos log normal de un match. Los errores y advertencias contenidas en el primer recuadro son de cuando la DS inicio por primera vez y puede ser ignorado. Esto es confirmado por observar que estos eventos suceden antes del evento. El último evento también puede ser ignorado, también surge de la primera conexión del robot a la DS (ocurre 3 segundos después de conectarse a la FMS) y sucede casi 30 segundos antes que inicie el match.

Desconectado de la FMS

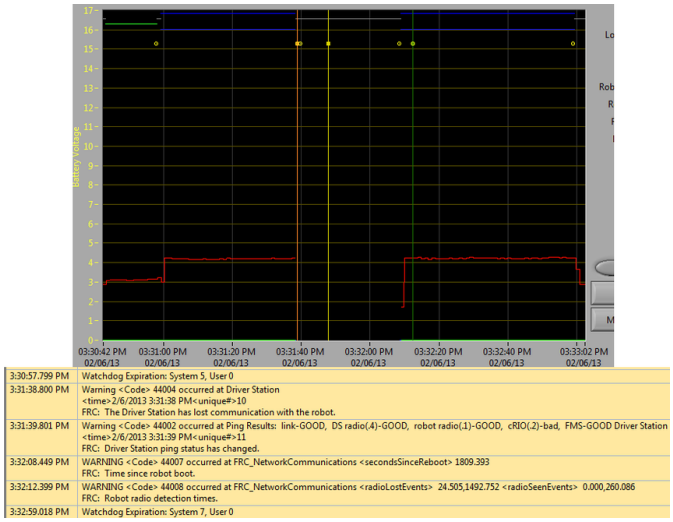


Cuando la DS se desconecta de la FMS, por lo tanto del robot, durante un match puede dividir el archivo en piezas. Los indicadores clave para esta falla son el último evento del primer archivo, indicando que la conexión a la FMS es ahora “mala” y que el 2do evento del segundo archivo, el cual es un mensaje nuevo de la FMS seguido por la transición inmediata de la DS a Teleop Enabled. La causa más común de este tipo de falla es un cable Ethernet mal conectado o un puerto Ethernet dañado de la computadora DS.

Reiniciar la roboRIO

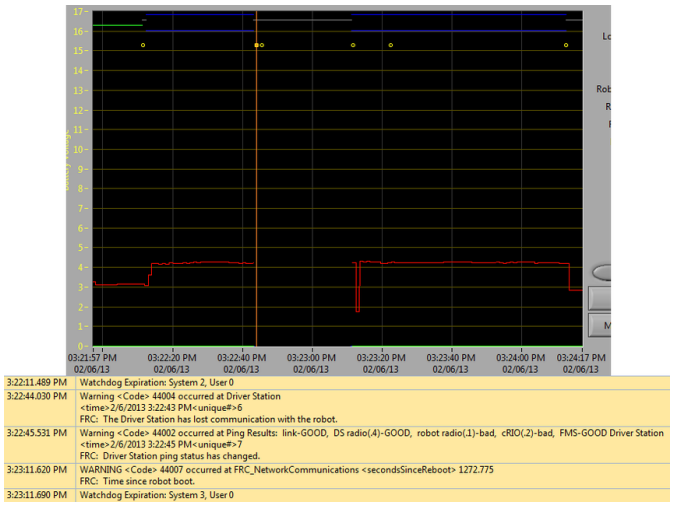
El mensaje “Time since robot boot” es el principal indicador de una falla de conexión causada por el reinicio de la roboRIO. En este archivo la DS pierde conexión con la roboRIO con 3:01:36 como se indica en el primer evento. El segundo evento indica que el sonido iniciado después que la conexión falló fue exitosa en todos los dipositivos además de la roboRIO. A las 3:01:47 la roboRIO inicia respondiendo a sonidos de Nuevo, un sonido adicional falla a las 3:01:52. A las 3:02:02 la Driver Station se conecta a la roboRIO y la roboRIO reporta que ha estado funcionando por 3.682 segundos. Este es un indicador muy claro que la roboRIO se ha reiniciado. El código continua cargándose y a las 3:02:24 el código reporta un error al comunicarse con la cámara. Una advertencia también se reporta indicando que ningún código del robot está ejecutándose antes que el código termine de iniciar.

Problema de cable Ethernet en el robot



Un problema con un cable de Ethernet en el robot es indicado principalmente por el ping de la roboRIO cambia a mal y los eventos Radio Lost y Radio Seen cuando la roboRIO se reconecta. El mensaje “Time since robot boot” cuando la roboRIO se reconecte también indicara que la roboRIO no se ha reiniciado. En este ejemplo, el cable de Ethernet del robot fue desconectado a las 3:31:38. El estatus del ping indica que el D-Link radio sigue conectado. Cuando el robot se reconecta a las 3:32:08 el“Tim since robot boot” está en 1809 segundos indicando que la roboRIO claramente no se reinició. A las 3:32:12 el robot indica que perdió la radio hace 24.505 segundos y regresó hace 0.000 segundos. Estos puntos se muestran como líneas verticales en la gráfica, amarillo para la radio y verde para la vista de radio. Note que los tiempos están ligeramente desacomodados de los eventos actuales como se muestran vía desconexión y conexión, pero ayuda a proporcionar información adicional sobre lo que esta pasando.

Reinicio del radio



El reinicio del radio del robot se caracteriza por la pérdida de conexión a la radio por 40-45 segundos. En este ejemplo, la radio pierde poder brevemente a las 3:22:44, causando

que empiece a reiniciarse .El evento a las 3:22:45 indica que el ping de la radio fallo. A las 3:23:11 la DS vuelve a ganar comunicación con la roboRIO y la roboRIO indica que ha estado funcionando por 1272.775 segundos, descartando un reinicio de la roboRIO. Note que el network switch en la radio regresa arriba rápidamente entonces una pérdida de poder puede que no resulte en el par de eventos "radio lost" /"radio seen". Una molestia más larga puede que resulte en que los eventos de la radio sean archivados por la DS. En ese caso, el factor distinguido que apunta a un reinicio de la radio es el estatus del ping de la radio de la DS. Si la radio se reinicia, esta será inalcanzable. Si el problema es el cableado o conexión del robot, el ping de la radio debería permanecer en "GOOD".

20.4 Errores/Advertencias de la 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 Joystick Desconectado

```
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.
```

Este error es desencadenado cuando un joystick es desconectado. Al contrario que el mensaje de texto este error se mostrara aún si el robot no está funcionando, o conectado a la DS. Usted verá una sola instancia de este mensaje suceder cada vez que la Driver Station inicia, incluso si los Joysticks están conectados correctamente y funcionando.

Nota: Joystick Unplugged warnings can be silenced by calling `DriverStation.silenceJoystickConnectionWarning(true)` (Java, C++)

20.4.2 Comunicación Perdida

```
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.
```

Este mensaje de advertencia es mostrado cuando la Driver Station pierde comunicación con el robot (Indicador de comunicación cambia de verde a rojo). Una sola instancia de este mensaje es mostrada cuando la DS inicia, antes que se establezca comunicación.

20.4.3 Ping Status

```
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 Tiempo Desde El Reinicio Del Robot

```
WARNING<Code>44007 occurred at FRC_NetworkCommunications
**<secondsSinceReboot> 3.585**
FRC: Time since robot boot.
```

Este mensaje se muestra cada vez que la DS empieza a comunicarse con la roboRIO. El mensaje indica el tiempo de actividad, en segundos, de la roboRIO y puede usarse para determinar si la perdida de comunicación fue por el reinicio de la roboRIO.

20.4.5 Radio Detection Times

```
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.
```

Este mensaje puede mostrarse cuando la DS inicie a comunicarse con la roboRIO e indica el tiempo, en segundos, desde la última vez que el radio paso por los eventos “lost” y “seen”. En el primer ejemplo, la imagen sobre el mensaje indica que la conexión entre radio y roboRIO se perdió 19 segundos antes que el mensaje se mostrará y el radio se viera correctamente cuando el mensaje se mostró. Si múltiples eventos de radioLost o radioSeen han ocurrido desde que la roboRIO se reinició, se incluirán hasta 2 eventos de cada tipo, separados por comas.

20.4.6 No Robot Code

```
Warning<Code>44003 occurred at Driver Station
<time>2/8/2013 9:50:13 AM<unique#>8
FRC: No robot code is currently running.
```

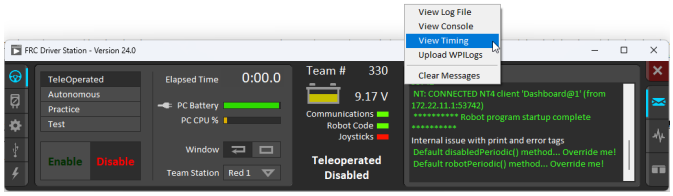
Este mensaje se muestra cuando la DS inicia a comunicarse con la roboRIO, pero detecta que no hay un código ejecutándose. Una sola instancia de este mensaje será mostrada si la Driver Station está abierta y funcionando mientras la roboRIO está arrancando mientras la DS empezara una comunicación con la roboRIO antes que el código del robot termine de cargar.

20.5 Driver Station Timing Viewer

The 2024 Driver Station has a a new window to help diagnose robot control issues.

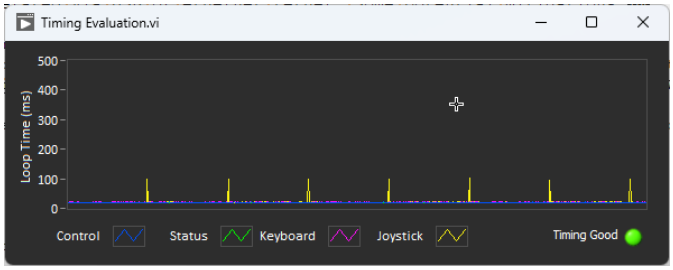
20.5.1 Opening the Timing Viewer

To start the Driver Station Timing Viewer, select the gear icon and then select *View Timing*.

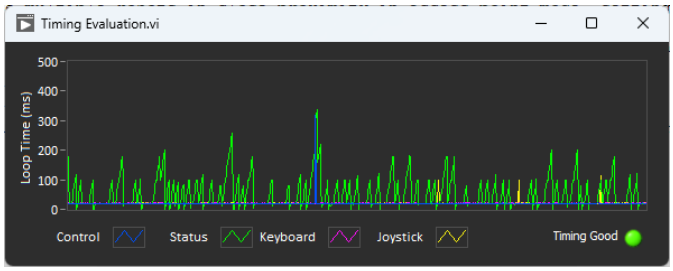


20.5.2 Viewing Timing

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 Programación de Radios para 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 Requisitos Previos

Install the FRC® Radio Configuration Utility software per the instructions in *Programming your radio*

Antes que empiece a usar el software:

1. Deshabilite las conexiones WiFi en su computadora, ya que puede impedir a la utilidad de la configuración comunicarse adecuadamente con el puente de red.
2. Plug directly from your computer into the wireless bridge ethernet port closest to the power jack. Make sure no other devices are connected to your computer via ethernet. If powering the radio via PoE, plug an Ethernet cable from the PC into the socket side of the PoE adapter (where the roboRIO would plug in). If you experience issues configuring through the PoE adapter, you may try connecting the PC to the alternate port on the radio.

Configuración Programada

La FRC Radio Configuration Utility programa una serie de ajustes de configuración en la radio cuando se inicia. Esta configuración se aplica a la radio en todos los modos (incluso en eventos). Éstos incluyen:

- Set a static IP of `10.TE.AM.1` (*TE.AM IP Notation*)
- Establecer una IP alternativa en el lado cableado de `192.168.1.1` para programación futura
- Conectar los puertos cableados para que puedan ser usados indistintamente
- The LED configuration noted in the status light referenced below.
- Límite de ancho de banda de 4 Mb/s en el lado de salida de la interfaz inalámbrica (puede deshabilitarse para uso doméstico)
- Reglas QoS para priorización interna de paquetes (afecta el búfer interno y qué paquetes descartar si se alcanza el límite de ancho de banda). Estas reglas son:

  - Robot Control y Estatus (UDP `1110`, `1115`, `1150`)
  - Robot TCP & *NetworkTables* (TCP `1735`, `1740`)
  - Bulk (todo el resto del tráfico). (deshabilitado si el límite de ancho de banda está deshabilitado)
- *DHCP* server enabled. Serves out:
  - `10.TE.AM.11` - `10.TE.AM.111` en el lado cableado
  - `10.TE.AM.138` - `10.TE.AM.237` en el lado inalámbrico
  - Máscara de subred `255.255.255.0`
  - Dirección de transmisión `10.TE.AM.255`
- DNS server enabled. DNS server IP and domain suffix (`.lan`) are served as part of the DHCP.

Truco: Consulte la *Status Light Reference* para obtener detalles sobre el comportamiento de las luces de estado de la radio cuando está configurada

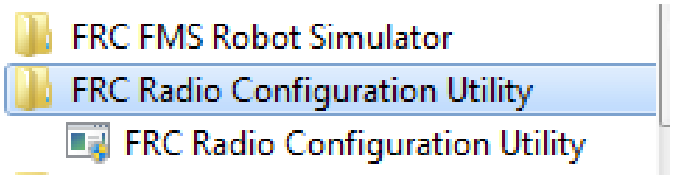
---

Cuando se programa con la versión del equipo de Radio Configuration - Utility, las cuentas de usuario se dejarán en (o se establecerán en) el firmware - predeterminado **for the DAPs only**:

- Nombre de Usuario: root
- Contraseña: root

Nota: No se recomienda modificar la configuración manualmente.

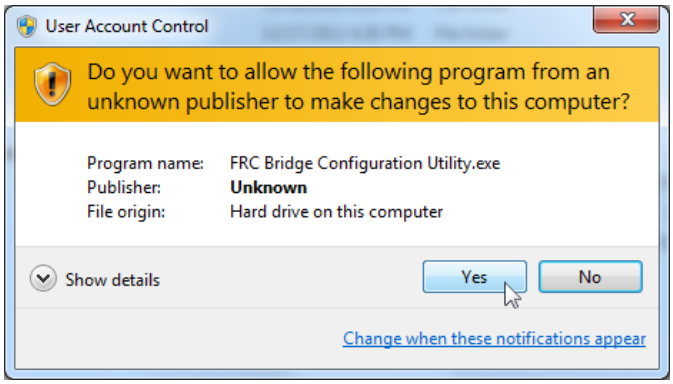
20.6.2 Iniciar el software



Use el menu de inicio o el atajo del escritorio para iniciar el programa.

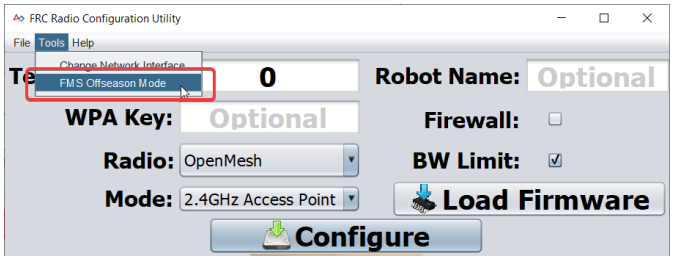
Nota: Si usted necesita localizar el programa, está instalado en `C:/Program Files (x86)/FRC Radio Configuration Utility`. Para computadoras de 32-bit la ubicación es `C:/Program Files/FRC Radio Configuration Utility/`

20.6.3 Permitir al programa hacer cambios si lo solicita



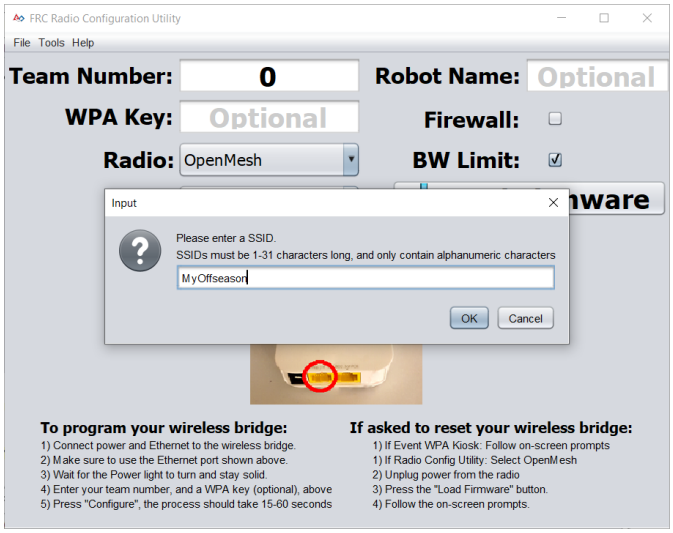
A prompt may appear about allowing the configuration utility to make changes to the computer. Click *Yes* if the prompt appears.

20.6.4 Enter FMS Offseason Mode



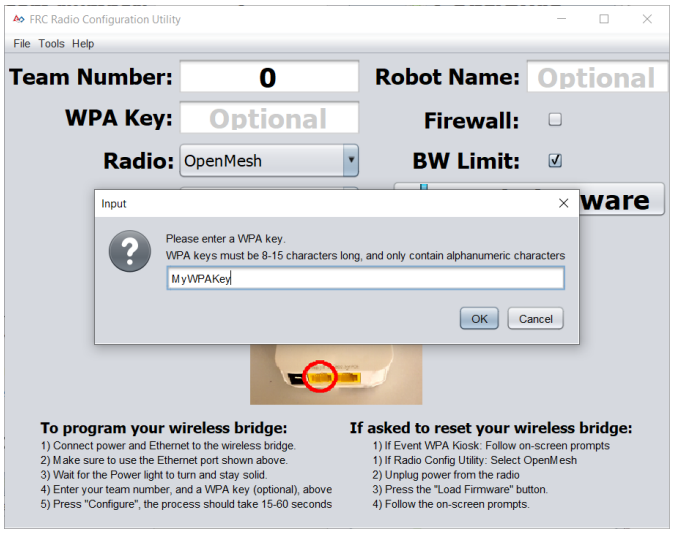
De clic en `Tools -> FMS-Lite Mode` para seleccionar FMS-Lite Mode.

20.6.5 Ingrese un SSID



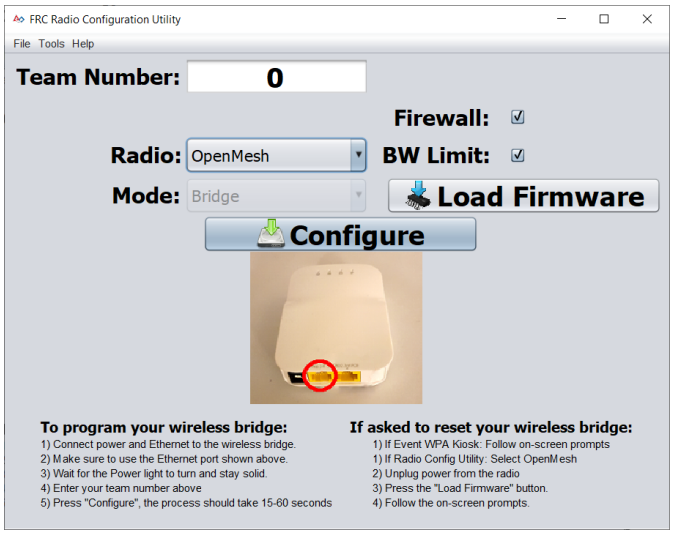
Ingrese el SSID (nombre) de su red inalámbrica en el recuadro y de clic en OK.

20.6.6 Ingrese una clave WPA



Ingrese una clave de WPA para su red en el recuadro y de clic en OK. Deje el recuadro en vacío si está usando una red insegura.

20.6.7 Programar Radios



El Kiosk está ahora listo para programar cualquier número de radios para conectarse a la red ingresada. Para programar cada radio, conecte la radio a el Kiosk, ingrese su número de equipo en el recuadro, y de clic en Configure.

El Kiosk programará radios OpenMesh, D-Link Rev A or D-Link Rev B para trabajar en una red FMS Offseason al seleccionar la opción adecuada en el menú “Radio”.

Nota: Los límites de banda ancha y QoS no se configurarán en los radios D-Link en este modo.

20.6.8 Cambiar el SSID o una clave de WPA

Si usted ingresa algo incorrecto o necesita cambiar la SSID o la clave de WPA, vaya al menú Tools y de clic en FMS-Lite Mode para sacar el Kiosk de FMS-Lite Mode. Cuando de clic de nuevo para configurar el Kiosk de nuevo en FMS-Lite Mode, se le solicitará otra vez por el SSID y la clave WPA.

20.6.9 Solución de problemas

See the troubleshooting steps in *Programming your radio*

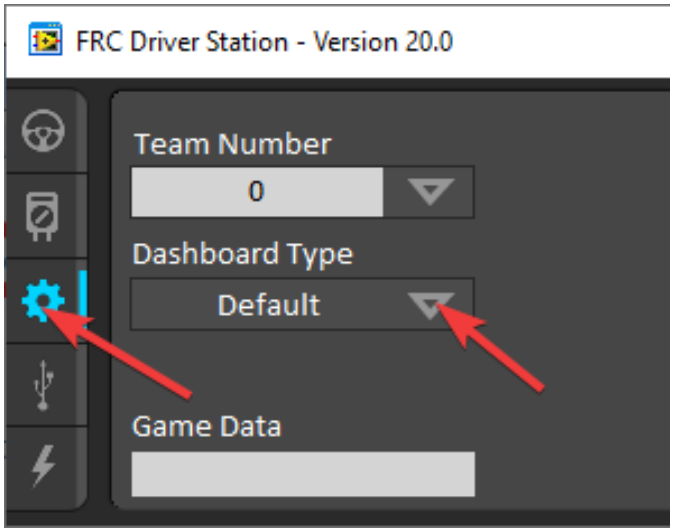
20.7 Configurar manualmente la Driver Station para iniciar a personalizar la Dashboard

---

Nota: Si WPILib no está instalando en la ubicación predeterminada (como cuando un archivo es copiado a la computadora manualmente), es probable que la dashboard seleccionada no se inicie correctamente. Para hacer que la DS inicie una dashboard personalizada cuando inicie, tiene que modificar manualmente los ajustes predeterminados en la dashboard.

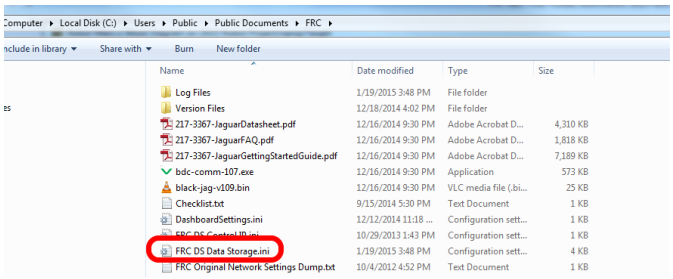
Advertencia: Esto no es necesario para la mayoría de las instalaciones, trate de usar el tipo de configuración apropiada *Dashboard Type setting* de acuerdo al lenguaje que use.

20.7.1 Establecer la Driver Station a Default



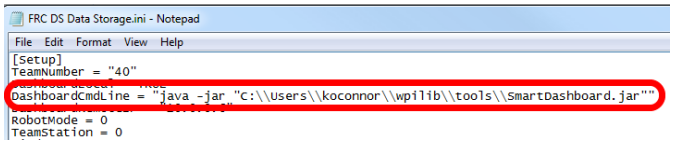
Abra el software de Driver Station, dé click en la pestaña “Setup” y seleccione “Default” en el tipo de Dashboard. **Then close the Driver Station!**

20.7.2 Abrir el archivo DS Data Storage file



Vaya a la siguiente ubicación `C:\Users\Public\Documents\FRC` y dé doble click en `FRC DS Data Storage` para abrir el archivo.

20.7.3 DashboardCmdLine



Localice la linea empezando con `DashboardCmdLine`. Modifíquelo para que apunte la dashboard para que se inicie cuando abra la driver station

LabVIEW Custom Dashboard

Remplace la cadena después del `=` con `"C:\\PATH\\TO\\DASHBOARD.exe"` donde el path/camino especificado es el camino al archivo `exe` de la dashboard. Guarde el archivo `FRC DS Data Storage`.

Java Dashboard

Reemplace la línea después de = with `java -jar "C:\\PATH\\TO\\DASHBOARD.jar"` donde la trayectoria especificada es la trayectoria del archivo dashboard `jar` . Guarde el archivo `FRC DS Data Storage`.

Truco: Shuffleboard y Smartdashboard requieren Java 11.

Dashboard desde el instalador de WPILib

Remplace la cadena después del `=` con `wscript "C:\\Users\\Public\\wpilib\\YYYY\\tools\\DASHBOARD.vbs"` donde `YYYY` es el año y `DASHBOARD.vbs` es ya sea `Shuffleboard.vbs` o `Smartdashboard.vbs`. Guarde el archivo `FRC DS Data Storage`.

20.7.4 Inicie la Driver Station

Ahora la Driver Station debe iniciar automáticamente la dashboard cada vez que se abra.

21 RobotBuilder

21.1 RobotBuilder - Introducción

21.1.1 Descripción general de RobotBuilder

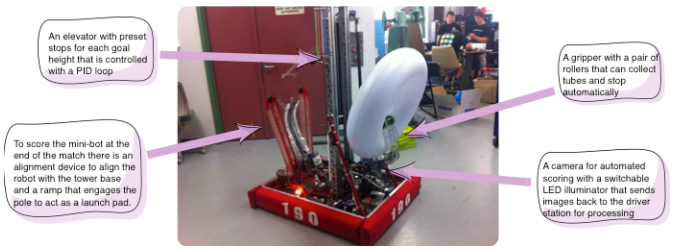
RobotBuilder es una aplicación diseñada para ayudar al proceso de desarrollo de robots. RobotBuilder puede ayudarlo a:

- Generación de código boilerplate.
- Organice su robot y averigüe cuáles son sus principales subsistemas.
- Compruebe que tiene suficientes canales para todos sus sensores y actuadores.
- Generar diagramas de cableado.
- Modifique fácilmente su interfaz de operador.
- Más...

La creación de un programa con RobotBuilder es un procedimiento muy sencillo al seguir pocos pasos que son los mismos para cualquier robot. Esta lección describe los pasos que puede seguir. Puede encontrar más detalles sobre cada uno de estos pasos en las secciones siguientes del documento.

Nota: RobotBuilder genera código utilizando el nuevo marco de trabajo de comandos. Para más detalles sobre el nuevo marco de trabajo, consulte *Command Based Programming*.

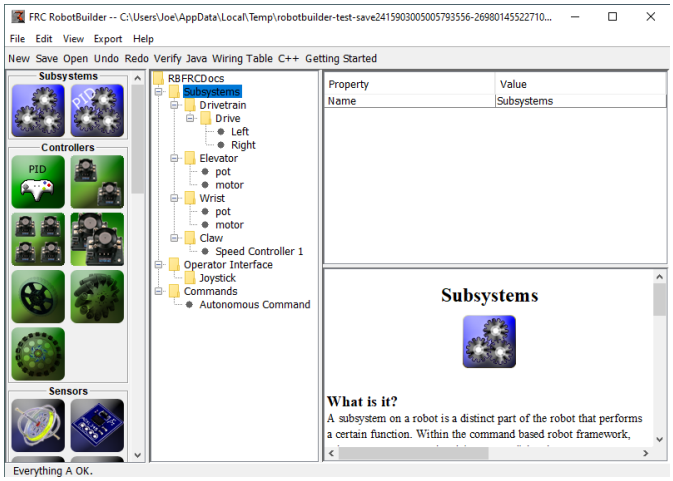
Dividir el robot en subsistemas



Su robot se compone naturalmente de varios sistemas más pequeños como drive trains, brazos, disparadores, recolectores, manipuladores, articulaciones de muñeca, etc. Debe observar el diseño de su robot y dividirlo en subsistemas más pequeños operados por separado. En este ejemplo particular, hay un ascensor, un dispositivo de alineación de minibot, una pinza y un sistema de cámara. Además, se podría incluir la base de la unidad. Cada una de estas partes del robot se controla por separado y son buenas candidatas para subsistemas.

Para obtener más información, consulte :doc:`Creating a Subsystem<robotbuilder-creating-subsystem> `.

Agregar cada subsistema al proyecto



Cada subsistema se agregará a la carpeta «Subsystems» en RobotBuilder y se le dará un nombre significativo. Para cada uno de los subsistemas hay varios atributos que se completan para especificar más información sobre los subsistemas. Además, hay dos tipos de subsistemas que quizás desee crear:

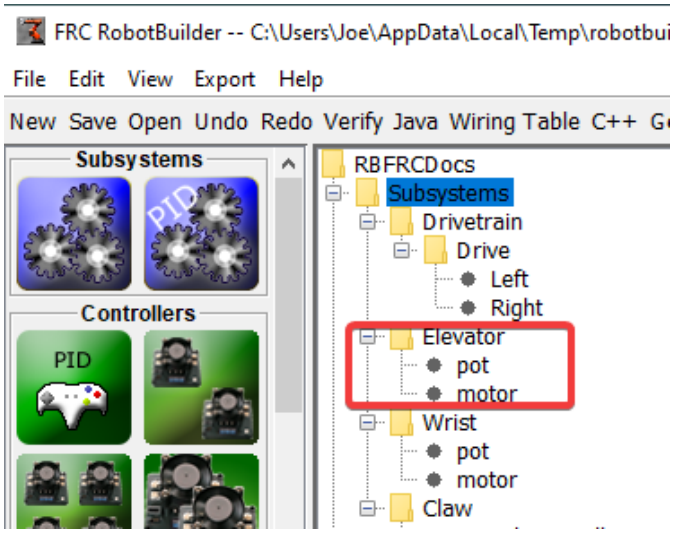
1. SubsistemasPID - a menudo es deseable controlar la operación de un subsistema con un controlador PID. Este es un código en su programa que hace que el elemento del subsistema, por ejemplo el ángulo del brazo, sea más rápido hasta una posición deseada y luego se detenga al alcanzarla. Los SubsistemasPID tienen el código del controlador PID incorporado y a menudo son más convenientes que agregarlo usted mismo. Los SubsistemasPID tienen un sensor que determina cuando el dispositivo ha alcanzado la posición deseada y un actuador (controlador de motor) que es conducido al punto de ajuste.
2. Subsistema regular - estos subsistemas no tienen un controlador PID integrado y se utilizan para subsistemas sin control PID para retroalimentación o para subsistemas

que requieren un control más complejo que el que se puede manejar con el controlador PID integrado predeterminado.

A medida que lea más de esta documentación, las diferencias entre los tipos de subsistemas se harán más evidentes.

Para obtener más información, consulte *Creación de un subsistema* y *Escritura de código para un subsistema*.

Agregar componentes a cada uno de los subsistemas



Cada subsistema consta de una serie de actuadores, sensores y controladores que utiliza para realizar sus operaciones. Estos sensores y actuadores se agregan al subsistema con el que están asociados. Cada uno de los sensores y actuadores proviene de la paleta RobotBuilder y se arrastra al subsistema apropiado. Para cada uno, generalmente hay otras propiedades que se deben configurar, como números de puerto y otros parámetros específicos del componente.

En este ejemplo, hay un subsistema Elevator que utiliza un motor y un potenciómetro (motor y potenciómetro) que se han arrastrado al subsistema Elevator.

Agregar comandos que describen los objetivos del subsistema

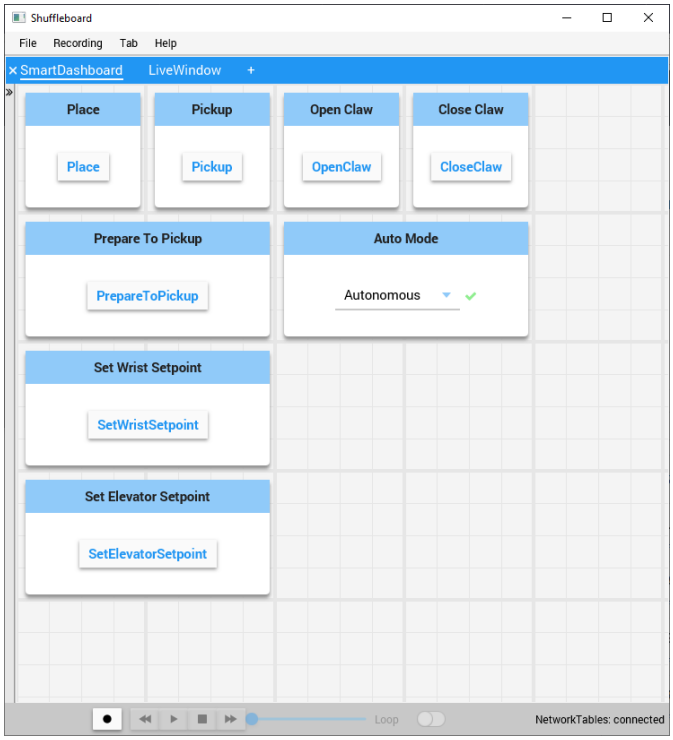
Los comandos son distintos objetivos que el robot realizará. Estos comandos se añaden arrastrando el comando bajo la carpeta «Comandos». Al crear un comando, hay 7 opciones (mostradas en la paleta de la izquierda de la imagen):

- Comandos normales - estos son los comandos más flexibles, debe escribir todo el código para realizar las acciones deseadas necesarias para lograr el objetivo.
- Comandos temporizados - estos comandos son una versión simplificada de un comando que termina después de un tiempo de espera
- Instant commands - these commands are a simplified version of a command that runs for one iteration and then ends
- Grupos de comandos - estos comandos son una combinación de otros comandos que se ejecutan tanto en orden secuencial como en paralelo. Úselos para crear acciones más complicadas después de haber implementado una serie de comandos básicos.

- Comandos de punto de ajuste - los comandos de punto de ajuste mueven un subsistema PID a un punto de ajuste fijo o la ubicación deseada.
- Comandos PID - estos comandos tienen un controlador PID incorporado para ser utilizado con un subsistema regular.
- Comandos temporizados - estos comandos son una versión simplificada de un comando que termina después de un tiempo de espera

Para obtener más información, consulte *Creación de un comando* y *Escritura del código de comando*.

Probar cada comando

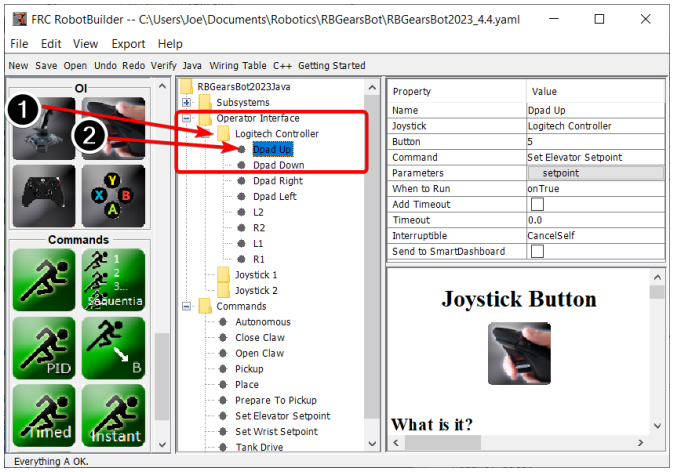


Cada comando se puede ejecutar desde SmartDashboard. Esto es útil para probar comandos antes de agregarlos a la interfaz del operador o a un grupo de comandos. Siempre que deje marcada la propiedad «Botón en SmartDashboard», se creará un botón en SmartDashboard. Cuando presiona el botón de inicio, el comando se ejecutará y podrá verificar que realiza la acción deseada.

Al crear botones, cada comando se puede probar individualmente. Si todos los comandos funcionan individualmente, puede estar bastante seguro de que el robot funcionará como un todo.

Para obtener más información, consulte *Testing with Smartdashboard*.

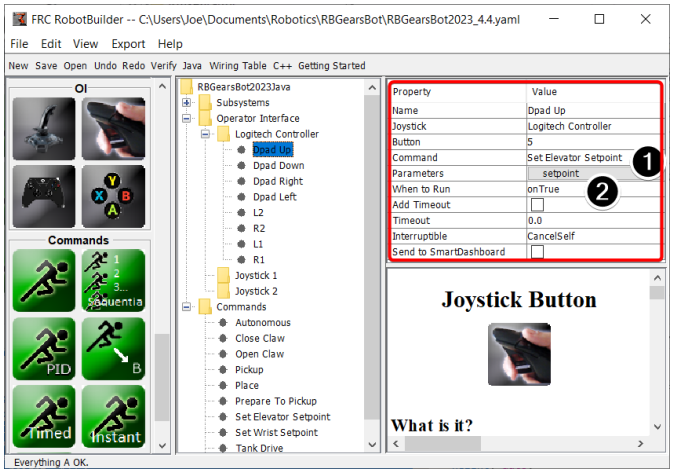
Adición de componentes de la interfaz del operador



La interfaz del operador consta de joysticks, gamepads y otros dispositivos de entrada HID. Puede agregar componentes de interfaz de operador (joysticks, botones de joystick) a su programa en RobotBuilder. Generará automáticamente un código que inicializará todos los componentes y permitirá que se conecten a los comandos.

Los componentes de la interfaz del operador se arrastran desde la paleta a la carpeta «Interfaz del operador» en el programa RobotBuilder. Primero (1) agregue Joysticks al programa, luego coloque botones debajo de los joysticks asociados (2) y asígneles nombres significativos, como ShootButton.

Conexión de los comandos a la interfaz del operador

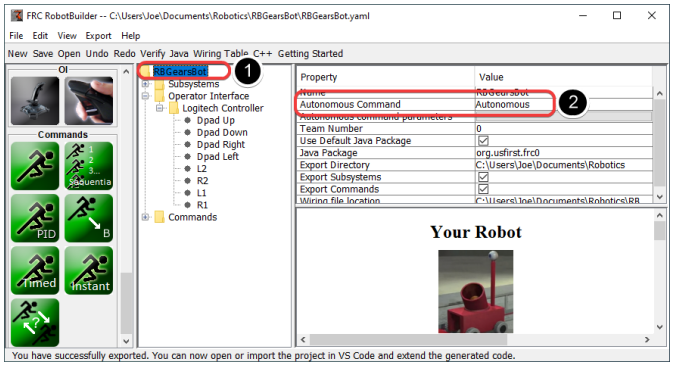


Los comandos se pueden asociar con botones para que cuando se presione un botón, el comando esté programado. Esto debería, en su mayor parte, manejar la mayor parte de la parte teleoperada de su programa de robot.

Esto se hace simplemente (1) agregando el comando al objeto JoystickButton en el programa RobotBuilder, luego (2) estableciendo la condición en la que está programado el comando.

Para obtener más información, consulte *Conexión de la interfaz del operador a un comando*.

Desarrollo de comandos autónomos

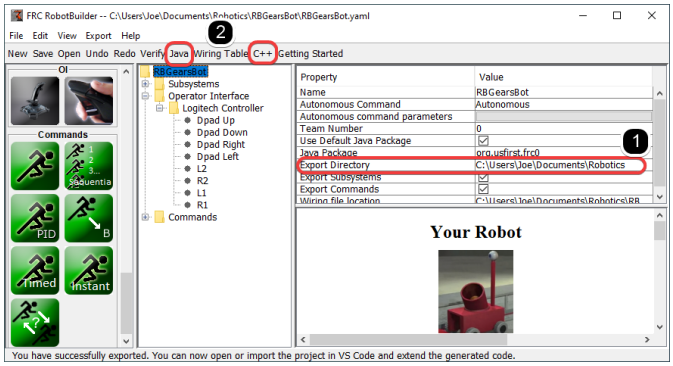


Los comandos simplifican el desarrollo de programas autónomos. Simplemente especifique qué comando debe ejecutarse cuando el robot ingrese al período autónomo y se programará automáticamente. Si ha probado comandos como se mencionó anteriormente, esto debería ser simplemente una cuestión de elegir qué comando debe ejecutarse.

Seleccione el robot en la raíz del proyecto RobotBuilder (1), luego edite la propiedad Autonomous Command (2) para elegir el comando a ejecutar. ¡Es así de simple!

Para obtener más información, consulte *Setting the Autonomous Commands*.

Generar código



En cualquier punto del proceso descrito anteriormente, puede hacer que RobotBuilder genere un programa C++ o Java que representará el proyecto que ha creado. Esto se hace especificando la ubicación del proyecto en las propiedades del proyecto (1), luego haciendo clic en el botón apropiado de la barra de herramientas para generar el código (2).

Para obtener más información, consulte *Generación del código de RobotBuilder*.

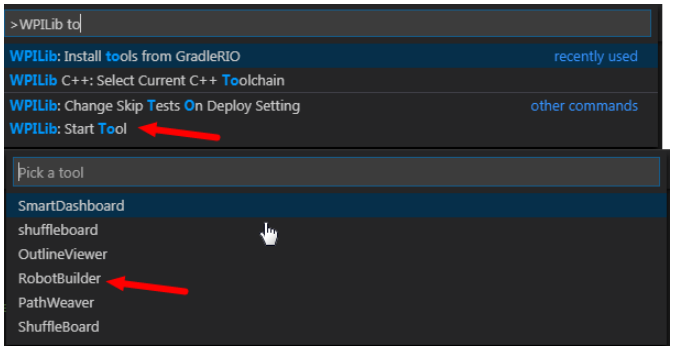
21.1.2 Iniciando RobotBuilder

Nota: RobotBuilder es un programa de Java y como tal debe ser capaz de funcionar en cualquier plataforma que sea compatible con Java. Hemos estado ejecutando RobotBuilder en MacOS, Windows y varias versiones de Linux con éxito.

Consiguiendo RobotBuilder

RobotBuilder se descarga como parte del instalador offline de WPILib. Para más información, vea la *Guía de instalación en Windows/macOS/Linux*

Opción 1 - Empezando por el código de Visual Studio

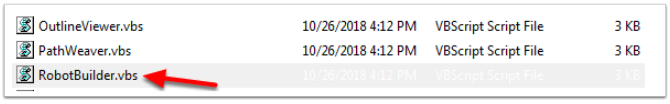


Presione `Ctrl+Shift+P` y escriba «WPILib» o haga clic en el logo de WPILib en la parte superior derecha para lanzar la paleta de comandos de WPILib. Seleccione:guilabel:*Start Tool*, luego seleccione *Robot Builder*.

Opción 2 - Atajos

Shortcuts are installed to the Windows Start Menu and the `2024 WPILib Tools` folder on the desktop.

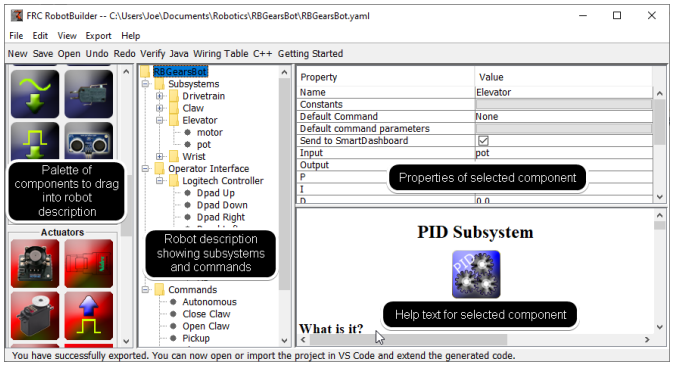
Opción 3 - Ejecución desde el Script



El proceso de instalación instala las herramientas a `~/wpilib/YYYY/tools` (donde YYYY es el año y ~`C:UsersPublic`` en Windows).

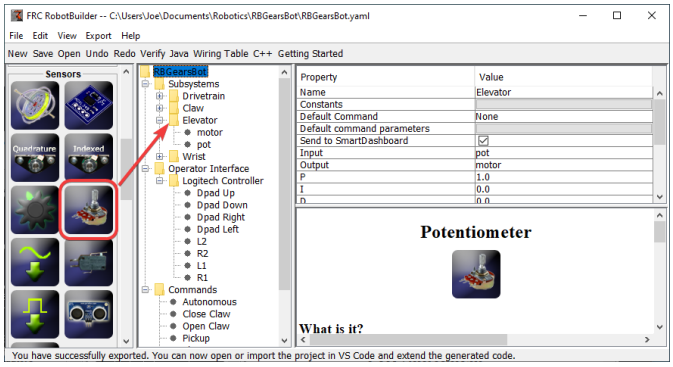
Dentro de esta carpeta encontrará archivos `.vbs` (Windows) y `.py` (macOS / Linux) que puede usar para iniciar cada herramienta. Estos scripts ayudan a iniciar las herramientas utilizando el JDK correcto y son los que debe utilizar para iniciar las herramientas.

21.1.3 Interfaz de Usuario de RobotBuilder



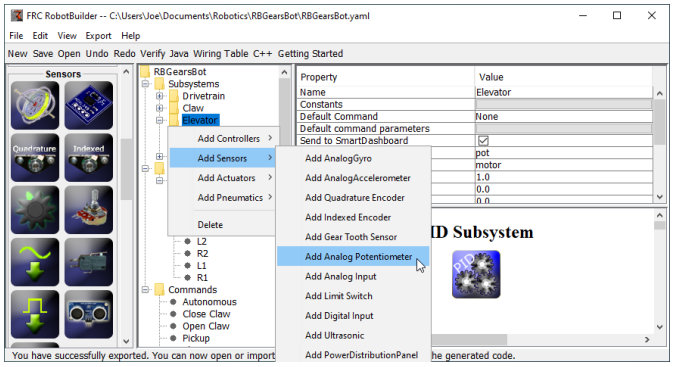
RobotBuilder tiene una interfaz de usuario diseñada para el desarrollo rápido de programas de robots. Casi todas las operaciones se realizan mediante arrastrar y soltar o seleccionando opciones de las listas desplegables.

Arrastrando elementos de la Paleta a la Descripción del Robot



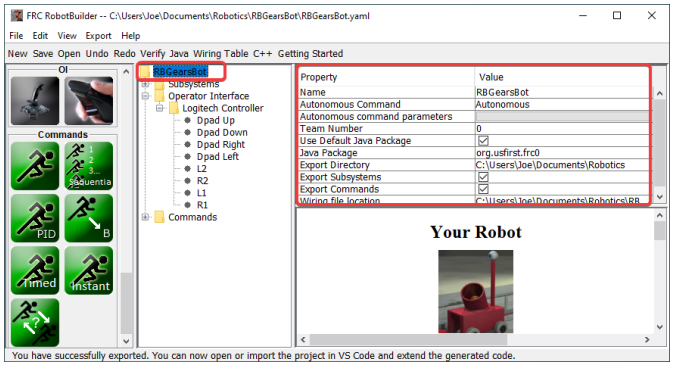
Puedes arrastrar los elementos de la paleta a la descripción del robot empezando el arrastre en el elemento de la paleta y terminando en el contenedor donde quieres que se ubique el elemento. En este ejemplo, dejando caer un potenciómetro al subsistema del Elevador.

Agregar Componentes usando el clic derecho del Menú Contextual



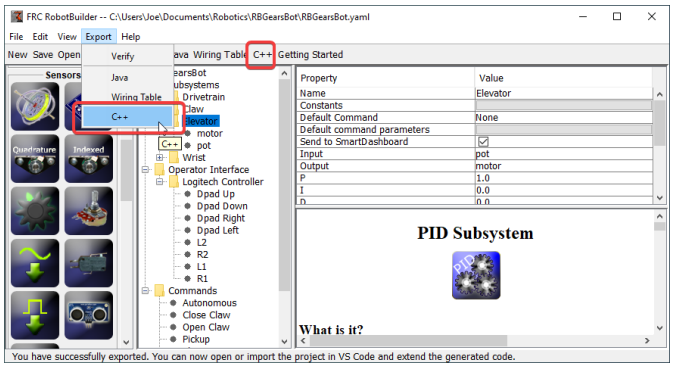
Un método abreviado para añadir elementos a la descripción del robot es hacer clic con el botón derecho del ratón en el objeto contenedor (Elevador) y seleccionar el elemento que debe añadirse (Potenciómetro). Esto es idéntico al uso de arrastrar y soltar, pero podría ser más fácil para algunas personas.

Edición de las Propiedades de los Elementos de Descripción de los Robots



Las propiedades de un elemento seleccionado aparecerán en el visor de propiedades. Las propiedades se pueden editar seleccionando el valor en la columna de la derecha.

Usando el Sistema de Menús



Las operaciones de RobotBuilder pueden seleccionarse ya sea a través del sistema de menús o el elemento equivalente (si está disponible) de la barra de herramientas.

21.1.4 Configurando el proyecto de robot

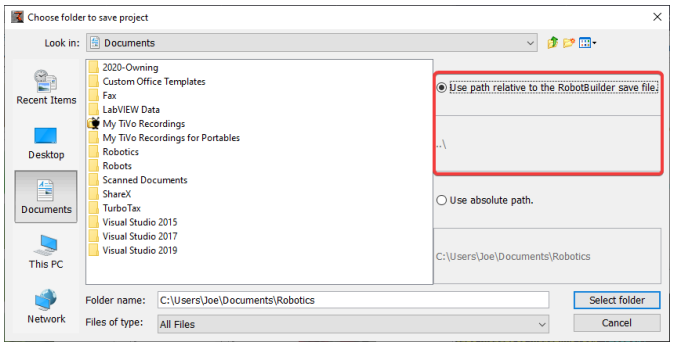
El programa RobotBuilder tiene algunas propiedades predeterminadas que deben configurarse para que el programa y otros archivos generados funcionen correctamente. Esta información se almacena en las propiedades para la descripción del robot (la primera línea).

Propiedades del proyecto de robot

Las propiedades que describen al robot son:

- **Name** - El nombre del proyecto de robot que fue creado
- **Comando Autónomo** - el comando que se ejecutará por defecto cuando el programa se ponga en modo autónomo
- **Parámetros del Comando Autónomo** - Parámetros del comando autónomo
- **Número del Equipo** - El número de equipo del proyecto, que se utilizará para localizar el robot al desplegar el código.
- **Utiliza el Paquete de Java Predeterminado** - Si se marca, RobotBuilder utilizará el paquete predeterminado (frc.robot). De lo contrario, puede especificar un nombre de paquete personalizado para ser utilizado.
- **Java Package** - El nombre del paquete Java generado que se utilizó al generar el código del proyecto.
- **Directorio de Exportación** - La carpeta en la que se genera el proyecto cuando se selecciona Exportar a Java o C++
- **Export Subsystems** -Marcado si RobotBuilder deberá exportar las clases de subsistema de su proyecto
- **Export Commands** - Marcado si RobotBuilder deberá exportar las clases de comandos de su proyecto
- **Wiring File location** - the location of the html file to generate that contains the wiring diagram for your robot
- **Desktop Support** - Enables unit test and simulation. While WPILib supports this, third party software libraries may not. If libraries do not support desktop, then your code may not compile or may crash. It should be left unchecked unless unit testing or simulation is needed and all libraries support it.

Uso de la fuente del control con el proyecto RobotBuilder



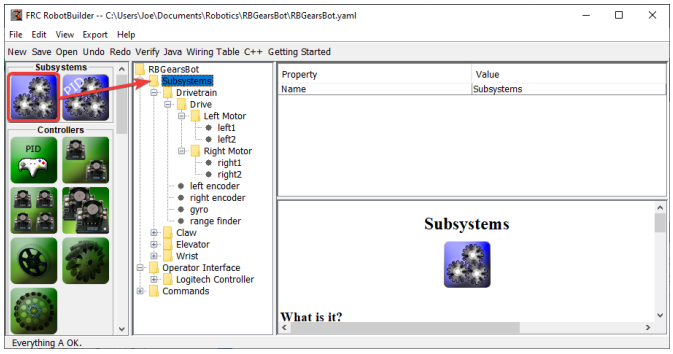
Cuando se utiliza el control de código fuente, el proyecto se utilizará normalmente en varias computadoras y la ruta al directorio del proyecto puede ser diferente de una computadora a otra. Si el archivo de proyecto de RobotBuilder se almacena utilizando una ruta absoluta, normalmente contendrá el nombre de usuario y no se podrá utilizar en varias computadoras. Para que esto funcione, seleccione «ruta relativa» y especifique la ruta como un desplazamiento de directorio de los archivos del proyecto. En el ejemplo anterior, el archivo del proyecto se almacena en la carpeta justo encima de los archivos del proyecto en la jerarquía de archivos.

En este caso, el nombre de usuario no forma parte de la ruta y será portable en todas sus computadoras.

21.1.5 Creando un subsistema

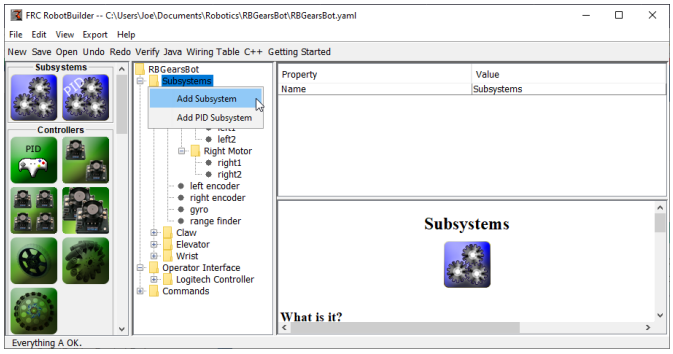
Subsystems are classes that encapsulate (or contain) all the data and code that make a subsystem on your robot operate. The first step in creating a robot program with the RobotBuilder is to identify and create all the subsystems on the robot. Examples of subsystems are grippers, ball collectors, the drive base, elevators, arms, etc. Each subsystem contains all the sensors and actuators that are used to make it work. For example, an elevator might have a Victor SPX motor controller and a potentiometer to provide feedback of the robot position.

Creación de un subsistema utilizando la paleta



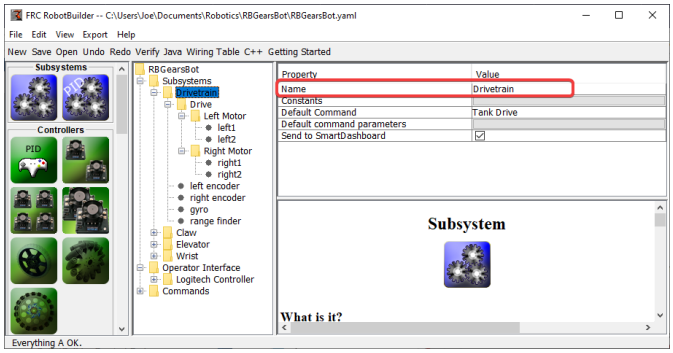
Arrastra el ícono del subsistema de la paleta a la carpeta Subsystems en la descripción del robot para crear una clase de subsistema.

Creación de un subsistema utilizando el menú contextual



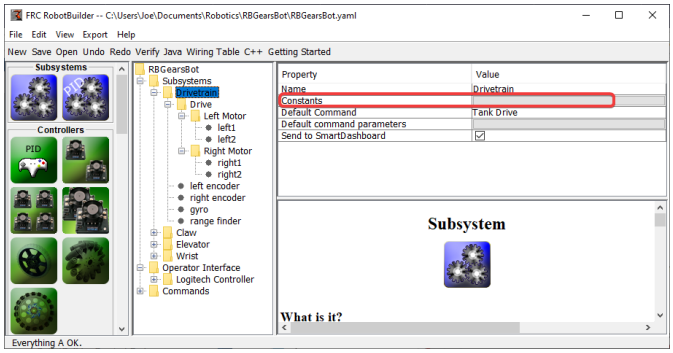
Haga clic con el botón derecho del ratón en la carpeta Subsystem de la descripción del robot para añadir un subsistema a esa carpeta.

Nombrando el subsistema



Después de crear el subsistema, ya sea arrastrando o usando el menú contextual como se describe arriba, simplemente escriba el nombre que desea darle al subsistema. El nombre puede ser varias palabras separadas por espacios, RobotBuilder concatenará las palabras para hacer un nombre de clase Java o C++ apropiado para usted.

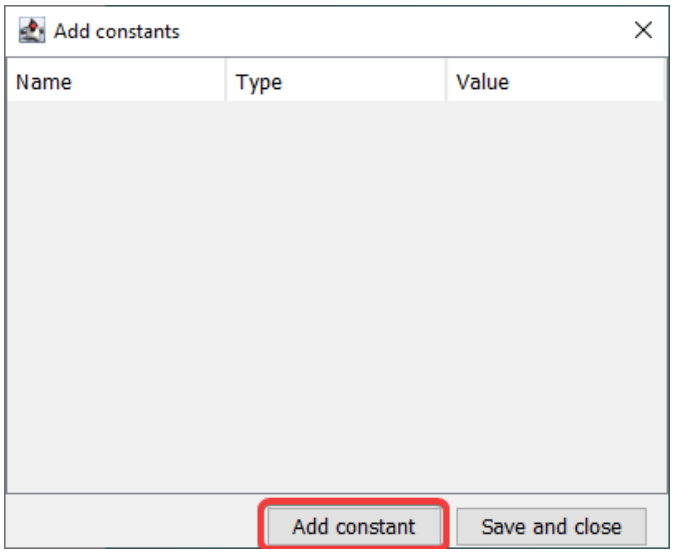
Agregando Constantes



Las constantes son muy útiles para reducir la cantidad de números mágicos en tu código. En los subsistemas, se pueden usar para llevar un registro de ciertos valores, como los valores de los sensores para alturas específicas de un ascensor, o la velocidad a la que se conduce el robot.

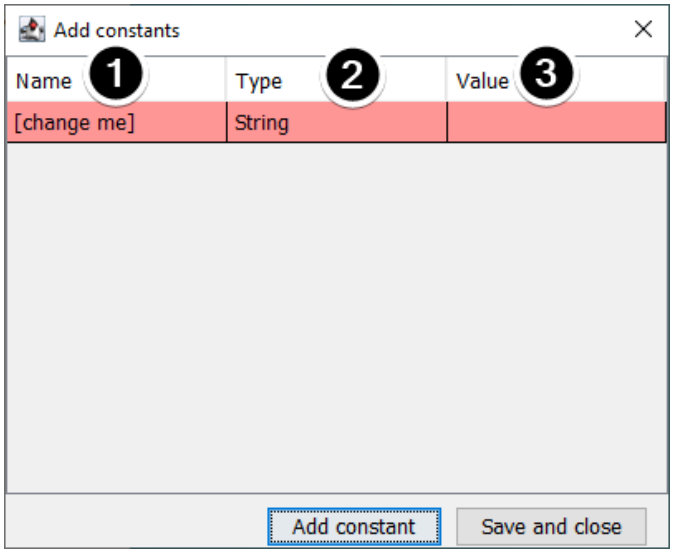
Por defecto, no habrá constantes en un subsistema. Presiona el botón junto a «Constantes» para abrir un diálogo para crear algunas.

Creando Constantes



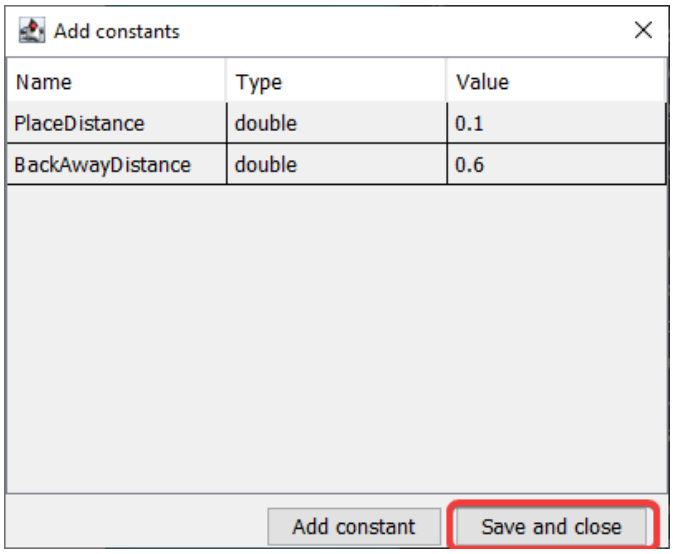
La mesa de las constantes estará vacía al principio. Presiona «Añadir constante» para añadir una.

Añadir Constantes



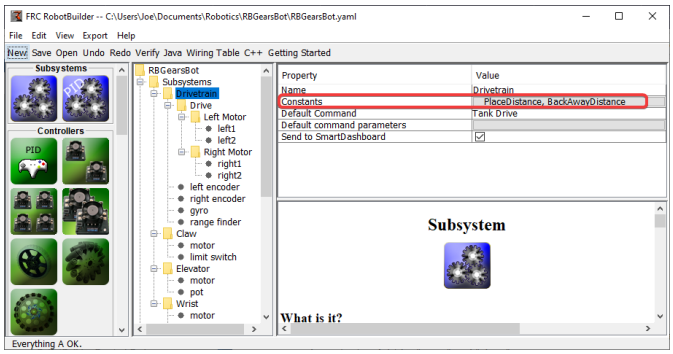
1. El nombre de la constante. Cámbialo por algo descriptivo. En este ejemplo de una transmisión, algunas buenas constantes podrían ser «PlaceDistance» y «BackAwayDistance».
2. El tipo de la constante. Lo más probable es que sea un doble, pero puede elegir uno de los siguientes: String, double, int, long, boolean, o byte.
3. El valor de la constante.

Guardando Constantes



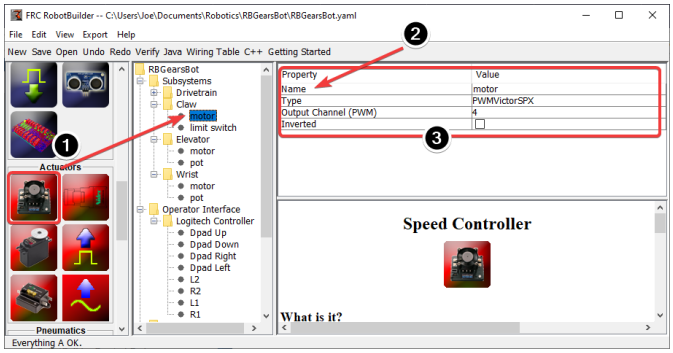
Después de añadir constantes y establecer sus valores, sólo tienes que pulsar «Guardar y cerrar» para guardar las constantes y cerrar el diálogo. Si no quieres guardar, presiona el botón de salida en la parte superior de la ventana.

Después de guardar



Después de guardar las constantes, los nombres aparecerán en el botón «Constantes» en las propiedades del subsistema.

Arrastrando los actuadores/sensores al subsistema



Hay tres pasos para añadir componentes a un subsistema:

1. Arrastre los actuadores o sensores de la paleta al subsistema según sea necesario.
2. Dale al actuador o al sensor un nombre significativo
3. Edita las propiedades como los números de módulo y los números de canal para cada elemento del subsistema.

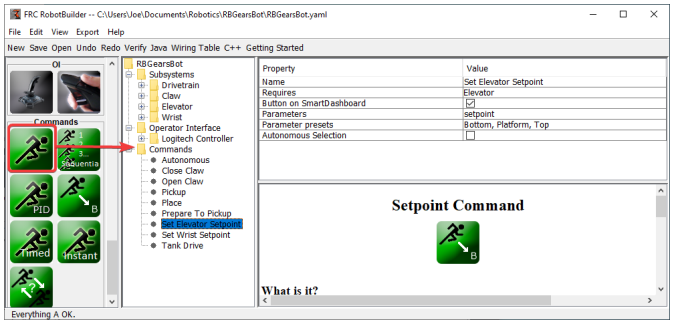
RobotBuilder usará automáticamente números de canal crecientes para cada módulo del robot. Si aún no ha cableado el robot, puede dejar que RobotBuilder asigne números de canal únicos para cada sensor o actuador y cablear el robot según la tabla de cableado generado.

Esto sólo crea el subsistema en RobotBuilder, y posteriormente generará el código del esqueleto para el subsistema. Para hacer que funcione realmente su robot, por favor, consulte: ref:*Escribiendo un Código para un Subsistema <docs/software/wpilib-tools/robotbuilder/writing-code/robotbuilder-writing-subsystem-code:Writing the Code for a Subsystem>*.

21.1.6 Creando un comando

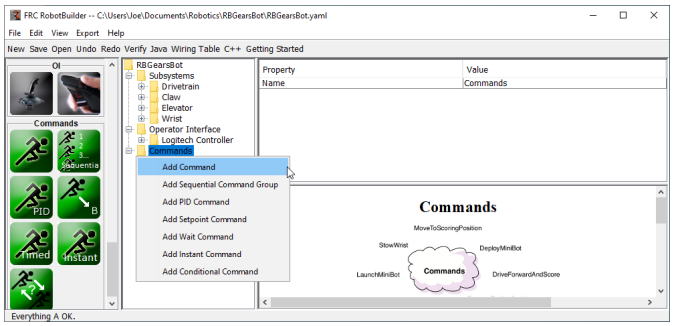
Los comandos son clases que creas que proporcionan comportamientos o acciones para los subsistemas. La clase de subsistema debe establecer el funcionamiento del subsistema, como MoveElevator para iniciar el movimiento del ascensor, o ElevatorToSetPoint para establecer el punto de ajuste PID del ascensor. Los comandos inician la operación del subsistema y llevan un registro de cuándo ha terminado.

Arrastre el comando a la carpeta de comandos



Se pueden arrastrar comandos simples de la paleta a la descripción del robot. El comando se creará en la carpeta Comandos.

Creación de comandos usando el menú contextual



También puede crear comandos utilizando el menú contextual del botón derecho del ratón en la carpeta Comando de la descripción del robot.

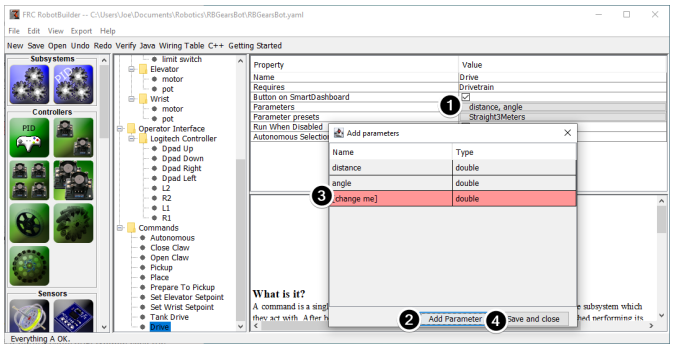
Configurando el comando

Property	Value
Name	Set Elevator Setpoint
Requires	Elevator
Button on SmartDashboard	☑
Parameters	setpoint
Parameter presets	Bottom, Platform, Top
Autonomous Selection	☐

1. Nombra el comando con algo significativo que describa lo que el comando hará. Los comandos deben ser nombrados como si estuvieran en código, aunque puede haber espacios entre las palabras.
2. Configure el subsistema requerido por este comando. Cuando este comando está programado, detendrá automáticamente cualquier comando que se esté ejecutando y que también requiera este comando. Si se está ejecutando un comando para abrir la garra (que requiere el subsistema de garra) y el comando de cierre de la garra está programado, dejará de abrirse inmediatamente y comenzará a cerrarse.
3. Dile a RobotBuilder si debe crear botones en el SmartDashboard para el comando. Se creará un botón para cada parámetro preestablecido.
4. Establezca los parámetros que toma este comando. Un solo comando con parámetros puede hacer lo mismo que dos o más comandos que no toman parámetros. Por ejemplo, los comandos «Avance», «Retroceso» y «Distancia» pueden consolidarse en un solo comando que toma valores de dirección y distancia.
5. Establecer preselecciones para los parámetros. Estos pueden ser utilizados en cualquier parte de RobotBuilder cuando se utiliza el comando, como por ejemplo, enlazarlo a un botón del joystick o establecer el comando por defecto para un subsistema.
6. *Run When Disabled*. Permite que el comando se ejecute cuando el robot está desactivado. Sin embargo, los actuadores comandados mientras están desactivados no se activarán.
7. *Autonomous Selection*. Si el comando debe agregarse al Selector de envío para que pueda seleccionarse para el autónomo.

Los comandos de punto de ajuste vienen con un solo parámetro (“punto de ajuste”, de tipo doble); los parámetros no pueden ser añadidos, editados o borrados para los comandos de punto de ajuste.

Añadiendo y editando parámetros

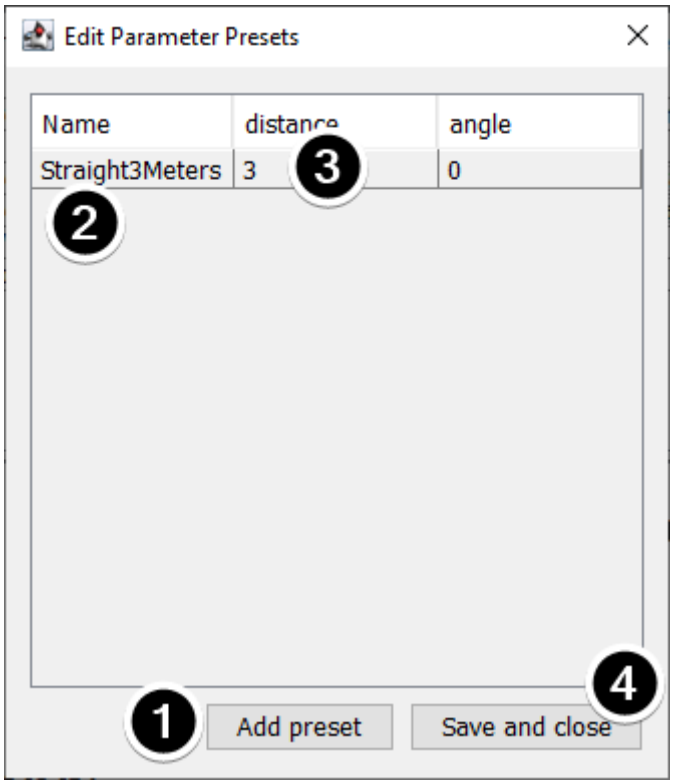


Para añadir o editar parámetros:

1. Dé clic al botón en la columna *Value* de la tabla de propiedades.
2. Presione el botón:guilabel:*Add Parameter* para agregar un parámetro.
3. Un parámetro que se acaba de agregar. El nombre predeterminado será *[change me]* y el tipo predeterminado es String. El nombre predeterminado no es válido, por lo que deberá cambiarlo antes de exportar. Haga doble clic en la celda:guilabel:*Name* para comenzar a cambiar el nombre. Haga doble clic en la celda *Type* para seleccionar el tipo.
4. El botón de guardar y cerrar guardará todos los cambios y cerrará la ventana.

Las filas se pueden reordenar simplemente arrastrando, y se pueden eliminar seleccionándolas y pulsando eliminar o retroceder.

Añadiendo y editando preajustes de parámetros

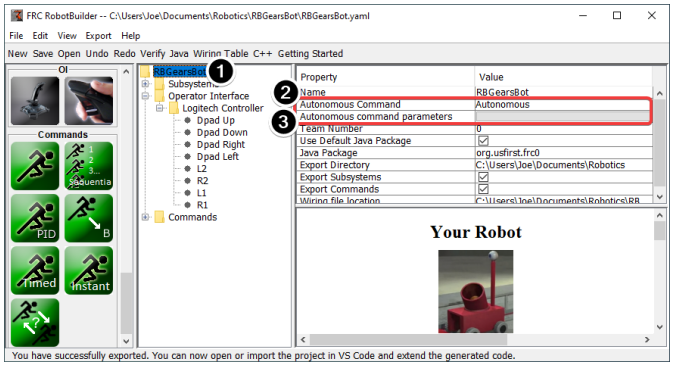


1. Haga clic en *Add parameter set* para agregar un nuevo preset.
2. Cambie el nombre de la preselección por algo descriptivo. Los preajustes en este ejemplo son para abrir y cerrar el subsistema de la pinza.
3. Cambie el valor de los parámetro(s) para el preajuste. Puede escribir un valor (por ejemplo, «3.14») o seleccionar entre las constantes definidas en el subsistema que el comando requiere. Tenga en cuenta que el tipo de la constante tiene que ser del mismo tipo que el parámetro – no se puede pasar una constante de tipo int a un parámetro de tipo doble, por ejemplo
4. Haga clic en *Save and close* para guardar los cambios y salir del diálogo; para salir sin guardar, presione el botón de salida en la barra superior de la ventana.

21.1.7 Configuración de los comandos autónomos

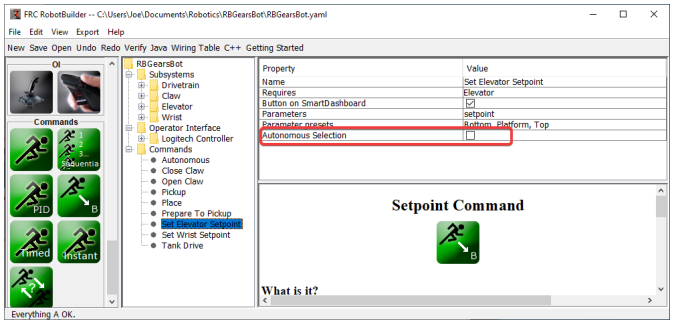
Dado que un comando es simplemente una o más acciones (comportamientos) que el robot realiza, hace sentido para describir la operación autónoma de un robot como un comando. Aunque podría ser un solo comando, es más probable que sea un grupo de comandos (un grupo de comandos que sucederán juntos).

RobotBuilder generates code for a *Sendable Chooser* which allows the autonomous command to run to be chosen from the dashboard.



Para designar el comando autónomo por defecto que se ejecuta si no se selecciona otro comando en el dashboard:

- Seleccione el robot en la descripción del programa del robot
- Rellene el campo Comando autónomo con el comando que debe ejecutarse cuando el robot se ponga en modo autónomo. Este es un campo desplegable y le dará la opción de seleccionar cualquier comando que haya sido definido.
- Establece los parámetros que toma el comando, si los hay.



Para seleccionar los comandos que se añadirán como opciones al seleccionador enviable, seleccione la casilla de selección autónoma.

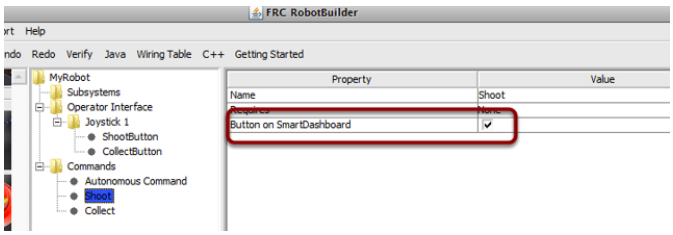
Cuando el robot se ponga en modo autónomo, se programará el comando autónomo elegido.

21.1.8 Usar Shuffleboard para Probar un Comando

Los comandos se prueban fácilmente agregando un botón en el Shuffleboard/SmartDashboard para activar el comando. De esta manera, no es necesaria la integración con el resto del programa del robot y los comandos pueden ser fácilmente probados de forma independiente. Esta es la forma más fácil de verificar los comandos ya que con una sola línea de código en su programa, se puede crear un botón en Shuffleboard que ejecutará el comando. Estos botones pueden dejarse en su lugar para verificar los subsistemas y comandar operaciones en el futuro.

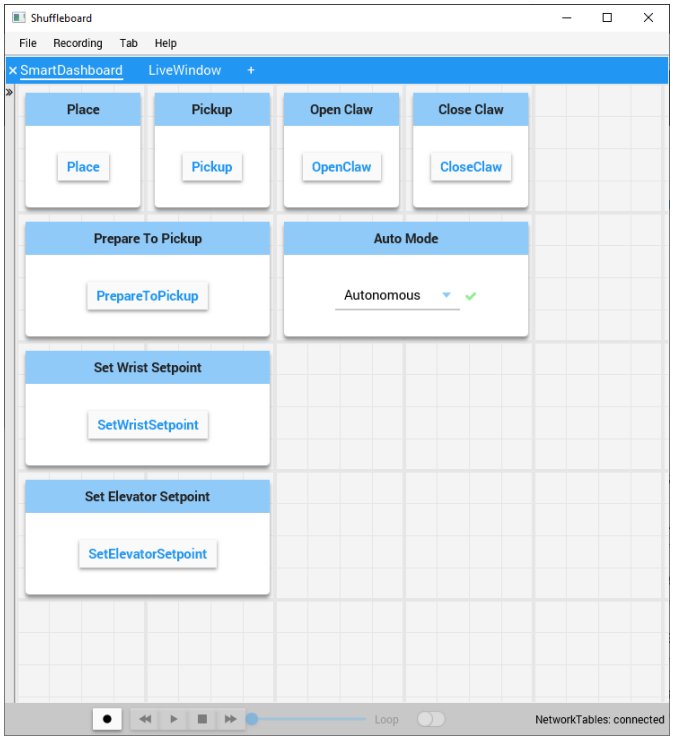
Esto tiene el beneficio añadido de acomodar múltiples programadores, cada uno escribiendo comandos. Como el código se comprueba en el proyecto principal del robot, los comandos pueden ser probados individualmente.

Crear el botón en Shuffleboard



El botón se crea en el SmartDashboard poniendo una instancia del comando desde el programa del robot a la dashboard. Esta es una operación común que ha sido añadida a RobotBuilder como una casilla de verificación. Cuando escriba comandos, asegúrese de que la casilla esté marcada, y los botones se generarán automáticamente para usted.

Operando los Botones



Los botones se generarán automáticamente y aparecerán en la pantalla de dashboard. Puede reacomodar los botones en Shuffleboard. En este ejemplo hay un número de comandos, cada uno con un botón asociado para la prueba. Presionando el botón del comando, se ejecutará. Una vez presionado, presionarlo otra vez interrumpirá el comando llamando al método `Interrupted()`.

Agregando Comandos Manualmente

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));
```

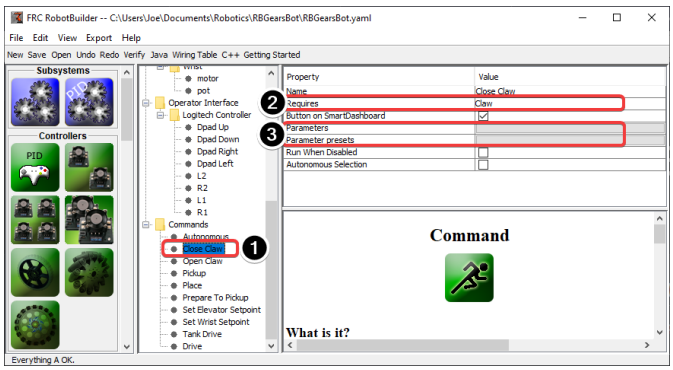
Puede agregar comandos manualmente a Shuffleboard escribiendo el código por usted. Esto se hace pasando las instancias del comando del método PutData con el nombre que debería ser asociado con el botón en Shuffleboard. Estas estancias están programadas cuando el botón esté presionado. El resultado es exactamente igual al código generado con RobotBuillder, aunque dar clic en la casilla de RobotBuilder es más sencillo que escribir todo el código manualmente.

21.1.9 Conectando la Interfaz del Operador a un Comando

Los comandos manejan los comportamientos de su robot. El comando inicia un subsistema para algunos modos de operación como subir un elevador y que continúe ejecutándose hasta que alcanza algún punto de ajuste o tiempo de espera. El comando entonces se encarga de esperar a que el subsistema termine. De esta manera los comandos pueden ejecutarse en secuencia para desarrollar comportamientos más complejos.

RobotBuilder también generará un código para programar un comando que se ejecute siempre que un botón en la interfaz del operador es presionado. También puede escribir un código para ejecutar un comando cuando una determinada condición del gatillo haya sucedido.

Ejecutar un Comando con la Pulsación de un Botón.

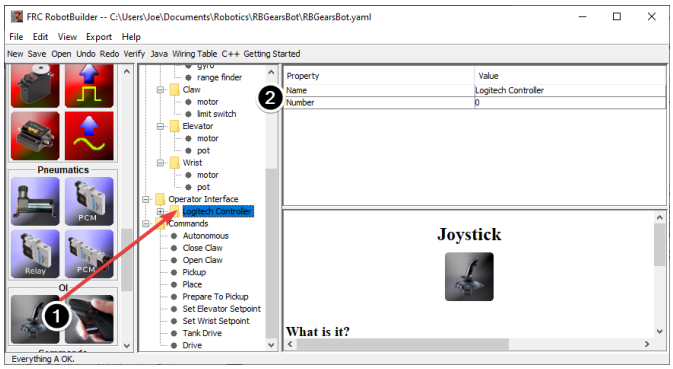


En este ejemplo queremos programa el comando «Close Claw» para ejecutar cada vez que se presiona el botón de dirección derecha en un gamepad logitech (botón 6).

1. El comando para ejecutar se llama «Close Claw» y su función es cerrar la garra del robot

2. Tenga en cuenta que el comando requiere el subsistema Claw. Esto asegurará que este comando comience a ejecutarse incluso si hubo otra operación al mismo tiempo que usó la garra. En este caso se interrumpiría el comando anterior.
3. Los parámetros hacen posible que un comando haga múltiples cosas; los preajustes permiten definir los valores que se pasan al comando y reutilizarlos.

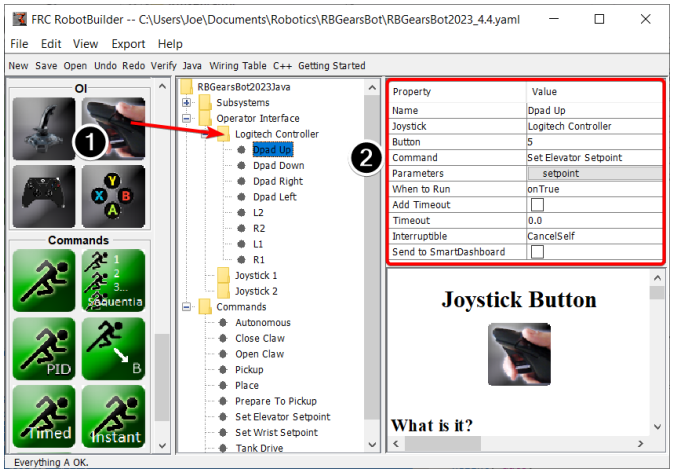
Añadiendo el Joystick al Programa del Robot.



Añadir el joystick al programa del robot

1. Arrastre el joystick a la carpeta de la Interfaz del Operador en el programa del robot.
2. Nombre el joystick para que refleje el uso del mismo y establezca el número del puerto USB.

Vincular un Botón al Comando «Move Elevator»



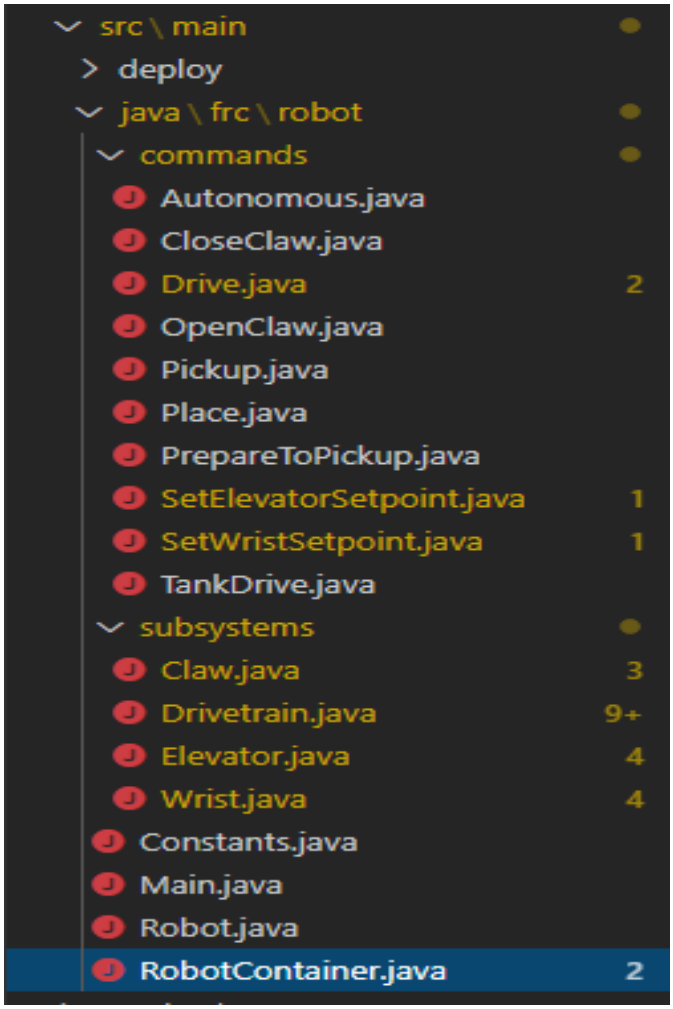
Añada el botón que debe ser presionado al programa

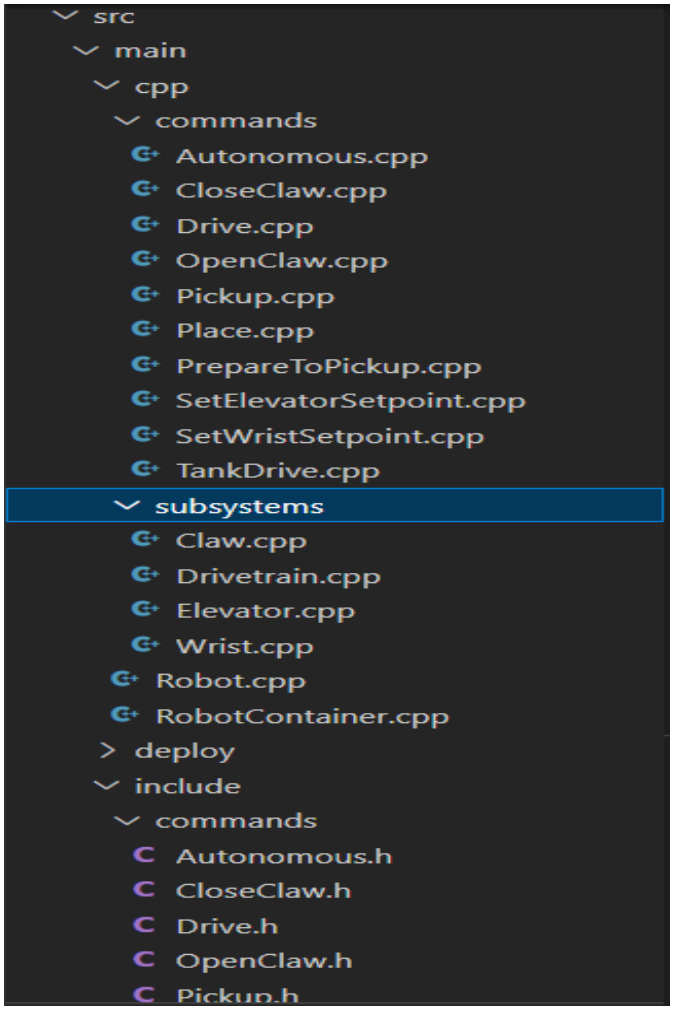
1. Arrastre el botón del joystick al Joystick (controlador Logitech) para que quede debajo del joystick
2. Set the properties for the button: the button number, the command to run when the button is pressed, parameters the command takes, and the *When to run* property to *onTrue* to indicate that the command should run whenever the joystick button is pressed.

Nota: Los botones del joystick deben ser arrastrados a (debajo de) un joystick en la carpeta de la Interfaz del Operador antes de añadir botones.

21.1.10 Código Creado con RobotBuilder

El Diseño de un Proyecto Generado de RobotBuilder.





Un proyecto generado por RobotBuilder consta de un paquete (en Java) o una carpeta (en C++) para los comandos y otra para los subsistemas. Cada comando u objeto de subsistema se almacena bajo esos contenedores. En el nivel superior del proyecto se encuentra el programa principal del robot (RobotContainer.java/C++).

Para más información sobre la organización de un robot basado en comandos, consulte *Estructuración de un proyecto de robot basado en comandos*

Código Autogenerado

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);
```

Cuando la descripción del robot es modificada y el código es re-exportado, RobotBuilder está diseñado para no modificar ningún cambio que haya hecho en el archivo, preservando así tu código. Esto hace de RobotBuilder una herramienta de ciclo de vida completo. Para saber qué código está bien para ser modificado por RobotBuilder, genera secciones que potencialmente tendrán que ser reescritas delimitadas con algunos comentarios especiales. Estos comentarios se muestran en el ejemplo anterior. No añada ningún código dentro de estos bloques de comentarios, se reescribirá la próxima vez que el proyecto se exporte desde RobotBuilder.

Si el código dentro de uno de estos bloques debe ser modificado, los comentarios pueden ser eliminados, pero esto evitará que se produzcan más actualizaciones más adelante. En el ejemplo anterior, si los comentarios //BEGIN y //END fueran removidos, y más tarde se añadió otro subsistema necesario en RobotBuilder, no se generaría en esa próxima exportación.

JAVA

```
// ROBOTBUILDER TYPE: Robot.
```

C++

```
// ROBOTBUILDER TYPE: Robot.
```

Additionally, each file has a comment defining the type of file. If this is modified or deleted, RobotBuilder will completely regenerate the file deleting any code added both inside and outside the AUTOGENERATED CODE blocks.

Programa Principal del Robot

Java

```
11 // ROBOTBUILDER TYPE: Robot.
12
13 package frc.robot;
14
15 import edu.wpi.first.hal.FRCNetComm.tInstances;
16 import edu.wpi.first.hal.FRCNetComm.tResourceType;
17 import edu.wpi.first.hal.HAL;
18 import edu.wpi.first.wpilibj.TimedRobot;
19 import edu.wpi.first.wpilibj2.command.Command;
20 import edu.wpi.first.wpilibj2.command.CommandScheduler;
21
22 /**
23  * The VM is configured to automatically run this class, and to call the
```

(continúe en la próxima página)

(proviene de la página anterior)

```
 * 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() {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
73      }
74
75      /**
76      * This autonomous runs the autonomous command selected by your {@link␣
↪RobotContainer} class.
77      */
78      @Override
79      public void autonomousInit() {
80          m_autonomousCommand = m_robotContainer.getAutonomousCommand();  // (3)
81
82          // schedule the autonomous command (example)
83          if (m_autonomousCommand != null) {
84              m_autonomousCommand.schedule();
85          }
86      }
87
88      /**
89      * This function is called periodically during autonomous.
90      */
91      @Override
92      public void autonomousPeriodic() {
93      }
94
95      @Override
96      public void teleopInit() {
97          // This makes sure that the autonomous stops running when
98          // teleop starts running. If you want the autonomous to
99          // continue until interrupted by another command, remove
100         // this line or comment it out.
101         if (m_autonomousCommand != null) {
102             m_autonomousCommand.cancel();
103         }
104     }
105
106     /**
107      * This function is called periodically during operator control.
108      */
109     @Override
110     public void teleopPeriodic() {
111     }
112
113     @Override
114     public void testInit() {
115         // Cancels all running commands at the start of test mode.
116         CommandScheduler.getInstance().cancelAll();
117     }
118
119     /**
120     * This function is called periodically during test mode.
121     */
122     @Override
123     public void testPeriodic() {
124     }
125
126 }
```

C++ (Encabezado)

```
// 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++ (Fuente)

```
// 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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
 */
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
```

Este es el programa principal generado por RobotBuilder. Hay un número de partes en este programa (secciones resaltadas):

1. Esta clase extiende TimedRobot. TimedRobot llamará a los métodos `autonomousPeriodic()` y `teleopPeriodic()` cada 20 ms.
2. En el método robotPeriodic que se llama cada 20ms, hacer un pase de horario.
3. El comando autónomo proporcionado está programado al inicio del autónomo en el método `autonomousInit()` y se cancela al final del periodo autónomo en `teleopInit()`.

RobotContainer

Java

```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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  */
  private RobotContainer() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
    // Smartdashboard Subsystems
    SmartDashboard.putData(m_wrist); // (6)
    SmartDashboard.putData(m_elevator);
    SmartDashboard.putData(m_claw);
    SmartDashboard.putData(m_drivetrain);


    // SmartDashboard Buttons
    SmartDashboard.putData("Close Claw", new CloseClaw( m_claw )); // (6)
    SmartDashboard.putData("Open Claw: OpenTime", new OpenClaw(1.0, m_claw));
    SmartDashboard.putData("Pickup", new Pickup());
    SmartDashboard.putData("Place", new Place());
    SmartDashboard.putData("Prepare To Pickup", new PrepareToPickup());
    SmartDashboard.putData("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0,
↪ m_elevator));
    SmartDashboard.putData("Set Elevator Setpoint: Platform", new␣
↪SetElevatorSetpoint(0.2, m_elevator));
    SmartDashboard.putData("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3,␣
↪m_elevator));
    SmartDashboard.putData("Set Wrist Setpoint: Horizontal", new SetWristSetpoint(0,␣
↪m_wrist));
    SmartDashboard.putData("Set Wrist Setpoint: Raise Wrist", new SetWristSetpoint(-
↪45, m_wrist));
    SmartDashboard.putData("Drive: Straight3Meters", new Drive(3, 0, m_drivetrain));
    SmartDashboard.putData("Drive: Place", new Drive(Drivetrain.PlaceDistance,␣
↪Drivetrain.BackAwayDistance, m_drivetrain));

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
    // Configure the button bindings
    configureButtonBindings();

    // Configure default commands
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SUBSYSTEM_DEFAULT_COMMAND
    m_drivetrain.setDefaultCommand(new TankDrive(() -> getJoystick1().getY(), () ->␣
↪getJoystick2().getY(), m_drivetrain)); // (5)


    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SUBSYSTEM_DEFAULT_COMMAND

    // Configure autonomous sendable chooser
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS

    m_chooser.addOption("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0, m_
↪elevator));
    m_chooser.addOption("Set Elevator Setpoint: Platform", new SetElevatorSetpoint(0.
↪2, m_elevator));
    m_chooser.addOption("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3, m_
↪elevator));
    m_chooser.setDefaultOption("Autonomous", new Autonomous()); // (2)

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS

    SmartDashboard.putData("Auto Mode", m_chooser);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  }

  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;
    }

```

(continúe en la próxima página)

(proviene de la página anterior)

```
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++ (Encabezado)

```
// 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

```

(continúe en la próxima página)

(proviene de la página anterior)

```
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++ (Fuente)

```
// 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(
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR
){



    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
    // Smartdashboard Subsystems
    frc::SmartDashboard::PutData(&m_drivetrain);
    frc::SmartDashboard::PutData(&m_claw);
    frc::SmartDashboard::PutData(&m_elevator);
    frc::SmartDashboard::PutData(&m_wrist);


    // SmartDashboard Buttons
    frc::SmartDashboard::PutData("Drive: Straight3Meters", new Drive(3, 0, &m_
↪drivetrain)); // (6)
    frc::SmartDashboard::PutData("Drive: Place", new Drive(Drivetrain::PlaceDistance,␣
↪Drivetrain::BackAwayDistance, &m_drivetrain));
    frc::SmartDashboard::PutData("Set Wrist Setpoint: Horizontal", new␣
↪SetWristSetpoint(0, &m_wrist));
    frc::SmartDashboard::PutData("Set Wrist Setpoint: Raise Wrist", new␣
↪SetWristSetpoint(-45, &m_wrist));
    frc::SmartDashboard::PutData("Set Elevator Setpoint: Bottom", new␣
↪SetElevatorSetpoint(0, &m_elevator));
    frc::SmartDashboard::PutData("Set Elevator Setpoint: Platform", new␣
↪SetElevatorSetpoint(0.2, &m_elevator));
    frc::SmartDashboard::PutData("Set Elevator Setpoint: Top", new␣
↪SetElevatorSetpoint(0.3, &m_elevator));
    frc::SmartDashboard::PutData("Prepare To Pickup", new PrepareToPickup());
    frc::SmartDashboard::PutData("Place", new Place());
    frc::SmartDashboard::PutData("Pickup", new Pickup());
    frc::SmartDashboard::PutData("Open Claw: OpenTime", new OpenClaw(1.0_s, &m_claw));
    frc::SmartDashboard::PutData("Close Claw", new CloseClaw( &m_claw ));

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD

    ConfigureButtonBindings();

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT-COMMANDS
m_drivetrain.SetDefaultCommand(TankDrive([this] {return getJoystick1()->GetY();},␣
↪[this] {return getJoystick2()->GetY();}, &m_drivetrain)); // (5)

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT-COMMANDS

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS

    m_chooser.AddOption("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0, &
↪m_elevator));
    m_chooser.AddOption("Set Elevator Setpoint: Platform", new SetElevatorSetpoint(0.
↪2, &m_elevator));
    m_chooser.AddOption("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3, &m_
↪elevator));

    m_chooser.SetDefaultOption("Autonomous", new Autonomous()); // (2)
```

(continúe en la próxima página)

(proviene de la página anterior)

```
67
68      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
69
70      frc::SmartDashboard::PutData("Auto Mode", &m_chooser);
71
72  }
73
74  RobotContainer* RobotContainer::GetInstance() {
75      if (m_robotContainer == NULL) {
76          m_robotContainer = new RobotContainer();
77      }
78      return(m_robotContainer);
79  }
80
81  void RobotContainer::ConfigureButtonBindings() {
82      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=BUTTONS
83
84  frc2::JoystickButton m_dpadUp{&m_logitechController, 5}; // (4)
85  frc2::JoystickButton m_dpadDown{&m_logitechController, 7};
86  frc2::JoystickButton m_dpadRight{&m_logitechController, 6};
87  frc2::JoystickButton m_dpadLeft{&m_logitechController, 8};
88  frc2::JoystickButton m_l2{&m_logitechController, 9};
89  frc2::JoystickButton m_r2{&m_logitechController, 10};
90  frc2::JoystickButton m_l1{&m_logitechController, 11};
91  frc2::JoystickButton m_r1{&m_logitechController, 12};
92
93  m_dpadUp.OnTrue(SetElevatorSetpoint(0.3, &m_elevator).
     ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));
94
95  m_dpadDown.OnTrue(SetElevatorSetpoint(0, &m_elevator).
     ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));
96
97  m_dpadRight.OnTrue(CloseClaw( &m_claw ).
     ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));
98
99  m_dpadLeft.OnTrue(OpenClaw(1.0_s, &m_claw).
     ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));
100
101 m_l2.OnTrue(PrepareToPickup().
     ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));
102
103 m_r2.OnTrue(Pickup().
     ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));
104
105 m_l1.OnTrue(Place().
     ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));
106
107 m_r1.OnTrue(Autonomous().
     ↪WithInterruptBehavior(frc2::Command::InterruptionBehavior::kCancelSelf));
108
109
110     // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=BUTTONS
111 }
112
113 // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=FUNCTIONS
114
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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();
}
```

Este es el RobotContainer generado por RobotBuilder que es donde se definen los subsistemas y la interfaz del operador. Este programa consta de varias partes (secciones resaltadas):

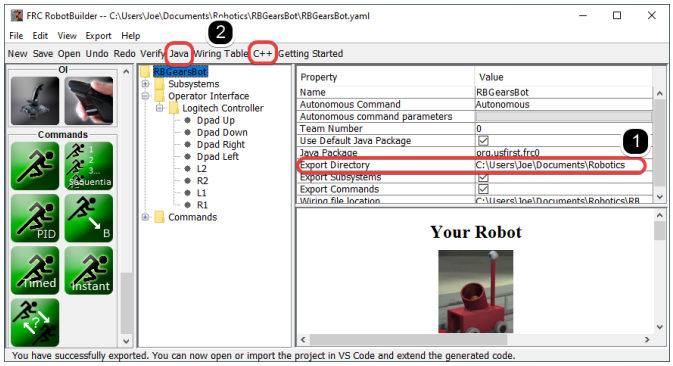
1. Aquí se declara cada uno de los subsistemas. Se pueden pasar como parámetros a cualquier comando que los requiera.
2. Si hay un comando autónomo proporcionado en las propiedades del robot de RobotBuilder, se añade al selector de envíos para ser seleccionado en el dashboard
3. Aquí se genera el código de todos los componentes de la interfaz del operador.
4. Además el código para enlazar los botones OI con los comandos que deben ejecutarse también se genera aquí.
5. Commands to be run on a subsystem when no other commands are running are defined here.
6. Aquí se definen los comandos que se ejecutan a través de un dashboard.

21.2 RobotBuilder - Escribiendo el Código

21.2.1 Generando el Código para un Proyecto

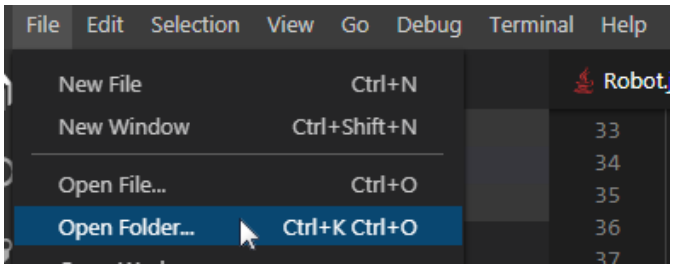
Después de configurar la estructura de su robot en RobotBuilder, tendrá que exportar el código y cargarlo en el código de Visual Studio Code. Este artículo describe el proceso para hacerlo.

Generar el Código para el Proyecto



Verifique que el Directorio de Exportación apunte a donde quiera usted (1) y luego haga clic en Java o C++ (2) para generar un proyecto de VS Code o actualizar el código.

Abrir el Proyecto en Visual Studio Code

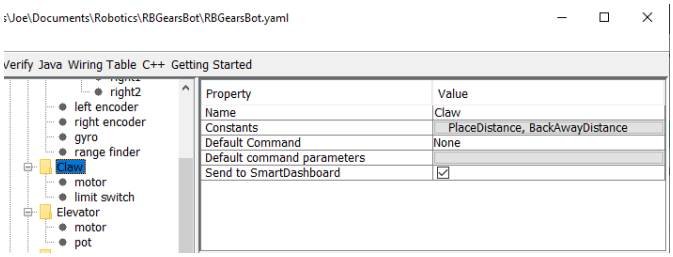


Abra VS Code y seleccione **File -> Open Folder**. Navegue a la ubicación de exportación y haga clic en **Select Folder**.

21.2.2 Escribiendo el Código para un Subsistema

Añadir código para crear un subsistema que funcione es muy sencillo. Para los subsistemas simples que no usan retroalimentación resulta ser extremadamente sencillo. En esta sección veremos un ejemplo de un subsistema *Claw*. El subsistema *Claw* también tiene un limit switch para determinar si un objeto está en el agarre.

Representación de RobotBuilder del Subsistema de la Garra



La garra en el extremo de un brazo robótico es un subsistema operado por un solo controlador de motor VictorSPX. Hay tres cosas que queremos que haga el motor, empezar a abrirse,

empezar a cerrarse y dejar de moverse. Esta es la responsabilidad del subsistema. El tiempo de apertura y cierre será manejado por un comando más adelante en este tutorial. También definiremos un método para saber si la garra está agarrando un objeto.

Añadiendo Capacidades del Subsistema

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);



    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    }

    @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 when in simulation

    }

    public void open() {
        motor.set(1.0);
    }

    public void close() {
        motor.set(-1.0);
    }

    public void stop() {
        motor.set(0.0);
    }

    public boolean isGripping() {
        return limitswitch.get();
    }

}
```

C++

```
// ROBOTBUILDER TYPE: Subsystem.

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES
#include "subsystems/Claw.h"
#include <frc/smartdashboard/SmartDashboard.h>

// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=INCLUDES

Claw::Claw(){
    SetName("Claw");
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    SetSubsystem("Claw");

 AddChild("limit switch", &m_limitswitch);


 AddChild("motor", &m_motor);
 m_motor.SetInverted(false);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
```

(continúe en la próxima página)

(proviene de la página anterior)

```
}

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();
}
```

Añade métodos a `claw.java` o `claw.cpp` que abrirán, cerrarán y detendrán el movimiento de la garra y obtendrán el limit switch. Estos serán utilizados por los comandos que realmente operan la garra.

Nota: Se han eliminado los comentarios de este archivo para facilitar la visualización de los cambios en este documento.

Observa que las variables miembro `motor` y `limitswitch` son creadas por RobotBuilder para que puedan ser utilizadas en todo el subsistema. Cada uno de los elementos de la paleta arrastrados tendrá una variable miembro con el nombre dado en RobotBuilder.

Añadir las Declaraciones del Método al Archivo de Cabecera (Sólo C++)

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


};
```

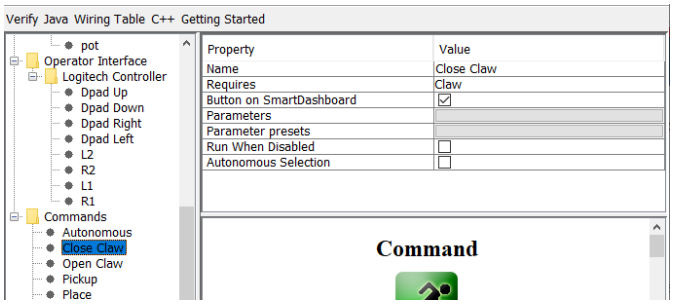
Además de añadir los métodos al fichero de implementación de la clase, `Claw.cpp`, es necesario añadir las declaraciones de los métodos al fichero de cabecera, `Claw.h`. Estas declaraciones que deben añadirse se muestran aquí.

Para añadir el comportamiento al subsistema de la garra para manejar la apertura y el cierre es necesario *definir comandos*.

21.2.3 Escribiendo el Código para un Comando

Las clases de subsistemas hacen que los mecanismos de su robot se muevan, pero para que se detenga en el momento adecuado y realice operaciones más complejas, debe escribir Comandos. Previamente en: writing the code for a subsystem 1 desarrollamos el código para el subsistema *Claw* en un robot para iniciar la apertura, cierre o parada de la garra. Ahora escribiremos el código para un comando que realmente hará funcionar el motor de la garra en el momento adecuado para que la garra se abra y se cierre. Nuestro ejemplo de garra es un mecanismo muy simple en el que hacemos funcionar el motor durante 1 segundo para abrirlo o hasta que se dispara el interruptor de límite para cerrarlo.

Comando de cierre de garras en RobotBuilder



Esta es la definición del comando *CloseClaw* en RobotBuilder. Observa que requiere el subsistema "claw". Esto se explica en el siguiente paso.

Clase CloseClaw generada

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
32
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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 generará los archivos de clase para el comando *CloseClaw*. El comando representa el comportamiento de la garra, es decir el funcionamiento en el tiempo. Para hacer funcionar este mecanismo de garra tan simple, el motor necesita funcionar en la dirección de cierre,. El subsistema *Claw* tiene métodos para poner en marcha el motor en la dirección correcta y para pararlo. La responsabilidad de los comandos es hacer funcionar el motor durante el tiempo correcto. Las líneas de código que se muestran en los recuadros se añaden para añadir este comportamiento.

1. Inicia el movimiento del motor de la garra en la dirección de cierre llamando al método `Close()` que fue añadido al subsistema *Claw* en el método *CloseClaw* Initialize.

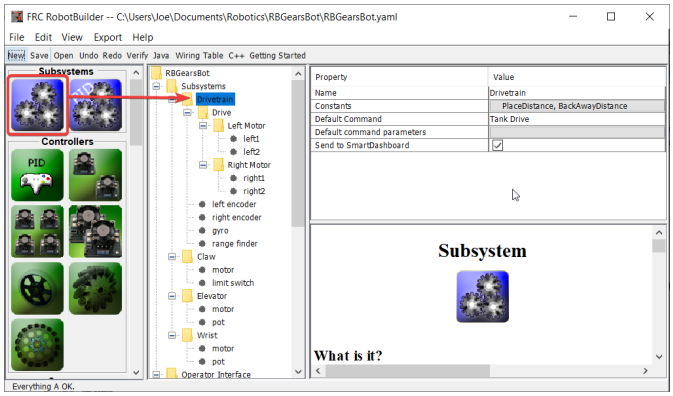
2. This command is finished when the limit switch in the *Claw* subsystem is tripped.
3. El método `End()` se llama cuando el comando ha terminado. En este caso, el motor está parado desde que el tiempo se ha agotado.

21.2.4 Manejando el Robot con Modo Tanque y Controles

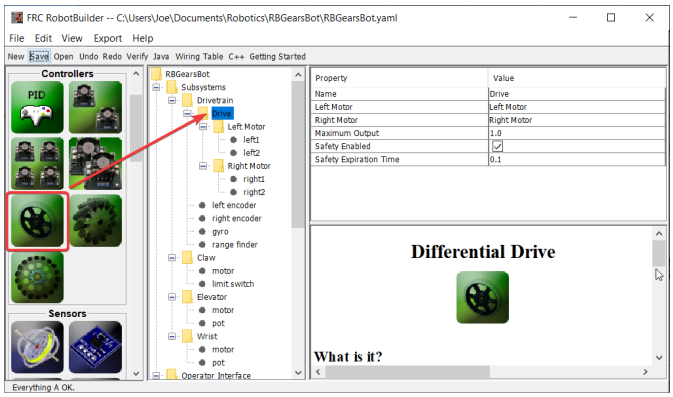
Un caso de uso común es tener un joystick que debería controlar algunos actuadores que forman parte de un subsistema. El problema es que el joystick se crea en la clase RobotContainer y los motores a controlar están en el subsistema. La idea es crear un comando que, cuando esté programado, lea la entrada del joystick y llame a un método que se crea en el subsistema que impulsa los motores.

En este ejemplo se muestra un subsistema de base de conducción que funciona en la conducción de un tanque usando un par de controles.

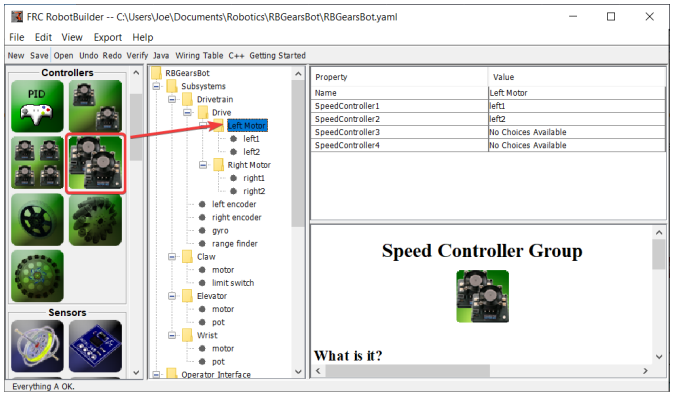
Crear un Subsistema del Chasis



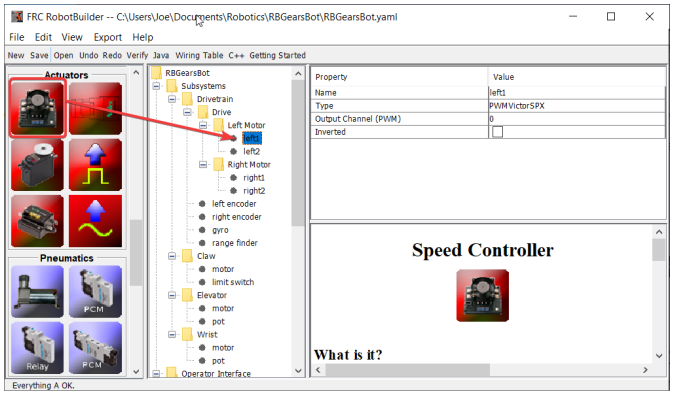
Cree un subsistema llamado Drive Train. Su responsabilidad será manejar la conducción de la base del robot.



Dentro del tren de transmisión/Drive Train, cree un objeto de transmisión diferencial para una transmisión de dos motores. Hay un motor izquierdo y un motor derecho como parte de la clase Differential Drive de transmisión diferencial.

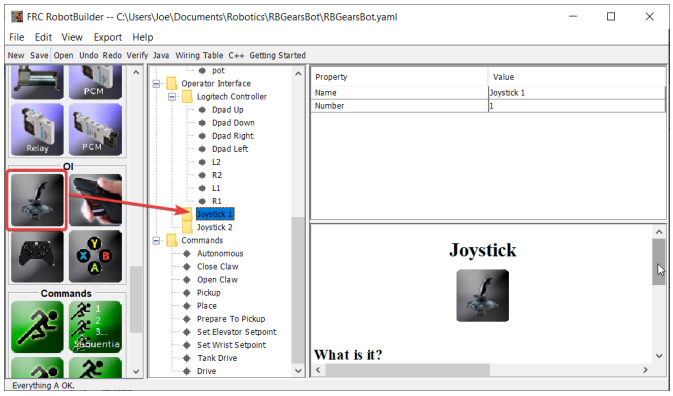


Since we want to use more then two motors to drive the robot, inside the Differential Drive, create two Motor Controller Groups. These will group multiple motor controllers so they can be used with Differential Drive.



Finally, create two Motor Controllers in each Motor Controller Group.

Agregar los controles a la Interfaz Operadora



Agregar dos controles a la Interfaz Operadora, uno es la palanca izquierda y el otro es la palanca derecha. El eje Y de los dos controles se usa para manejar los robots a la izquierda y a la derecha.

Nota: Asegúrese de exportar su programa a C++ o Java antes de continuar con el siguiente

paso.

Crear un método para escribir los motores en el subsistema

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

    /**
    *
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    */
    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() {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        // 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)

```
// 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};
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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);


```

(continúe en la próxima página)

(proviene de la página anterior)

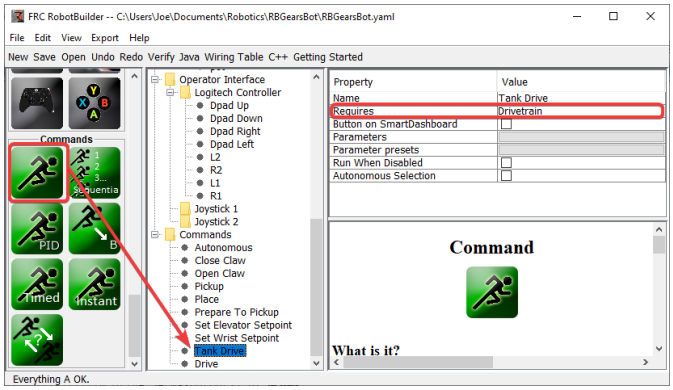
```
 AddChild("Right Motor", &m_rightMotor);


 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.

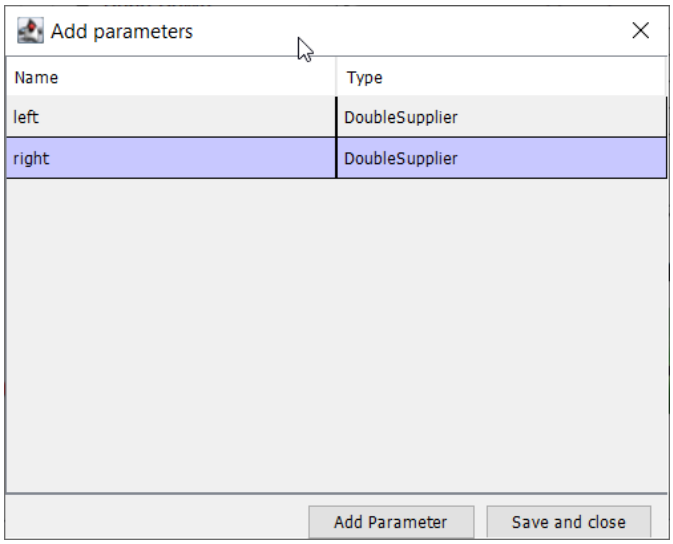
Nota: Se ha eliminado parte de la salida de RobotBuilder para este ejemplo para mayor claridad

---

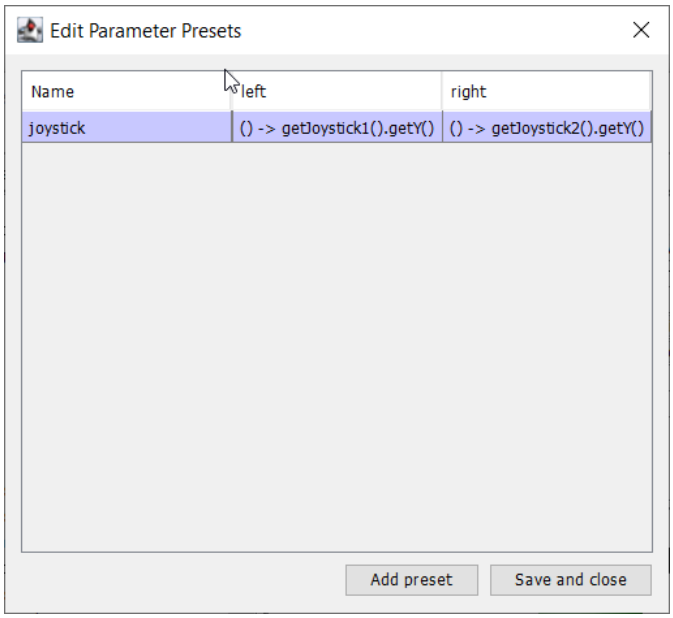
Leer los valores de las palancas y llamar los métodos del subsistema



Cree un comando, en este caso llamado Tank Drive. Su propósito será leer los valores del joystick y enviarlos al subsistema Drive Base. Tenga en cuenta que este comando requiere el subsistema Drive Train. Esto hará que deje de funcionar cada vez que cualquier otra cosa intente utilizar el tren de transmisión.



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();}`

Nota: Asegúrese de exportar su programa a C++ o Java antes de continuar con el siguiente paso.

Añada el Código para hacer la conducción

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
31
32      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
33
34
35      public TankDrive(Drivetrain subsystem) {
36
37
38      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
39          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING
40
41      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING
42          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
43
44          m_drivetrain = subsystem;
45          addRequirements(m_drivetrain);
46
47      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
48      }
49
50      // Called when the command is initially scheduled.
51      @Override
52      public void initialize() {
53      }
54
55      // Called every time the scheduler runs while the command is scheduled.
56      @Override
57      public void execute() {
58          m_drivetrain.drive(m_left.getAsDouble(), m_right.getAsDouble());
59      }
60
61      // Called once the command ends or is interrupted.
62      @Override
63      public void end(boolean interrupted) {
64          m_drivetrain.drive(0.0, 0.0);
65      }
66
67      // Returns true when the command should end.
68      @Override
69      public boolean isFinished() {
70          return false;
71      }
72
73      @Override
74      public boolean runsWhenDisabled() {
75          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
76          return false;
77
78      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
79      }
80  }
```

C++ (Header)

```
// 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
};
```

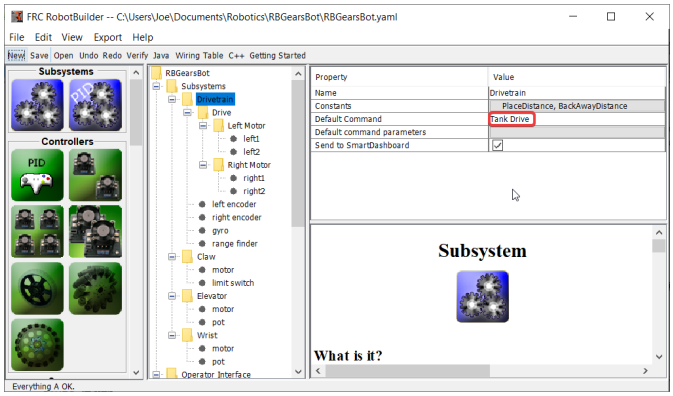
C++ (Source)

```
// 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
}
```

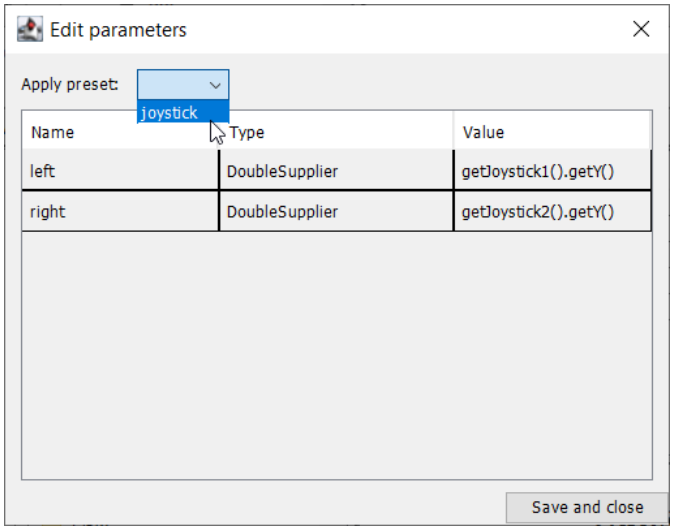
Add code to the execute method to do the actual driving. All that is needed is pass the for the left and right parameters to the Drive Train subsystem. The subsystem just uses them for the tank steering method on its DifferentialDrive object. And we get tank steering.

También completamos el método `end()` para que cuando este comando se interrumpa o se detenga, los motores se detendrán como medida de seguridad.

Hacer el Comando por defecto



El último paso es hacer que el comando Tank Drive sea el «Comando predeterminado» para el subsistema Drive Train. Esto significa que siempre que ningún otro comando esté usando Drive Train, los Joysticks tendrán el control. Este es probablemente el comportamiento deseable. Cuando el código autónomo se está ejecutando, también requerirá el tren de transmisión e interrumpirá el comando Tank Drive. Cuando finalice el código autónomo, el comando DriveWithJoysticks se reiniciará automáticamente (porque es el comando predeterminado) y los operadores volverán a tener el control. Si escribe cualquier código que haga teleoperación automática, esos comandos también deberían «requerir» el DriveTrain para que también interrumpan el comando Tank Drive y tengan control total.



The final step is to choose the joystick parameter preset previously set up.

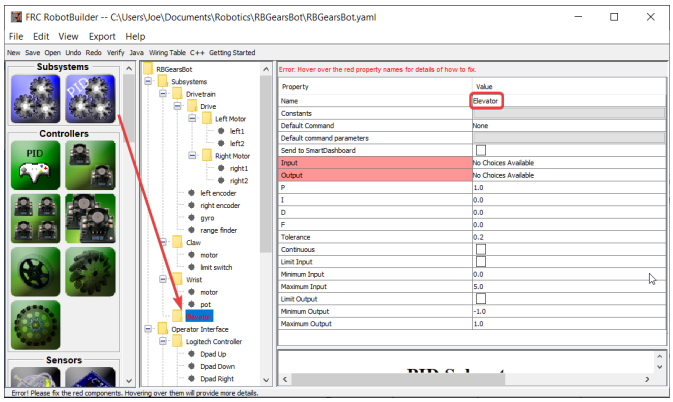
Nota: Asegúrese de exportar su programa a C++ o Java antes de continuar.

21.3 RobotBuilder - Avanzado

21.3.1 Uso del subsistema PID para controlar actuadores

More advanced subsystems will use sensors for feedback to get guaranteed results for operations like setting elevator heights or wrist angles. PIDSubsystems use feedback to control the actuator and drive it to a particular position. In this example we use an elevator with a 10-turn potentiometer connected to it to give feedback on the height. The PIDSubsystem has a built-in PIDController to automatically control the mechanism to the correct setpoints.

Crear un subsistema PID

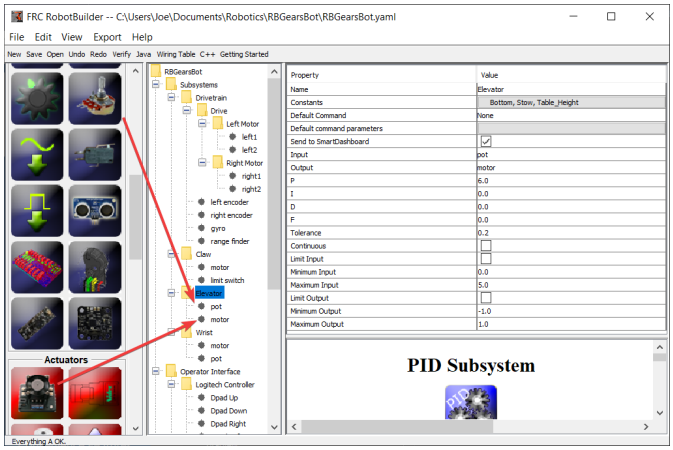


Crear un subsistema que use retroalimentación para controlar la posición o velocidad de un mecanismo es muy fácil.

1. Arrastre un subsistema PID desde la paleta a la carpeta Subsistemas en la descripción del robot.
2. Cambiar el nombre del subsistema PID por un nombre más significativo para el subsistema, en este caso Elevator

Observe que algunas de las partes de la descripción del robot se han vuelto rojas. Esto indica que estos componentes (el subsistema PID) no se han completado y deben completarse. Las propiedades que faltan o son incorrectas se muestran en rojo.

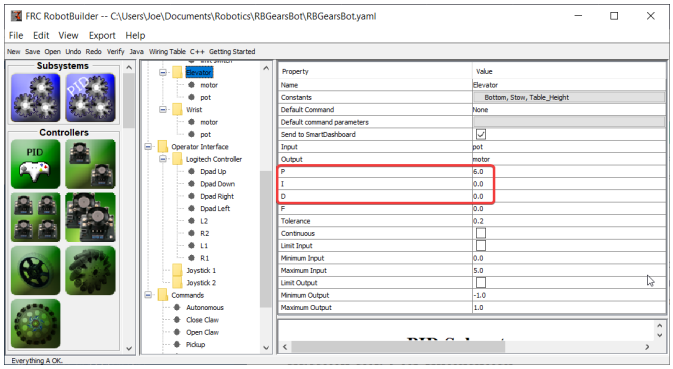
Adición de sensores y actuadores al subsistema PID



Agregue los componentes que faltan para el subsistema PID

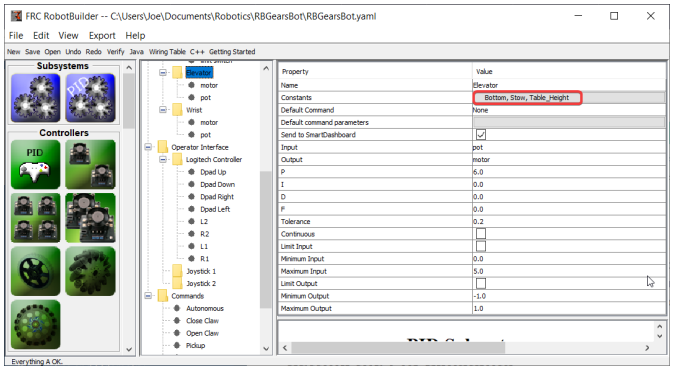
1. Arrastrar el actuador (un controlador de motor) al subsistema particular - en este caso el Elevador
2. Arrastre el sensor que se utilizará para la retroalimentación al subsistema, en este caso el sensor es un potenciómetro que podría dar retroalimentación de la altura del ascensor.

Rellenar los parámetros PID

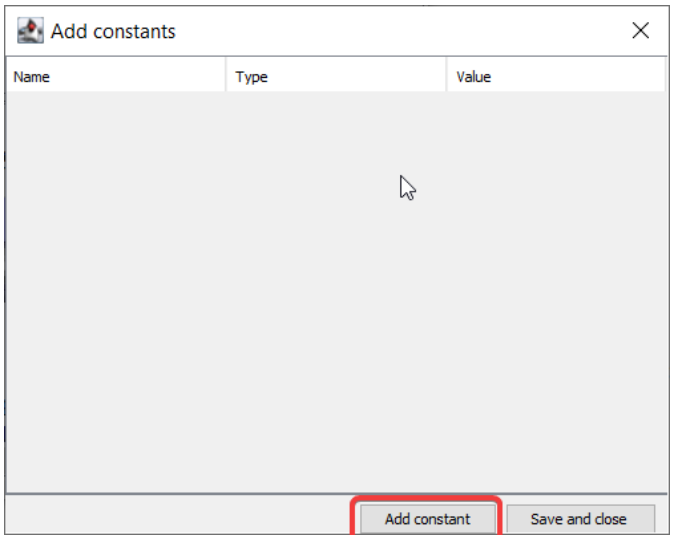


Los valores P, I y D deben ser rellenados para obtener la sensibilidad y estabilidad deseada del componente. En el caso de nuestro ascensor utilizamos una constante proporcional de 6,0 y 0 para los términos I y D.

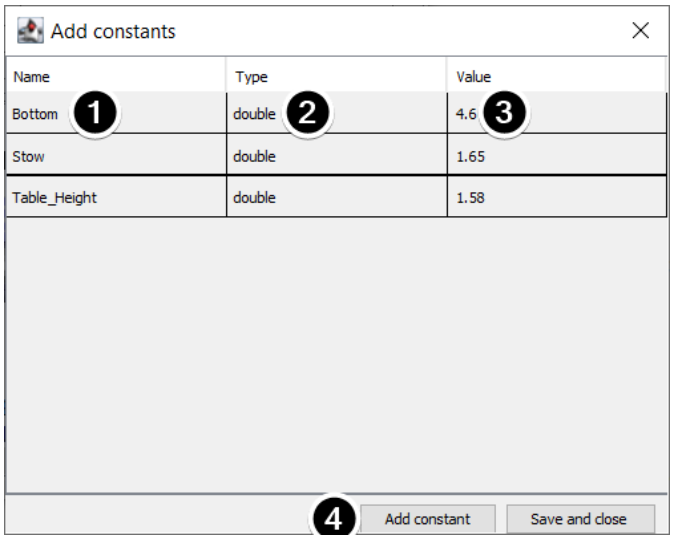
Crear Constantes de Setpoint



Para facilitar la gestión de los puntos de ajuste del ascensor, crearemos constantes para gestionar los puntos de ajuste. Haga clic en el cuadro de constantes para que aparezca el diálogo de constantes.



Haga clic en el botón *add constant*



1. Ponga un nombre para la constante, en este caso: Bottom
2. Seleccione un tipo de constante en el menú desplegable, en este caso: double
3. Seleccione un valor para la constante, en este caso: 4.65
4. Haga clic en *add constant* para seguir agregando cosntantes
5. After entering all constants, Click *Save and close*

21.3.2 Escribiendo el Código para un Subsistema PID

The skeleton of the PIDSubsystem is generated by the RobotBuilder and we have to fill in the rest of the code to provide the potentiometer value and drive the motor with the output of the embedded PIDController.

Asegúrese de que el subsistema Elevator PID se haya creado en RobotBuilder. Una vez que esté todo configurado, genere código Java/C++ para el proyecto usando el menú Exportar o el menú de la barra de herramientas Java/C++.

RobotBuilder generates the PIDSubsystem methods such that no additional code is needed for basic operation.

Establecer las Constantes PID

The height constants and PID constants are automatically generated.

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
```

Get Potentiometer Measurement

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
}
```

The `getMeasurement()` method is used to set the value of the sensor that is providing the feedback for the PID controller. In this case, the code is automatically generated and returns the potentiometer voltage as returned by the `get()` method.

Calculate PID Output

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
}
```

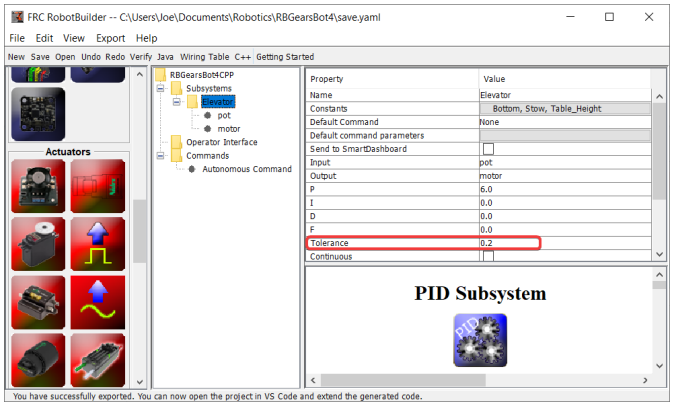
The `useOutput` method writes the calculated PID output directly to the motor.

Eso es todo lo que se requiere para crear un subsistema PID del elevador.

21.3.3 Comando de punto de ajuste

A Setpoint Command works in conjunction with a PIDSubsystem to drive an actuator to a particular angle or position that is measured using a potentiometer or encoder. This happens so often that there is a shortcut in RobotBuilder to do this task.

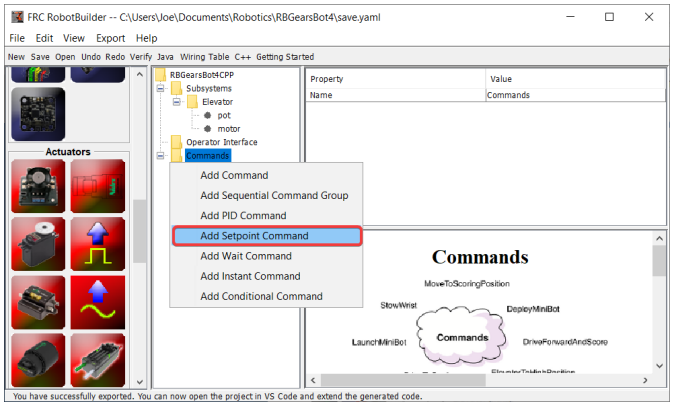
Comience con un subsistema PIDS



Suponga que en un robot hay una articulación de muñeca con un potenciómetro que mide el ángulo. Primero *cree un subsitema* que incluya el motor que mueve la articulación de la muñeca y el potenciómetro que mide el ángulo. El subsistema PIDSubsystem debe tener todas las constantes PID completadas y funcionando correctamente.

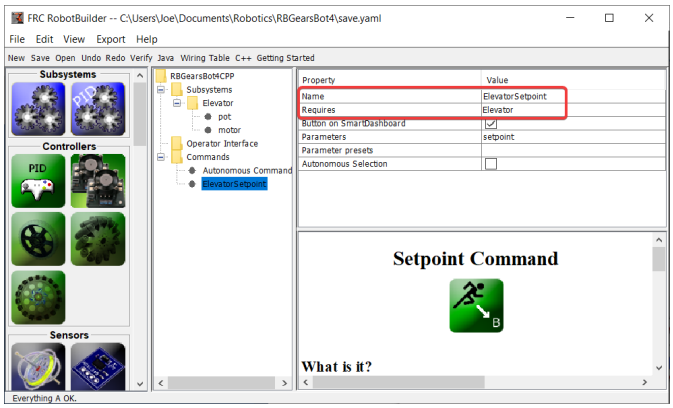
Es importante configurar el parámetro **Tolerancia**. Esto controla qué tan lejos puede estar el valor actual del punto de ajuste y considerarse dentro del objetivo. Este es el criterio que utiliza SetpointCommand para pasar al siguiente comando.

Creación del comando de punto de ajuste

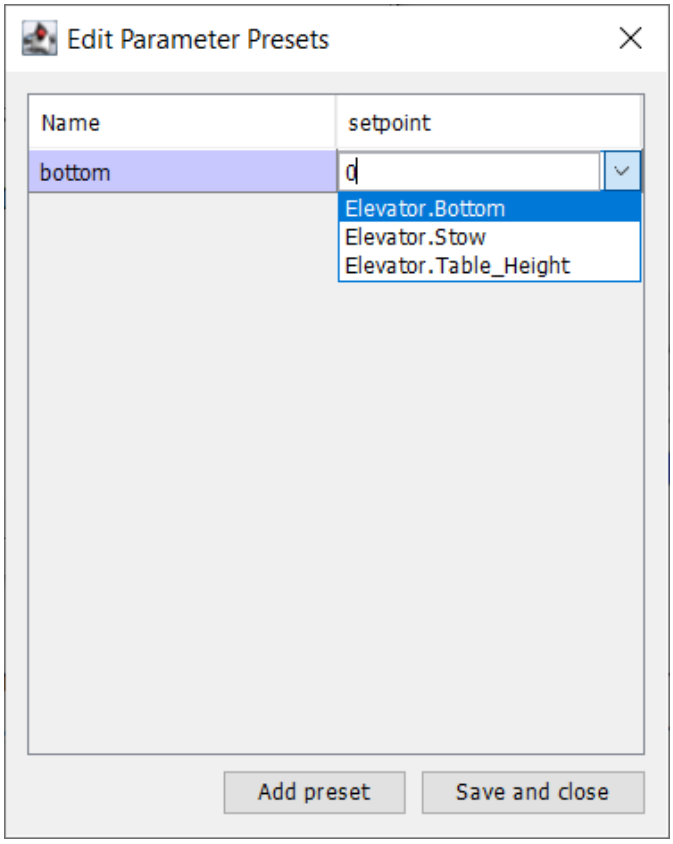


Haga clic derecho en la carpeta Comandos de la paleta y seleccione «Agregar comando de punto de ajuste».

Parámetros de comando de punto de ajuste



Fill in the name of the new command. The Requires field is the PIDSubsystem that is being driven to a setpoint, in this case the Elevator subsystem.



1. Click on the Parameter Presets to set up the setpoints.
2. Select *Add Preset*
3. Enter a preset name (in this case “bottom”
4. Click the dropdown next to the setpoint entry box
5. Select the Elevator.Bottom constant, that was created in the Elevator subsystem previously
6. Repeat steps 2-5 for the other setpoints.
7. Click *Save and close*

There is no need to fill in any code for this command, it is automatically created by RobotBuilder.

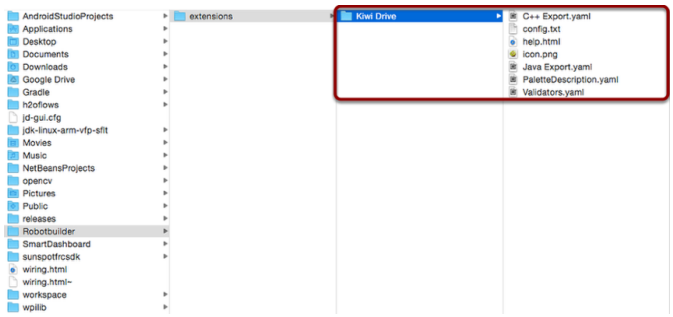
Siempre que se programe este comando, conducirá automáticamente el subsistema al punto de ajuste especificado. Cuando se alcanza el punto de ajuste dentro de la tolerancia especificada en el subsistema PIDS, el comando finaliza y comienza el siguiente comando. Es importante especificar una tolerancia en el subsistema PIDS o es posible que este comando nunca finalice porque no se logra la tolerancia.

Nota: Para obtener más información sobre el control PID, consulte *Advanced Controls Introduction*.

21.3.4 Agregar componentes personalizados

RobotBuilder funciona muy bien para crear programas de robots que solo usan WPILib para motores, controladores y sensores. Pero para los equipos que usan clases personalizadas, RobotBuilder no tiene ningún soporte para esas clases, por lo que se deben seguir algunos pasos para usarlas en RobotBuilder.

Estructura de componentes personalizada



Todos los componentes personalizados van en `~/wpilib/YYYY/Robotbuilder/extensions` donde ~ es `C:\Users\Public` en Windows y YYYY es el año de FRC®.

Hay siete archivos y una carpeta que se necesitan para un componente personalizado. La carpeta contiene los archivos que describen el componente y cómo exportarlo. Debe tener el mismo nombre que el componente (por ejemplo, «Kiwi Drive» para un controlador de unidad de kiwi, «Robot Drive 6» para un controlador de unidad de seis motores, etc.). Los archivos deben tener los mismos nombres y extensiones que los que se muestran aquí. Otros archivos pueden estar en la carpeta junto con estos siete, pero los siete deben estar presentes para que RobotBuilder reconozca el componente personalizado.

PaletteDescription.yaml

PaletteDescription.yaml × Validators.yaml × C++ Export.yaml

```
!Component
  name: Kiwi Drive
  type: Controller
  supports: {PIDOutput: 3}
  help: A type of drivetrain with three omni wheels
  properties:
    - !ChildSelectionProperty
      name: Motor 1
      type: PIDOutput
      validators: [KiwiDriveValidator, ChildDropdownSelected]
    - !ChildSelectionProperty
      name: Motor 2
      type: PIDOutput
      validators: [KiwiDriveValidator, ChildDropdownSelected]
    - !ChildSelectionProperty
      name: Motor 3
      type: PIDOutput
      validators: [KiwiDriveValidator, ChildDropdownSelected]
```

Linea por linea:

- !Component: declara el comienzo de un nuevo componente
- name: El nombre del componente. Esto es lo que se mostrará en la paleta/árbol; también debe ser el mismo que el nombre de la carpeta que lo contiene.

- type: el tipo de componente (se explicarán en profundidad más adelante)
- supports: un mapa de la cantidad de cada tipo de componente que puede soportar. Los controladores de motor en RobotBuilder son todos PIDOutputs, por lo que una unidad de kiwi puede admitir tres PIDOutputs. Si un componente no admite nada (como sensores o controladores de motor), simplemente deje esta línea fuera
- help: una cadena corta que da un mensaje útil cuando se coloca el cursor sobre uno de estos componentes
- properties: una lista de las propiedades de este componente. En este ejemplo de unidad de kiwi, hay tres propiedades muy similares, una para cada motor. Una ChildSelection-Property permite al usuario elegir un componente del tipo dado de los subcomponentes del que se está editando (por lo que aquí, mostrarían un menú desplegable solicitando una PIDOutput, es decir, un controlador de motor, que se ha agregado a la unidad de kiwi)

Los tipos de componentes que admite RobotBuilder (distinguen entre mayúsculas y minúsculas):

- Comando
- Subsistema
- PIDOutput (controlador de motor)
- PIDSource (sensor que implementa PIDSource, por ejemplo, potenciómetro analógico, encoder)
- Sensor (sensor que no implementa PIDSource, por ejemplo, interruptor de límite)
- Controlador (unidad de robot, controlador PID, etc.)
- Actuador (una salida que no es un motor, por ejemplo, solenoide, servo)
- Joystick
- Botón de joystick

Propiedades

Las propiedades relevantes para un componente personalizado:

- StringProperty: se utiliza cuando un componente necesita una cadena, p. Ej. el nombre del componente
- BooleanProperty: se usa cuando un componente necesita un valor booleano, p. Ej. poner un botón en el SmartDashboard
- DoubleProperty: se utiliza cuando un componente necesita un valor numérico, p. Ej. Constantes PIDChoicesProperty
- ChildSelectionProperty: se usa cuando necesita elegir un componente secundario, p. ej. controladores de motor en un RobotDrive
- TypeSelectionProperty: se usa cuando necesita elegir cualquier componente del tipo dado desde cualquier lugar del programa, p. entrada y salida para un comando PID

Los campos de cada propiedad se describen a continuación:

```
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.
```

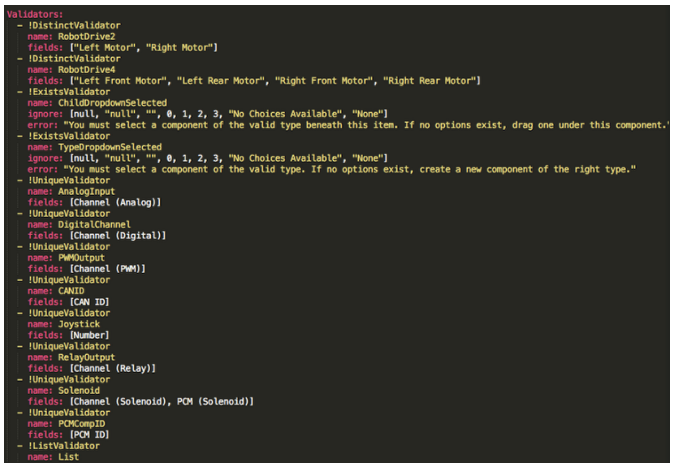
Validators.yaml

Validators.yaml

```
1  !DistinctValidator
2    name: KiwiDriveValidator
3    fields: ["Motor 1", "Motor 2", "Motor 3"]
```

Es posible que haya notado «KiwiDriveValidator» en la entrada de los validadores de cada una de las propiedades del motor en PaletteDescription.yaml. No es un validador integrado, por lo que tuvo que definirse en Validators.yaml. Este validador de ejemplo es muy simple: solo se asegura de que cada uno de los campos nombrados tenga un valor diferente al de los demás.

Validadores integrados y tipos de validador



Los validadores integrados son muy útiles (especialmente los UniqueValidators para uso de puertos / canales), pero a veces se necesita un validador personalizado, como en el paso anterior

- DistinctValidator: se asegura de que los valores de cada uno de los campos dados sean únicos
- ExistsValidator: se asegura de que se haya establecido un valor para la propiedad usando este validador
- UniqueValidator: se asegura de que el valor de la propiedad sea único globalmente para los campos dados
- ListValidator: se asegura de que todos los valores de una propiedad de lista sean válidos

C++ Export.yaml

Un desglose línea por línea del archivo:

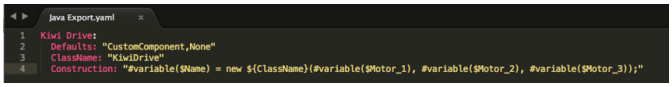
- Kiwi Drive: el nombre del componente que se exporta. Este es el mismo que el nombre establecido en PaletteDescription.yaml y el nombre de la carpeta que contiene este archivo
- Valores predeterminados: proporciona algunos valores predeterminados para las inclusiones que necesita este componente, el nombre de la clase, una plantilla de construcción y más. El CustomComponent predeterminado agrega una inclusión para``Custom/${ClassName}.h`` a cada archivo generado que usa el componente (por ejemplo, `RobotDrive.h` tendría `#include "Custom/KiwiDrive.h` la parte superior del archivo)
- ClassName: el nombre de la clase personalizada que está agregando.
- Construcción: una instrucción sobre cómo se debe construir el componente. Las variables se reemplazarán con sus valores («${ClassName}» se reemplazará con «KiwiDrive»), luego se evaluarán las macros (por ejemplo, `#variable($Name)` se puede reemplazar con `drivebaseKiwiDrive`).

Este ejemplo espera una clase KiwiDrive con el constructor

```
KiwiDrive(SpeedController, SpeedController, SpeedController)
```

Si su equipo usa Java, este archivo puede estar vacío.

Java Export.yaml



Muy similar al archivo de exportación de C++; la única diferencia debería ser la línea de Construcción. Este ejemplo espera una clase KiwiDrive con el constructor

```
KiwiDrive(SpeedController, SpeedController, SpeedController)
```

Si su equipo usa C++, este archivo puede estar vacío.

Usar macros y variables

Las macros son funciones simples que utiliza RobotBuilder para convertir variables en texto que se insertará en el código generado. Siempre comienzan con el símbolo «#» y tienen una sintaxis similar a las funciones: `<macro_name>( arg0, arg1, arg2, ...)`. La única macro que probablemente necesitará usar es `#variable( component_name )`

`#variable` toma una cadena, generalmente una variable definida en algún lugar (es decir, «Name» es el nombre que se le da al componente en RobotBuilder, como «Arm Motor»), y lo convierte en el nombre de una variable definida en el código generado. Por ejemplo, `#variable("Arm Motor")` da como resultado la cadena `ArmMotor`

Se hace referencia a las variables colocando un signo de dólar («$») delante del nombre de la variable, que opcionalmente se coloca entre llaves para distinguir fácilmente la variable de otro texto en el archivo. Cuando se analiza el archivo, el signo de dólar, el nombre de la variable y las llaves se reemplazan por el valor de la variable (por ejemplo, `${ClassName}` is replaced with `KiwiDrive`).

Las variables son propiedades de los componentes (por ejemplo, «Motor 1», «Motor 2», «Motor 3» en el ejemplo de la unidad de kiwi) o una de las siguientes:

1. Short_Name: el nombre que se le da al componente en el panel del editor en RobotBuilder
2. Name: el nombre completo del componente. Si el componente está en un subsistema, este será el nombre corto adjunto al nombre del subsistema
3. Export: el nombre del archivo en el que se debe crear este componente, si lo hubiera. Debe ser «RobotMap» para componentes como actuadores, controladores y sensores; o «OI» para cosas como gamepads u otros componentes personalizados de OI. Tenga en cuenta que el valor predeterminado «CustomComponent» se exportará a RobotMap.
4. Import: archivos que deben incluirse o importarse para que este componente pueda utilizarse.
5. Declaration: instrucción, similar a Construction, sobre cómo declarar una variable de este tipo de componente. Esto se soluciona con el valor predeterminado «None»
6. Construction: una instrucción sobre cómo crear una nueva instancia de este componente

7. LiveWindow: una instrucción sobre cómo agregar este componente a LiveWindow
8. Extra: instrucciones para cualquier función adicional o llamadas a métodos para que este componente se comporte correctamente, como los codificadores que necesitan establecer el tipo de codificación.
9. Prototype (solo C++): el prototipo de una función que se creará en el archivo en el que se declara el componente, generalmente un captador en la clase OI
10. Function: una función que se creará en el archivo en el que se declara el componente, normalmente un captador en la clase OI
11. PID: Una instrucción sobre cómo obtener la salida PID del componente, si tiene una (por ejemplo, `#variable($Short_Name)->PIDGet()`)
12. ClassName: el nombre de la clase que representa el componente (por ejemplo, `KiwiDrive` o `Joystick`)

Si tiene variables con espacios en el nombre (como «Motor 1», «Right Front Motor», etc.), los espacios deben reemplazarse por guiones bajos cuando los use en los archivos de exportación.

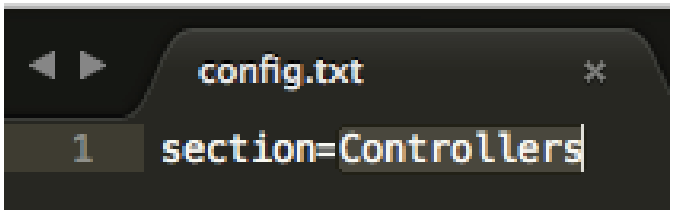
help.html

help.html

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <link rel="stylesheet" href="styles.css" type="text/css" media="screen" />
</head>

<body>
    <h1>Kiwi Drive</h1>
    <center><img src="icon.png" /></center>
    <h2>What is it?</h2>
    <p>
        Kiwi drive is a type of omni-directional drivetrain with three omni wheels,
        usually at 120° angles to each other.
    </p>
    <h2>Properties</h2>
    <dl>
        <dt>Motor 1</dt>
        <dd>The first motor</dd>
        <dt>Motor 2</dt>
        <dd>The second motor</dd>
        <dt>Motor 3</dt>
        <dd>The third motor</dd>
    </dl>
    <h2>See Also</h2>
    <ul>
        <li>
            <a href="http://en.wikipedia.org/wiki/Kiwi_drive">Kiwi drive on Wikipedia</a>
        </li>
    </ul>
</body>

</html>

```

Un archivo HTML que proporciona información sobre el componente. Es mejor que esto sea lo más detallado posible, aunque ciertamente no es necesario si los programadore(s) están lo suficientemente familiarizados con el componente, o si es tan simple que no tiene mucho sentido una descripción detallada.

config.txt



Un archivo de configuración para contener información diversa sobre el componente. Actualmente, esto solo tiene la sección de la paleta para colocar el componente.

Las secciones de la paleta (distinguen entre mayúsculas y minúsculas):

- Subsistemas
- Controladores
- Sensores
- Actuadores
- Neumática
- OI
- Comandos

icon.png

El icono que aparece en la paleta y la página de ayuda. Debe ser un archivo``.png`` de 64x64.

Debe usar la combinación de colores y el estilo general de la sección en la que se encuentra para evitar el desorden visual, pero esto es completamente opcional. Los archivos `` .psd “” de Photoshop de los iconos y fondos están en src/main/icons/icons y los archivos png de los iconos y fondos están en src/main/resources/icons.

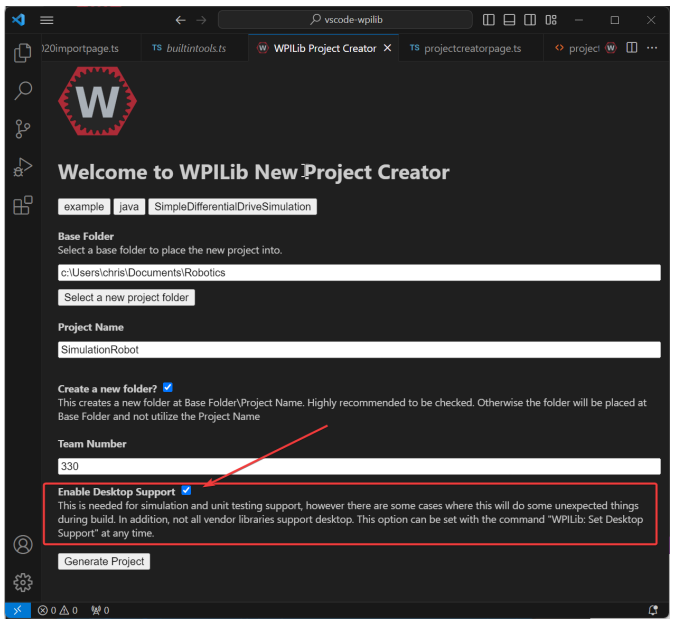
22 Simulación del robot

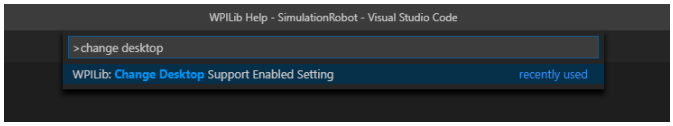
22.1 Introducción a la Simulación del Robot

A veces un equipo puede querer probar su código sin tener un robot disponible. WPILib proporciona a los equipos la capacidad de simular varias características del robot usando simples comandos gradle.

Java/C++

El uso del Simulador de Escritorio requiere que el Soporte de Escritorio esté habilitado. Esto se puede hacer marcando la casilla «Enable Desktop Support Checkbox» o Activar Soporte de Escritorio al crear su proyecto del robot o ejecutando «WPILib: Change Desktop Support Enabled Setting» desde la paleta de comandos de Visual Studio Code.





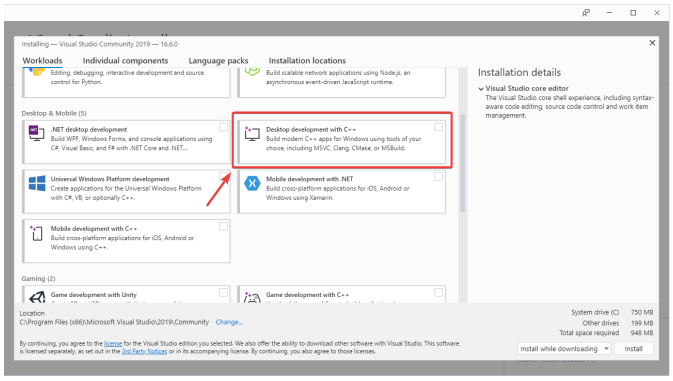
El soporte de escritorio también puede ser habilitado de forma manual editando su archivo `build.gradle` ubicado en la raíz del proyecto de su robot. Simplemente cambie `includeDesktopSupport = false` por `includeDesktopSupport = true`

Importante: Es importante señalar que la habilitación del soporte de escritorio/simulación puede tener consecuencias imprevistas. No todos los proveedores soportan esta opción, y el código que utiliza sus bibliotecas puede incluso bloquearse al intentar ejecutar la simulación.

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.

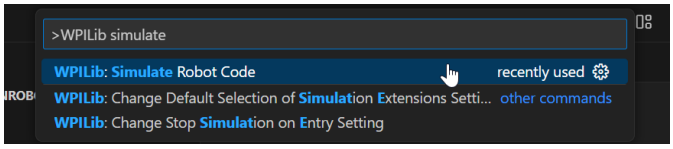
Nota: C++ robot simulation requires that a native compiler to be installed. For Windows, this would be Visual Studio 2022 version 17.9 or later (**not** VS Code), macOS requires Xcode 14 or later, and Linux (Ubuntu) requires the `build-essential` package.

Ensure the *Desktop Development with C++* option is checked in the Visual Studio installer for simulation support.



Running Robot Simulation

La simulación básica del robot puede ser ejecutada usando Visual Studio Code. Esto puede ser hecho sin usar ningún comando usando la paleta de comandos de Visual Studio Code.



La salida de información de la consola en Visual Studio Code debería verse como la siguiente. Sin embargo, los equipos probablemente querrán probar su código en vez de sólo ejecutar la simulación. Esto se puede hacer usando *WPILib's Simulation GUI*.

```
********** Robot program starting **********
Default disabledInit() method... Override me!
Default disabledPeriodic() method... Override me!
Default robotPeriodic() method... Override me!
```

Importante: Simulation can also be run outside of VS Code using `./gradlew simulateJava` for Java or `./gradlew simulateNative` for C++.

Nota: 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

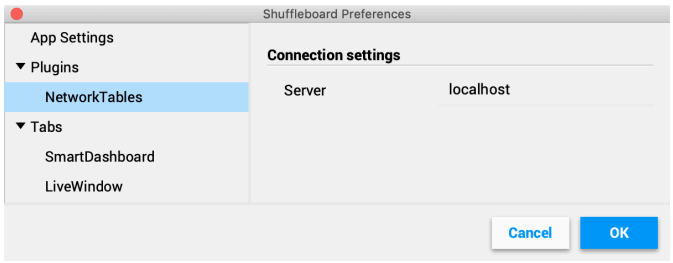
```
python3 -m robotpy sim
```

22.1.1 Ejecución de Dashboards del 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`).

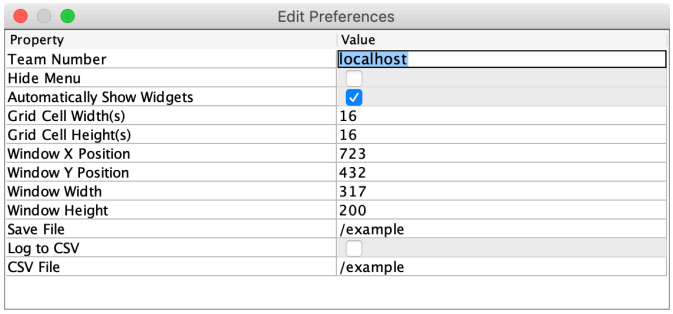
Shuffleboard

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`.



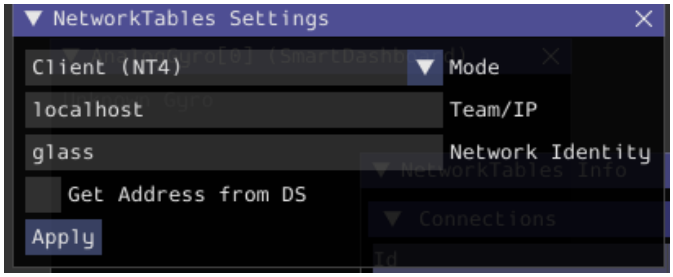
SmartDashboard

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`.



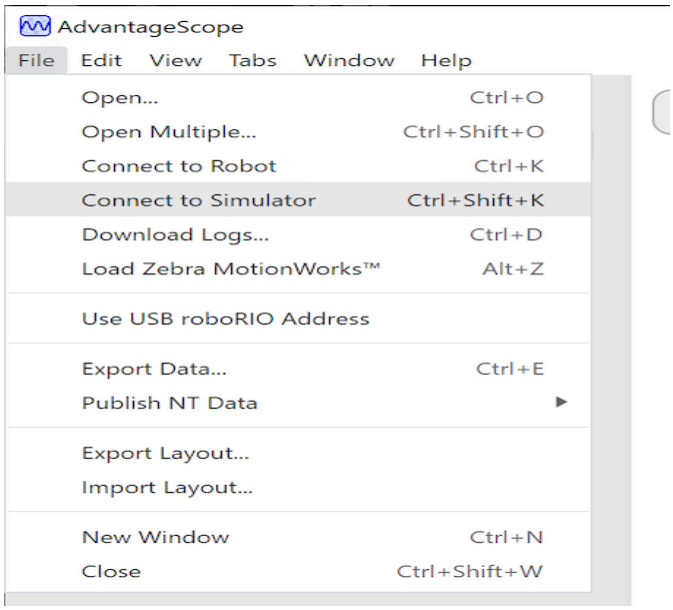
Glass

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`.

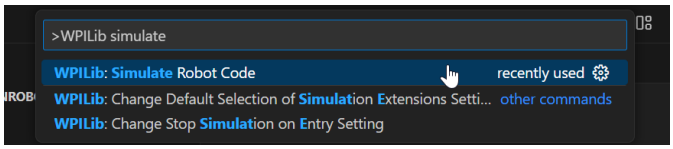


22.2 Elementos de la interfaz de usuario específicos de la simulación

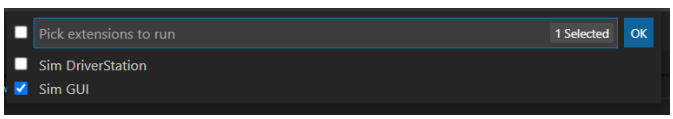
WPILib ha ampliado la simulación del robot para introducir una interfaz gráfica del usuario (GUI por sus sigla en inglés: Graphical User Interface). Esto permite a los equipos visualizar fácilmente las entradas y salidas de información de sus robots.

Nota: The Simulation GUI is very similar in many ways to *Glass*. Some of the following pages will link to Glass sections that describe elements common to both GUIs.

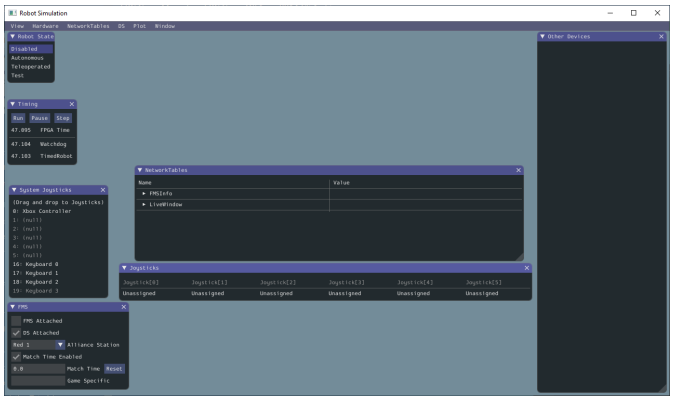
22.2.1 Ejecutando la GUI



Usted puede simplemente abrir la GUI a través de la opción **Run Simulation** de la paleta de comandos.

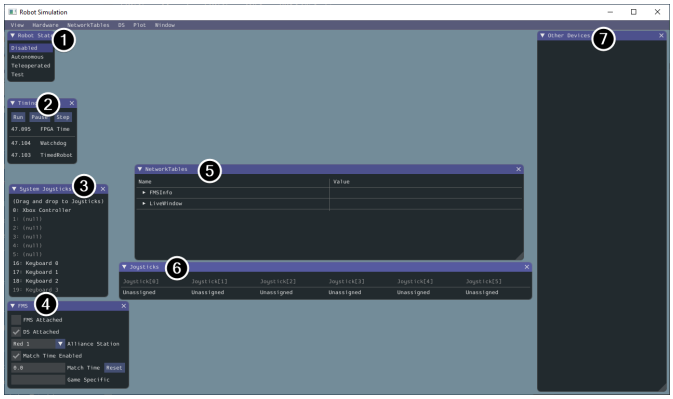


And the `Sim GUI` option should popup in a new dialog and will be selected by default. Press *Ok*. This will now launch the Simulation GUI!



22.2.2 Usando la GUI

Aprendiendo el Layout



Los siguientes elementos se muestran por defecto en la interfaz gráfica de la simulación:

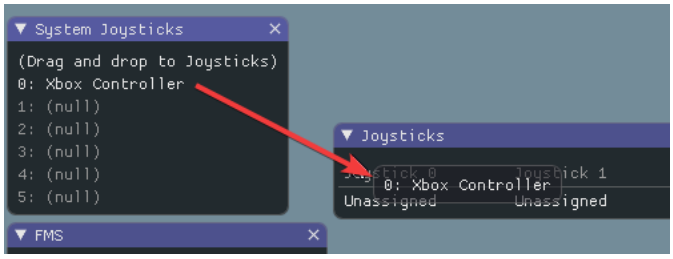
1. **Robot State** - Este es el estado actual del robot o “mode”. Usted puede hacer clic en las etiquetas para cambiar el modo como lo harías en la Driver Station.
2. **Temporización** - Muestra los valores de los temporizadores del Robot y permite manipular la temporización.
3. **System Joysticks** - Esta es una lista de los joysticks conectados actualmente a su sistema.
4. **FMS** - This is used for simulating many of the common *FMS* systems.
5. **NetworkTables** - Muestra los datos que se han publicado en NetworkTables.
6. **Joysticks** - Estos son los joysticks que el código del robot puede extraer directamente.
7. **Otros dispositivos** - Esto incluye dispositivos que no entran en ninguna de las otras categorías, como el giroscopio ADXRS450 que se incluye en el kit de piezas o dispositivos de terceros que soportan la simulación.

Los siguientes elementos pueden añadirse desde el menú Hardware, pero no se muestran por defecto.

1. **LEDs direccionables** - Muestra los LEDs controlados por la clase `AddressableLED`.
2. **Analog Inputs** - Esto incluye cualquier dispositivo que normalmente usaría la **ANALOG IN** o pin analógico en el roboRIO, como cualquier giroscopio analógico.
3. **DIO** - (Digital Input Output) - Esto incluye cualquier dispositivo que utilice el conector DIO en el roboRIO.
4. **Encoders** - Esto mostrará cualquier dispositivo instanciado que extienda o utilice la clase `Encoder` .
5. **PDPs** - Muestra el objeto Panel de Distribución de Energía.
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** - Esto incluye cualquier dispositivo de relé. Esto incluye los relés VEX Spike.
8. **Solenoids** - Esta es una lista de los solenoides “connected” o conectados. Cuando usted crea un objeto solenoide y empuja las salidas de información, estas se muestran aquí.

Añadiendo un System Joystick a Joysticks

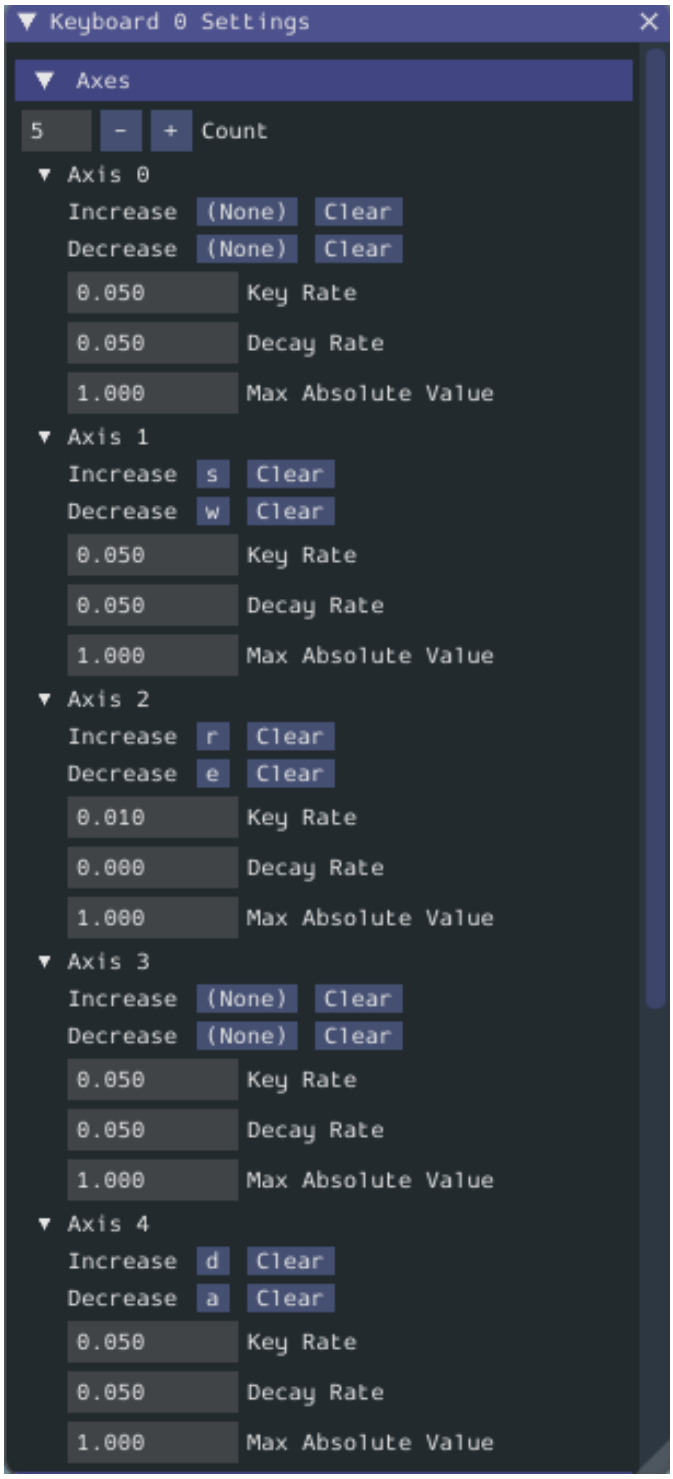
Para añadir un joystick de la lista de joysticks del sistema, simplemente haga clic y arrastre un joystick mostrado en el menú «System Joysticks» al menú «Joysticks» .



Nota: La Driver Station de FRC® hace un mapeo especial a los gamepads conectados y el simulador WPILib no los «mapea» por defecto. Puedes activar este comportamiento presionando la tecla «Map gamepad» debajo del menú «Joysticks».

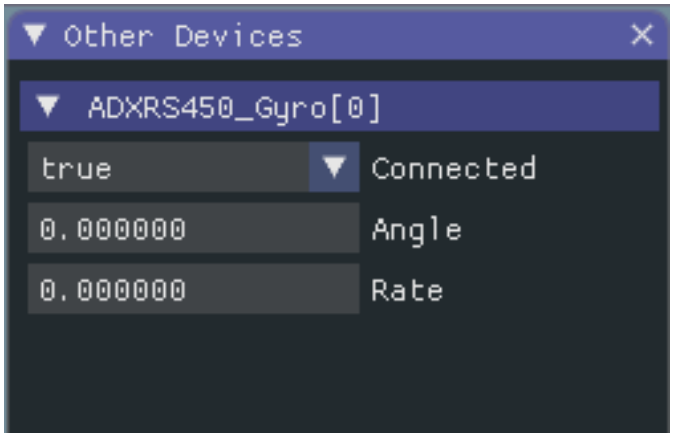
Utilizar el teclado como joystick

Se añade un teclado a la lista de joysticks del sistema haciendo clic y arrastrando uno de los elementos del teclado (por ejemplo, el Teclado 0) al igual que un joystick anterior. Para editar la configuración del teclado vaya al elemento *DS* en la barra de menú y luego elija *Keyboard 0 Settings*. Esto le permite controlar qué botones del teclado controlan cada eje. Este es un ejemplo común de cómo hacer que el teclado sea similar a un mando arcade de split sticks en un controlador de Xbox (utiliza los ejes 1 y 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

Modificar las Entradas del Giroscopio ADXRS450.

Usar el objeto ADXRS450 es una forma fantástica de probar las salidas basadas en giroscopios. Esto se expondrá en el menú «Other Devices». Entonces se expone un menú desplegable que muestra varias opciones como «Connected», «Angle», y «Rate». Todos estos valores son valores que puede cambiar en el código y que su robot puede usar en la marcha.



22.2.3 Determinando la Simulación a Partir del Código del Robot.

En los casos en que las librerías de los proveedores no compilan al ejecutar la simulación del robot, usted puede envolver su contenido con `RobotBase.isReal()` que regresa un `boolean`.

JAVA

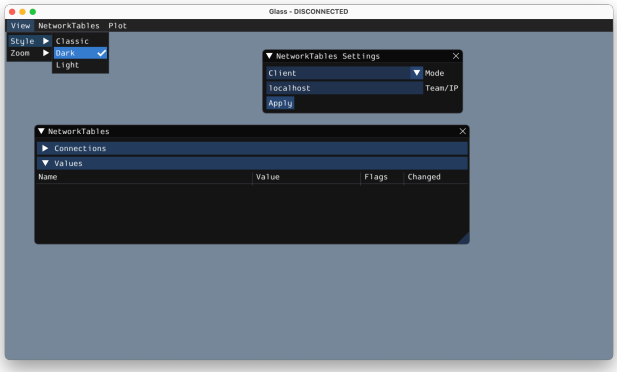
```
TalonSRX motorLeft;
TalonSRX motorRight;

public Robot() {
 if (RobotBase.isReal()) {
   motorLeft = new TalonSRX(0);
   motorRight = new TalonSRX(1);
 }
}
```

Nota: Reasignar los tipos de valores en C++ requiere mover o copiar la asignación; las clases de proveedores que ambos no soportan la simulación y carecen de un operador de asignación de movimiento o copia, no puede ser trabajado con asignación condicional a menos que se use un puntero, en lugar de un tipo de valor.

22.2.4 Cambiar la configuración de la vista

El elemento *View* del menú contiene configuraciones de *Zoom* y *Style* que se pueden personalizar. La opción de *Zoom* dicta el tamaño del texto en la aplicación mientras que la opción de *Style* le permite seleccionar entre los modos `Clásico`, `Claro`, y `Oscuro`.

Un ejemplo sobre la configuración en estilo `Oscuro` esta abajo:



22.2.5 Borrar datos de la aplicación

Los datos de la aplicación para la GUI de la simulación, incluyendo los tamaños y posiciones de los widgets, así como otra información personalizada para los widgets, se almacenan en un archivo `imgui.ini`. Este archivo se almacena en la raíz del directorio del proyecto desde el que se ejecuta la simulación.

El archivo de configuración `imgui.ini` puede ser simplemente borrado para restaurar la GUI de Simulación a una «pizarra limpia».

22.3 Simulación física con WPILib

Debido a que la :ref: *notación de espacio de estado <docs/software/advanced-controls/state-space/state-space-intro:What is State-Space Notation?>* nos permite representar de manera compacta el *dinámica* de *sistemas*, podemos aprovecharlo para proporcionar un backend para simular sistemas físicos en robots. El objetivo de estos simuladores es simular el movimiento de los mecanismos del robot sin modificar el código de usuario existente que no sea de simulación. El flujo básico de dichos simuladores es el siguiente:

- En código de usuario normal:
  - PID o algoritmos de control similares generan comandos de voltaje a partir de lecturas del encoder (u otro sensor)
  - Los valores de salida del motor están configurados
- En el código periódico de simulación:
  - El *estado* de la simulación se actualiza usando *entradas*, generalmente voltajes de motores configurados desde un bucle PID
  - Las lecturas del encoder (u otro sensor) se configuran para que el código de usuario las utilice en el siguiente paso de tiempo

22.3.1 Clases de simulación de WPILib

Las siguientes clases de simulación física están disponibles en WPILib:

- LinearSystemSim, para simular sistemas con dinámica lineal
- FlywheelSim
- DifferentialDrivetrainSim
- ElevatorSim, which models gravity in the direction of elevator motion
- SingleJointedArmSim, which models gravity proportional to the arm angle
- BatterySim, que simplemente estima la caída de voltaje de la batería en función de las corrientes dibujadas

Todas las clases de simulación (con la excepción del simulador de diferencial drive) heredan de la clase `LinearSystemSim`. Por defecto, la dinámica es la dinámica del sistema lineal $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$. Las subclases anulan el método `UpdateX(x, u, dt)` para proporcionar dinámicas personalizadas y no lineales, como simular la gravedad.

Nota: Swerve support for simulation is in the works, but we cannot provide an ETA. For updates on progress, please follow this pull request.

22.3.2 Uso en el código de usuario

Lo siguiente está disponible en el WPILib `elevatorsimulation` proyecto de ejemplo.

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.

Nota: Los simuladores de elevador y brazo evitarán que la posición simulada supere las alturas o ángulos mínimos o máximos dados. Si desea simular un mecanismo con rotación o movimiento infinito, `LinearSystemSim` puede ser una mejor opción.

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,
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        Constants.kMinElevatorHeightMeters,
        Constants.kMaxElevatorHeightMeters,
        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};
```

A continuación, `teleopPeriodic/TeleopPeriodic` (Java/C++) utiliza un circuito de control PID simple para conducir nuestro elevador a un punto de ajuste de 30 pulgadas sobre el suelo.

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++

```
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);
  }
}
```

```
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);
}
```

A continuación, `simulationPeriodic/SimulationPeriodic` (Java/C++) utiliza el voltaje aplicado al motor para actualizar la posición simulada del elevador. Usamos **:código:`SimulationPeriodic`** porque se ejecuta periódicamente solo para robots simulados. Esto significa que nuestro código de simulación no se ejecutará en un robot real.

Nota: 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`.

Finalmente, la lectura de distancia del encoder se establece usando la posición del elevador simulado, y el voltaje de la batería del robot se establece usando la corriente estimada consumida por el elevador.

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 Simulación de Dispositivos

WPILib proporciona una forma de gestionar los datos de los dispositivos de simulación en forma de la API SimDevice.

22.4.1 Simulación de las clases de dispositivos del núcleo WPILib

Las clases principales de dispositivos WPILib (por ejemplo, `Encoder`, `Ultrasonic`, etc.) tienen clases de simulación llamadas `EncoderSim`, `UltrasonicSim`, etc. Estas clases permiten interacciones con los datos del dispositivo que no serían posibles o válidas fuera de la simulación. Construirlas fuera de la simulación probablemente no interferirá con tu código, pero llamar a sus funciones y similares es un comportamiento indefinido - en el mejor de los casos no harán nada, ¡en los peores casos podrían bloquear su código! Coloque el código de simulación funcional en funciones sólo de simulación (como `simulationPeriodic()`) o envuélvalas con comprobaciones `RobotBase.isReal()`/ `RobotBase::IsReal()` (que son `constexpr` en C++).

Nota: Este ejemplo utilizará la clase «EncoderSim» como ejemplo. El uso de otras clases de simulación será casi idéntico.

Creación de objetos de dispositivos de simulación

El objeto de dispositivo de simulación puede construirse de dos maneras:

- un constructor que acepta el objeto hardware normal.
- un constructor o método de fábrica que acepte el número de puerto/índice/canal al que está conectado el dispositivo. Este sería el mismo número que se utilizó para construir el objeto de hardware regular. Esto es especialmente útil para las :doc:` pruebas unitarias<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};
```

Lectura y escritura de datos del dispositivo

Cada clase de simulación tiene funciones getter (`getXxx()`/*GetXxx()*`) y setter (`setXxx(valor)`/*SetXxx(valor)*`) para cada campo `Xxx`. Las funciones getter devolverán lo mismo que el getter de la clase de dispositivo normal.

JAVA

```
simEncoder.setCount(100);
encoder.getCount(); // 100
simEncoder.getCount(); // 100
```

C++

```
simEncoder.SetCount(100);
encoder.GetCount(); // 100
simEncoder.GetCount(); // 100
```

Registro de devoluciones de llamada

Además de los getters y setters, cada campo tiene una función `registerXxxCallback()` que registra una llamada de retorno que se ejecutará cada vez que el valor del campo cambie y devuelve un objeto `CallbackStore`. Las llamadas de retorno aceptan un parámetro de cadena con el nombre del campo y un objeto `HALValue` que contiene el nuevo valor. Antes de recuperar valores de un `HALValue`, comprueba el tipo de valor que contiene. Los tipos posibles son `HALValue.kBoolean`/*HAL_BOOL*`, `HALValue.kDouble`/*HAL_DOUBLE*`, `HALValue.kEnum`/*HAL_ENUM*`, `HALValue.kInt`/*HAL_INT*`, `HALValue.kLong`/*HAL_LONG*`.

En Java, llama a `close()` en el objeto `CallbackStore` para cancelar la llamada de retorno. Mantén una referencia al objeto para que no sea recolectado por la basura - de lo contrario la devolución de llamada será cancelada por la GC. Para proporcionar datos arbitrarios a la llamada de retorno, captúralos en la lambda o utiliza una referencia a un método.

En C++, guarda el objeto `CallbackStore` en el ámbito correcto - la llamada de retorno se cancelará cuando el objeto salga del ámbito y se destruya. Se pueden pasar datos arbitrarios a las llamadas de retorno a través del parámetro `param`.

Advertencia: Intentar recuperar un valor de un tipo desde un `HALValue` que contiene un tipo diferente es un comportamiento indefinido.

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 Simulación de otros dispositivos - La clase SimDeviceSim

Nota: Los proveedores pueden implementar su conexión a la API SimDevice de forma ligeramente diferente a la descrita aquí. También pueden proporcionar una clase de simulación específica para su clase de dispositivo. Consulte la documentación de su proveedor para obtener más información sobre lo que soportan y cómo.

The `SimDeviceSim` (**not** `SimDevice`!) class is a general device simulation object for devices that aren't core WPILib devices and therefore don't have specific simulation classes - such as vendor devices. These devices will show up in the *Other Devices* tab of the *SimGUI*.

El objeto `SimDeviceSim` se crea utilizando una clave de cadena idéntica a la que el proveedor utilizó para construir el `SimDevice` subyacente en su clase de dispositivo. Esta clave es la que muestra el dispositivo en la pestaña *Otros dispositivos*, y suele tener la forma `Prefijo:Nombre del dispositivo[índice]`. Si la clave contiene números de puertos/índices/canales, pueden pasarse como argumentos separados al constructor de `SimDeviceSim`. La clave contiene un prefijo que está oculto por defecto en la SimGUI, puede mostrarse seleccionando la opción *Show prefix*. Si no se incluye este prefijo en la clave que se pasa a `SimDeviceSim`, ¡no coincidirá con el dispositivo!

JAVA

```
SimDeviceSim device = new SimDeviceSim(deviceKey, index);
```

C++

```
frc::sim::SimDeviceSim device{deviceKey, index};
```

Una vez que tenemos el `SimDeviceSim`, podemos obtener objetos `SimValue` que representan los campos del dispositivo. También existen subclases específicas de tipo `SimDouble`, `SimInt`, `SimLong`, `SimBoolean` y `SimEnum`, que deben utilizarse en lugar de la clase `SimValue`, que no es segura. Se construyen a partir de `SimDeviceSim` utilizando una clave de cadena idéntica a la que el proveedor utilizó para definir el campo. Esta clave es la que aparece en la SimGUI. Si se intenta recuperar un objeto `SimValue` fuera de la simulación o cuando las claves del dispositivo o del campo no coinciden, se devolverá `null` - esto puede causar una `NullPointerException` en Java o un comportamiento indefinido en 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 Tutorial de simulación de transmisión

This is a tutorial for implementing a simulation model of your differential drivetrain using the simulation classes. Although the code that we will cover in this tutorial is framework-agnostic, there are two full examples available – one for each framework.

- `StateSpaceDifferentialDriveSimulation` (Java, C++) uses the command-based framework.
- `SimpleDifferentialDriveSimulation` (Java, C++) utiliza un enfoque más tradicional para el flujo de datos.

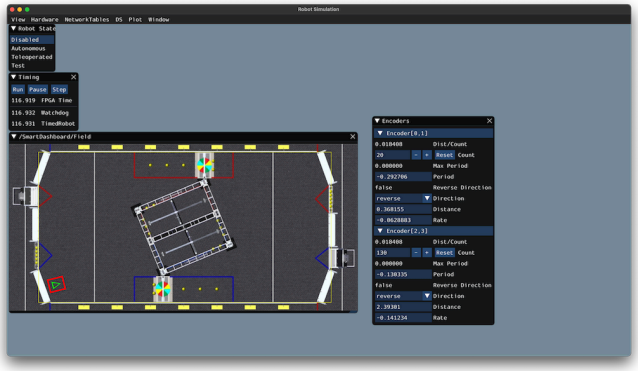
Ambos ejemplos también están disponibles en la ventana VS Code *New Project*.

22.5.1 Descripción general de la simulación de transmisión

Nota: El código de este tutorial no utiliza ningún marco específico (es decir, basado en comandos frente a flujo de datos simple); sin embargo, se proporcionará orientación en ciertas áreas sobre cómo implementar mejor ciertas piezas de código en tipos de marcos específicos.

El objetivo de este tutorial es proporcionar orientación sobre la implementación de capacidades de simulación para un robot de tren de transmisión diferencial. Al final de este tutorial, debería poder:

1. Comprender los conceptos básicos subyacentes detrás del marco de simulación de WPILib.
2. Crear un modelo de simulación de transmisión utilizando los parámetros físicos de su robot.
3. Utilizar el modelo de simulación para predecir cómo se moverá su robot real dadas las entradas de voltaje específicas.
4. Ajustar las constantes de retroalimentación y elimine los errores comunes (por ejemplo, inversión del motor) antes de tener acceso al hardware físico.
5. Utilizar la GUI de simulación para visualizar el movimiento del robot en un campo virtual.



¿Por qué simular una transmisión?

La transmisión de un robot es uno de los mecanismos más importantes del robot; por lo tanto, es importante asegurarse de que el software que alimenta la transmisión sea lo más robusto posible. Al poder simular cómo responde una transmisión física, puede comenzar a escribir software de calidad antes de tener acceso al hardware físico. Con el marco de simulación, puede verificar no solo la funcionalidad básica, como asegurarse de que las inversiones en motores y codificadores sean correctas, sino también capacidades avanzadas como verificar la precisión del seguimiento de la ruta.

22.5.2 Paso 1: creación de instancias simuladas de hardware

The WPILib simulation framework contains several `XXXSim` classes, where `XXX` represents physical hardware such as encoders or gyroscopes. These simulation classes can be used to set positions and velocities (for encoders) and angles (for gyroscopes) from a model of your drivetrain. See *the Device Simulation article* for more info about these simulation hardware classes and simulation of vendor devices.

Nota: Simulation objects associated with a particular subsystem should live in that subsystem. An example of this is in the `StateSpaceDriveSimulation` (Java, C++) example.

Simulación de codificadores

La clase `EncoderSim` permite a los usuarios establecer las posiciones y velocidades del codificador en un objeto `Codificador` dado. Cuando se ejecuta en hardware real, la clase `Encoder` interactúa con sensores reales para contar las revoluciones (y convertirlas en unidades de distancia automáticamente si está configurado para hacerlo); sin embargo, en la simulación no existen tales medidas para realizar. La clase `EncoderSim` puede aceptar estas lecturas simuladas de un modelo de su transmisión.

Nota: No es posible simular codificadores conectados directamente a controladores de motor CAN utilizando clases WPILib. Para obtener más información sobre su controlador de motor específico, lea la documentación de su proveedor.

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};
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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};
```

Simulación de giroscopios

Similar a la clase `EncoderSim`, también existen clases de giroscopio simulado para giroscopios WPILib de uso común: `AnalogGyroSim` y `ADXRS450_GyroSim`. Estos también se construyen de la misma manera.

Nota: 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 Paso 2: Creación del modelo de la cadena de transmisión

Para determinar con precisión cómo responderá su tren motriz físico a las entradas de voltaje del motor dadas, se debe crear un modelo preciso de su tren motriz. Este modelo generalmente se crea midiendo varios parámetros físicos de su robot real. En WPILib, este modelo de simulación de transmisión está representado por la clase `DifferentialDrivetrainSim`.

Creación de un `DifferentialDrivetrainSim` a partir de mediciones físicas

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:

- El tipo y número de motores en un lado de la cadena de transmisión
- 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 masa de la cadena de transmisión(se recomienda utilizar la masa de todo el robot, ya que modelará con mayor precisión las características de aceleración de su robot para el seguimiento de la trayectoria).
- El radio de las ruedas motrices
- El ancho de la pista (distancia entre las ruedas izquierda y derecha).
- Desviaciones estándar del ruido de medición: esto representa la cantidad de ruido de medición que espera de sus sensores reales. El ruido de medición es una matriz con 7 elementos, y cada elemento representa la desviación estándar del ruido de medición en x, y, rumbo, velocidad izquierda, velocidad derecha, posición izquierda y posición derecha, respectivamente. Esta opción puede omitirse en C++ o establecerse en `null` en Java si el ruido de medición no es deseable.

Puede calcular el ruido de medición de sus sensores tomando múltiples puntos de datos del estado que está tratando de medir y calculando la desviación estándar usando una herramienta como Python. Por ejemplo, para calcular la desviación estándar en la estimación de velocidad de sus codificadores, puede mover su robot a una velocidad constante, tomar varias medidas y calcular su desviación estándar de la media conocida. Si este proceso es demasiado tedioso, los valores usados en el siguiente ejemplo deberían ser una buena representación del ruido promedio de los codificadores.

Nota: La desviación estándar del ruido para una medida tiene las mismas unidades que esa medida. Por ejemplo, la desviación estándar del ruido de velocidad tiene unidades de m / s.

Nota: Es muy importante utilizar las unidades del SI (es decir, metros y radianes) al pasar los parámetros en Java. En C++, la *librería de unidades* puede ser usada para especificar cualquier tipo de unidad.

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).
  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}};
```

Creating a `DifferentialDrivetrainSim` from 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.

Importante: You must need two sets of `Kv` and `Ka` gains from the identification tool – one from straight-line motion and the other from rotating in place. We will refer to these two sets of gains as linear and angular gains respectively.

Este constructor toma los siguientes parámetros:

- A linear system representing the drivetrain – this can be created using the identification gains.
- El ancho de la pista(distancia entre las ruedas izquierda y derecha).
- El tipo y número de motores en un lado de la cadena de transmisión
- The gear ratio between the motors and the wheels as output *torque* over input *torque* (this number is usually greater than 1 for drivetrains).
- El radio de las ruedas motrices
- Desviaciones estándar del ruido de medición: esto representa la cantidad de ruido de medición que espera de sus sensores reales. El ruido de medición es una matriz con 7 elementos, y cada elemento representa la desviación estándar del ruido de medición en x, y, rumbo, velocidad izquierda, velocidad derecha, posición izquierda y posición derecha, respectivamente. Esta opción puede omitirse en C++ o establecerse en `null` en Java si el ruido de medición no es deseable.

Puede calcular el ruido de medición de sus sensores tomando múltiples puntos de datos del estado que está tratando de medir y calculando la desviación estándar usando una herramienta como Python. Por ejemplo, para calcular la desviación estándar en la estimación de velocidad de sus codificadores, puede mover su robot a una velocidad constante, tomar varias medidas y calcular su desviación estándar de la media conocida. Si este proceso es demasiado tedioso, los valores usados en el siguiente ejemplo deberían ser una buena representación del ruido promedio de los codificadores.

Nota: La desviación estándar del ruido para una medida tiene las mismas unidades que esa medida. Por ejemplo, la desviación estándar del ruido de velocidad tiene unidades de m / s.

Nota: Es muy importante utilizar las unidades del SI (es decir, metros y radianes) al pasar los parámetros en Java. En C++, la *librería de unidades* puede ser usada para especificar cualquier tipo de unidad.

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:
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  // 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),
  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}};
```

Crear un `DifferentialDrivetrainSim` del chasis KoP

La clase `DifferentialDrivetrainSim` también tiene un método estático `createKitbotSim()` (Java) / `CreateKitbotSim()` (C++) que puede crear una instancia del `DifferentialDrivetrainSim` utilizando los parámetros estándar del Kit de Piezas Chasis. Este método toma 5 argumentos, dos de los cuales son opcionales:

- El tipo y número de motores en un lado de la cadena de transmisión
- The gear ratio between the motors and the wheels as output *torque* over input *torque* (this number is usually greater than 1 for drivetrains).
- El diámetro de las ruedas instaladas en el chasís.
- El momento de inercia de la base motriz (opcional).
- Desviaciones estándar del ruido de medición: esto representa la cantidad de ruido de medición que espera de sus sensores reales. El ruido de medición es una matriz con 7 elementos, y cada elemento representa la desviación estándar del ruido de medición en x, y, rumbo, velocidad izquierda, velocidad derecha, posición izquierda y posición derecha, respectivamente. Esta opción puede omitirse en C++ o establecerse en `null` en Java si el ruido de medición no es deseable.

Puede calcular el ruido de medición de sus sensores tomando múltiples puntos de datos del estado que está tratando de medir y calculando la desviación estándar usando una herramienta como Python. Por ejemplo, para calcular la desviación estándar en la estimación de velocidad de sus codificadores, puede mover su robot a una velocidad constante, tomar varias medidas y calcular su desviación estándar de la media conocida. Si este proceso es demasiado tedioso, los valores usados en el siguiente ejemplo deberían ser una buena representación del ruido promedio de los codificadores.

Nota: La desviación estándar del ruido para una medida tiene las mismas unidades que esa medida. Por ejemplo, la desviación estándar del ruido de velocidad tiene unidades de m / s.

Nota: Es muy importante utilizar las unidades del SI (es decir, metros y radianes) al pasar los parámetros en Java. En C++, la *librería de unidades* puede ser usada para especificar cualquier tipo de unidad.

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.
);
```

Nota: Puedes utilizar los enum (Java) / struct (C++) `KitbotMotor`, `KitbotGearing`, y `KitbotWheelSize` para obtener las configuraciones más utilizadas del Kit de Piezas Chasis.

Importante: La construcción de su instancia `DifferentialDrivetrainSim` de esta manera es sólo una aproximación y está pensada para que los equipos se pongan en marcha rápidamente con la simulación. El uso de valores empíricos medidos a partir de su robot físico siempre dará resultados más precisos.

22.5.4 Paso 3: Actualización del modelo de transmisión

Ahora que se ha hecho el modelo de transmisión, necesita ser actualizado periódicamente con los últimos comandos de voltaje del motor. Se recomienda hacer este paso en un método separado `simulationPeriodic()` / `SimulationPeriodic()` dentro de su subsistema y sólo llamar a este método en la simulación.

Nota: 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.

Hay tres pasos principales para actualizar el modelo:

1. Establezca el *entrada* del modelo de transmisión. Estos son los voltajes del motor de los dos lados del tren de transmisión.
2. Haga avanzar el modelo en el tiempo por el paso de tiempo periódico nominal (generalmente 20 ms). Esto actualiza todos los estados de la transmisión (es decir, pose, posiciones del codificador y velocidades) como si hubieran pasado los 20 ms.
3. Actualice los sensores simulados con nuevas posiciones, velocidades y ángulos para usar en otros lugares.

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);
}

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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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());
}
```

Importante: Si el lado derecho de su transmisión está invertido, DEBE negar el voltaje correcto en la llamada `SetInputs()` para asegurarse de que los voltajes positivos corresponden al movimiento hacia adelante.

Importante: Debido a que el modelo del simulador de tren de conducción devuelve las posiciones y las velocidades en metros y m/s respectivamente, éstas deben ser convertidas en ticks de codificación y ticks/s cuando se llama a `SetDistance()` y `SetRate()`. Alternativamente, puedes configurar `SetDistancePerPulse` en los codificadores para que el objeto `Encoder` se encargue de esto automáticamente - este es el enfoque que se toma en el ejemplo anterior.

Ahora que las posiciones simuladas del codificador, las velocidades y los ángulos del giroscopio se han establecido, puedes llamar a `m_leftEncoder.GetDistance()`, etc. en tu código de robot como normalmente y se comportará exactamente como lo haría en un robot real. Esto implica realizar cálculos de odometría, ejecutar bucles de retroalimentación PID de velocidad para el seguimiento de la trayectoria, etc.

22.5.5 Paso 4: Actualización de la odometría y visualización de la posición del robot

Ahora que las posiciones simuladas del codificador, las velocidades y los ángulos del giroscopio se actualizan con información precisa periódicamente, estos datos pueden utilizarse para actualizar la posición del robot en un bucle periódico (como el método `periodic()` en un `Subsistema`). En la simulación, el bucle periódico utilizará lecturas simuladas del codificador y del giroscopio para actualizar la odometría, mientras que en el robot real, el mismo código utilizará lecturas reales del hardware físico.

Nota: Para más información sobre el uso de la odometría, ver *este documento*.

Visualización de la postura del robot

La pose del robot puede ser visualizada en el GUI del simulador (durante la simulación) o en un dashboard como Glass (en un robot real) enviando la pose de odometría sobre un objeto «Field2d». Un `Campo2d` puede ser construido trivialmente sin ningún argumento constructivo:

JAVA

```
private Field2d m_field = new Field2d();
```

C++

```
#include <frc/smartdashboard/Field2d.h>

..

frc::Field2d m_field;
```

Esta instancia de `Field2d` debe enviarse a través de NetworkTables. El mejor lugar para hacer esto es en el constructor de su subsistema.

JAVA

```
public Drivetrain() {
  ...
  SmartDashboard.putData("Field", m_field);
}
```

C++

```
#include <frc/smartdashboard/SmartDashboard.h>

Drivetrain() {
  ...
  frc::SmartDashboard::PutData("Field", &m_field);
}
```

Nota: The `Field2d` instance can also be sent using a lower-level NetworkTables API or using the *Shuffleboard API*.

Finalmente, la posición de su odometría debe actualizarse periódicamente en el objeto `Field2d`. Recuerde que esto debe ser en un método general `periodic()`, es decir, uno que se ejecute tanto durante la simulación como durante la operación real del 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());
}
```

Importante: Es importante que este código se coloque en un método `periodic()` regular - uno que se llame periódicamente independientemente del modo de operación. Si está usando la biblioteca basada en comandos, este método ya existe. Si no, eres responsable de llamar a este método periódicamente desde la clase principal `Robot`.

Nota: 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 Unit Testing

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 Escribir código para pruebas

Nota: This example can be easily adapted to the command-based paradigm by having `Intake` inherit from `SubsystemBase`.

Nuestro subsistema será un mecanismo Intake para Infinite Recharge que contiene un pistón y un motor: el pistón despliega/retrae la admisión y el motor arrastrará las Power Cells hacia adentro. No queremos que el motor funcione si el mecanismo de admisión no está desplegado porque no hará nada.

Para proporcionar una «pizarra limpia» para cada prueba, necesitamos tener una función para destruir el objeto y liberar todas las asignaciones de hardware. En Java, esto se hace implementando la interfaz `AutoCloseable` y su método `.close()`, destruyendo cada objeto de los miembros al llamar al método `.close()` del objeto - un objeto sin un método `.close()` probablemente no necesite ser cerrado. En C ++, el destructor predeterminado se llamará automáticamente cuando el objeto salga del alcance y llamará a los destructores de los objetos miembros.

Nota: Es posible que los proveedores no admitan el cierre de recursos de la misma manera que se muestra aquí. Consulte la documentación de su proveedor para obtener más información sobre qué admiten y cómo.

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
  }
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  public void activate(double speed) {
    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++ (Encabezado)

```
#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++ (Fuente)

```
#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
}
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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 Writing Tests

Importante: Los tests se colocan dentro del conjunto de fuentes de `test`: `/src/test/java/` y `/src/test/cpp/` para tests de Java y C++ respectivamente. Los archivos fuera de esa raíz de origen no tienen acceso al marco de prueba; esto fallará en la compilación debido a referencias sin resolver.

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.

Cada método de prueba debe contener al menos una *aserción* (`assert*()` en Java o `EXPECT_*()` en C++). Estas aserciones verifican una condición en tiempo de ejecución y fallan la prueba si la condición no se cumple. Si hay más de una aserción en un método de prueba, la primera aserción fallida bloqueará la prueba - la ejecución no llegará a las aserciones posteriores.

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.

Nota: La comparación de valores de punto flotante no es precisa, por lo que la comparación debe hacerse con un parámetro de error aceptable (`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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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());
}
```

Para un uso más avanzado de JUnit y Google Test, consulte la documentación del framework.

22.6.3 Pruebas para correrlo

Nota: Las pruebas se ejecutarán siempre en simulación en su escritorio. Para conocer los requisitos previos y más información, consulte el documento *introducción a la simulación*.

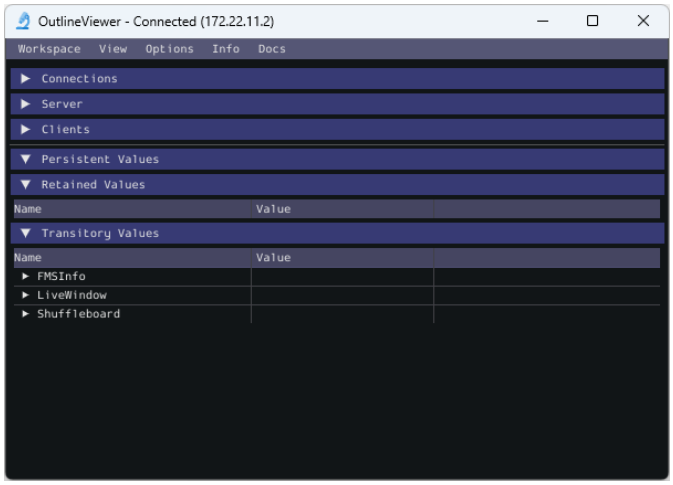
Para que las pruebas Java se ejecuten, asegúrese de que su archivo `build.gradle` contenga el siguiente bloque:

```
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.

Por defecto, Gradle ejecuta las pruebas cada vez que se construye el código del robot, incluyendo los despliegues. Esto aumentará el tiempo de despliegue, y las pruebas fallidas harán que la construcción y el despliegue fallen. Para evitar que esto ocurra, puedes utilizar *Change Skip Tests On Deploy Setting* de la paleta de comandos para configurar si se ejecutan las pruebas al desplegar.

23 OutlineViewer

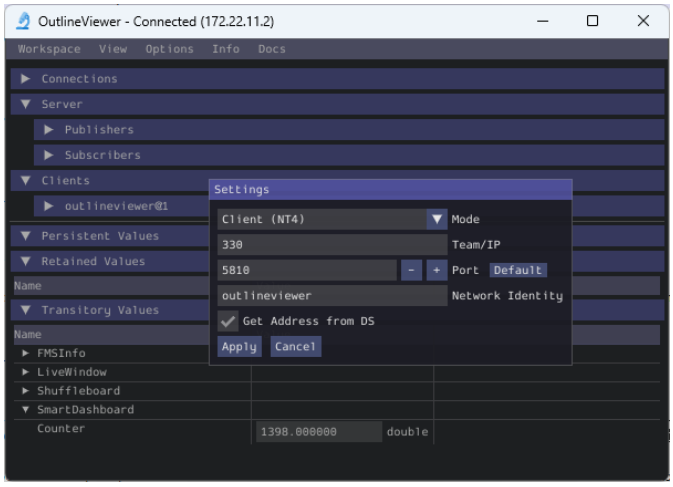


OutlineViewer es una utilidad que se utiliza para ver, modificar y agregar al contenido de NetworkTables con fines de depuración. Muestra todos los pares clave-valor que se encuentran actualmente en NetworkTables y se pueden utilizar para modificar el valor de claves existentes o agregar claves nuevas a la tabla. OutlineViewer se incluye en las actualizaciones de los lenguajes C++ y Java.

En Visual Studio Code, presione `Ctrl+Shift+P` y escriba ``WPILib``o haga clic en el logotipo de WPILib en la parte superior derecha para iniciar la paleta de comandos de WPILib. Seleccione *Start Tool*, luego seleccione *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*.

Nota: You can use `localhost` instead of a team number to point OutlineViewer at a simulated robot, Romi or XRP.

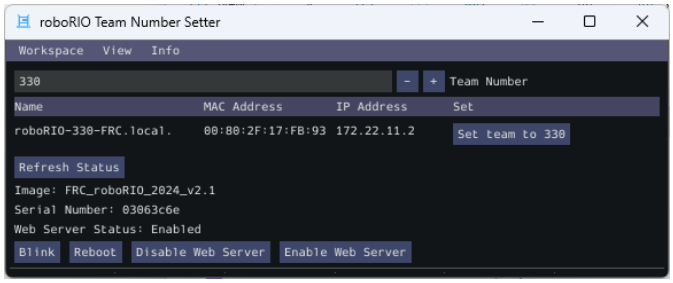


Para agregar pares adicionales de clave/valor a NetworkTables, haga clic derecho en una ubicación y elija el tipo de datos correspondiente.

Nota: Los equipos de LabVIEW pueden utilizar la pestaña Variables de LabVIEW Dashboard para lograr la misma funcionalidad 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 xxxxx*. 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 Procesamiento de la visión

25.1 Introducción a la Visión

25.1.1 ¿Qué es La Visión?

Visión en FRC® utiliza una cámara conectada al robot para ayudar a los equipos a anotar y conducir, tanto durante el período autónomo como el período teleoperado.

Métodos de visión

Hay dos métodos principales los cuales la mayoría de los equipos de FRC utilizan

Streaming

Este método implica el envío de la cámara a la estación del conductor para que el conductor y el manipulador puedan obtener información visual desde el punto de vista del robot. Este método es simple y toma poco tiempo para implementarlo, lo que lo convierte en una buena opción si no se necesitan características del procesamiento de la visión.

- Streaming utilizando la roboRIO

Procesamiento

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.

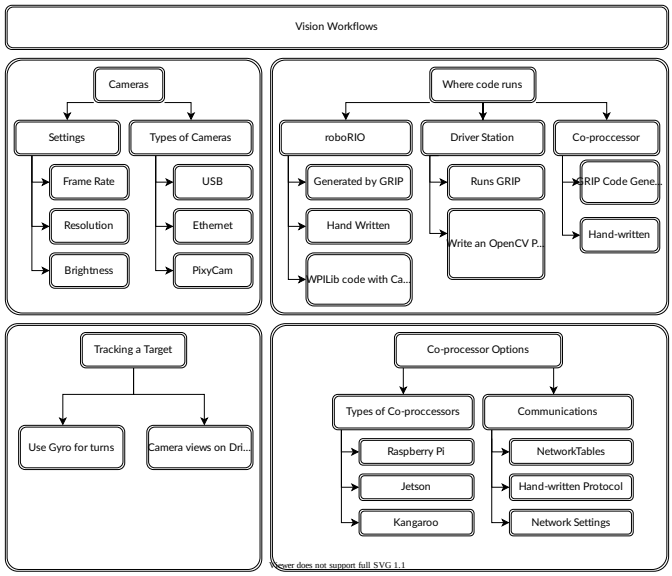
- *Procesamiento de Visión con Raspberry Pi*
- *Procesamiento de la visión con la roboRIO*

Para información adicional de pros y contras de usar un coprocesador para el procesamiento de la visión, vaya a la siguiente página, Estrategias para la programación de la visión. *Estrategias para la programación de la visión*.

25.1.2 Estrategias para la programación de la visión

Usar la visión por computadora es una gran manera de hacer que tu robot responda a los elementos en el campo y hacerlo mucho más autónomo. A menudo en los juegos de FRC® hay puntos de bonificación por disparar de forma autónoma pelotas u otras piezas de juego a las metas o navegar a lugares en el campo. La visión por computador es una gran manera de resolver muchos de estos problemas. Y si tienes código autónomo que puede hacer el desafío, entonces puede ser usado durante el período de teleoperado para ayudar a los conductores.

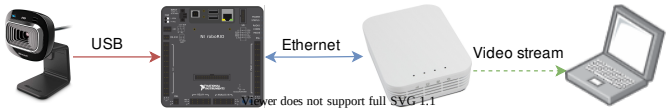
Hay muchas opciones para elegir los componentes para el procesamiento de la visión y donde el programa de visión debería funcionar. WPILib y las herramientas asociadas apoyan un número de opciones y dan a los equipos mucha flexibilidad para decidir qué hacer. Este artículo tratará de darte un poco de de muchas de las opciones y compensaciones que están disponibles.



Biblioteca de Visión por Computadora OpenCV

OpenCV es una biblioteca de visión de ordenador de código abierto que se utiliza ampliamente en todo el mundo académico y la industria. Cuenta con el apoyo de los fabricantes de hardware que proporcionan procesamiento acelerado en la GPU, tiene enlaces para varios lenguajes incluyendo C++, Java y Python. También está bien documentado con muchos sitios web, libros, vídeos y cursos de formación, por lo que hay muchos recursos disponibles para ayudar a aprender a usarla. Las versiones C++ y Java de WPILib incluyen las bibliotecas de OpenCV, hay apoyo en la biblioteca para la captura, procesamiento y para mostrar video, y herramientas para ayudarte a crear tus algoritmos de visión. Para obtener más información sobre OpenCV, ver https://opencv.org.

Código de visión en roboRIO

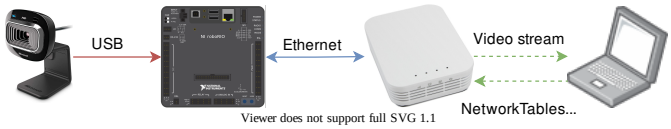


Vision code can be embedded into the main robot program on the roboRIO. Building and running the vision code is straightforward because it is built and deployed along with the robot program. The vision code can be written in C++, Java, or Python. The disadvantage of this approach is that having vision 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.

Utilizando las funciones proporcionadas por la clase CameraServer, el flujo de vídeo puede ser enviado a tableros como el Shuffleboard para que los operadores puedan ver lo que la cámara ve. Además, se pueden agregar anotaciones a las imágenes usando comandos OpenCV para que los objetivos u otros interesantes, los objetos pueden ser identificados en la vista del tablero.

Código de visión en computadora DS

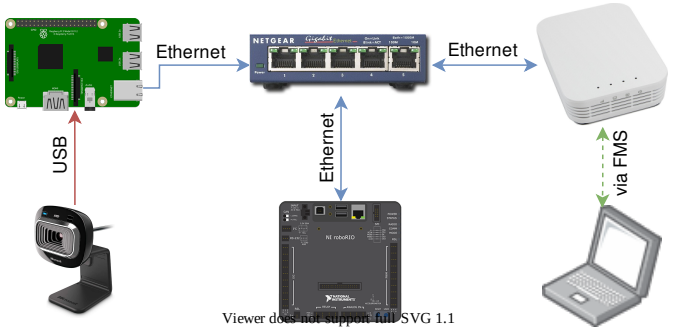


When vision code is running on the DS computer, the video is streamed back to the Driver Station laptop for processing. Even the older Classmate laptops are substantially faster at vision processing than the roboRIO. You can write your own vision program using a language of your choosing. Python makes a good choice since there is a native NetworkTables implementation and the OpenCV bindings are very good.

Después de que las imágenes sean procesadas, los valores clave como la posición del objetivo, la distancia o cualquier otra cosa que necesite pueden ser enviados de vuelta al robot con NetworkTables. Este enfoque generalmente tiene una mayor latencia, ya que se añade un retraso debido a que las imágenes necesitan ser enviadas al portátil. Las limitaciones del ancho de banda también limitan la resolución máxima y el FPS de las imágenes sean utilizadas para el procesamiento.

The video stream can be displayed on Shuffleboard or SmartDashboard.

Código de video en coprocesador



Los coprocesadores como el Pi de Raspberry son ideales para apoyar el código de visión (ver docs/software/vision-processing/frcvision/using-the-raspberry-pi-for-frc:Using the Raspberry Pi for FRC). La ventaja es que pueden correr a toda velocidad y no interferir con el programa de robots. En este caso, la cámara está probablemente conectada al coprocesador o (en el caso de las cámaras Ethernet) un interruptor Ethernet en el robot. El programa puede ser escrito en cualquier idioma; Python es una buena elección por su simple unión a OpenCV y NetworkTables. Algunos equipos han utilizado coprocesadores de visión de alto rendimiento como el Nvidia Jetson para una mayor velocidad y resolución, aunque este enfoque generalmente requiere conocimientos avanzados de Linux y de programación.

Este enfoque requiere un poco más de experiencia en programación así como una pequeña cantidad adicional de espacio pero, por lo demás, aporta lo mejor de ambos mundos en comparación con los otros dos enfoques, ya que los coprocesadores son mucho más rápidos que el roboRIO y el procesamiento de la imagen se puede realizar con un mínimo de latencia o uso de ancho de banda.

Los datos pueden ser enviados desde el programa de visión en el coprocesador al robot usando NetworkTables o un protocolo privado a través de una red o una conexión en serie.

Opciones de cámara

Hay un número de opciones de cámara soportadas por WPILib. Las cámaras tienen un número de parámetros que afectan al funcionamiento; por ejemplo, la velocidad de fotogramas y la resolución de la imagen afectan a la calidad de las imágenes recibidas, pero cuando se establece un tiempo de procesamiento de impacto demasiado alto y, si se envía a la estación del conductor, puede exceder el ancho de banda disponible en el campo.

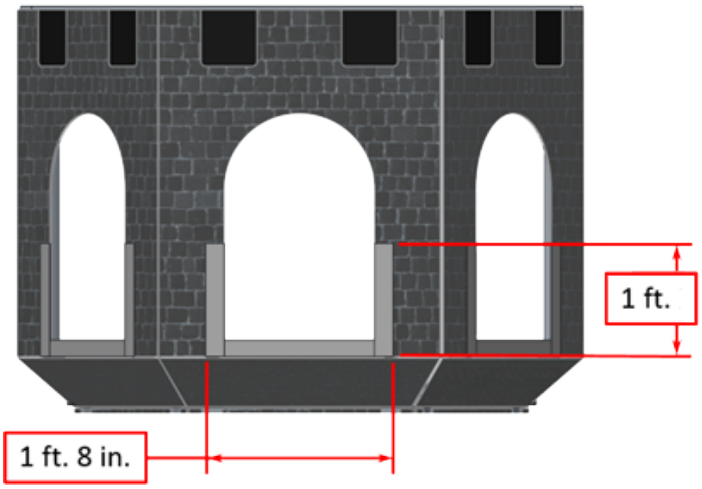
La clase CameraServer en C++ y Java se utiliza para interactuar con las cámaras conectadas al robot. Recupera cuadros para el procesamiento local a través de un objeto fuente y envía el flujo a su puesto de conductor para ser visto o procesado allí.

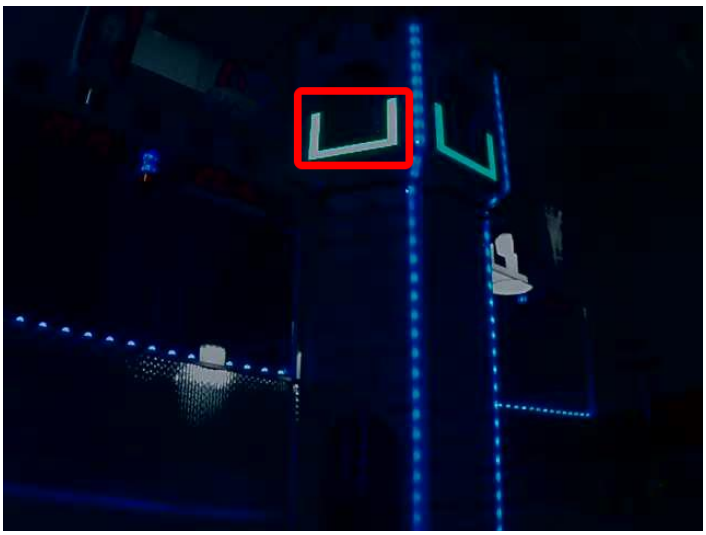
25.1.3 Información del objetivo y retrorreflexión

Muchos juegos de FRC® tienen cinta retrorreflectante adherida a los elementos del campo para ayudar en el procesamiento de la visión. Este documento describe los objetivos de visión del juego de FRC 2016 y la propiedades del material que compone los objetivos.

Nota: Para las dimensiones oficiales y los dibujos de todos los componentes del campo, por favor vea los Dibujos Oficiales del Campo.

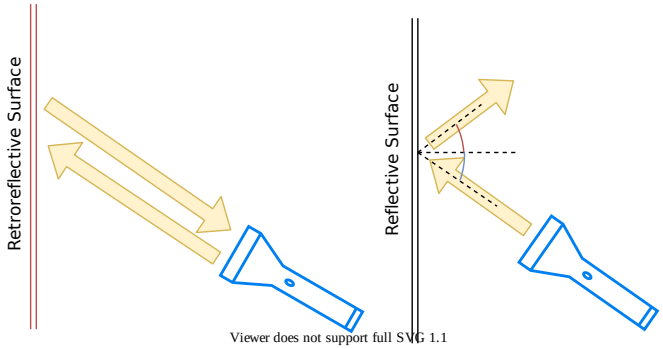
Objetivos





Cada objetivo de visión de 2016 consiste en una forma de U de 1” 8» de ancho y 1” de alto hecha de un retrorreflector de 2» de ancho (3M 8830 Silver Marking Film). Los objetivos están localizados inmediatamente adyacentes a la parte inferior de cada meta alta. Cuando está bien iluminada, la cinta retrorreflectante produce un brillante y/o marcador de color saturado.

Retrorreflectividad vs Reflectividad



Los materiales altamente reflectantes son generalmente reflejados para que la luz «rebote» en un ángulo suplementario. Como se muestra arriba a la izquierda, los ángulos azul y rojo suman 180 grados. Un equivalente explicativo es que la luz refleja sobre la superficie normal la línea verde dibujada perpendicularmente a la superficie. Noten que una luz apuntada a la superficie volverá a la fuente de luz sólo si el ángulo azul es de ~90 grados.

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.

Ejemplos de retrorreflexión

Este material debería ser relativamente familiar ya que se utiliza con frecuencia para mejorar la visibilidad nocturna, señales de tráfico, bicicletas y peatones.

Inicialmente, la retro-reflexión puede no parecer una propiedad útil para la seguridad nocturna, pero cuando la luz y el ojo están cerca el uno del otro, como se muestra arriba, la

luz reflejada regresa al ojo, y el material brilla intensamente incluso a grandes distancias. Debido al pequeño ángulo entre los ojos del conductor y los faros del vehículo, los materiales retro-reflectantes pueden aumentar mucho la visibilidad de objetos distantes durante la conducción nocturna.

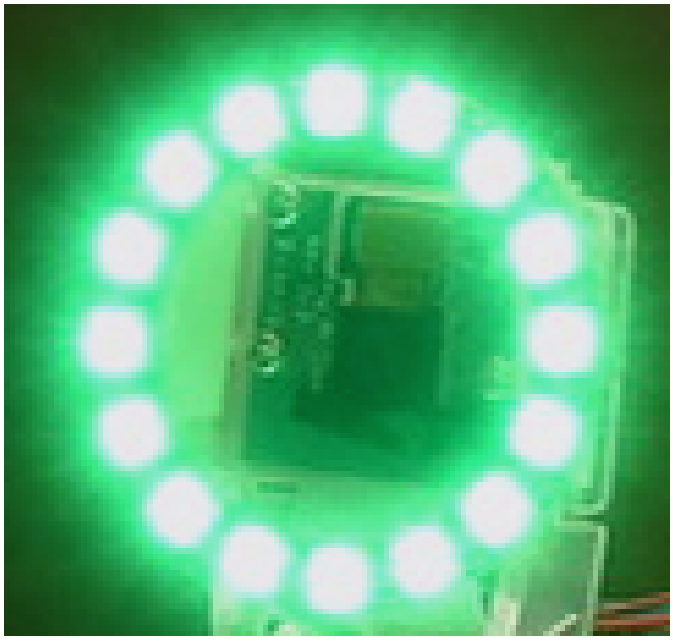
Demostración

Para explorar más a fondo las propiedades de los materiales retro-reflectantes:

1. Coloca un trozo del material en una pared o superficie vertical
2. Párese a 10-20 pies de distancia y apunte una pequeña linterna al material.
3. Empieza con la luz sostenida en tu ombligo, y levántala lentamente hasta que esté entre tus ojos. A medida que la luz se acerca a los ojos, la intensidad de la luz devuelta aumentará rápidamente
4. Altere el ángulo moviéndose a otros lugares de la sala y repitiendo. La brillante reflexión debe ocurrir en una amplia gama de ángulos de visión, pero el ángulo de la luz la fuente a ojo es clave y debe ser bastante pequeña.

Experimentar con diferentes fuentes de luz. El material es cientos de veces más reflectante que la pintura blanca; así que las fuentes de luz tenue funcionarán bien. Por ejemplo, una luz roja de seguridad para bicicletas demostrará que el color de la fuente de luz determina el color de la luz reflejada. Si es posible, coloque varios miembros del equipo en diferentes lugares, cada uno con su propia luz fuente. Esto demostrará que los efectos son en gran medida independientes, y el material puede aparecer simultáneamente de diferentes colores a varios miembros del equipo. Esto también demuestra que el material es en gran parte inmune a la iluminación ambiental. La luz que regresa al espectador es casi enteramente determinada por una fuente de luz que controlan o una directamente detrás de ellos. Usando una linterna, identifique otros artículos retro-reflectantes que ya estén en su entorno: en la ropa, mochilas, zapatos, etc.

ILUMINACIÓN



Hemos visto que la cinta retro-reflectante no brillará a menos que una fuente de luz se dirija a ella, y la fuente de luz debe pasar muy cerca de la lente de la cámara o de los ojos del observador. Mientras que allí son varias formas de lograrlo, un tipo de fuente de luz muy útil para investigar es el flash de anillo, o luz de anillo, que se muestra arriba. Coloque la fuente de luz directamente sobre o alrededor de la lente de la cámara y proporciona una iluminación muy uniforme. Debido a su brillante salida y pequeño tamaño, los LEDs son particularmente útiles para construir este tipo de dispositivo.

Como se muestra arriba, se dispone de económicos arreglos circulares de LEDs en una variedad de colores y tamaños y son fáciles de acoplar a las cámaras, y algunas pueden incluso ser apagadas de un Raspberry Pi. Aunque no están diseñados para una iluminación difusa, funcionan bastante bien para causar cinta retro-reflectante para brillar. Un pequeño anillo de LED verde está disponible a través de FIRST Choice. Otros anillos LED similares están disponibles en proveedores como SuperBrightLEDs.

Imágenes de muestra

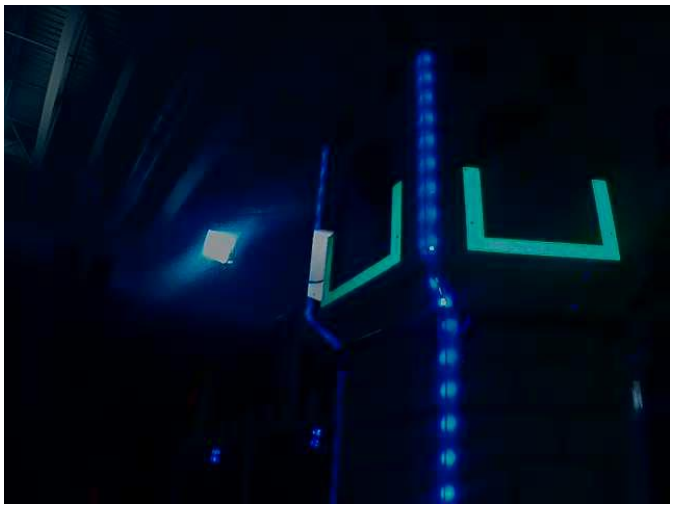
Las imágenes de muestra se encuentran con los ejemplos de código de cada idioma (empaquetadas con LabVIEW, y en un ZIP separado en el mismo lugar que las muestras de C++/Java).

25.1.4 Identificando y procesando los objetivos

Una vez que se captura una imagen, el siguiente paso es identificar el/los objetivo(s) de la visión en la imagen. Este documento se ocupará de un enfoque para la identificación de los objetivos de 2016. Obsérvese que las imágenes utilizadas en esta sección fueron tomadas con la cámara intencionalmente configurada para subexponer las imágenes, produciendo imágenes muy oscuras con la excepción de los objetivos iluminados, ver la sección de ajustes de la cámara para más detalles.

Imagen original

La imagen que se muestra a continuación es la imagen inicial del ejemplo descrito aquí. La imagen era tomada usando la luz verde del anillo disponible en *FIRST*®Choice combinada con un anillo adicional de luz de un tamaño diferente. Se proporcionan imágenes de muestra adicionales con los ejemplos de código de visión.



¿Qué es HSL/HSV?

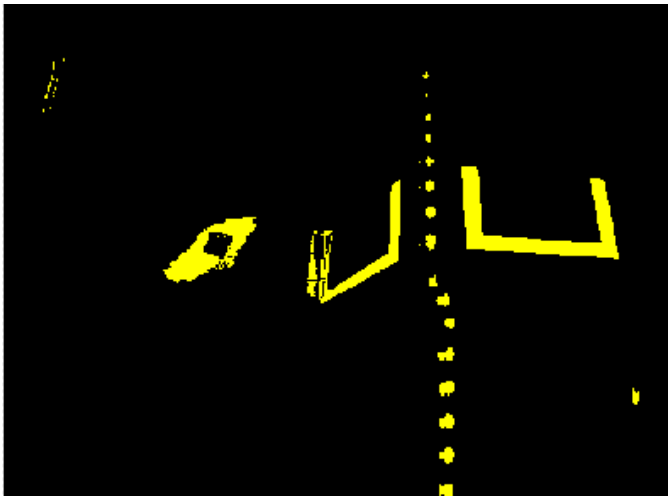
El tono del color se ve comúnmente en la rueda de colores del artista y contiene los colores del arco iris: rojo, naranja, amarillo, verde, azul, índigo y violeta. El tono se especifica usando un ángulo radial en la rueda, pero en la imagen el círculo típicamente contiene sólo 256 empezando con el rojo en el cero, pasando por el arco iris, y volviendo al rojo en el extremo superior. La saturación de un color especifica la cantidad de color, o la proporción del color de la tonalidad a una sombra de gris. Una proporción más alta significa más colorido, menos gris. La saturación cero no tiene ningún matiz y es completamente gris. La luminancia o valor indica el tono de gris que el tono está mezclado con. El negro es 0 y el blanco es 255.

El código de ejemplo utiliza el espacio de color HSV para especificar el color del objetivo. La razón principal es que permite utilizar fácilmente el brillo de los objetivos en relación con el resto de la imagen como un criterio de filtrado utilizando el componente de Valor (HSV) o Luminancia (HSL). Otra razón para usar el sistema de color HSV es que la operación de umbral utilizada en el ejemplo funciona más eficientemente en el roboRIO cuando se hace en el espacio de color del HSV.

Enmascarando

En este paso inicial, los valores de los píxeles se comparan con los valores de color o brillo constantes para crear una máscara binaria que se muestra abajo en amarillo. Este único paso elimina la mayoría de los píxeles que son no es parte de la cinta retro-reflectante del objetivo. El enmascaramiento basado en el color funciona bien siempre que el color es relativamente saturado, brillante y consistente. Las desigualdades de color son generalmente más precisas cuando se especifica usando el HSL (Tono, Saturación y Luminancia) o el HSV (Tono, Saturación, y Valor) que el espacio de color RGB (Rojo, Verde y Azul). Esto es especialmente cierto cuando la gama de colores es bastante grande en una o más dimensiones.

Observe que además del objetivo, otras partes brillantes de la imagen (la luz superior y la torre iluminación) también son atrapados por el paso de la máscara.



Análisis de partículas

Después de la operación de enmascaramiento, se utiliza una operación de informe de partículas para examinar la zona, delimitando rectángulo, y un rectángulo equivalente para las partículas. Estos se utilizan para ayudar a elegir las formas más rectangulares. Cada prueba descrita a continuación genera una puntuación (0-100) que luego se compara con los límites de puntuación predefinidos para decidir si el de las partículas es un objetivo o no.

Área de cobertura

La puntuación del área se calcula comparando el área de la partícula con el área del cuadro delimitador dibujado alrededor de la partícula. El área de las bandas retrorreflectoras es de 80 pulgadas cuadradas (~516 cm^2). El área del rectángulo que contiene el objetivo es de 240 pulgadas cuadradas (~0.15 m^2). Esto significa que la relación ideal entre el área y el área del cuadro delimitador es de 1/3. Las relaciones de área cercanas a 1/3 producirán un puntaje cercano a 100, a medida que la relación se aleje de 1/3 el puntaje se acercará a 0.

Relación de aspecto

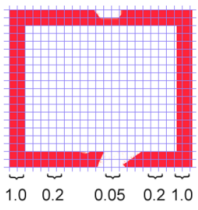
La puntuación de la relación de aspecto se basa en (Ancho de la partícula / Altura de la partícula). El ancho y la altura de la partícula se determinan usando algo llamado «rectángulo equivalente». El rectángulo equivalente es el rectángulo con longitudes laterales :math: ´x´ y ´y´ donde :math: ´2x+2y´ es igual a la partícula perímetro y :math: ´x cdot y´ es igual al área de la partícula. El rectángulo equivalente se utiliza para el aspecto de cálculo de la proporción, ya que se ve menos afectado por el sesgo del rectángulo que por el uso del límite caja. Cuando se utiliza el rectángulo del cuadro delimitador para la relación de aspecto, como el rectángulo está sesgado la altura aumenta y la anchura disminuye.

El objetivo tiene 20» (508 mm) de ancho por 12» (304,8 mm) de alto, para una relación de 1,6. La relación de aspecto detectada se compara con esta relación ideal. La puntuación de la relación de aspecto se normaliza para obtener 100 cuando la relación coincide con la del objetivo y desciende linealmente a medida que la relación varía por debajo o por encima.

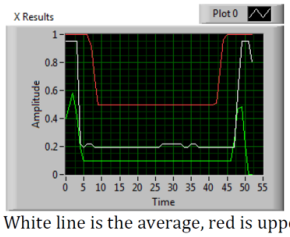
Momento

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.

Perfiles X/Y



Column averages for a particle rectangle.



White line is the average, red is upper limit, and green is lower limit..

El puntaje del borde describe si la partícula coincide con el perfil apropiado tanto en el X y las direcciones Y. Como se muestra, se calcula usando los promedios de fila y columna a través de la caja delimitadora extraída de la imagen original y comparada con una máscara de perfil. La puntuación oscila entre 0 y 100 en función del número de valores dentro de los promedios de las filas o columnas que están entre los valores límite superior e inferior.

Medidas

Si una partícula puntúa lo suficientemente bien como para ser considerada un objetivo, tiene sentido calcular algunas medidas del mundo real como la posición y la distancia. El código de ejemplo incluye estas medidas básicas, así que veamos las matemáticas involucradas para entenderlo mejor.

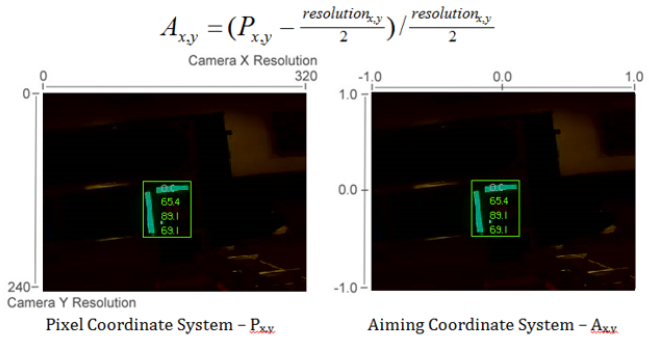
Posición

La posición del objetivo está bien descrita tanto por la partícula como por el cuadro delimitador, pero todas las coordenadas están en píxeles con el 0,0 estando en la parte superior izquierda de la pantalla y la derecha y la parte inferior de los bordes determinados por la resolución de la cámara. Este es un sistema útil para la matemática de los píxeles, pero no casi tan útil para conducir un robot; así que cambiémoslo a algo que pueda ser más útil.

Para convertir un punto del sistema de píxeles al sistema de puntería, podemos usar la fórmula mostrada debajo.

The resulting coordinates are close to what you may want, but the Y axis is inverted. This could be corrected by multiplying the point by [1,-1] (Note: this is not done in the sample code). This coordinate system is useful because it has a centered origin and the scale is similar to joystick outputs and Drive inputs.

$$A_{x,y} = \left(P_{x,y} - \frac{resolution_{x,y}}{2}\right) / \frac{resolution_{x,y}}{2}$$



Vista de campo

Puede utilizar constantes conocidas y la posición del objetivo en el plano de coordenadas para determinar la distancia, la guiñada y el cabeceo del objetivo. Sin embargo, para calcularlas, debes determinar tu FOV (campo de visión). Para determinar empíricamente el campo vertical de la vista, coloca tu cámara a una distancia determinada de una superficie plana, y mide la distancia entre la fila de píxeles más alta y la más baja.

$$\frac{1}{2}FOV_{vertical} = tan\left(\frac{\frac{1}{2}distance_y}{distance_z}\right)$$

Puede encontrar el FOV horizontal usando el mismo método, pero usando la distancia entre la primera y última columna de píxeles.

Lanzamiento y guiño

Encontrar el lanzamiento y el guiño del objetivo en relación con tu robot es simple una vez que conoces tu FOV y la ubicación de su objetivo en el sistema de coordenadas de puntería.

$$pitch = \frac{A_y}{2} FOV_{vertical}$$

$$yaw = \frac{A_x}{2} FOV_{horizontal}$$

Distancia

Si tu objetivo está a una altura significativamente diferente a la de tu robot, puedes usar constantes conocidas, como la altura física del objetivo y tu cámara, así como el ángulo en el que se monta la cámara, para calcular la distancia entre la cámara y el objetivo.

$$distance = \frac{height_{target} - height_{camera}}{tan(angle_{camera} + pitch)}$$

Otra opción es crear una tabla de búsqueda de área a distancia, o estimar el inverso constante de variación de área y distancia. Sin embargo, este método es menos preciso.

Nota: Para obtener los mejores resultados de los métodos anteriores de estimación del ángulo y la distancia, puede calibrar su cámara usando OpenCV para deshacerse de cualquier distorsión que pueda estar afectando a la precisión reproyectando los píxeles del objetivo usando la matriz de calibración

---

25.1.5 Vídeo de Lectura y Procesamiento: Clase de CameraServer

Conceptos

The cameras typically used in FRC® (commodity USB and Ethernet cameras) offer relatively limited modes of operation. In general, they provide only a single image output (typically in an RGB compressed format such as JPG) at a single resolution and frame rate. USB cameras are particularly limited as only one application may access the camera at a time.

CameraServer soporta múltiples cámaras. Este cuenta con detalles como reconexión automática cuando la cámara se desconecta, también hace imágenes desde la disponibilidad de la cámara hacia múltiples “clientes” (p.ej. El código del robot y su dashboard pueden conectarse simultáneamente).

Nombres de Cámaras

Each camera in CameraServer must be uniquely named. This is also the name that appears for the camera in the Dashboard. Some variants of the CameraServer `startAutomaticCapture()` functions will automatically name the camera (e.g. «USB Camera 0»), or you can give the camera a more descriptive name (e.g. «Intake Cam»). The only requirement is that each camera have a unique name.

Notas de Cámara USB

Uso de CPU

La CameraServer está diseñada para minimizar el uso de CPU realizando solo comprensión y operaciones de descompresión cuando sea necesario y deshabilitar automáticamente la transmisión cuando no hay clientes conectados.

Para minimizar el uso de CPU, la resolución de las dashboard debe establecerse en la misma resolución que la cámara; esto permite que CameraServer no se descomprima y recomprimir la imagen, en lugar, simplemente puede reenviar la imagen JPEG recibida desde la cámara directamente al dashboard. Es importante notar que cambiando la resolución en el dashboard no hace cambio en la resolución de la cámara; cambiando la resolución de la cámara puede ser hecho llamando setResolution() en el objeto de la cámara.

Ancho de Banda de USB

La RoboRIO solo puede transmitir y recibir cierta información al mismo tiempo sobre las interfaces de la USB. Las imágenes de la cámara requiere mucha información, y en sí es relativamente fácil reproducir las imágenes en estos límites. El error más común en el ancho de banda de USB es seleccionar un modo de video non-JPEG o corriendo una resolución muy alta, particularmente cuando múltiples cámaras están conectadas.

Arquitectura

La CameraServer consiste en dos capas, el nivel alto WPILib CameraServer clase y el nivel bajo librería cscore.

Clase de CameraServer

La clase de CameraServer (parte de WPILib) provisiona un alto nivel de interface para añadir cámaras al código del robot. También es responsable por publicar información sobre las cámaras y los servidores de cámaras a las NetworkTables para que las dashboards de la Driver Station como la LabVIEW, Dashboard y Shuffleboard pueden enlistar las cámaras y determinar dónde está localizada la transmisión de estas. Esto usa un patrón único para mantener una base de datos de todos los servidores y cámaras creadas.

Algunas funciones clave en la CameraServer son:

- startAutomaticCapture(): Agregue una cámara USB (p.ej. Microsoft LifeCam) e inicie un servidor para que pueda verse desde el dashboard.
- getVideo(): Obtenga acceso a OpenCV a la cámara. Esto le permite obtener las imágenes desde la cámara para el procesamiento de la imagen en la RoboRIO (en el código del robot).
- putVideo(): Inicie un servidor al que pueda alimentar imágenes de OpenCV. Esto le permite pasar imágenes personalizadas procesadas y/o anotadas en el dashboard.

Librería cscore

La librería cscore provisiona el menor nivel de implementación para:

- Get images from USB and HTTP cameras
- Cambiar la configuración de la cámara (p.ej. contraste y brillo)
- Cambiar los modos de vídeo de la cámara (formato del pixel, resolución y cuadros por segundo)
- Actuar como un servidor web y servir imágenes como una transmisión MJPEG estándar
- Convertir imágenes a/desde objetos Mat OpenCV para el procesamiento de la imagen

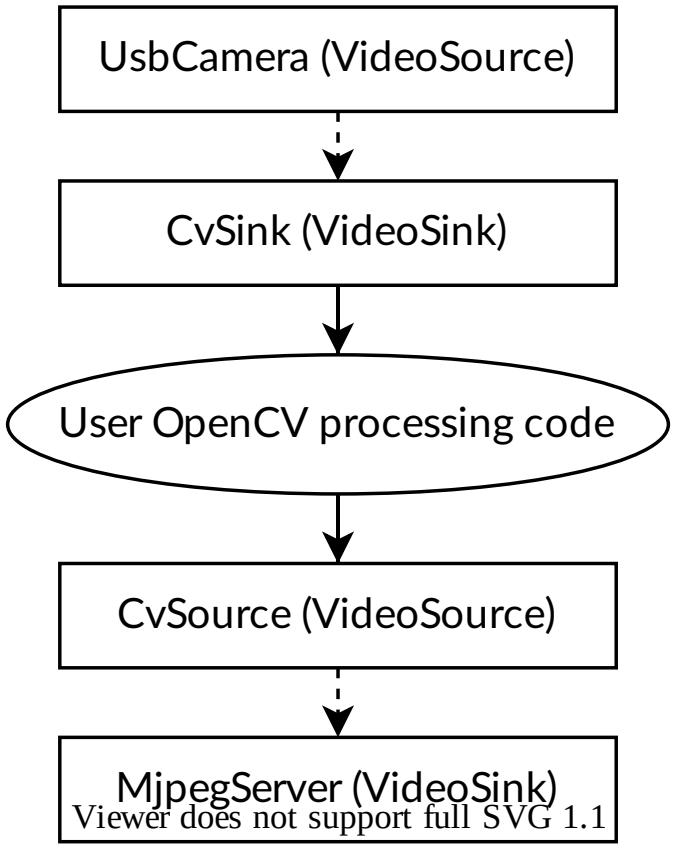
Fuentes y Receptores

La arquitectura básica de la Librería cscore a la de MJPGStreamer, con funcionalidad dividida entre fuentes y receptores. Puede haber múltiples fuentes y múltiples receptores creado y operando simultáneamente.

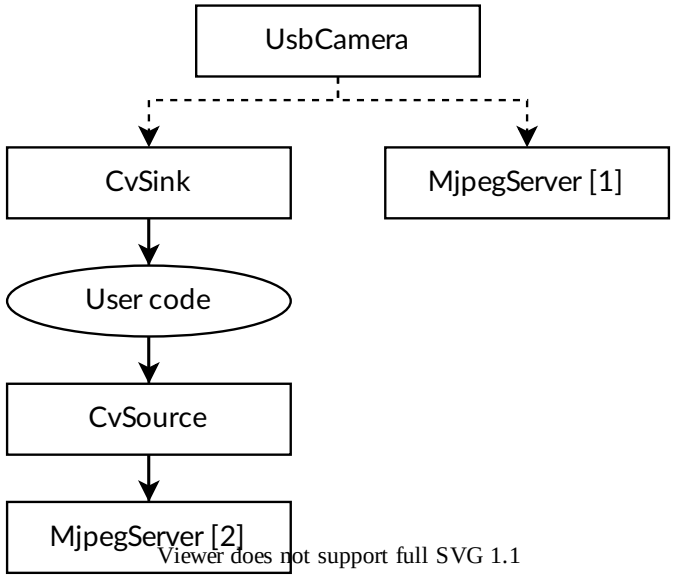
Un objeto que genera imágenes es una fuente y un objeto que acepta / consume imágenes es un receptor. El generar / consumir es desde la perspectiva de la biblioteca. Por lo tanto, las cámaras son fuentes (generan imágenes). El servidor web MJPEG es un receptor porque acepta imágenes de dentro del programa (aunque puede reenviar esas imágenes a un navegador web o dashboard). Las fuentes pueden estar conectadas a múltiples receptores, pero los receptores pueden conectarse a uno y solo una fuente. Cuando un receptor se conecta a una fuente, la biblioteca cscore se encarga de pasar cada imagen desde la fuente al sumidero.

- Las fuentes obtienen marcos individuales (como los proporcionados por una cámara USB) y disparan un evento cuando hay un nuevo marco disponible. Si ningún receptor está escuchando una fuente en particular, la biblioteca puede pausar o desconectarse de una fuente para ahorrar procesadores y recursos de I / O. La biblioteca maneja de forma autónoma las desconexiones y reconexiones de la cámara simplemente pausando y reanudando la activación de eventos (por ejemplo, una desconexión no genera nuevos marcos, no hay un error).
- Los receptores escuchan el evento de una fuente en particular, toman la última imagen y la reenvían a su destino en el formato apropiado. De manera similar a las fuentes, si un receptor particular está inactivo (por ejemplo, ningún cliente está conectado a un servidor MJPEG configurado a través de HTTP), la biblioteca puede deshabilitar partes de su procesamiento para ahorrar recursos del procesador.

El código de usuario (como el utilizado en un programa de robot FRC) puede actuar como fuente (proporcionando cuadros procesados como si fuera una cámara) o como un receptor (que recibe un cuadro para su procesamiento) a través de la fuente de OpenCV y recibir objetos. Por lo tanto, una tubería de procesamiento de imágenes que obtiene imágenes de una cámara y sirve las imágenes procesadas se ve como el siguiente gráfico:



Debido a que las fuentes pueden tener múltiples receptores conectados, la tubería puede ramificarse. Por ejemplo, la imagen de la cámara original también se puede servir conectando la fuente USBCamera a un segundo receptor MjpegServer además del CvSink, lo que da como resultado el siguiente gráfico:



Cuando la cámara captura una nueva imagen, tanto el CvSink como el MjpegServer [1] la reciben.

El gráfico anterior es lo que crea el siguiente fragmento de CameraServer:

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);
```

La implementación de CameraServer efectivamente hace lo siguiente en el nivel del cscore (con fines explicativos). El CameraServer se encarga de muchos de los detalles, como la creación de nombres únicos para todos los objetos cscore y seleccionando automáticamente los números de puerto. CameraServer también mantiene un registro único de los objetos creados para que no se destruyan si salen del alcance.

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
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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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);
```

Recuento de Referencias

Todos los objetos cscore se cuentan internamente como referencia. La conexión de un receptor a una fuente aumenta el recuento de referencias de la fuente, por lo que solo es estrictamente necesario mantener el receptor dentro del alcance. La clase CameraServer mantiene un registro de todos los objetos creados con las funciones de CameraServer, por lo que las fuentes y los receptores creados de esa manera nunca salen del alcance (a menos que explícitamente se remueva).

25.1.6 Ejemplos de visión 2017

LabVIEW

El ejemplo de Visión de 2017 de LabVIEW se incluye con los demás ejemplos de LabVIEW. Desde la pantalla de inicio, haga clic en Support->Find FRC® Examples o desde cualquier otra ventana de LabVIEW, haga clic en Help->Find Examples y localice la carpeta Vision para encontrar el ejemplo de Visión 2017. Las imágenes se incluyen con el ejemplo.

25.2 Visión con WPILibPi

25.2.1 Un video de demostración del uso de WPILibPi con el Raspberry Pi

Nota: El video menciona a FRCVision, que es el nombre antiguo de WPILibPi.

En la «RSN Spring Conference, Presented by WPI» en 2020, Peter Johanson del equipo WPILib dio una presentación acerca de la Visión con Raspberry PI en FRC®.

El link de la presentación estará aquí <https://docs.google.com/presentation/d/1yViG-k5PS4jWVrxY3o7eD8h5YXnK9Deqm6RsUXfCnRA/edit?usp=sharing>`_.

25.2.2 Uso de un coprocesador para el procesamiento de la visión

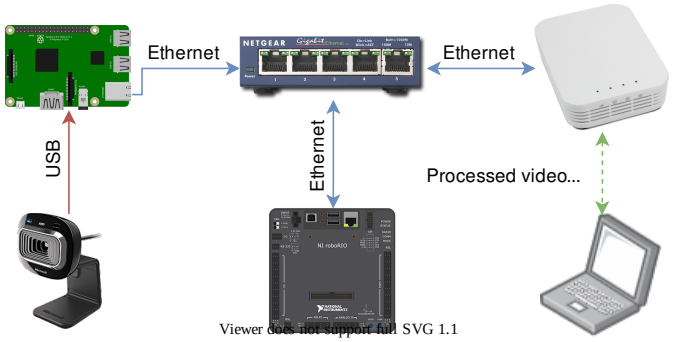
El procesamiento de Visión utilizando librerías como OpenCV para reconocer objetivos de campo o piezas de juego a menudo puede ser un proceso intensivo de CPU. A menudo la carga no es demasiado significativa y el procesamiento puede ser manejado fácilmente por la roboRIO. En casos donde hay más transmisiones de cámara o el procesamiento de imágenes es complejo, es deseable quitar la roboRIO colocando el código y la conexión de la cámara en un procesador diferente. Hay varias opciones de procesadores que son populares en FRC® como el Raspberry PI, el Kangaroo basado en inteligencia, la LimeLight para lo último en simplicidad, o para un código de visión más complejo, un acelerador de gráficos como uno de los modelos nVidia Jetson.

Estrategia

En general, la idea es configurar el coprocesador con el software requerido que generalmente incluye:

- OpenCV- la biblioteca de visión por computadora de código abierto
- *NetworkTables* - to commute the results of the image processing to the roboRIO program
- Librería del Camera Sever- para manejar las conexiones de la cámara y publicar secuencias que pueden ser vistas en un dashboard
- La librería de idiomas para cualquier lenguaje de computadora utilizado para el programa de visión.
- El programa de visión real que hace la detección de objetos.

The coprocessor is connected to the roboRIO network by plugging it into the extra ethernet port on the network router or, for more connections, adding a small network switch to the robot. The cameras are plugged into the coprocessor, it acquires the images, processes them, and publishes the results, usually target location information, to NetworkTables so it is can be consumed by the robot program for steering and aiming.



Transmitir datos de la cámara al dashboard

A menudo es deseable simplemente transmitir los datos de la cámara al dashboard a través de la red del robot. En este caso, se pueden enviar una o más conexiones de cámara a la red y verlas en un dashboard de instrumentos como Shuffleboard o un navegador web. Usar Shuffleboard tiene la ventaja de tener controles sencillos para establecer la resolución de la cámara y la velocidad de bits, así como integrar las transmisiones de la cámara con otros datos enviados desde el robot.

También es posible procesar imágenes y agregar anotaciones a la imagen, como líneas de destino o cuadros que muestran lo que ha detectado el código de procesamiento de imagen y luego enviarlo al dashboard para que sea más fácil para los operadores ver una imagen clara de lo que hay alrededor del robot.

25.2.3 Uso de Raspberry Pi para FRC

Una de las opciones de coprocesador más populares es la Raspberry Pi porque:

- Bajo costo- alrededor de $ 35
- Alta disponibilidad: es fácil encontrar Raspberry Pis de varios proveedores, incluidos Amazon
- Muy buen rendimiento -el Raspberry Pi 3b + actual tiene las siguientes especificaciones:
- Especificaciones técnicas: - Broadcom BCM2837BO 64 bit ARMv8 QUAD Core A53 64bit Procesador alimentado por una sola placa computora que funciona a 1,4 GHz - 1 GB de RAM - BCM43143 WiFi a bordo - Bluetooth Low Energy (BLE) a bordo - GPIO extendido de 40 pines - 4 x puertos USB2 - Salida estéreo de 4 polos y puerto de video compuesto - HDMI de tamaño completo - Puerto de cámara CSI para Conexión de la Raspberry - Cámara Pi - Puerto de pantalla DSI para conectar la Raspberry - Pantalla táctil Pi - Puerto MicroSD para cargar su sistema operativo y almacenar datos: fuente de alimentación micro USB conmutada actualizada (ahora admite hasta 2.5 amperios.

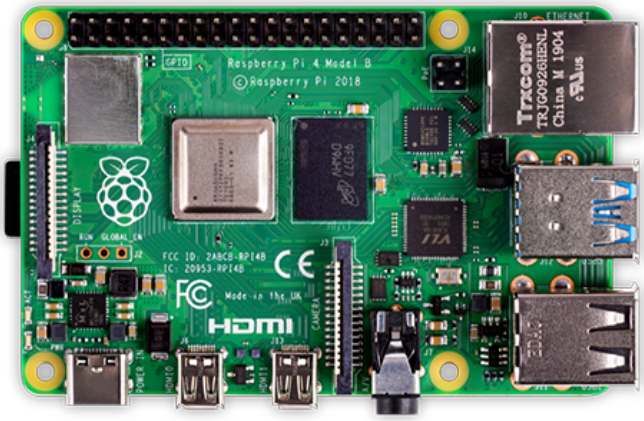


Imagen de Raspberry Pi preconstruida

Para hacer que el uso de la Raspberry Pi sea lo más fácil posible para los equipos, se proporciona una Raspberry Pi imagen. La imagen puede copiarse en una tarjeta micro SD, insertarse en el Pi y arrancarse. Por defecto admite:

- Una configuración de interfaz web para las funciones más comunes
- Admite un número arbitrario de transmisiones de cámara (el valor predeterminado es uno) que se publican en la interfaz de red
- OpenCV, *NetworkTables*, Camera Server y bibliotecas de lenguaje para programas personalizados en C++, Java y Python

Si el único requisito es transmitir una o más cámaras a la red (y al dashboard) entonces no se requiere programación y se puede configurar completamente a través de la interfaz web.

La siguiente sección analiza cómo instalar la imagen en una tarjeta flash y arrancar la Pi.

25.2.4 Lo necesario para ejecutar la imagen Pi

Para comenzar a usar Raspberry Pi como un coprocesador de vídeo o imagen, necesita lo siguiente:

- Un Raspberry Pi 3 B, Raspberry Pi 3 B + o un Raspberry Pi 4 B
- Una tarjeta micro SD de al menos 8 GB para contener todo el software proporcionado, con una clase de velocidad recomendada de 10 (10MB/s)
- Un cable de ethernet para conectar el Pi a su red roboRIO
- Un cable de alimentación micro USB para conectar al módulo regulador de voltaje (VRM) en su robot. Se recomienda utilizar la conexión VRM para la alimentación en lugar de energizarla desde uno de los puertos USB roboRIO para mayor confiabilidad
- Una computadora portátil que pueda escribir la tarjeta MicroSD, ya sea usando un dongle USB (preferido) o una SD al adaptador MicroSD que se envía con la mayoría de las tarjetas MicroSD

Se muestra una USB que no es cara, que escribirá la imagen de FRC® en la tarjeta MicroSD.

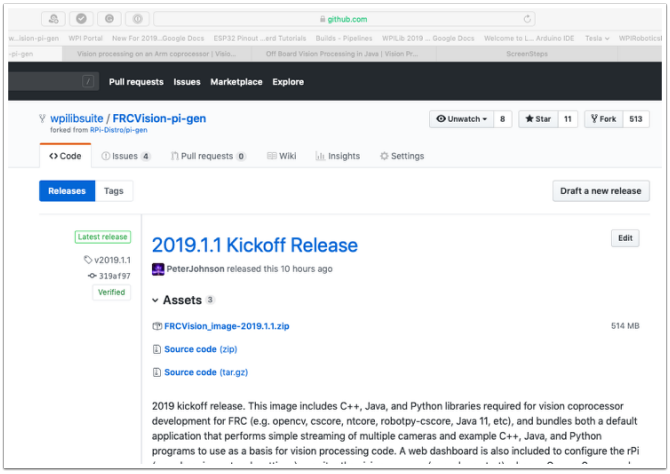
25.2.5 Instalación de la imagen en su tarjeta MicroSD

Obteniendo la imagen FRC Raspberry PI

La imagen está almacenada en la página de lanzamiento de GitHub para el repositorio WPILibPi.

Además de las instrucciones de esta página, consulte la documentación en la página web de GitHub (abajo).

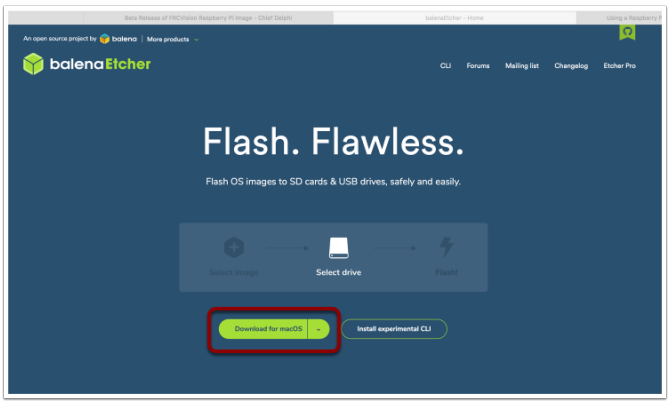
La imagen es bastante grande, así que tenga una conexión rápida a Internet cuando la descargue. Siempre use el lanzamiento más reciente de la parte superior de la lista de lanzamientos.

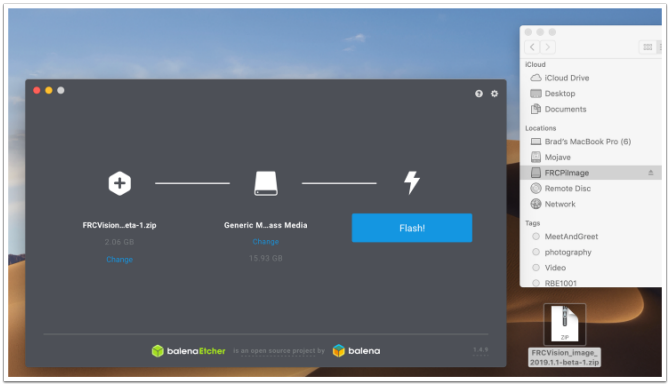


Copie la imagen a su tarjeta MicroSD

Download and install Etcher to image the micro SD card. The micro SD card needs to be at least 8 GB. A micro SD to USB dongle works well for writing to micro SD cards.

Escriba la tarjeta MicroSD con la imagen usando Etcher seleccionando el archivo zip como fuente, su tarjeta SD como destino y haga clic en «Flash». Espere que el proceso toma alrededor de 3 minutos en una laptop bastante rápida.





Probando la Raspberry Pi

1. Coloque la tarjeta micro SD en un rPi 3 y aplique energía.
2. Conecte el rPi 3 ethernet a una LAN o PC. Abra un navegador web y conéctese a “http://wpilibpi.local/”. para abrir el dashboard . En el primer arranque, se podrá escribir en el sistema de archivos, pero los arranques posteriores serán de solo lectura de forma predeterminada, por lo que es necesario hacer clic en el botón «writable» para realizar cambios.

Iniciar sesión en la Raspberry PI

La mayoría de las tareas con el rPi se pueden realizar desde la interfaz de la consola web. A veces para uso avanzado como el desarrollo de programas en el rPi es necesario iniciar sesión. Para iniciar sesión, utilice la contraseña predeterminada de Raspberry PI:

```
Username: pi
 Password: raspberry
```

25.2.6 La Raspberry Pi

Consola FRC

La imagen de FRC® para el Raspberry PI incluye una consola que se puede ver en cualquier navegador que hace fácil:

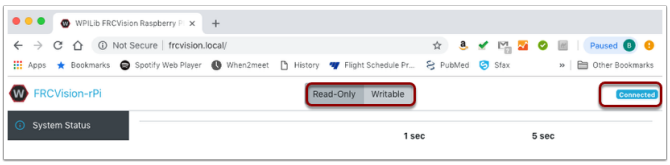
- Ver el estado de Raspberry PI
- Ver el estado del proceso en segundo plano ejecutando la cámara
- Ver o cambiar la configuración de red
- Ver cada cámara conectada al rPI y agregar cámaras adicionales
- Cargar un nuevo programa de visión en el rPI

Configurar el rPI para que sea de solo lectura en lugar de escribible

El rPI normalmente está configurado en Solo lectura, lo que significa que el sistema de archivos no se puede cambiar. Esto garantiza que si se corta la alimentación sin apagar primero el rPi, el sistema de archivos no se apagará. Cuando se cambia la configuración (siguientes secciones), la nueva configuración no se puede guardar mientras el sistema de archivos rPI está configurado como de solo lectura. Se proporcionan botones que permiten el sistema de archivos para cambiar de solo lectura a escritura y viceversa cada vez que se realicen cambios. Si los otros botones que cambian la información almacenada en el rPI no se pueden presionar, verifique el estado de solo lectura del sistema.

Estado de la conexión de red al rPI

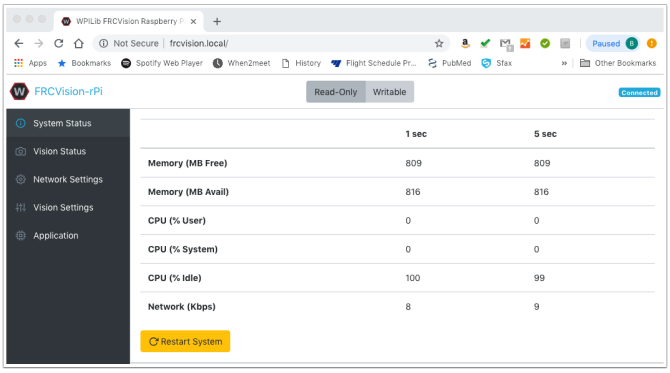
Hay una etiqueta en la esquina superior derecha de la consola que indica si el rPi está actualmente conectado. Cambiará de Conectado a Desconectado si ya no hay una conexión red a la rPi.



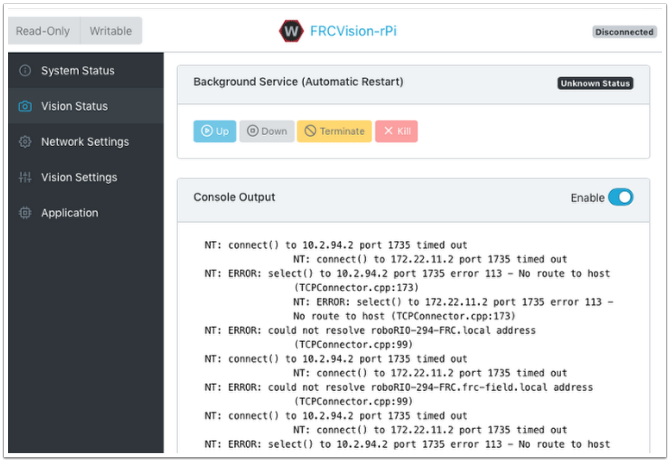
Estado de la visión

El estado del sistema muestra lo que está haciendo la CPU en el rPI en cualquier momento. Hay dos columnas de valores de estado, al ser un promedio de 1 segundo y el otro un promedio de 5 segundos. Lo que se muestra es:

- 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



Estado de la visión



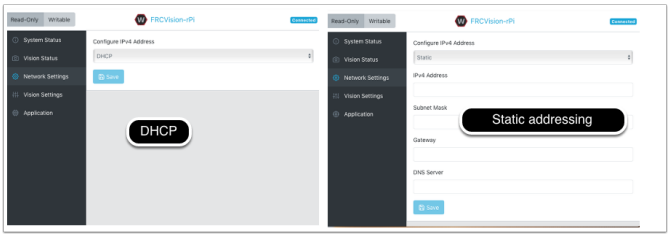
Permite monitorizar la tarea que está ejecutando el código de la cámara en el rPI, ya sea uno de los programas por defecto o su propio programa en Java, C++ o Python. También puede habilitar y ver la salida de la consola para ver los mensajes procedentes del servicio de cámara en segundo plano. En este caso hay una serie de mensajes sobre la imposibilidad de conectarse a *NetworkTables* (NT: connect()) porque en este ejemplo el rPI está simplemente conectado a un portátil sin un servidor de NetworkTables en ejecución (normalmente la roboRIO.)

Configuración de la red

La configuración de red rPI tiene opciones para conectarse al PI:

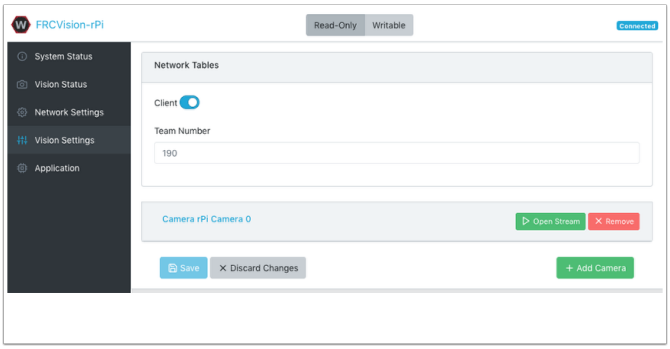
- *DHCP* - the default name resolution usually used by the roboRIO. The default name is wpilibpi.local.
- Estática - donde una dirección IP fija, una máscara de red y la configuración del enrutador se completan explícitamente
- DHCP con recuperación estática - DHCP con recuperación estática - el PI intentará obtener una dirección IP a través de DHCP, pero si no puede encontrar un servidor DHCP, utilizará la IP estática proporcionando dirección y parámetros

La imagen de arriba muestra la configuración de DHCP y direccionamiento IP estático. El nombre mDNS para el rPi siempre debería funcionar independientemente de las opciones



seleccionadas anteriormente.

Configuraciones de visión

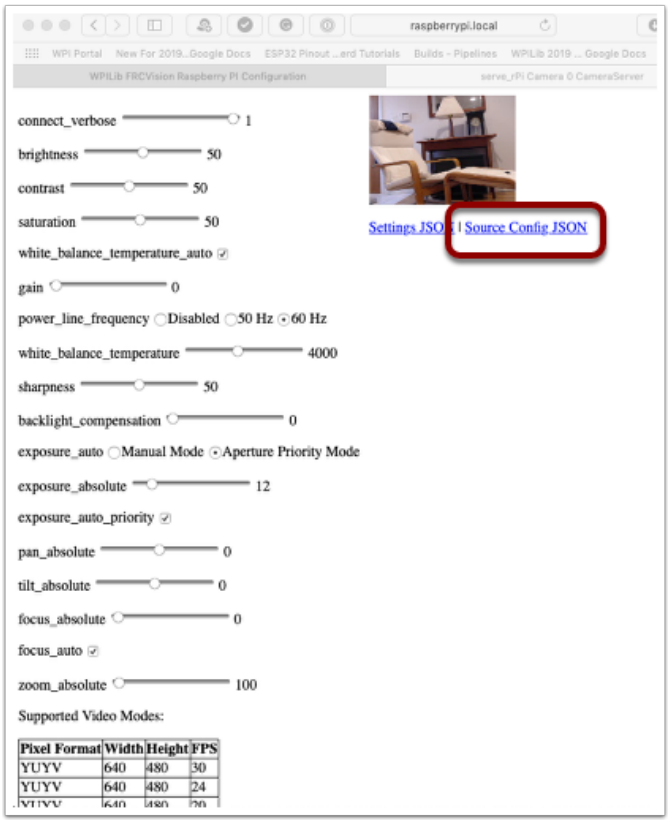


The Vision Settings are to set the parameters for each camera and whether the rPI should be a NetworkTables client or server. There can only be one server on the network and the roboRIO is always a server. Therefore when connected to a roboRIO, the rPI should always be in client mode with the team number 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).

Para ver y manipular todos los ajustes de la cámara, haga clic en la cámara en cuestión. En este caso la cámara se llama «Camera rPi Camera 0» y al hacer clic en el nombre se muestra la actual vista de cámara y la configuración asociada.

La manipulación de la configuración de la cámara se refleja en la vista actual de la cámara. El fondo de la página muestra todos los modos de cámara posibles (combinaciones de ancho, alto y velocidad de fotogramas) que son compatibles con esta cámara.

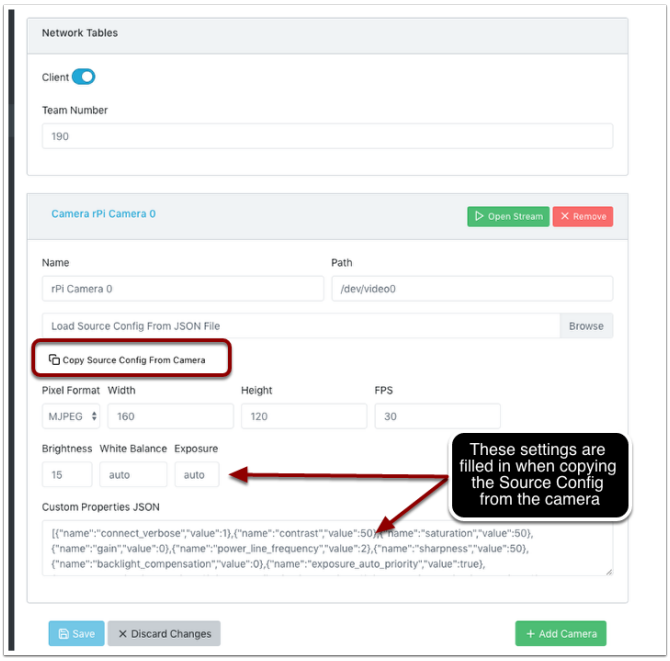
Nota: Si la imagen de la cámara no es visible en la pantalla *Open Stream* , compruebe los modos de vídeo compatibles en la parte inferior de la página. A continuación, vuelva a “Vision Settings” y haga clic en la cámara en cuestión y verifique que el formato de píxeles, la anchura, la altura y los FPS aparecen en los modos de video admitidos.



Conseguir que la configuración actual persista durante los reinicios

El rPi cargará todos los ajustes de la cámara al inicio. La edición de la configuración de la cámara en el la pantalla de arriba es temporal. Para que los valores persistan, haga clic en «Cargar configuración de origen desde Cámara » y la configuración actual se completará en los campos de configuración de la cámara. Luego haga clic en «Guardar» en la parte inferior de la página. Nota: debe configurar el sistema de archivos de escritura en orden para guardar la configuración. *El botón de escritura se encuentra en la parte superior de la página.*

Hay algunos valores de configuración de la cámara de uso común que se muestran en la configuración de la cámara (arriba). Estos valores de Brillo, Balance de blancos y Exposición se cargan en la cámara antes de Se aplica el archivo JSON del usuario. Entonces, si un archivo JSON de usuario contiene esa configuración, se sobrescribirán los del campo de texto.



Aplicación

La pestaña Aplicación muestra la aplicación que se está ejecutando actualmente en el rPi.

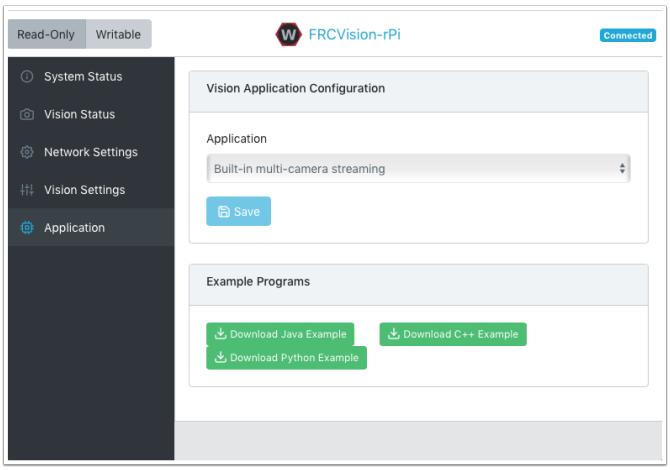
Flujos de trabajo de Visión

Hay un programa de visión de muestra que usa OpenCV en cada uno de los lenguajes compatibles, C++, Java o Python. Cada programa de muestra puede capturar y transmitir video desde el rPi. Además, las muestras tienen algunos OpenCV mínimos. Todos están configurados para ser extendidos para reemplazar el código de muestra de OpenCV proporcionado con el código necesario para la aplicación del robot. La pestaña Aplicación de rPI admite varios flujos de trabajo de programación:

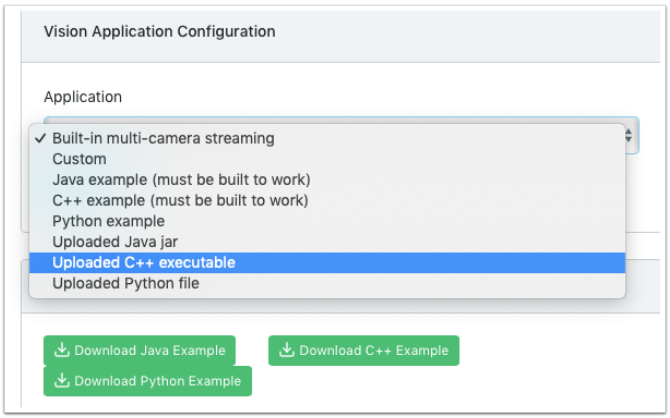
- Transmita una o más cámaras desde el rPi para consumo en la computadora de la driver station y se muestra usando ShuffleBoard
- Edite y cree uno de los programas de muestra (uno para cada idioma: Java, C++ o Python) en el rPi usando las cadenas de herramientas incluidas
- Descargue un programa de muestra para el idioma elegido y edítelo y compílelo en su computadora de desarrollo. Luego cargue ese programa construido de nuevo en el rPi
- Haga todo usted mismo utilizando aplicaciones y scripts completamente personalizados (probablemente basados en una de las muestras)

La aplicación en ejecución se puede cambiar seleccionando una de las opciones en el menú desplegable. Las opciones son:

- 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.
- Aplicación personalizada que no carga nada en el rPi y supone que el desarrollador quiere tener un programa y script personalizados.

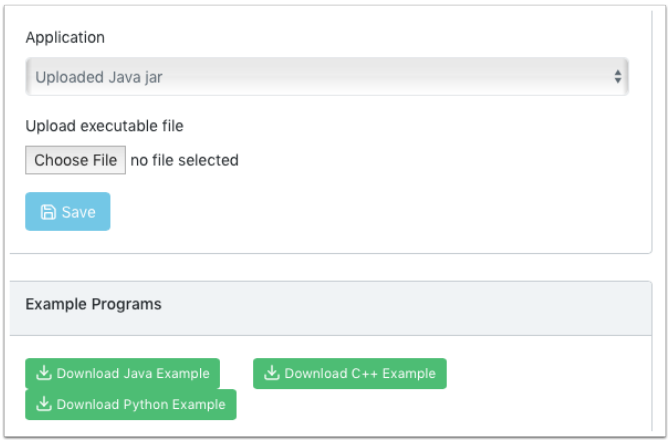


- Programas de muestra preinstalados de Java, C ++ o Python que se pueden editar en su propia solicitud.
- Programa cargado en Java, C ++ o Python. Los programas Java requieren un archivo `.jar` con el programa compilado y los programas C ++ requieren que se cargue un ejecutable rPi en el rPI.



Al seleccionar una de las opciones de Carga, se presenta un selector de archivos donde el archivo jar, ejecutable o el programa Python se puede seleccionar y cargar en el rPi. En la siguiente imagen Java jar subida es elegida y el botón «Elegir archivo» seleccionará un archivo y hará clic en el botón «Guardar» cargará el archivo seleccionado.

Nota: para guardar un nuevo archivo en el rPi, el sistema de archivos debe configurarse para que se pueda escribir usando el botón «Writable» en la esquina superior izquierda de la página web. Después de guardar el nuevo archivo, configure el archivo sistema de nuevo a «Solo lectura» para que esté protegido contra cambios accidentales.



25.2.7 Usando la CameraServer

Capturando marcos de CameraServer

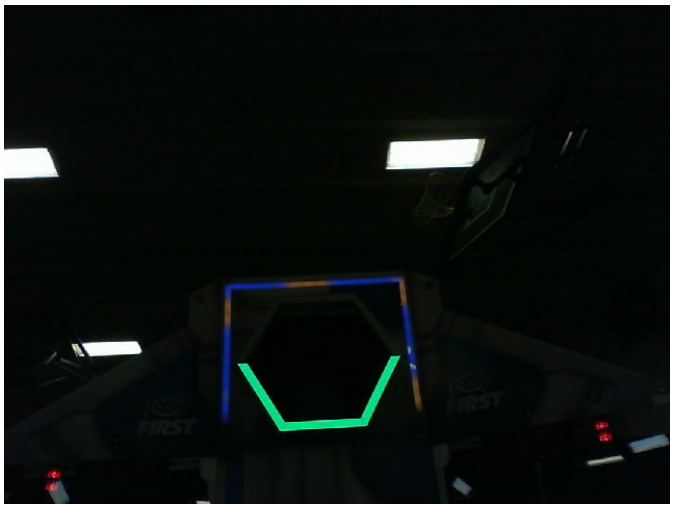
La imagen WPILibPi viene con todas las bibliotecas necesarias para crear su propio sistema de procesamiento de visión. Para obtener el fotograma actual de la cámara, puede utilizar la biblioteca CameraServer. Para obtener información sobre CameraServer, consulte *Vídeo de Lectura y Procesamiento: Clase de 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
```

Nota: OpenCV reads in the image as **BGR**, not **RGB** for historical reasons. Use `cv2.cvtColor` if you want to change it to RGB.

A continuación se muestra un ejemplo de una imagen que podría capturarse desde CameraServer.



Envío de marcos a CameraServer

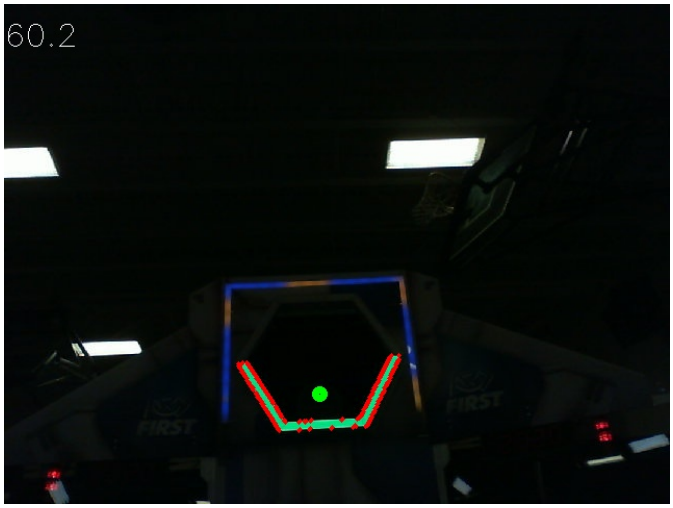
A veces, es posible que desee enviar fotogramas de video procesados a la instancia de CameraServer para fines de depuración o para ver en una aplicación del dashboard como Shuffleboard.

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)
```

Como ejemplo, el código de procesamiento podría delinear el objetivo en rojo y mostrar las esquinas en amarillo para fines de depuración.

A continuación se muestra un ejemplo de una imagen completamente procesada que se enviaría de vuelta a CameraServer y se muestra en la computadora Driver Station.



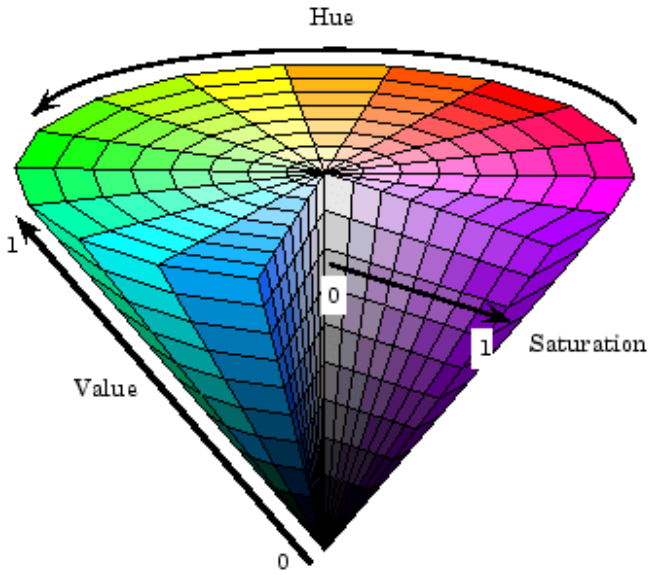
25.2.8 Descomponer los colores de una imagen

Para convertir una imagen en color, como la capturada por su cámara, en una imagen binaria, con el objetivo como el «primer plano», necesitamos poner el umbral de la imagen usando el tono, saturación y valor de cada píxel.

El Modelo HSV

A diferencia de RGB, HSV le permite no sólo filtrar según los colores de los píxeles, sino también por la intensidad del color y el brillo.

- Tono: mide el color del píxel.
- Saturación: mide la intensidad del color del píxel.
- Value: Measures the brightness of the pixel.



Puede usar OpenCV para convertir una matriz de imagen BGR a HSV.

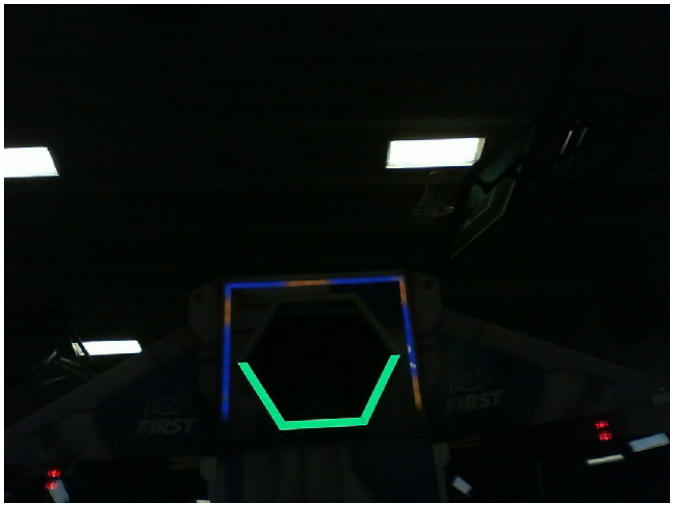
PY

```
hsv_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2HSV)
```

Nota: El rango de tono de OpenCV es de 1 ° a 180 ° en lugar del común de 1 ° a 360 °. A fin de que convierta un valor de tono común a OpenCV, dividir por 2.

Descomponer los colores

Utilizaremos esta imagen de campo como ejemplo para todo el proceso de procesamiento de imágenes.



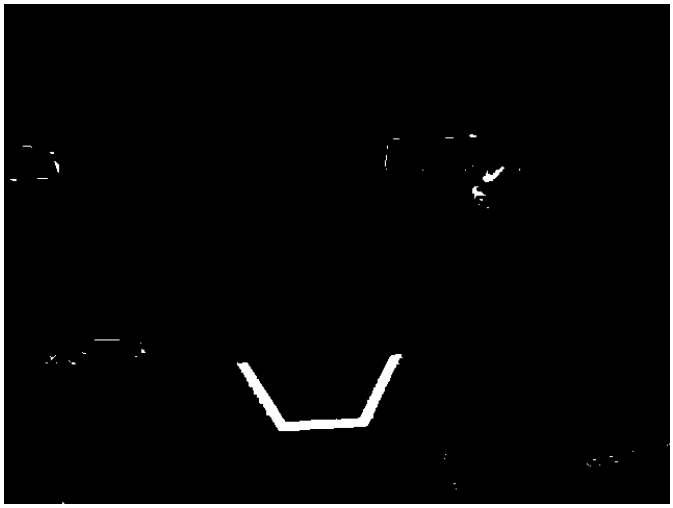
Al descomponer en colores la imagen usando HSV, usted puede separar la imagen en: objetivo de visión (primer plano) y todo lo demás (segundo plano). El siguiente código de ejemplo convierte una imagen HSV a una imagen binaria al descomponer en colores con valores HSV.

PY

```
binary_img = cv2.inRange(hsv_img, (min_hue, min_sat, min_val), (max_hue, max_sat, max_
↪val))
```

Nota: Es posible que estos valores tengan que ajustarse según el lugar, ya que la iluminación ambiental puede diferir según los lugares. Se recomienda permitir la edición de estos valores a través de NetworkTables para facilitar la edición sobre la marcha.

Después de la descomposición, su imagen debería verse así.



Como puede ver, el proceso de descomposición puede no ser 100% limpio. Puedes usar operaciones morfológicas para lidiar con el ruido.

25.2.9 Operaciones morfológicas

A veces, después de descomponer su imagen, tiene ruido no deseado en su imagen binaria. Las operaciones morfológicas pueden ayudar a eliminar ese ruido de la imagen.

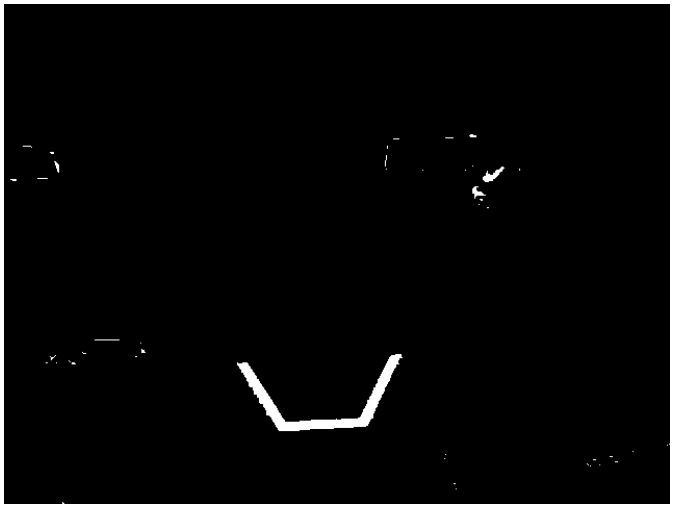
Núcleo

El núcleo es una forma simple donde el origen se superpone en cada píxel de valor 1 de la imagen binaria. OpenCV limita el núcleo a una matriz NxN donde N es un número impar. El origen del núcleo es el centro. Un núcleo común es

$$kernel = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Diferentes núcleos pueden afectar la imagen de manera diferente, como erosionarse o dilatarse verticalmente.

Como referencia, esta es nuestra imagen binaria que creamos:

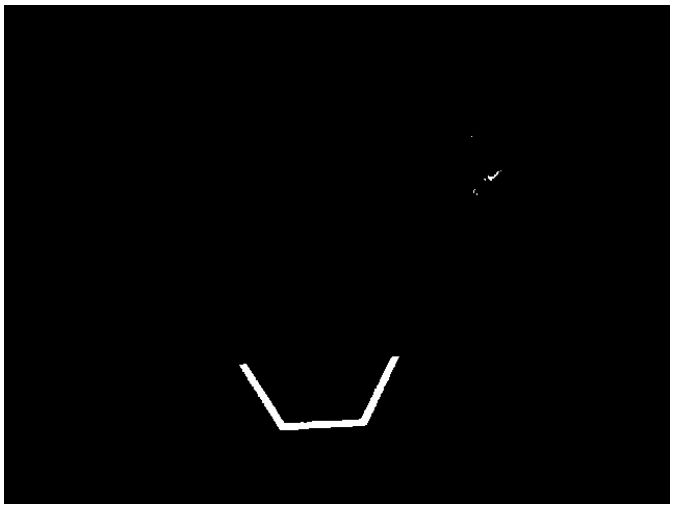


Erosión

La erosión en la visión por computadora es similar a la erosión en el suelo. Le quita las fronteras de objetos en primer plano. Este proceso puede eliminar el ruido del fondo.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.erode(binary_img, kernel, iterations = 1)
```



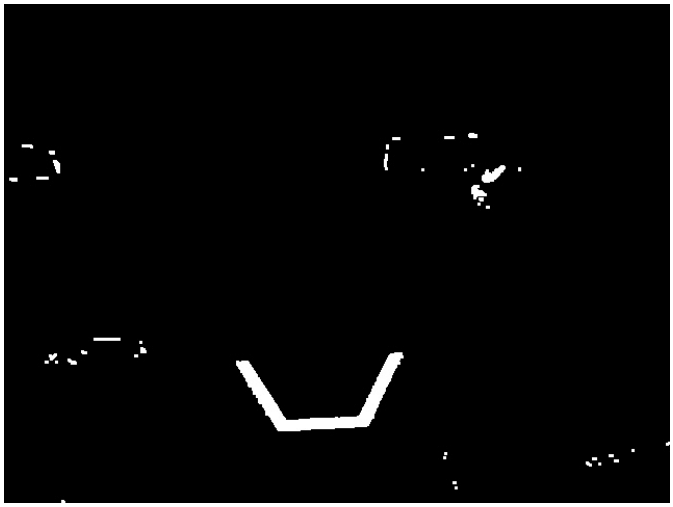
Durante la erosión, si los píxeles superpuestos del núcleo no están contenidos completamente por los píxeles de la imagen binaria, se elimina el píxel en el que se superpuso.

Dilatación

La dilatación es opuesta a la erosión. En lugar de alejarse de las fronteras, se agrega a ellos. Este proceso puede eliminar pequeños agujeros dentro de una región más grande.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.dilate(binary_img, kernel, iterations = 1)
```



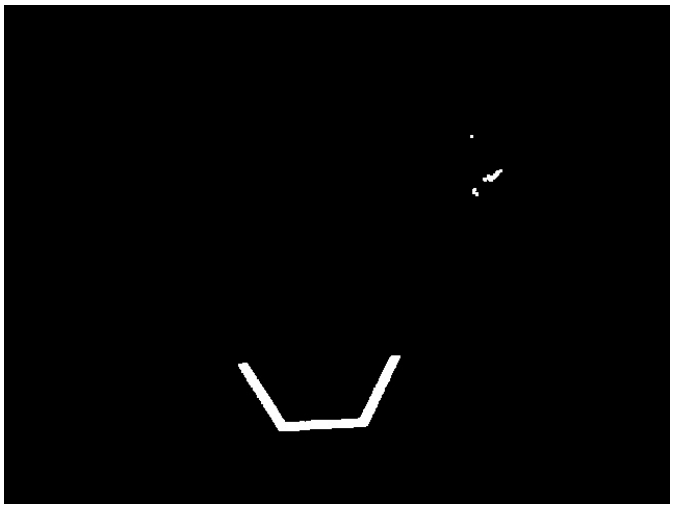
Durante la dilatación, cada píxel de cada núcleo superpuesto se incluye en la dilatación.

Apertura

La apertura es erosión seguida de dilatación. Este proceso elimina el ruido sin afectar el forma de características más grandes.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel)
```



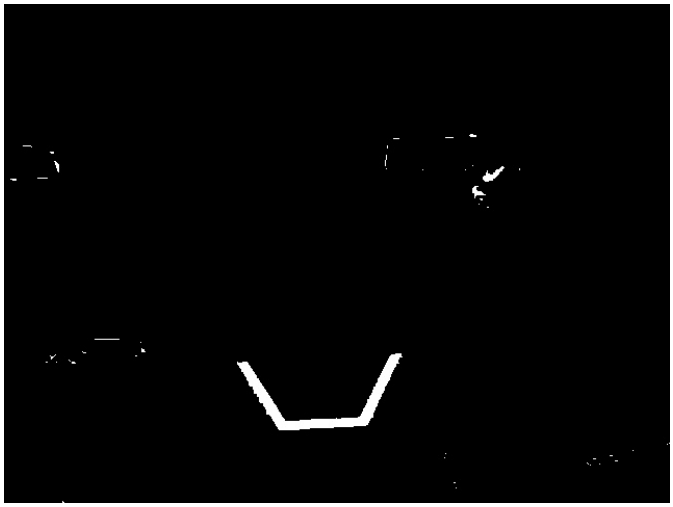
Nota: In this specific case, it is appropriate to do more iterations of opening in order to get rid of the pixels in the top right.

Clausura

El cierre es dilatación seguida de erosión. Este proceso elimina pequeños agujeros o roturas sin afectar la forma de las características más grandes.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, kernel)
```



25.2.10 Trabajar con Contornos

Después de reducir y eliminar el ruido con operaciones morfológicas, ahora está listo para usar el método findContours de OpenCV. Este método le permite generar contornos basados en tu imagen binaria.

Encontrar y filtrar Contornos

PY

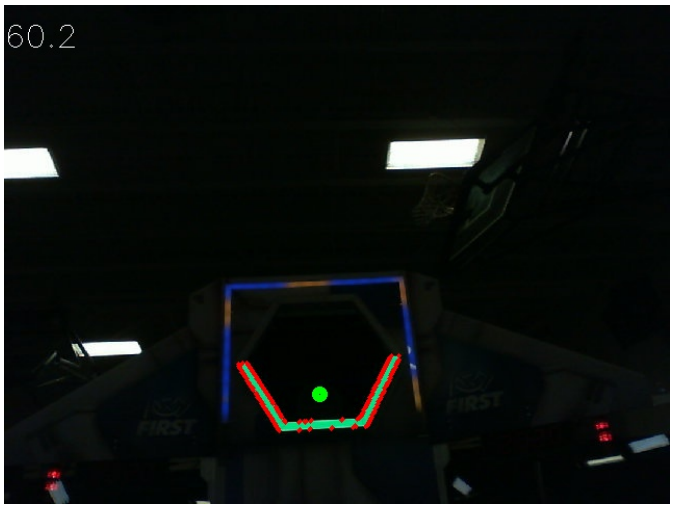
```
_, contours, _ = cv2.findContours(binary_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
↪SIMPLE)
```

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 dibuja el contorno que acaba de encontrar, debería verse más o menos así:



Extracción de información de Contornos

Ahora que ha encontrado los contorno(s) que desea, ahora desea obtener información sobre estos como el centro, las esquinas y la rotación.

Centro

PY

```
rect = cv2.minAreaRect(contour)
center, _, _ = rect
center_x, center_y = center
```

Esquinas

PY

```
corners = cv2.convexHull(contour)
corners = cv2.approxPolyDP(corners, 0.1 * cv2.arcLength(contour), True)
```

Rotación

PY

```
_, _, rotation = cv2.fitEllipse(contour)
```

Para obtener más información sobre cómo puede usar estos valores, consulte *Medidas*

Publicar a NetworkTables

Puede usar NetworkTables para enviar estas propiedades a la Driver Station y RoboRIO. Se podría realizar un procesamiento adicional en la Raspberry Pi o en el propio RoboRIO.

PY

```
import ntcore

nt = ntcore.NetworkTableInstance.getDefault().getTable('vision')

#
# Initialization code here
#

while True:

   #
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   # Image processing code here
   #

   nt.putNumber('center_x', center_x)
   nt.putNumber('center_y', center_y)
```

25.2.11 Ejemplo de visión básica

Este es un ejemplo de una configuración de visión básica que publica la ubicación del objetivo en el sistema de coordenadas de puntería descrito *aquí* en NetworkTables, y utiliza CameraServer para mostrar un límite rectángulo del contorno detectado. Este ejemplo mostrará la velocidad de fotogramas del código de procesamiento en las imágenes enviadas a 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']

   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()
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      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)

            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 ¿Qué son los AprilTags?

Los AprilTags son un sistema de tags visuales desarrollado por investigadores de la Universidad de Michigan para proveer gastos generales bajos y localización de alta precisión para diferentes aplicaciones.

Información adicional sobre el sistema de etiquetas y sus creadores se puede encontrar en su sitio web <https://april.eecs.umich.edu/software/apriltag>`__. Este documento intenta resumir el contenido para propósitos relacionados con la robótica FIRST.

Aplicación en FRC

En el contexto de FRC, las AprilTags son útiles para ayudar a su robot a saber dónde está en el campo, para que pueda alinearse a si mismo a alguna posición para anotar.

Los AprilTags han estado en desarrollo desde 2011, y han sido refinados sobre los años para aumentar la robustez y velocidad de detección.

Starting in 2023, FIRST is providing a number of tags, scattered throughout the field, each at a known *pose*.

In 2024, the tag family was updated to the 36h11 family.

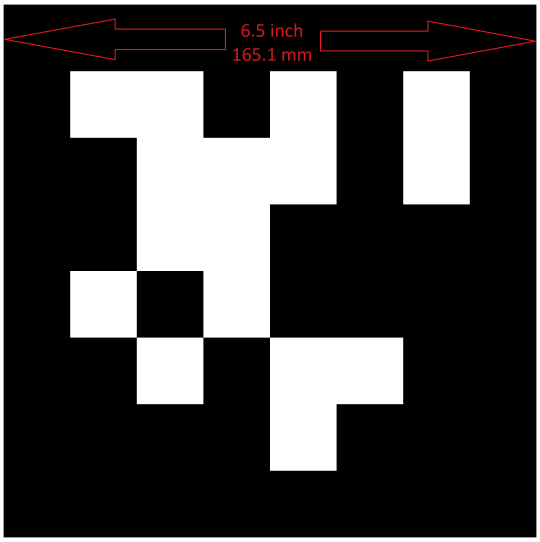
The AprilTag library implementation defines standards on how sets of tags should be designed. Some of the possible tag families are described here.

FIRST has chosen the 36h11 family for 2024. This family of tags is made of a 6x6 grid of pixels, each representing one bit of information. An additional black and white border must be present around the outside of the bits.

While there are $2^{36} = 68,719,476,736$ theoretical possible tags, only 587 are actually used. These are chosen to:

1. Be robust against some bit flips (IE, issues where a bit has its color incorrectly identified).
2. Not involve «simple» geometric patterns likely to be found on things which are not tags. (IE, squares, stripes, etc.)
3. Ensure the geometric pattern is asymmetric enough that you can always figure out which way is up.

All tags will be printed such that the tag's main «body» is 6.5 inches in length.



For home usage, tag files may be printed off and placed around your practice area. Mount them to a rigid backing material to ensure the tag stays flat, as the processing algorithm assumes the tags are flat.

Software Support

The main repository for the source code that detects and decodes AprilTags is located here.

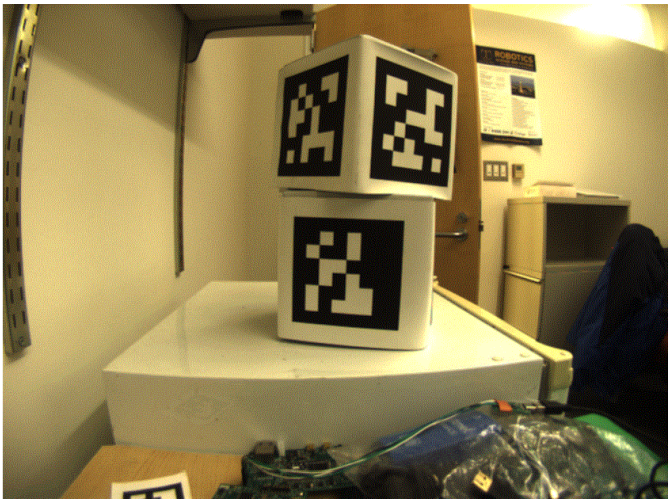
WPILib has forked the repository to add new features for FRC. These include:

1. Building the source code for common FRC targets, including the roboRIO and Raspberry Pi.
2. Adding Java Native Interface (JNI) support to allow invoking its functionality from Java
3. Gradle & Maven publishing support

Processing Technique

While most FRC teams should not have to implement their own code to identify AprilTags in a camera image, it is useful to know the basics of how the underlying libraries function.

Original Image



An image from a camera is simply an array of values, corresponding to the color and brightness of each pixel.

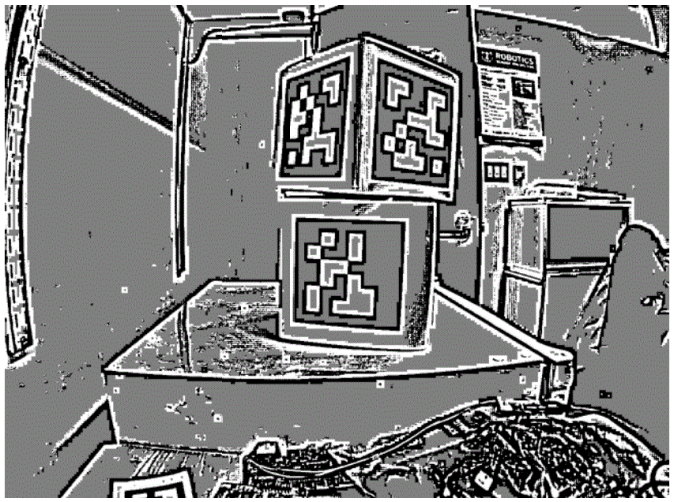
Remove Colors

The first step is to convert the image to a grey-scale (brightness-only) image. Color information is not needed to detect the black-and-white tags.

Decimate

The next step is to convert the image to a lower resolution. Working with fewer pixels helps the algorithm work faster. The full-resolution image will be used later to refine early estimates.

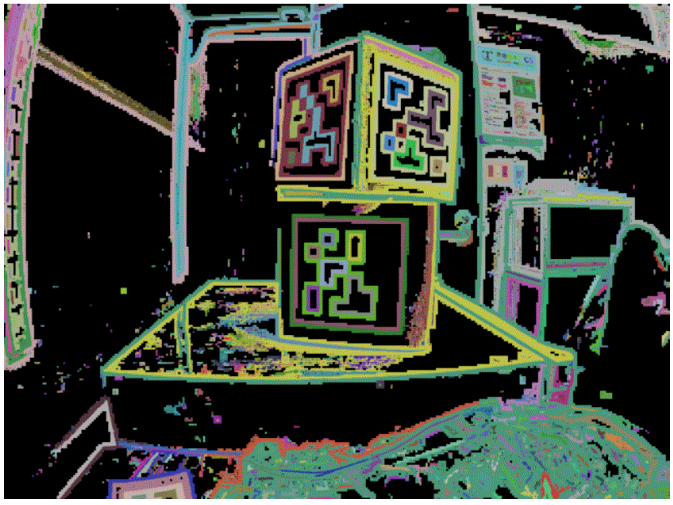
Adaptive Threshold



An adaptive threshold algorithm is run to classify each pixel as «definitely light», «definitely dark», or «not sure».

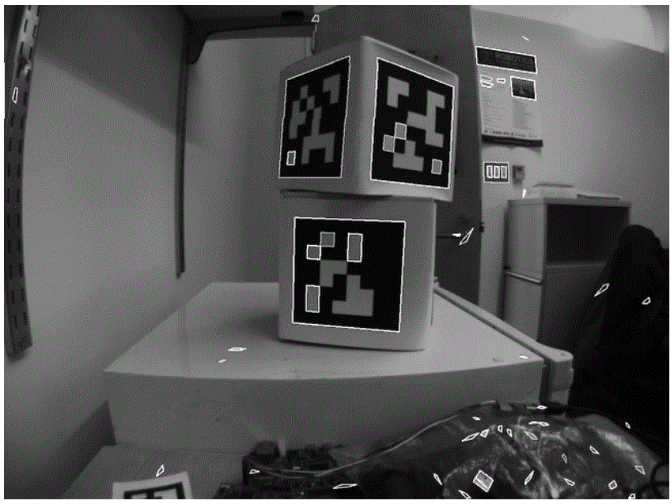
The threshold is calculated by looking at the pixel's brightness, compared to a small neighborhood of pixels around it.

Segmentation



Next, the known pixels are clumped together. Any clumps which are too small to reasonably be a meaningful part of a tag are discarded.

Quad Detection



An algorithm for fitting a quadrilateral to each clump is now run:

- Identify likely «corner» candidates by pixels which are outliers in both dimensions.
- Iterate through all possible combinations of corners, evaluating the fit each time
- Pick the best-fit quadrilateral

Given the set of all quadralaterals, Identify a subset of quadrilaterals which is likely a tag.

A single large exterior quadrilateral with many interior quadrilateral is likely a good candidate.

If all has gone well so far, we are left with a four-sided region of pixels that is likely a valid tag.

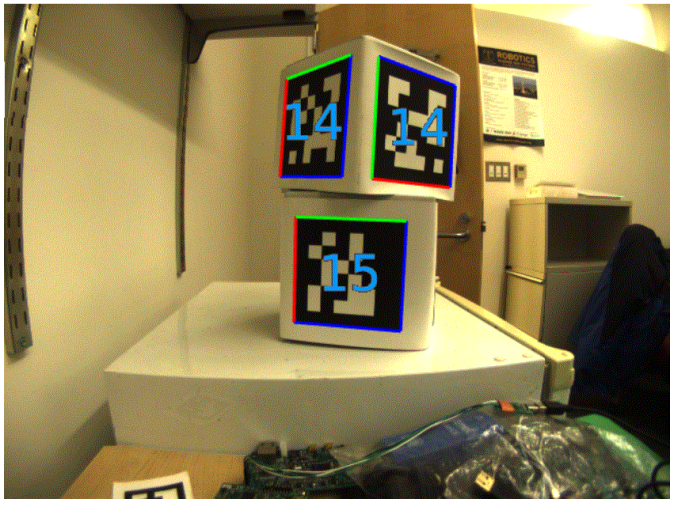
Decode ID

Now that we have one or more regions of pixels which we believe to be a valid AprilTag, we need to identify which tag we are looking at. This is done by «decoding» the pattern of light and dark squares on the inside.

- Calculate the expected interior pixel coordinates where the center of each bit should be
- Mark each location as «1» or «0» by comparing the pixel intensity to a threshold
- Find the tag ID which most closely matches what was seen in the image, allowing for one or two bit errors.

It is possible there is no valid tag ID which matches the suspect tag. In this case, the decoding process stops.

Fit External Quad



Now that we have a tag ID for the region of pixels, we need to do something useful with it.

For most FRC applications, we care about knowing the precise location of the corners of the tag, or its center. In both cases, we expect the resolution-lowering operation we did at the

beginning to have distorted the image, and we want to undo those effects.

The algorithm to do this is:

- Use the detected tag location to define a region of interest in the original-resolution image
- Calculate the *gradient* at pre-defined points in the region of interest to detect where the image most sharply transitions between black to white
- Use these gradient measurements to rapidly re-fit an exterior quadrilateral at full resolution
- Use geometry to calculate the exact center of the re-fit quadrilateral

Note that this step is optional, and can be skipped for faster image processing. However, skipping it can induce significant errors into your robot's behavior, depending on how you are using the tag outputs.

Usage

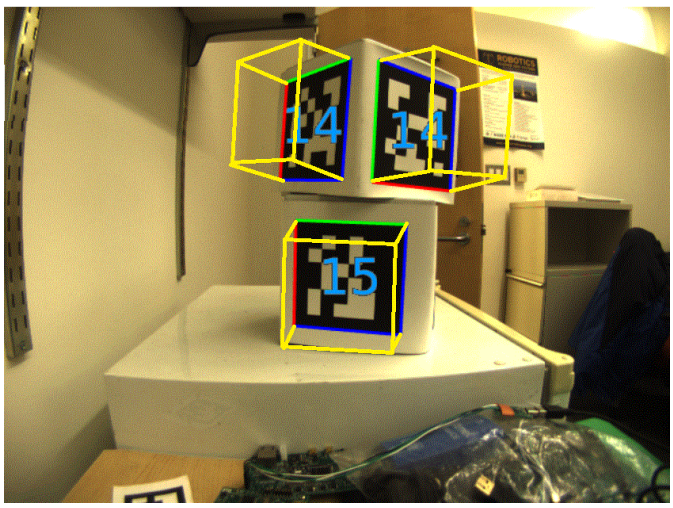
2D Alignment

A simple strategy for using targets is to move the robot until the target is centered in the image. Assuming the field and robot are constructed such that the gamepiece, scoring location, vision target, and camera are all aligned, this method should proved a straightforward method to automatically align the robot to the scoring position.

Using a camera, identify the *centroid* of the AprilTags in view. If the tag's ID is correct, apply drivetrain commands to rotate the robot left or right until the tag is centered in the camera image.

This method does not require calibrating the camera or performing the homography step.

3D Alignment



A more advanced usage of AprilTags is to use their corner locations to help perform *pose estimation*.

Each image is searched for AprilTags using the algorithm described on this page. Using assumptions about how the camera's lense distorts the 3d world onto the 2d array of pixels in the camera, an estimate of the camera's position relative to the tag is calculated. A good camera calibration is required for the assumptions about its lens behavior to be accurate.

The tag's ID is also decoded. from the image. Given each tag's ID, the position of the tag on the field can be looked up.

Knowing the position of the tag on the field, and the position of the camera relative to the tag, the 3D geometry classes can be used to estimate the position of the camera on the field.

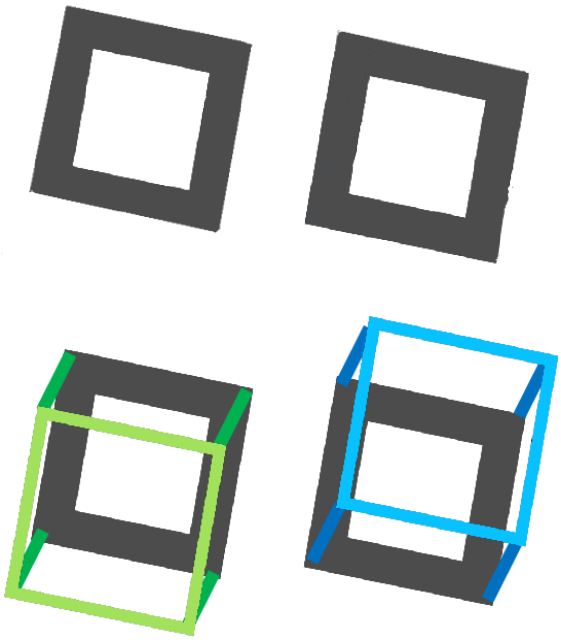
If the camera's position on the robot is known, the robot's position on the field can also be estimated.

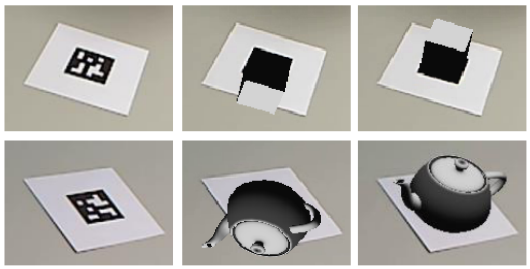
These estimates can be incorporated into the WPILib pose estimation classes.

2D to 3D Ambiguity

The process of translating the four known corners of the target in the image (two-dimensional) into a real-world position relative to the camera (three-dimensional) is inherently ambiguous. That is to say, there are multiple real-world positions that result in the target corners ending up in the same spot in the camera image.

Humans can often use lighting or background clues to understand how objects are oriented in space. However, computers do not have this benefit. They can be tricked by similar-looking targets:





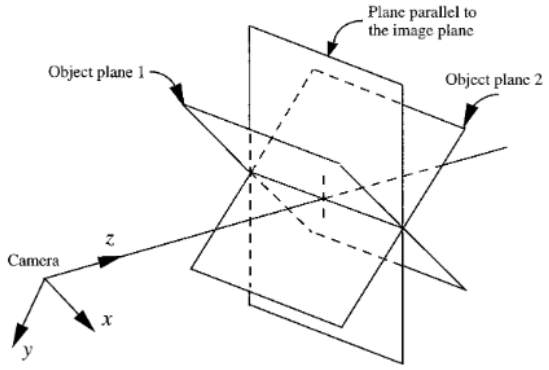


FIG. 4. Two object poses giving the same image under the SOP approximation.

Resolving which position is «correct» can be done in a few different ways:

1. Use your *odometry* history from all sensors to pick the pose closest to where you expect the robot to be.
2. Reject poses which are very unlikely (ex: outside the field perimeter, or up in the air)
3. Ignore pose estimates which are very close together (and hard to differentiate)
4. Use multiple cameras to look at the same target, such that at least one camera can generate a good pose estimate
5. Look at many targets at once, using each to generate multiple pose estimates. Discard the outlying estimates, use the ones which are tightly clustered together.

Adjustable Parameters

`Decimation factor` impacts how much the image is down-sampled before processing. Increasing it will increase detection speed, at the cost of not being able to see tags which are far away.

`Blur` applies smoothing to the input image to decrease noise, which increases speed when fitting quads to pixels, at the cost of precision. For most good cameras, this may be left at zero.

`Threads` changes the number of parallel threads which the algorithm uses to process the image. Certain steps may be sped up by allowing multithreading. In general, you want this to be approximately equal to the number of physical cores in your CPU, minus the number of cores which will be used for other processing tasks.

Detailed information about the tunable parameters can be found here.

Further Learning

The three major versions of AprilTags are described in three academic papers. It's recommended to read them in order, as each builds upon the previous:

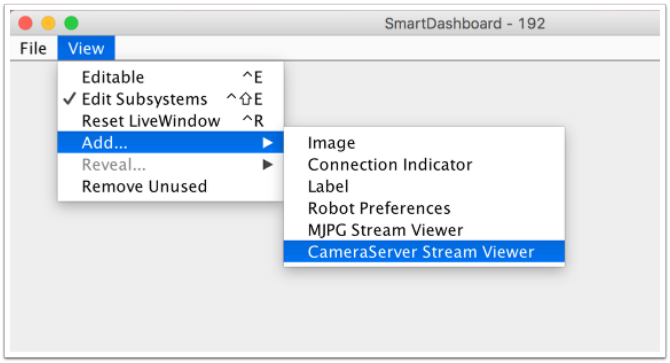
- AprilTags v1
- AprilTags v2
- AprilTags v3
- Pose Ambiguity

25.4 Visión por el RoboRIO

25.4.1 Uso de CameraServer en roboRIO

Programa simple de CameraServer

El siguiente programa inicia la captura automática de una cámara USB como la Microsoft LifeCam que está conectada al roboRIO. En este modo, la cámara capturará fotogramas y los enviará al dashboard. Para ver las imágenes, cree un widget CameraServer Stream Viewer usando el menú «View», luego «Add» en el dashboard. Las imágenes no están procesadas y simplemente se envían desde la cámara al dashboard.



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() {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    CameraServer.startAutomaticCapture();
  }
}
```

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.
    """
```

Programa de servidor de cámara avanzado

In the following example a thread created in robotInit() gets the Camera Server instance. Each frame of the video is individually processed, in this case drawing a rectangle on the image using the OpenCV `rectangle()` method. The resultant images are then passed to the output stream and sent to the dashboard. You can replace the `rectangle` operation with any image processing code that is necessary for your application. You can even annotate the image using OpenCV methods to write targeting information onto the image being sent to the dashboard.

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
                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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      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.

Advertencia: 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. It's easiest to just place it next to your `robot.py`
- 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.
- `robotpy-cscore` is not installed on the roboRIO by default, you need to update your `pyproject.toml` file to install it

Advertencia: `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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
            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)

        # Give the output stream a new image to display
        outputStream.putFrame(mat)
```

You need to update `pyproject.toml` contents to include cscore in the robotpy-extras key (this only shows the portions you need to update):

```
[tool.robotpy]

...

# Add cscore to the robotpy-extras list
robotpy_extras = ["cscore"]
```

Notice that in these examples, the `PutVideo()` method writes the video to a named stream. To view that stream on SmartDashboard or Shuffleboard, select that named stream. In this case that is «Rectangle».

25.4.2 Usando varias cámaras

Cambio de vistas del Driver

Si está interesado en cambiar lo que ve el controlador, y está utilizando SmartDashboard, el visor de flujos del SmartDashboard CameraServer tiene una opción («Ruta de la cámara seleccionada») que lee la clave *NetworkTables* dada y cambia la «Elección de la cámara» a ese valor (mostrando esa cámara). El código del robot sólo tiene que establecer la clave *NetworkTables* con el nombre correcto de la cámara. Asumiendo que la «Ruta de la Cámara Seleccionada» se establece en «Selección de Cámara», el siguiente código utiliza el estado del botón de disparo del joystick 1 para mostrar la cámara1 y la cámara2.

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");
}
```

(continúe en la próxima página)

(proviene de la página anterior)

```
@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

Nota: Python requires you to place your image processing code in a separate file from your robot code. You can create `robot.py` and `vision.py` in the same directory.

`robot.py` contents:

```
import wpilib
from ntcore import NetworkTableInstance

class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        self.joy1 = wpilib.Joystick(0)
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        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()

    camera1 = CameraServer.startAutomaticCapture(0)
    camera2 = CameraServer.startAutomaticCapture(1)

    CameraServer.waitForever()
```

`pyproject.toml` contents (this only shows the portions you need to update):

```
[tool.robotpy]

...

# Add cscore to the robotpy-extras list
robotpy_extras = ["cscore"]
```

Si está utilizando algún otro dashboard, puede cambiar la cámara que utiliza el servidor de la cámara de forma dinámica. Si abre un visor de transmisión nominalmente a camera1, el código del robot cambiará el contenido de la transmisión a camera1 o camera2 según el gatillo del control.

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() {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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;
    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
```

Mantener las transmisiones abiertas

De forma predeterminada, la biblioteca cscore es bastante agresiva al apagar las cámaras que no están en uso. Esto significa que cuando cambie de cámara, es posible que se desconecte de la cámara que no está en uso, por lo que volver a cambiar tendrá cierta demora cuando se reconecte a la cámara. Para mantener abiertas las conexiones de ambas cámaras, use el método `` SetConnectionStrategy() “” para decirle a la biblioteca que mantenga las transmisiones abiertas, incluso si no las está usando.

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

Nota: Python requires you to place your image processing code in a separate file from your robot code. You can create `robot.py` and `vision.py` in the same directory.

`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():
    CameraServer.enableLogging()

    camera1 = CameraServer.startAutomaticCapture(0)
    camera2 = CameraServer.startAutomaticCapture(1)

    camera1.setConnectionStrategy(VideoSource.ConnectionStrategy.kConnectionKeepOpen)
    camera2.setConnectionStrategy(VideoSource.ConnectionStrategy.kConnectionKeepOpen)

    CameraServer.waitForever()
```

`pyproject.toml` contents (this only shows the portions you need to update):

```
[tool.robotpy]

...

# Add cscore to the robotpy-extras list
robotpy_extras = ["cscore"]
```

Nota: Si ambas cámaras son USB, puede encontrarse con limitaciones de ancho de banda USB con resoluciones más altas, ya que en todos estos casos la roboRIO transmitirá datos de ambas cámaras a la roboRIO simultáneamente (por un período corto en las opciones 1 y 2, y continuamente en la opción 3). Teóricamente es posible que la biblioteca evite esta simultaneidad en el caso de la opción 2 (solamente), pero esto no está implementado actualmente.

Las diferentes cámaras informan el uso de ancho de banda de manera diferente. La biblioteca le dirá si está llegando al límite; obtendrá este mensaje de error:

```
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)
```

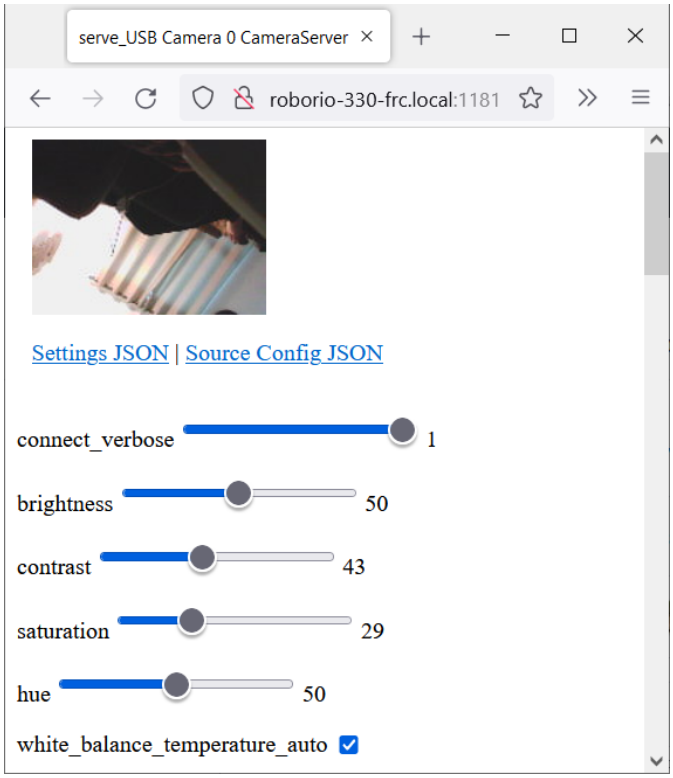
Si está utilizando la Opción 3, le dará este error durante `RobotInit()`. Por lo tanto, debe probar la resolución deseada y ajustar según sea necesario hasta que ambos no obtengan ese error y no excedan las limitaciones de ancho de banda de radio.

25.4.3 CameraServer Web Interface

When CameraServer opens a camera, it creates a webpage that you can use to view the camera stream and view the effects of various camera settings. To connect to the web interface, use a web browser to navigate to `http://roboRIO-TEAM-frc.local:1181`. There is no additional code needed other then *Programa simple de CameraServer*.

Nota: 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.

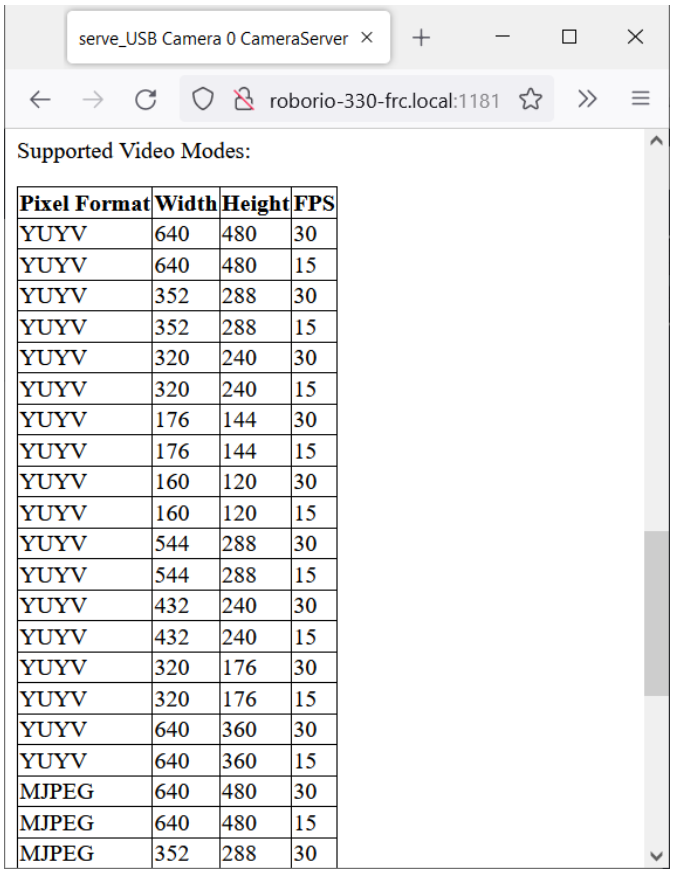
Camera Settings



The web server will show a live camera image and has sliders to adjust various camera settings, such as brightness, contrast, sharpness and many other options. You can adjust the

values and see the results live, and then use the `VideoCamera` class to set those in your robot code.

Camera Video Modes



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

One useful feature is the list of supported video modes at the bottom of the web page. This shows all the supported modes that the camera supports to enable you to choose the one that is the best combination of resolution and frame rate for your requirements.

26

Programación basada en comandos

Esta secuencia de artículos sirve como introducción y referencia para el marco basado en comandos WPILib.

Para obtener una colección de proyectos de ejemplo que utilizan el marco basado en comandos, consulte *Ejemplos basados en comandos*.

26.1 ¿Qué es la programación «basada en comandos»?

WPILib admite una metodología de programación de robots llamada programación «basada en comandos». En general, «basado en comandos» puede referirse tanto al paradigma de programación general como al conjunto de recursos de la biblioteca WPILib incluidos para facilitarlo.

«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.

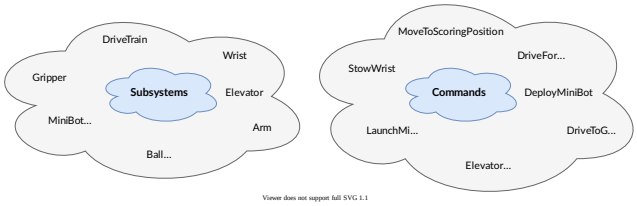
JAVA

```
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 Subsistemas y comandos



El patrón basado en comandos se basa en dos abstracciones centrales: **comandos** y **subsistemas.**

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 *Comandos* 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 Cómo se ejecutan los comandos

Nota: Para obtener una explicación más detallada, consulte: doc: *command-scheduler*.

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 Comandos

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 estructura de un comando

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).

Inicialización

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.

Ejecución

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.

Finalizando

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()`.

Especificar condiciones finales

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.

Importante: 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`.

Nota: 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++ (Encabezado)

```
  /**
   * Grabs the hatch.
   */
  frc2::CommandPtr GrabHatchCommand();

  /**
   * Releases the hatch.
   */
  frc2::CommandPtr ReleaseHatchCommand();
```

C++ (Fuente)

```
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 *PIDCommand*.
- `TrapezoidProfileCommand` tracks a trapezoid motion profile. For more info, see *Comando TrapezoideProfile*.
- `ProfiledPIDCommand` combines PID control with trapezoid motion profiles. For more info, see *Comando ProfiledPID*.
- `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.

Nota: 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:
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  /**
   * Creates a new ExampleCommand.
   *
   * @param subsystem The subsystem used by this command.
   */
  explicit ExampleCommand(ExampleSubsystem* subsystem);

 private:
  ExampleSubsystem* m_subsystem;
};
```

26.2.5 Ejemplo de comando simple

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++ (Encabezado)

```
#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++ (Fuente)

```
#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.

¿Qué tal un caso más complicado? A continuación se muestra un comando de unidad, del mismo proyecto de ejemplo:

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++ (Encabezado)

```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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
 ↪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++ (Fuente)

```
#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);
}

void DefaultDrive::Execute() {
  m_drive->ArcadeDrive(m_forward(), m_rotation());
}
```

And then usage:

JAVA

```
    // Configure default commands
    // Set the default drive command to split-stick arcade drive
    m_robotDrive.setDefaultCommand(
        // A split-stick arcade command, with forward/backward controlled by the left
        // hand, and turning controlled by the right.
        new DefaultDrive(
            m_robotDrive,
            () -> -m_driverController.getLeftY(),
            () -> -m_driverController.getRightX()));
```

C++

```
  // Set up default drive command
  m_drive.SetDefaultCommand(DefaultDrive(
      &m_drive, [this] { return -m_driverController.GetLeftY(); },
      [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

```
    // Configure default commands
    // Set the default drive command to split-stick arcade drive
    m_robotDrive.setDefaultCommand(
        // A split-stick arcade command, with forward/backward controlled by the left
        // hand, and turning controlled by the right.
        Commands.run(
            () ->
                m_robotDrive.arcadeDrive(
                    -m_driverController.getLeftY(), -m_driverController.getRightX()),
            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.

Importante: 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(
```

(continúe en la próxima página)

(proviene de la página anterior)

```
                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)

```
  // 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.

> **Advertencia:** 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.

Nota: 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)

```
#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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   *
   * @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 Subsistemas

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 Creando un subsistema

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 Ejemplo de un subsistema 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) {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  SubsystemBase::InitSendable(builder);

  // Publish the solenoid state to telemetry.
  builder.AddBooleanProperty(
      "extended",
      [this] { return m_hatchSolenoid.Get() == frc::DoubleSolenoid::kForward; },
      nullptr);
}
```

Fíjese en que el subsistema oculta la presencia del DoubleSolenoid al código exterior (está declarado como `privado`), y en su lugar expone públicamente dos acciones de robot de nivel superior y descriptivo: `grabHatch()` y `releaseHatch()`. Es extremadamente importante que los «detalles de implementación» como el solenoide doble se «oculten» de esta manera; esto asegura que el código fuera del subsistema nunca causará que el solenoide esté en un estado inesperado. También permite al usuario cambiar la implementación (por ejemplo, se podría utilizar un motor en lugar de un neumático) sin que el código fuera del subsistema tenga que cambiar con él.

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`
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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

Nota: 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));
```

Nota: A command that is assigned as the default command for a subsystem must require that subsystem.

26.5 Enlazando comandos a Triggers

Además de los comandos autónomos, que se programan al inicio del período autónomo, y los comandos predeterminados, que se programan automáticamente cuando su subsistema no está en uso, la forma más común de ejecutar un comando es vinculándolo a un evento de activación, como un botón que presiona un operador humano. El paradigma basado en comandos hace que esto sea extremadamente fácil de hacer.

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

Nota: 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.

Nota: 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 Componer botones

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 Debouncing Triggers

To avoid rapid repeated activation, triggers (especially those originating from digital inputs) can be debounced with the *WPILib Debouncer class* using the *debounce* method:

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 Estructuración de un proyecto de robot basado en comandos

Si bien los usuarios son libres de usar las bibliotecas basadas en comandos como lo deseen (y se recomienda a los usuarios avanzados que lo hagan), los nuevos usuarios pueden necesitar orientación sobre cómo estructurar un proyecto de robot básico basado en comandos.

En el repositorio de ejemplos de WPILib se incluye una plantilla estándar para un proyecto de robot basado en comandos (Java, C++). Esta sección guiará a los usuarios a través de la estructura de esta plantilla.

El directorio / paquete raíz generalmente contendrá cuatro clases:

`Main`, que es la aplicación principal del robot (solo Java). Los nuevos usuarios * no deben * tocar esta clase. `Robot`, que es responsable del flujo de control principal del código del robot. `RobotContainer`, que contiene los subsistemas y comandos del robot, y es donde se realiza la mayor parte de la configuración declarativa del robot (por ejemplo, enlaces de botones). `Constants`, que contiene constantes accesibles globalmente para su uso en todo el robot.

El directorio raíz también contendrá dos subpaquetes / subdirectorios: `Subsystems` contiene todas las clases de subsistemas definidas por el usuario.``Commands`` contiene todas las clases de comandos definidas por el usuario.

26.6.1 Robot

Como `Robot` (Java, C++ (Header), C++ (Source)) es responsable del flujo de control del programa, y basado en comandos es un paradigma declarativo diseñado para minimizar La cantidad de atención que el usuario debe prestar al flujo de control explícito del programa, la clase `Robot` de un proyecto basado en comandos debe estar prácticamente vacía. Sin embargo, hay algunas cosas importantes que deben incluirse

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, se debe construir una instancia de `RobotContainer` durante el método `robotInit()`; esto es importante, ya que la mayor parte de la configuración declarativa del robot se llamará desde el constructor `RobotContainer`.

En C++, esto no es necesario ya que RobotContainer es un miembro de valor y se construirá durante la construcción 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();
}
```

La inclusión de la llamada `CommandScheduler.getInstance().run()` en el método `robotPeriodic()` es esencial; sin esta llamada, el programador no ejecutará ningún comando programado. Dado que `TimedRobot` se ejecuta con una frecuencia de bucle principal predeterminada de 50 Hz, esta es la frecuencia con la que se llamarán los comandos periódicos y los métodos de subsistema. No se recomienda que los nuevos usuarios llamen a este método desde cualquier otro lugar de su código.

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();
  }
}
```

El método `autonomousInit()` programa un comando autónomo devuelto por la instancia de `RobotContainer`. La lógica para seleccionar qué comando autónomo ejecutar se puede manejar dentro 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 }
```

El método `teleopInit()` cancela cualquier comando autónomo que aún se esté ejecutando. En general, esta es una buena práctica.

Los usuarios avanzados tienen la libertad de agregar código adicional a los diversos métodos de inicio y periódicos como mejor les parezca; sin embargo, debe tenerse en cuenta que incluir grandes cantidades de código de robot imperativo en `Robot.java` es contrario a la filosofía de diseño declarativo del paradigma basado en comandos y puede resultar en un código desorganizado o estructurado de manera confusa.

26.6.2 RobotContainer

Esta clase (*Java <https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi*
__, C ++ (Encabezado) <https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/t
__, C ++ (Fuente) <https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/templ
__) es donde se llevará a cabo la mayor parte de la configuración de su robot basado en comandos. En esta clase, definirá los subsistemas y comandos de su robot, vinculará esos comandos a eventos de activación (como botones) y especificará qué comando ejecutará en su rutina autónoma. Hay algunos aspectos de esta clase para los que los nuevos usuarios pueden querer explicaciones:

Java

```
23   private final ExampleSubsystem m_exampleSubsystem = new ExampleSubsystem();
```

C++ (Header)

```
32   ExampleSubsystem m_subsystem;
```

Tenga en cuenta que los subsistemas se declaran como campos privados en `RobotContainer`. Esto está en marcado contraste con la encarnación anterior del marco basado en comandos, pero está mucho más alineado con las mejores prácticas orientadas a objetos acordadas. Si los subsistemas se declaran como variables globales, permite al usuario acceder a ellos desde cualquier parte del código. Si bien esto puede facilitar ciertas cosas (por ejemplo, no habría necesidad de pasar subsistemas a comandos para que esos comandos accedan a ellos), hace que el flujo de control del programa sea mucho más difícil de rastrear, ya que no es inmediato. obvio qué partes del código pueden cambiar o cambiarse por qué otras partes del código. Esto también evita la capacidad del sistema de administración de recursos para hacer su trabajo, ya que la facilidad de acceso facilita que los usuarios realicen accidentalmente

llamadas conflictivas a métodos del subsistema fuera de los comandos administrados por recursos.

Java

```
61     return Autos.exampleAuto(m_exampleSubsystem);
```

C++ (Source)

```
34   return autos::ExampleAuto(&m_subsystem);
```

Dado que los subsistemas se declaran como miembros privados, deben pasarse explícitamente a los comandos (un patrón llamado «inyección de dependencia») para que esos comandos llamen a sus métodos. Esto se hace aquí con `ExampleCommand`, que se pasa un puntero a 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)

```
void RobotContainer::ConfigureBindings() {
  // Configure your trigger bindings here

  // Schedule `ExampleCommand` when `exampleCondition` changes to `true`
  frc2::Trigger([this] {
    return m_subsystem.ExampleCondition();
  }).OnTrue(ExampleCommand(&m_subsystem).ToPtr());

  // Schedule `ExampleMethodCommand` when the Xbox controller's B button is
  // pressed, cancelling on release.
  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);
}
```

Finalmente, el método `getAutonomousCommand()` proporciona una forma conveniente para que los usuarios envíen su comando autónomo seleccionado a la clase principal de `Robot` (que necesita acceso a él para programarlo cuando se inicie autónomo).

26.6.3 Constants

La clase `Constants` (Java, C++ (Header)) (en C++ no es una clase, sino simplemente un archivo de cabecera en el que se definen varios espacios de nombres) es donde pueden almacenarse las constantes del robot de acceso global (como las velocidades, los factores de conversión de unidades, las ganancias PID y los puertos de sensores/motores). Se recomienda que los usuarios separen estas constantes en clases internas individuales correspondientes a los subsistemas o modos del robot, para mantener los nombres de las variables más cortos.

En Java, todas las constantes deben declararse como `public static final` para que sean accesibles globalmente y no se puedan cambiar. En C++, todas las constantes deben ser `constexpr`.

Para obtener más ejemplos ilustrativos de cómo debería verse una clase de `constants` en la práctica, consulte los de varios proyectos de ejemplo basados en comandos:

- FrisbeeBot (Java, C++)
- Comandos de accionamiento del giroscopio (Java, C++)
- Hatchbot (Java, C++)
- RapidReactCommandBot (Java, C++)

En Java, se recomienda que las constantes se utilicen de otras clases importando estáticamente la clase interna necesaria. Una declaración de `import static` importa el espacio de nombres estático de una clase en la clase en la que está trabajando, de modo que cualquier constante `static` pueda ser referenciada directamente como si se hubiera definido en esa clase. En C++, el mismo efecto se puede lograr con `using namespace`:

JAVA

```
import static edu.wpi.first.wpilibj.templates.commandbased.Constants.OIConstants.*;
```

C++

```
using namespace OIConstants;
```

26.6.4 Subsistemas

Los subsistemas definidos por el usuario deben ir en este paquete / directorio.

26.6.5 Comandos

Los comandos definidos por el usuario deben ir en este paquete / directorio.

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),
  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:

Nota: 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:

Nota: 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:

Nota: 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

```java
public class RunIntakeCommand extends Command {
    private Intake m_intake;

    public RunIntakeCommand(Intake intake) {
        this.m_intake = intake;
        addRequirements(intake);
    }

    @Override
    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++

```cpp
// 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 El planificador de comandos

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.

El `CommandScheduler` también ejecuta el método `periodic()` de cada `` Subsistema`` registrado.

26.8.1 Uso del programador de comandos

El `CommandScheduler` es un *singleton*, lo que significa que es una clase accesible globalmente con una sola instancia. En consecuencia, para acceder al programador, los usuarios deben llamar al comando `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).

Sin embargo, hay una excepción: los usuarios *deben* llamar a `CommandScheduler.getInstance().run()` del método `robotPeriodic()` de su clase `Robot` . Si no se hace esto, el programador nunca se ejecutará y el marco de comandos no funcionará. La plantilla de proyecto a base de comandos proporcionada ya tiene esta llamada incluida.

26.8.2 El método `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     }
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;
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        }
      }
      for (Subsystem requirement : requirements) {
        Command requiring = requiring(requirement);
        if (requiring != null) {
          cancel(requiring);
        }
      }
      initCommand(command, requirements);
    }
  }
```

```
  private void initCommand(Command command, Set<Subsystem> requirements) {
    m_scheduledCommands.add(command);
    for (Subsystem requirement : requirements) {
      m_requirements.put(requirement, command);
    }
    command.initialize();
    for (Consumer<Command> action : m_initActions) {
      action.accept(command);
    }

    m_watchdog.addEpoch(command.getName() + ".initialize()");
```

C++ (Source)

```
void CommandScheduler::Schedule(Command* command) {
  if (m_impl->inRunLoop) {
    m_impl->toSchedule.emplace_back(command);
    return;
  }

  RequireUngrouped(command);

  if (m_impl->disabled || m_impl->scheduledCommands.contains(command) ||
      (frc::RobotState::IsDisabled() && !command->RunsWhenDisabled())) {
    return;
  }

  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);
    }
  }

```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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 secuencia de ejecución del programador

Nota: El método `initialize()` de cada `Comando` es llamado cuando el comando es programado, lo cual no es necesariamente cuando un programador este ejecutándose (a menos que ese comando este atado a un botón).

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++).

Paso 1: ejecutar los métodos periódicos del subsistema

First, the scheduler runs the `periodic()` method of each registered `Subsystem`. In simulation, each subsystem's `simulationPeriodic()` method is called as well.

Java

```
    // Run the periodic method of all registered subsystems.
    for (Subsystem subsystem : m_subsystems.keySet()) {
      subsystem.periodic();
      if (RobotBase.isSimulation()) {
        subsystem.simulationPeriodic();
      }
      m_watchdog.addEpoch(subsystem.getClass().getSimpleName() + ".periodic()");
    }
```

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   }
```

Paso 2: Activadores de programación de comandos

Nota: Para obtener más información sobre cómo funcionan los enlaces de activadores, consulte *Enlazando comandos a Triggers*

En segundo lugar, el programador sondea el estado de todos los desencadenantes registrados para ver si se debe programar algún comando nuevo que se haya vinculado a esos desencadenantes. Si se cumplen las condiciones para programar un comando vinculado, se programa el comando y se ejecuta su método `Initialize()`.

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()");
```

Paso 3: ejecutar/finalizar comandos programados

En tercer lugar, el planificador llama al método `execute()` de cada comando programado actualmente, y luego verifica si el comando ha terminado llamando al método `isFinished()`. Si el comando ha finalizado, también se llama al método `end()`, y el comando se desprograma y se liberan los subsistemas necesarios.

Tenga en cuenta que esta secuencia de llamadas se realiza en orden para cada comando; por lo tanto, un comando puede tener su método `end()` llamado antes de que otro tenga su método `execute()` llamado. Los comandos se manejan en el orden en que fueron programados.

Java

```
    // Run scheduled commands, remove finished commands.
    for (Iterator<Command> iterator = m_scheduledCommands.iterator(); iterator.
↪hasNext(); ) {
      Command command = iterator.next();

      if (!command.runsWhenDisabled() && RobotState.isDisabled()) {
        command.end(true);
        for (Consumer<Command> action : m_interruptActions) {
          action.accept(command);
        }
        m_requirements.keySet().removeAll(command.getRequirements());
        iterator.remove();
        m_watchdog.addEpoch(command.getName() + ".end(true)");
        continue;
      }

      command.execute();
      for (Consumer<Command> action : m_executeActions) {
        action.accept(command);
      }
      m_watchdog.addEpoch(command.getName() + ".execute()");
      if (command.isFinished()) {
        command.end(false);
        for (Consumer<Command> action : m_finishActions) {
          action.accept(command);
        }
        iterator.remove();

        m_requirements.keySet().removeAll(command.getRequirements());
        m_watchdog.addEpoch(command.getName() + ".end(false)");
      }
    }
```

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);
      }

```

(continúe en la próxima página)

(proviene de la página anterior)

```
      for (auto&& requirement : command->GetRequirements()) {
        m_impl->requirements.erase(requirement);
      }

      m_impl->scheduledCommands.erase(command);
      m_watchdog.AddEpoch(command->GetName() + ".End(false)");
    }
  }
```

Paso 4: Programar comandos predeterminados

Finalmente, cualquier `Subsistema` registrado tiene su comando predeterminado programado (si lo tiene). Tenga en cuenta que en este momento se llamará al método `initialize()` del comando predeterminado.

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)

```
  // Add default commands for un-required registered subsystems.
  for (auto&& subsystem : m_impl->subsystems) {
    auto s = m_impl->requirements.find(subsystem.getFirst());
    if (s == m_impl->requirements.end() && subsystem.getSecond()) {
      Schedule({subsystem.getSecond().get()});
    }
  }
```

26.8.4 Deshabilitar el programador

El planificador se puede desactivar llamando a `CommandScheduler.getInstance().disable()`. Cuando está deshabilitado, los comandos `schedule()` y `run()` del programador no harán nada.

El programador se puede volver a habilitar al llamar a `CommandScheduler.getInstance().enable()`.

26.8.5 Métodos de eventos de comando

Occasionally, it is desirable to have the scheduler execute a custom action whenever a certain command event (initialization, execution, or ending) occurs. This can be done with the following methods:

- `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 ->
82                Shuffleboard.addEventMarker(
83                    "Command interrupted", command.getName(), EventImportance.
↪kNormal));
84    CommandScheduler.getInstance()
85        .onCommandFinish(
86            command ->
87                Shuffleboard.addEventMarker(
88                    "Command finished", command.getName(), EventImportance.kNormal));
```

C++ (Source)

```
23  // Log Shuffleboard events for command initialize, execute, finish, interrupt
24  frc2::CommandScheduler::GetInstance().OnCommandInitialize(
25      [](const frc2::Command& command) {
26        frc::Shuffleboard::AddEventMarker(
27            "Command initialized", command.GetName(),
28            frc::ShuffleboardEventImportance::kNormal);
29      });
30  frc2::CommandScheduler::GetInstance().OnCommandExecute(
31      [](const frc2::Command& command) {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      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 Una discusión técnica sobre los comandos de C ++

Nota: 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.

Nota: The model was further changed in 2023, as described *below*.

26.9.1 Modelo de propiedad

El antiguo marco basado en comandos empleaba el uso de punteros sin procesar, lo que significa que los usuarios tenían que usar `nuevo` (lo que resultaba en asignaciones de montón manuales) en su código de robot. Dado que no había una indicación clara sobre quién era el propietario de los comandos (el programador, los grupos de comandos o el usuario mismo), no era evidente quién se suponía que debía encargarse de liberar la memoria.

Varios ejemplos en el antiguo marco basado en comandos involucraban código como este:

```
#include "PlaceSoda.h"
#include "Elevator.h"
#include "Wrist.h"

PlaceSoda::PlaceSoda() {
  AddSequential(new SetElevatorSetpoint(Elevator::TABLE_HEIGHT));
  AddSequential(new SetWristSetpoint(Wrist::PICKUP));
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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).

Este problema evidente fue una de las razones de la reescritura del marco. Se introdujo un modelo de propiedad integral con esta reescritura, junto con el uso de punteros inteligentes que liberarán memoria automáticamente cuando se salgan del alcance.

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`

El uso de `std::unique_ptr` nos permite determinar claramente quién es el propietario del objeto. Debido a que un `std::unique_ptr` no se puede copiar, nunca habrá más de una instancia de un `` td::unique_ptr`` que apunte al mismo bloque de memoria en el montón. Por ejemplo, un constructor para `SequentialCommandGroup` toma un `std::vector<std::unique_ptr<Command>> &&`. Esto significa que requiere una referencia rvalue a un vector de `std::unique_ptr <Command>`. Repasemos un código de ejemplo paso a paso para comprender esto mejor:

```
// 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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
// 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.
```

Con `std::shared_ptr`, no hay un modelo de propiedad claro porque puede haber varias instancias de un `std::shared_ptr` que apuntan al mismo bloque de memoria. Si los comandos estuvieran en instancias `std::shared_ptr`, un grupo de comandos o el programador de comandos no puede tomar posesión y liberar la memoria una vez que el comando ha terminado de ejecutarse porque el usuario, sin saberlo, todavía podría tener un `std::shared_ptr` instancia que apunta a ese bloque de memoria en algún lugar del alcance.

26.9.2 Uso 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:

Decoradores de comando

El nuevo marco basado en comandos incluye una función conocida como «decoradores de comandos», que permite al usuario hacer algo como esto:

```
auto task = MyCommand().AndThen([] { std::cout << "This printed after my command␣
↪ended."; },
  requirements);
```

Cuando se programa la `tarea` , primero ejecutará `MyCommand()` y una vez que ese comando haya terminado de ejecutarse, imprimirá el mensaje en la consola. La forma en que esto se logra internamente es mediante el uso de un grupo de comando secuencial.

Recuerde de la sección anterior que para construir un grupo de comando secuencial, necesitamos un vector de punteros únicos para cada comando. Crear el puntero único para la función de impresión es bastante trivial:

```
temp.emplace_back(
   std::make_unique<InstantCommand>(std::move(toRun), requirements));
```

Aquí, `temp` almacena el vector de comandos que necesitamos pasar al constructor `SequentialCommandGroup` . Pero antes de agregar ese `InstantCommand`,necesitamos agregar `MyCommand()` al``SequentialCommandGroup`` . ¿Como hacemos eso?

```
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.

Soluciones al problema

Nuestra solución inicial a esto fue crear un método virtual en `Command` llamado `TransferOwnership()` que cada subclase de `Command` tenía que anular. Tal anulación se habría visto así:

```
std::unique_ptr<Command> TransferOwnership() && override {
  return std::make_unique<MyCommand>(std::move(*this));
}
```

Debido a que el código estaría en la subclase derivada, `*this` realmente apuntaría a la instancia de subclase deseada y el usuario tiene la información de tipo de la clase derivada para hacer el puntero único.

Después de unos días de deliberación, se propuso un método CRTP. Aquí, una clase derivada intermedia de `Command` llamada `CommandHelper` esxistiría. `CommandHelper` tendría dos argumentos de plantilla, la clase base original y la subclase derivada deseada. Echemos un vistazo a una implementación básica de `CommandHelper` para entender esto:

```
// 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)));
  }
};
```

Así, haciendo que tus comandos personalizados extiendan `CommandHelper` en lugar de `Command` implementará automáticamente esta plantilla para ti y este es el razonamiento detrás de pedir a los equipos que usen lo que puede parecer una forma bastante oscura de hacer las cosas.

Volviendo a nuestro `AndThen()` ejemplo, ahora podemos hacer lo siguiente:

```
// 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 Falta de decoradores avanzados

La mayoría de los decoradores de C++ toman `std::function<void()>` en lugar de comandos reales ellos mismos. La idea de tomar comandos reales en decoradores como `AndThen()`, `BeforeStarting()`, etc. fue considerada pero luego abandonada debido a una variedad de razones.

Decoradores de plantillas

Debido a que necesitamos saber los tipos de los comandos que estamos añadiendo a un grupo de comandos en tiempo de compilación, necesitaremos usar plantillas (variadic para múltiples comandos). Sin embargo, esto podría no parecer un gran problema. Los constructores de grupos de comandos lo hacen de todas formas:

```
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));
}
```

Nota: Este es un constructor secundario para el `SequentialCommandGroup`, además del constructor de vectores que describimos anteriormente.

Sin embargo, cuando realizamos una función de plantilla, su definición debe ser declarada en línea. Esto significa que tendremos que instanciar el `SequentialCommandGroup` en el encabezado «Command.h», lo que plantea un problema. `SequentialCommandGroup` incluye «Command.h». Si incluimos `SequentialCommandGroup` dentro de «Command.h», tenemos una dependencia circular. ¿Cómo lo hacemos ahora entonces?

Usamos una declaración hacia adelante en la parte superior de `Command.h`:

```
class SequentialCommandGroup;

class Command { ... };
```

Y luego incluimos `SequentialCommandGroup.h` en «Command.cpp». Sin embargo, si estas funciones de decorador se templaron, no podemos escribir definiciones en los archivos `.cpp`, resultando en una dependencia circular.

Sintaxis de Java vs Sintaxis de C++

Estos decoradores suelen ahorrar más verbosidad en Java (porque Java requiere llamadas `nuevas` sin procesar) que en C++, así que en general, no hay mucha diferencia sintáctica en C++ si se crea el grupo de comandos manualmente en código de usuario.

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
          {drive})
          .ToPtr());
}
```

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 Control PID a través de subsistemas PID y comandos PID

Nota: Para obtener una descripción de las funciones de control de WPILib PID utilizadas por estos contenedores basados en comandos, consulte *Control PID en 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 Subsistemas 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.

Creación de un subsistema PIDS

Nota: If `periodic` is overridden when inheriting from `PIDSubsystem`, make sure to call `super.periodic()`! Otherwise, PID functionality will not work properly.

Al subclasificar `PIDSubsystem`, los usuarios deben anular dos métodos abstractos para proporcionar la funcionalidad que la clase usará en su operación ordinaria:

getMeasurement()

Java

```
protected abstract double getMeasurement();
```

C++

```
virtual double GetMeasurement() = 0;
```

El método `getMeasurement` devuelve la medición actual de la variable de proceso. El `PIDSubsystem` llamará automáticamente a este método desde su bloque `periodic()` y pasará su valor al bucle de control.

Los usuarios deben anular este método para devolver cualquier lectura del sensor que deseen utilizar como medida de la variable de proceso.

useOutput()

Java

```
protected abstract void useOutput(double output, double setpoint);
```

C++

```
virtual void UseOutput(double output, double setpoint) = 0;
```

El método `useOutput()` consume la salida del controlador PID y el punto de ajuste actual (que a menudo es útil para calcular un feedforward). El `PIDSubsystem` llamará automáticamente a este método desde su bloque `periodic()` y le pasará la salida calculada del bucle de control.

Los usuarios deben anular este método para pasar la salida de control calculada final a los motores de su subsistema.

Pasando el controlador

Los usuarios también deben pasar un `PIDController` a la clase base `PIDSubsystem` a través de la llamada al constructor de superclase de su subclase. Esto sirve para especificar las ganancias de PID, así como el período (si el usuario está utilizando un período de bucle de robot principal no estándar).

Se pueden realizar modificaciones adicionales (por ejemplo, habilitar la entrada continua) al controlador en el cuerpo del constructor llamando a `getController()`.

Usando un subsistema PIDS

Una vez que se ha creado una instancia de una subclase `PIDSubsystem`, los comandos pueden usarla a través de los siguientes métodos:

setSetpoint()

El método `setSetpoint()` se puede utilizar para establecer el punto de ajuste del `PIDSubsystem`. El subsistema rastreará automáticamente el punto de ajuste usando la salida definida:

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() y disable()

Los métodos `enable()` y `disable()` habilitan y deshabilitan el control PID del `PIDSubsystem`.Cuando el subsistema está habilitado, automáticamente ejecutará el circuito de control y rastreará el punto de ajuste. Cuando está deshabilitado, no se realiza ningún control.

Además, el método `enable()` restablece el `PIDController` interno, y el método `disable()` llama al método *useOutput()* definido por el usuario con la salida y el punto de ajuste establecidos en `0`.

Ejemplo completo de subsistema PIDS

¿Cómo se ve un `PIDSubsystem` cuando se usa en la práctica? Los siguientes ejemplos se han extraído del proyecto de ejemplo de FrisbeeBot (Java, C++):

Java

```
package edu.wpi.first.wpilibj.examples.frisbeebot.subsystems;

import edu.wpi.first.math.controller.PIDController;
import edu.wpi.first.math.controller.SimpleMotorFeedforward;
import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.examples.frisbeebot.Constants.ShooterConstants;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
import edu.wpi.first.wpilibj2.command.PIDSubsystem;

public class ShooterSubsystem extends PIDSubsystem {
  private final PWMSparkMax m_shooterMotor = new PWMSparkMax(ShooterConstants.
↪kShooterMotorPort);
  private final PWMSparkMax m_feederMotor = new PWMSparkMax(ShooterConstants.
↪kFeederMotorPort);
  private final Encoder m_shooterEncoder =
      new Encoder(
          ShooterConstants.kEncoderPorts[0],
          ShooterConstants.kEncoderPorts[1],
          ShooterConstants.kEncoderReversed);
  private final SimpleMotorFeedforward m_shooterFeedforward =
      new SimpleMotorFeedforward(
          ShooterConstants.kSVolts, ShooterConstants.kVVoltSecondsPerRotation);

  /** The shooter subsystem for the robot. */
  public ShooterSubsystem() {
    super(new PIDController(ShooterConstants.kP, ShooterConstants.kI,␣
↪ShooterConstants.kD));
    getController().setTolerance(ShooterConstants.kShooterToleranceRPS);
    m_shooterEncoder.setDistancePerPulse(ShooterConstants.kEncoderDistancePerPulse);
    setSetpoint(ShooterConstants.kShooterTargetRPS);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  }

  @Override
  public void useOutput(double output, double setpoint) {
    m_shooterMotor.setVoltage(output + m_shooterFeedforward.calculate(setpoint));
  }

  @Override
  public double getMeasurement() {
    return m_shooterEncoder.getRate();
  }

  public boolean atSetpoint() {
    return m_controller.atSetpoint();
  }

  public void runFeeder() {
    m_feederMotor.set(ShooterConstants.kFeederSpeed);
  }

  public void stopFeeder() {
    m_feederMotor.set(0);
  }
}
```

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)

```cpp
#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));
}

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);
}
```

Usar un `PIDSubsystem` con comandos puede ser muy simple:

Java

```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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  // 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 PIDCommand

The `PIDCommand` class allows users to easily create commands with a built-in PIDController.

Crear un comando 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.

Nota: If subclassing `PIDCommand` and overriding any methods, make sure to call the `super` version of those methods! Otherwise, PID functionality will not work properly.

En cualquier caso, se crea un `PIDCommand` pasando los parámetros necesarios a su constructor (si se define una subclase, esto se puede hacer con una llamada *super()* call):

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++

```
  /**
   * 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 reference (aka setpoint)
   * @param useOutput         the controller's output
   * @param requirements      the subsystems required by this command
   */
  PIDCommand(frc::PIDController controller,
             std::function<double()> measurementSource,
             std::function<double()> setpointSource,
             std::function<void(double)> useOutput,
             Requirements requirements = {});
```

controller

El parámetro `controller` es el objeto `PIDController` que será utilizado por el comando. Al pasar esto, los usuarios pueden especificar las ganancias de PID y el período para el controlador (si el usuario está utilizando un período de bucle de robot principal no estándar).

Al subclasificar `PIDCommand`, se pueden realizar modificaciones adicionales (por ejemplo, habilitar la entrada continua) al controlador en el cuerpo del constructor llamando a `getController()`.

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`.

Al realizar una subclasificación de `PIDCommand`, los usuarios avanzados pueden modificar aún más el proveedor de medición modificando el campo `m_measurement` de la clase.

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.

Al subclasificar `PIDCommand`, los usuarios avanzados pueden modificar aún más el proveedor del punto de ajuste modificando el campo``m_setpoint`` de la clase.

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`.

Al subclasificar `PIDCommand`, los usuarios avanzados pueden modificar aún más el consumidor de salida modificando el campo `m_useOutput` de la clase.

requisitos

Como todos los comandos en línea, `PIDCommand` permite al usuario especificar los requisitos de su subsistema como un parámetro de constructor.

Ejemplo completo de comando PID

¿Qué aspecto tiene un `PIDCommand` cuando se utiliza en la práctica? Los siguientes ejemplos proceden del proyecto de ejemplo 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   /**
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   * Turns to robot to the specified angle.
   *
   * @param targetAngleDegrees The angle to turn to
   * @param drive The drive subsystem to use
   */
  public TurnToAngle(double targetAngleDegrees, DriveSubsystem drive) {
    super(
        new PIDController(DriveConstants.kTurnP, DriveConstants.kTurnI,␣
↪DriveConstants.kTurnD),
        // Close loop on heading
        drive::getHeading,
        // Set reference to target
        targetAngleDegrees,
        // Pipe output to turn robot
        output -> 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().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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   */
  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
                    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,
```

(continúe en la próxima página)

(proviene de la página anterior)

```
                // 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
              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 Creación de perfiles de movimiento a través de los subsistemas TrapezoidProfile y los comandos TrapezoidProfile

Nota: Para obtener una descripción de las funciones de creación de perfiles de movimiento de WPILib utilizadas por estos contenedores basados en comandos, consulte *Perfil de movimiento trapezoidal en WPILib*.

Nota: Los contenedores de comando `TrapezoidProfile` generalmente están destinados a la composición con controladores personalizados o externos. Para combinar la creación de perfiles de movimiento trapezoidal con el `PIDController` de WPILib, consulte *Combinando Motion Profiling y PID Basado en Comandos*.

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++).

Para ayudar aún más a los equipos a integrar la creación de perfiles de movimiento en sus proyectos de robots basados en comandos, WPILib incluye dos envoltorios de conveniencia para

la clase TrapezoidProfile: TrapezoidProfileSubsystem, que genera y ejecuta automáticamente perfiles de movimiento en su método periodic() , y el``TrapezoidProfileCommand``, que ejecuta un único TrapezoidProfile proporcionado por el usuario.

26.11.1 TrapezoideProfileSubsystem

Nota: En C ++, la clase TrapezoidProfileSubsystem se basa en el tipo de unidad utilizada para las mediciones de distancia, que puede ser angular o lineal. Los valores pasados *deben* tener unidades consistentes con las unidades de distancia, o se lanzará un error en tiempo de compilación. Para obtener más información sobre las unidades C ++, consulte *Biblioteca de unidades de 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.

Creación de un subsistema de perfil trapezoidal

Nota: If periodic is overridden when inheriting from TrapezoidProfileSubsystem, make sure to call super.periodic()! Otherwise, motion profiling functionality will not work properly.

Al subclasificar TrapezoidProfileSubsystem, los usuarios deben anular un único método abstracto para proporcionar la funcionalidad que la clase usará en su operación ordinaria:

useState()

Java

```
protected abstract void useState(TrapezoidProfile.State state);
```

C++

```
virtual void UseState(State state) = 0;
```

El método useState() consume el estado actual del perfil de movimiento. El TrapezoidProfileSubsystem llamará automáticamente a este método desde su bloque periodic() y le pasará el estado del perfil de movimiento correspondiente al progreso actual del subsistema a través del perfil de movimiento.

Los usuarios pueden hacer lo que quieran con este estado; un caso de uso típico (como se muestra en el *Full TrapezoidProfileSubsystem Example*) es usar el estado para obtener un punto de ajuste y un avance para un controlador de motor «inteligente» externo.

Parámetros de constructor

Los usuarios deben pasar un conjunto de `TrapezoidProfile.Constraints` a la clase base `TrapezoidProfileSubsystem` a través de la llamada al constructor de superclase de su subclase. Esto sirve para restringir los perfiles generados automáticamente a una velocidad y aceleración máximas determinadas.

Los usuarios también deben pasar en una posición inicial para el mecanismo.

Los usuarios avanzados pueden pasar un valor alternativo para el período de bucle, si se está utilizando un período de bucle principal no estándar.

Uso de un TrapezoidProfileSubsystem

Una vez que se ha creado una instancia de una subclase `TrapezoidProfileSubsystem`, los comandos pueden usarla a través de los siguientes métodos:

setGoal()

Nota: Si desea establecer el objetivo en una distancia simple con una velocidad objetivo implícita de cero, existe una sobrecarga de `setGoal()` que toma un valor de distancia único, en lugar de un estado de perfil de movimiento completo.

El método `setGoal()` se puede utilizar para establecer el estado objetivo del `TrapezoidProfileSubsystem`. El subsistema ejecutará automáticamente un perfil para el objetivo, pasando el estado actual en cada iteración al método *useState()* proporcionado.

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.

Ejemplo completo de TrapezoidProfileSubsystem

¿Qué aspecto tiene un `TrapezoidProfileSubsystem` cuando se utiliza en la práctica? Los siguientes ejemplos están tomados del proyecto de ejemplo ArmbotOffobard (Java, C++):

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) {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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);
}

frc2::CommandPtr ArmSubsystem::SetArmGoalCommand(units::radian_t goal) {
  return frc2::cmd::RunOnce([this, goal] { this->SetGoal(goal); }, {this});
}
```

Usar un `TrapezoidProfileSubsystem` con comandos puede ser bastante 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 Comando TrapezoideProfile

Nota: En C++, la clase `TrapezoidProfileCommand` se basa en el tipo de unidad utilizada para las mediciones de distancia, que puede ser angular o lineal. Los valores pasados *deben* tener unidades consistentes con las unidades de distancia, o se lanzará un error en tiempo de compilación. Para obtener más información sobre las unidades C++, consulte *Biblioteca de unidades de 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.

Creando 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.

Nota: 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.

En cualquier caso, se crea un `TrapezoidProfileCommand` pasando los parámetros necesarios a su constructor (si se define una subclase, esto se puede hacer con una llamada *super ()*):

Java

```
28    * Creates a new TrapezoidProfileCommand that will execute the given {@link␣
↪TrapezoidProfile}.
29    * Output will be piped to the provided consumer function.
30    *
31    * @param profile The motion profile to execute.
32    * @param output The consumer for the profile output.
33    * @param goal The supplier for the desired state
34    * @param currentState The current state
35    * @param requirements The subsystems required by this command.
36    */
37   @SuppressWarnings("this-escape")
38   public TrapezoidProfileCommand(
39       TrapezoidProfile profile,
40       Consumer<State> output,
41       Supplier<State> goal,
42       Supplier<State> currentState,
43       Subsystem... requirements) {
```

C++

```
35   /**
36    * Creates a new TrapezoidProfileCommand that will execute the given
37    * TrapezoidalProfile. Output will be piped to the provided consumer function.
38    *
39    * @param profile      The motion profile to execute.
40    * @param output       The consumer for the profile output.
41    * @param goal The supplier for the desired state
42    * @param currentState The current state
43    * @param requirements The list of requirements.
44    */
45   TrapezoidProfileCommand(frc::TrapezoidProfile<Distance> profile,
46                           std::function<void(State)> output,
47                           std::function<State()> goal,
48                           std::function<State()> currentState,
49                           Requirements requirements = {})
```

profile

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.

output

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.

requisitos

Como todos los comandos en línea, `TrapezoidProfileCommand` permite al usuario especificar los requisitos de su subsistema como un parámetro de constructor.

Ejemplo completo de TrapezoidProfileCommand

¿Qué aspecto tiene un `TrapezoidProfileSubsystem` cuando se utiliza en la práctica? Los siguientes ejemplos están tomados del proyecto de ejemplo DriveDistanceOffboard (Java, C++):

Java

```
package edu.wpi.first.wpilibj.examples.drivedistanceoffboard.commands;

import edu.wpi.first.math.trajectory.TrapezoidProfile;
import edu.wpi.first.wpilibj.examples.drivedistanceoffboard.Constants.DriveConstants;
import edu.wpi.first.wpilibj.examples.drivedistanceoffboard.subsystems.DriveSubsystem;
import edu.wpi.first.wpilibj2.command.TrapezoidProfileCommand;

/** Drives a set distance using a motion profile. */
public class DriveDistanceProfiled extends TrapezoidProfileCommand {
  /**
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   * Creates a new DriveDistanceProfiled command.
   *
   * @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{
```

(continúe en la próxima página)

(proviene de la página anterior)

```
          frc::TrapezoidProfile<units::meters>{
              // 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>(
            // 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 Combinando Motion Profiling y PID Basado en Comandos

Nota: Para obtener una descripción de las funciones de control de WPILib PID utilizadas por estos contenedores basados en comandos, consulte *Control PID en WPILib*.

Una solución común de controles FRC® es emparejar un perfil de movimiento trapezoidal para la generación de puntos de ajuste con un controlador PID para el seguimiento de puntos de ajuste. Para facilitar esto, WPILib incluye su propia clase *ProfiledPIDController*. Para ayudar aún más a los equipos a integrar esta funcionalidad en sus robots, el marco basado en comandos contiene dos envoltorios de conveniencia para la clase `ProfiledPIDController`: `ProfiledPIDSubsystem`, que integra el controlador en un subsistema, y `ProfiledPIDCommand`, que integra el controlador en un comando.

26.12.1 Subsistema ProfiledPIDS

Nota: En C++, la clase `ProfiledPIDSubsystem` se basa en el tipo de unidad utilizada para las mediciones de distancia, que puede ser angular o lineal. Los valores pasados *deben* tener unidades consistentes con las unidades de distancia, o se lanzará un error en tiempo de compilación. Para obtener más información sobre las unidades C++, consulte *Biblioteca de unidades de 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.

Creación de un subsistema ProfiledPIDS

Nota: If `periodic` is overridden when inheriting from `ProfiledPIDSubsystem`, make sure to call `super.periodic()`! Otherwise, control functionality will not work properly.

Al subclasificar `ProfiledPIDSubsystem`, los usuarios deben anular dos métodos abstractos para proporcionar la funcionalidad que la clase utilizará en su operación ordinaria:

getMeasurement()

Java

```
protected abstract double getMeasurement();
```

C++

```
virtual Distance_t GetMeasurement() = 0;
```

El método `getMeasurement` devuelve la medición actual de la variable de proceso. El `PIDSubsystem` llamará automáticamente a este método desde su bloque `periodic()` y pasará su valor al bucle de control.

Los usuarios deben anular este método para devolver cualquier lectura del sensor que deseen utilizar como medida de la variable de proceso.

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.

Los usuarios deben anular este método para pasar la salida de control calculada final a los motores de su subsistema.

Pasando el controlador

Los usuarios también deben pasar un `ProfiledPIDController` a la clase base `ProfiledPIDSubsystem` a través de la llamada al constructor de superclase de su subclase. Esto sirve para especificar las ganancias de PID, las restricciones del perfil de movimiento y el período (si el usuario está utilizando un período de bucle de robot principal no estándar).

Se pueden realizar modificaciones adicionales (por ejemplo, habilitar la entrada continua) al controlador en el cuerpo del constructor llamando a `getController()`.

Uso de un subsistema ProfiledPIDS

Una vez que se ha creado una instancia de una subclase `PIDSubsystem`, los comandos pueden usarla a través de los siguientes métodos:

setGoal()

Nota: Si desea establecer el objetivo en una distancia simple con una velocidad objetivo implícita de cero, existe una sobrecarga de `setGoal()` que toma un valor de distancia único, en lugar de un estado de perfil de movimiento completo.

El método `setGoal()` se puede utilizar para establecer el punto de ajuste del `PIDSubsystem`. El subsistema rastreará automáticamente el punto de ajuste usando la salida definida:

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() y disable()

Los métodos `enable()` y `disable()` habilitan y deshabilitan el control automático del `ProfiledPIDSubsystem`. Cuando el subsistema está habilitado, ejecutará automáticamente el perfil de movimiento y el bucle de control y seguirá hasta la meta. Cuando está deshabilitado, no se realiza ningún control.

Además, el método `enable()` restablece el `ProfiledPIDController` interno, y el método``disable()`` llama al método *useOutput()* definido por el usuario con la salida y el punto de ajuste establecidos en `0`.

Ejemplo completo de subsistema ProfiledPIDS

¿Qué aspecto tiene un subsistema PIDS cuando se utiliza en la práctica? Los siguientes ejemplos están tomados del proyecto de ejemplo ArmBot (Java, C++):

Java

```
5  package edu.wpi.first.wpilibj.examples.armbot.subsystems;
6
7  import edu.wpi.first.math.controller.ArmFeedforward;
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.examples.armbot.Constants.ArmConstants;
12 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
13 import edu.wpi.first.wpilibj2.command.ProfiledPIDSubsystem;
14
15 /** A robot arm subsystem that moves with a motion profile. */
16 public class ArmSubsystem extends ProfiledPIDSubsystem {
17   private final PWMSparkMax m_motor = new PWMSparkMax(ArmConstants.kMotorPort);
18   private final Encoder m_encoder =
19       new Encoder(ArmConstants.kEncoderPorts[0], ArmConstants.kEncoderPorts[1]);
20   private final ArmFeedforward m_feedforward =
21       new ArmFeedforward(
22           ArmConstants.kSVolts, ArmConstants.kGVolts,
23           ArmConstants.kVVoltSecondPerRad, ArmConstants.kAVoltSecondSquaredPerRad);
24
25   /** Create a new ArmSubsystem. */
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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;
  }
}
```

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:
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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});
}

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;
}
```

Usar un `ProfiledPIDSubsystem` con comandos puede ser muy 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 Comando ProfiledPID

Nota: En C++, la clase `ProfiledPIDCommand` se basa en el tipo de unidad utilizado para las mediciones de distancia, que puede ser angular o lineal. Los valores pasados *deben* tener unidades consistentes con las unidades de distancia, o se lanzará un error en tiempo de compilación. Para obtener más información sobre las unidades C++, consulte *Biblioteca de unidades de C++*.

The `ProfiledPIDCommand` class (Java, C++) allows users to easily create commands with a built-in ProfiledPIDController.

Creación de un comando 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.

Nota: If subclassing `ProfiledPIDCommand` and overriding any methods, make sure to call the `super` version of those methods! Otherwise, control functionality will not work properly.

En cualquier caso, se crea un `ProfiledPIDCommand` pasando los parámetros necesarios a su constructor (si se define una subclase, esto se puede hacer con una llamada *super()*):

Java

```
/**
 * Creates a new PIDCommand, which controls the given output with a ProfiledPIDController. Goal
 * velocity is specified.
 *
 * @param controller the controller that controls the output.
 * @param measurementSource the measurement of the process variable
 * @param goalSource the controller's goal
 * @param useOutput the controller's output
 * @param requirements the subsystems required by this command
 */
public ProfiledPIDCommand(
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    ProfiledPIDController controller,
    DoubleSupplier measurementSource,
    Supplier<State> goalSource,
    BiConsumer<Double, State> useOutput,
    Subsystem... requirements) {
```

C++

```
/**
 * Creates a new PIDCommand, which controls the given output with a
 * ProfiledPIDController.
 *
 * @param controller        the controller that controls the output.
 * @param measurementSource the measurement of the process variable
 * @param goalSource   the controller's goal
 * @param useOutput         the controller's output
 * @param requirements      the subsystems required by this command
 */
ProfiledPIDCommand(frc::ProfiledPIDController<Distance> controller,
                   std::function<Distance_t()> measurementSource,
                   std::function<State()> goalSource,
                   std::function<void(double, State)> useOutput,
                   Requirements requirements = {})
```

controlador

El parámetro `controller` es el objeto `ProfiledPIDController` que será utilizado por el comando. Al pasar esto, los usuarios pueden especificar las ganancias de PID, las restricciones del perfil de movimiento y el período para el controlador (si el usuario está utilizando un período de bucle de robot principal no estándar).

Al subclasificar `ProfiledPIDCommand`, se pueden realizar modificaciones adicionales (por ejemplo, habilitar la entrada continua) al controlador en el cuerpo del constructor llamando a `getController()`.

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`.

Al subclasificar `ProfiledPIDCommand`, los usuarios avanzados pueden modificar aún más el proveedor de medidas modificando el campo `m_measurement` de la clase.

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.

Al subclasificar `ProfiledPIDCommand`, los usuarios avanzados pueden modificar aún más el proveedor del punto de ajuste modificando el campo `m_goal` de la clase.

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`.

Al subclasificar `ProfiledPIDCommand`, los usuarios avanzados pueden modificar aún más el consumidor de salida modificando el campo `m_useOutput` de la clase.

requisitos

Como todos los comandos en línea, `ProfiledPIDCommand` permite al usuario especificar los requisitos de su subsistema como un parámetro de constructor.

Ejemplo completo de ProfiledPIDCommand

¿Qué aspecto tiene un `ProfiledPIDCommand` cuando se utiliza en la práctica? Los siguientes ejemplos proceden del proyecto de ejemplo GyroDriveCommands (Java, C++):

Java

```
5  package edu.wpi.first.wpilibj.examples.gyrodrivecommands.commands;
6
7  import edu.wpi.first.math.controller.ProfiledPIDController;
8  import edu.wpi.first.math.trajectory.TrapezoidProfile;
9  import edu.wpi.first.wpilibj.examples.gyrodrivecommands.Constants.DriveConstants;
10 import edu.wpi.first.wpilibj.examples.gyrodrivecommands.subsystems.DriveSubsystem;
11 import edu.wpi.first.wpilibj2.command.ProfiledPIDCommand;
12
13 /** A command that will turn the robot to the specified angle using a motion profile.␣
   ↪*/
14 public class TurnToAngleProfiled extends ProfiledPIDCommand {
15   /**
16    * Turns to robot to the specified angle using a motion profile.
17    *
18    * @param targetAngleDegrees The angle to turn to
19    * @param drive The drive subsystem to use
20    */
21   public TurnToAngleProfiled(double targetAngleDegrees, DriveSubsystem drive) {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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();
  }
}
```

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.
   *
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   * @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);
          },
          // 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 *Method References*.

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 *Lambda Expressions in Java*.

26.13.3 Lambda Expressions (C++)

> **Advertencia:** 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 *Lambda Expressions in C++*.

27

Cinemática y Odometría

27.1 Introduction to Kinematics and The ChassisSpeeds Class

Nota: Kinematics and odometry uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.1.1 ¿Qué es la cinemática?

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 ¿Qué es la odometría?

La odometría implica el uso de sensores en el robot para crear una estimación de la posición del robot en el campo. En FRC, estos sensores suelen ser varios codificadores (el número exacto depende del tipo de unidad) y un giroscopio para medir el ángulo del robot. Las clases de odometría utilizan las clases de cinemática junto con las entradas periódicas del usuario sobre velocidades (y ángulos en el caso de viraje) para crear una estimación de la ubicación del robot en el campo.

27.1.3 The ChassisSpeeds Class

El objeto `ChassisSpeeds` es esencial para la nueva suite de cinemática y odometría de WPILib. El objeto `ChassisSpeeds` representa las velocidades de un chasis de robot. Esta estructura tiene tres componentes:

- `vx`: la velocidad del robot en la dirección x (hacia adelante).
- `vy`: la velocidad del robot en la dirección y (lateral). (Los valores positivos significan que el robot se mueve hacia la izquierda).
- `omega`: la velocidad angular del robot en radianes por segundo.

Nota: Un tren motriz no holonómico (es decir, un tren motriz que no se puede mover hacia los lados, por ejemplo, un accionamiento diferencial) tendrá un componente `vy` de cero debido a su incapacidad para moverse hacia los lados.

27.1.4 Construcción de un objeto 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

```
// 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++

```
// 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

```
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 Creación de un objeto ChassisSpeeds a partir de velocidades relativas al campo

También se puede crear un objeto `ChassisSpeeds` a partir de un conjunto de velocidades relativas al campo cuando se proporciona el ángulo del robot. Esto convierte un conjunto de velocidades deseadas relativas al campo (por ejemplo, hacia la estación de alianza opuesta y hacia el límite del campo derecho) en un objeto `ChassisSpeeds` que representa velocidades relativas al marco del robot. Esto es útil para implementar controles orientados al campo para un robot de accionamiento giratorio o mecanum.

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))
```

Nota: No se establece explícitamente que la velocidad angular sea «relativa al campo» porque la velocidad angular es la misma que se mide desde una perspectiva de campo o una perspectiva de robot.

27.2 Cinemática de accionamiento diferencial

La clase `DifferentialDriveKinematics` es una herramienta útil que convierte entre un objeto `ChassisSpeeds` y un objeto `DifferentialDriveWheelSpeeds`, que contiene velocidades para los lados izquierdo y derecho de un robot de accionamiento diferencial.

27.2.1 Construyendo el objeto de cinemática

El objeto `DifferentialDriveKinematics` acepta un argumento constructor, que es el ancho de pista del robot. Esto representa la distancia entre los dos juegos de ruedas en una transmisión diferencial.

Nota: 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 Conversión de velocidades del chasis a velocidades de las ruedas

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;
```

(continúe en la próxima página)

(proviene de la página anterior)

```
// Right velocity
double rightVelocity = wheelSpeeds.rightMetersPerSecond;
```

C++

```
// 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

```
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 Conversión de velocidades de las ruedas a velocidades del chasis

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
# Angular velocity
angularVelocity = chassisSpeeds.omega
```

27.3 Odometría de mando diferencial

Un usuario puede utilizar las clases de cinemática de accionamiento diferencial para realizar: ref:*odometría <docs/software/kinematics-and-odometry/intro-and-chassis-speeds:What is odometry?>*. WPILib contiene una clase `DifferentialDriveOdometry` que se puede utilizar para rastrear la posición de un robot de accionamiento diferencial en el campo.

Nota: Debido a que este método solo usa codificadores y un giróscopo, la estimación de la posición del robot en el campo variará con el tiempo, especialmente cuando tu robot entre en contacto con otros robots durante el juego. Sin embargo, la odometría suele ser muy precisa durante el período autónomo.

27.3.1 Creación del objeto de odometría

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`.

Nota: 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 Actualización de la postura del robot

El método de `actualización` se puede utilizar para actualizar la posición del robot en el campo. Este método debe ser llamado periódicamente, preferiblemente en el método `periodic()` de un *Subsistema*. El método de `actualización` devuelve la nueva pose actualizada del robot. Este método tiene en cuenta el ángulo de giro del robot, junto con la distancia del codificador izquierdo y la distancia del codificador derecho.

Nota: 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 Restablecer la postura del 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.

Importante: 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.

Nota: 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 Swerve Drive Kinematics

La clase `SwerveDriveKinematics` es una herramienta útil que convierte entre un objeto `ChassisSpeeds` a varios objetos `SwerveModuleState` , los cuales contienen velocidades y ángulos para cada módulo swerve de un robot.

Nota: Swerve drive kinematics uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.4.1 Clase swerve module state

La clase `SwerveModuleState` contiene información de la velocidad y ángulo de un modo singular de swerve drive. El constructor para una `SwerveModuleState` toma dos argumentos, la velocidad de la llanta en el módulo, y el ángulo del módulo.

Nota: 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.

Nota: Un ángulo de 0 corresponde a los módulos orientados hacía adelante.

27.4.2 Construyendo los objetos cinemáticos.

La clase `SwerveDriveKinematics` acepta un número variable de argumentos del constructor, con cada argumento siendo la ubicación de un módulo swerve relativo al centro del robot (así como `Translation2d`. El número de argumentos del constructor corresponde al número de módulos swerve.

Nota: Un swerve drive tiene que tener 2 módulos o más.

Nota: En C++, la clase se planifica en función del número de módulos. Por lo tanto, al construir un objeto `SwerveDriveKinematics` como un miembro variable de una clase, el número de módulos debe ser pasado como un argumento de plantilla. Por ejemplo, para un swerve drive común con cuatro módulos, el objeto cinemático debe ser construido como: `frc::SwerveDriveKinematics<4>m_kinematics{...}`.

Las ubicaciones de los módulos deben ser relativas con el centro del robot. Los valores positivos x representan el movimiento hacía la parte delantera del robot, mientras que los calores positivos representan el movimiento del robot hacía la izquierda.

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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 Convertir las velocidades del chasis a estados de los módulos

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.

Los elementos en la matriz que se devuelven por este método son el mismo orden en el cual se construyó el objeto cinemático. Por ejemplo, si el objeto cinemático se construyó con la ubicación del módulo frontal izquierdo, la ubicación del módulo frontal derecho, la ubicación del módulo posterior izquierdo y la ubicación del módulo posterior derecho en ese orden, los elementos de la matriz serían el estado del módulo frontal izquierdo, el estado del módulo frontal derecho, el estado del módulo posterior izquierdo y el estado del módulo posterior derecho en ese orden.

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)
```

Optimización del ángulo del módulo

La clase `SwerveModuleState` contiene un método estático `optimize()` (Java) / `Optimize()` (C++) que se utiliza para «optimizar» la velocidad y el punto de ajuste del ángulo de un determinado `SwerveModuleState` para minimizar el cambio de rumbo. Por ejemplo, si el punto de ajuste angular de un cierto módulo de cinemática inversa es de 90 grados, pero su ángulo actual es de -89 grados, este método negará automáticamente la velocidad del punto de ajuste del módulo y hará que el punto de ajuste angular sea de -90 grados para reducir la distancia. el módulo tiene que viajar.

Este método toma dos parámetros: el estado deseado (generalmente del método `toSwerveModuleStates`) y el ángulo actual. Devolverá el nuevo estado optimizado que puede usar como punto de ajuste en su circuito de control de retroalimentación.

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()))
```

Field-oriented drive

Recuerde que un objeto `ChassisSpeeds` puede crearse a partir de un conjunto de velocidades deseadas orientadas al campo. Esta característica puede ser utilizada para obtener estados de módulo a partir de un conjunto de velocidades deseadas orientadas al campo.

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));
```

(continúe en la próxima página)

(proviene de la página anterior)

```
// 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)
```

Usando centros de rotación personalizados

En ocasiones, girar alrededor de una esquina específica puede ser deseable para ciertas maniobras evasivas. Este tipo de comportamiento también es apoyado por las clases de WPILib. El mismo método `ToSwerveModuleStates()` acepta un segundo parámetro para el centro de rotación (así como``Translation2d``). Justo como la ubicación de las llantas, la `Translation2d` representa que el centro de rotación debe ser relativo al centro del robot.

Nota: Debido a que todos los robots son un marco rígido, las velocidades `vx` y `vy` del objeto `ChassisSpeeds` aplican a la totalidad del robot. Sin embargo, el `omega` del objeto `ChassisSpeeds` se medirá desde el centro de rotación.

Por ejemplo, uno puede establecer el centro de rotación en un cierto módulo y si el objeto `ChassisSpeeds` tiene `vx` y `vy` de cero y un `omega` que no sea cero, el robot parecerá rotar alrededor de ese particular módulo swerve.

27.4.4 Convirtiendo estados de módulo a velocidades del chasis

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
    });
  }
}
```

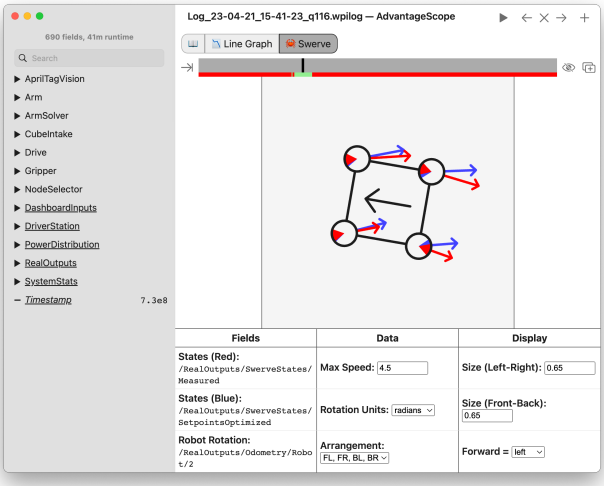
C++

```
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 Odometría para un Swerve Drive

Un usuario puede usar las clases cinemáticas del swerve drive para ejecutar *odometría*. WPILib contiene una clase `SwerveDriveOdometry` que puede ser utilizada para rastrear la posicion de un robot con swerve drive en el campo.

Nota: Dado a que este método utiliza solamente encoders y un giroscopio, el estimado de la posición del robot en el campo variará con el tiempo, especialmente cuando el robot tenga contacto con otros robots durante el juego. Sin embargo, la odometría es usualmente muy acertada para el periodo autónomo.

27.5.1 Creando el objeto odométrico

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`.

Nota: 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 Actualizando la posición del 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.

Este método `update` debe ser llamado periódicamente, de preferencia en el método `periodic()` de un *Subsistema*. El método `update` regresa la nueva posición actualizada del robot.

JAVA

```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++

```cpp
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

```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 Restableciendo la posición del 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.

Importante: 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.

Nota: 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 Cinemática de accionamiento Mecanum

La clase `MecanumDriveKinematics` es una herramienta útil que convierte entre un objeto `ChassisSpeeds` y un objeto `MecanumDriveWheelSpeeds`, que contiene las velocidades de cada una de las cuatro ruedas de un mecanismo mecanum.

Nota: Mecanum kinematics uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.6.1 Construyendo el objeto cinemático

La clase `MecanumDriveKinematics` acepta cuatro argumentos de constructor, y cada argumento es la ubicación de una rueda en relación con el centro del robot (como un `Translation2d`). El orden de los argumentos es frontal izquierdo, frontal derecho, posterior izquierdo y posterior derecho. Las ubicaciones de las ruedas deben ser relativas al centro del robot. Los valores x positivos representan un movimiento hacia la parte delantera del robot, mientras que los valores y positivos representan un movimiento hacia la izquierda del 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 Conversión de velocidades del chasis a velocidades de las ruedas

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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
double frontRight = wheelSpeeds.frontRightMetersPerSecond
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)
```

Accionamiento orientado al campo

Recall que un objeto `ChassisSpeeds` se puede crear a partir de un conjunto de velocidades deseadas orientadas al campo. Esta función se puede utilizar para obtener velocidades de rueda a partir de un conjunto de velocidades deseadas orientadas al campo.

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)
```

Usar centros de rotación personalizados

A veces, girar alrededor de una esquina específica puede ser deseable para ciertas maniobras evasivas. Este tipo de comportamiento también es compatible con las clases WPILib. El mismo método `ToWheelSpeeds()` acepta un segundo parámetro para el centro de rotación (como un `Translation2d`). Al igual que las ubicaciones de las ruedas, la `Translation2d` que representa el centro de rotación debe ser relativa al centro del robot.

Nota: Debido a que todos los robots son un marco rígido, las velocidades `vx` y `vy` proporcionadas del objeto `ChassisSpeeds` aún se aplicarán a la totalidad del robot. Sin embargo, el `omega` del objeto `ChassisSpeeds` se medirá desde el centro de rotación.

Por ejemplo, se puede establecer el centro de rotación en una determinada rueda y si el objeto `ChassisSpeeds` proporcionado tiene un `vx` y un `vy` de cero y un `omega` distinto de cero, el robot parecerá girar alrededor de esa rueda en particular.

27.6.3 Conversión de velocidades de rueda a velocidades de chasis

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 Odometría Mecanum Drive

Un usuario puede utilizar las clases de cinemática de la unidad mecanum para realizar *odometría*. WPILib contiene una clase `MecanumDriveOdometry` que se puede utilizar para rastrear la posición de un robot de accionamiento mecanum en el campo.

Nota: Debido a que este método solo usa codificadores y un giróscopo, la estimación de la posición del robot en el campo variará con el tiempo, especialmente cuando tu robot entre en contacto con otros robots durante el juego. Sin embargo, la odometría suele ser muy precisa durante el período autónomo.

27.7.1 Creando el objeto de odometría

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`.

Nota: 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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
// 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.
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)
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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
)

# 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 Actualizar la pose del 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 Restablecer la postura del 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.

Importante: 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.

Nota: 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.

Importante: 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 ¿Qué es una Tabla de Enrutamiento?

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.

Nota: 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.

The Network Tables Protocol Specification contains detailed documentation on the current wire protocol.

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 automa-

tically 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:

Nombre de la Tabla	Uso
/SmartDashboard	Usada para guardas valores escritos en la SmartDashboard o Shuffleboard usando el conjunto de métodos `SmartDashboard.put()`.
/LiveWindow	Usada para guardar valores del Modo de prueba (Modo de la driver station). Normalmente estos son subsistemas y los sensores y actuadores asociados.
/FMSInfo	Información acerca del partido en ejecución que viene de la driver station y el 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
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}");
  }

  public void periodic() {
    // publish a default value
    dblPub.setDefault(0.0);

    // publish a value with current timestamp
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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␣
↪must
    // be a JSON map.
    dblPub = dblTopic.PublishEx("double", {{"myprop", 5}});
  }

  void Periodic() {
    // publish a default value
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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);
  }

  ~Example() {
    // stop publishing
    nt::Unpublish(dblPub);
  }
};
```

C

```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

```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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        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));
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    // 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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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() {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    // 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);
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;
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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
        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:
    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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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});

    // 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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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);
    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);
  }
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  ~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);

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

```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
        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));
    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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    // 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");
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    // 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();

    // 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

```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}',
            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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        # 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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
↪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();
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    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});

    // 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;
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(
```

(continúe en la próxima página)

(proviene de la página anterior)

```
            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 NetworkTables Networking

The advantage of the robot program being the server is that it's at a known network name (and typically at a known address) that is based on the team number. This is why it's possible in both the NetworkTables client API and in most dashboards to simply provide the team number, rather than a server address. As the robot program is the server, note this means the NetworkTables server is running on the local computer when running in simulation.

28.5.1 Starting a NetworkTables Server

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 Starting a NetworkTables Client

Java

```
NetworkTableInstance inst = 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", 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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    }

    // 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,
↪"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):
        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.

Advertencia: 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");
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    // 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
    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);

    // 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"
```

(continúe en la próxima página)

(proviene de la página anterior)

```
            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()

        # 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)
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        # 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):
        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

```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);
    yPub.set(y);
    x += 0.05;
    y += 1.0;
  }
}
```

C++

```cpp
#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() {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    // 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()
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        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 programa personalizado básico es como el siguiente ejemplo.

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");
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        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)

```cpp
#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

```
#!/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».

A continuación, esta instancia se inicia como cliente de NetworkTables con el número de equipo (el roboRIO es siempre el servidor). Además, si el programa se ejecuta en la Driver Station, mediante el método startDSClient(), NetworkTables obtendrá la dirección IP del robot de la Driver Station.

Luego este programa de muestra simplemente se repite una vez por segundo y obtiene los valores de “x” y “y” y los captura en la consola. En un programa más realista, el cliente podría estar procesando o generando valores para que el robot los consuma.

28.8.1 Compilando usando using Gradle

Example build.gradle files are provided in the StandaloneAppSamples Repository Update the GradleRIO version to correspond to the desired WPILib version.

Java

```
1  plugins {
2      id "java"
3      id 'application'
4      id 'com.github.johnrengelman.shadow' version '8.1.1'
5      id "edu.wpi.first.GradleRIO" version "2024.2.1"
6      id 'edu.wpi.first.WpilibTools' version '1.3.0'
7  }
8
9  application {
10     mainClass = 'Program'
```

(continúe en la próxima página)

(proviene de la página anterior)

```
}

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.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

            sources.cpp {
                source {
                    srcDir 'src/main/cpp'
                    include '**/*.cpp', '**/*.cc'
                }
                exportedHeaders {
                    srcDir 'src/main/include'
                }
            }

            // Enable run tasks for this component
            wpi.cpp.enableExternalTasks(it)

            wpi.cpp.deps.wpilibStatic(it)
        }
    }
}

wrapper {
    gradleVersion = '8.5'
}
```

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();
  }
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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

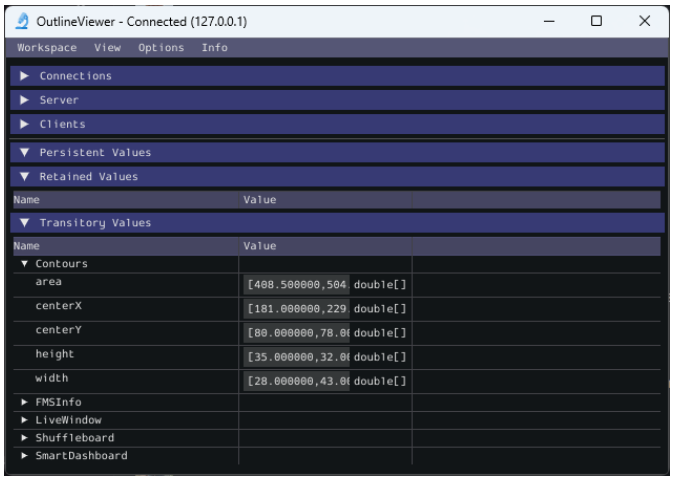
C++ values are now returned/used as value objects (plain `nt::Value`) instead of shared pointers to them (`std::shared_ptr<nt::Value>`).

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.

Comúnmente los valores son para una o más áreas de interés tales como objetivos o piezas de juego y son regresadas múltiples instancias. En el ejemplo a continuación, varios valores x, y, ancho, altura, y áreas son entregados por el procesamiento de imagen y el programa del robot puede decidir cuál de los valores entregados son de interés para los siguientes procesamientos.

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.

Los dos ejemplos siguientes son programas extremadamente simplificados que solamente ilustran el uso de las Tablas de Enrutamiento. Todo el código se encuentra en el método robotInit() que solo se ejecuta cuando el programa inicia. En sus programas, usted buscará obtener los valores del código que evalúen en que dirección apuntar el robot en un comando o un bucle de control durante los periodos de autónomo o teleoperado.

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) {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      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;
}
```

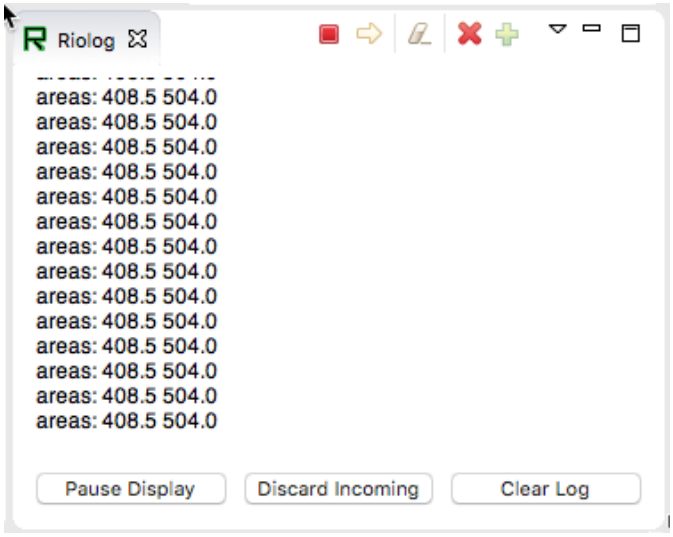
PYTHON

```
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)
```

Los pasos para obtener los valores y, en este programa, escribirlos son:

1. Declarar la tabla variable que guardará la instancia de la subtabla que tendrá los valores.
2. Inicializar la instancia de la subtabla para que pueda ser usada después para recuperar los valores.
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



En este caso el programa solamente está buscando en la matriz de las areas, pero en un ejemplo real sería más probable usar todos los valores. Usando el Riolog en VS Code o la Driver Station Log puede ver los valores así como son enviados. Este programa está usando de muestra una imagen estática así que las areas no cambian, pero puede imaginar que con la cámara en el robot, los valores estarían constantemente cambiando.

29 Path Planning

Path Planning is the process of creating and following trajectories. These paths use the WPILib trajectory APIs for generation and a *Ramsete Controller* for following. This section highlights the process of characterizing your robot for system identification, trajectory following and usage of PathWeaver. Users may also want to read the *generic trajectory following documents* for additional information about the API and non-commandbased usage.

29.1 Notice on Swerve Support

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.

We are sorry for the inconvenience.

29.1.1 Trajectory Tutorial

This is full tutorial for implementing trajectory generation and following on a differential-drive robot. The full code used in this tutorial can be found in the RamseteCommand example project (Java, C++).

Descripción de Tutorial de Trayectoria

Nota: Antes de seguir este tutorial, es útil (pero no necesario) estar familiarizado con las características de WPILib *PID control*, *feedforward*, y *trajectory*

Nota: El código del robot en este tutorial usa el marco de trabajo *command-based*. El espacio de trabajo basado en comandos es altamente recomendado para equipos principiantes.

El objetivo de este tutorial es proporcionar instrucciones de «principio a fin» para implementar una rutina autónoma de seguimiento de la trayectoria para un robot diferencial. Al seguir este tutorial, los lectores aprenderán cómo:

1. Caracterizar con precisión la cadena de tracción de su robot para obtener cálculos precisos de avance y ganancias aproximadas de retroalimentación.
2. Configure un subsistema de unidad para rastrear la pose del robot utilizando la biblioteca de odometría de WPILib.
3. Generar una trayectoria simple a través de un conjunto de waypoints usando la clase `TrajectoryGenerator` de WPILib.
4. Seguir la trayectoria generada en una rutina autónoma usando la clase `RamseteCommand` de WPILib con las ganancias calculadas de feedforward/feedback.

Este tutorial está hecho para que sea accesible para equipos que no tengan tanta experiencia en programación. Mientras que la librería de WPILib ofrece una flexibilidad significativa en la forma en la que se implementan sus características de seguimiento de trayectoria, seguir la implementación de este tutorial debería proporcionar a los equipos una solución relativamente simple, limpia y repetible para el movimiento autónomo.

El código de robot completo para este tutorial se puede encontrar en el RamseteCommand Example Project (Java, C++).

¿Por qué seguir la trayectoria?

Los juegos en FRC® a menudo presentan tareas autónomas que requieren que un robot se mueva de manera efectiva y precisa desde un lugar de inicio conocido a un lugar de puntuación conocido. Históricamente, la solución más común para este tipo de tareas en FRC ha sido un enfoque «drive-turn-drive», es decir, conducir hacia adelante por una distancia conocida, girar por un ángulo conocido y conducir hacia adelante por otra distancia conocida.

Aunque el enfoque «conducir-girar-impulsar» es ciertamente funcional, en los últimos años los equipos han comenzado a rastrear trayectorias que requieren que el robot conduzca y gire al mismo tiempo. Aunque se trata de una tarea técnica fundamentalmente más complicada, ofrece importantes beneficios: en particular, dado que el robot ya no tiene que detenerse para cambiar de dirección, las trayectorias pueden ser recorridas mucho más rápido, lo que permite al robot anotar más piezas de juego durante el período autónomo.

Desde 2020, WPILib suministra ahora a los equipos soluciones de código de trabajo avanzadas para la generación de trayectorias y el seguimiento, educiendo significativamente la «barrier-to-entry» para este tipo de movimiento autónomo avanzado y efectivo.

Equipo Requerido

Para seguir este tutorial, necesitarás acceso a los siguientes materiales:

1. A differential-drive robot (such as the AndyMark AM14U5), equipped with:
   - Codificadores de cuadratura para medir la rotación de la rueda de cada lado de la unidad.
   - Un giroscopio para medir la dirección del robot.
2. Una computadora driver-station configurada con:
   - *FRC Driver Station*.
   - *WPILib*.
   - *The System Identification Toolsuite*.

Paso 1: Caracterización de la conducción del robot

Nota: For detailed instructions on using the System Identification tool, see its *dedicated documentation*.

Nota: The drive identification process requires ample space for the robot to drive. Be sure to have *at least* a 10” stretch (ideally closer to 20”) in which the robot can drive during the identification routine.

Nota: The identification data for this tutorial has been generously provided by Team 5190, who generated it as part of a demonstration of this functionality at the 2019 North Carolina State University P2P Workshop.

Before accurately following a path with a robot, it is important to have an accurate model for how the robot moves in response to its control inputs. Determining such a model is a process called «system identification.» WPILib’s System Identification tool can accurately determine such a model.

Recopilación de los datos

We begin by gathering our drive identification data.

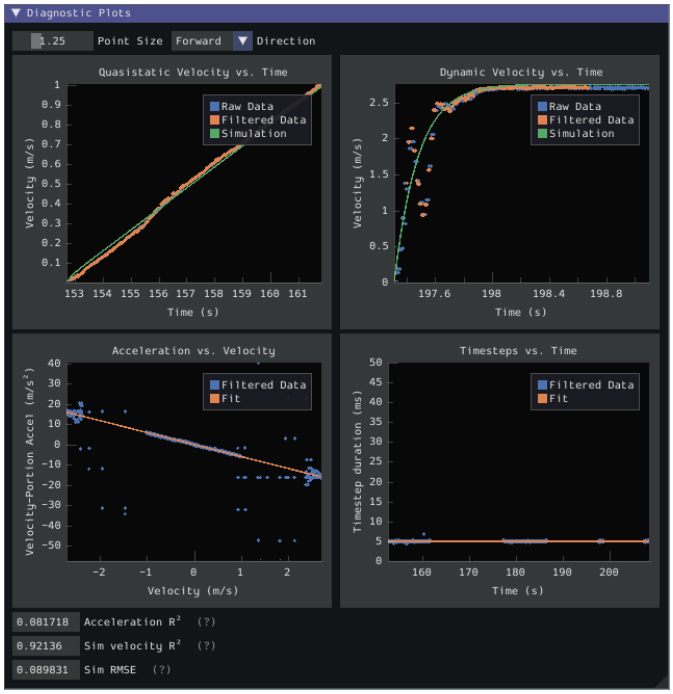
1. *Configure and Deploy your robot project*.
2. *Run the identification Routine*.

Análisis de los datos

Once the identification routine has been run and the data file has been saved, it is time to *open it in the analysis pane*.

Chequeo de Diagnósticos

Per the *system identification guide*, we first view the diagnostics to ensure that our data look reasonable:

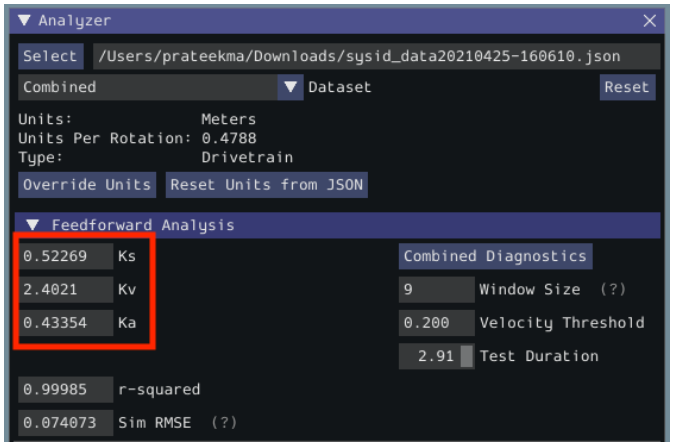


As our data look reasonably linear, and the fit metrics are within acceptable parameters, we proceed to the next step.

Registro de ganancias de compensación

Nota: Las ganancias de compensación no se transfieren, en general, a través de los robots. No uses las ganancias de este tutorial para tu propio robot.

Ahora registramos las ganancias de compensación calculadas por la herramienta:



Como el diámetro de nuestra rueda se especificó en metros, nuestras ganancias de compensación están en las siguientes unidades:

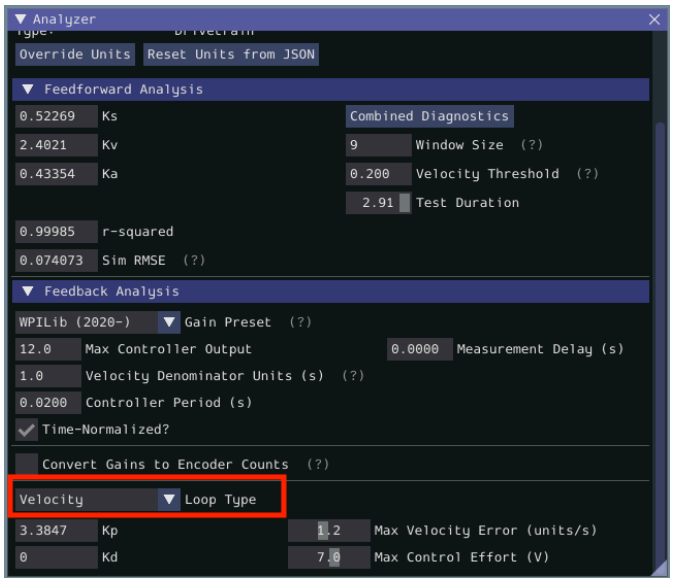
- kS: Volts
- kV: Volts * Segundos / Metros
- kV: Volts * Segundos^2 / Metros

Si ha especificado sus unidades correctamente, sus ganancias de compensación estarán probablemente dentro de un orden de magnitud de las reportadas aquí (existe una posible excepción para kA, que puede ser muy pequeña si su robot es liviano). Si no lo son, es posible que haya especificado uno de sus parámetros de conducción de forma incorrecta al generar su proyecto de robot. Una buena prueba para esto es calcular el valor «teórico» de kV, que es 12 voltios dividido por la velocidad libre teórica de su transmisión (que es, a su vez, la velocidad libre del motor multiplicada por la circunferencia de la rueda dividida por la reducción del engranaje). Este valor debe coincidir muy estrechamente con el kV medido por la herramienta - si no es así, es probable que hayas cometido un error en algún lugar.

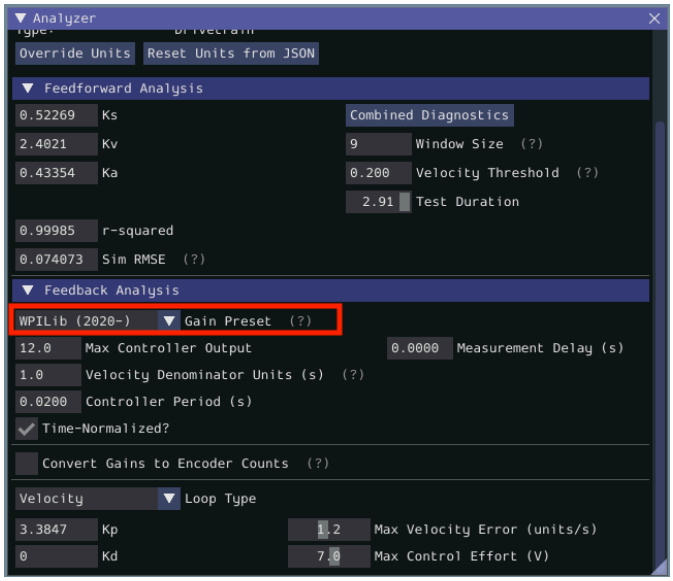
Cálculo de las ganancias de la retroalimentación

Nota: Las ganancias de la retroalimentación no se transfieren, en general, a través de los robots. No uses las ganancias de este tutorial para tu propio robot.

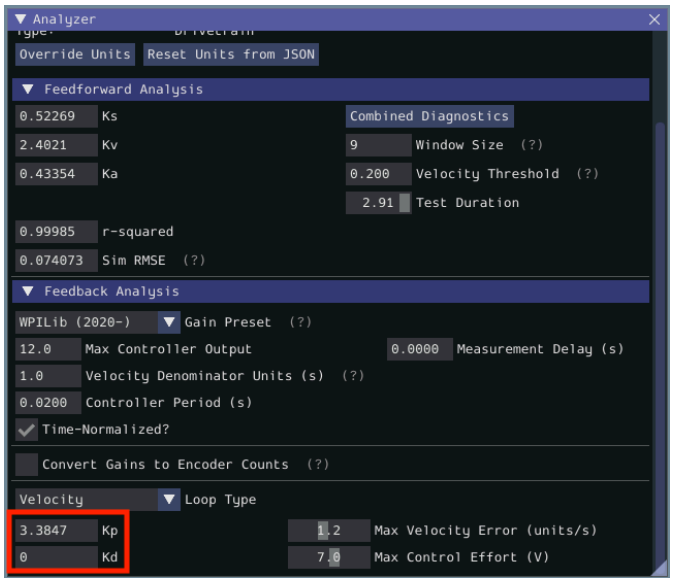
We now *calculate the feedback gains* for the PID control that we will use to follow the path. Trajectory following with WPILib's RAMSETE controller uses velocity closed-loop control, so we first select `Velocity` mode in the identification tool:



Como usaremos el controlador PID WPILib para nuestro bucle de velocidad, seleccionaremos además la opción «WPILib (2020-)`` del menú desplegable «preestablecidos». Esto es *muy* importante, ya que las ganancias de retroalimentación no estarán en las unidades correctas si no seleccionamos el preestablecido correcto:



Finalmente, calculamos y registramos las ganancias de retroalimentación para nuestro bucle de control. Como es un controlador de velocidad, sólo se requiere una ganancia P:



Assuming we have done everything correctly, our proportional gain will be in units of Volts * Seconds / Meters. Thus, our calculated gain means that, for each meter per second of velocity error, the controller will output an additional 3.38 volts.

Paso 2: Introducción de las constantes calculadas

Nota: En C++, es importante que las constantes feedforward se ingresen como el tipo de unidad correcto. Para obtener más información sobre las unidades C++, consulte *Biblioteca de unidades de C++*.

Now that we have our system constants, it is time to place them in our code. The recommended place for this is the `Constants` file of the *standard command-based project structure*.

The relevant parts of the constants file from the RamseteCommand Example Project (Java, C++) can be seen below.

Ganancias de Feedforward/Feedback

Firstly, we must enter the feedforward and feedback gains which we obtained from the identification tool.

Nota: Las ganancias de feedforward y feedback, en general, *no* se transfieren entre robots. *No* use las ganancias de este tutorial para su propio 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

Además, debemos crear una instancia de la clase `DifferentialDriveKinematics`, que nos permite usar el ancho de vía (es decir, la distancia horizontal entre las ruedas) del robot para convertir las velocidades del chasis a las velocidades de las ruedas. Como en otros lugares, mantenemos nuestras unidades en metros.

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;
```

Velocidad máxima de la trayectoria/Aceleración

También debemos decidir sobre una aceleración máxima nominal y una velocidad máxima para el robot durante el seguimiento de la trayectoria. El valor de velocidad máxima debe establecerse algo por debajo de la velocidad libre nominal del robot. Debido al uso posterior del `DifferentialDriveVoltageConstraint`, el valor máximo de aceleración no es extremadamente crucial.

Advertencia: 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;
```

Parámetros de Ramsete

Finalmente, debemos incluir un par de parámetros para el controlador RAMSETE. Los valores que se muestran a continuación deberían funcionar bien para la mayoría de los robots, siempre que las distancias se hayan medido correctamente en metros; para obtener más información sobre cómo ajustar estos valores (si es necesario), consulte *Construcción del objeto controlador Ramsete*.

JAVA

```
60      // Reasonable baseline values for a RAMSETE follower in units of meters and␣
↪seconds
61      public static final double kRamseteB = 2;
62      public static final double kRamseteZeta = 0.7;
```

C++

```
64  // Reasonable baseline values for a RAMSETE follower in units of meters and
65  // seconds
66  inline constexpr auto kRamseteB = 2.0 * 1_rad * 1_rad / (1_m * 1_m);
67  inline constexpr auto kRamseteZeta = 0.7 / 1_rad;
```

Paso 3: creación de un subsistema de desplazamiento o drive

Ahora que nuestra unidad está caracterizada, es hora de comenzar a escribir nuestro código de robot *adecuadamente*. Como se mencionó anteriormente, utilizaremos el Framework *command-based* para nuestro código de robot. En consecuencia, nuestro primer paso es escribir una clase adecuada *subsistema* .

The full drive class from the RamseteCommand Example Project (Java, C++) can be seen below. The rest of the article will describe the steps involved in writing this class.

Java

```
5   package edu.wpi.first.wpilibj.examples.ramsetecommand.subsystems;
6
7   import edu.wpi.first.math.geometry.Pose2d;
8   import edu.wpi.first.math.kinematics.DifferentialDriveOdometry;
9   import edu.wpi.first.math.kinematics.DifferentialDriveWheelSpeeds;
10  import edu.wpi.first.util.sendable.SendableRegistry;
11  import edu.wpi.first.wpilibj.ADXRS450_Gyro;
12  import edu.wpi.first.wpilibj.Encoder;
13  import edu.wpi.first.wpilibj.drive.DifferentialDrive;
14  import edu.wpi.first.wpilibj.examples.ramsetecommand.Constants.DriveConstants;
15  import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
16  import edu.wpi.first.wpilibj2.command.SubsystemBase;
17
18  public class DriveSubsystem extends SubsystemBase {
19    // The motors on the left side of the drive.
20    private final PWMSparkMax m_leftLeader = new PWMSparkMax(DriveConstants.
    ↪kLeftMotor1Port);
21    private final PWMSparkMax m_leftFollower = new PWMSparkMax(DriveConstants.
    ↪kLeftMotor2Port);
22
23    // The motors on the right side of the drive.
24    private final PWMSparkMax m_rightLeader = new PWMSparkMax(DriveConstants.
    ↪kRightMotor1Port);
25    private final PWMSparkMax m_rightFollower = new PWMSparkMax(DriveConstants.
    ↪kRightMotor2Port);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
26
27    // The robot's drive
28    private final DifferentialDrive m_drive =
29        new DifferentialDrive(m_leftLeader::set, m_rightLeader::set);
30
31    // The left-side drive encoder
32    private final Encoder m_leftEncoder =
33        new Encoder(
34            DriveConstants.kLeftEncoderPorts[0],
35            DriveConstants.kLeftEncoderPorts[1],
36            DriveConstants.kLeftEncoderReversed);
37
38    // The right-side drive encoder
39    private final Encoder m_rightEncoder =
40        new Encoder(
41            DriveConstants.kRightEncoderPorts[0],
42            DriveConstants.kRightEncoderPorts[1],
43            DriveConstants.kRightEncoderReversed);
44
45    // The gyro sensor
46    private final ADXRS450_Gyro m_gyro = new ADXRS450_Gyro();
47
48    // Odometry class for tracking robot pose
49    private final DifferentialDriveOdometry m_odometry;
50
51    /** Creates a new DriveSubsystem. */
52    public DriveSubsystem() {
53      SendableRegistry.addChild(m_drive, m_leftLeader);
54      SendableRegistry.addChild(m_drive, m_rightLeader);
55
56      m_leftLeader.addFollower(m_leftFollower);
57      m_rightLeader.addFollower(m_rightFollower);
58
59      // We need to invert one side of the drivetrain so that positive voltages
60      // result in both sides moving forward. Depending on how your robot's
61      // gearbox is constructed, you might have to invert the left side instead.
62      m_rightLeader.setInverted(true);
63
64      // Sets the distance per pulse for the encoders
65      m_leftEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
66      m_rightEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
67
68      resetEncoders();
69      m_odometry =
70          new DifferentialDriveOdometry(
71              m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
    ↪getDistance());
72    }
73
74    @Override
75    public void periodic() {
76      // Update the odometry in the periodic block
77      m_odometry.update(
78          m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
    ↪getDistance());
79    }
```

(continúe en la próxima página)

(proviene de la página anterior)

```
80
81    /**
82     * Returns the currently-estimated pose of the robot.
83     *
84     * @return The pose.
85     */
86    public Pose2d getPose() {
87      return m_odometry.getPoseMeters();
88    }
89
90    /**
91     * Returns the current wheel speeds of the robot.
92     *
93     * @return The current wheel speeds.
94     */
95    public DifferentialDriveWheelSpeeds getWheelSpeeds() {
96      return new DifferentialDriveWheelSpeeds(m_leftEncoder.getRate(), m_rightEncoder.
↪getRate());
97    }
98
99    /**
100    * Resets the odometry to the specified pose.
101    *
102    * @param pose The pose to which to set the odometry.
103    */
104   public void resetOdometry(Pose2d pose) {
105     m_odometry.resetPosition(
106         m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
↪getDistance(), pose);
107   }
108
109   /**
110    * Drives the robot using arcade controls.
111    *
112    * @param fwd the commanded forward movement
113    * @param rot the commanded rotation
114    */
115   public void arcadeDrive(double fwd, double rot) {
116     m_drive.arcadeDrive(fwd, rot);
117   }
118
119   /**
120    * Controls the left and right sides of the drive directly with voltages.
121    *
122    * @param leftVolts the commanded left output
123    * @param rightVolts the commanded right output
124    */
125   public void tankDriveVolts(double leftVolts, double rightVolts) {
126     m_leftLeader.setVoltage(leftVolts);
127     m_rightLeader.setVoltage(rightVolts);
128     m_drive.feed();
129   }
130
131   /** Resets the drive encoders to currently read a position of 0. */
132   public void resetEncoders() {
133     m_leftEncoder.reset();
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    m_rightEncoder.reset();
  }

  /**
   * Gets the average distance of the two encoders.
   *
   * @return the average of the two encoder readings
   */
  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.
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   *
   * @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();
```

(continúe en la próxima página)

(proviene de la página anterior)

```
49
50  /**
51   * Gets the average distance of the TWO encoders.
52   *
53   * @return the average of the TWO encoder readings
54   */
55  double GetAverageEncoderDistance();
56
57  /**
58   * Gets the left drive encoder.
59   *
60   * @return the left drive encoder
61   */
62  frc::Encoder& GetLeftEncoder();
63
64  /**
65   * Gets the right drive encoder.
66   *
67   * @return the right drive encoder
68   */
69  frc::Encoder& GetRightEncoder();
70
71  /**
72   * Sets the max output of the drive.  Useful for scaling the drive to drive
73   * more slowly.
74   *
75   * @param maxOutput the maximum output to which the drive will be constrained
76   */
77  void SetMaxOutput(double maxOutput);
78
79  /**
80   * Returns the heading of the robot.
81   *
82   * @return the robot's heading in degrees, from -180 to 180
83   */
84  units::degree_t GetHeading() const;
85
86  /**
87   * Returns the turn rate of the robot.
88   *
89   * @return The turn rate of the robot, in degrees per second
90   */
91  double GetTurnRate();
92
93  /**
94   * Returns the currently-estimated pose of the robot.
95   *
96   * @return The pose.
97   */
98  frc::Pose2d GetPose();
99
100 /**
101  * Returns the current wheel speeds of the robot.
102  *
103  * @return The current wheel speeds.
104  */
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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
  // 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);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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
  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 {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  return m_gyro.GetRotation2d().Degrees();
}

double DriveSubsystem::GetTurnRate() {
  return -m_gyro.GetRate();
}

frc::Pose2d DriveSubsystem::GetPose() {
  return m_odometry.GetPose();
}

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);
}
```

Configuración de los codificadores de unidad

Los codificadores de transmisión miden la rotación de las ruedas en cada lado de la transmisión. Para configurar correctamente los codificadores, necesitamos especificar dos cosas: los puertos a los que están conectados los codificadores y la distancia por pulso del codificador. Luego, necesitamos escribir métodos que permitan el acceso a los valores del codificador desde el código que usa el subsistema.

Puertos del codificador

Los puertos del codificador se especifican en el constructor del codificador, así:

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]},
```

Distancia del codificador por pulso

The distance per pulse is specified by calling the encoder's `setDistancePerPulse` method. Note that for the WPILib Encoder class, «pulse» refers to a full encoder cycle (i.e. four edges), and thus will be 1/4 the value that was specified in the SysId config. Remember, as well, that the distance should be measured in meters!

Java

```
    m_leftEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
    m_rightEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
```

C++ (Source)

```
  m_leftEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
  m_rightEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
```

Método de acceso del codificador

Para acceder a los valores medidos por los codificadores, incluimos el siguiente método:

Importante: ¡Las velocidades devueltas **deben** estar en metros! Dado que hemos configurado la distancia por pulso en los codificadores anteriormente, al llamar a `getRate()` se aplicará automáticamente el factor de conversión de las unidades del codificador a metros. Si no estás usando la clase `Encoder` de WPILib, debes realizar esta conversión a través de la API del proveedor respectivo o multiplicando manualmente por un factor de conversión.

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)

```
88 frc::DifferentialDriveWheelSpeeds DriveSubsystem::GetWheelSpeeds() {
89   return {units::meters_per_second_t{m_leftEncoder.GetRate()},
90           units::meters_per_second_t{m_rightEncoder.GetRate()}};
91 }
```

Envolvemos los valores medidos del codificador en un objeto `DifferentialDriveWheelSpeeds` para facilitar la integración con la clase `RamseteCommand` más adelante.

Configurar el giroscopio

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

```
45   // The gyro sensor
46   private final ADXRS450_Gyro m_gyro = new ADXRS450_Gyro();
```

C++ (Header)

```
134  // The gyro sensor
135  frc::ADXRS450_Gyro m_gyro;
```

Método de acceso al giroscopio

Para acceder al rumbo actual medido por el giroscopio, incluimos el siguiente método:

Java

```
178  /**
179   * Returns the heading of the robot.
180   *
181   * @return the robot's heading in degrees, from -180 to 180
182   */
183  public double getHeading() {
184    return m_gyro.getRotation2d().getDegrees();
185  }
```

C++ (Source)

```
76 units::degree_t DriveSubsystem::GetHeading() const {
77   return m_gyro.GetRotation2d().Degrees();
78 }
```

Configuración de la odometría

Ahora que tenemos nuestros codificadores y giroscopio configurados, es hora de configurar nuestro subsistema de transmisión para calcular automáticamente su posición a partir de las lecturas del codificador y del giroscopio.

Primero, creamos una instancia de miembro de la clase `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}} {
```

Actualización de la odometría

La clase de odometría debe actualizarse periódicamente para incorporar nuevas lecturas del codificador y el giroscopio. Logramos esto dentro del método `periódico` del subsistema, que se llama automáticamente una vez por iteración del ciclo principal:

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()});
}
```

Método de acceso de odometría

Para acceder a la pose calculada actual del robot, incluimos el siguiente método:

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();
}
```

Importante: 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.

Método de accionamiento basado en voltaje

Finally, we must include one additional method - a method that allows us to set the voltage to each side of the drive using the `setVoltage()` method of the `MotorController` interface. The default WPILib drive class does not include this functionality, so we must write it ourselves:

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();
}
```

It is very important to use the `setVoltage()` method rather than the ordinary `set()` method, as this will automatically compensate for battery «voltage sag» during operation. Since our feedforward voltages are physically-meaningful (as they are based on measured identification data), this is essential to ensuring their accuracy.

Advertencia: `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.

Paso 4: Creando y Siguiendo una trayectoria

Con nuestro subsistema de accionamiento escrito, es hora de generar una trayectoria y escribir un comando autónomo para seguirla.

As per the *standard command-based project structure*, we will do this in the `getAutonomousCommand` method of the `RobotContainer` class. The full method from the RamseteCommand Example Project (Java, C++) can be seen below. The rest of the article will break down the different parts of the method in more detail.

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)),
```

(continúe en la próxima página)

(proviene de la página anterior)

```
            // End 3 meters straight ahead of where we started, facing forward
            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++ (Fuente)

```
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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      {frc::Translation2d{1_m, 1_m}, frc::Translation2d{2_m, -1_m}},
      // 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); }, {}));
}
```

Configurando las Restricciones de Trayectoria

Primero, debemos establecer algunos parámetros de configuración para la trayectoria que aseguren que la trayectoria generada sea seguible.

Creando una Restricción de Voltaje

La primera pieza de configuración que necesitaremos es una restricción de voltaje. Esto asegurará que la trayectoria generada nunca ordene al robot ir más rápido de lo que es capaz de lograr con el suministro de voltaje dado:

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),
            DriveConstants.kDriveKinematics,
            10);
```

C++ (Fuente)

```
  // 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};
```

Nótese que fijamos el votaje máximo a 10V, en lugar del voltaje nominal de la batería de 12V. Esto nos da un poco de «margen» para lidiar con la «caída de voltaje» durante la operación.

Creando la Configuración

Ahora que tenemos nuestra restricción de voltaje, podemos crear nuestra instancia `TrajectoryConfig`, que envuelve todas nuestras restricciones de camino:

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++ (Fuente)

```
  // 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);
```

Generando la Trayectoria

Con la configuración de nuestra trayectoria en mano, estamos listos para generar nuestra trayectoria. Para este ejemplo, generaremos una trayectoria «clamped cubic» - esto significa que especificaremos las posiciones completas del robot en los puntos finales, y las posiciones sólo para los waypoints interiores (también conocidos como «knot points»). Como en cualquier otro lugar, todas las distancias están en metros.

Java

```
    // An example trajectory to follow. All units in meters.
    Trajectory exampleTrajectory =
        TrajectoryGenerator.generateTrajectory(
            // Start at the origin facing the +X direction
            new Pose2d(0, 0, new Rotation2d(0)),
            // Pass through these two interior waypoints, making an 's' curve path
            List.of(new Translation2d(1, 1), new Translation2d(2, -1)),
            // End 3 meters straight ahead of where we started, facing forward
            new Pose2d(3, 0, new Rotation2d(0)),
            // Pass config
            config);
```

C++ (Fuente)

```
  // 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}},
      // End 3 meters straight ahead of where we started, facing forward
      frc::Pose2d{3_m, 0_m, 0_deg},
      // Pass the config
      config);
```

Nota: Instead of generating the trajectory on the roboRIO as outlined above, one can also *import a PathWeaver JSON*.

Creando el RamseteCommand

Primero reajustaremos la posición de nuestro robot a la posición inicial de la trayectoria. Esto asegura que la posición del robot en el sistema de coordenadas y la posición inicial de la trayectoria sean las mismas.

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++ (Fuente)

```
  // 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 *Transformando trayectorias*.

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++ (Fuente)

```
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})};
```

Esta declaración es bastante sustancial, así que la revisaremos argumento por argumento:

1. La trayectoria: Esta es la trayectoria a seguir; en consecuencia, pasamos el comando de la trayectoria que acabamos de construir en nuestros pasos anteriores.
2. The pose supplier: This is a method reference (or lambda) to the *drive subsystem method that returns the pose*. The RAMSETE controller needs the current pose measurement to determine the required wheel outputs.
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. The wheel speed supplier: This is a method reference (or lambda) to the *drive subsystem method that returns the wheel speeds*
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. The output consumer: This is a method reference (or lambda) to the *drive subsystem method that passes the voltage outputs to the drive motors*.
10. El impulso del robot: Este es el subsistema de accionamiento en sí, incluido para asegurar que el comando no funcione en el accionamiento al mismo tiempo que cualquier otro comando que utilice el accionamiento.

Finalmente, note que añadimos un comando final de «stop» en secuencia después del comando de seguimiento de la trayectoria, para asegurar que el robot deje de moverse al final de la trayectoria.

Video

Si todo ha ido bien, la rutina autónoma de tu robot debería ser algo así:

29.1.2 PathWeaver

Nota: 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.

Introducción a PathWeaver

Nota: 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.

El período autónomo es una sección muy importante en un match; es emocionante cuando los robots realizan cosas impresionante en el período autónomo. Para conseguirlo, el robot normalmente necesita ir a algún sitio. Cuanto más rápido el robot llega a ese lugar, ¡más pronto podrá ganar puntos! El método tradicional para el período autónomo es conducir en línea recta, girar a cierta ángulo, y conducir en línea recta de nuevo. Este enfoque funciona bien, pero el robot pasa una cantidad no despreciable de tiempo parando y volviendo a empezar después de cada línea recta y cada giro.

Un enfoque más avanzado de la autonomía se denomina «planificación de la trayectoria». En lugar de conducir en línea recta y girar una vez completada la línea, el robot se desplaza continuamente, conduciendo con un movimiento similar a una curva. Esto puede reducir el tiempo de parada para girar.

WPILib contiene un conjunto de generación de trayectorias que puede ser utilizado por los equipos para generar y seguir trayectorias. Esta serie de artículos repasará cómo generar y visualizar trayectorias usando PathWeaver. Para un completo tutorial sobre seguir trayectorias, por favor visite *end-to-end trajectory tutorial*.

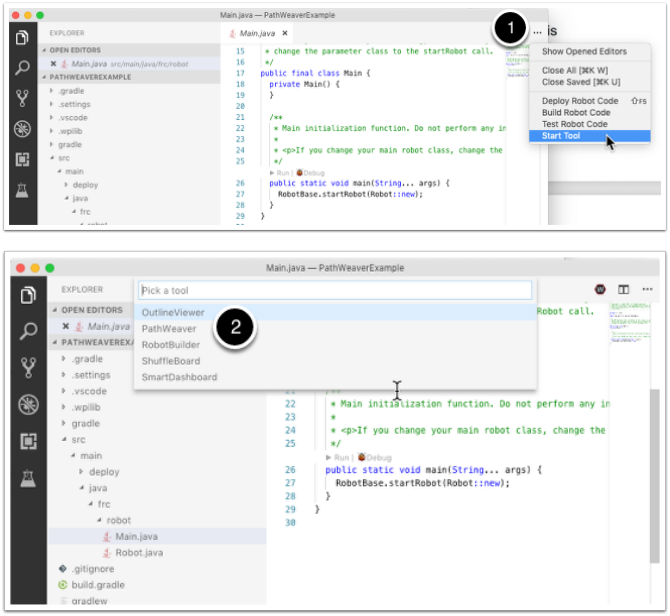
Nota: *Seguir la trayectoria* es necesario para utilizar PathWeaver. Le recomendamos que comience con el seguimiento de trayectorias y lo haga funcionar con rutas simples. A partir de ahí puede continuar probando trayectorias más complicadas generadas por PathWeaver.

Crear un Proyecto de PathWeaver

PathWeaver es la herramienta usada para dibujar rutas que debe seguir el robot. Las trayectorias para un solo programa se almacenan en un proyecto de PathWeaver.

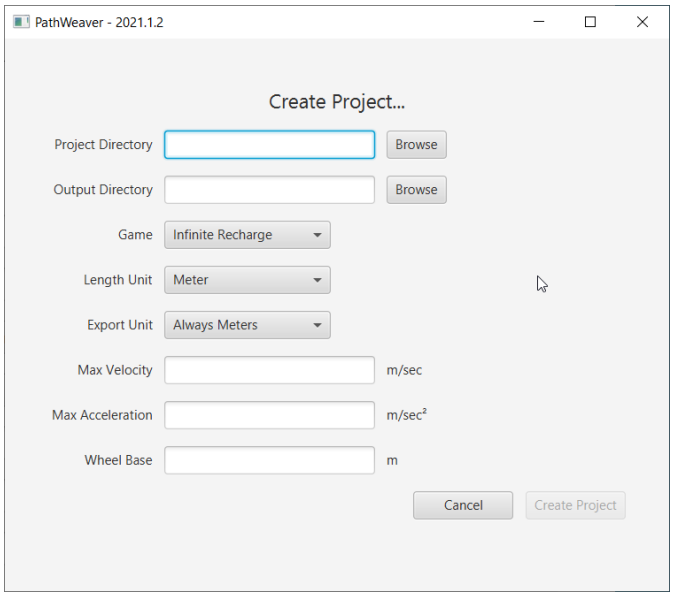
Iniciando PathWeaver

PathWeaver se inicia haciendo clic en el ícono de los tres puntos en la parte superior de la esquina de la interfaz de Visual Studio Code. Luego, haga clic en el “Start tool” o herramienta de inicio y luego haga clic en «PathWeaver» como se muestra a continuación.



Creando el proyecto

Para crear un proyecto PathWeaver, haga clic en «Create project» y luego llene la creación del proyecto. Note que al pasar el cursor sobre cualquiera de los campos del formulario se mostrará más información acerca de lo que se requiere.



Project Directory: Este es normalmente el directorio de proyecto de nivel superior que contiene el build.gradle y archivos src para el programa de su robot. Elegir este directorio es la forma esperada de usar PathWeaver y hará que localice todos los archivos de salida en los directorios correctos para el despliegue automático de la ruta a su robot.

Output directory: El directorio donde se almacenan las rutas para su implementación en su robot. Si especificó la carpeta del proyecto de nivel superior de nuestro proyecto de robot en el paso anterior (como se recomienda), completar el directorio de salida es opcional.

Juego: El juego (el juego FRC ® que se está usando) hará que se use la superposición de imagen de campo correcta. También puede crear sus propias imágenes de campo y el procedimiento se describirá más adelante en esta serie.

Length Unit: Las unidades que se utilizarán para describir su robot y para las mediciones de campo al visualizar trayectorias con PathWeaver.

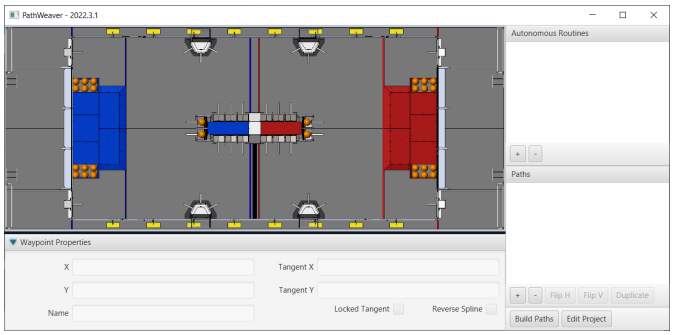
Export Unit: Las unidades que se utilizarán al exportar las rutas y los puntos de referencia. Si está planeando utilizar WPILib Trajectories, entonces debe elegir *Always Meters*. Si elige *Same as Project* se exportará en las mismas unidades que *Length Unit*.

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: Se refiere a la distancia entre los ejes del chasis del robot. Esto se usa para asegurar que ninguna rueda de un accionamiento diferencial pasará por encima de la velocidad máxima especificada en las vueltas.

Interfaz de usuario de PathWeaver

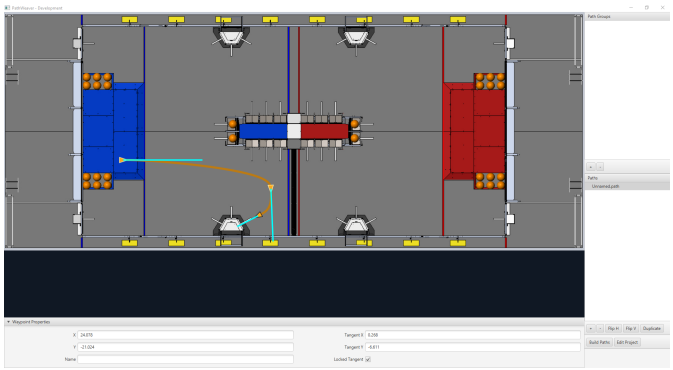


La interfaz de usuario de PathWeaver consiste en lo siguiente:

1. El área del campo de juego en la esquina superior izquierda. Las trayectorias son dibujadas en esta parte del programa.
2. Las propiedades de la actual ruta seleccionada se muestran en el panel inferior. Estas propiedades incluyen la X & Y, junto con las tangentes en cada punto.
3. Un grupo de rutas (o un modo «autónomo») es mostrado en la ventana superior derecha. Esta es una conveniente forma para ver todas las trayectorias en un solo modo autónomo.
4. Las trayectorias individuales que un robot podría seguir se muestran en la parte inferior derecha de la ventana.
5. El botón “Build Paths” o construir trayectorias, exportará las trayectorias en un formato JSON. Estos archivos JSON pueden ser usados del código del robot para seguir la trayectoria.
6. El botón “Edit Project” o editar proyecto le permite editar las propiedades del proyecto.

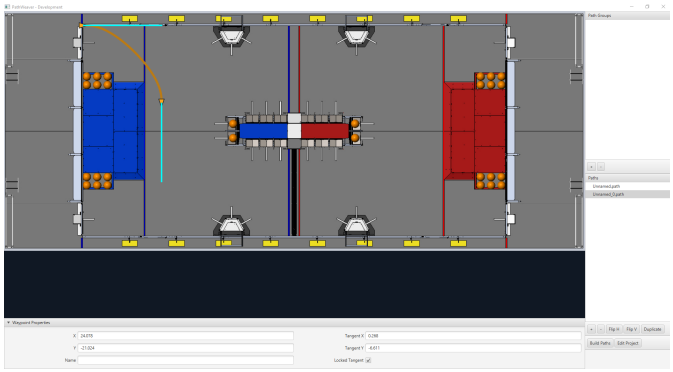
Visualizar las Trayectorias de PathWeaver

La principal característica de PathWeaver es visualizar trayectorias. Las siguientes imágenes muestran una trayectoria suave que representa una trayectoria que un robot podría tomar durante el período autónomo. Las rutas pueden tener cualquier número de puntos que permiten una conducción más compleja para ser descritas. En este caso hay 3 puntos de ruta (incluyendo el inicio y la parada) representados con los íconos de triángulos. Cada punto de ruta consiste de una posición X, Y en el campo de juego, así como una dirección del robot descritas como las líneas tangentes X & Y.



Crear la Trayectoria Inicial

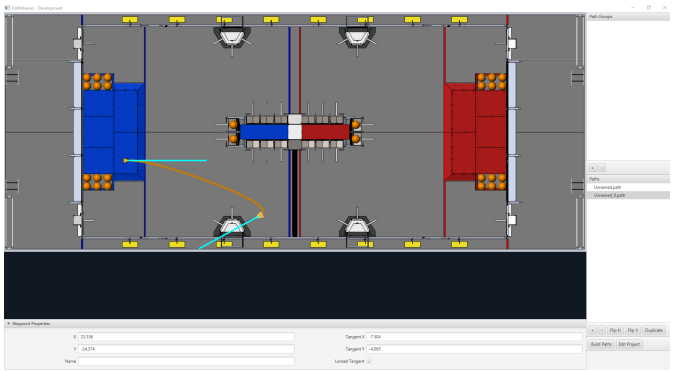
Para empezar a crear una trayectoria, haga clic en el botón + (más) en la ventana Path. Una trayectoria por defecto se creará que probablemente no tenga los puntos de inicio y final adecuados que se desea. El camino también muestra los vectores tangentes (líneas azul verdoso) para los puntos de inicio y final. Cambiando el ángulo de los vectores tangentes cambia la forma de la trayectoria.



Arrastre los puntos de inicio y final de la trayectoria a los lugares deseados por usted. Note que, en este caso, la trayectoria por defecto no comienza en un punto legal para el juego de 2019 Deep Space. Podemos arrastrar el punto inicial para hacer que el robot se ponga en marcha en el HAB o hábitat: punto de partida del robot durante la temporada Deep Space en 2019.

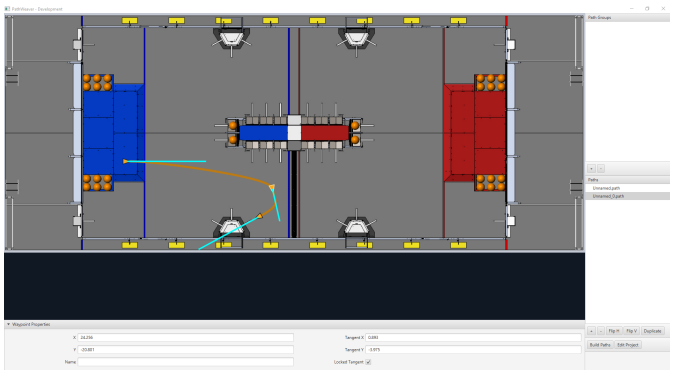
Cambiar el Rumbo del Punto de Ruta

La dirección del robot puede ser cambiada arrastrando la línea del vector tangente (línea azul verdosa). En la siguiente imagen, el punto final fue arrastrado a la posición final deseada y fue rotado para estar frente al cohete.



Añadir puntos de ruta adicionales para controlar la ruta que seguirá el robot.

Añadir puntos de ruta adicionales y cambiar los vectores tangentes puede afectar el camino que es seguido. Los puntos de ruta adicionales pueden ser añadidos arrastrando la ruta de en medio. En este caso, añadimos otro punto de ruta en medio del camino.



Bloqueando las líneas tangentes.

Bloquear las líneas tangentes impide que cambien cuando se manipula la trayectoria. Las líneas tangentes también se bloquearán cuando el punto es movido.

Control más preciso de los puntos de ruta

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.

Creating Autonomous Routines

Autonomous Routines (also known as Path Groups) are a way of visualizing where one path ends and the next one starts. An example is when the robot program drives one path, does something after the path has completed, drives to another location to obtain a game piece, then back again to score it. It’s important that the start and end points of each path in the routine have common end and start points. By adding all the paths to a single autonomous routine and selecting the routine, all paths in that routine will be shown. Then each path can be edited while viewing all the paths.

Creating an Autonomous Routine

Press the + button underneath Autonomous Routines. Then drag the Paths from the Paths section into your Autonomous Routine.

Each path added to an autonomous routine will be drawn in a different color making it easy to figure out what the name is for each path.

If there are multiple paths in a routine, selection works as follows:

1. Selecting the routine displays all paths in the routine making it easy to see the relationship between them. Any waypoint on any of the paths can be edited while the routine is selected and it will only change the path containing the waypoint.
2. Selecting on a single path in the routine will only display that path, making it easy to precisely see what all the waypoints are doing and preventing clutter in the interface if multiple paths cross over or are close to each other.

Importar un archivo JSON de PathWeaver

La clase `TrajectoryUtil` puede ser usada para importar un PathWeaver JSON en el código del robot para seguirlo. Este artículo repasará la importación de la trayectoria. Por favor, visite *end-to-end trajectory tutorial* para más información sobre cómo seguir la trayectoria.

Los métodos estáticos `fromPathweaverJson` (Java) / `FromPathweaverJson` (C++) en `TrajectoryUtil` pueden utilizarse para crear una trayectoria a partir de un archivo JSON almacenado en el sistema de archivos de la roboRIO.

Importante: Para ser compatible con la vista `Field2d` del simulador GUI, las coordenadas del JSON han cambiado. Anteriormente (antes del 2021), el rango de la coordenada y era de -27 a 0 pies, mientras que ahora el rango de la coordenada y es de 0 a 27 pies (con 0 estando en la parte inferior de la pantalla y 27 pies en la parte superior). Esto no debería afectar a los equipos que están correctamente *restableciendo su odometría a la posición inicial de la trayectoria* antes de seguirla.

Nota: PathWeaver coloca los archivos JSON en `src/main/deploy/paths` que automáticamente se colocarán en el sistema de archivos de la roboRIO en `/home/lvuser/deploy/paths` y se podrá acceder a esto usando getDeployDirectory como se muestra a continuación.

JAVA

```
String trajectoryJSON = "paths/YourPath.wpilib.json";
Trajectory trajectory = new Trajectory();

@Override
public void robotInit() {
   try {
      Path trajectoryPath = Filesystem.getDeployDirectory().toPath().
↪resolve(trajectoryJSON);
      trajectory = TrajectoryUtil.fromPathweaverJson(trajectoryPath);
   } catch (IOException ex) {
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      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());
}
```

En los ejemplos de arriba, `YourPath` debe de ser reemplazado por el nombre de su path/camino.

Advertencia: Cargar un JSON de PathWeaver desde un archivo en Java puede llevar más de una iteración de bucle, por lo que se recomienda encarecidamente que el robot cargue estas rutas al iniciarse.

Añadiendo Imágenes dela Cancha a PathWeaver

Aquí están las instrucciones para añadir sus propias imágenes del campo usando el juego de 2019 como ejemplo.

Los juegos se cargan desde el `~/PathWeaver/Games` en Linux y macOS o el directorio `%USERPROFILE%/PathWeaver/Games` en Windows. Los archivos pueden estar en un subdirectorio específico del juego, o en un archivo ZIP en el directorio Games. El archivo ZIP debe seguir el mismo diseño que un directorio del juego; el archivo JSON debe estar en la raíz del archivo ZIP (no puede estar en un subdirectorio).

Download the example *FIRST* Destination Deep Space field definition `here`. Other field definitions are available in the allwpilib GitHub repository.

Diseño del Archivo

```
~/PathWeaver
  /Games
   /Custom Game
      custom-game.json
      field-image.png
   OtherGame.zip
```

Formato 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"
}
```

La ruta de la imagen del campo es relativa al archivo JSON. Para simplificar, el archivo de la imagen debe estar en el mismo directorio que el archivo JSON.

Las esquinas del campo son las coordenadas X & Y de los píxeles de arriba a la izquierda y de abajo a la derecha que definen el límite rectangular del área de juego en la imagen del campo. Las áreas de juego no rectangulares no son compatibles.

El tamaño del campo es el ancho y el largo del área jugable del campo en las unidades proporcionadas.

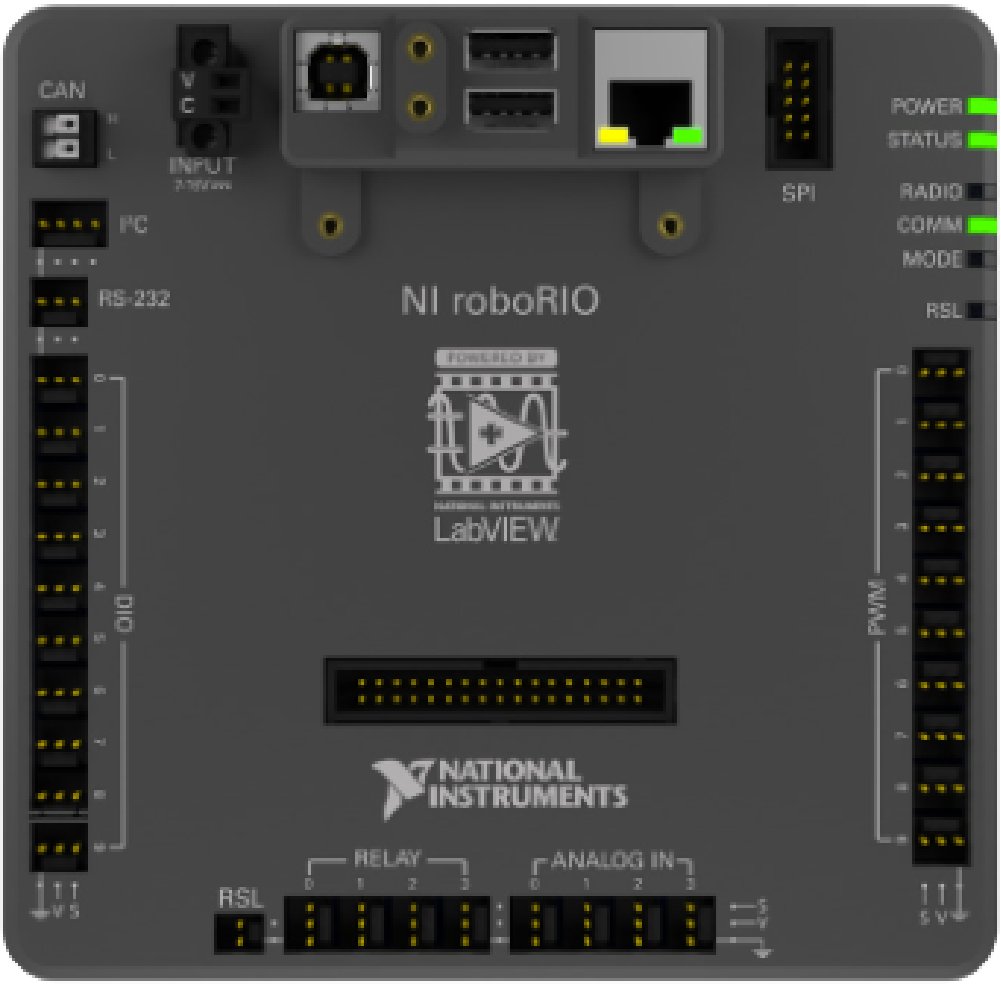
Las unidades de campo no distinguen entre mayúsculas y minúsculas y pueden estar en metros, centímetros, milímetros, pulgadas, pies, yardas o millas. Se admiten el singular, el plural y las abreviaturas(por ejemplo, «meter», «meters», y «m» son todas válidas para especificar la unidad de metros).

Nota: Al crear una nueva imagen de campo, se debe dejar un borde (se recomienda un mínimo de 20 píxeles) alrededor del exterior para que los waypoints del borde del campo sean accesibles.

30

roboRIO

30.1 Introducción a la roboRIO



La roboRIO está diseñado específicamente con FIRST en mente. La roboRIO tiene una arquitectura básica de procesadores en tiempo real + FPGA (matriz de puertas programables

en campo) pero es más potente, más ligera y más pequeña que algunos sistemas similares utilizados en la industria.

La roboRIO es un controlador de robótica reconfigurable que incluye puertos incorporados para circuitos integrados (12C), interfases periféricas seriales (SPI), RS232, USB, Ethernet, modulación de ancho de pulso (PWM), y relés para conectar rápidamente los sensores y actuadores comunes usados en robótica. El controlador incluye LEDs, botones, un acelerómetro integrado, y un puerto electrónico personalizado. Tiene integrado un procesador Cortex-A9 en tiempo real ARM de doble núcleo y un FPGA Xilinx personalizable.

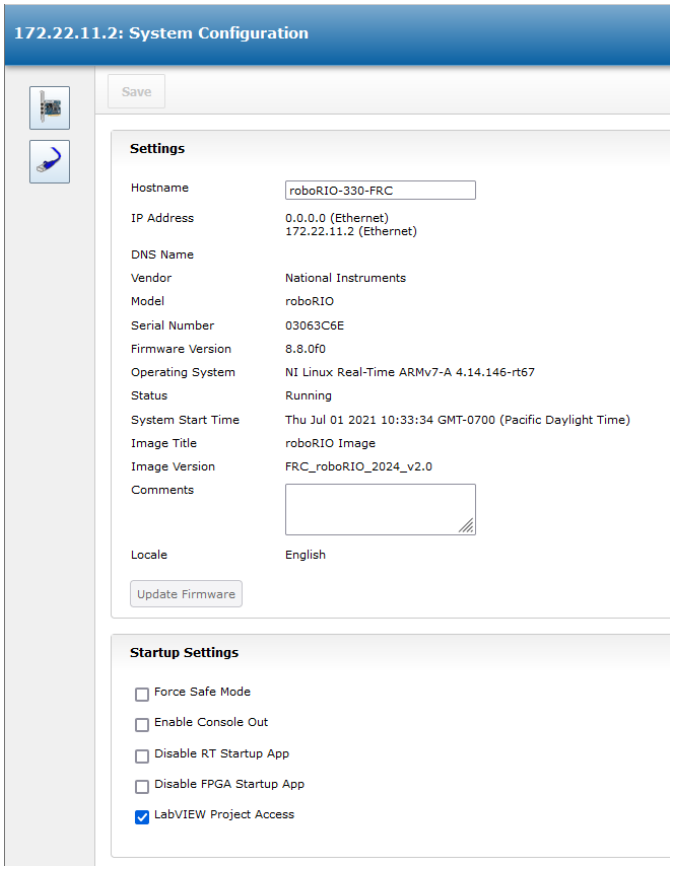
Detailed information on the roboRIO can be found in the roboRIO User Manual and in the roboRIO technical specifications.

Before deploying programs to your roboRIO, you must first image the roboRIO: *roboRIO 1 roboRIO 2*.

30.2 RoboRIO Web DashBoard

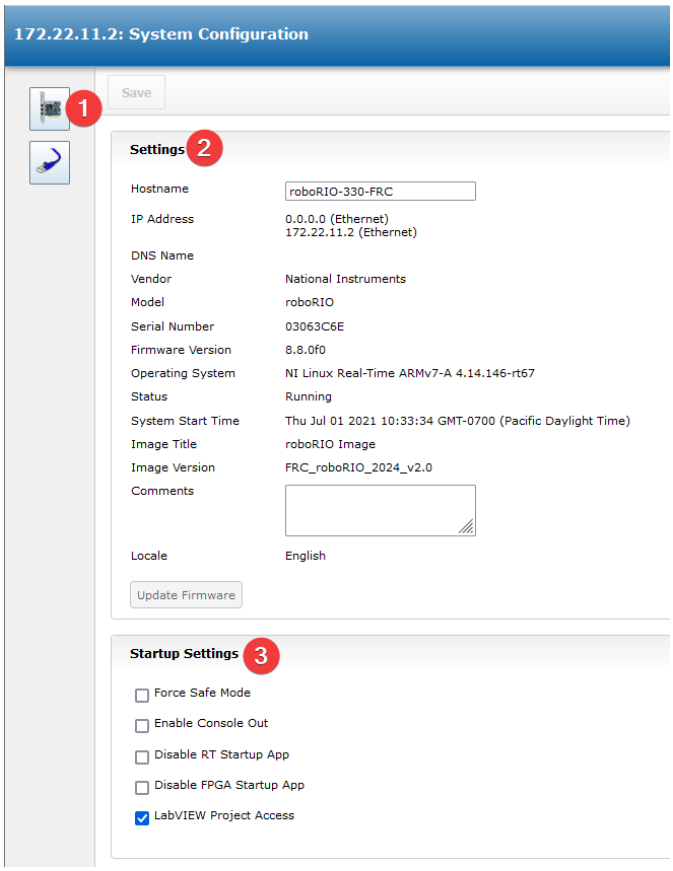
El roboRIO web DashBoard es una página web compilada en el roboRIO que puede ser usada para comprobar el estado y configuraciones de las actualizaciones del roboRIO.

30.2.1 Abriendo el WebDash



To open the web dashboard, open a web browser and enter the address of the roboRIO into the address bar (172.22.11.2 for USB, or «roboRIO-#####-FRC.local where ##### is your team number, with no leading zeroes, for either interface). See this document for more details about mDNS and roboRIO networking: *Configuraciones IP*

30.2.2 Pestaña de configuración del sistema



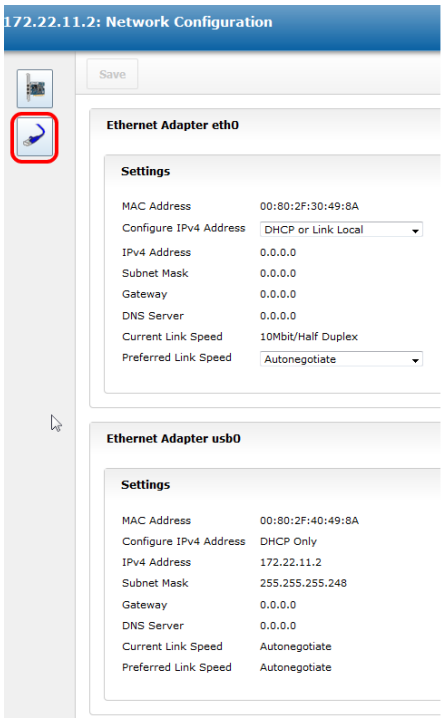
La pantalla de inicio del web Dashboard es la pestaña de configuración del sistema la cual tiene 5 secciones principales:

1. Barra de navegación - Esta sección le permite navegar a diferentes secciones del web Dashboard. Las diferentes paginas accesibles mediante esta barra de navegación se comentará abajo.
2. Configuración del sistema - Esta sección contiene la información acerca de la configuración de sistema. El campo del Hostname no deberá ser modificado manualmente, en su lugar se deberá usar la herramienta Imaging del roboRIO para establecer el Hostname basado en el número de su equipo. Esta sección contiene información tal como la IP del dispositivo, la versión del firmware y la versión de la imagen.
3. Configuración de inicio - Esta sección contiene la configuración de inicio del roboRIO. Estos se describirán los siguientes pasos.
4. Recursos del sistema (sin imagen): Esta sección proporciona una imagen de los recursos del sistema tal como la memoria y la carga de la CPU.

Configuración de Inicio

- Modo de forzado seguro - Forzar el controlador de manera segura. Esto puede usarse para solucionar problemas de imagen, pero es recomendado usar el botón de Reset en el RoboRIO para poder el dispositivo en Modo seguro (con voltaje, apretar el botón Reset por 5 segundos). **Valor de casilla predeterminado, no marcado.**
- 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.**
- Aplicación para deshabilitar RT Startup - Marcar esta casilla inhabilita el código para ejecutarse al inicio. Esto puede ser usado para solucionar problemas cuando el roboRIO no responde a un programa nuevo descargado. Valor de casilla predeterminado, no marcado.
- Aplicación para deshabilitar FPGA Startup - **Esta casilla no debe ser marcada.**
- LabVIEW Project Access - **It is recommended to leave this box checked.** This setting allows LabVIEW projects to access the roboRIO.

30.2.3 Configuración de Red



Esta página muestra la configuración del adaptador de red del roboRIO. **No es recomendable hacer algún cambio en esta página.** Para más información de la red del roboRIO ver el siguiente artículo: *Configuraciones IP*

30.3 roboRIO FTP

Nota: El roboRIO tiene ambas STFP y anónimas FTP habilitadas. Este artículo describe cómo usar cada una para acceder al archivo del sistema del roboRIO.

30.3.1 SFTP

SFTP es la manera recomendada para acceder al archivo del sistema del roboRIO. Porque usted usara la misma cuenta que su programa mientras se ejecuta, archivos copiados siempre deberán tener permisos de compatibilidad de su código.

Software

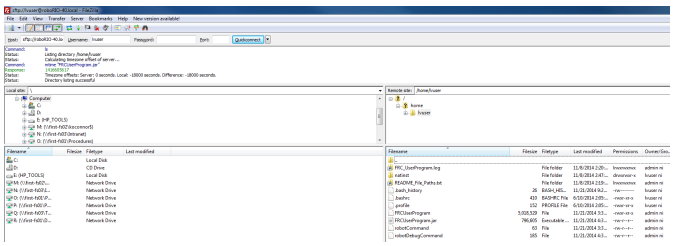
Hay varios programas disponibles gratuitamente para SFTP- Este artículo hablara el uso de FileZilla. Usted puede descargar e instalar FileZilla antes de proceder o extrapolar las direcciones a continuación de su cliente de preferencia,

Conectarse al roboRIO

Para conectarse al roboRIO:

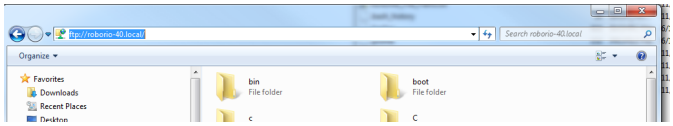
1. Escribir el nombre mDNS (roboRIO-TEAM-frc.local) en la casilla “Host”.
2. Escribir “lvuser” en la casilla Username (es la cuenta con la que se ejecuta el programa)
3. Dejar vacía la casilla de contraseña.
4. Escribir “22” en el casilla port (el puerto predeterminado SFTP).
5. Clic en Quickconnect

Navegando por el sistema de archivos del roboRIO



Después de conectarse al roboRIO, FileZilla abrirá el directorio \home\lvusar. El panel derecho es para el sistema remoto (el roboRIO), el panel izquierdo es para el sistema local (su computadora). La parte superior de cada panel muestra la jerarquía del directorio que está navegando, el panel inferior muestra el contenido del directorio. Para transferir archivos, simplemente dele clic y arrastre de un lado a otro. Para crear directorios en el roboRIO, dar clic con botón derecho y seleccionar “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 Cuentas de usuarios de roboRIO y SSH

Nota: Este documento contiene temas avanzados que no son necesarios para la programación común de FRC®

La imagen del roboRIO contiene un número de cuentas, este artículo destacará las dos utilizadas para el FRC y proporcionará algunos detalles sobre su propósito. También describirá cómo conectarse al roboRIO a través de SSH.

30.4.1 Cuentas de usuarios de roboRIO

La imagen del roboRIO contiene un número de cuentas de usuarios, pero hay dos de mayor interés para FRC.

admin

La cuenta de "admin" tiene un root Access al sistema y puede ser usado para manipular archivos OS o configuraciones. Los equipos deberán tener cuidado cuando se usa esta cuenta ya que permite modificar configuraciones y archivos que podrían corromper el sistema operativo del roboRIO. Las credenciales de esta cuenta son:

```
Usuario: admin
```

```
Contraseña:
```

Nota: La contraseña está intencionalmente vacía

lvuser

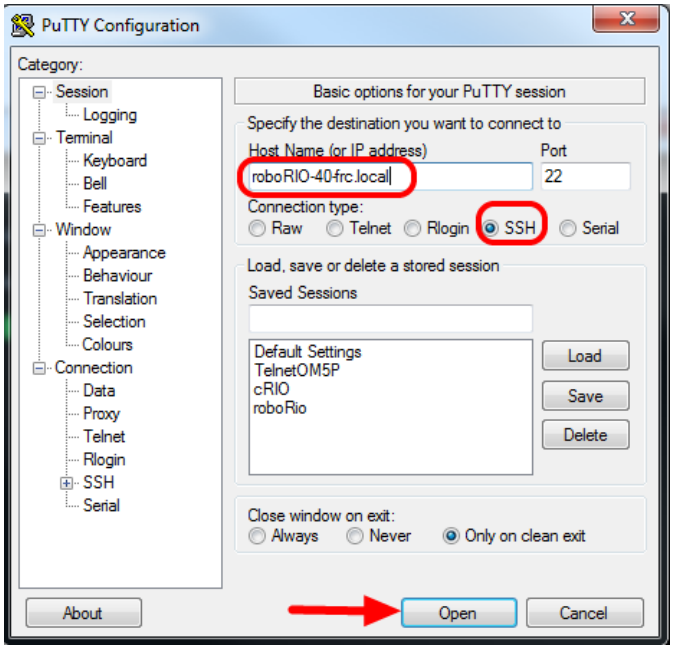
La cuenta "lvuser" es una cuenta usada para ejecutar el código del usuario para los 3 lenguajes. Las credenciales para esta cuenta no deberán cambiarse. Los equipos podrían querer usar esta cuenta (via ssh o sftp) cuando estén trabajando con el roboRIO para asegurar que los archivo o los cambios de configuración se realicen en la misma cuenta donde se ejecuta el archivo.

Peligro: 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) es un protocolo usado para la comunicación segura de datos. Cuando se hace referencia a un sistema Linux (como el que se ejecuta en el roboRIO) generalmente se refiere al acceso de la línea de comando de la consola usando el protocolo SSH. Esto puede ser usado para ejecutar comandos en el sistema remoto. Un cliente gratuito que se puede usar para SSH es PuTTY: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Abrir Putty



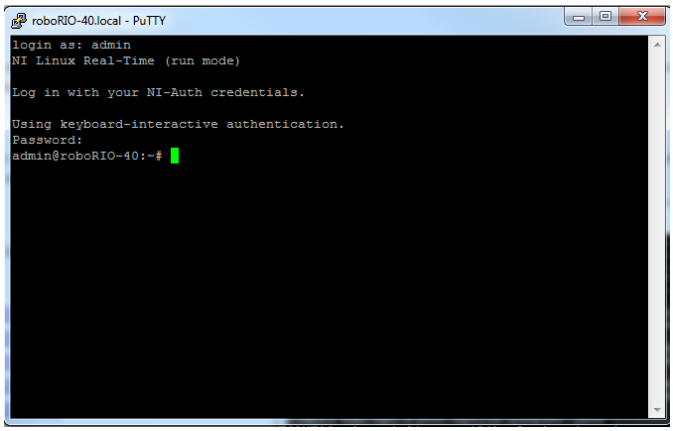
Abrir Putty (Dar clic en OK a cualquier aviso de seguridad). Enseguida establezca la siguiente configuración:

1. Host name: roboRIO-TEAM-frc-loc (en TEAM escribir el número de su equipo, el ejemplo muestra al equipo 40)
2. Tipo de conexión: SSH.

Otras configuraciones pueden dejarse predeterminadas. Dar clic en abrir la conexión. Si aparece un aviso acerca índices de SSH, dele clic OK.

Si usted está conectado mediante USB puede usar 172.22.11.2 como Hostname. Si su roboRIO está configurado a una IP estática usted puede usar esa IP como Hostname si se conecta a través de Ethernet o inalámbrico.

Iniciar Sesión



Cuando vea esta pantalla, ingrese el usuario deseado (vea descripción anterior) y presione enter. Para la pantalla de contraseña presione enter (no tienen contraseña ambas cuentas).

30.5 Caída de voltaje en roboRIO y Comprender el Consumo de Corriente

Para ayudar a mantener el voltaje de la batería y preservarse a sí misma y a otros componentes de control como el radio durante eventos de alto consumo de corriente, el roboRIO contiene un esquema de protección de caída de tensión. Este artículo describe dicho esquema, provee información acerca de una planeación proactiva de protecciones para el consumo de corriente actual del sistema, y describe como usar la nueva función de la PDP así como del DS LogFile Viewer para comprender los eventos de caída de tensión si es que ocurren en su robot.

30.5.1 Protección de Caída de Tensión del roboRIO

El roboRIO utiliza un esquema de protección contra de caída de tensión para mantener su voltaje de alimentación y de otros componentes del sistema de control con el fin de evitar reinicios de dispositivos en eventos de alto consumo de tensión que lleven el voltaje de la batería a un nivel peligrosamente bajo.

Etapa 1 – Salida de voltaje 6V

Disparo de voltaje – 6.8V

When the voltage drops below 6.8V, the 6V output on the *PWM* pins will start to drop.

Etapa 2 - Salida Deshabilitada

Disparo de voltaje – 6.3V

Cuando el voltaje cae por debajo de 6.3 V, el controlador iniciará el estado de protecciones para caídas de tensión. Los siguientes indicadores mostraran que dicha condición ha ocurrido:

- Led de encendido en el roboRIO encenderá amarillo.
- El fondo del monitor de voltaje en la Driver Station se tornará rojo.
- En la Driver Station el modo de visualización se volverá a caída de tensión.
- La etiqueta de CAN/Power del DS incrementará la falla de 12V en 1.
- La DS registrará una caída de tensión en el DS Log.

El controlador hará los siguientes pasos para tratar de preservar el voltaje de la batería:

- Las salidas PWM se desactivarán. Para las salidas PWM que han fijado su valor de neutro (todos los controladores de motor en WPILib) se enviará un único pulso de neutro antes de que se desactive la salida.
- Vías de 6V, 5V, 3.3V deshabilitados (Esto incluye las salidas de 6V en los pines de PWM, los pines de 5V de los conectores del grupo DIO, los pines de 5V de los conectores del grupo Analógicos, los pines de 3.3V en los grupos de SPI e I2C y los 5V y 3V pines en el grupo MXP)

- GPIO configurado como salidas go to High-Z
- Relevadores de salida son deshabilitados (driven low)
- Los controladores de motor basado en CAN se les envía un comando de deshabilitados.
- Pneumatic Devices such as the CTRE Pneumatics Control Module and REV Pneumatic Hub are disabled

El controlador se mantendrá en este estado hasta que el voltaje sea mayor a 7.5 o baje al siguiente disparo de voltaje para la siguiente etapa de las protecciones de caída de voltaje.

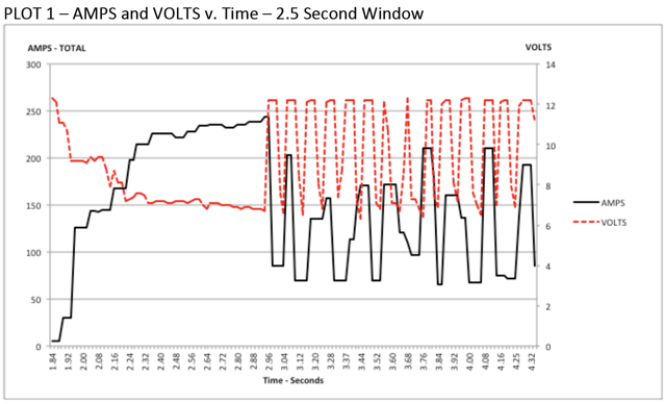
Etapa 3 – Apagón de dispositivo

Disparo de voltaje – 4.5V

Debajo de 4.5V el dispositivo puede apagarse. El voltaje exacto pude ser menor que esto y depende de la carga en el dispositivo.

El controlador se mantendrá en este estado hasta que el voltaje sea mayor a 4.65V, cuando esto suceda el dispositivo comenzará se secuencia normal de arranque.

30.5.2 Prevención de caída de tensión – Planeación proactiva para el consumo de corriente



La clave para prevenir una condición de caída de tensión es una planeación proactiva para el consumo de la corriente actual del robot. La mejor manera para hacer esto es crear un tipo de presupuesto de corriente. Esto puede ser un documento complejo que intente cuantificar ambas, el consumo de corriente estimado y tiempo de esfuerzo, para una mayor comprensión del uso de corriente y con esto el estado de la batería al finalizar el partido, o simplemente puede ser un inventario de la corriente usada. Para hacer esto:

1. Establezca el consumo máximo de corriente «sostenida» (entendiendo por sostenida lo que no es momentáneo). Esta es probablemente la parte más difícil de crear el presupuesto de energía. El consumo de corriente exacto que una batería puede sostener mientras mantiene un voltaje de 7+ voltios depende de una variedad de factores tales como la salud de la batería (ver *este artículo* para medir la salud de la batería) y el estado de carga. Como se muestra en la hoja de datos de la *NP18-12 <https://www.farnell.com/datasheets/575631.pdf>*, el gráfico de voltaje de los terminales se vuelve muy empinado a medida que el estado de carga disminuye, especialmente

cuando el consumo de corriente aumenta. Esta hoja de datos muestra que a una carga continua de 3CA (54A) una batería nueva puede funcionar de forma continua durante más de 6 minutos manteniendo una tensión en bornes de más de 7V. Como se muestra en la imagen anterior (utilizada con permiso del documento Team 234s Drive System Testing), incluso con una batería nueva, el consumo de 240A durante más de un segundo o dos es probable que cause un problema. Esto nos da algunos límites en el establecimiento de nuestro consumo de corriente sostenida. Para los propósitos de este ejercicio, vamos a establecer nuestro límite en 180A.

2. Enlistar las diferentes funciones de su robot tales como el chasis, algún manipulador, mecanismo principal del juego, 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 usted ha determinado funciones mutuamente exclusivas en su análisis, considere la posibilidad de forzar la exclusión en el programa. Usted también puede usar el monitor actual de la corriente del PDP (explicado más detalle a continuación) en su programa del robot para proveer límites de salida o exclusiones dinámicas (por ejemplo, no encender un motor de un mecanismo cuando la corriente del chasis este por encima de X o solamente permita a un motor trabajar a la mitad de su capacidad cuando la corriente el chasis este por encima de Y).

30.5.3 Settable Brownout

The NI roboRIO 1.0 does not support custom brownout voltages. It is fixed at 6.3V as mentioned in Stage 2 above.

The NI roboRIO 2.0 adds the option for a software settable brownout level. The default brownout level (Stage 2) of the roboRIO 2.0 is 6.75V.

JAVA

```
RobotController.setBrownoutVoltage(7.0);
```

C++

```
frc::RobotController::SetBrownoutVoltage(7_V);
```

30.5.4 Measuring Current Draw using the PDP/PDH

The FRC® Driver Station works in conjunction with the roboRIO and PDP/PDH to extract logged data from the PDP/PDH and log it on your DS PC. A viewer for this data is still under development.

In the meantime, teams can use their robot code and manual logging, a LabVIEW front panel or the SmartDashboard to visualize current draw on their robot as mechanisms are developed. In LabVIEW, you can read the current on a PDP/PDH channel using the Get PD Currents VI found on the Power pallet. For C++ and Java teams, use the PowerDistribution class as described in the *Power Distribution* article. Plotting this information over time (easiest with a LV Front Panel or with the SmartDashboard by using a Graph indicator can provide information to compare against and update your power budget or can locate mechanisms which do not seem to be performing as expected (due to incorrect load calculation, incorrect efficiency assumptions, or mechanism issues such as binding).

30.5.5 Identificar caídas de tensión

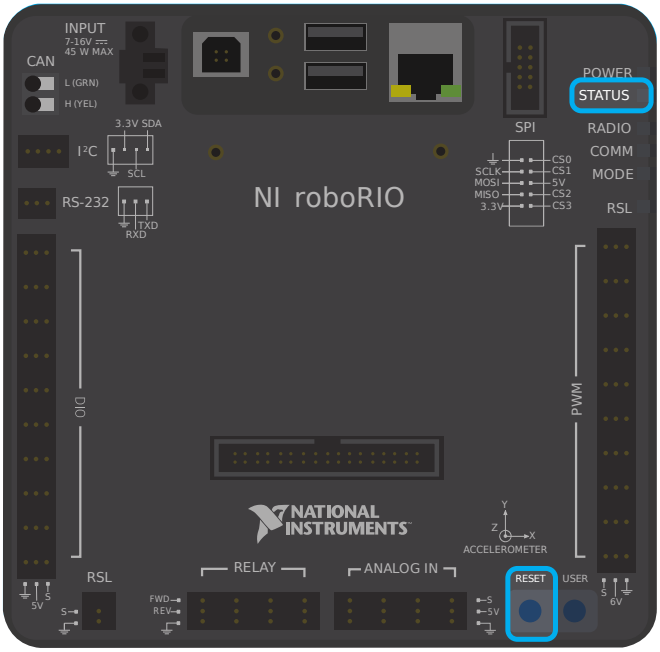
La manera más sencilla de identificar una caída de tensión es dando clic en la pestaña de CANPower de la DS y revisar el contador de la falla de 12V. Al mismo tiempo, puede revisar el registro de la Driver Station después de usar el Driver Station Log Viewer. Este registro identificará caídas de tensión con una línea naranja brillante, tal como de la imagen a continuación (nota, estas caídas de tensión fueron introducidas con un osciloscopio y no reflejan la duración ni el comportamiento típico de caídas de tensión en un robot de FRC).

30.6 Recuperando un roboRIO usando el MODO SEGURO

Ocasionalmente un roboRIO puede llegar a corromperse hasta el punto que no se pude recuperar usando un encendido normal y procesamiento de imagen. Prendiendo el roboRIO en Modo Seguro puede permitir que el dispositivo se inicie correctamente.

Importante: 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 Iniciando en Modo Seguro



Para encender el roboRIO en Modo Seguro:

1. Suministrar voltaje al roboRIO.
2. Dejar apretado el botón de Reset hasta cuando el LED de Status se encienda (~5 segundos) luego soltar el botón de Reset.
3. Entonces el roboRIO se iniciará en Modo Seguro (indicado por el LED de estatus parpadeando en grupos de 3)

30.6.2 Recuperando el roboRIO

The roboRIO can now be imaged by using the roboRIO Imaging Tool as described in *Imaging your roboRIO*.

30.6.3 Acerca del Modo Seguro

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 Avanzado

GradleRIO es el mecanismo que potencia el despliegue del código del robot en el roboRIO. GradleRIO se basa en el popular sistema de gestión de dependencias y construcción Gradle. Esta sección destaca las configuraciones **avanzadas** que los equipos pueden utilizar para mejorar su flujo de trabajo.

31.1 Uso de bibliotecas externas con el código del robot

Advertencia: El uso de bibliotecas externas puede tener un comportamiento no deseado con el código de su robot. No se recomienda a menos que sea consciente de lo que está haciendo.

A menudo un equipo puede querer añadir librerías externas Java o C++ para utilizarlas con su código de robot. Este artículo destaca la adición de bibliotecas Java a sus dependencias de Gradle, o las opciones que tienen los equipos de C++.

31.1.1 Java

Nota: Cualquier dependencia externa que dependa de bibliotecas nativas (JNI) probablemente no va a funcionar.

En Java es bastante sencillo añadir dependencias externas. Basta con añadir los `repositorios` y las `dependencias` necesarias.

Los proyectos Robot por defecto no tienen un bloque `repositorios {}` en el archivo ``build.gradle``. Tendrás que añadirlo tú mismo. Por encima del bloque `dependencies {}`, añade lo siguiente:

```
repositories {
   mavenCentral()
   ...
}
```

`mavenCentral()` puede ser sustituido por cualquier repositorio que la biblioteca que quieres importar esté utilizando. Ahora tienes que añadir la dependencia de la propia biblioteca. Esto se hace añadiendo la línea necesaria a tu bloque de ``dependencias {}`. El siguiente ejemplo muestra la adición de Apache Commons a tu proyecto Gradle.

```
dependencies {
   implementation 'org.apache.commons:commons-lang3:3.6'
   ...
}
```

Ahora ejecute una compilación y asegúrese de que las dependencias se descargan. Es posible que Intellisense no funcione correctamente hasta que se ejecute una compilación.

31.1.2 C++

Añadir dependencias de C++ a tu proyecto de robot no es trivial debido a la necesidad de compilar para el roboRIO. Tienes un par de opciones.

1. Copie el código fuente de la biblioteca deseada en su proyecto de robot.
2. Utilice la vendordep template como ejemplo y crear un vendordep.

Copiar el código fuente

Simplemente copie el código fuente y/o las cabeceras necesarias en su proyecto de robot. A continuación, puede configurar los argumentos de plataforma necesarios como se indica a continuación:

```
nativeUtils.platformConfigs.named("linuxx86-64").configure {
   it.linker.args.add('-lstdc++fs') // links in C++ filesystem library
}
```

Creación de un Vendordep

Siga las instrucciones del vendordep repository.

31.2 Configurando un CI para Código de Robot usando Acciones GitHub

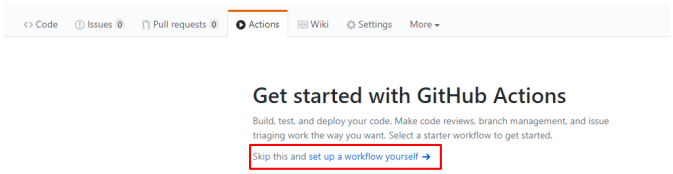
Un aspecto importante de trabajar en un equipo es ser capaz de probar el código que es enviado a una librería como GitHub. Por ejemplo, el director del proyecto o desarrollador principal podría ejecutar un grupo de unidades de prueba antes de fusionarla a una solicitud de extracción o podría querer asegurarse que todos los códigos en la rama dominante de una librería estén trabajando correctamente

Acciones GitHub es un servicio que permite a los equipos e individuos construyan y ejecuten pruebas de unidades en un código con varias ramas y solicitudes de extracción. Este tipo de servicios son comúnmente conocidos conocidos como servicios de “Continuous Integration”. Este tutorial le enseñará como preparar GitHub Actions en proyectos con códigos en robots.

Nota: Este tutorial asume que el código del robot de tu equipo está alojado en GitHub. Para una introducción a Git y GitHub, por favor vea esta *guía de introducción*.

31.2.1 Creando una acción

Las instrucciones para llevar a cabo el proceso de CI están almacenados en un archivo YAML. Para crearlo, dé click en "Actions" en la parte superior de su repositorio. Después dé click en hipervínculo de "set up a workflow yourself".



Ahora será recibido con un editor de texto. Reemplace todo el texto predeterminado con lo siguiente:

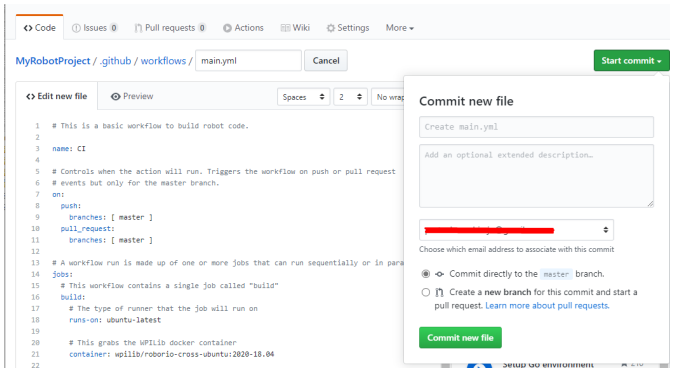
```
# This is a basic workflow to build robot code.

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.
    - 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
```

(continúe en la próxima página)

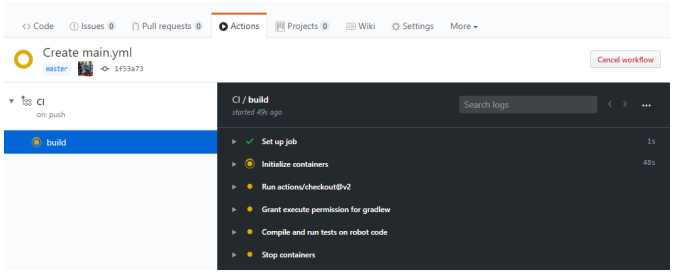
(proviene de la página anterior)

```
    # Runs a single command using the runners shell
    - name: Compile and run tests on robot code
      run: ./gradlew build
```

Después, guarde los cambios dando click en botón de “Start commit” en la esquina superior derecha de la pantalla. Puede modificar el mensaje predeterminado si lo desea. Después, de click en el botón verde de “Commit new file”.



El GitHub ahora ejecutará automáticamente una construcción cada vez que un commit sea empujado al principal o se abra una solicitud pull. Para monitorear el estado de cualquier construcción, puedes hacer clic en la pestaña «Acciones» en la parte superior de la pantalla.



31.2.2 Una descompostura del archivo Acciones YAML

Aquí hay un desglose del archivo YAML de arriba. Aunque no se requiere una comprensión estricta de cada línea, algún nivel de comprensión le ayudará a añadir más características y depurar posibles problemas que puedan surgir.

```
# 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 ]
```

Este bloque de código dicta cuándo se ejecutará la Acción. Actualmente, la acción se ejecutará cuando las confirmaciones se empujen al main o cuando se abran peticiones pull contra la del 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
```

Cada flujo de trabajo de Action es hecho por uno o más trabajos ya sea que serán ejecutados (uno después de otro) o en paralelo (al mismo tiempo). En nuestro flujo de trabajo, solo hay un trabajo de "construcción".

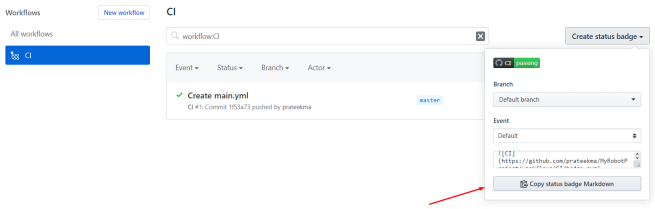
Especificamos que queremos que el trabajo ejecute una máquina virtual de Ubuntu y en un contenedor Docker que contiene JDK, compilador C++ y roboRIO toolchains.

```
# 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
```

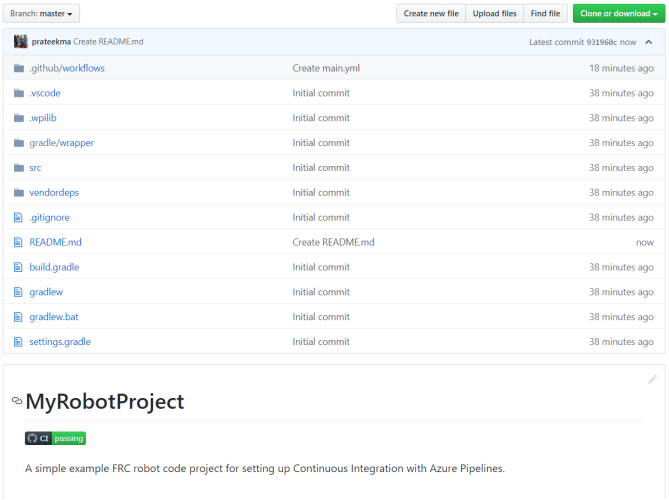
Each job has certain steps that will be executed. This job has four steps. The first step involves checking out the repository to access the robot code. The second step is a workaround for a GitHub Actions issue. The third step involves giving the virtual machine permission to execute gradle tasks using `./gradlew`. The final step runs `./gradlew build` to compile robot code and run any unit tests.

31.2.3 Agregando un Distintivo de estatus de construcción a un archivo README.md

Es útil agregar una etiqueta CI de estatus en la parte superior del archivo README de su repositorio para revisar rápidamente el estado de la última construcción en la rama principal. Para hacer esto, dé clic en "Actions" en la parte superior de la pantalla y seleccione "CI" en el lado izquierdo de la pantalla. Después, dé clic en el botón de "Create status badge" en la parte superior derecha y copie el código de Markdown de la etiqueta de estado.



Finalmente, pegue el código de Markdown que copió en la parte de arriba de su archivo README, asigne, y presione sus cambios. Ahora, debería de ver la etiqueta de estado de GitHub Actions en la página principal de su repositorio.



31.3 Uso de un formateador de código

Los formateadores de código existen para asegurar que el estilo del código escrito sea consistente en toda la base de código. Esto se utiliza en muchos proyectos importantes, desde Android hasta OpenCV. Los equipos pueden querer añadir un formateador en todo el código de su robot para asegurarse de que el código base mantiene la legibilidad y la coherencia en todo momento.

Para este artículo, destacaremos el uso de Spotless para equipos Java y wpiformat para equipos C++.

31.3.1 Spotless

Configuración

Es necesario realizar los cambios necesarios en `build.gradle` para que Spotless sea funcional. En el bloque `plugins {}` de tu `build.gradle`, añade el plugin Spotless para que aparezca de forma similar a la siguiente.

```
plugins {
   id "java"
   id "edu.wpi.first.GradleRIO" version "2024.1.1"
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   id 'com.diffplug.spotless' version '6.20.0'
}
```

A continuación, asegúrese de añadir un bloque requerido `spotless {}` para configurar correctamente spotless. Esto puede colocarse al final de tu `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()
   }
}
```

Ejecutar Spotless

Spotless puede ser ejecutado usando `./gradlew spotlessApply` que aplicará todas las opciones de formato. También se puede especificar una tarea concreta añadiendo simplemente el nombre del formateador. Un ejemplo es `./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:

```
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
```

Explicación de las opciones

Cada sección `format` destaca el formato de los archivos personalizados en el proyecto. Las secciones `java` y `groovyGradle` son soportadas nativamente por spotless, por lo que se definen de forma diferente.

Desglosando esto, podemos dividirlo en varias partes.

- Formateo de Java
- Formateo de archivos Gradle
- Formateo de archivos XML
- Formateo de archivos varios

Todos son similares, excepto por algunas pequeñas diferencias que se explicarán. El siguiente ejemplo destacará el bloque `java {}`.

```
java {
   target fileTree('.') {
      include '**/*.java'
      exclude '**/build/**', '**/build-*/**'
   }
   toggleOffOn()
   googleJavaFormat()
   removeUnusedImports()
   trimTrailingWhitespace()
```

(continúe en la próxima página)

(proviene de la página anterior)

```
   endWithNewline()
}
```

Vamos a explicar qué significa cada una de las opciones.

```
target fileTree('.') {
   include '**/*.java'
   exclude '**/build/**', '**/build-*/**'
}
```

El ejemplo anterior le dice a spotless dónde están nuestras clases Java y que excluya el directorio `build`. El resto de las opciones se explican por sí mismas.

- `toggleOffOn()` añade la posibilidad de que spotless ignore partes específicas de un proyecto. El uso es como el siguiente

```
// format:off

public void myWeirdFunction() {

}

// format:on
```

- `googleJavaFormat()` le dice a spotless que formatee según la Guía de Estilo de Google
- `removeUnusedImports()` will remove any unused imports from any of your Java classes
- `trimTrailingWhitespace()` eliminará cualquier espacio en blanco extra al final de las líneas
- `endWithNewline()` añadirá un carácter de nueva línea al final de tus clases

En el bloque `groovyGradle`, hay una opción `greclipse`. Este es el formateador que spotless utiliza para formatear los archivos gradle.

Además, hay una opción `eclipseWtp` en el bloque `xml`. Esto significa «Gradle Web Tools Platform» y es el formateador para formatear archivos `xml`. Los equipos que no utilicen ningún archivo XML pueden desear no incluir esta configuración.

Nota: Una lista completa de configuraciones está disponible en el Spotless README

Problemas con los finales de línea

Spotless intentará aplicar finales de línea por SO, lo que significa que los diffs de Git cambiarán constantemente si dos usuarios están en diferentes SOs (Unix vs Windows). Se recomienda que los equipos que contribuyen al mismo repositorio desde múltiples sistemas operativos utilicen un archivo `.gitattributes`. Lo siguiente debería ser suficiente para manejar los finales de línea.

```
*.gradle text eol=lf
*.java text eol=lf
*.md text eol=lf
*.xml text eol=lf
```

31.3.2 wpiformat

Requerimientos:

- Python 3.6 o superior

Puede instalar wpiformat escribiendo `pip3 install wpiformat` en un terminal o en la línea de comandos.

Uso

wpiformat se puede ejecutar escribiendo `wpiformat` en una consola. Esto formateará con `clang-format`. Se necesitan tres archivos de configuración (`.clang-format`, `.styleguide`, `.styleguide-license`). Estos deben existir en la raíz del proyecto.

- `.clang-format`: `Download`
- `.styleguide-license`: `Download`

A continuación se muestra un ejemplo de guía de estilo:

```
cppHeaderFileInclude {
   \.h$
   \.hpp$
   \.inc$
   \.inl$
}

cppSrcFileInclude {
   \.cpp$
}

modifiableFileExclude {
   gradle/
}
```

Nota: Los equipos pueden adaptar `.styleguide` y `.styleguide-license` como quieran. ¡Es importante que no se eliminen, ya que son necesarios para ejecutar 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:

```
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
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      - 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
      - name: Run
        run: wpiformat
      - name: Check output
        run: git --no-pager diff --exit-code HEAD
```

31.4 Tareas de Gradlew

Este artículo tiene como objetivo resaltar los comandos de gradle compatibles con el equipo de WPILib para el uso de los usuarios. Estos comandos se pueden ver escribiendo `./gradlew tasks` en la raíz de su proyecto de robot. No todos los comandos que se muestran en ```./gradlew tasks``y los comandos no compatibles no se documentarán aquí.

31.4.1 Build tasks

`./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 CompileCommands tasks

`./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` - Deploy all artifacts on all targets. This will deploy your robot project to the available targets (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 GradleRIO tasks

`./gradlew $TOOL$` - Runs the tool `$TOOL$` (Replace `$TOOL$` with the name of the tool. 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` - Launches simulation for the Java projects

`./gradlew simulateNative` - Launches simulation for C++ projects

`./gradlew vendordep` - Install vendordep JSON file from URL or local installation. See *Bibliotecas de 3ros*

31.5 Including Git Data in 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.

Nota: 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"
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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:

```
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.

Ignoring Generated Files with 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 Using Compiler Arguments

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.

Advertencia: Modifying arguments is dangerous and can cause unexpected behavior.

31.6.1 C++

Platforms

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

Configuring for a Platform

```
nativeUtils.platformConfigs.named('windowsx86-64').configure {
  it.cppCompiler.args.add("/utf-8")
}
```

native-utils is used to configure the platform, in this case, *windowsx86-64*. This can be replaced for any platform listed in the previous section. Then arguments, such as */utf-8* is appended to the C++ compiler.

31.6.2 Java

Arguments can also be configured for Java. This can be accomplished by editing *build.gradle* and appending arguments to the *FRCJavaArtifact*. An example of this is shown below.

```
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.TE.AM.2") // Replace TE.AM 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. (*TE.AM IP Notation*)

Importante: The hostname when connected via USB-B should be `172.22.11.2`.

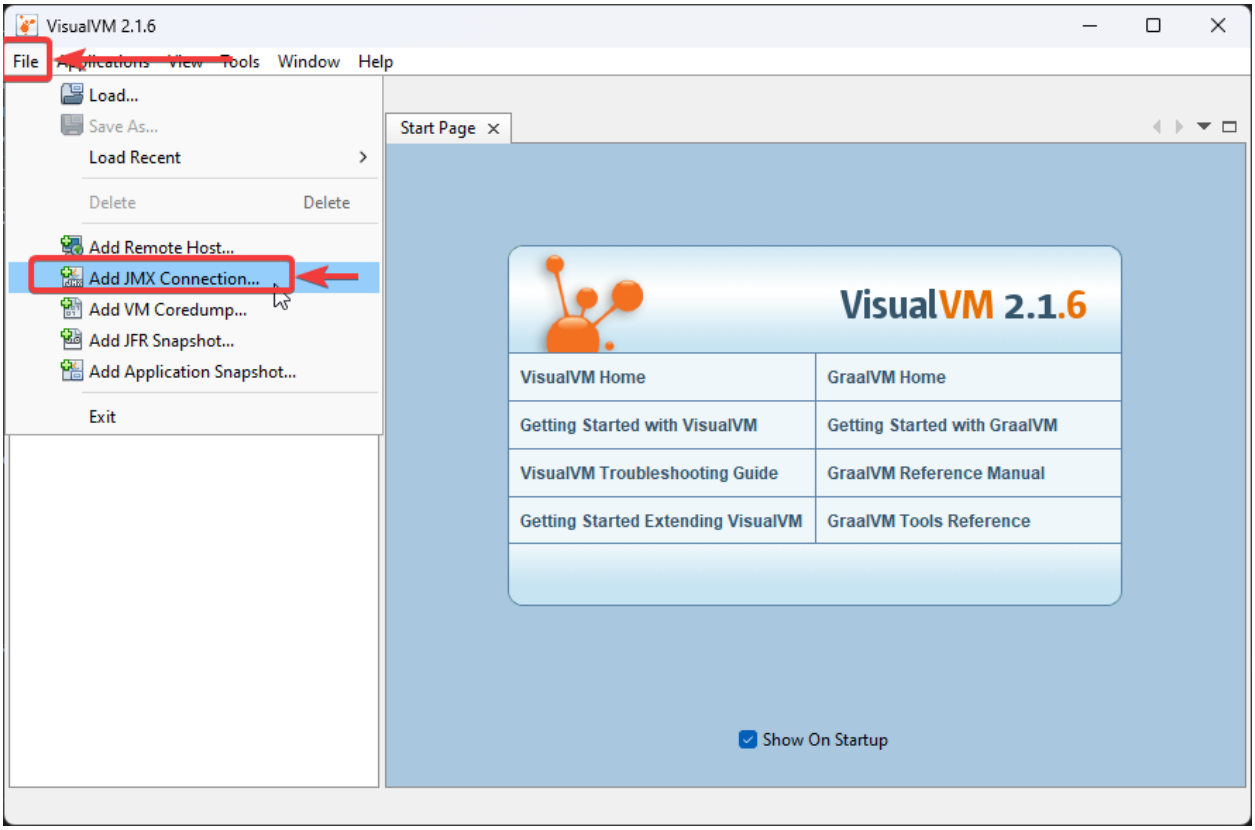
31.7.3 Running VisualVM

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:

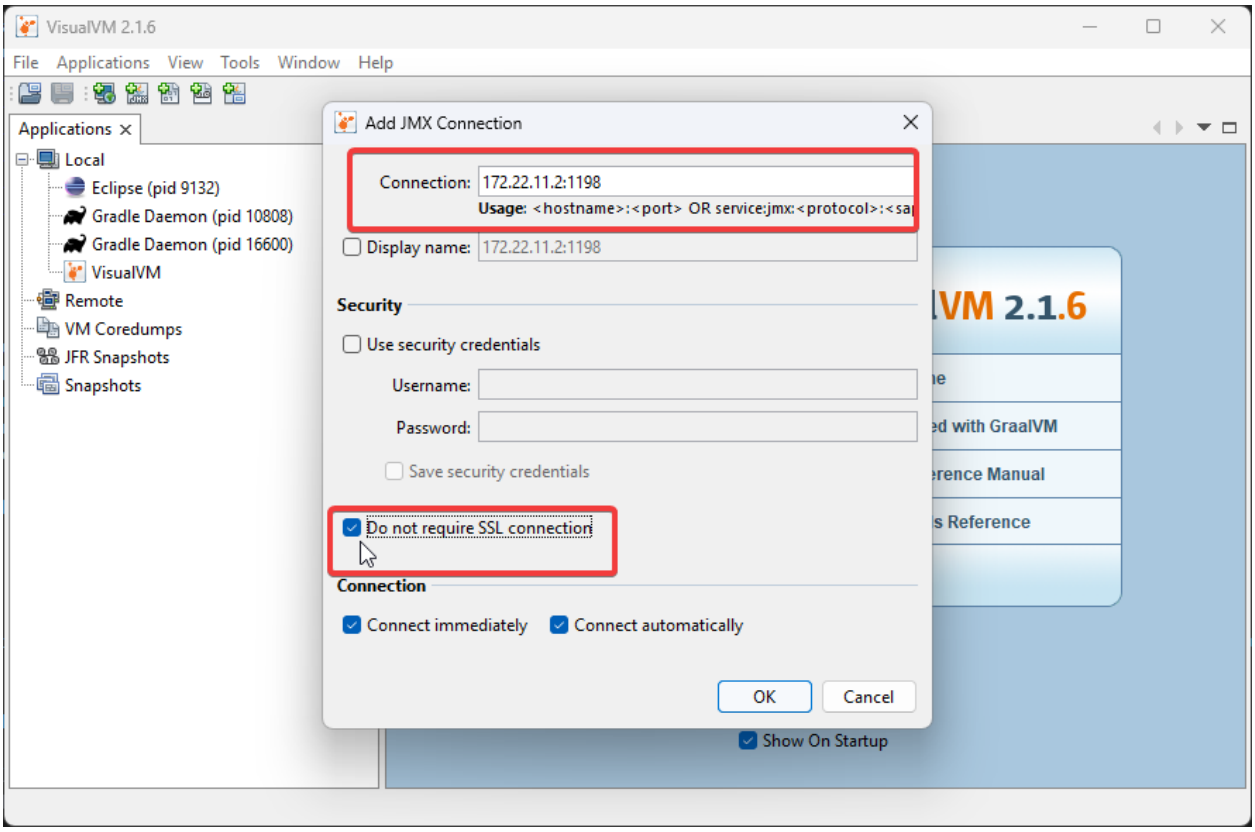
```
cd "C:\Users\Public\wpilib\visualvm_217\bin"
./visualvm --jdkhome "C:\Users\Public\wpilib\2024\jdk"
```

Importante: The exact path `visualvm_217` may vary and depends on the version of VisualVM downloaded.

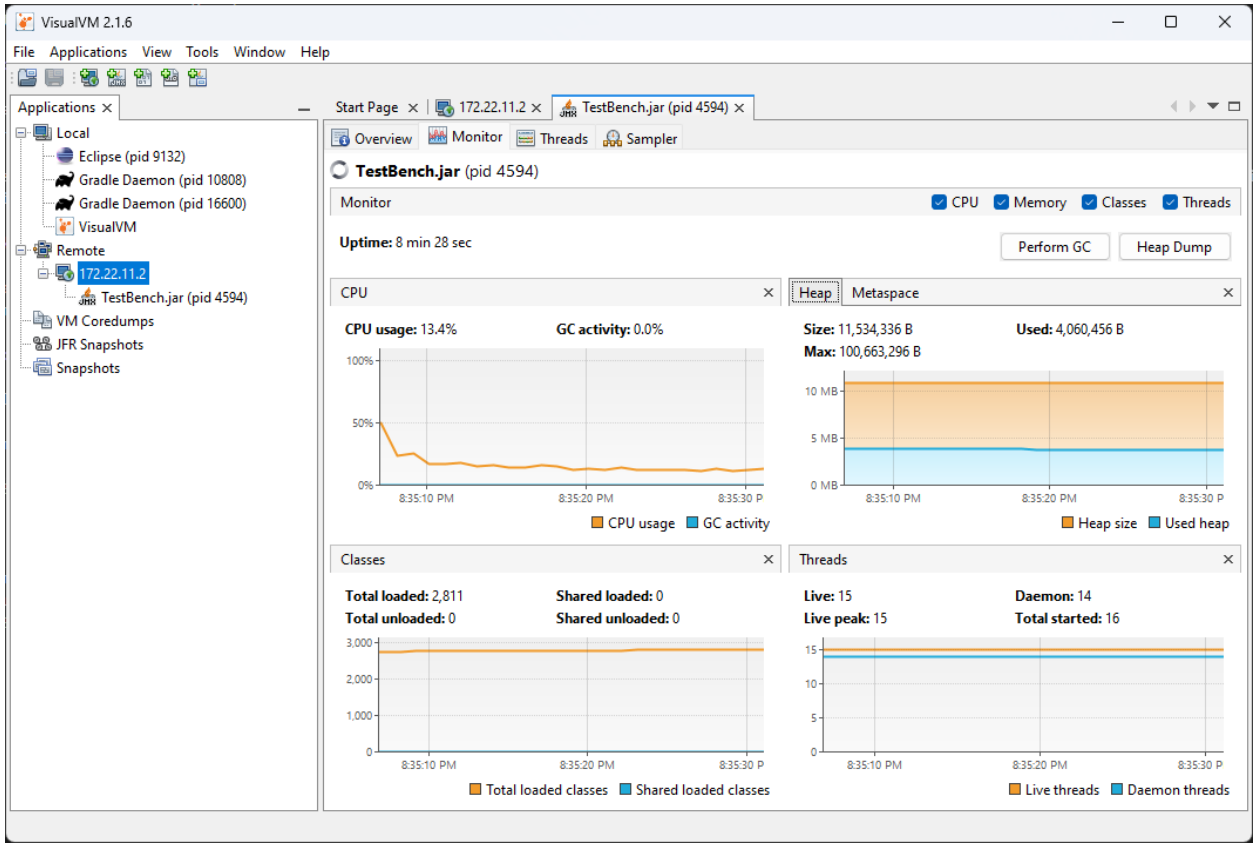
This should launch VisualVM. Once launched, open the *Add JMX Connection* dialog.



Once opened, configure the connection details and hostname. Ensure that *Do not require SSL connection* is ticked.

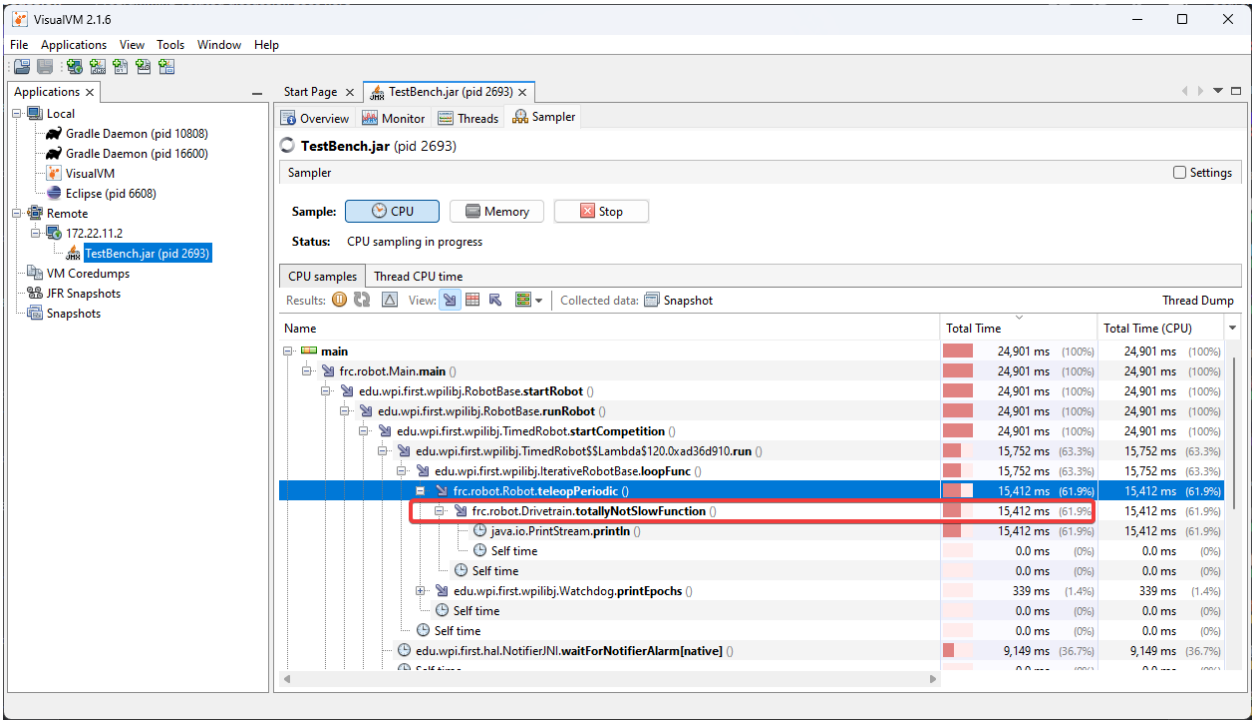


If correctly done, a new menu option in the left-hand sidebar will appear. Clicking on it will show you a detailed dashboard of the running JVM application.



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.

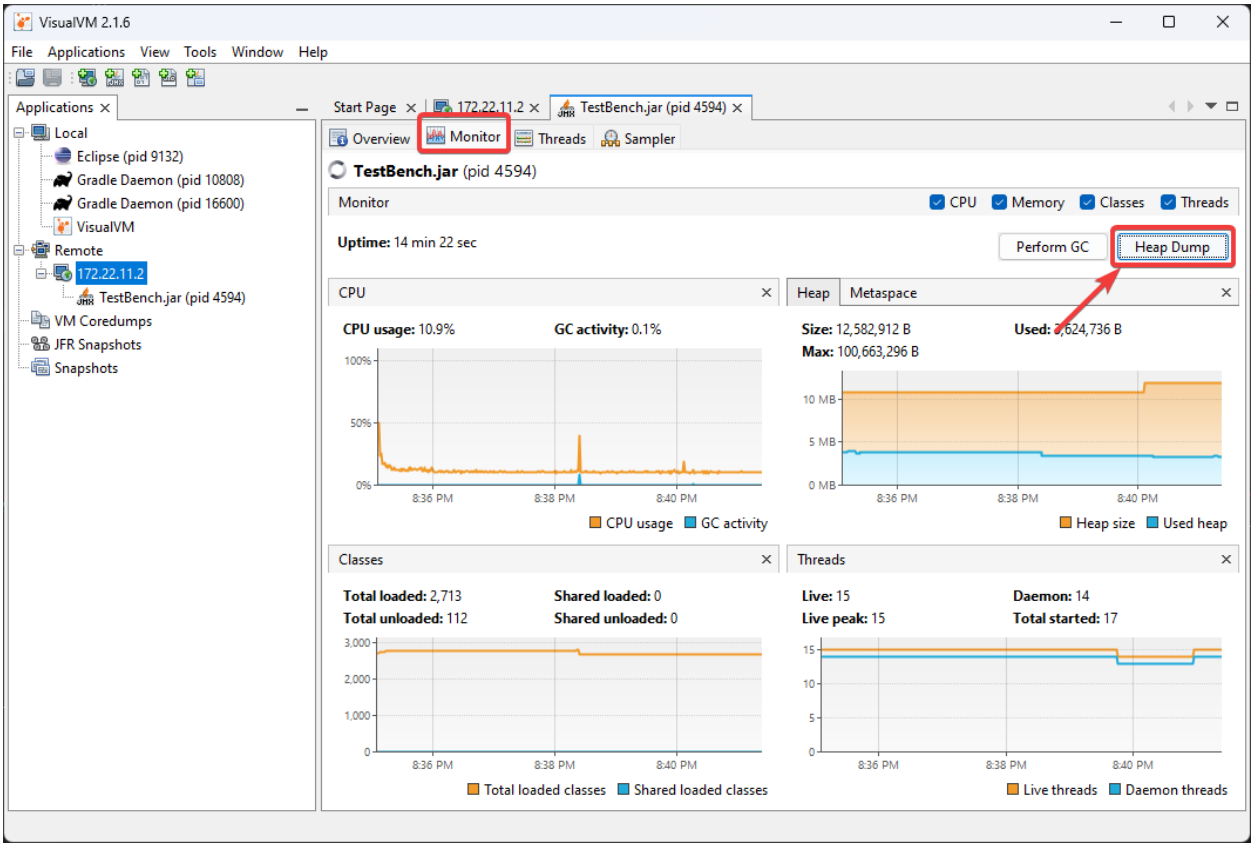
```
public static void totallyNotSlowFunction() {
   for (int i = 0; i < 2000; i++) {
      System.out.println("HAHAHAHA");
   }
}
```

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*.



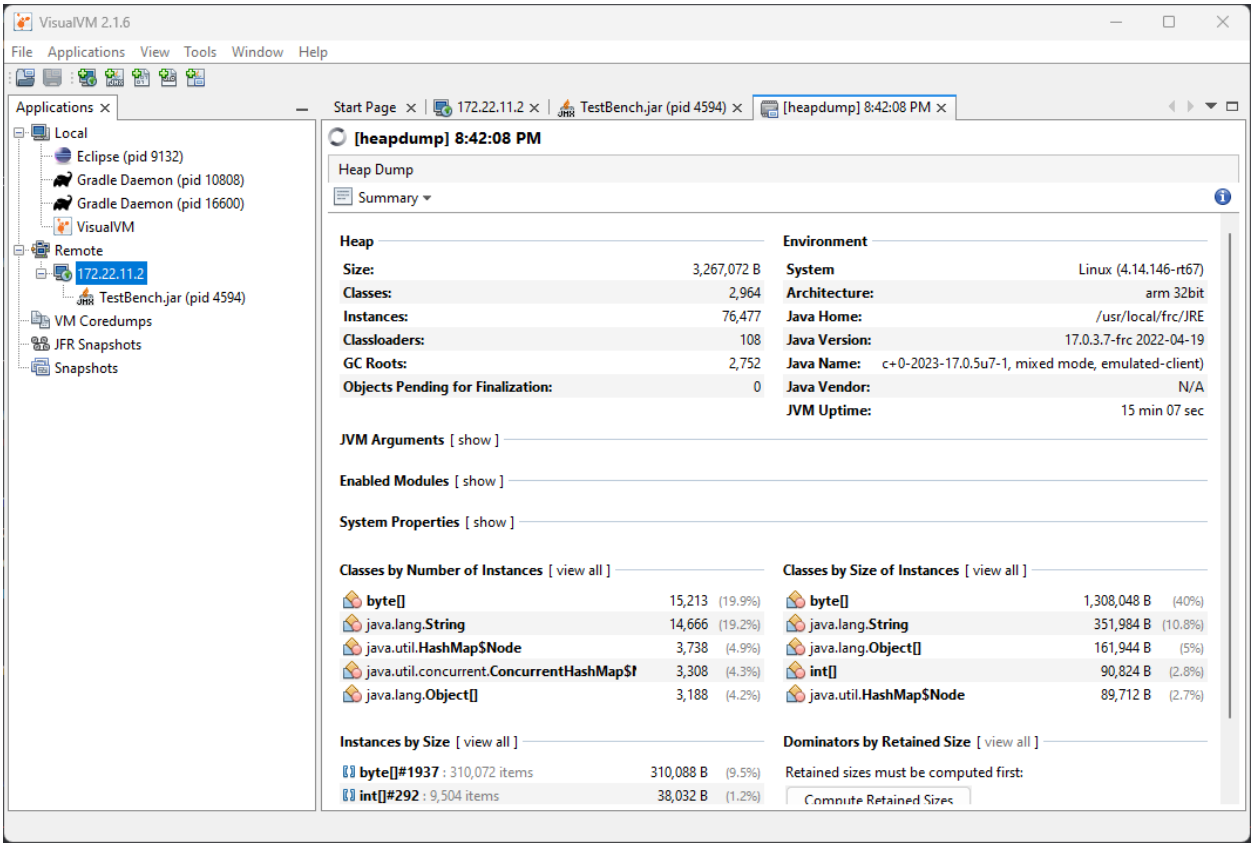
This heap dump will be stored on the target system (roboRIO) and must be retrieved using SFTP. See *this article* for information on retrieving the dump from the roboRIO.

Once downloaded, the dump can be analyzed with VisualVM.

Truco: You can also *configure the JVM to take a heap dump automatically when your robot code runs out of memory*.

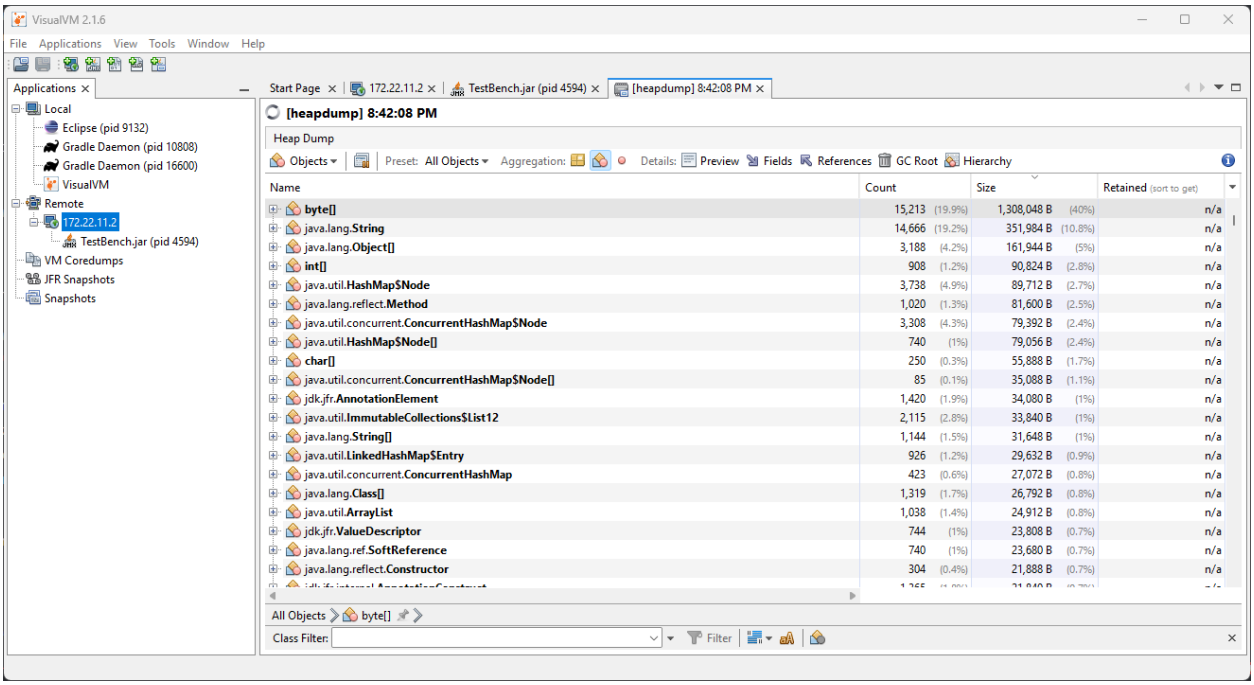
31.7.6 Analyzing a Heap Dump

Reopen VisualVM if closed using the previous instructions. Then click on *File* and *Load*. Navigate to the retrieved dump file and load it.

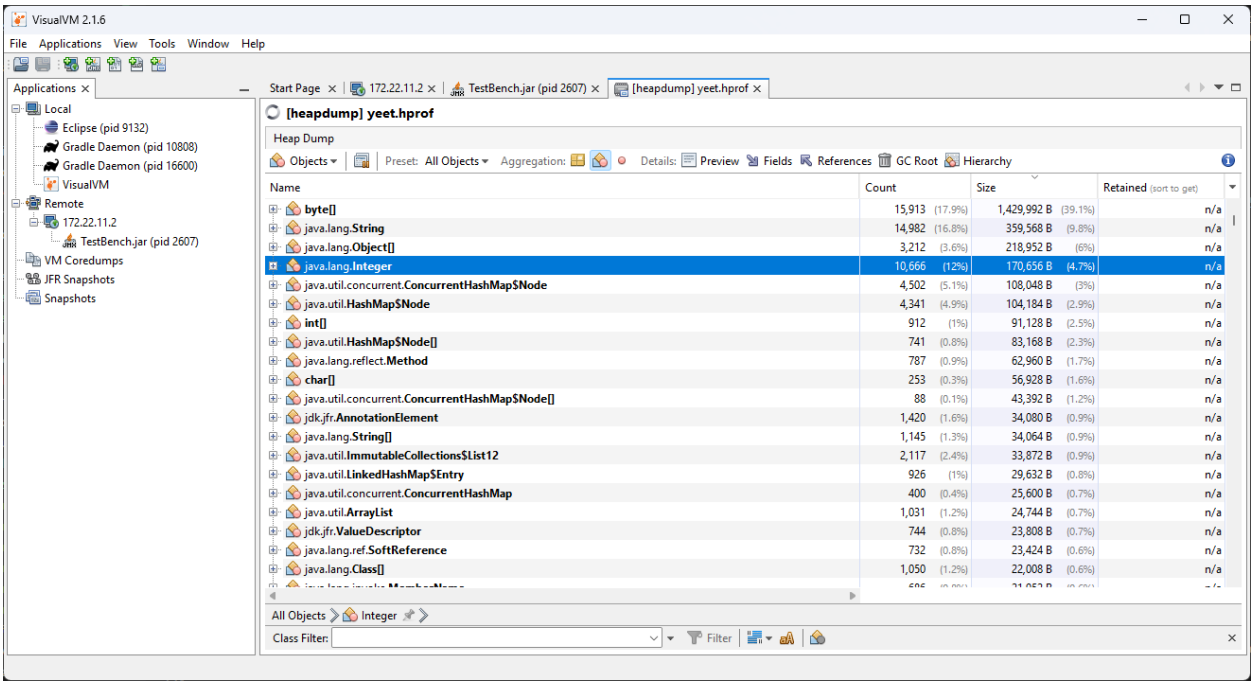


Clicking on *Summary* and selecting *Objects* instead will show a breakdown of objects by quantity. The below screenshot showcases a completely empty robot program, and then one that creates an million large `ArrayList` of integers.

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 Controles avanzados

Esta sección cubre funciones de control avanzadas en WPILib, como varios algoritmos de control de retroalimentación/avance y seguimiento de trayectoria.

32.1 Un Video tutorial de Modelo Basado en la Validación de Autónomos en FRC

En la «RSN Spring Conference, Presented by WPI» en 2020, Tyler Veness miembro del equipo de WPILib dio una presentación sobre Validación basada en Modelos de Autónomos en FRC®.

The link to the presentation is available here.

32.2 Introducción a los controles avanzados

32.2.1 Sistema Básico de Control

Nota: This article includes sections of Controls Engineering in FRC by Tyler Veness with permission.

The Need for Control Systems

Sistemas de control están alrededor de nosotros e interactuamos diariamente con ellos. Una pequeña lista de algunos que puede que haya visto incluyen calentadores y aires acondicionados con termostatos, control crucero y sistema antibloqueo de ruedas (Frenos ABS) en automóviles, la modulación de la velocidad de un ventilador en las laptops modernas. Monitores de sistemas de control o controlar el comportamiento de sistemas como estos puede consistir en humanos controlándolos directamente (control manual) o solo en máquinas (control automático).

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.

¿Como podemos comprobar un lazo cerrado de control en un carro autónomo? Por ejemplo, ¿se comportará de manera segura y cumplirá las especificaciones de rendimiento deseadas en presencia de alguna incertidumbre? La teoría de control es una aplicación del algebra y geometría usada para analizar y predecir el comportamiento de sistemas, haciendo que respondan como nosotros queremos haciéndolos robustos contra perturbaciones e incertidumbres.

Controles de ingeniería es, en pocas palabras, el proceso de ingeniería aplicado a la teoría del control. Como tal, es más que solo aplicar matemáticas. Mientras la teoría de control tiene una matemática hermosa detrás de ella, controles de ingeniería es una disciplina de la ingeniería así como cualquier otra está llena de compensaciones. Las soluciones de la teoría de control siempre deben ser verificadas e informadas por las especificaciones de rendimiento. No tiene que ser perfecto; solamente ser suficientemente bueno para alcanzar nuestras especificaciones.

Nomenclatura

La mayoría de los recursos para temas de ingeniería avanzada asumen un nivel de conocimientos muy superior al necesario. Parte del problema es el uso de términos. Mientras comunica eficientemente ideas a aquellos que están relacionados al tema, gente nueva no familiarizada al tema se pierde.

El sistema o colección de actuadores siendo controlados por un sistema es llamado *Planta*. Un controlador es usado para manejar la planta de su estado actual a un estado deseado (referencia). Controladores que no incluyen información medida a la salida de la planta son llamados controladores de lazo abierto.

Controladores que incorporan una información entregada de la salida de la planta son llamados controladores de lazo cerrado o controladores de retroalimentación.

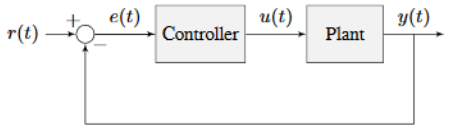


Figure 1.1: Control system nomenclature for a closed-loop system

$r(t)$	reference	$u(t)$	control input
$e(t)$	error	$y(t)$	output

Nota: La entrada y salida de un sistema están definidos desde el punto de visto de la planta. La retroalimentación negativa del controlador mostrada es dada por la diferencia entre la

referencia y la salida, también conocido como error, a cero.

¿Qué es la ganancia?

La *ganancia* es un valor proporcional que muestra la relación entre la magnitud de una señal de entrada con la magnitud de una señal de salida a un estado estable. Muchos sistemas contienen un método con el cual la ganancia puede ser alterada, dando al sistema mayor o menos "poder".

La siguiente figura muestra un sistema con una entrada y salida hipotética. Debido a que la salida es dos veces la amplitud de la entrada, es sistema tiene una ganancia de dos.

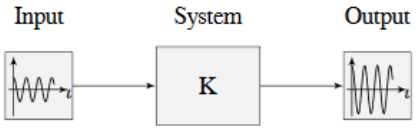


Figure 1.2: Demonstration of system with a gain of $K = 2$

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.

Diagramas de Bloque

Cuando se diseñe o analice un sistema de control, es útil graficar el modelo. Los diagramas de bloques son usados para dicho propósito. Estos pueden modificarse sistemáticamente o simplificarse.

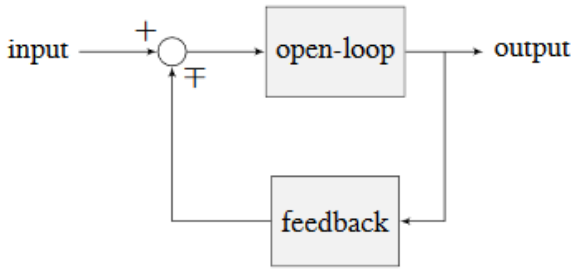


Figure 1.3: Block diagram with nomenclature

La ganancia de un circuito de lazo abierto es la ganancia total de la suma del nodo en la entrada (el circulo) a la línea de salida. Esto será la ganancia del sistema si el bucle de retroalimentación fuera desconectado. La retroalimentación de ganancia es la ganancia total del regreso de la salida a la suma del nodo de entrada. La salida de un nodo de suma es la suma de sus entradas.

La siguiente figura es un diagrama de bloques con notación más formal en una configuración de retroalimentación.

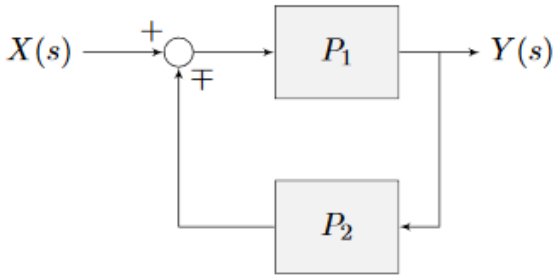


Figure 1.4: Feedback block diagram

$\mp$ significa “menos o más” donde menos presenta una retroalimentación negativa.

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

Nota: 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:

Nota: 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

Nota: For a guide on implementing PID control in code with WPILib, see *Control Feedforward 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 Introducción a PID

Nota: For a guide on implementing PID control with WPILib, see *Control PID en 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*
$e(t)$	*error*	$y(t)$	*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.

Nota: 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.

Término Proporcional

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*.

$$u(t) = K_p e(t)$$

donde K_p es la ganancia proporcional y $e(t)$ es el error en tiempo actual t.

La imagen a continuación muestra un diagrama de bloques para un *system* controlado por un controlador P.

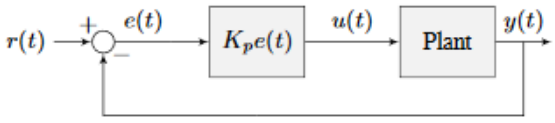


Figure 2.1: P controller block diagram

La ganancia proporcional actúa como "un resorte definido por programa" que jala el *system* a la posición deseada. Recordando de la física donde modelamos resortes como $F = -kx$ donde F es la fuerza aplicada, k es la constante proporcional y x es el desplazamiento del punto de equilibrio. Esto puede ser escrito de otra manera $F = k(0 - x)$ donde 0 es el punto de equilibrio. Si permitimos que el punto de equilibrio sea nuestro *punto de referencia* de la retroalimentación del controlador, la ecuación correspondería uno a uno.

$$F = k(r - x)$$
$$u(t) = K_p e(t) = K_p(r(t) - y(t))$$

entonces la "fuerza" con la cual el controlador proporcional jala la salida del *system's* hacia la *referencia* es proporcional al *error*, así como un resorte.

Término Derivativo

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*.

$$u(t) = K_p e(t) + K_d \frac{de}{dt}$$

Donde K_p es la ganancia proporcional, K_d es la ganancia derivativa y $e(t)$ es el error en el tiempo actual t.

La figura a continuación muestra un diagrama de bloques para un sistema controlado por un controlador PD.

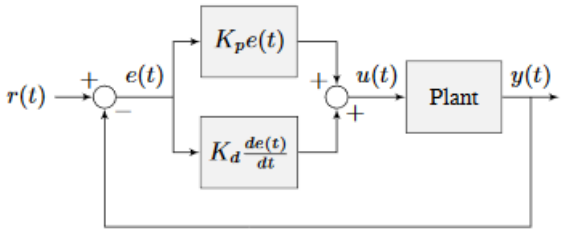


Figure 2.2: PD controller block diagram

Un controlador PD tiene un controlador proporcional para la posición (K_p) y un controlador proporcional para la velocidad (K_d). El *punto de referencia* de la velocidad es dada implícitamente conforme al cambio del *punto de referencia* de la posición conforme al tiempo. Para comprobar esto, reacomodaremos la ecuación para un controlador 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.

$$\begin{aligned}
u_k &= K_p(r_k - x_k) + K_d \frac{(r_k - x_k) - (r_{k-1} - x_{k-1})}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{r_k - x_k - r_{k-1} + x_{k-1}}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{r_k - r_{k-1} - x_k + x_{k-1}}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{(r_k - r_{k-1}) - (x_k - x_{k-1})}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \left(\frac{r_k - r_{k-1}}{dt} - \frac{x_k - x_{k-1}}{dt} \right)
\end{aligned}$$

Vea como $\frac{r_k - r_{k-1}}{dt}$ es la velocidad del *punto de referencia*. Por la misma razón, $\frac{x_k - x_{k-1}}{dt}$ es la velocidad del *sistema* en un tiempo unitario. Esto significa que el término K_d del controlador PD lleva la velocidad estimada al *punto de referencia* de la velocidad.

Si el *punto de referencia* es una constante, el *punto de referencia* implícito de la velocidad es cero, entonces el término K_d ralentiza el *sistema* si no está variando. Esto actúa como un «amortiguador definido por programa». Estos son encontrados comúnmente en cierrapuertas y su velocidad de amortiguamiento aumenta linealmente con la velocidad.

Término Integral

Importante: 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.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau$$

Donde K_p es la ganancia proporcional, K_i es la ganancia integral, $e(t)$ es el error en el tiempo actual t y τ es la integración de la variable.

La integral integra desde el tiempo 0 hasta el tiempo t. Usamos τ para la integración debido a que requerimos una variable que tome múltiples valores a través de la integral, pero no se podemos usar debido a que ya la definimos como tiempo actual.

La imagen a continuación muestra un diagrama de bloque para un *sistema* controlado por un controlador PI.

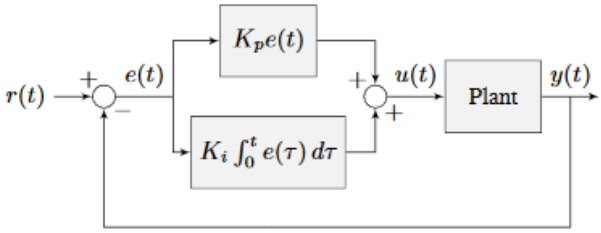


Figure 2.3: PI controller block diagram

Cuando el *sistema* está cerca del *punto de referencia* en estado estable, el término proporcional puede ser muy pequeño para llevar la salida hasta el *punto de referencia*, y el término derivativo es cero. Esto pude resultar en un *error de estado estable* como se muestra en la figura 2.4.

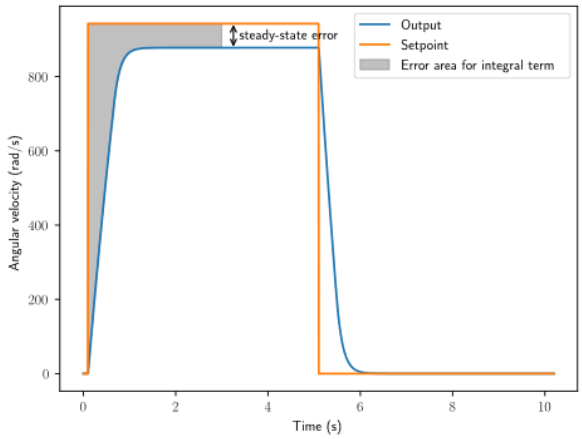


Figure 2.4: P controller with steady-state error

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.

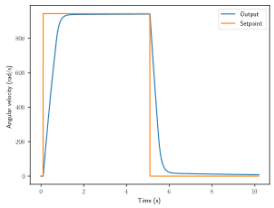


Figure 2.5: PI controller without steady-state error

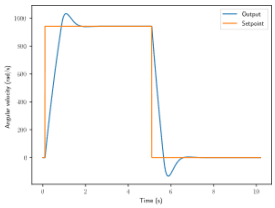


Figure 2.6: PI controller with overshoot from large K_i gain

Putting It All Together

Nota: Para obtener información sobre el uso de WPILib proporcionada por PIDController, ver el *artículo relevante*.

When these terms are combined by summing them all together, one gets the typical definition for a PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de}{dt}$$

Donde K_p es la ganancia proporcional, K_i es la ganancia integral, K_d es la ganancia derivativa, $e(t)$ es el error en el tiempo actual t y τ es la variable de integración.

A continuación, se muestra un diagrama de bloques para un controlador PID.

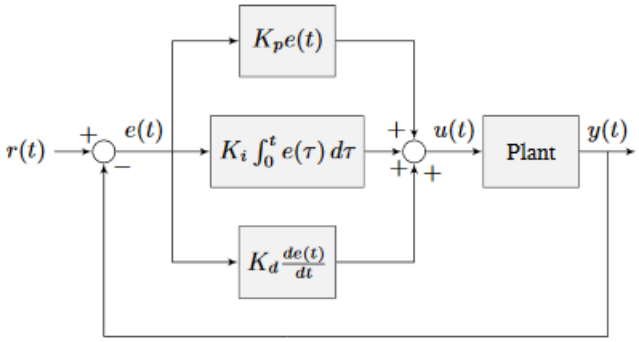


Figure 2.7: PID controller block diagram

Tipos de Respuesta

Un *sistema* manejado por un controlador PID generalmente tiene tres tipos de respuesta: subamortiguada, sobreamortiguada y críticamente amortiguada. Son mostradas en la figura 2.8.

En la figura 2.7 para las *respuestas unitarias*, el *tiempo de subida* es el tiempo que le toma al *sistema* alcanzar inicialmente la referencia después de haber aplicado la *entrada unitaria*. El *tiempo de estabilización* es el tiempo que le lleva al *sistema* estabilizarse en la *referencia* después de haber aplicado la *entrada unitaria*.

Una respuesta *subamortiguada* oscila alrededor de la *referencia* antes de estabilizarse. Una respuesta *sobreamortiguada*

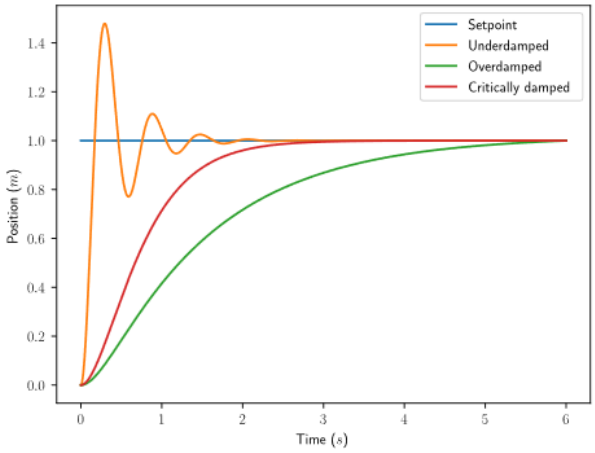


Figure 2.8: PID controller response types

tarda en elevarse y no excede el valor de la *referencia*. Una respuesta *críticamente amortiguada* tiene el *tiempo de subida* más rápido sin exceder la *referencia*.

32.2.5 Video de Introducción PID por WPI

¿Alguna vez ha tenido problemas diseñando un sistema del robot que se mueva rápido y luego se detenga a una exacta posición deseada? Retos como este pueden aparecer al conducir distancias o velocidades fijas, utilizando un brazo o elevador, o cualquier otro sistema de motor controlado que requiera un específico movimiento. En este video, el Profesor Dmitry Berenson de WPI habla acerca controles de robot y cómo funciona el 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

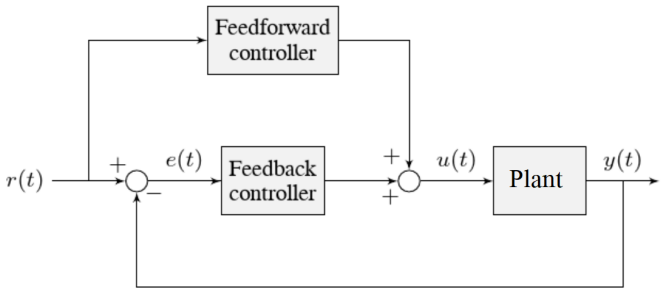
In this section, we will tune a simple velocity controller for a flywheel. The tuning principles explained here will also work for almost any velocity control scenario.

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 *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

Nota: 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.

Nota: 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.

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.

[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.

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*.

Nota: 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.

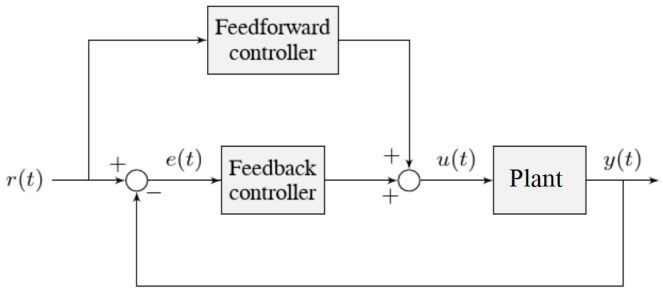
Turret Model Description

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 *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.

Nota: 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.

Nota: 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 *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 Ajuste de un controlador de posición de brazo vertical

En esta sección, ajustaremos un controlador de posición simple para un brazo vertical. Los mismos principios de ajuste que se explican a continuación también funcionarán para casi todos los escenarios de control de posición bajo la carga de la gravedad.

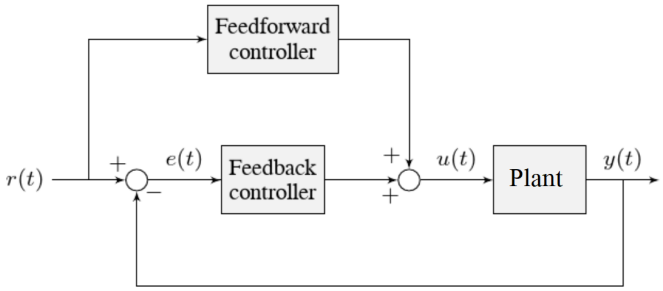
Descripción del modelo de brazo

Los brazos verticales se usan comúnmente para levantar piezas de juego desde el suelo hasta una posición de puntuación. Otros ejemplos similares incluyen campanas de tiro y elevadores.

Nuestro «brazo vertical» consiste en:

- Una masa sobre un palo, bajo la fuerza de la gravedad, que gira alrededor de un eje.
- Un motor y una caja de cambios que impulsan el eje al que está unida la masa en un palo

Para los fines de este tutorial, esta planta se modela con la misma ecuación utilizada por WPILib's *ArmFeedforward* , con un ajuste adicional para la demora del sensor y la ineficiencia de la caja de cambios. La simulación asume que la planta está controlada por controladores feedforward y feedback, compuestos de esta manera:



Donde:

- 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.

Nota: 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.

Nota: 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 *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.

Nota: 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 Common Control Loop Tuning Issues

There are a number of common issues which can arise while tuning feedforward and feedback controllers.

Integral Term Windup

Beware that if K_i is too large, integral windup can occur. Following a large change in *setpoint*, the integral term can accumulate an error larger than the maximal *control effort*. As a result, the system overshoots and continues to increase until this accumulated error is unwound.

There are a few ways to mitigate this:

1. Decrease the value of K_i, down to zero if possible.
2. Add logic to reset the integrator term to zero if the *output* is too far from the *setpoint*. Some smart motor controllers and WPILib's `PIDController` implement this with a `setIZone()` method.
3. Cap the integrator at some maximum value. WPILib's `PIDController` implements this with the `setIntegratorRange()` method.

Importante: Most mechanisms in FRC do not require any integral control, and systems that seem to require integral control to respond well probably have an inaccurate feedforward model.

Voltage Sag

When we operate mechanisms on our robot, we draw current from its battery. This causes the available «bus voltage» that all the robot mechanisms operate off of to drop. This means that the performance of our mechanisms will vary depending on the loading and action of the robot - this is not ideal.

To fix this, most voltage controllers offer a «voltage compensation» setting for their internal control loops that keep the output voltage of the control loops constant despite changes in the bus voltage. The WPILib `MotorController` class offers a `setVoltage` method can do the same thing if the control loops are being run on the RIO (provided you call it every robot loop iteration).

Keep in mind that voltage compensation cannot increase the voltage applied to the motor beyond what is available on the bus - if your actuator is saturating (described below), you'll have to account for that separately.

Actuator Saturation

A controller calculates its output based on the error between the *setpoint* and the current *state*. *Plant* in the real world don't have unlimited control authority available for the controller to apply - that is to say, real mechanisms have some maximum achievable torque/acceleration and velocity.

If our control gains are too aggressive, our control algorithm might try to move the mechanism faster than it is capable of actually going. In this case, the mechanism will «saturate», and behave as if the control gains were smaller than they are. This might adversely affect control response (i.e., result in errors and instability).

If you are encountering problems with actuator saturation, consider modifying your mechanism gearing or powering it with a bigger motor.

32.3 Filtros

Nota: Los datos utilizados para generar los distintos gráficos de demostración se pueden encontrar `aquí`.

Esta sección describe una serie de filtros incluidos con WPILib que son útiles para la reducción de ruido y / o suavizado de entrada.

32.3.1 Introducción a los Filtros

Los filtros son unas de las herramientas más usadas en la tecnología moderna y encontrados en numerosas aplicaciones en la robótica tanto en procesamiento de señal y controles. Comprender la noción de un filtro es crucial para entender la utilidad de varios tipos de filtro proporcionados por WPILib.

¿Qué es un filtro?

Nota: Por el bien de este artículo, asumiremos que toda la información es una serie de datos temporales unidimensionales. Obviamente, los conceptos involucrados son más generales que esto – pero una discusión más a detalle y rigurosa de señales y filtrado esta fuera del alcance de esta documentación.

Entonces, ¿qué *es* exactamente un filtro? Simple, un filtro es un mapeo de flujo de entradas a un flujo de salidas. Eso es decir, el valor de salida de un filtro (en principio) puede depender no solo del valor *actual* de la entrada, sino también *en todo el conjunto de valores pasados y futuros* (por supuesto, en práctica, los filtros provistos por WPILib son implementables en transmisión de datos en tiempo real; por consiguiente, solamente pueden depender de valores *pasados* de la entrada, y no en valores futuros). Esto es un concepto importante, debido a que generalmente usamos los filtros para remover o mitigar *dinámicas* no deseadas en la señal. Cuando filtramos una señal, estamos interesados en modificar *como cambia la señal conforme al tiempo*.

Efectos Usando un Filtro

Reducción de Ruido

Uno de los usos más usuales de los filtros es para la reducción de ruido. Un filtro que reduce el ruido se llama filtro *paso bajo* (debido a que permite que frecuencias bajas «pasen a través» de él mientras bloquea las frecuencias altas). La mayoría de los filtros incluidos en WPILib son filtros efectivos de bajo paso.

Limitación de velocidad

Los filtros también se usan comúnmente para reducir la velocidad a la cual una señal cambia. Esto está estrechamente relacionado con la reducción de ruido, y los filtros que reducen el ruido también tienden a limitar la velocidad de cambia de su salida.

Detección de Borde

La contraparte del filtro paso bajo es el filtro paso alto, el cual solamente permite que altas frecuencias pasen a través hacia la salida. Los filtros de paso alto suelen ser complicados para ser intuitivos, pero un uso común para estos filtros es la detección de bordes - debido a que los filtros paso alto reflejarán cambios repentinos en la entrada mientras ignoran los cambios más lentos o sutiles, estos filtros son útiles para determinar la ubicación de discontinuidades agudas en la señal.

Retraso de Fase

Un efecto negativo inevitable de un filtro paso bajo en tiempo real es la generación del «retraso de fase». Como, se mencionó antes, un filtro en tiempo real depende solamente en valores pasados de la señal (no podemos viajar en el tiempo para obtener futuros valores), al valor filtrado a veces le toma un tiempo en «alcanzar» cuando la entrada comience a cambiar. A mayor reducción del ruido, se genera un mayor retraso. Esto es, en muchos sentidos, *la* compensación del filtrado en tiempo real y deberá ser el factor principal de su diseño del filtro.

De manera interesante, los filtros paso alto generan un *adelanto* de fase, opuesto al retraso de fase ya que exacerban los cambios locales en el valor de entrada.

32.3.2 Filtros Lineales

El primer tipo (y más comúnmente empleado) de filtros que WPILib admite es el filtro lineal o, más específicamente un filtro lineal invariante en el tiempo (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*.

Hay dos amplios tipos de respuestas de impulso: respuesta infinita de impulsos (IIR) y respuesta finita de impulsos (FIR).

Las respuestas infinitas de impulsos tiene un «apoyo» infinito - esto es, ellas son distintas a cero en una región infinitamente grande. Esto significa, que también tienen «memoria» infinita – cuando un valor aparece en el flujo de entrada este influirá en todas las salidas subsecuentes, *para siempre*. Esto es típicamente indeseable desde una perspectiva estricta de procesamiento de señales, sin embargo los filtros con respuestas infinitas de impulsos tienden calcularse fácilmente, ya que se pueden expresar mediante relaciones de recursión simples.

Las respuestas finitas de impulsos tiene un «apoyo» finito – es decir, las respuestas son distintas a cero en una región definida. El «arquetipo» del filtro FIR es un promedio móvil plano – es decir, configurando la salida igual al promedio de n entradas pasadas. Los filtros FIR tienden a tener propiedades más deseadas que los filtros IIR, pero son más complicados para calcular.

Linear filters are supported in WPILib through the `LinearFilter` class (Java, C++, , Python).

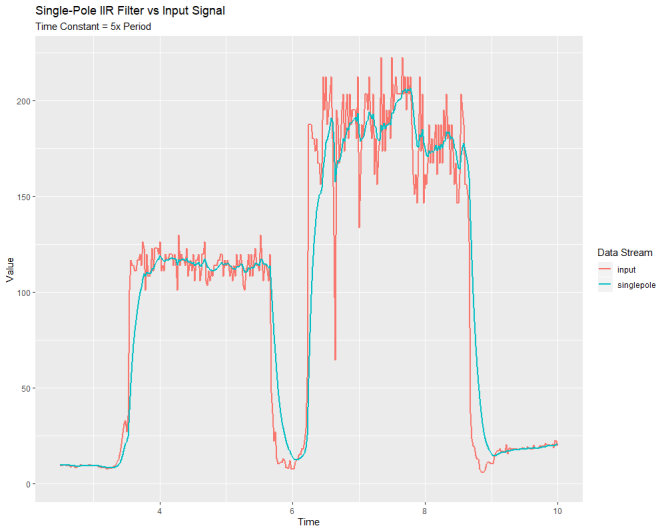
Crear un Filtro Lineal

Nota: La clase C++ `LinearFilter` se basa en el tipo de información usada en el entrada.

Nota: Debido a que los filtros tienen «memoria», cada flujo de entrada requiere su propio objeto de filtro. No intente usar el mismo objeto de filtro para múltiples flujos de entradas.

Mientras sea posible instanciar directamente la clase `LinearFilter` para construir un filtro personalizado, es mucho más conveniente (y común) usar uno de los Factory methods suministrados, en su lugar:

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);
```

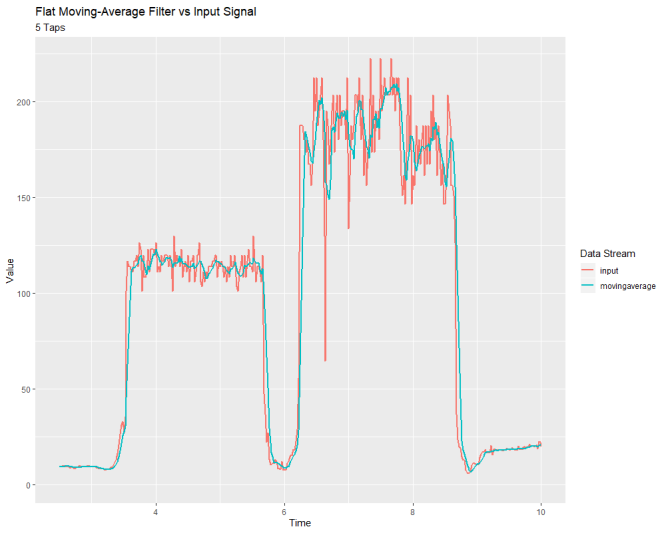
PYTHON

```
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)
```

El parámetro de la «constante de tiempo» determina la «escala de tiempo característica» del impulso de respuesta del filtro; el filtro cancelará cualquier dinámica que ocurra en la escala de tiempo significativamente menor que esto. Relacionada, también está la escala de tiempo aproximada de la introducción del *phase lag*. El recíproco de esta escala de tiempo, multiplicado por 2 pi, es la «frecuencia de corte» del filtro.

El parámetro «periodo» es el periodo en el cual el método `calculate()` del filtro será llamado. Para la mayoría de las implementaciones, esto será el periodo cíclico estándar de 0.02 segundos del main robot.

Media Móvil



El método de fabrica de `media móvil` crea un filtro simple de media móvil plana. Este es el filtro paso bajo FIR más simple, y es útil en muchos contextos iguales como los filtros monopolar IIR. Es un poco más complicado de calcular, pero generalmente se comporta de una manera más suave.

JAVA

```
// Creates a new flat moving average filter
// Average will be taken over the last 5 samples
LinearFilter filter = LinearFilter.movingAverage(5);
```

C++

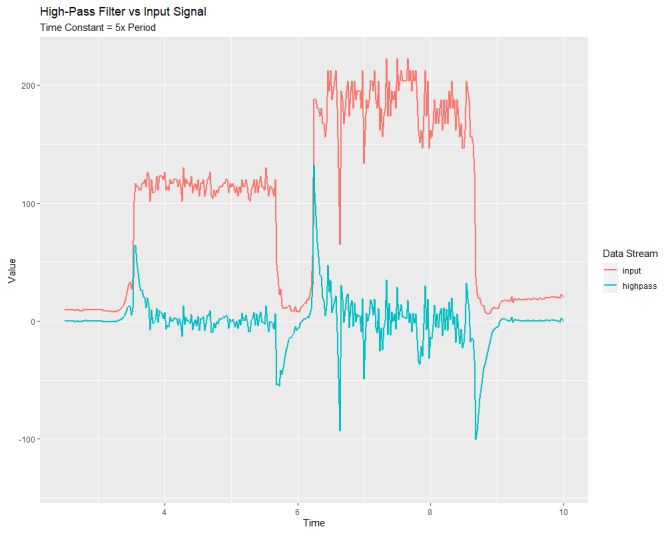
```
// Creates a new flat moving average filter
// Average will be taken over the last 5 samples
frc::LinearFilter<double> filter = frc::LinearFilter<double>::MovingAverage(5);
```

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)
```

El parámetro «taps» es el nombre de ejemplos que serán incluidos en la media móvil plana. Esto se comporta similarmente a la «constante de tiempo» anterior – se le llama constante de tiempo efectiva al número de fases de tiempo multiplicado por el periodo en cual `calculate()` es llamado.

highPass



El método de fábrica `highPass` crea un filtro simple de respuesta de impulso infinita de primer orden. Esto es la «contraparte» del IIR monopolar.

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)
```

El parámetro de la «constante de tiempo» determina la «escala de tiempo característica» de la respuesta del impulso del filtro; el filtro cancelará cualquier dinámica de la señal que ocurra en la escala de tiempo significativamente más larga que esta. Relacionado, esta también es

la escala de tiempo aproximada del *phase lead* introducida. El recíproco de esta escala de tiempo, multiplicado por 2 pi, es la «frecuencia de corte» del filtro.

El parámetro «periodo» es el periodo en el cual el método `calculate()` del filtro será llamado. Para la mayoría de las implementaciones, esto será el periodo cíclico estándar de 0.02 segundos del main robot.

Usando un filtro Lineal

Nota: En orden para que el filtro creado obedezca el parámetro de tiempo de escala especificado, la función `calculate()` deberá ser llamada constantemente en el periodo especificado. Si, por alguna razón, un lapso significante en `calculate()` debe ocurrir, el método de `reset()` del filtro deberá ser llamado antes para su uso futuro.

Una vez que su filtro sea creado, usarlo es fácil – solamente llame el método `calculate()` con la entrada más reciente para obtener la salida filtrada.

JAVA

```
// Calculates the next value of the output
filter.calculate(input);
```

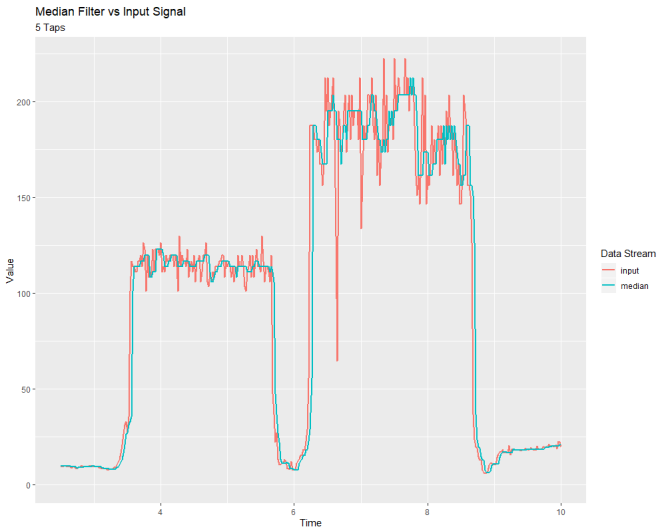
C++

```
// Calculates the next value of the output
filter.Calculate(input);
```

PYTHON

```
# Calculates the next value of the output
filter.calculate(input)
```

32.3.3 Filtro de Mediana



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.

El filtro de mediana es el más útil para remover valores atípicos del flujo de entrada. Esto lo hace particularmente bueno para filtrar entradas de sensores distantes, que son propensos a interferencias ocasionales. A diferencia de una media móvil, el filtro de mediana permanecerá intacto por pocas alteraciones de los valores, no importa cuán extremo sean.

The median filter is supported in WPILib through the `MedianFilter` class (Java, C++, , `Python`).

Creando un MedianFilter

Nota: La clase MedianFilter de C++ está basada en el tipo de datos usados para la entrada.

Nota: Debido a que los filtros tienen "memoria", cada entrada de flujo requiere su propio objeto filtro. *No* intente usar el mismo objeto filtro para múltiples entradas de flujo.

Crear un MedianFilter es simple:

JAVA

```java
// Creates a MedianFilter with a window size of 5 samples
MedianFilter filter = new MedianFilter(5);
```

C++

```cpp
// Creates a MedianFilter with a window size of 5 samples
frc::MedianFilter<double> filter(5);
```

PYTHON

```python
from wpimath.filter import MedianFilter

# Creates a MedianFilter with a window size of 5 samples
filter = MedianFilter(5)
```

Usando un MedianFilter

Una vez que su filtro ha sido creado, usarlo es sencillo - simplemente llame al método calculate() con la entrada más reciente para obtener la salida filtrada:

JAVA

```java
// Calculates the next value of the output
filter.calculate(input);
```

C++

```cpp
// Calculates the next value of the output
filter.Calculate(input);
```

PYTHON

```python
# Calculates the next value of the output
filter.calculate(input)
```

32.3.4 Slew Rate Limiter - Limitador de velocidad de respuesta

Un uso común para los filtros en FRC® es suavizar el comportamiento de las entradas del control (por ejemplo, las entradas del joysitick de los controladores del driver). Desafortunadamente, un simple filtro de bajo-paso está pobremente adecuado para el trabajo; mientras que el filtro de bajo-paso suaviza la respuesta de un flujo de entrada a cambios súbitos, también limpiará detalles finos de control e introducirá una fase de retraso. Una mejor solución es limitar directamente la tasa de cambio de la entrada del control. Esto se hace con un *slew rate limiter*, un filtro que se encarga de la máxima tasa de cambio de la señal.

Un slew rate limiter puede ser pensado como un tipo de perfil de movimiento primitivo, De hecho, el slew rate limiter es el equivalente en primer orden de un *Perfil de movimiento Trapezoidal* soportado por WPILib – precisamente es el caso limitador de un movimiento trapezoidal cuando la restricción de aceleración es permitida a tender a infinito. De acuerdo con esto, un slew rate limiter es una buena opción para aplicar un perfil de movimiento verdadero a un flujo de velocidad con punto fijo (o voltajes, que son usualmente aproximadamente proporcionales a la velocidad). Para flujos de entrada que controlan posición, usualmente es mejor usar un perfil trapezoidal adecuado.

Slew rate limiting is supported in WPILib through the `SlewRateLimiter` class (Java, C++, `Python`).

Creando un SlewRateLimiter

Nota: La clase de C++ `SlewRateLimiter` está basada en el tipo de unidad de la entrada. Para más informacion de unidades de C++, ver *Biblioteca de unidades de C++*.

Nota: Debido a que los filtros tienen "memoria", cada entrada de flujo requiere su propio objeto filtro. *No* intente usar el mismo objeto filtro para múltiples entradas de flujo.

Crear un SlewRateLimiter es 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)
```

Usar un SlewRateLimiter

Una vez que su filtro ha sido creado, usarlo es sencillo - simplemente llame al método ``calculate()``con la entrada más reciente para obtener la salida filtrada:

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)
```

Using a SlewRateLimiter with DifferentialDrive

Nota: The C++ example below templates the filter on `units::scalar` for use with doubles, since joystick values are typically dimensionless.

A typical use of a SlewRateLimiter is to limit the acceleration of a robot's drive. This can be especially handy for robots that are very top-heavy, or that have very powerful drives. To do this, apply a SlewRateLimiter to a value passed into your robot drive function:

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 Debouncer

A debouncer is a filter used to eliminate unwanted quick on/off cycles (termed «bounces,» originally from the physical vibrations of a switch as it is thrown). These cycles are usually due to a sensor error like noise or reflections and not the actual event the sensor is trying to record.

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

The WPILib `Debouncer` can be configured in three different modes:

- Rising (default): Debounces rising edges (transitions from *false* to *true*) only.
- Falling: Debounces falling edges (transitions from *true* to *false*) only.
- Both: Debounces all transitions.

Usage

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)

# 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 Clases Geométricas

Esta sección cubre las clases geométricas de WPILib.

32.4.1 Traslación, Rotación y Pose.

Traslación

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.

Usted puede obtener la distancia a otro objeto `Translation2d` al usar `getDistance(Translation2d other)`, el cual regresa la distancia a otro Transalation2d al usar el teorema de Pitágoras.

Nota: `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.

Rotación

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.

Nota: `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.

Nota: `Rotation2d` does not wrap the value of the angle, so if a value of 400 degrees is passed into the constructor, then 400 degrees will be returned in subsequent value calls.

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 Transformaciones

Translation2d

Las operaciones en un `Translation2d` llevan a cabo operaciones en el vector representado por `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.
- Rotación: La rotación de un `Translation2d` por una rotación contraria a las manecillas del reloj θ sobre el origen puede ser logrado al usar `rotateBy`. Esto es equivalente a multiplicar el vector por la matriz $\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$
- Additionally, you can rotate a `Translation2d` by 180 degrees by using `unaryMinus` in Java, or the unary `-` operator in C++/Python.

Rotation2d

Las transformaciones para `Rotation2d` son solo operaciones aritméticas en la medida del ángulo representadas por `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 y 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` representa una transformación **relativa**. Tiene un componente de rotación y traslación. Transformar un `Pose2d` por un `Transform2d` rota el componente de traslación de la transformación por la rotación de la pose, y entonces suma el componente de traslación y el componente de rotación de la pose. En otras palabras, `Pose2d.plus(Transform2d)` regresa $\begin{bmatrix} x_p \\ y_p \\ \theta_p \end{bmatrix} + \begin{bmatrix} cos\theta_p & -sin\theta_p & 0 \\ sin\theta_p & cos\theta_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$

`Twist2d` representa un cambio en distancia a través de un arco, Usualmente, esta clase es usada para representar el movimiento de un tren motriz, donde el componente x es la distancia frontal manejada, el componente y es la distancia manejada hacia un lado (Izquierda positiva), y el componente θ es el cambio de rumbo, La matemática subyacente detrás de encontrar la pose exponencial (Posición nueva tras moverse de frente a traves de la curvatura del giro) puede ser encontrada aquí en el capítulo 10.

Nota: Para trenes motrices no holonómicos, el componente y de `Twist2d` debe siempre ser 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

Nota: 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.

Nota: 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

Nota: 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.

Nota: 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.

Nota: 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.

Advertencia: 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.

Advertencia: 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:

Nota: A drivetrain routine is shown below, but the same motions will occur on any mechanism.

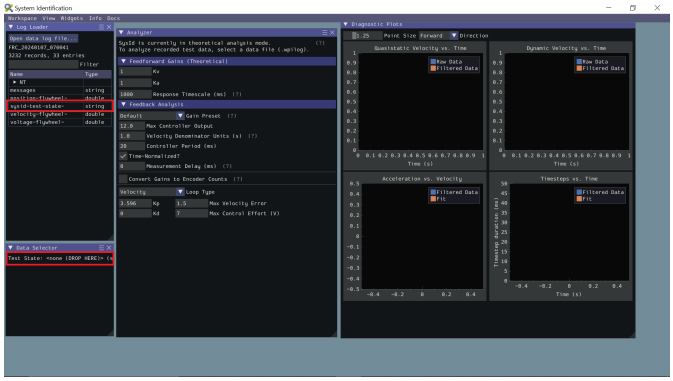
After all four tests have been completed, use the `DataLogTool` to retrieve the log file from the roboRIO.

32.5.4 Loading Data

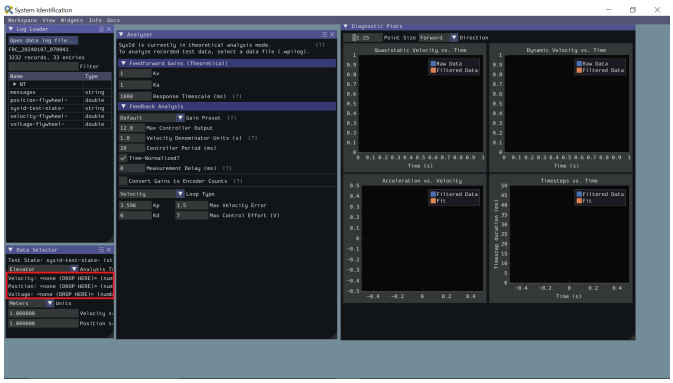
After downloading the WPILog containing the tests from the roboRIO, go to the `Log Loader` pane in SysId and click `Open data log file...`.

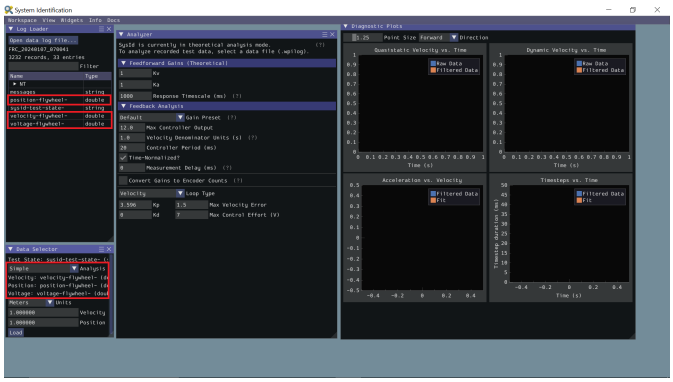
After the file loads, look for a `string` type entry with a name containing «state». Drag this entry into the Data Selector pane's Test State slot.

Nota: SysIdRoutine will name the entry «sysid-test-state-mechanism», where «mechanism» is the name passed to the `Mechanism` contructor or the subsystem name.

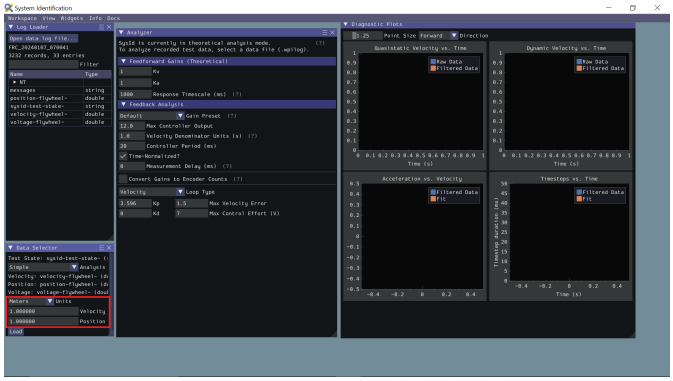


Now the `Data Selector` pane will present `Position`, `Velocity`, and `Voltage` slots. In the `Log Loader` pane, find entries starting with each of those terms and containing the motor name you set in the `log` callback, and drag those into the `Data Selector` slots.





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.

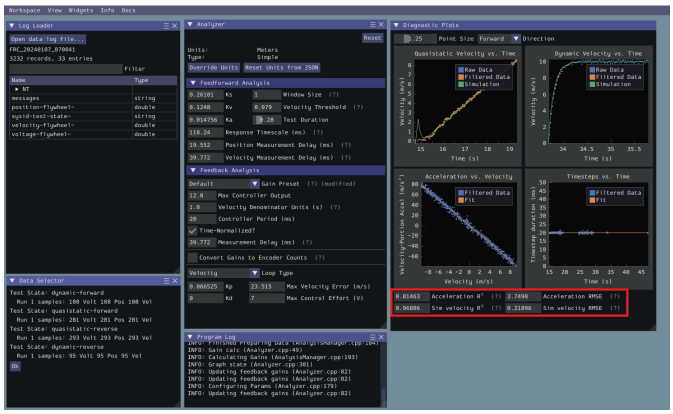


Ensure the correct analysis type has been selected, then click the `Load` button and move on to checking the fit diagnostics in the `Diagnostics` pane.

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*.



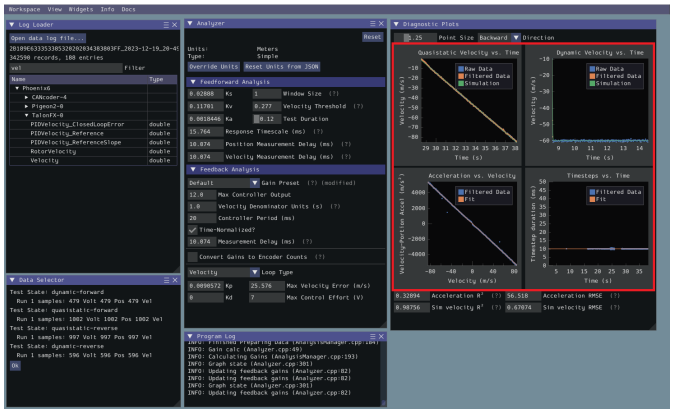
The acceleration r-squared is the fraction of the variance in measured acceleration (used as the independent variable in the SysId regression) explained by the linear model. This can be quite variable, because acceleration is very susceptible to system noise. Mechanisms tend to vibrate quite a bit, so 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

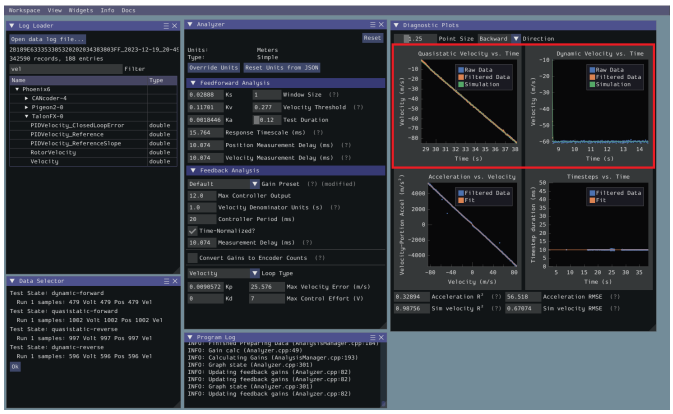
SysId also produces several diagnostic plots to help users evaluate the quality of their model fit.



Time-Domain Plots

Nota: 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:



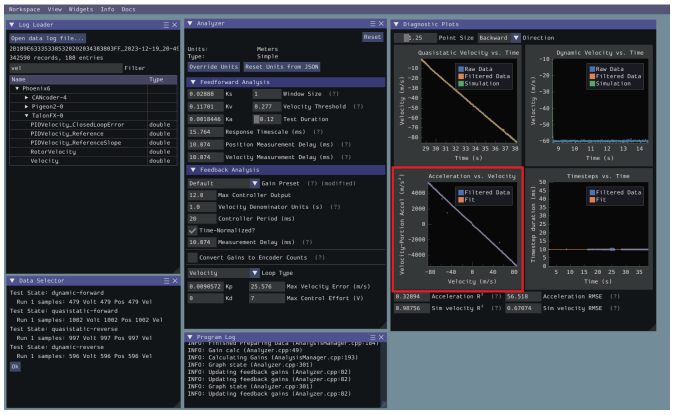
The velocity time domain plots contain three sets of data: Raw Data, Filtered Data, and Simulation. The Raw Data is the recorded data from your robot, the Filtered Data is the data after a median filter has been applied to the data, and the Simulation represents the velocity predictions of a model based 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.

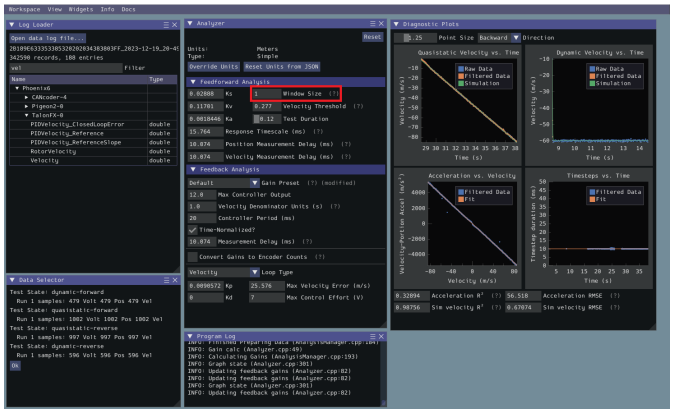
Deviation from this behavior is a sign of an *error*, either in your robot setup, analysis settings, or your test procedure.

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.



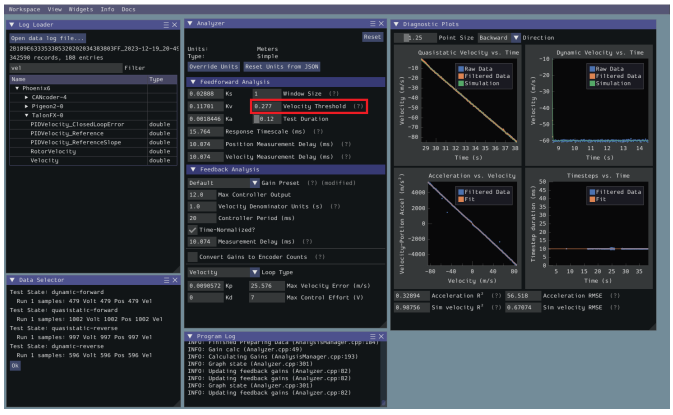
However, if your robot or mechanism has low mass compared to the motor power, this may «eat» what little meaningful acceleration data you have. In these cases kA will tend towards zero and can be ignored for feedforward purposes. However, if kA cannot be accurately measured, the calculated feedback gains are likely to be inaccurate, and manual tuning may be required.

Common Failure Modes

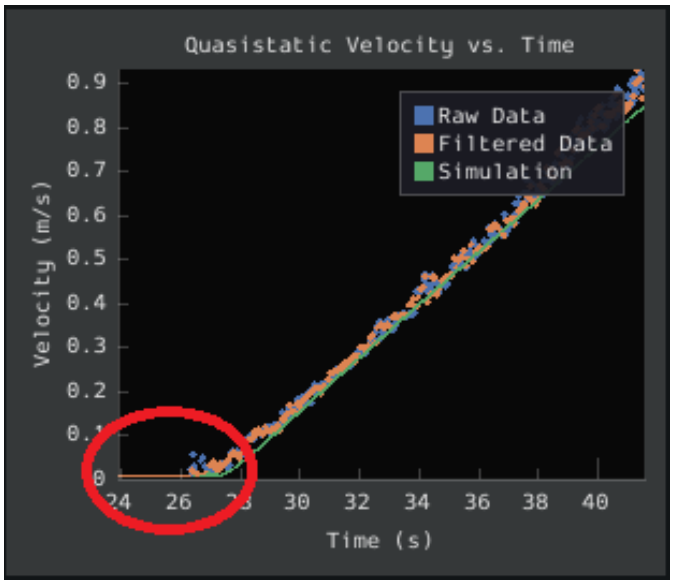
When something has gone wrong with the identification, diagnostic plots and console output provide crucial clues as to *what* has gone wrong. This section describes some common failures encountered while running the system identification tool, the identifying features of their diagnostic plots, and the 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.



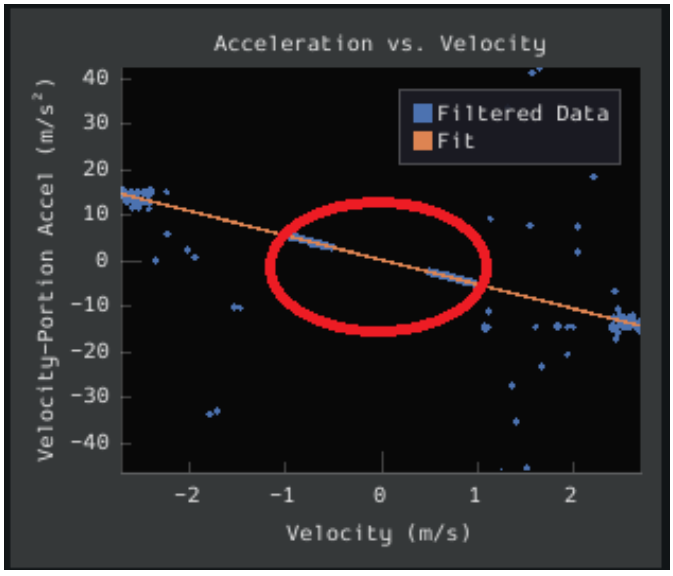
Velocity Threshold Too Low



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

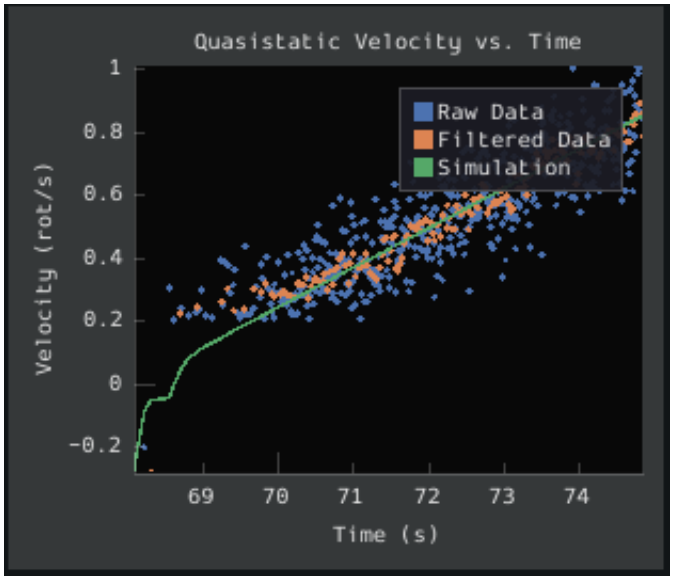


While not nearly as problematic as a too-low threshold, a velocity threshold that is too high will result in a large «gap» in the acceleration-versus-velocity plot.

To solve this, decrease the velocity threshold and re-analyze the data.

Noisy Velocity Signals

Nota: 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).

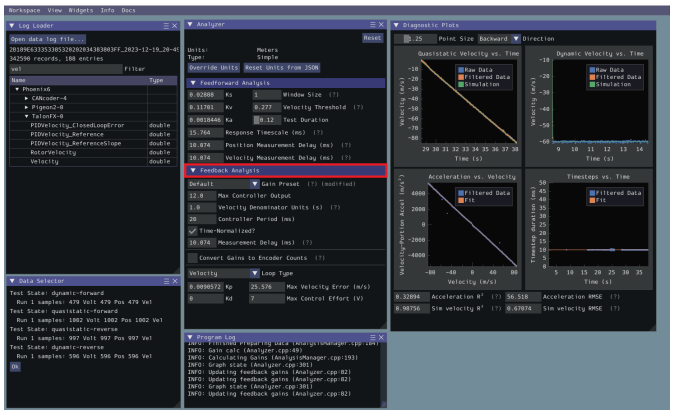
32.5.6 Analyzing Data

Feedforward Analysis

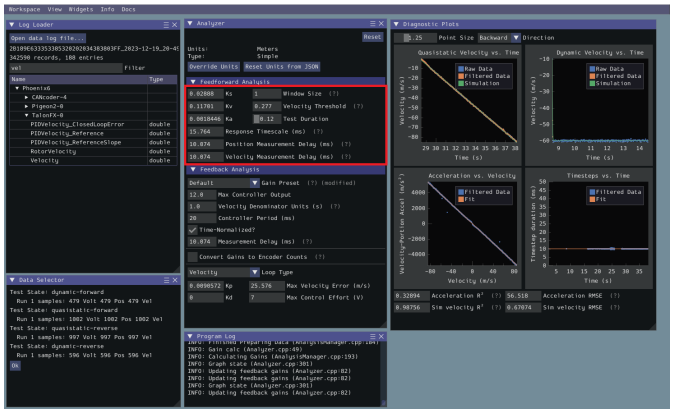
Nota: For information on what the calculated feedback gains mean, see *The Permanent-Magnet DC Motor Feedforward Equation*. For information on using the calculated feedback gains in code, see *feedforward control*.

Click the dropdown arrow on the *Feedforward* Section.

Nota: If you would like to change units, you will have to press the *Override Units* button and fill out the information on the popup.



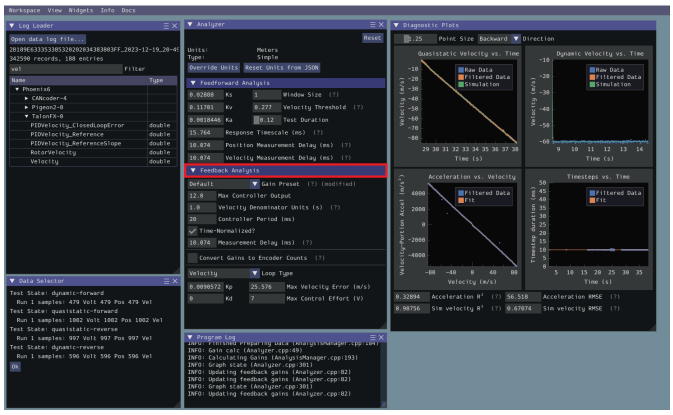
The computed mechanism system parameters will then be displayed.



Feedback Analysis

Importante: 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.



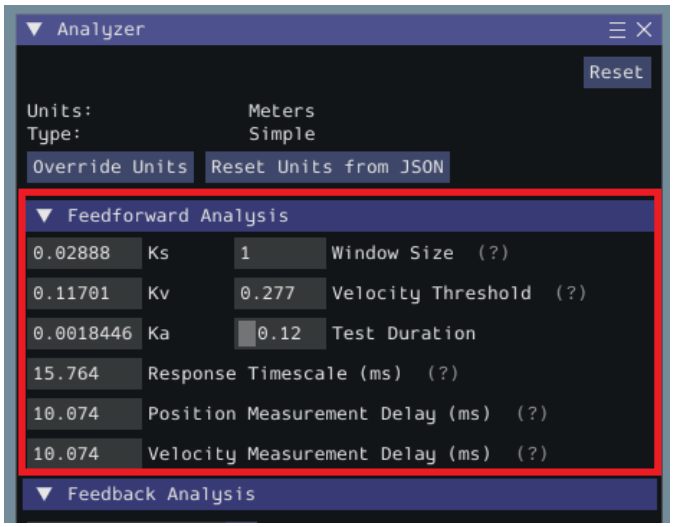
This view can be used to calculate optimal feedback gains for a PD or P controller for your mechanism (via *LQR*).

Enter Controller Parameters

Nota: 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.

Nota: 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

Nota: 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*.

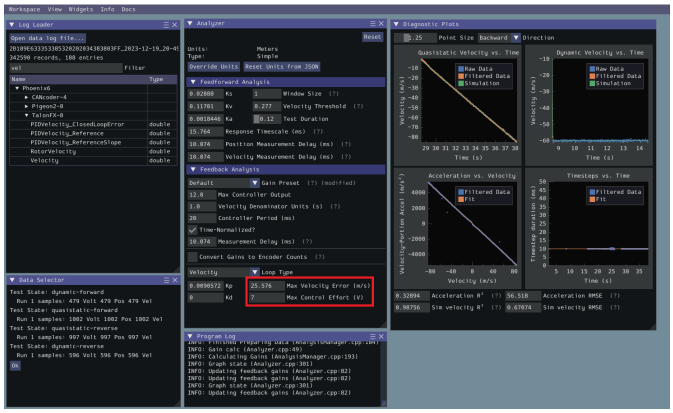
The following only applies if the user decides to implement their own custom filtering settings (e.g. adding a moving average filter to a WPILib PID loop or changing smart motorcontroller measurement period and/or measurement window size) as the measurement delay must be recalculated. Here is the general 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.

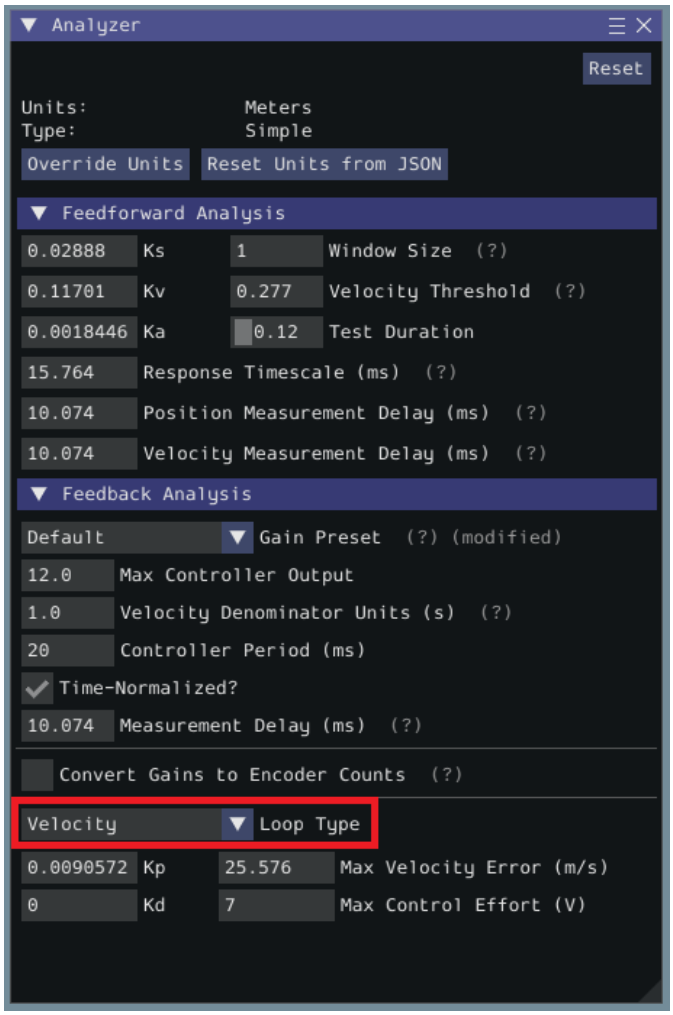


As a rule, smaller values for the *Max Acceptable Error* and larger values for the *Max Acceptable Control Effort* will result in larger gains - this will result in larger control efforts, which can grant better setpoint-tracking but may cause more violent behavior and greater wear on components.

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 Additional Utilities and Tools

This page mainly covers useful information about additional functionality that this tool provides.

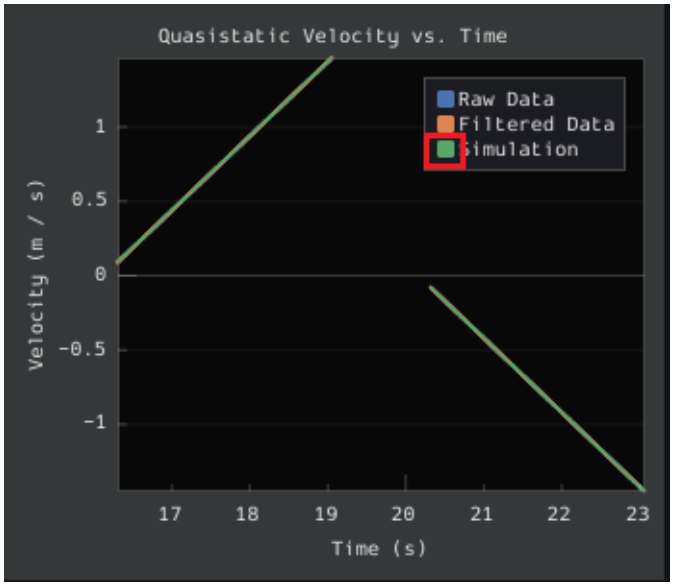
ImGui Tips

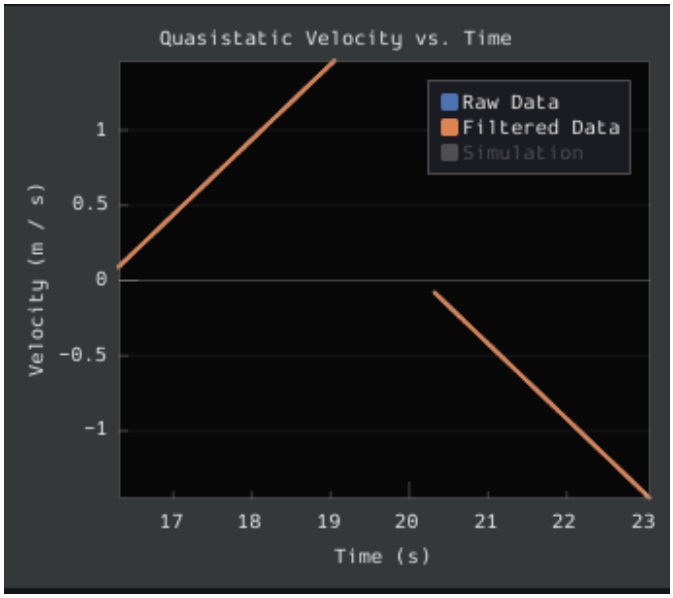
The following are essentially handy features that come with the ImGui framework that SysId uses:

Showing and Hiding Plot Data

To add or remove certain data from the plots, click on the color of the data that you would like to hide or remove.

For example, if we want to hide sim data, we can click the green color box.

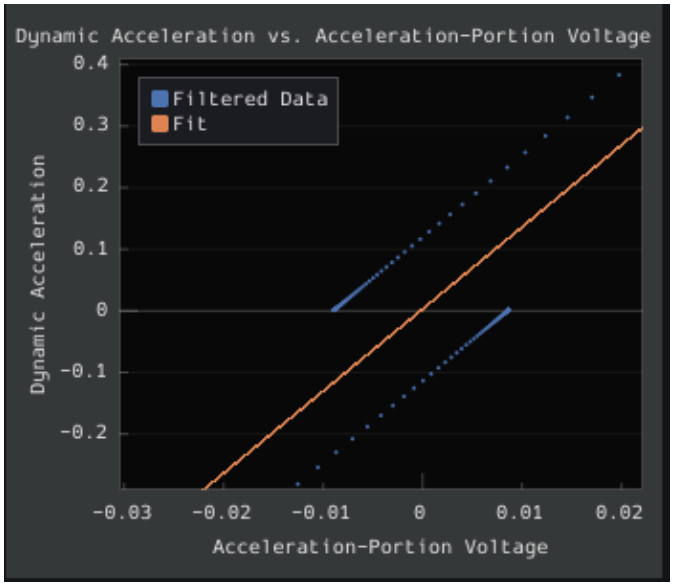




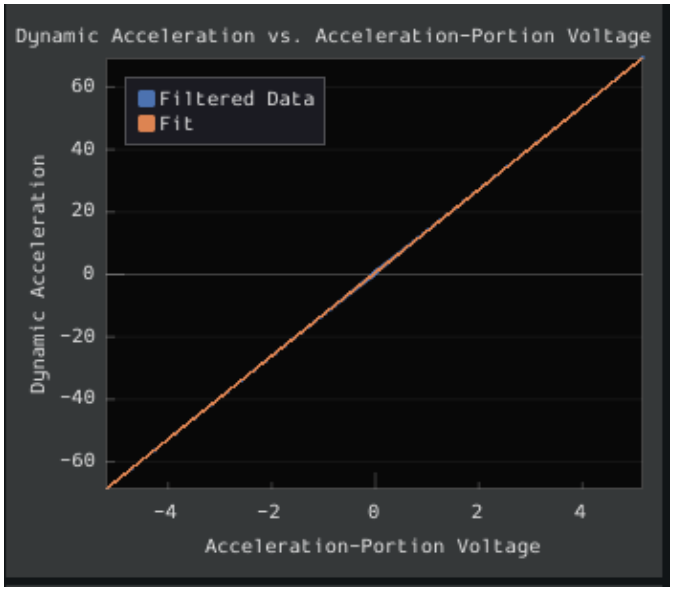
Auto Sizing Plots

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:



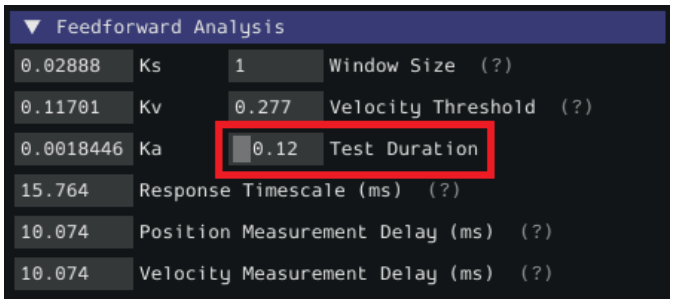
After double clicking, it is automatically resized:



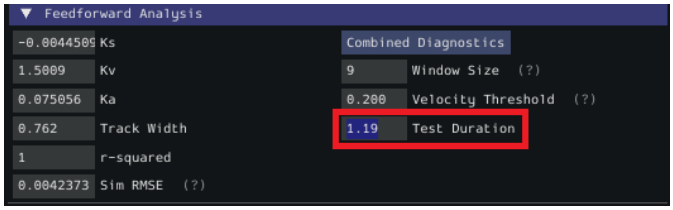
Setting Slider Values

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 Controladores

This section describes various WPILib feedback and feedforward controller classes that are useful for controlling the motion of robot mechanisms, as well as motion-profiling classes that can automatically generate setpoints for use with these controllers.

32.6.1 Control PID en WPILib

Nota: This article focuses on in-code implementation of PID control in WPILib. For a conceptual explanation of the working of a PIDController, see *Introducción a PID*

Nota: Para una guía en implementar controles PID a través de un *marco de referencia base comandos*, ver *Control PID a través de subsistemas PID y comandos PID*.

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.

Usando la Clase PIDController

Construyendo un PIDController

Nota: Mientras que `PIDController` puede ser usado de manera asincronica, este *no* provee ninguna función de seguridad de hilos - el asegurar una operación segura para hilos se deja completamente al usuario, y de este modo el uso asincronico es recomendado solamente para equipos avanzados.

Para usar las funcionalidades de control PID de WPILib, los usuarios deben primero construir un objeto `PIDController` con las ganancias deseadas:

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 cuarto parámetro opcional puede proveerse al constructor, especificando el periodo en el cual el controlador será ejecutado. El objeto `PIDController` está echo principalmente para uso sincrónico desde el loop principal del robot, y este valor está puesto por default a 20ms.

Usando la salida del loop de retroalimentación.

Nota: El `PIDController` asume que el método `calculate()` es llamado regularmente en un intervalo consistente con el periodo configurado. La falla en esto resultará un resultado no deseado en el comportamiento del loop.

Usar un construido `PIDController` es simple: Solo llame el método `calculate()` desde el loop principal del robot (ejemplo: El método `autonomousPeriodic()` del 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))
```

Checando errores

Nota: `getPositionError()` y `getVelocityError()` son nombrados asumiendo que el loop está controlando una posición - para un loop que está controlando velocidad, estos regresan el error de velocidad y el error de la aceleración, respectivamente.

El error actual del proceso medido se regresa por medio de la función `getPositionError()`, mientras que su derivada regresa por medio de la función `getVelocityError()`:

Especificando y checando tolerancias

Nota: Si solo una tolerancia de posición es especificada, la tolerancia de velocidad se pone como default a infinito.

Nota: Tal como arriba, «position» se refiere al proceso de medición variable, y «velocity» a su derivada - de esta forma, para un loop de velocidad, estos son, de hecho, velocidad y aceleración respectivamente.

Ocasionalmente, es útil saber si un controlador ha rastreado el setpoint a una tolerancia dada - por ejemplo, para determinar si un comando debería ser finalizado, o (mientras siguiendo un perfil de movimiento) si el movimiento está siendo impedido y necesita ser planeado otra vez.

Para hacer esto, primero necesitamos especificar las tolerancias con el método `setTolerance()`; entonces, podemos checarlo con el método `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()
```

Reiniciando el controlador

Algunas veces es deseable aclarar el estado interno (más importante, el acumulador integral) de un `PIDController`, teniendo en cuenta que puede que ya no sea valido (ejemplo: cuando el `PIDController` ha sido desactivado y luego reactivado). Esto puede lograrse llamando el método `reset()`.

Ajustando un valor integrador máximo.

Nota: Los integradores introducen inestabilidad e histéresis al sistema de retroalimentación del loop. Se recomienda fuertemente que los equipos eviten usar ganancias integrales a menos de que ninguna otra opción sirva - muy a menudo, los problemas que pueden ser resueltos con un integrador pueden ser resueltos por medio del uso de un *feedforward* más preciso.

A typical problem encountered when using integral feedback is excessive «wind-up» causing the system to wildly overshoot the setpoint. This can be alleviated in a number of ways - the WPILib `PIDController` class enforces an integrator range limiter to help teams overcome this issue.

By default, the total output contribution from the integral gain is limited to be between -1.0 and 1.0.

The range limits may be increased or decreased using the `setIntegratorRange()` method.

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)
```

Ajustando entradas continuas

Advertencia: Si su mecanismo no es capaz de usar un movimiento de rotación continua completo (ejemplo: una torreta con cables que se giran mientras rota), *no* active la entrada continua a menos de que tenga implementada una característica de seguridad adicional para prevenir que el mecanismo se mueva más allá de sus limites.

Advertencia: La función de entrada continua *no* envuelve automáticamente sus valores de entrada - asegúrese que sus valores de entrada, mientras use esta función, nunca estén fuera del rango establecido.

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.

Para configurar un `PIDController` para automáticamente hacer esto, use el método `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)
```

Salida del controlador de sujeción

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 Control Feedforward en WPILib

Nota: 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.

The WPILib Feedforward Classes

WPILib proporciona una serie de clases para ayudar a los usuarios a implementar un control de retroalimentación preciso para sus mecanismos. En muchos sentidos, una retroalimentación precisa es más importante que la retroalimentación para el control efectivo de un mecanismo. Dado que la mayoría de los mecanismos de FRC® obedecen de cerca a las ecuaciones del sistema bien entendidas, comenzar con un feedforward preciso es fácil y enormemente beneficioso para un control de mecanismo preciso y robusto.

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 actualmente proporciona las siguientes tres clases auxiliares para el control anticipado:

- *SimpleMotorFeedforward* (Java, C++, `Python`)
- *ArmFeedforward* (Java, C++, `Python`)
- *ElevatorFeedforward* (Java, C++, `Python`)

SimpleMotorFeedforward

Nota: En C ++, la clase `SimpleMotorFeedforward` se basa en el tipo de unidad utilizada para las mediciones de distancia, que puede ser angular o lineal. Las ganancias pasadas *deben* tener unidades consistentes con las unidades de distancia, o se arrojará un error de tiempo de compilación. `kS` debe tener unidades de `voltios` , `kV` debe tener unidades de `voltios*segundos / distancia` y `kA` debe tener unidades de ``voltios*segundos^2 / distan-

cia ``. Para obtener más información sobre las unidades C++, consulte docs/software/basic-planning/cpp-units:The C ++ Units Library.

Nota: Los componentes de feedforward de Java calcularán los resultados en unidades determinadas por las unidades de las ganancias de feedforward proporcionadas por el usuario. Los usuarios *deben* tener cuidado de mantener las unidades consistentes, ya que WPILibJ no tiene un sistema de unidad de tipo seguro.

Nota: 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 clase `SimpleMotorFeedforward` calcula el feedforward para mecanismos que constan de motores DC de imanes permanentes sin carga externa más que fricción e inercia, como volantes y accionamientos de robots.

Para crear un `SimpleMotorFeedforward`, simplemente constrúyalo con las ganancias requeridas:

Nota: La ganancia de kA se puede omitir y, si es así, tendrá un valor predeterminado de cero. Para muchos mecanismos, especialmente aquellos con poca inercia, no es necesario.

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)
```

Para calcular el feedforward, simplemente llame al método `calculate()` con la velocidad y aceleración deseadas del motor:

Nota: El argumento de aceleración se puede omitir de la llamada `calculate()` y, si es así, se tomará por defecto un valor de cero. Esto debe hacerse siempre que no haya un punto de ajuste de aceleración claramente definido.

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)
```

ArmFeedforward

Nota: In C++, the `ArmFeedforward` class assumes distances are angular, not linear. The passed-in gains *must* have units consistent with the angular unit, or a compile-time error will be thrown. `kS` and `kG` should have units of `volts`, `kV` should have units of `volts * seconds / radians`, and `kA` should have units of `volts * seconds^2 / radians`. For more information on C++ units, see *Biblioteca de unidades de C++*.

Nota: Los componentes de feedforward de Java calcularán los resultados en unidades determinadas por las unidades de las ganancias de feedforward proporcionadas por el usuario. Los usuarios *deben* tener cuidado de mantener las unidades consistentes, ya que WPILibJ no tiene un sistema de unidad de tipo seguro.

Nota: 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 clase `ArmFeedforward` calcula el feedforwards para brazos que están controlados directamente por un motor DC de imán permanente, con carga externa de fricción, inercia y masa del brazo. Este es un modelo preciso de la mayoría de los brazos en FRC.

Para crear un `ArmFeedforward`, simplemente constrúyalo con las ganancias requeridas:

Nota: La ganancia de kA se puede omitir y, si es así, tendrá un valor predeterminado de cero. Para muchos mecanismos, especialmente aquellos con poca inercia, no es necesario.

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)
```

Para calcular el feedforward, simplemente llame al método `` calculate () `` con la posición, velocidad y aceleración deseadas del brazo:

Nota: El argumento de aceleración se puede omitir de la llamada `calculate()` y, si es así, se tomará por defecto un valor de cero. Esto debe hacerse siempre que no haya un punto de ajuste de aceleración claramente definido.

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)
```

ElevatorFeedforward

Nota: 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 *Biblioteca de unidades de C++*.

Nota: Los componentes de feedforward de Java calcularán los resultados en unidades determinadas por las unidades de las ganancias de feedforward proporcionadas por el usuario. Los usuarios *deben* tener cuidado de mantener las unidades consistentes, ya que WPILibJ no tiene un sistema de unidad de tipo seguro.

Nota: 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 clase `ElevatorFeedforward` calcula el feedforwards para ascensores que consisten en motores DC de imán permanente cargados por fricción, inercia y la masa del ascensor. Este es un modelo preciso de la mayoría de los ascensores en FRC.

Para crear un `ElevatorFeedforward`, simplemente constrúyalo con las ganancias requeridas:

Nota: La ganancia de `kA` se puede omitir y, si es así, tendrá un valor predeterminado de cero. Para muchos mecanismos, especialmente aquellos con poca inercia, no es necesario.

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)
```

Para calcular el feedforward, simplemente llame al método `calculate()` con la velocidad y aceleración deseadas del motor:

Nota: El argumento de aceleración se puede omitir de la llamada `calculate()` y, si es así, se tomará por defecto un valor de cero. Esto debe hacerse siempre que no haya un punto de ajuste de aceleración claramente definido.

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)
```

Uso de Feedforward para controlar los mecanismos

Nota: 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.

El control Feedforward se puede utilizar completamente por sí solo, sin un controlador de retroalimentación. Esto se conoce como control de «bucle abierto» y para muchos mecanismos (especialmente los accionamientos de robots) puede resultar perfectamente satisfactorio. Se puede emplear un `SimpleMotorFeedforward` para controlar el accionamiento de un robot de la siguiente manera:

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 Combinando prealimentación y Control PID

Nota: Este artículo cubre la implementación en código del control PID / prealimentación combinado con las clases de biblioteca proporcionadas por WPILib. Próximamente se publicará documentación que describe los conceptos involucrados con más detalle.

Los controladores de realimentación y retroalimentación se pueden usar de forma aislada, pero son más efectivos cuando se combinan. Afortunadamente, combinar estos dos métodos de control es *extremadamente* sencillo - uno simplemente suma sus resultados.

Usando prealimentación con un PIDController

Users may add any feedforward they like to the output of the controller before sending it to their motors:

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)
```

Además, feedforward es una característica completamente separada de la retroalimentación y, por lo tanto, no tiene ninguna razón para manejarse en el mismo objeto de controlador, ya que esto viola la separación de preocupaciones. WPILib viene con varias clases auxiliares para calcular voltajes de retroalimentación precisos para mecanismos FRC|reg| - para obtener más información, consulte *Control Feedforward en WPILib*.

Uso de componentes Feedforward con PID

Nota: 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.

¿Cómo sería un ejemplo más completo de control PID / prealimentación combinado? Considera el *drive example* de la página de prealimentación. Podemos modificar esto fácilmente para incluir control de retroalimentación (con un componente `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)
    )
```

Otros tipos de mecanismos se pueden manejar de manera similar.

32.6.4 Perfil de movimiento trapezoidal en WPILib

Nota: Este artículo cubre la generación en código de perfil de movimiento trapezoidal. A continuación se presentan más detalles que describen los conceptos involucrados.

Nota: Para obtener una guía sobre la implementación de la clase *TrapezoidProfile`* en *command-based framework* framework, ver *Creación de perfiles de movimiento a través de los subsistemas TrapezoidProfile y los comandos TrapezoidProfile*.

Nota: La clase `TrapezoidProfile`, utilizada por su cuenta, es más útil cuando se compone con un controlador personalizado (tales como un controlador de motor «inteligente» con una funcionalidad PID integrada). Para integrarlo con un WPILib `ControladorPID`, vea *Combinando perfiles de movimiento y control de PID con ProfiledPIDController*.

Mientras el avance y el control de la retroalimentación ofrecen maneras convenientes de lograr un punto fijo, estamos constantemente enfrentándonos al problema de la generación de un punto fijo para nuestros mecanismos. Si bien el enfoque ingenuo de comandar inmediatamente un mecanismo a su estado deseado puede funcionar, usualmente no es óptimo. Para mejorar el manejo de nuestros mecanismos, normalmente deseamos ordenar mecanismos a una *secuencia* de puntos fijos que interpolan suavemente entre su estado actual, y su estado deseado.

To help users do this, WPILib provides a `TrapezoidProfile` class (Java, C++, `Python`).

Creando un Perfil Trapezoidal

Nota: En C++, la clase `TrapezoidProfile` está basada en el tipo de unidad utilizada para medidas de distancia, las cuales pueden ser angulares o lineales. Los valores pasados *deben* tener unidades consistentes con las unidades de distancia, o se producirá un error de compilación de tiempo. Para más información de las unidades C++, vea *Biblioteca de unidades de C++*.

Restricciones

Nota: Los diversos *feedforward helper classes* proporcionan métodos para calcular la máxima velocidad simultáneamente alcanzable y la aceleración de un mecanismo. Esto puede ser muy útil para calcular restricciones de movimientos adecuadas para ``Perfil Trapezoidal`.

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)
```

Estados de Inicio y 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)
```

Poniéndolo todo junto

Nota: C++ normalmente es capaz de inferir el tipo de las clases internas y, por lo tanto, se puede enviar una lista de inicializadores simple (sin el nombre de clase) como parámetro. Los nombres completos de la clase están incluidos en el siguiente ejemplo.

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))
```

Usando un Perfil Trapezoidal

Muestreo del Perfil

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))
```

Usando el Estado

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)
```

Ejemplo de Uso Completo

Nota: 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 =
23       new TrapezoidProfile(new TrapezoidProfile.Constraints(1.75, 0.75));
24   private TrapezoidProfile.State m_goal = new TrapezoidProfile.State();
25   private TrapezoidProfile.State m_setpoint = new TrapezoidProfile.State();
26
27   @Override
28   public void robotInit() {
29     // Note: These gains are fake, and will have to be tuned for your robot.
30     m_motor.setPID(1.3, 0.0, 0.7);
31   }
32
33   @Override
34   public void teleopPeriodic() {
35     if (m_joystick.getRawButtonPressed(2)) {
36       m_goal = new TrapezoidProfile.State(5, 0);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    } 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);
  }

  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,
```

(continúe en la próxima página)

(proviene de la página anterior)

```
                        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)

        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)
```

(continúe en la próxima página)

(proviene de la página anterior)

```

        # 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 Combinando perfiles de movimiento y control de PID con ProfiledPIDController

Nota: Para una guia implementando la clase `ProfiledPIDController` en el *command-based framework* entorno de trabajo, vea *Combinando Motion Profiling y PID Basado en Comandos*.

En el artículo anterior, vimos cómo usar la clase `TrapezoidProfile` para crear y usar un perfil de movimiento trapezoidal. El código de ejemplo de ese artículo demuestra la composición manual de la clase `TrapezoidProfile` con la función de control PID externo de un controlador de motor «smart».

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.

Usando la clase ProfiledPIDController

Nota: En C ++, la clase `ProfiledPIDController` se basa en el tipo de unidad utilizada para las mediciones de distancia, que puede ser angular o lineal. Los valores pasados *deben* tener unidades consistentes con las unidades de distancia, o se lanzará un error en tiempo de compilación. Para obtener más información sobre las unidades C ++, consulte *Biblioteca de unidades de C++*.

Nota: Gran parte de la funcionalidad de `ProfiledPIDController` es efectivamente idéntica a la de `PIDController`. En consecuencia, este artículo solo cubrirá las características que han cambiado sustancialmente para adaptarse a la funcionalidad de creación de perfiles de

movimiento. Para obtener información sobre las características estándar del `PIDController`, consulte *Control PID en WPILib*.

Construyendo un ProfiledPIDController

Nota: C++ a menudo puede inferir el tipo de las clases internas y, por lo tanto, se puede enviar una lista de inicializadores simple (sin el nombre de la clase) como parámetro. El nombre completo de la clase se incluye en el siguiente ejemplo para mayor claridad.

Crear un `ProfiledPIDController` es casi idéntico a *creating a PIDController*. La única diferencia es la necesidad de suministrar un conjunto de *trapezoidal profile constraints*, que se reenviará automáticamente a las instancias `TrapezoidProfile` generadas internamente:

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))
```

Objetivo vs punto de ajuste

Una diferencia importante entre un `PIDController` estándar y un `ProfiledPIDController` es que el usuario no especifica directamente el *setpoint* real del lazo de control. Más bien, el usuario especifica una posición o estado *meta*, y el punto de ajuste para el controlador se calcula automáticamente a partir del perfil de movimiento generado entre el estado actual y el objetivo. Entonces, mientras que la llamada del lado del usuario parece casi idéntica:

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))
```

El valor de `meta` especificado (que puede ser un valor de posición o un `TrapezoidProfile.State`, si se desea una velocidad distinta de cero) *no* es necesariamente el *actual* del bucle, sino que es el *eventual* punto de ajuste una vez que finaliza el perfil generado.

Obtención / uso del punto de ajuste

Dado que el objetivo del `ProfiledPIDController` difiere del punto de ajuste, si a menudo es deseable sondear el punto de ajuste actual del controlador (por ejemplo, para obtener valores para usar con:ref:*feedforward <docs/software/advanced-controls/controllers/combining-feedforward-feedback:Using Feedforward Components with PID>*). Esto se puede hacer con el método `getSetpoint()`.

El punto de ajuste devuelto podría usarse como en el siguiente ejemplo:

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)
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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()
```

Ejemplo de uso completo

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;
21   private static double kI = 0.0;
22   private static double kD = 0.7;
23   private static double kS = 1.1;
24   private static double kG = 1.2;
25   private static double kV = 1.3;
26
27   private final Joystick m_joystick = new Joystick(1);
28   private final Encoder m_encoder = new Encoder(1, 2);
29   private final PWMSparkMax m_motor = new PWMSparkMax(1);
30
31   // Create a PID controller whose setpoint's change is subject to maximum
32   // velocity and acceleration constraints.
33   private final TrapezoidProfile.Constraints m_constraints =
34       new TrapezoidProfile.Constraints(kMaxVelocity, kMaxAcceleration);
35   private final ProfiledPIDController m_controller =
36       new ProfiledPIDController(kP, kI, kD, m_constraints, kDt);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
  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>
#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);
    }
```

(continúe en la próxima página)

(proviene de la página anterior)

```

    // 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

```
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)

```

(continúe en la próxima página)

(proviene de la página anterior)

```
        # 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 Bang-Bang Control with BangBangController

The *bang-bang control* algorithm is a control strategy that employs only two states: on (when the measurement is below the setpoint) and off (otherwise). This is roughly equivalent to a proportional loop with infinite gain.

This may initially seem like a poor control strategy, as PID loops are known to become unstable as the gains become large - and indeed, it is a *very bad idea to use a bang-bang controller on anything other than velocity control of a high-inertia mechanism*.

However, when controlling the velocity of high-inertia mechanisms under varying loads (like a shooter flywheel), a bang-bang controller can yield faster recovery time and thus better/more consistent performance than a proportional controller. Unlike an ordinary P loop, a bang-bang controller is *asymmetric* - that is, the controller turns on when the process variable is below the setpoint, and does nothing otherwise. This allows the control effort in the forward direction to be made as large as possible without risking destructive oscillations as the control loop tries to correct a resulting overshoot.

Asymmetric bang-bang control is provided in WPILib by the BangBangController class (Java, C++, Python).

Constructing a BangBangController

Since a bang-bang controller does not have any gains, it does not need any constructor arguments (one can optionally specify the controller tolerance used by `atSetpoint`, but it is not required).

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()
```

Using a BangBangController

Advertencia: Bang-bang control is an extremely aggressive algorithm that relies on response asymmetry to remain stable. Be *absolutely certain* that your motor controllers have been set to «coast mode» before attempting to control them with a bang-bang controller, or else the braking action will fight the controller and cause potentially destructive oscillation.

Using a bang-bang controller is easy:

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))
```

Combining Bang Bang Control with Feedforward

Like a PID controller, best results are obtained in conjunction with a *feedforward* controller that provides the necessary voltage to sustain the system output at the desired speed, so that the bang-bang controller is only responsible for rejecting disturbances. Since the bang-bang controller can *only* correct in the forward direction, however, it may be preferable to use a slightly conservative feedforward estimate to ensure that the shooter does not over-speed.

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 Generación de trayectoria y seguimiento con WPILib

Esta sección describe el soporte de WPILib para generar trayectorias de spline parametrizadas y seguir esas trayectorias con los accionamientos típicos de los robots de FRC®.

32.7.1 Generación de Trayectoria

WPILib contains classes that help generating trajectories. A trajectory is a smooth curve, with velocities and accelerations at each point along the curve, connecting two endpoints on the field. Generation and following of trajectories is incredibly useful for performing autonomous tasks. Instead of a simple autonomous routine – which involves moving forward, stopping, turning 90 degrees to the right, then moving forward – using trajectories allows for motion along a smooth curve. This has the advantage of speeding up autonomous routines, creating more time for other tasks; and when implemented well, makes autonomous navigation more accurate and precise.

Este artículo trata sobre cómo generar una trayectoria. Los próximos artículos de esta serie repasarán cómo seguir realmente la trayectoria generada. Hay algunas cosas que tu robot debe tener antes de sumergirse en el mundo de las trayectorias:

- Una forma de medir la posición y la velocidad de cada lado del robot. Un codificador es la mejor manera de hacerlo; sin embargo, otras opciones pueden incluir sensores ópticos de flujo, etc.
- Una forma de medir el ángulo o la velocidad angular del chasis del robot. Un giroscopio es la mejor manera de hacerlo. Aunque la tasa angular puede ser calculada usando las velocidades del codificador, este método NO es recomendado debido al lavado de las ruedas.

If you are looking for a simpler way to perform autonomous navigation, see *the section on driving to a distance*.

Splines

Un spline se refiere a un conjunto de curvas que se interpolan entre puntos. Piense en esto como conectar puntos, excepto con las curvas. WPILib apoya dos tipos de splines: cúbico sujetado por la hérmite y quístico sujetado por la hérmite.

- Hermite clamped cubic: Esta es la opción recomendada para la mayoría de los usuarios. La generación de trayectorias utilizando estas splines implica especificar las coordenadas (x, y) de todos los puntos, y los encabezamientos en los puntos de inicio y final. Los encabezamientos en los waypoints interiores se determinan automáticamente para asegurar una curvatura continua (tasa de cambio de rumbo) en todo momento.
- Hermite quintic: Esta es una opción más avanzada que requiere que el usuario especifique coordenadas (x, y) y encabezamientos para *todos* los waypoints. Esto debería utilizarse si no se está satisfecho con las trayectorias que están siendo generadas por las clamped cubic splines o si se desea un control más fino de los encabezamientos en los puntos interiores.

Los splines son usados como una herramienta para generar trayectorias; sin embargo, el propio spline no tiene ninguna información sobre las velocidades y aceleraciones. Por lo tanto, no se recomienda utilizar las clases de spline directamente. Para generar una trayectoria suave con velocidades y aceleraciones, se debe generar una *trayectoria*.

Creando la configuración de la trayectoria

Se debe crear una configuración para generar una trayectoria. La configuración contiene información sobre las restricciones especiales, la velocidad máxima, la aceleración máxima además de la velocidad inicial y la velocidad final. La configuración también contiene información sobre si la trayectoria debe ser invertida (el robot viaja hacia atrás a lo largo de los puntos de ruta). La clase `TrajectoryConfig` debe ser usada para construir una configuración. El constructor de esta clase toma dos argumentos, la velocidad máxima y la aceleración máxima. Los otros campos (`startVelocity`, `endVelocity`, `reversed`, `constraints`) son predeterminados a valores razonables (`0`, `0`, `false`, `{}`) cuando se crea el objeto. Si desea modificar los valores de cualquiera de estos campos, puede llamar a los siguientes métodos:

- `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++)

Nota: La propiedad «invertida» simplemente representa si el robot está viajando hacia atrás. Si especificas cuatro puntos, a, b, c y d, el robot seguirá viajando en el mismo orden a través de los puntos cuando la bandera «invertida» se ponga en «verdadero». Esto también significa que debe tener en cuenta la dirección del robot cuando proporcione los puntos de ruta. Por ejemplo, si su robot está de cara a la pared de la estación de la alianza y viaja hacia atrás a algún elemento del campo, el waypoint inicial debe tener una rotación de 180 grados.

Generando la trayectoria

El método usado para generar una trayectoria es `generateTrajectory(...)`. Hay cuatro sobrecargas para este método. Dos que usan clamped cubic splines y otras dos que usan quintic splines. Por cada tipo de spline, hay dos formas para construir una trayectoria. Los métodos más sencillos son las sobrecargas que aceptan objetos `Pose2d`.

Para los clamped cubic splines, este método acepta dos objetos `Pose2d`, uno para el punto de inicio y otro para el punto final. El método toma un vector de objetos «Translation2d» que representan los puntos de ruta interiores. Los encabezamientos de estos waypoints interiores se determinan automáticamente para asegurar una curvatura continua. Para las curvas quínticas, el método simplemente toma una lista de objetos `Pose2d`, con cada `Pose2d` representando un punto y un encabezamiento en la cancha.

La sobrecarga más compleja acepta «vectores de control» para los splines. Este método se utiliza cuando se generan trayectorias con Pathweaver, en las que se puede controlar la magnitud del vector tangente en cada punto. La clase `ControlVector` consiste de dos arreglos «dobles». Cada arreglo representa una dimensión (x o y), y sus elementos representan las derivadas en ese punto. Por ejemplo, el valor en el elemento 0 de la matriz `x` representa la coordenada x (0ª derivada), el valor en el elemento 1 representa la 1ª derivada en la dimensión x y así sucesivamente.

Al usarse clamped cubic splines, la longitud de la matriz debe ser 2 (0ª y 1ª derivadas), mientras que cuando se utilizan quintic splines, la longitud del conjunto debe ser 3 (0ª, 1ª y 2ª

derivadas). A menos que sepas exactamente lo que estás haciendo, el primer y más simple método es ALTAMENTE recomendado para generar trayectorias manualmente. (es decir, cuando no se utilizan los archivos JSON de Pathweaver).

Aquí hay un ejemplo de cómo generar una trayectoria usando clamped cubic splines para el juego de 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
    )
```

Nota: The Java code utilizes the Units utility, for easy unit conversions.

Nota: Generating a typical trajectory takes about 10 ms to 25 ms. This isn't long, but it's still highly recommended to generate all trajectories on startup (`robotInit`).

Concatenating Trajectories

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.

Advertencia: It is up to the user to ensure that the end of the initial and start of the appended trajectory match. It is also the user's responsibility to ensure that the start and end velocities of their trajectories match.

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)));
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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

```
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 Restricciones de trayectoria

En el *artículo previo*, es posible que haya notado que no se agregaron restricciones personalizadas al generar las trayectorias. Las restricciones personalizadas permiten a los usuarios imponer más restricciones a la velocidad y la aceleración en puntos a lo largo de la trayectoria según la ubicación y la curvatura.

Por ejemplo, una restricción personalizada puede mantener la velocidad de la trayectoria por debajo de un cierto umbral en una determinada región o ralentizar el robot cerca de los giros por motivos de estabilidad.

Restricciones proporcionadas por WPILib

WPILib incluye un conjunto de restricciones predefinidas que los usuarios pueden utilizar al generar trayectorias. La lista de restricciones proporcionadas por WPILib es la siguiente:

- `CentripetalAccelerationConstraint`: Limita la aceleración centrípeta del robot a medida que atraviesa la trayectoria. Esto puede ayudar a reducir la velocidad del robot en curvas cerradas.
- `DifferentialDriveKinematicsConstraint`: Limita la velocidad del robot en los giros de modo que ninguna rueda de un robot differential-drive supere una velocidad máxima especificada.
- `DifferentialDriveVoltageConstraint`: Limita la aceleración de un robot de differential drive de modo que ningún voltaje comandado supere un máximo especificado.
- `EllipticalRegionConstraint`: Impone una restricción solo en una región elíptica del campo.
- `MaxVelocityConstraint`: Impone una restricción de velocidad máxima. Esto se puede componer con `EllipticalRegionConstraint` o `` RectangularRegionConstraint`` para limitar la velocidad del robot solo en una región específica.
- `MecanumDriveKinematicsConstraint`: Limita la velocidad del robot en los giros de modo que ninguna rueda de un robot de mecanum-drive supere una velocidad máxima especificada.
- `RectangularRegionConstraint`: Impone una restricción solo en una región rectangular del campo.
- `SwerveDriveKinematicsConstraint`: Limita la velocidad del robot alrededor de los giros de manera que ninguna rueda de un robot de swerve-drive supere una velocidad máxima especificada.

Nota: El `` DifferentialDriveVoltageConstraint “” solo asegura que los comandos de voltaje teórico no superen el máximo especificado usando un *feedforward model*. Si el robot se desviara de la referencia durante el seguimiento, el voltaje ordenado puede ser más alto que el máximo especificado.

Crear una restricción personalizada

Los usuarios pueden crear su propia restricción implementando la `TrajectoryConstraint` .

JAVA

```
@Override
public double getMaxVelocityMetersPerSecond(Pose2d poseMeters, double␣
↪curvatureRadPerMeter,
                                            double velocityMetersPerSecond) {
  // code here
}

@Override
public MinMax getMinMaxAccelerationMetersPerSecondSq(Pose2d poseMeters,
```

(continúe en la próxima página)

(proviene de la página anterior)

```
                                                    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

```
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:
        ...
```

El método `MaxVelocity` debe devolver la velocidad máxima permitida para la pose, curvatura y velocidad original de la trayectoria dadas sin ninguna restricción. El método `MinMaxAcceleration` debe devolver la aceleración mínima y máxima permitida para la pose, curvatura y velocidad restringidas dadas.

See the source code (Java, C++) for the WPILib-provided constraints for more examples on how to write your own custom trajectory constraints.

32.7.3 Manipulando trayectorias

Una vez que se ha generado una trayectoria, puede recuperar información de ella utilizando ciertos métodos. Estos métodos serán útiles al escribir el código para seguir estas trayectorias.

Obteniendo la duración total de la trayectoria

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()
```

Una muestra de la trayectoria

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 estructura `Trajectory::Sample` tiene varias piezas de información acerca del punto de muestra:

- `t`: El tiempo transcurrido desde el inicio de la trayectoria hasta el punto de muestra.
- `velocity`: La velocidad en el punto de muestra.
- `acceleration`: La aceleracion en el punto de muestra.
- `` `pose ``: la pose (x, y, rumbo) en el punto de muestra.
- `curvature`: la curvatura (tasa de cambio de rumbo con respecto a la distancia a lo largo de la trayectoria) en el punto de muestra.

Nota: La velocidad angular en el punto de muestra se puede calcular multiplicando la velocidad por la curvatura.

Obteniendo todos los estados de la trayectoria (avanzado)

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 Transformando trayectorias

Las trayectorias se pueden transformar de un sistema de coordenadas a otro y moverse dentro de un sistema de coordenadas usando los métodos `relativeTo` y `transformBy`. Estos métodos son útiles para mover trayectorias dentro del espacio o re definir una trayectoria ya existente en otro marco de referencia.

Nota: Ninguno de estos métodos cambia la forma de la trayectoria original.

El metodo `relativeTo`

El método `relativeTo` es usado para redefinir una trayectoria ya existente en otro marco de referencia. Este método toma un argumento: pose, (objeto dirigido a `Pose2d`) que es definido respecto al sistema de coordenadas actual , que representa el origen de un nuevo sistema de coordenadas.

Por ejemplo, una trayectoria definida en el sistema de coordenadas A se puede redefinir en el sistema de coordenadas B, cuyo origen está en (3, 3, 30 grados) en el sistema de coordenadas A, utilizando el método `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)
```

En el diagrama de arriba, la trayectoria original (`aTrajectory` en el código de arriba) se ha definido en el sistema de coordenadas A, representado por los ejes negros. Los ejes rojos,

ubicados en (3, 3) y 30 ° con respecto al sistema de coordenadas original, representan el sistema de coordenadas B. Llamar a `relativeTo` en `aTrajectory` redefinirá todas las poses en la trayectoria para que sean relativas a sistema de coordenadas B (ejes rojos).

El método `transformBy`

El método `transformBy` se puede utilizar para mover (es decir, trasladar y rotar) una trayectoria dentro de un sistema de coordenadas. Este método toma un argumento: una transformación (a través de un objeto `Transform2d`) que mapea la posición inicial actual de la trayectoria a una posición inicial deseada de la misma trayectoria.

Por ejemplo, uno puede querer transformar una trayectoria que comienza en (2, 2, 30 grados) para que comience en (4, 4, 50 grados) usando el método `transformBy`.

JAVA

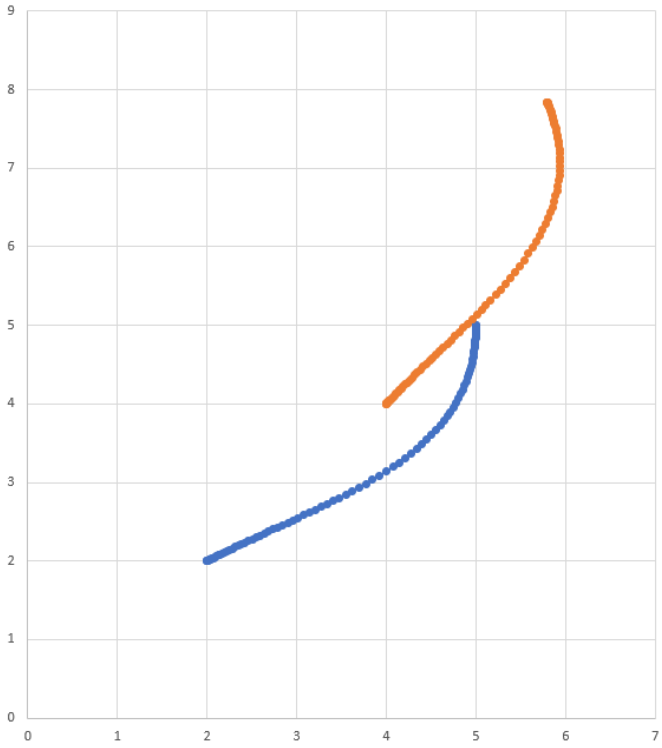
```
Transform2d transform = new Pose2d(4, 4, Rotation2d.fromDegrees(50)).minus(trajectory.
↪getInitialPose());
Trajectory newTrajectory = trajectory.transformBy(transform);
```

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)
```



En el diagrama de arriba, la trayectoria original, que comienza en (2, 2) y en 30 °, es visible en azul. Después de aplicar la transformación anterior, la ubicación inicial de la trayectoria resultante se cambia a (4, 4) a 50 °. La trayectoria resultante es visible en naranja.

32.7.5 Controlador ramsete

El controlador Ramsete es un rastreador de trayectoria integrado en WPILib. Este rastreador se puede utilizar para rastrear trayectorias con precisión con corrección de perturbaciones menores.

Construcción del objeto controlador Ramsete

El controlador Ramsete debe inicializarse con dos ganancias, a saber, `b` y `zeta`. Los valores más altos de `b` hacen que la convergencia sea más agresiva como un término proporcional, mientras que los valores más altos de `zeta` proporcionan más amortiguación en la respuesta. Estas ganancias del controlador solo dictan cómo el controlador generará velocidades ajustadas. NO afecta el seguimiento de la velocidad real del robot. Esto significa que estas ganancias del controlador son generalmente independientes del robot.

Nota: Las ganancias de `2.0` y `0.7``para ``b` y `zeta` se han probado repetidamente para producir resultados deseables cuando todas las unidades estaban en metros. Como tal, existe un constructor de argumento cero para `RamseteController` con ganancias predeterminadas a estos valores.

JAVA

```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++

```cpp
// 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

```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)
```

Obtener velocidades ajustadas

El controlador Ramsete devuelve «velocidades ajustadas» de modo que cuando el robot rastrea estas velocidades, alcanza con precisión el punto objetivo. El controlador debe actualizarse periódicamente con el nuevo objetivo. El objetivo comprende una pose deseada, una velocidad lineal deseada y una velocidad angular deseada. Además, la posición actual del robot también debe actualizarse periódicamente. El controlador usa estos cuatro argumentos para devolver la velocidad lineal y angular ajustada. Los usuarios deben ordenar a su robot estas velocidades lineales y angulares para lograr un seguimiento de trayectoria óptimo.

Nota: 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)
```

Estos cálculos deben realizarse en cada iteración de bucle, con una posición y un objetivo actualizados del robot.

Usando las velocidades ajustadas

Las velocidades ajustadas son del tipo `ChassisSpeeds`, que contiene un `vx` (velocidad lineal en la dirección de avance), una `vy` (velocidad lineal en la dirección lateral) y un `omega` (velocidad angular alrededor del centro del marco del robot). Debido a que el controlador Ramsete es un controlador para robots no holonómicos (robots que no pueden moverse hacia los lados), el objeto de velocidades ajustadas tiene una `vy` de cero.

Las velocidades ajustadas de regreso pueden ser convertidas a velocidades para usar utilizando las clases cinemáticas para el tipo de chasis. Por ejemplo, las velocidades ajustadas pueden convertirse en velocidades de derecha e izquierda para un manejo diferencial usando el objeto `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 en el Framework con base comando

Con motivos de facilitar a los usuarios, la clase `RamseteCommand` está contruida en WPILib. Para un tutorial completo en la implementación de un autónomo que siga un camino usando RamseteCommand, vea *Trajectory Tutorial*.

32.7.6 Controlador de impulsión holonómico

El controlador de propulsión holonómico es un rastreador de trayectoria para robots con transmisiones holonómicas (por ejemplo, swerve, mecanum, etc.). Esto se puede usar para rastrear trayectorias con precisión con corrección para pequeñas perturbaciones.

Construcción de un controlador de impulsión holonómico

El controlador de accionamiento holonómico debe instanciarse con 2 controladores PID y 1 controlador PID perfilado.

Nota: Para obtener más información sobre el control PID, consulte *Control PID en WPILib*.

Los 2 controladores PID son controladores que deben corregir el error en las direcciones X e Y relativas al campo, respectivamente. Por ejemplo, si los primeros 2 argumentos son `PIDController(1, 0, 0)` y `PIDController(1.2, 0, 0)` respectivamente, el controlador holonómico agregará un metro adicional por segundo en la x dirección por cada metro de error en la dirección x y agregará 1,2 metros adicionales por segundo en la dirección y por cada metro de error en la dirección y.

El último parámetro es un `ProfiledPIDController` para la rotación del robot. Debido a que la dinámica de rotación de un tren motriz holonómico está desacoplada del movimiento en las direcciones x,y, los usuarios pueden establecer referencias de rumbo personalizadas mientras siguen una trayectoria. Estas referencias de encabezado se perfilan de acuerdo con los parámetros establecidos en el `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.
```

Obtener velocidades ajustadas

El controlador de impulsión holonómico devuelve «velocidades ajustadas» de modo que cuando el robot sigue estas velocidades, alcanza con precisión el punto objetivo. El controlador debe actualizarse periódicamente con el nuevo objetivo. El objetivo se compone de una pose deseada, velocidad lineal y rumbo.

Nota: 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)
)
```

Usando velocidades ajustadas.

Las velocidades ajustadas son un tipo de `ChassisSpeeds`, que contiene un `vx` (velocidad lineal en dirección de avance), a `vy` (velocidad lineal en direcciones laterales), y `omega` (velocidad angular al rededor del centro del marco del robot).

La velocidad ajustada que nos da puede ser convertida en velocidad utilizable usando clases de cinemática para su chasis. En el código de ejemplo de abajo, podemos asumir un robot con swerve drive; de todas formas, el codigo cinemático es exactamente igual para un robot con mecanum drive usando `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)
```

Porque estos módulos swerve son aún velocidades y ángulos, se necesitará usar controladores PID para marcar estos ángulos y velocidades.

32.7.7 Solución de problemas

Solución a fallas totales

Hay algunas cosas que pueden causar una falla total en tu robot. La lista inferior cubre algunos de estos errores comunes.

- Mi robot no se mueve
  - ¿En verdad esta dando energía a sus motores?
  - Si a usted le esta apareciendo el mensaje `MalformedSplineException` en la driver station, vaya a la sección `MalformedSplineException`
  - ¿Su trayectoria es muy corta o está en unidades equivocadas?
- Mi robot gira alrededor para conducir la trayectoria en la otra dirección
  - ¿Son incorrectas las orientaciones inicial y final de tu trayectoria?
  - ¿El giróscopo de su robot se restablece al rumbo incorrecto?
  - *¿Tiene la bandera de reversa configurada incorrectamente?*
  - Si los ángulos de giro son positivos asegúrese de cambiarlos a negativo.
- Mi robot simplemente conduce en línea recta aunque debería girar.
  - ¿Su giróscopo está configurado correctamente y devuelve buenos datos?
  - ¿Está pasando su giróscopo rumbo a su objeto de odometría con las unidades correctas?
  - ¿Es correcto el ancho de su ruta? ¿Está en las unidades correctas?

- Apareció el mensaje `MalformedSplineException` impreso en la driver station y el robot no se mueve
  - *¿Tiene la bandera de reversa configurada incorrectamente?*
  - ¿Tiene dos coordenadas muy cercas con grados aproximadamente opuestos?
  - ¿Tiene dos puntos con las mismas(o casi las mismas) coordenadas?
- Mi robot conduce demasiado lejos.
  - ¿Están configuradas correctamente las conversiones de unidades de codificador?
  - ¿Están tus enconders conectados?
- Mi robot generalmente hace lo correcto, pero es un poco inexacto.
  - Diríjase a la siguiente sección

Solución de problemas de bajo rendimiento

Nota: Esta sección se ocupa principalmente de la resolución de problemas de rendimiento de seguimiento de trayectoria deficiente, como un metro de error, no fallas catastróficas como errores de compilación, robots que giran y van en la dirección incorrecta o `MalformedSplineExceptions`.

Nota: Esta sección está diseñada para robots de accionamiento diferencial, pero la mayoría de las ideas se pueden adaptar para desvío o mecanum.

El rendimiento deficiente del seguimiento de la trayectoria puede ser difícil de solucionar. Aunque el generador de trayectoria y el seguidor están pensados para ser fáciles de usar y con un rendimiento inmediato, hay situaciones en las que su robot no termina donde debería. El generador de trayectoria y los seguidores tienen muchas perillas para sintonizar y muchas partes móviles, por lo que puede ser difícil saber por dónde empezar, especialmente porque es difícil localizar la fuente de los problemas de trayectoria a partir del comportamiento general del robot.

Debido a que puede ser muy difícil de localizar la capa de el generador de trayectoria y los siguientes que se están portando mal, se recomienda un enfoque sistemático capa por capa para un rendimiento de seguimiento deficiente en general (por ejemplo, el robot se aleja unos pocos pies o más de veinte grados). Los pasos siguientes se enumeran en el orden en el que debe realizarlos; Es importante seguir este orden para poder aislar los efectos de los diferentes pasos entre sí.

Nota: The below examples put diagnostic values onto *NetworkTables*. The easiest way to graph these values is to *use Shuffleboard's graphing capabilities*.

Verifique la odometría

Si su odometría es mala, entonces su controlador Ramsete puede comportarse mal, porque modifica las velocidades objetivo de su robot en función de dónde cree que la odometría está el robot.

Nota: *Sending your robot pose and trajectory to field2d* can help verify that your robot is driving correctly relative to the robot trajectory.

---

1. Configure su código para que registre la posición del robot cada vez que se actualice la odometría.

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. Coloque una cinta métrica paralela a su robot y empuje su robot aproximadamente un metro a lo largo de la cinta métrica. Coloque una cinta métrica a lo largo del eje Y y

comience de nuevo, empujando su robot un metro a lo largo del eje X y un metro a lo largo del eje Y en un arco aproximado.

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.

Verificar Feedforward

Si sus feedforwards son malos, entonces los controladores P para cada lado del robot no seguirán tan bien, y su `DifferentialDriveVoltageConstraint` no limitará la aceleración de su robot con precisión. En general, queremos apagar los controladores P de la rueda para poder aislar y probar los feedforwards.

1. Primero, debemos configurar la desactivación del controlador P para cada rueda. Poner la ganancia `P` a 0 para cada controlador. En el ejemplo de «Comando Ramsete», pondrías «kPDriveVel» a 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. Next, we want to disable the Ramsete controller to make it easier to isolate our problematic behavior. To do so, simply call `setEnabled(false)` on the `RamseteController` passed into your `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. Finalmente, necesitamos registrar la velocidad de la rueda deseada y la velocidad real de la rueda (deberías poner las velocidades real y deseada en el mismo gráfico si estás usando Shuffleboard, o si tu software de gráficos tiene esa capacidad):

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. Ejecute el robot en una variedad de trayectorias (línea curva y recta), y compruebe si la velocidad real sigue la velocidad deseada mirando los gráficos de las NetworkTables.
5. If the desired and actual are off by *a lot* then you should check if the wheel diameter and `encoderEPR` you used for system identification were correct. If you've verified that your units and conversions are correct, then you should try recharacterizing on the same floor that you're testing on to see if you can get better data.

Verificación de la Ganancia P

Si has completado el paso anterior y el problema ha desaparecido, entonces es probable que tu problema se encuentre en uno de los siguientes pasos. En este paso vamos a verificar que los controladores P de tu rueda están bien ajustados. Si estás usando Java entonces queremos que apague Ramsete para que podamos ver nuestros controladores PF por su cuenta.

1. Debes reutilizar todo el código del paso anterior que registra la velocidad actual frente a la deseada (y el código que desactiva Ramsete, si usas Java), excepto que **la ganancia P debe volver a su valor anterior distinto de cero.**
2. Conduzca de nuevo el robot en distintas trayectorias, y revise que las gráficas reales se vean bien en comparación a las gráficas deseadas.

3. Si las gráficas no se ven bien (ej. la velocidad al momento es muy diferente a la deseada) entonces debería intentar afinar su ganancia P y re ejecutar sus trayectorias de prueba.

Compruebe las restricciones

Nota: Asegúrese de que su ganancia P sea distinta de cero en este paso y que aún tenga el código de registro añadido en los pasos anteriores. Si estás usando Java entonces deberías quitar el código para desactivar Ramsete.

Si su problema de exactitud persiste a través de todos los pasos anteriores, entonces podría tener un problema con sus limitaciones. A continuación hay una lista de síntomas que las diferentes restricciones disponibles exhibirán cuando no estén bien afinadas.

¡Prueba una restricción a la vez! Elimina las otras restricciones, ajusta la que te queda y repite el proceso para cada restricción que quieras usar. La siguiente lista de comprobación supone que sólo se utiliza una restricción a la vez.

- `DifferentialDriveVoltageConstraint`:
  - Si tu robot acelera muy lentamente, entonces es posible que el voltaje máximo para esta restricción sea demasiado bajo.
  - If your robot doesn't reach the end of the path then your system identification data may problematic.
- `DifferentialDriveKinematicsConstraint`:
  - Si tu robot termina en el rumbo equivocado, es posible que la velocidad máxima del lado de la transmisión sea demasiado baja, o que sea demasiado alta. La única manera de saberlo es ajustando la velocidad máxima y ver qué pasa.
- `CentripetalAccelerationConstraint`:
  - Si su robot termina en la dirección equivocada, entonces este podría ser el culpable. Si su robot no parece girar lo suficiente, entonces debe aumentar la aceleración centrípeta máxima, pero si parece girar en giros cerrados demasiado rápido, entonces debe disminuir la aceleración centrípeta máxima.

Comprobación de los puntos de ruta de la trayectoria

Es posible que su trayectoria en sí no sea muy manejable. Intente mover las coordenadas (y rumbos en las coordenadas, si corresponde) para reducir los giros bruscos.

32.8 Control basado en modelo y el estado del espacio con WPILib

Esta sección proporciona una introducción y describe el soporte de WPILib para el control del estado del espacio.

32.8.1 Introducción al Control del Estado-Espacio

Nota: Este artículo es de Ingeniería de Controles en FRC por Tyler Veness con permiso.

De PID a Control Basado en Modelo

Al ajustar los controladores PID, nos centramos en jugar con los parámetros del controlador relacionados con el presente, el pasado y el futuro *error* (términos P, I y D) en lugar de los estados subyacentes del sistema. Si bien este enfoque funciona en muchas situaciones, es una visión incompleta del mundo.

El control basado en modelos se enfoca en desarrollar un modelo preciso de el *system* (mecanismo) que estamos tratando de controlar. Estos modelos ayudan a informar *gains* tomadas de controladores basados en respuestas físicas del sistema, en vez de un arbitrario *gain* proporcional derivado mediante pruebas. Esto nos permite no solo predecir a futuro como reaccionará el sistema, y también probará nuestros controladores sin un robot que esté físicamente y ahorrar tiempo depurando errores simples.

Nota: 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.

If you've used WPILib's feedforward classes for `SimpleMotorFeedforward` or its sister classes, or used SysId to pick PID *gains* for you, you're already familiar with model-based control! The `kv` and `ka` *gains* can be used to describe how a motor (or arm, or drivetrain) will react to voltage. We can put these constants into standard state-space notation using WPILib's `LinearSystem`, something we will do in a later article.

Vocabulario

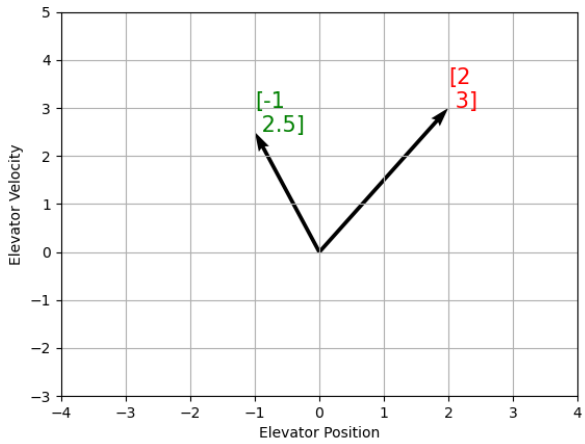
Para el vocabulario de fondo que se utilizará a lo largo de este artículo, consulte el *Glosario*.

Introducción a Algebra Lineal

Para una introducción corta e intuitiva a los conceptos básicos de Álgebra Lineal, recomendamos los capítulos 1 al 4 de Series 3Blue1Brown Esencia del algebra lineal (Vectores, ¿qué son esos?, Combinaciones lineales, lapso, y vectores base, Transformaciones lineales y matrices, y Multiplicación de matrices como composición).

¿Qué es Estado-Espacio?

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 $[\text{position}, \text{velocity}]$:



En ésta imagen, los vectores representando estados en estado-espacio son flechas. Desde ahora en éstos vectores serán representados simplemente por un punto en la punta del vector, pero recuerde que el resto del vector seguirá ahi.

Añadiendo al *estado*, *entradas* y *salidas* son representados como vectores. Dado que el mapeo de los estados actuales y las entradas al cambio de estado es un sistema de ecuaciones, es natural escribirlo en forma de matriz. Esta ecuación matricial se puede escribir en notación de estado-espacio.

¿Qué es la Notación de Estado-Espacio?

La notación de estado-espacio es un conjunto de ecuaciones matriciales que describen como el sistema evoluciona en el tiempo. Estas ecuaciones muestran el cambio en el estado $\dot{\mathbf{x}}$, y el *output* **y**, para combinaciones lineales del estado actual del vector **x** y *input* vector **u**.

El control del estado-espacio puede soportar sistemas de tiempo continuo y discreto. En el caso de tiempo continuo, la tasa de cambio del estado del sistema **x** es expresado como una combinación lineal el estado actual **x** y entrada **u**.

En contraste, los sistemas de tiempo discreto expresan el estado del sistema en nuestro siguiente paso del tiempo $\mathbf{x}_{k+1}$ basado en el estado actual $\mathbf{x}_k$ y la entrada $\mathbf{u}_k$, donde k es el paso de tiempo actual y $k+1$ es el siguiente paso de tiempo.

Tanto en la forma de tiempo continuo como en la discreta, el vector salida **y** se expresa como una combinación lineal del *estado* y *entrada* actual. En muchos casos, la salida es un subconjunto de estados del sistema, y no tiene contribución de la entrada actual.

Cuando modelamos el sistema, primero derivamos la representación del tiempo continuo poruqe las ecuaciones de movimiento son naturalmente escritas como la tasa de cambio de un

estado del sistema como una combinación lineal del actual estado y las entradas. Convertimos la representación a tiempo discreto en el robot porque actualizamos el sistema en pasos de tiempo discretos ahí en vez de continuamente.

Los siguientes dos conjuntos de ecuaciones son la forma estándar de la notación estado-espacio del tiempo continuo y tiempo discreto:

$$\text{Continuous: } \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$$\text{Discrete: } \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$
$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k$$

A	system matrix	**x**	state vector
B	input matrix	**u**	input vector
C	output matrix	**y**	output vector
D	feedthrough matrix		

Un sistema de estado-espacio con tiempo continuo puede ser convertido en un sistema de tiempo discreto mediante un proceso llamado discretización.

Nota: En la forma de tiempo discreto, el estado del sistema es mantenido constantemente mediante actualizaciones. Esto significa que solo podemos reaccionar a los disturbios tan rápido como nuestra estimación de estado sea actualizada. Actualizando nuestra estimación más rápido puede ayudar a mejorar la actuación, hasta un punto. La clase `Notifier` de WPILib se puede utilizar si se desean actualizaciones más rápidas que el bucle principal del robot.

Nota: Mientras que las matrices A, B, C y D de tiempo continuo y discreto de un sistema tengan los mismos nombres, no son equivalentes. Las matrices de tiempo continuo describen la tasa de cambio del estado, **x**, mientras que las matrices de tiempo discreto describen el estado del sistema en el siguiente paso de tiempo como una función del estado actual y la entrada.

Importante: El Sistema lineal de WPILib toma las matrices del sistema de tiempo continuo, y las convierte internamente al tiempo discreto donde sea necesario.

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.

Podemos convertir esta ecuación a notación estado-espacio. Podemos crear un sistema con un estado (velocidad), una *entrada* (voltaje), y una salida (velocidad). Recuerde que la primera derivada de la velocidad es la aceleración, podemos escribir nuestra ecuación como sigue, reemplazando velocidad con **x**, aceleración con $\dot{\mathbf{x}}$, y voltaje **V** con **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:

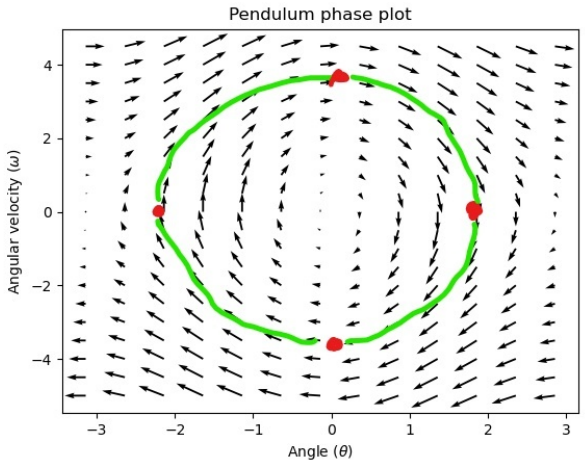
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \end{bmatrix} \mathbf{u}$$

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.

Visualizando las Respuestas del Estado-Espacio: Retrato de Fase

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 **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.

Para trazar una trayectoria potencial que un sistema puede tomar por el estado-espacio, escoja un punto para empezar y siga las flechas alrededor. En éste ejemplo, empezaremos en $[-2, 0]$. Desde aquí, la velocidad aumenta como se balancea verticalmente y empieza a reducir hasta que alcanza el extremo opuesto del columpio. Este ciclo de giro sobre el origen se repite indefinidamente.



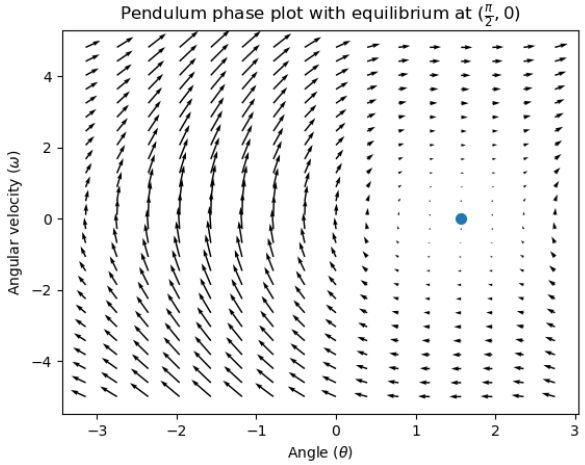
Note que cerca de los bordes del retrato de fase, el eje X se envuelve como una rotación de π radianes en sentido de las manecillas del reloj y una rotación de π radianes en sentido de las manecillas del reloj terminará con el mismo punto.

Para más en ecuaciones diferenciales y retratos de fase, vea el video de 3Blue1Brown Ecuaciones Diferenciales – hacen un gran trabajo de animación de la fase del péndulo alrededor del 15:30.

Visualizando Feedfoward

Este retrato de fase muestra las respuestas de «bucle abierto» del sistema, es decir, cómo reaccionará si dejamos que el estado evolucione naturalmente. Si queremos, digamos, equilibrar el péndulo horizontal (en $(\frac{\pi}{2}, 0)$ en el espacio de estados), necesitaríamos aplicar de alguna manera un control *input* para contrarrestar la tendencia de bucle abierto del péndulo a oscilar hacia abajo. Esto es lo que el feedforward está tratando de hacer: hacerlo de manera que nuestro retrato de fase tenga un equilibrio en la posición de *reference* (o punto de ajuste) en el espacio de estado.

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 $(\frac{\pi}{2}, 0)$:



Control de Retroalimentación

En el caso de los motores de control directo, con solo un modelo matemático y conocimiento de todos los estados actuales del sistema (ej. velocidad angular), podemos predecir todos los futuros estados dados en las entradas de futuros voltajes. Pero si el sistema es cambiado en alguna manera que no esté modelada por nuestras ecuaciones, como una carga o fricción inesperada, la velocidad angular del motor va a derivar del modelo por el tiempo. Para combatir esto, podemos dar al motor comandos correctivos usando el controlador de retroalimentación.

Un controlador de PID es una forma de control de retroalimentación. El control del estado-espacio usa la siguiente *ley de control*, donde $\mathbf{K}$ es algún controlador de matriz de *ganancia*, $\mathbf{r}$ es un estado de *referencia*, y $\mathbf{x}$ es el estado actual en estado-espacio. La diferencia entre estos dos vectores, $\mathbf{r} - \mathbf{x}$, es el *error*.

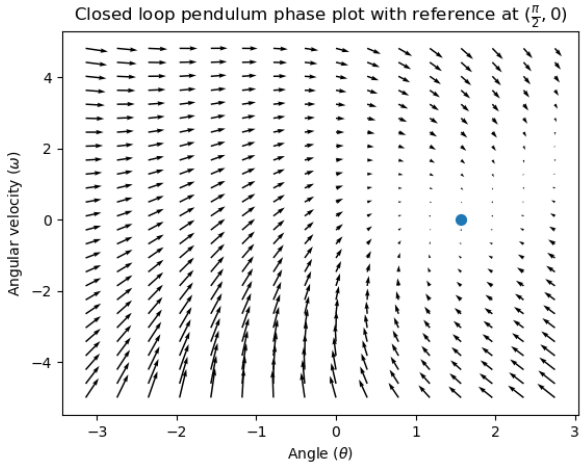
$$\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$$

Esta *ley de control* es un controlador proporcional para cada estado del sistema. Controladores proporcionales crean resortes definidos por el software que empujan nuestro estado del sistema hacia nuestro estado de referencia en el estado-espacio. En el caso de que el sistema

siendo controlado tenga estados de posición y velocidad, la *ley de control* de arriba como un controlador PD, que también intenta llevar al error de posición y velocidad a cero.

Veamos un ejemplo de esta ley de control en acción. Usaremos un sistema de péndulo desde abajo. donde el péndulo balanceante rodea al origen en estado-espacio. El caso donde **K** es la matriz cero (una matriz con todos cero) escogería las ganancias P y D de cero – no se aplicaría *entrada* de control, y el retrato de fase va a verse idéntico al de arriba.

Para añadir algo de retroalimentación, arbitrariamente escogemos **K** of [2, 2], donde nuestra *entrada* al péndulo es la aceleración angular. Esta K significará que por cada radian de posición *error*, la aceleración angular será 2 radianes por segundo cuadrado; similarmente, aceleramos por 2 radianes por segundo cuadrado por cada radian por segundo de *error*. Intente seguir una flecha desde algún punto en el estado-espacio hacia adentro – no importa las condiciones iniciales, el estado sentará en la *referencia* en vez de rodear sin fin con feedforward pura.



¿Pero cómo podemos elegir una matriz K optima de *ganancia* para nuestro sistema? Mientras podemos manualmente elegir las *ganancias* y simular la respuesta del sistema o sintonizarlo en el robot como controlador PID, la teoría de control moderna tiene mejor respuesta: el Regulador Lineal Cuadrático (LQR, en inglés)

El Regulador Lineal Cuadrático

Ya que el control basado en modelo significa que podemos predecir los futuros estados del sistema dada la condición inicial y las entradas de control futuras, podemos tomar una matriz matemática óptima de *ganancia* **K**. Para hacer esto, primero tenemos que definir que tan «bien» o «mal» **K** se vería. Hacemos esto sumando el cuadrado del error y entrada de control con el paso del tiempo, que nos da un número representado que tan «mal» será nuestra ley de control. Si minimizamos la suma, llegaremos a la ley de control óptima.

LQR: Definición

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*.

Nota: Las matrices de diseño **Q** y **R** de LQR no necesitan discretización, pero **K** calcula por tiempo continuo y discreto *systems* será diferente.

LQR: Afinación

Como los controladores PID pueden ser afinados ajustando sus ganancias, también queremos cambiar como la ley de control balancea el error y la entrada. Por ejemplo, la nave puede querer minimizar el combustible que expande para alcanzar una referencia dada, mientras que un brazo robótico a alta velocidad puede necesitar reaccionar rápidamente a las alteraciones.

Podemos pesar el esfuerzo del error y control de nuestro LQR con las matrices **Q** y **R**. En nuestra función de costo (que describe que tan «mal» nuestra ley de control actuará), **Q** y **R** pesa nuestro error y entrada de control relativo a cada uno. En nuestro ejemplo del cohete de arriba, podemos usar **Q** con números relativamente pequeños para mostrar que no queremos penalizar altamente el error, mientras

Con WPILib, las clases de LQR toman el vector de las excursiones de estado máximo deseado y esfuerzos de control y los convierte internamente a unas matrices completas Q y R con la regla de Bryson. Usualmente usamos el caso menor de **q** y **r** para referirse a esos vectores, y **Q** and **R** para referirse a las 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.

Por ejemplo, podríamos usar el siguiente Q y R para un sistema de elevador con estados de posición y velocidad.

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),
```

(continúe en la próxima página)

(proviene de la página anterior)

```
      // 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: ejemplo de aplicación

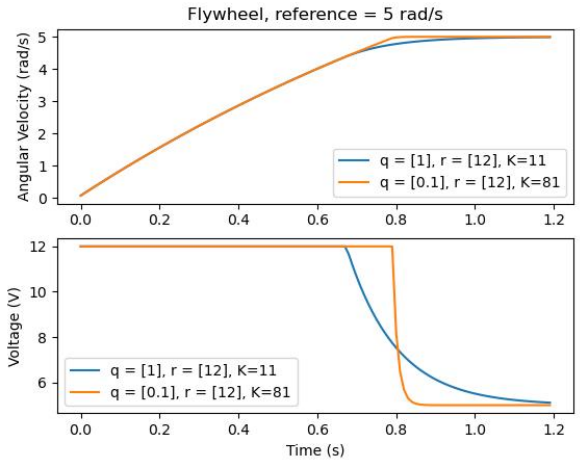
Let's apply a Linear-Quadratic Regulator to a real-world example. Say we have a flywheel velocity system determined through system identification to have $K_v = 1 \frac{\text{volts}}{\text{radian per second}}$ and $K_a = 1.5 \frac{\text{volts}}{\text{radian per second squared}}$. Using the flywheel example above, we have the following linear *system*:

$$\mathbf{x} = \left[\frac{-K_v}{K_a}\right] v + \left[\frac{1}{K_a}\right] V$$

Arbitrariamente escogemos la excursión del estado deseado (error máximo) de $q = [0.1 \text{ rad/seg}]$, y un $\mathbf{r}$ de $[12 \text{ volts}]$. Después de la discretización con paso de 20ms, encontramos una *ganancia* de $\mathbf{K} = 81$. Esta *ganancia* K actúa como el componente proporcional del bucle PID en la velocidad del volante.

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**.

La siguiente gráfica muestra la velocidad angular del volante y el voltaje aplicado en el tiempo con dos diferentes *ganancia*s. Podemos ver como una *ganancia* más alta puede hacer que el sistema alcance la referencia más rápido (a t = 0.8 segundos), mientras dejamos nuestro motor saturado a 12V por más tiempo. Es igual a incrementar la ganancia P de un controlador PID por un factor de ~8x.



LQR y Medición de Compensación de Latencia

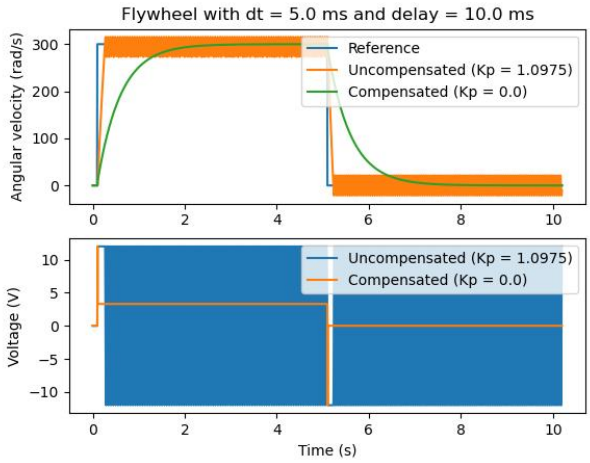
La mayoría del tiempo, nuestros sensores tienen un retraso asociado con sus medidas. Por ejemplo el controlador de motor SPARK MAX sobre CAN puede tener 30ms de retraso asociado con las mediciones de velocidad.

Este retraso significa que nuestro controlador de retroalimentación va a estar generando comandos de voltaje basados en estimaciones de estado del pasado. Regularmente tiene el efecto de introducir inestabilidad y oscilaciones en el sistema, como se muestra en la gráfica de abajo.

However, we can model our controller to control where the system's *state* is delayed into the future. This will reduce the LQR's *gain* matrix **K**, trading off controller performance for stability. The below formula, which adjusts the *gain* matrix to account for delay, is also used in system identification.

$$\mathbf{K_{compensated}} = \mathbf{K} \cdot (\mathbf{A} - \mathbf{BK})^{\text{delay}/dt}$$

Multiplicando **K** por $\mathbf{A} - \mathbf{BK}$ esencialmente avanza las ganancias por un paso de tiempo. En este caso, multiplicamos por $(\mathbf{A} - \mathbf{BK})^{\text{delay}/dt}$ para avanzar las ganancias por el retraso de las mediciones.



Nota: Esto puede tener efecto de reducir **K** a cero, de manera efectiva inhabilitando el control de retroalimentación.

Nota: El controlador de motor SPARK MAX utiliza un filtro FIR de 40 tomas con un retraso de 19,5 ms, y los marcos de estado se envían por defecto cada 20 ms.

El siguiente código muestra cómo ajustar la ganancia K del controlador LQR para los retardos de entrada del sensor:

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++

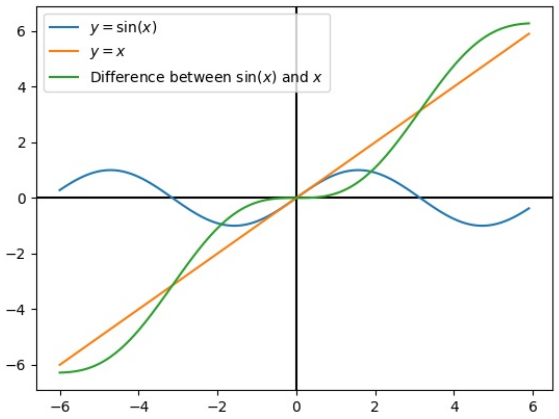
```
// Adjust our LQR's controller for 25 ms of sensor input delay. We
// provide the linear system, discretization timestep, and the sensor
// input delay as arguments.
controller.LatencyCompensate(elevatorSystem, 20_ms, 25_ms);
```

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)
```

Linealización

La linealización es una herramienta usada para aproximar las funciones no lineales y el sistema de estado-espacio usando las lineales. En el espacio bidimensional, las funciones lineales son líneas derechas mientras las funciones no lineales son curvas. Un ejemplo común de la función no lineal y correspondiente aproximación lineal es $y = \sin x$. Esta función puede estar aproximada por $y = x$ cercano a cero. Esta aproximación es precisa mientras esté cerca de $x = 0$, pero pierde la precisión mientras más nos alejamos del punto de linealización. Por ejemplo, la aproximación $\sin x \approx x$ es precisa a 0.02 - 0.5 radianes de $y = 0$, pero rápidamente pierde la precisión después de eso. La siguiente imagen, podemos ver:math:*y =sin{x}*, $y = x$ y la diferencia entre la aproximación y el valor real de $\sin x$ en x.



Podemos también linealizar el sistema estado-espacio con dinámicas no lineales. Hacemos esto eligiendo un punto $\mathbf{x}$ en el estado-espacio y usando esto como entrada para nuestras funciones no lineales. Como el ejemplo de arriba, esto funciona bien para los estados cerca del punto sobre el cual se linealiza el sistema, pero podemos divergir rápidamente lejos de ese estado.

32.8.2 Tutorial de Controlador de Estado-Espacio

Nota: Antes de seguir este tutorial, se recomienda que los lectores hayan leído *Introducción al Control del Estado-Espacio*.

La meta de este tutorial es proveer «de principio a fin» instrucciones para implementar el controlador de estado-espacio para un volante. Siguiendo este tutorial, los lectores podrán aprender a hacer:

1. Create an accurate state-space model of a flywheel using *system identification* or *CAD* software.
2. Implementar un Filtro Kalman para filtrar las mediciones de velocidad del encoder sin retraso.
3. Implementar un controlador de retroalimentación *LQR* que, cuando lo combinamos con uno de feedforward basado en modelo, generará *inputs 1* de voltaje para manejar el volante a una *reference*.

Este tutorial tiene el propósito de ser alcanzable para los equipos sin experiencia en programación o buen manejo de ella. Mientras que la librería de WPILib ofrece cierta flexibilidad en el modo que las características del control de estado-espacio son implementadas, siguen de cerca la implementación descrita en este tutorial debe dar a los equipos una estructura base que pueda ser reusada para sistemas variados de estado-espacio.

The full example is available in the state-space flywheel (Java/C++/Python) and state-space flywheel system identification (Java/C++/Python) example projects.

¿Por qué usar Control de Estado-Espacio?

Porque el control de estado-espacio se enfoca en crear un modelo preciso de nuestro sistema, podemos predecir de manera certera como el *modelo* responderá para controlar *entradas*. Esto nos permite simular nuestros mecanismos sin acceso al robot físico, como bien escoger fácilmente las *ganancias* que sabemos que trabajarán bien. Teniendo un modelo también nos permite crear filtros sin retrasos, como los Filtros Kalman, para filtrar óptimamente las lecturas del sensor.

Modelando Nuestro Flywheel

Recuerde que el sistema continuo de estado-espacio son modelados usando el siguiente sistema de ecuaciones:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

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*.

Let's use this system of equations to model our flywheel in two different ways. We'll first model it using *system identification* using the SysId toolsuite, and then model it based on the motor and flywheel's *moment of inertia*.

El primer paso para construir nuestro sistema estado-espacio es elegir nuestros estados del sistema. Podemos elegir cualquiera que queramos como estado – podemos elegir estados completamente diferentes si queremos – pero ayudará si elegimos estados que son importantes.

Podemos incluir *estados ocultos* en nuestro estado (como la velocidad del elevador si solo podemos medir su posición) y dejar al Filtro Kalman estimar sus valores. Recuerda que los estados que elegimos serán conducidos hacia sus respectivas *referencias* por el controlador de retroalimentación (normalmente el *Regulador Lineal Cuadrático* ya que es el óptimo).

Para nuestro volante, solo nos importa un estado: su velocidad. Mientras que podamos elegir también modelar su aceleración, incluir este estado no es necesario para nuestro sistema.

A continuación, identificamos los *inputs* a nuestro sistema. Las entradas se pueden considerar como cosas que podemos poner «en» nuestro sistema para cambiar su estado. En el caso del flywheel (y muchos otros mecanismos de articulación simple en FRC®), tenemos solo una entrada: voltaje aplicado al motor. Al elegir el voltaje como nuestra entrada (sobre algo como el ciclo de trabajo del motor), podemos compensar la caída del voltaje de la batería a medida que aumenta la carga de la batería.

Un sistema de tiempo continuo de estado-espacio escribe *x-dot*, o la velocidad instantánea de cambio del *sistema*estado del sistema, proporcional al *estado* actual y las *entradas*. Porque nuestro estado es velocidad angular, $\mathbf{x}$ va a ser la aceleración angular del volante.

Después, modelaremos nuestro volante como un sistema de tiempo continuo de estado-espacio. `LinearSystem` de WPILib convertirá este tiempo discreto internamente. Repase *notación de estado-espacio* para más sistemas de tiempo continuo y discreto.

Modelando con Identificación de Sistema

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 $\mathbf{a}$.

$$V = kV \cdot \mathbf{v} + kA \cdot \mathbf{a}$$
$$\mathbf{a} = \mathbf{v} = \left[\frac{-kV}{kA}\right] v + \left[\frac{1}{kA}\right] V$$

Donde $\mathbf{v}$ es la velocidad del volante, $\mathbf{a}$ y $\mathbf{v}$ son la aceleración del volante, y V es el voltaje. Reescribiendo esto como la convención estándar de $\mathbf{x}$ para el vector del estado y $\mathbf{u}$ para el vector de entrada, encontramos:

$$\mathbf{x} = \left[\frac{-kV}{kA}\right] \mathbf{x} + \left[\frac{1}{kA}\right] \mathbf{u}$$

La segunda parte de la notación de estado-espacio relata que el *estado* actual del sistema y *entradas* a la salida. En caso de un volante, nuestro vector de salida $\mathbf{y}$ (o cosas que pueda medir un sensor) es la velocidad de nuestro volante, que también resulta ser un elemento de nuestro vector de *estado* $\mathbf{x}$. Por lo tanto, nuestra matriz de salida es $\mathbf{C} = \begin{bmatrix}1\end{bmatrix}$, y nuestra matriz de sistema de alimentación es $\mathbf{D} = \begin{bmatrix}0\end{bmatrix}$. Escribiendo esto en notación de estado-espacio con tiempo continuo produce lo siguiente.

$$\mathbf{x} = \left[\frac{-kV}{kA}\right] \mathbf{x} + \left[\frac{1}{kA}\right] \mathbf{u}$$
$$\mathbf{y} = \begin{bmatrix}1\end{bmatrix} \mathbf{x} + \begin{bmatrix}0\end{bmatrix} \mathbf{u}$$

La clase `LinearSystem` contiene métodos para crear fácilmente sistemas de estado-espacio identificados usando *system identification*. Este ejemplo muestra un modelo de volante con un kV de 0.023 y 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:
        #
```

(continúe en la próxima página)

(proviene de la página anterior)

```
        # 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
            )
        )
```

Modelando Usando el Volante con Momento de Inercia y Engranaje

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).

Nota: Para las clases de estado-espacio de WIPLib, los engranes son escritos como salida en vez de entrada – es esto, si el volante gira más lento que los motores, este número debe ser mayor que uno.

Nota: La clase LinearSystem de C++ usa *la Libreria de Unidades de C++* para prevenir mezla de unidades y afirmar dimensionalmente.

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
```

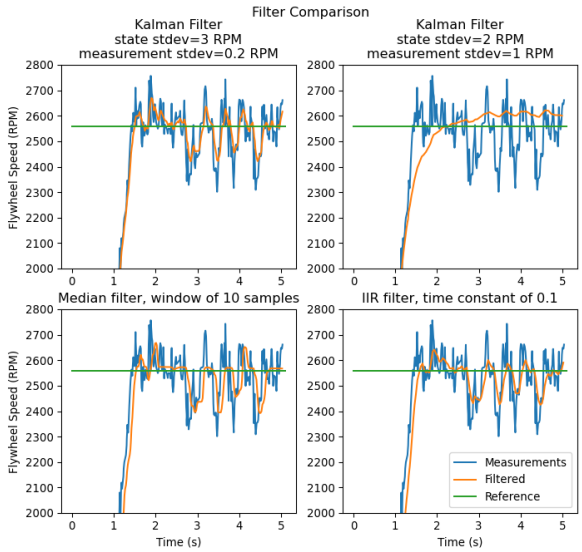
```
        # The plant holds a state-space model of our flywheel. This system has the␣
↪following properties:
        #
        # States: [velocity], in radians per second.
        # Inputs (what we can "put in"): [voltage], in volts.
        # Outputs (what we can measure): [velocity], in radians per second.
        self.flywheelPlant = wpimath.system.plant.LinearSystemId.flywheelSystem(
            wpimath.system.plant.DCMotor.NEO(2),
            kFlywheelMomentOfInertia,
            kFlywheelGearing,
        )
```

Filtros Kalman: Observando el Estado del Flywheel

Los filtros Kalman son usados para filtrar nuestras mediciones de velocidad usando nuestro modelo de estado-espacio para generar un estado estimado $\hat{\mathbf{x}}$. Como nuestro modelo de volante es lineal, podemos usar un filtro Kalman para estimar la velocidad del volante. EL filtro Kalman de WPILib toma un `LinearSystem` (que encontramos arriba), a lo largo con derivaciones estándar de mediciones de modelo y sensores. Podemos ajustar que tan «fluido» es nuestra estimación de estado es por ajustar estas cargas. Las desviaciones estándar de estado más grandes harán que el filtro «desconfíe» de nuestra estimación estatal y favorezca

más las nuevas mediciones, mientras que las desviaciones estándar de las mediciones más grandes harán lo contrario.

En el caso de un volante comenzamos con una desviación estándar de estado de 3 rad/s y una desviación estándar de medición de 0.01 rad/s. Estos valores son a elección del usuario – estos pesos produjeron un filtro que era tolerante a algo de ruido pero cuya estimación de estado reaccionó rápidamente a perturbaciones externas para *un* volante de inercia – y deben ajustarse para crear un filtro que se comporte bien para su volante específico. Graficar estados, medidas, entradas, referencias y salidas a lo largo del tiempo es una excelente forma visual de ajustar los filtros de Kalman.



La gráfica de arriba muestra 2 diferentes ajustes del filtro Kalman, así como *filtro IIR con un polo* y un *Filtro de Mediana*. Estos datos fueron recolectados con un lanzador arriba de ~5 segundos, y cuatro bolas pasaron a través del lanzador (como se ve en las cuatro caídas de velocidad). Si bien no existen reglas estrictas para elegir un buen estado y medir las desviaciones estándar, en general deben ajustarse para confiar en el modelo lo suficiente como para rechazar el ruido y reaccionar rápidamente a las perturbaciones externas.

Dado que el controlador de retroalimentación calcula el error utilizando el **:término:`x-hat`** estimado por el filtro Kalman, el controlador reaccionará a las perturbaciones sólo con la rapidez con la que cambie la estimación de estado del filtro. En el gráfico anterior, el gráfico superior de la izquierda (con una desviación estándar del estado de 3,0 y una desviación estándar de la medición de 0,2) produjo un filtro que reaccionó rápidamente a las perturbaciones mientras rechazaba el ruido, mientras que el gráfico superior de la derecha muestra un filtro que apenas se vio afectado por las caídas de velocidad.

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 <frc/estimator/KalmanFilter.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,
        )
```

Porque los filtros Kalman usan nuestros modelos de estado-espacio en el *Predecir paso*, es importante que nuestro modelo es tan preciso como es posible. Una manera de verificar esto es registrar el voltaje de entrada del volante y la velocidad en el tiempo, y vuelve a reproducir esta información llamando solo `predict` en el filtro Kalman. Después, las ganancias de kV y kA (o el momento de inercia y otras constantes) pueden ser justadas hasta que el modelo se acerque a igualar los datos registrados.

Reguladores Lineales Cuadráticos y Feedforward de Inversión de Planta

El Regulador Lineal Cuadrático encuentra un controlador de retroalimentación para manejar nuestro *sistema* de volante a su *referencia*. Porque nuestro volante solo tiene un estado, la ley de control tomada por nuestro LQR va a estar en la forma $\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$ donde $\mathbf{K}$ es una matriz de 1x1; en otras palabras, la ley de control tomada por el LQR es simplemente un controlador proporcional, o un controlador PID con solo ganancia P. Esta ganancia es elegida por nuestro LQR en función de la excursión de estado y los esfuerzos de control que la pasamos. Más de la afinación de controladores LQR puede encontrarse en *ejemplo de aplicación de LQR*.

Mucho como `SimpleMotorFeedforward` puede ser usado para generar la entrada de voltaje del feedfoward dadas las constantes kS, kV, kA, la clase Feedforward de Inversión de Planta genera entradas de voltaje de *feedforward* dado un sistema de estado-espacio. Los comandos de voltaje generados por la clase `LinearSystemLoop` son la suma de las entradas de feedfoward y de retroaliemntación.

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
      // using notifiers.
      20_ms};

  // The state-space loop combines a controller, observer, feedforward and plant
  // for easy control.
  frc::LinearSystemLoop<1, 1, 1> m_loop{m_flywheelPlant, m_controller,
                                        m_observer, 12_V, 20_ms};
```

Python

```
        # A LQR uses feedback to create voltage commands.
        self.controller = wpimath.controller.LinearQuadraticRegulator_1_1(
            self.flywheelPlant,
            [8],  # qelms. Velocity error tolerance, in radians per second. Decrease
            # this to more heavily penalize state excursion, or make the controller behave more
            # aggressively.
            [12],  # 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.
        )
```

Trayendo todo Junto: LinearSystemLoop

LinearSystemLoop combina nuestro sistema, controlador, y observador que creamos antes. El constructor que se muestra también crea una instancia `PlantInversionFeedforward`.

Java

```
  // The state-space loop combines a controller, observer, feedforward and plant for easy control.
  private final LinearSystemLoop<N1, N1, N1> m_loop =
      new LinearSystemLoop<>(m_flywheelPlant, m_controller, m_observer, 12.0, 0.020);
```

C++

```
#include <frc/system/LinearSystemLoop.h>
```

```
  // The state-space loop combines a controller, observer, feedforward and plant
  // for easy control.
  frc::LinearSystemLoop<1, 1, 1> m_loop{m_flywheelPlant, m_controller,
                                        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
        )
```

Una vez que tenemos nuestro `LinearSystemLoop`, lo único que queda por hacer es ejecutarlo. Para hacer eso, actualizaremos periódicamente nuestro filtro Kalman con nuestras nuevas

medidas de velocidad del codificador y le aplicaremos nuevos comandos de voltaje. Para hacer eso, primero establecemos el *referencia*, luego `corregir` con la velocidad actual del volante, `predecir` el filtro de Kalman en el siguiente paso de tiempo y aplicar las entradas generadas usando `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++

```
#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>
```

```
  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

```
    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.
        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;
```

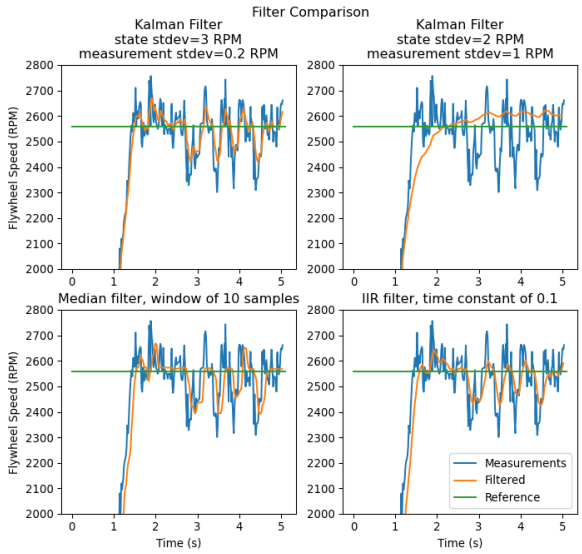
PYTHON

```
error = lqr.R() - x
error[0] = wpimath.angleModulus(error[0])
u = lqr.K() * error
```

32.8.3 Observadores de estado y filtros de Kalman

Los observadores de estados combinan información sobre el comportamiento de un sistema y mediciones externas para estimar el verdadero *estado* del sistema. Un observador común utilizado para sistemas lineales es el filtro de Kalman. Los filtros de Kalman son ventajosos sobre otros *filtros* ya que fusionan las mediciones de uno o más sensores con un modelo de espacio de estado del sistema para estimar de manera óptima el estado de un sistema.

Esta imagen muestra las mediciones de la velocidad del volante a lo largo del tiempo, ejecutadas a través de una variedad de filtros diferentes. Tenga en cuenta que un filtro de Kalman bien ajustado no muestra ningún retraso de medición durante el giro del volante, mientras que sigue rechazando datos ruidosos y reacciona rápidamente a las perturbaciones cuando las bolas pasan a través de él. Puede encontrar más información sobre filtros en *sección de filtros* .

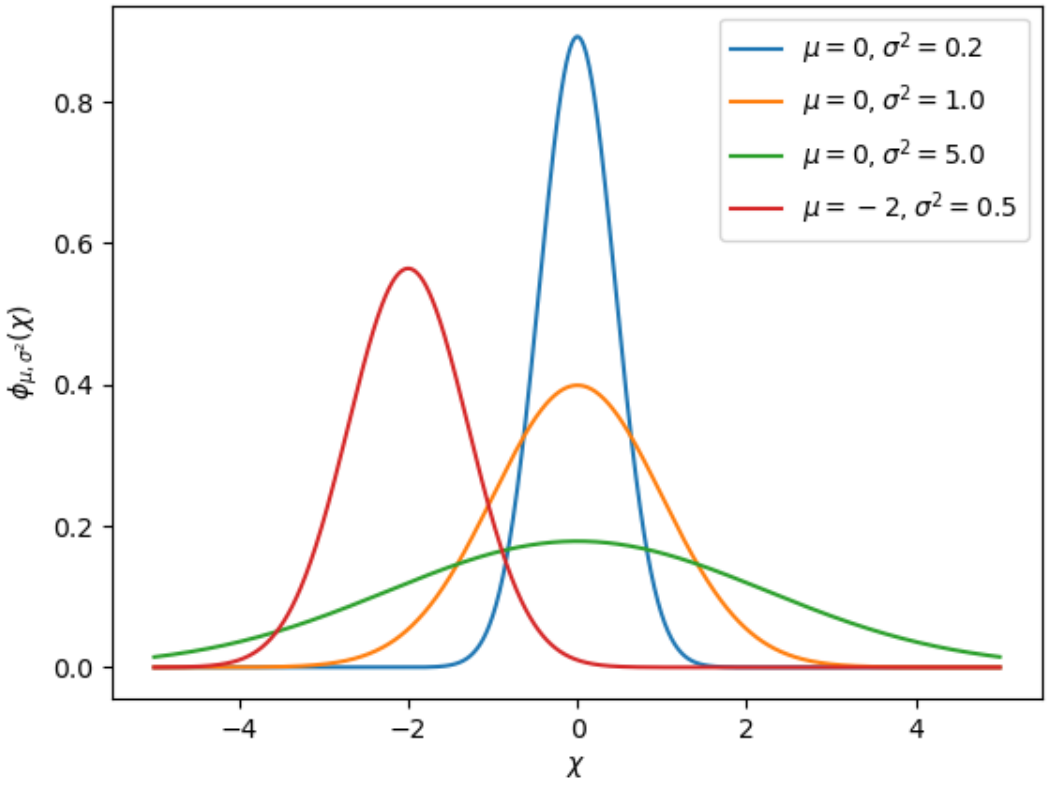


Funciones gaussianas

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*.

La idea de varianza y covarianza es fundamental para la función de un filtro de Kalman. La covarianza es una medida de cómo se correlacionan dos variables aleatorias. En un sistema con un solo estado, la matriz de covarianza es simplemente $mathbf\ textcov(x_1, x_1)$, o una matriz que contiene la varianza:math:*mathbf {text {var } (x_1)} `del estado: matemáticas:* x_1`. La magnitud de esta varianza es el cuadrado de la desviación estándar de la función gaussiana que describe la estimación del estado actual. Los valores relativamente grandes de covarianza pueden indicar datos ruidosos, mientras que las pequeñas covarianzas pueden indicar que el filtro tiene más confianza en su estimación. Recuerde que los valores «grandes» y «pequeños» para la varianza o covarianza son relativos a la unidad base que se está utilizando, por ejemplo, si $mathbf x_1$ se midió en metros, cov($\mathbf{x_1}, \mathbf{x_1}$) estaría en metros al cuadrado.

[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

Las matrices de covarianza se escriben de la siguiente forma:

$$
\begin{aligned}
&mathbf\ Sigma = \ beginbmatrix\ textcov(x_1, x_1) \\
&textcov(x_1, x_2)\ ldots \\
&textcov(x_1, x_n) \\
&textcov(x_2, x_1)\ textcov(x_2, x_2) \\
&\qquad ldots\ textcov(x_1, x_n) \\
&\qquad vdots\ vdots \\
&\qquad ddots\ vdots \\
&textcov(x_n, x_1)\ textcov(x_n, x_2) \\
&\qquad ldots\ textcov(x_n, x_n) \\
&\quad endbmatrix
\end{aligned}
$$

Filtros de Kalman

Importante: Es importante desarrollar una intuición de lo que realmente hace un filtro de Kalman. El libro *Kalman and Bayesian Filters in Python de Roger Labbe <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>* __ proporciona una excelente introducción visual e interactiva a los filtros bayesianos. Los filtros de Kalman en WPILib usan álgebra lineal para gentrificar las matemáticas, pero las ideas son similares al caso unidimensional. Sugerimos leer el Capítulo 4 para hacerse una idea de lo que hacen estos filtros.

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).

A continuación se muestran las ecuaciones de un filtro de Kalman de tiempo discreto:

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

La estimación del estado **x**, junto con **P**, describe la media y la covarianza de la función gaussiana que describe la estimación de nuestro filtro del estado verdadero del sistema.

Matrices de covarianza de proceso y medición de ruido

Las matrices de covarianza de ruido de proceso y medición **Q** y **R** describen la varianza de cada uno de nuestros estados y medidas. Recuerde que para una función gaussiana, la varianza es el cuadrado de la desviación estándar de la función. En un WPILib, Q y R son matrices diagonales cuyas diagonales contienen sus respectivas varianzas. Por ejemplo, un filtro de Kalman con estados: matemáticas *begin{bmatrix}text{position} \ text{velocity} end{bmatrix}* y medidas $\begin{bmatrix}\text{position}\end{bmatrix}$ con desviaciones de estado estándar $\begin{bmatrix}0.1\\1.0\end{bmatrix}$ y desviación estándar de medición $\begin{bmatrix}0.01\end{bmatrix}$ tendría las siguientes **Q** y **R** matrices:

$$Q = \begin{bmatrix}0.01 & 0\\0 & 1.0\end{bmatrix}, R = \begin{bmatrix}0.0001\end{bmatrix}$$

Matriz de covarianza de errores

La matriz de covarianza de error **P** describe la covarianza de la estimación del estado $\hat{\mathbf{x}}$. De manera informal, **P** describe nuestra certeza sobre el *estado* estimado. Si **P** es grande, nuestra incertidumbre sobre el estado real es grande. A la inversa, a **P** con elementos más pequeños implicaría menos incertidumbre sobre nuestro verdadero estado.

A medida que proyectamos el modelo hacia adelante, :math:mathbf{P}` aumenta a medida que disminuye nuestra certeza sobre el verdadero estado del sistema.

Predecir paso

En la predicción, nuestra estimación de estado se actualiza de acuerdo con la dinámica del sistema lineal $\mathbf{x} = \mathbf{Ax} + \mathbf{Bu}$. Además, nuestra covarianza de error **P** aumenta por la matriz de covarianza de ruido del proceso **Q**. Los valores más grandes de **Q** harán que nuestra covarianza de error **P** crezca más rápidamente. Esto **P** se usa en el paso de corrección para ponderar el modelo y las medidas.

Paso correcto

En el paso correcto, nuestra estimación de estado se actualiza para incluir nueva información de medición. Esta nueva información se pondera contra la estimación del estado $\hat{\mathbf{x}}$ por la ganancia de Kalman **K**. Los valores grandes de **K** ponderan más las medidas entrantes, mientras que los valores más pequeños de **K** ponderan más nuestra predicción de estado. Porque **K** está relacionado con **P**, los valores más grandes de **P** aumentarán **K** y medidas más pesadas. Si, por ejemplo, se predice un filtro para una duración prolongada, el grande **P** ponderaría mucho la nueva información.

Finalmente, la covarianza de error **P** disminuye para aumentar nuestra confianza en la estimación del estado.

Ajuste de los filtros de Kalman

Los constructores de las clases de filtros Kalman de WPILib toman un sistema lineal, un vector de desviaciones estándar de ruido de proceso y desviaciones estándar de ruido de medición. Éstos se convierten en matrices **Q** y **R** llenando las diagonales con el cuadrado de las desviaciones estándar, o varianzas, de cada estado o medida. Al disminuir la desviación estándar de un estado (y por lo tanto su entrada correspondiente en **Q**), el filtro desconfiará más de las mediciones entrantes. De manera similar, aumentar la desviación estándar de un estado confiará más en las mediciones entrantes. Lo mismo se aplica a las desviaciones estándar de la medición: disminuir una entrada hará que el filtro confíe más en la medición entrante para el estado correspondiente, mientras que aumentarla disminuirá la confianza en la medición.

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++

```
5  #include <numbers>
6
7  #include <frc/DriverStation.h>
8  #include <frc/Encoder.h>
9  #include <frc/TimedRobot.h>
10 #include <frc/XboxController.h>
11 #include <frc/controller/LinearQuadraticRegulator.h>
12 #include <frc/drive/DifferentialDrive.h>
13 #include <frc/estimator/KalmanFilter.h>
14 #include <frc/motorcontrol/PWMSparkMax.h>
15 #include <frc/system/LinearSystemLoop.h>
16 #include <frc/system/plant/DCMotor.h>
17 #include <frc/system/plant/LinearSystemId.h>
18 #include <units/angular_velocity.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

```
        # 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();

    // 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, Timer.
↪getFPGATimestamp());
```

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());
```

Tuning Pose Estimators

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 Depuración de controladores y modelos de espacio de estado

Comprobación de señales

Una de las causas más comunes de errores con los controladores de espacio de estado son los letreros que se invierten. Por ejemplo, los modelos incluidos en WPILib esperan que el voltaje positivo dé como resultado una aceleración positiva y viceversa. Si la aplicación de un voltaje positivo no hace que el mecanismo se acelere hacia adelante, o si el movimiento hacia «adelante» hace que el encoder (u otras lecturas del sensor) disminuyan, deben invertirse para que la entrada de voltaje positivo dé como resultado una lectura del codificador positivo. Por ejemplo, si aplico una term:*entrada* de $[12, 12]^T$ (hacia adelante completo para los motores izquierdo y derecho) a mi transmisión diferencial, mis ruedas deberían impulsar mi robot hacia «adelante » (a lo largo del eje + X localmente), y para que mis codificadores lean una velocidad positiva.

Importante: The WPILib `DifferentialDrive`, by default, does not invert any motors. You may need to call the `setInverted(true)` method on the motor controller object to invert so

that positive input creates forward motion.

La importancia de los gráficos

Reliable data of the *system's state*s, *input*s and *output*s over time is important when debugging state-space controllers and observers. One common approach is to send this data over NetworkTables and use tools such as *Shuffleboard*, which allow us to both graph the data in real-time as well as save it to a CSV file for plotting later with tools such as Google Sheets, Excel or Python.

Nota: De forma predeterminada, NetworkTables está limitado a una tasa de actualización de 10 Hz. Para las pruebas, esto se puede omitir con el siguiente fragmento de código para enviar datos a una velocidad de hasta 100 hz. Este código debe ejecutarse periódicamente para publicar nuevos datos a la fuerza.

Peligro: Esto enviará datos adicionales (hasta 100 hz) a través de NetworkTables, lo que puede causar retrasos tanto con el código de usuario como con los paneles de control del robot. Esto también aumentará la utilización de la red. Suele ser una buena idea desactivarlo durante las competencias.

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()
```

Compensación por retraso de entrada

Con frecuencia, algunos datos de entrada del sensor (es decir, lecturas de velocidad) pueden retrasarse debido al filtrado integrado que los controladores de motor inteligentes tienden a realizar. De forma predeterminada, la ganancia K de LQR asume que no hay retardo de entrada, por lo que introducir un retardo significativo del orden de decenas de milisegundos puede provocar inestabilidad. Para combatir esto, se puede reducir la ganancia K del LQR, intercambiando rendimiento por estabilidad. Está disponible un ejemplo de código sobre cómo compensar esta latencia de una manera matemáticamente rigurosa *aquí*.

32.9 Glosario de Control

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.

Esfuerzo de control
: *Control signal*

ley de control
: Una fórmula matemática que genera entradas <input>`para conducir un :term:`sistema a un: *estado* deseado, dado el actual *estado*. Un ejemplo común es la ley de control $\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$

controller
: Usado en una posición o compensación negativa en una *planta* para llegar al deseado *estado del sistema* haciendo cero la diferencia entre la señal de *referencia* y la salida.

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.

dinámica
: Rama de la física que se ocupa del movimiento de los cuerpos bajo la acción de fuerzas. En el control moderno, los sistemas evolucionan según su dinámica.

derivative
: A mathematical operation which evaluates the «rate-of-change» of a function at a given point. See derivative on Wikipedia for more info.

error
: Es la *referencia* menos la salida o el *estado*.

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.

ganancia
: 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.

gradiente
: La derivada, pero aplicada a una función con múltiples entradas. Como resultado, la salida es tanto la magnitud de la tasa de cambio como la dirección del vector a lo largo del cual ocurre.

estado oculto
: Un *estado* que no se puede medir directamente, pero cuya *dinámica* puede relacionarse con otros estados.

entrada
: Una entrada a la *planta* (circuito) que puede modificar el *estado* de la *planta*.

  - Ej. Un volante tendrá 1 entrada: el voltaje del motor que lo impulsa.
  - Ej. Un drivetrain puede tener 2 entradas: los voltajes de los motores izquierdo y derecho.

  Las entradas a menudo son representadas mediante la variable **u**, un vector de columna con una entrada por *input* a *system*.

regresión de minimos cuadrados
: Una técnica de ajuste que elige una curva para minimizar el *cuadrado* del error entre

la curva ajustada y los datos medidos reales. Consulte ordinary least-squares regression en Wikipedia para mas información.

RCL
Regulador Cuadrático Lineal - Un esquema de retroalimentación que busca operar un sistema de la manera «más optima» o «de menor costo», en el sentido de minimizar el cuadrado de alguna «función de costo» que representa una combinación de error del sistema y esfuerzo de control. Esto requiere un modelo matemático preciso del sistema que se esta controlando y una función que describa el «costo» de cualquer estado del sistema dado. Consulte LQR en Wikipedia para más información.

medición
Las mediciones son *salidas* que se miden a partir de una *planta*, o sistema físico, utilizando sensores.

modelo
Un conjunto de ecuaciones matemáticas que refleja algunos aspectos de un comportamiento físico del *sistema*.

observador
En la teoría de control, un sistema que proporciona una estimación del *estado* interno de un *sistema* real dado a partir de las mediciones del entrada`y :term:`salida del *sistema* real. WPILib incluye una clase de filtro de Kalman para observar sistemas lineales y las clases ExtendedKalmanFilter y UnscentedKalmanFilter para sistemas no lineales.

ortogonal
Tener la propiedad de ser independientes o carecer de influencia mutua. Por ejemplo, dos lineas son ortogonales si al mover cualquier cantidad de unidades a lo largo de una línea provoca un desplazamiento cero a lo largo de la otra linea. En un sistema de coordenadas cartesianas a menudo se dice que las lineas ortogonales tienen ángulos de 90 grados entre sí.

output
Medidas de sensores. Puede haber más medidas que estados. Estas salidas se utilizan en el paso «correcto» de los filtros de Kalman.

- Ej. Un volante puede tener 1 salida de un codificador que mide su velocidad.
- Ej. Un tren motriz puede usar solvePNP y V-SLAM para encontrar su posición x/y/posición de frente en el campo. Está bien que haya 6 medidas (solvePNP x/y/frente y V-SLAM x/y/frente) y 3 estados (robot x/y/frente).

Las salidas de un *sistema* a menudo se representan usando la variable **y**, un vector de columna con una entrada por salida (o cosa que podamos medir). Por ejemplo, si nuestro *sistema* tuviera estados para la velocidad y la aceleración pero nuestro sensor solo pudiera medir la velocidad, nuestro, nuestro el vector de salida solo incluiría la velocidad del *sistema*.

retrato de fase
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
Proporcional-Integral-Derivativo - Un controlador de retroalimentación que calcula una señal de control a partir de una suma ponderada del *error*, la tasa de cambio del error y una suma acumulada de errores anteriores. Consulte PID controller. en Wikipedia para mas información.

planta
El *sistema* o el conjunto de actuadores siendo controlados.

variable de proceso
Término usado para describir la salida de una :term:`planta `en contexto del control PID.

r-al cuadrado
Una medida estadística de qué tan bien un modelo predice un conjunto de datos, que representa la fracción de la variación observada en la variable independiente que el modelo predice con precisión. El valor generalmente va desde 0.0 (un ajuste terrible, equivalente a simplemente adivinar el valor promedio de su variable independiente) a 1.0 (un ajuste perfecto). Consulte Coefficient_of_determination en Wikipedia para mas información.

referencia
El estado deseado. Este valor es usado como punto de referencia para que el controlador calcule el error.

tiempo de subida
Es el tiempo que le lleva al *sistema* alcanzar la *referencia* después de haber aplicado la *entrada unitaria*.

RECM
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.

punto de referencia
Término usada para describir la *referencia* de un controlador PID.

tiempo de estabilización
Es el tiempo que le lleva al *sistema* estabilizarse en la *referencia* después de haber aplicado la *entrada unitaria*.

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.

estado
Una característica de un *sistema* (por ejemplo, la velocidad) que se puede utilizar para determinar el comportamiento futuro del *sistema*. En la notación del espacio de estados, el estado de un sistema se escribe como un vector de columna que describe su posición en el espacio de estados.

- Ej. Un sistema de chasis puede tener los estados $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$ para describir su posición en la cancha.
- Ej. Un sistema de elevador puede tener los estados $\begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix}$ para describir su altura y velocidad actual.

Un estado del *sistema* seguido es representado por la variable **x**, un vector de columna con una entrada por *estado*.

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.

error de estado estable
Es el term:*error* <*error*> después que el *sistema* alcanza el equilibrio.

entrada unitaria
Una *entrada* del *sistema* que es 0 para $t < 0$ y una constante mayor que 0 para $t \geq 0$. Una entrada unitaria que es 1 para $t \geq 0$ se le llama unit step input.

respuesta Unitaria
Es la respuesta del *sistema* a una *entrada unitaria*.

sistema
Un término que comprende a la *planta* y su interacción con el controlador y el *observador*, que es tratado como una sola entidad. Matemáticamente hablando, un *sistema* mapeado de *entradas* hacia *salidas* a través de una combinación linear de *estados*.

identificación del sistema
The process of capturing a *systems dynamics* in a mathematical model using measured data. The SysId toolsuite uses system identification to find kS, kV and kA terms.

Respuesta del sistema
Es el comportamiento de un *sistema* durante un tiempo después de aplicar una *entrada*.

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.

punto x
$\dot{\mathbf{x}}$, o punto x: la derivada del vector *state* vector $\mathbf{x}$. Si el *system* tuviera solo una velocidad *state*, entonces $\dot{\mathbf{x}}$ representaría la *system*'s aceleración.

x-hat
$\hat{\mathbf{x}}$, o x-hat: el *estado* estimado de un sistema, como es estimado por un *observador*.

33 Características convenientes

Esta sección cubre algunas características convenientes generales que se usan con otras características de programación avanzada.

33.1 Funciones de Programación en Frecuencias Personalizadas

El método `TimedRobot` de `addPeriodic()` permite ejecutar métodos personalizados a una velocidad mayor que la actualización predeterminada `TimedRobot` (20 ms). Previamente, los equipos tenían que hacer un `Notifier` para ejecutar los controles de retroalimentación más seguido que con el periodo de bucle `TimedRobot` de 20 ms (Ejecutar `TimedRobot` más seguido no se recomienda). Ahora, los usuarios pueden ejecutar controladores de retroalimentación más a menudo que el bucle principal del robot, pero sincronizado con las funciones periódicas `TimedRobot` eliminando posibles problemas de seguridad.

El método `addPeriodic()` (Java) / `AddPeriodic()` (C++) toma una lambda (la función a ejecutar), junto con el periodo solicitado y un desplazamiento opcional de la hora común de inicio. El tercer argumento opcional es útil para programar una función en una timeslot diferente en relación con los otros métodos periódicos `TimedRobot`.

Nota: Las unidades para el período y el desplazamiento son segundos en Java. En C ++, la *biblioteca de unidades* se puede usar para especificar un período y un desplazamiento en cualquier unidad de tiempo.

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++ (Encabezado)

```
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++ (Fuente)

```
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) {
      m_controller.SetSetpoint(30.0);
```

(continúe en la próxima página)

(proviene de la página anterior)

```
    } else {
      m_controller.SetSetpoint(0.0);
    }
  }
}
```

El método `teleopPeriodic()` en este ejemplo se ejecuta cada 20 ms, y la actualización del controlador se ejecuta cada 10 ms con un desfase de 5 ms desde que se ejecuta `teleopPeriodic()` para que sus franjas horarias no entren en conflicto (por ejemplo, `teleopPeriodic()` se ejecuta a 0 ms, 20 ms, 40 ms, etc.; el controlador se ejecuta a 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.

Nota: 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(); }
```

Advertencia: 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); });
```

Remember that button binding is *declarative*: bindings only need to be declared once, ideally some time during robot initialization. The library handles everything else.

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`.

Advertencia: 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++

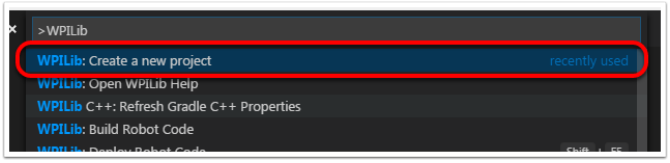
```
frc2::Trigger trigger = booleanEvent.CastTo<frc2::Trigger>();
```

Nota: 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 Proyectos ejemplo de WPILib

Advertencia: Mientras todos los intentos son hechos para mantener los ejemplos de WPILib funcionando, no están con la intención de ser la única manera de hacerlo. Al final, las constantes de robot específicos necesitarán cambiarse por el código para funcionar en un robot de usuario. Muchas constantes empíricas tienen valores «falsificados» para propósitos demostrativos. Los usuarios alientan fuertemente en escribir sus propios códigos (desde cero o desde una plantilla existente) en vez de copiar el código de ejemplo.

Los proyectos de ejemplo de WPILib demuestran un gran número de características en librerías y uso de patrones. El rango de proyectos es para simples demostraciones de una sola función a completar, programas para robots de competencias. Todos estos ejemplos están disponibles en VS Code entrando a `Ctrl+Shift+P`, después seleccionar *WPILib: Create a new project* y elegir un ejemplo.



34.1 Ejemplos básicos

Éstos ejemplos demuestran funciones básicas/mínimas para el robot. Son útiles para equipos que van empezando que van adquiriendo familiaridad con la programación de un robot, pero son realmente limitadas en su funcionalidad.

- **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 Ejemplos de control

Estos ejemplos demuestran las implementaciones de WPILib de controles de robot comunes. Los sensores pueden estar presentes, pero no son el concepto enfatizado de estos ejemplos.

- **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.
- **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 Ejemplos de sensores

Estos ejemplos demuestran la lectura de sensores y el procesamiento de datos usando WPILib. El control de mecanismos puede estar presente, pero no es el concepto enfatizado de estos ejemplos.

- **HTTP Camera** (Java, C++): Demonstrates the use of OpenCV and a HTTP Camera to overlay a rectangle on a captured video feed and stream it to the 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++, Python): Demonstrates the use of the SysIdRoutine API to gather characterization data for a differential drivetrain.

34.4 Ejemplos basados en comandos

Estos ejemplos demuestran el uso de *Command-Based framework*.

- **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 conjunto completo de código de robot para la demostración de robot de 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++): Demuestra la generación y seguimiento de trayectorias con un mecanum drive usando las clases `TrajectoryGenerator` y `MecanumControllerCommand`.
- **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++): Demuestra la generación y seguimiento de trayectorias con un swerve drive usando las clases `TrajectoryGenerator` y `SwerveControllerCommand`.

34.5 Ejemplos de Espacio de Estado

Estos ejemplos demuestran el uso del *State-Space Control*.

- **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 <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src__*, *C ++ <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/__*): Demuestra el control del espacio de estados de un ascensor.
- **StateSpaceArm** (Java, C++): Demuestra el control estado-espacio de un brazo.

34.6 Ejemplos de Simulación Física

Estos ejemplos demuestran el uso de la simulación física.

- **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.
- **SimpleDifferentialDriveSimulation** (Java, C++): Un ejemplo básico de un drivetrain básico que se puede utilizar en simulación.

34.7 Ejemplos varios

Estos ejemplos demuestran diversas funciones de WPILib que no encajan en ninguna de las categorías anteriores.

- **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 ++ <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/exampl* __): Demuestra el uso de HAL (Capa de abstracción de hardware) sin el uso del resto de WPILib. Este ejemplo es para usuarios avanzados (solo C ++).
- **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

Third Party Example Projects

This list helps you find example programs for use with third party devices. You can find support for many of these third parties on the *Recursos de Apoyo* page.

- Cross The Road Electronics (CTRE)
- Kauai Labs (navX)
- Limelight (additional examples, called tutorials, can be found on the left)
- PhotonVision
- REV Robotics

36

Hardware - Conceptos Básicos

36.1 Mejores prácticas de cableado

Truco: 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 Vibración / Choque

Un robot FRC® es un entorno increíblemente peligroso cuando se trata de vibraciones y cargas de impacto. Si bien muchos de los componentes electrónicos específicos de FRC se prueban exhaustivamente para determinar la robustez mecánica en estas condiciones, algunos componentes, como la radio, no están diseñados específicamente para su uso en una plataforma móvil. Tomar medidas para reducir los golpes y las vibraciones a los que están expuestos estos componentes puede ayudar a reducir las fallas. Algunas sugerencias que pueden reducir las fallas mecánicas:

- Aislamiento de vibraciones - Asegúrese de aislar cualquier componente que genere vibraciones excesivas, como los compresores, utilizando «aisladores de vibración». Esto ayudará a reducir la vibración en el robot, que puede aflojar los sujetadores y causar fallas por fatiga prematura en algunos componentes electrónicos.
- Bumpers - Utilice bumpers para cubrir la mayor parte del robot posible para su diseño. Si bien las reglas requieren una cobertura de bumpers específica alrededor de las esquinas de su robot, maximizar el uso de bumpers aumenta la probabilidad de que sus bumpers amortigüen todas las colisiones. Los bumpers reducen significativamente las fuerzas G experimentadas en una colisión en comparación con golpear directamente sobre la superficie dura de un robot, reduciendo el impacto experimentado por los componentes electrónicos y disminuyendo la posibilidad de una falla relacionada con el impacto.
- Montaje con amortiguación - puede optar por montar con amortiguación algunos o todos sus componentes electrónicos para reducir aún más las fuerzas que ven en las colisiones de robots. Esto es especialmente útil para la radio del robot y otros componentes electrónicos, como los coprocesadores, que pueden no estar diseñados para su uso en

plataformas móviles. Los aisladores de vibración, resortes, espumas o el montaje en materiales flexibles pueden reducir las fuerzas de impacto que perciben estos componentes.

36.1.2 Redundacía

Desafortunadamente, hay pocos lugares en el sistema de control FRC donde la redundancia es factible. Aprovechar las oportunidades de redundancia puede aumentar la confiabilidad. El ejemplo principal de esto es el cableado del conector cilíndrico a la radio además de la conexión PoE provista. Esto asegura que si uno de los cables se daña o se desprende, el otro mantendrá la alimentación de la radio. Esté atento a otras áreas potenciales para proporcionar redundancia al cablear y programar su robot.

36.1.3 Ahorradores de puertos

Para cualquier conexión en el robot o la driver station que se puedan enchufar y desenchufar con frecuencia (como joysticks DS, DS Ethernet, anclaje USB roboRIO y anclaje Ethernet), el uso de un «Port Saver» o «pigtail» puede reducir sustancialmente el potencial de daños el puerto. Este tipo de dispositivo puede cumplir una doble función, tanto para reducir el número de ciclos que ve el puerto del dispositivo electrónico como para reubicar la conexión a una ubicación más conveniente. Asegúrese de asegurar el protector de puertos (consulte el siguiente elemento) para evitar daños en el puerto.

36.1.4 Gestión de cables y alivio de tensión

Uno de los componentes más críticos para la confiabilidad y el mantenimiento del robot es una buena administración de cables y alivio de tensión. Una buena gestión de cables consta de algunos componentes:

- Asegúrese de que los cables tengan la longitud correcta. Cualquier exceso de longitud de cable es más fácil de manejar. Si debe tener un cable adicional debido a la longitud adicional del cableado COTS, asegure el cable adicional en un paquete pequeño con bridas para cables separados antes de asegurar el resto del cable.
- Asegúrese de que los cables estén asegurados cerca de los puntos de conexión, con suficiente holgura para evitar tensar los conectores. Esto se llama alivio de tensión y es fundamental para minimizar la probabilidad de que un cable se desconecte o se rompa en un punto de conexión (generalmente son concentradores de tensión).
- Asegure los cables cerca de cualquier componente móvil. Asegúrese de que todos los tendidos de cables sean seguros y estén protegidos de los componentes móviles, incluso si los componentes móviles se doblaran o se desplazaran demasiado.
- Asegure los cables en puntos adicionales según sea necesario para mantener el cableado ordenado y limpio. Tenga cuidado de no asegurar demasiado los cables; si los cables están asegurados en demasiados lugares, puede dificultar la resolución de problemas y el mantenimiento.

36.1.5 Documentación

Una excelente manera de facilitar el mantenimiento es crear documentación que describa qué está conectado en qué parte del robot. Hay varias formas de crear este tipo de documentación que van desde diagramas de cableado completos hasta gráficos de Excel y una lista rápida de las funciones que se adjuntan a qué canales. Muchos equipos también integran estas listas con etiquetado (consulte la siguiente viñeta).

Cuando un cable se corta accidentalmente, o un mecanismo no funciona correctamente o un componente se quema, será mucho más fácil de reparar si tiene alguna documentación que le indique qué está conectado y dónde sin tener que rastrear el cableado hasta el final (incluso si su cableado está ordenado)

36.1.6 Etiquetado

El etiquetado es una excelente manera de complementar la documentación de cableado descrita anteriormente. Existen muchas estrategias diferentes para etiquetar el cableado y la electrónica, todas con sus pros y sus contras. Las etiquetas para dispositivos electrónicos y las banderas para los cables se pueden hacer a mano o con una rotuladora (algunas también pueden hacer etiquetas termocontraíbles), o puede usar diferentes colores de cinta aislante o banderas de etiquetado para indicar diferentes cosas. Sea cual sea el sistema que elija, asegúrese de comprender cómo complementa su documentación y de que todos los miembros de su equipo estén familiarizados con él.

36.1.7 Verifique todo el cableado y las conexiones

Una vez que se haya completado todo el cableado del robot, asegúrese de verificar cada conexión, tirando de cada una, para asegurarse de que todo esté seguro. Además, asegúrese de que ningún «bigote» de cables sueltos sobresalga de ningún punto de conexión y de que no queden al descubierto conexiones sin aislamiento. Si alguna conexión se afloja durante la prueba, o se descubre algún «bigote», vuelva a hacer la conexión y asegúrese de que una segunda persona la revise cuando termine.

Una fuente común de malas conexiones son los sujetadores de tornillo o de tuerca y perno. Para cualquier conexión de este tipo en el robot (por ejemplo, conexiones de batería, interruptor principal, PDP, roboRIO), asegúrese de que los sujetadores estén apretados. Para conexiones tipo tuerca y perno, asegúrese de que el cable / terminal no se pueda girar con la mano; Si puede girar el cable de la batería o la conexión del disyuntor principal agarrando el terminal y girando, la conexión no está lo suficientemente apretada.

Otra fuente común de fallas son los fusibles al final del PDP. Asegúrese de que estos fusibles estén completamente asentados; es posible que deba aplicar más fuerza de la que espera para asentarlos por completo. Si los fusibles están colocados correctamente, probablemente será difícil o imposible quitarlos a mano.

Las conexiones a presión, como el conector SB-50, deben asegurarse con clips o bridas para asegurar que no se suelten durante los impactos.

36.1.8 Vuelva a verificar antes y con frecuencia

Vuelva a comprobar todo el sistema eléctrico lo más a fondo posible después de jugar el primer partido o dos (o realizar pruebas muy rigurosas). Los primeros impactos que ve el robot pueden aflojar los sujetadores o exponer problemas.

Cree una lista de verificación para volver a verificar las conexiones eléctricas con regularidad. Como un punto de partida muy aproximado, los sujetadores giratorios, como las conexiones de la batería y del PDP, deben revisarse cada 1-3 coincidencias. Las conexiones de tipo resorte, como los conectores WAGO y Weidmuller, probablemente solo deban verificarse una vez por evento. Asegúrese de que el equipo sepa quién es responsable de completar la lista de verificación y cómo documentarán que se ha hecho.

36.1.9 Mantenimiento de la batería

¡Cuida bien tus baterías! Una batería defectuosa puede hacer que un robot funcione mal o no funcione en absoluto durante un partido. Etiquete todas sus baterías para ayudar a realizar un seguimiento del uso durante el evento. Muchos equipos también incluyen información como la edad de la batería en esta etiqueta.

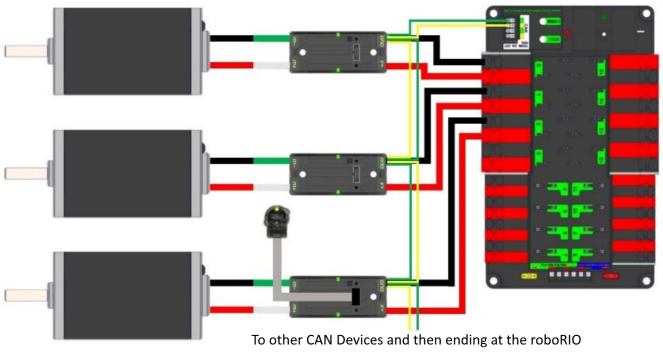
- ¡Nunca levante ni transporte las baterías por los cables! Llevar las baterías por los cables tiene el potencial de dañar la conexión interna entre los terminales y las placas, aumentando dramáticamente la resistencia interna y degradando el rendimiento.
- Marque cualquier batería caída como defectuosa hasta que se pueda realizar una prueba completa. Además de las conexiones terminales mencionadas, dejar caer una batería también tiene el potencial de dañar celdas individuales. Es posible que este daño no se registre en una prueba de voltaje simple, sino que se esconda hasta que la batería se coloque bajo carga.
- Gire las baterías de manera uniforme. Esto ayuda a garantizar que las baterías tengan más tiempo para cargarse y descansar y que se desgasten de manera uniforme (igual número de ciclos de carga / descarga)
- Cargue las baterías de prueba si es posible para monitorear la salud. Hay una serie de productos disponibles comercialmente que los equipos utilizan para cargar baterías de prueba, incluido al menos uno diseñado específicamente para FRC. Una prueba de carga puede proporcionar un indicador del estado de la batería midiendo la resistencia interna. Esta medida es mucho más significativa cuando se trata de igualar el rendimiento que un simple número de voltaje sin carga proporcionado por un multímetro.

36.1.10 Comprobar registros de DS

Después de cada partido, revise los registros de DS para ver cómo se ve el voltaje de la batería y el uso actual. Una vez que haya establecido cuál es el rango normal de estos elementos para su robot, es posible que pueda detectar problemas potenciales (baterías defectuosas, motores defectuosos, atascos mecánicos) antes de que se conviertan en fallas críticas.

36.2 Conceptos básicos del cableado CAN

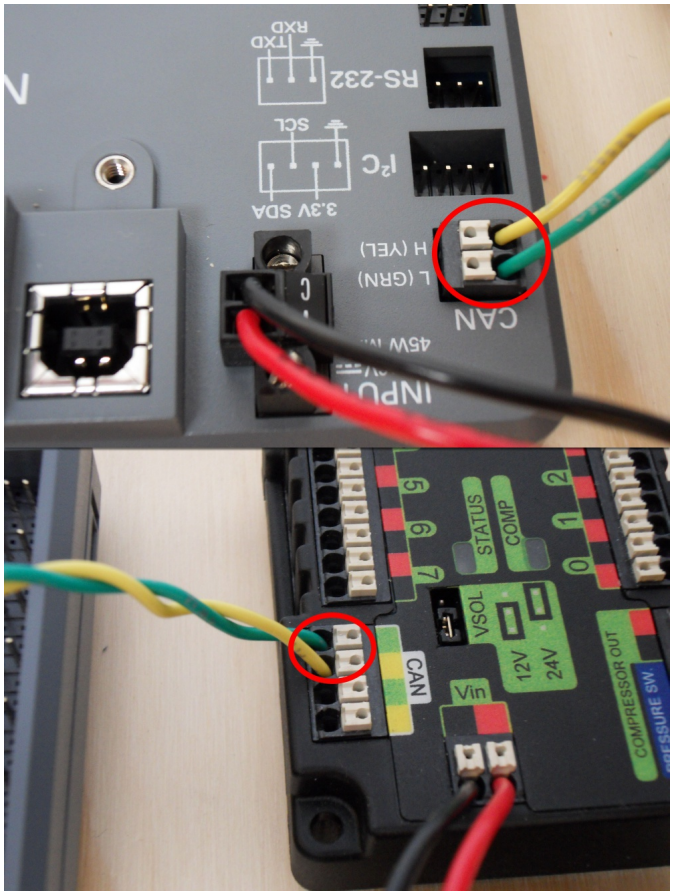
CAN is a two wire network that is designed to facilitate communication between multiple devices on your robot. It is recommended that CAN on your robot follow a «daisy-chain» topology. This means that the CAN wiring should usually start at your roboRIO and go into and out of each device successively until finally ending at the *PDP*.



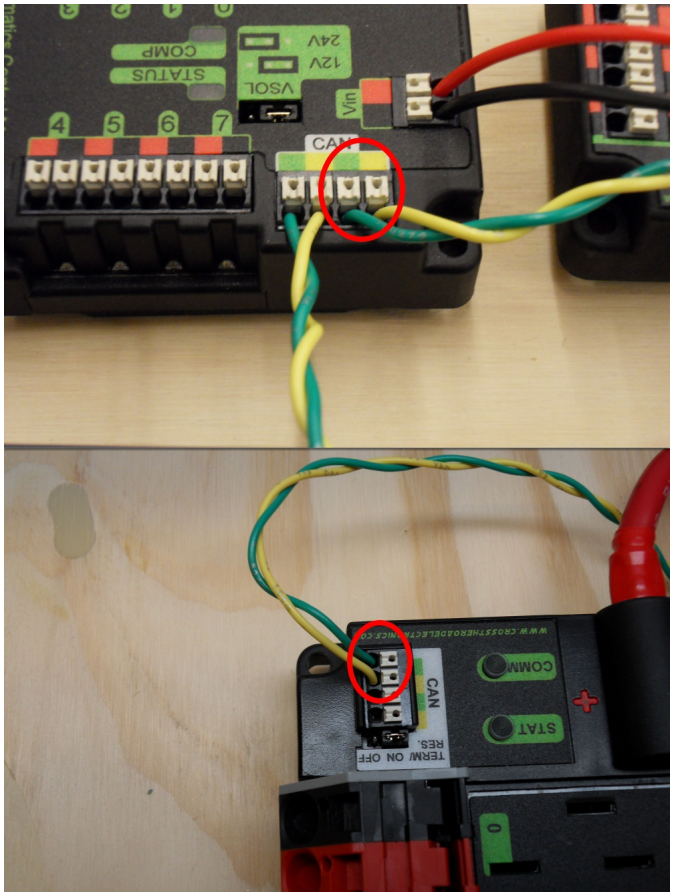
36.2.1 Cableado estándar

CAN generalmente está cableado con cables amarillo y verde, el amarillo actúa como CAN-High y el verde como las señales CAN-Low. Muchos dispositivos muestran este esquema de color amarillo y verde para indicar cómo se deben conectar los cables.

CAN cableado del roboRIO al PCM.



Cableado CAN desde el PCM al PDP.



36.2.2 Terminación

Se recomienda que el cableado comience en el roboRIO y termine en el PDP porque se requiere que la red CAN esté terminada con resistencias 120 Ω y estas están integradas en estos dos dispositivos. El PDP se envía con el puente de resistencia de terminación del bus CAN en la posición «ON». Se recomienda dejar el puente en esta posición y colocar cualquier nodo CAN adicional entre el roboRIO y el PDP (dejando el PDP como el final del bus). Si desea colocar el PDP en el medio del bus (utilizando ambos pares de terminales CAN PDP) mueva el puente a la posición «OFF» y coloque su propia resistencia de terminación 120 Ω al final de su Cadena de bus 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*.

Consejo: For pneumatics safety & mechanical requirements, consult this year's Robot Construction rules. For mechanical design guidelines, the FIRST Pneumatics Manual is located here

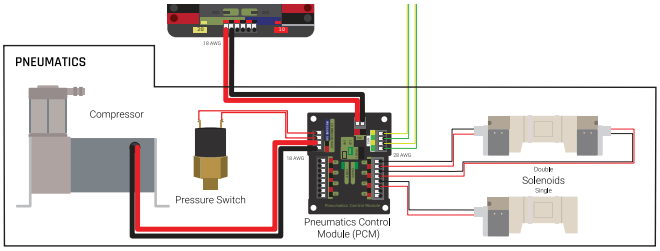
---

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 *Pneumatics Power (Optional)*

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.

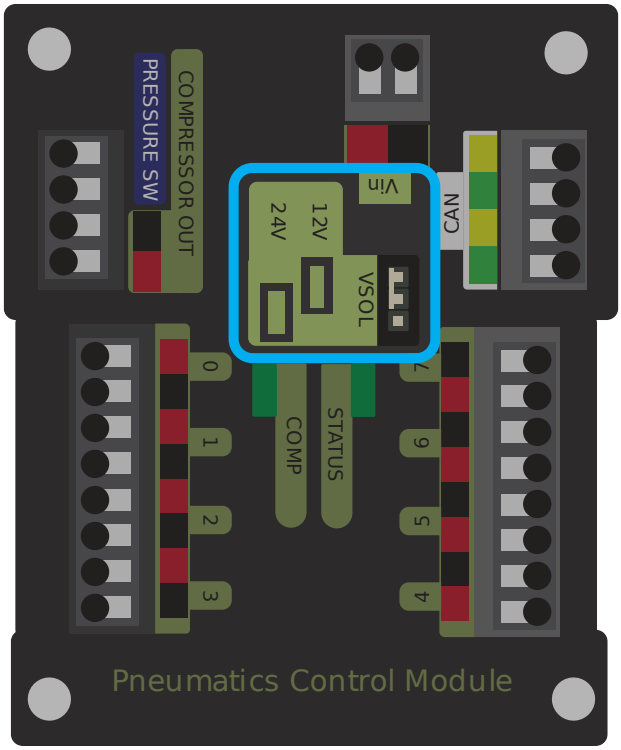
36.3.4 The Pressure Switch

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*.

Consejo: For pneumatics safety & mechanical requirements, consult this year's Robot Construction rules. For mechanical design guidelines, the FIRST Pneumatics Manual is located here

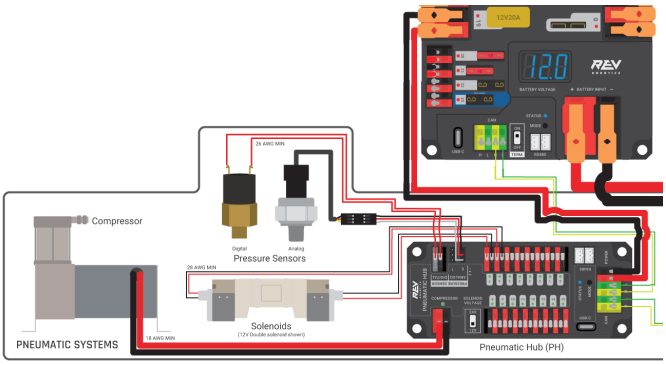
---

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 *Pneumatics Power (Optional)*

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.

Advertencia: 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 Referencia rápida del estado de la luz

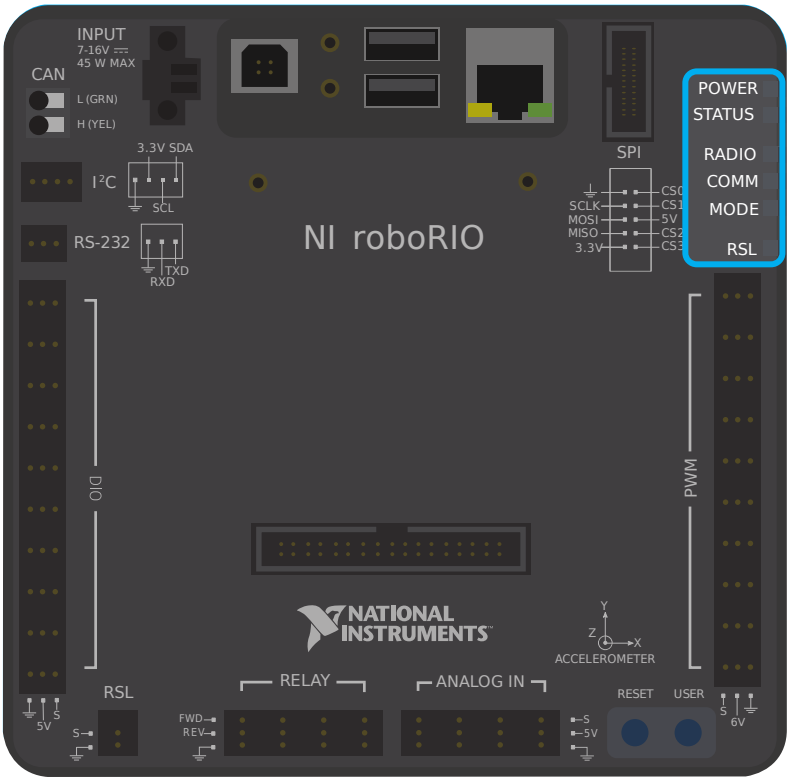
Muchos de los componentes del sistema de control de FRC® tienen luces indicadoras que se pueden usar para diagnosticar rápidamente los problemas con su robot. Esta guía muestra cada uno de los componentes de hardware y describe el significado de los indicadores. Fotos e información de Innovation FIRST y Cross the Road Electronics.

A compact and printable Status Light Quick Reference is available.

36.5.1 Luz de señal de robot (RSL)

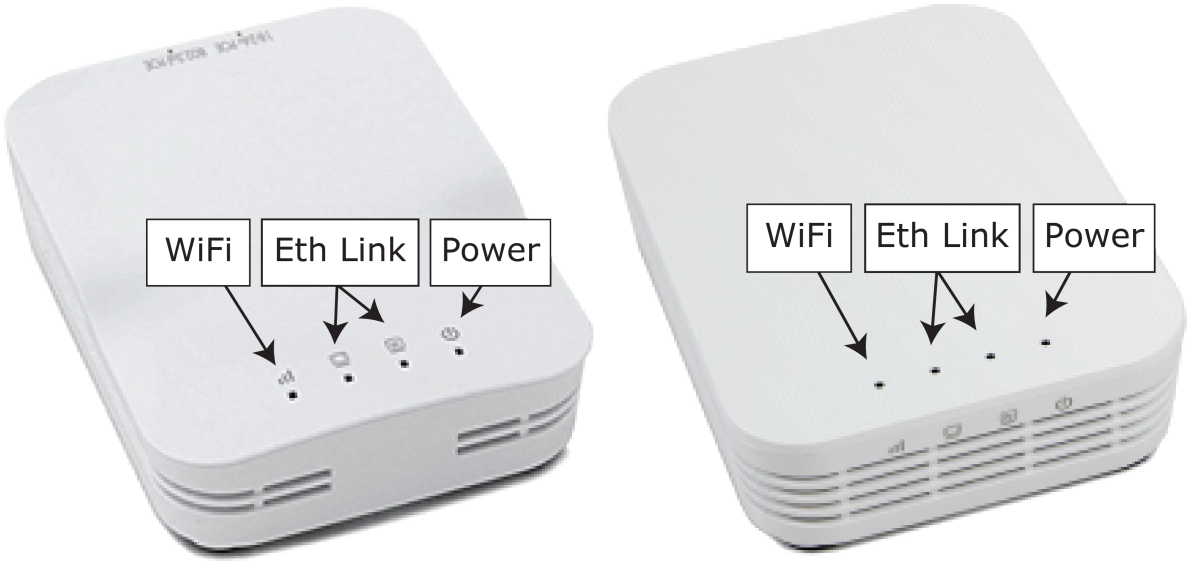
Encendido fijo	Robot encendido y deshabilitado
Parpadeando	Robot encendido y habilitado
Apagado	Robot apagado, roboRIO no encendido o RSL no cableado correctamente

36.5.2 roboRIO



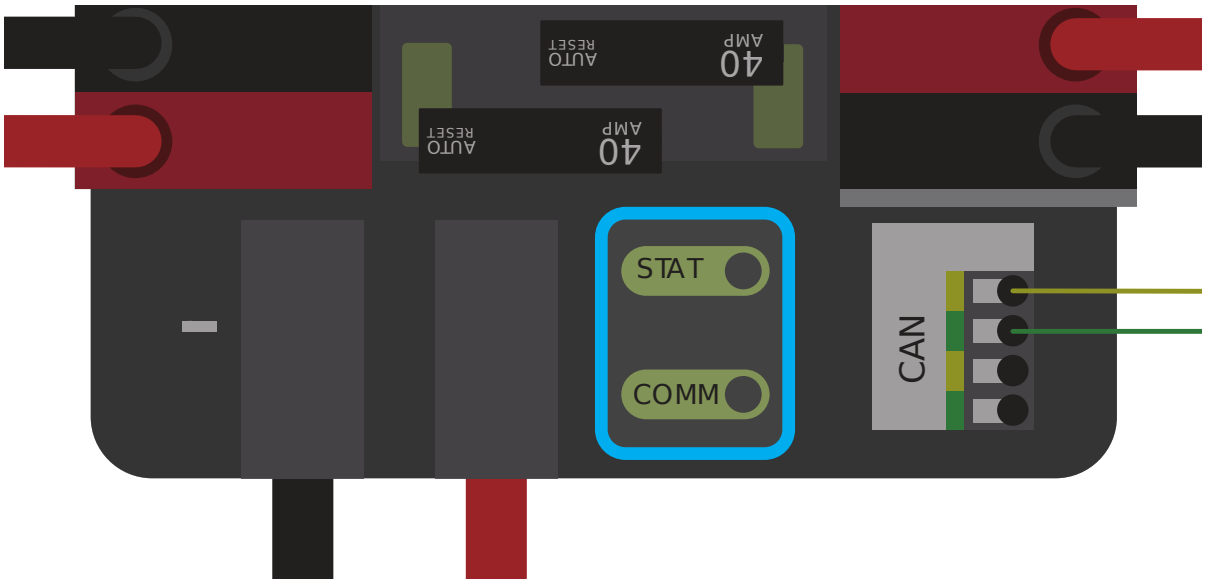
Power	Verde	La energía está bien
	Ámbar	Protección contra apagones activada, salidas desactivadas
	Rojo	Fallo de energía, revise los rieles del usuario para ver si hay cortocircuitos
Status	Encendido mientras el controlador se está iniciando, luego debería apagarse	
	2 parpadeos	Error de software, reimagen de roboRIO
	3 parpadeos	Modo seguro, reinicie roboRIO, vuelva a crear una imagen si no se resuelve
	4 parpadeos	El software se bloqueó dos veces sin reiniciar, reinicie roboRIO, vuelva a crear una imagen si no se resuelve
	Parpadeo constante o fijo	Error irrecuperable
Radio	No implementado actualmente	
Com	Apagado	Sin comunicacion
	Rojo sólido	Comunicación con DS, pero sin código de usuario en ejecución
	Parpadeando rojo	Paro de emergencia activado
	Verde sólido	Buena comunicación con DS
Mode	Apagado	Salidas deshabilitadas (robot deshabilitado, apagón, etc.)
	Naranja	Autónomo Habilitado
	Verde	Teleoperado Habilitado
	Rojo	Test Habilitado
RSL	*Ver arriba <#robot-signal-light-rsl>* _	

36.5.3 Radio OpenMesh



Energía	Azul	Encendido o encendiendo
	Parpadeando azul	Encendiendo
Eth Link	Azul	Vinculándose
	Parpadeando azul	Tráfico presente
WiFi	Apagado	Modo puente, firmware no vinculado o no de FRC
	Rojo	AP, desvinculado
	Amarillo/Naranja	AP, Vinculado
	Verde	Modo de puente, Vinculado

36.5.4 Panel de distribución de energía



Estatus PDP/Comm LEDs

LED	Estroboscópico	Lento
Verde	Sin fallas: robot habilitado	Sin fallas: robot deshabilitado
Naranja	NA	Falla pegajosa
Rojo	NA	Sin comunicación CAN

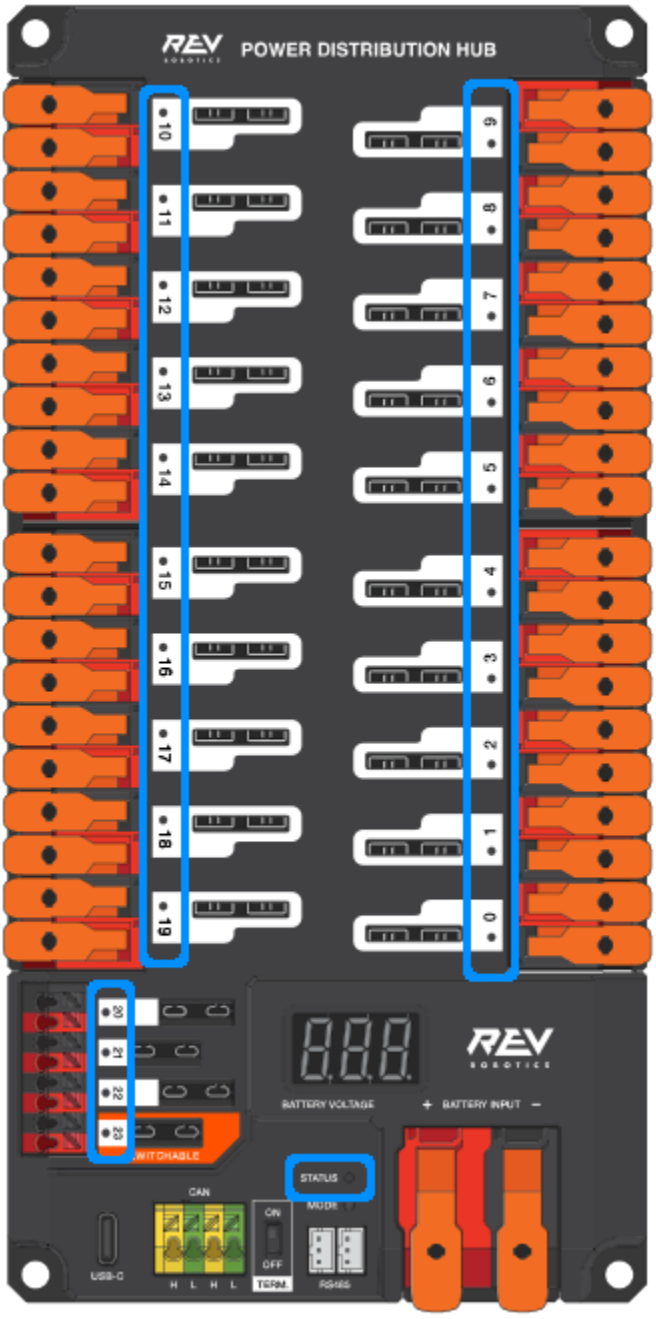
Truco: Si un LED del PDP muestra más de un color, consulte la tabla de estados especiales de LED del PDP a continuación. Para obtener más información sobre la resolución de fallas del PDP, consulte el Manual de usuario del PDP.

Nota: Note that the No *CAN* Comm fault will occur if the PDP cannot communicate with the roboRIO via CAN Bus.

Estados especiales del PDP

Colores LED	Problema
Rojo/Naranja	Hardware dañado
Verde/Naranja	En el cargador de arranque
Sin LED	Sin energía / Polaridad incorrecta

36.5.5 Power Distribution Hub



Nota: Esos patrones de led solo aplican a la versión de firmware 21.1.7 y superiores

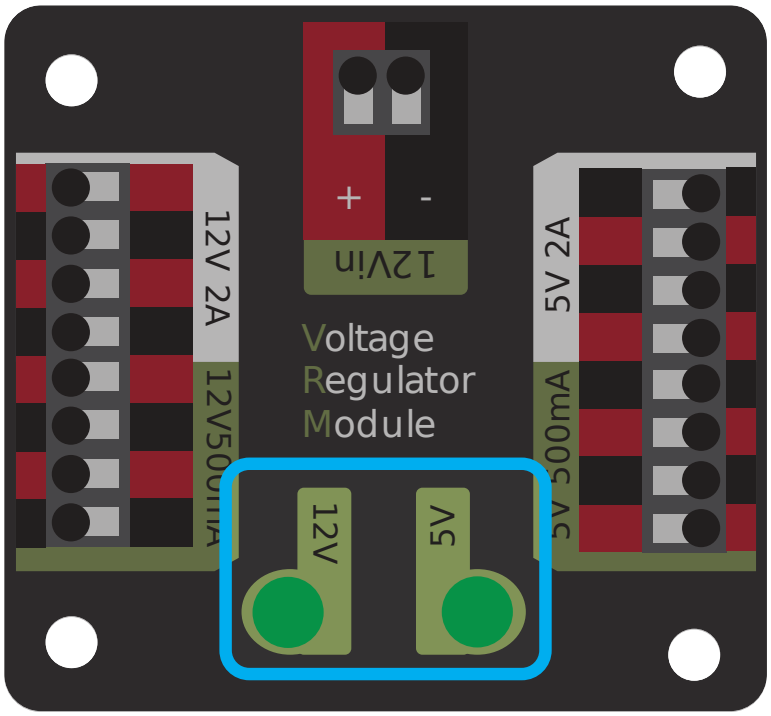
36.5.6 LED del estado de PDH

Color de LED	Estado
Azul sólido	Dispositivo encendido pero comunicación no establecida
Verde sólido	Comunicacion Principal con roboRIO establecida
Magenta Parpadeando	Mantener con vida el tiempo de espera
Solid Cyan	Secondary Heartbeat (Connected to REV Hardware Client)
Naranja/Azul parpadeando	Bateria baja
Naranja/Amarillo parpadeando	falla en CAN
Naranja/Cian parpadeando	falla del Hardware
Naranja/Rojo parpadeando	A prueba de fallos
Naranja/Magenta parpadeando	Dispositivo sobre corriente

Channel LEDs

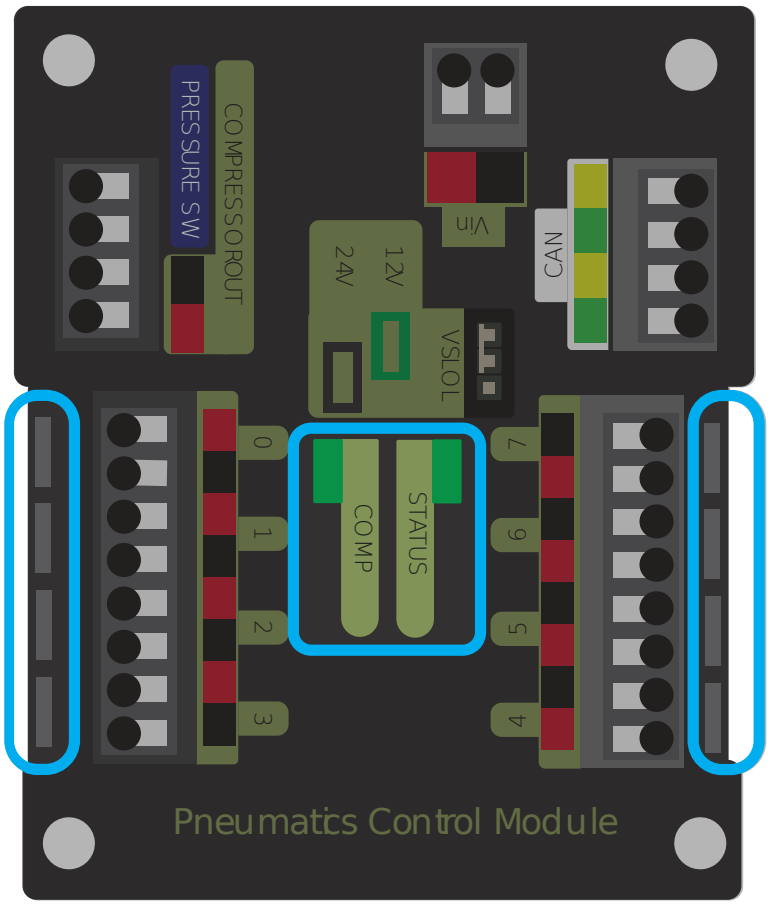
Color de LED	Estado
Apagado	El canal tiene voltaje y está operando como se esperaba
Rojo sólido	Channel has NO voltage and there is an active fault. Check for tripped or missing circuit breaker / fuse
Parpadeando rojo	Sticky fault on the channel. Check for tripped circuit breaker / fuse.

36.5.7 Módulo regulador de voltaje



Los LED de estado del VRM indican el estado de las dos fuentes de alimentación. Si el suministro funciona correctamente, el LED debe iluminarse en verde brillante. Si el LED no está encendido o está apagado, la salida puede estar en cortocircuito o consumir demasiada corriente.

36.5.8 Módulo de control neumático (PCM)



PCM Estado del LED

LED	Estroboscópico	Lento	Largo
Verde	No se ha activado ningún fallo en el robot	Falla pegajosa	NA
Naranja	NA	Falla pegajosa	NA
Rojo	NA	No hay comunicación con el CAN o hay fallo del solenoide (parpadea índice de solenoides)	Fallo del Compresor

Truco: Sí un LED en el PCM muestra más de un color, vea la tabla de los diferentes estados especiales del LED del PCM la cual se encuentra abajo. Para más información sobre cómo resolver errores del PCM, consulte el manual de usuario del PCM.

Nota: Tome en cuenta que el fallo de la comunicación del CAN no se producirá solo sí el dispositivo no puede comunicarse con cualquier otro dispositivo, si el PCM y la PDP pueden comunicarse entre sí, pero no con la roboRIO.

Tabla de estados especiales LED del PCM

LED	Problemas
Rojo/Naranja	Hardware dañado
Verde/Naranja	En el cargador de arranque
Sin LED	No hay energía / Polaridad incorrecta

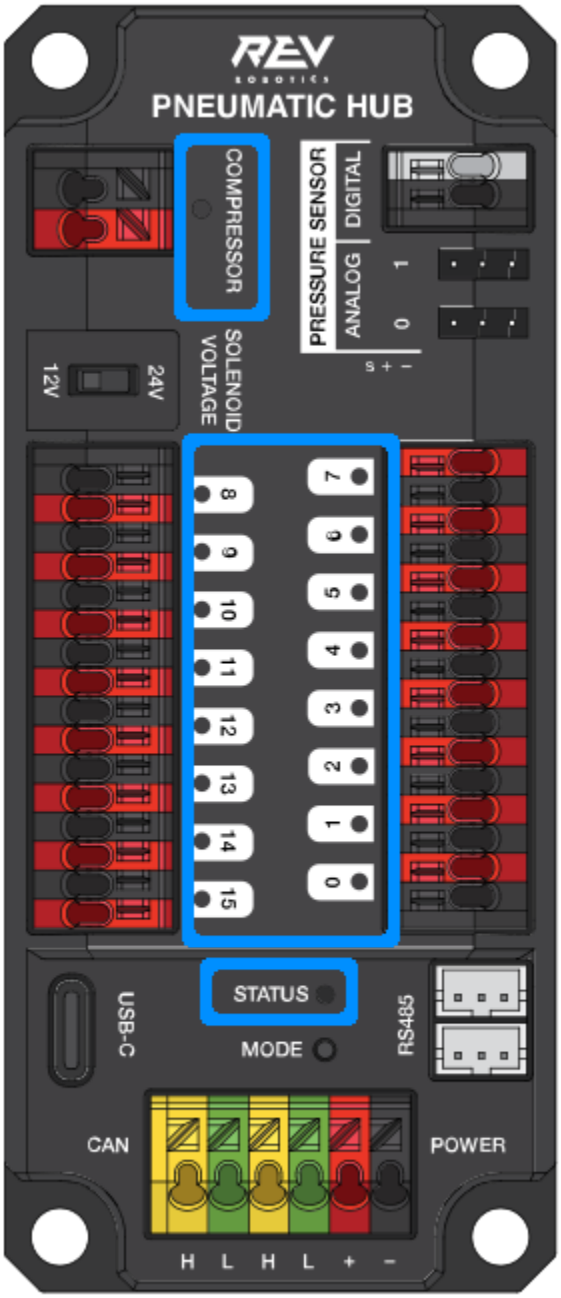
PCM LED del compresor

Este es el LED del compresor. Este LED es verde cuando la salida del compresor está activa (actualmente el compresor se encuentra encendido) y se apaga cuando la salida del compresor no está activa.

LEDs del canal de solenoide PCM

Estos LEDs se encienden en rojo si el canal del solenoide está activado y no se encienden si está desactivado.

36.5.9 Pneumatic Hub



Nota: Esos patrones de led solo aplican a la versión de firmware 21.1.7 y superiores

PH Status LED

Color de LED	Estado
Azul sólido	Dispositivo encendido pero comunicación no establecida
Verde sólido	Main Communication established
Magenta Parpadeando	Mantener con vida el tiempo de espera
Solid Cyan	Secondary Heartbeat (connected to REV HW Client)
Naranja/Azul parpadeando	falla del Hardware
Naranja/Amarillo parpadeando	falla en CAN
Naranja/Rojo parpadeando	A prueba de fallos
Naranja/Magenta parpadeando	Dispositivo sobre corriente
Orange/Green Blinking	Orange/Green Blinking

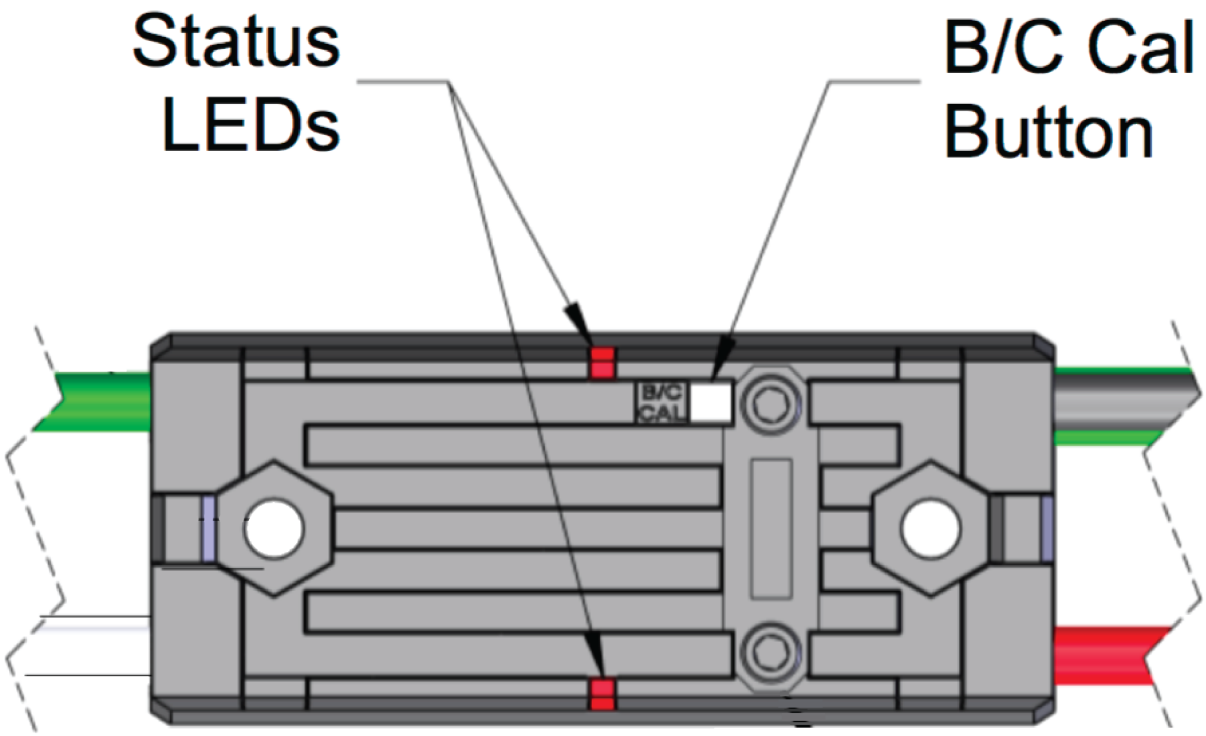
Compressor LED

Color de LED	Estado
Verde sólido	Compressor On
Black Solid	Compressor Off

Solenoid LEDs

Color de LED	Estado
Verde sólido	Solenoid On
Black Solid	Solenoid Off

36.5.10 Controladores de motor Talon SRX, Victor SPX y Talon FX



Estado de LEDs durante la operación normal

LEDs	Colores	Estado del Talon SRX
Ambos	Parpadeando verde	Se aplica el acelerador de avance. La tasa de parpadeo es proporcional al ciclo de trabajo.
Ambos	Parpadeando rojo	Se aplica el acelerador en reversa. La tasa de parpadeo es proporcional al ciclo de trabajo.
Ninguno	Ninguno	No hay voltaje aplicado al Talon SRX
LEDs alternando	Apagado/Naranja	Bus CAN detectado, robot desactivado
LEDs alternando	Apagado/Rojo lento	Bus CAN/PWM no es detectado
LEDs alternando	Apagado/Rojo rápido	Falla detectada
LEDs alternando	Rojo/Naranja	Hardware dañado
LEDs estroboscopio hacia (M-)	Apagado/Rojo	Interruptor de límite de avance o límite suave de avance
LEDs estroboscopio hacia (M+)	Apagado/Rojo	Interruptor de límite inverso o límite suave inverso
Solo LED1 (el más cercano a M+/V+)	Verde/Naranja	In Boot-loader
LEDs estroboscopio hacia (M+)	Apagado/Naranja	Falla térmica / apagado (solo Talon FX)

Estado de los LEDs durante la calibración

Status LEDs Blink Code	Estado del Talon SRX
Rojo/Verde intermitente	Modo de calibración
Parpadeando verde	Calibración exitosa
Parpadeando rojo	Calibración fallida

B/C CAL Códigos de parpadeo

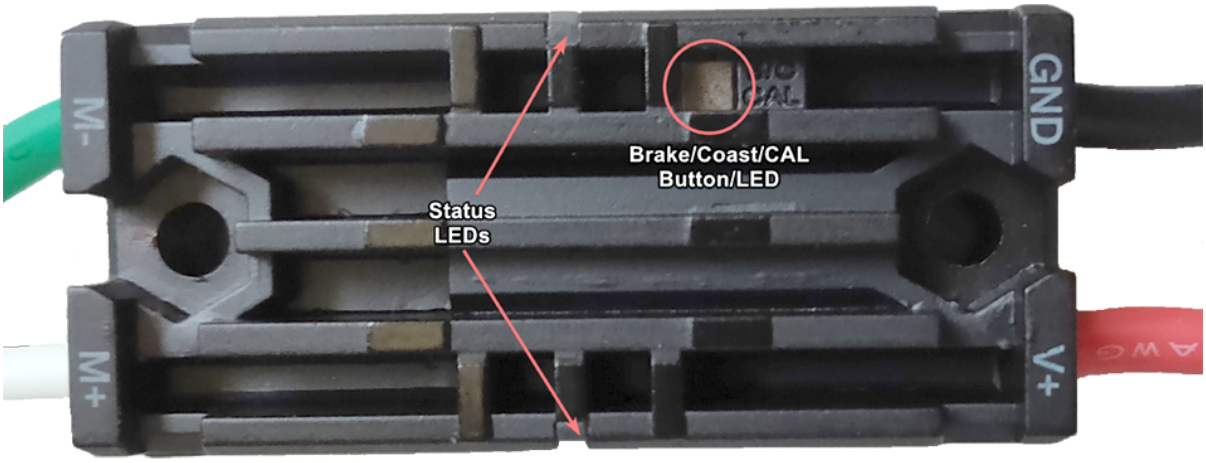
B/C CAL Color del botón	Estado del Talon SRX
Rojo	Modo Brake
Apagado	Modo Coast

36.5.11 Controlador de Motor 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 REV Robotics SPARK

36.5.13 Controlador de Motor Victor-SP

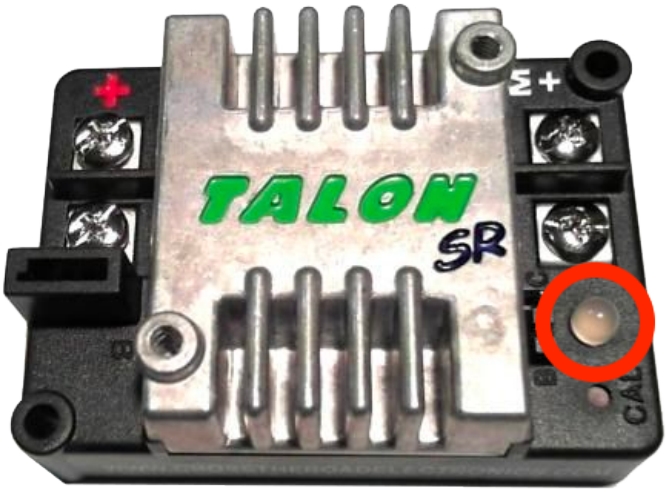


Brake/Coast/Cal Button/LED – Rojo sí el controlador está en modo brake, apagado sí el controlador está en modo coast.

Estado

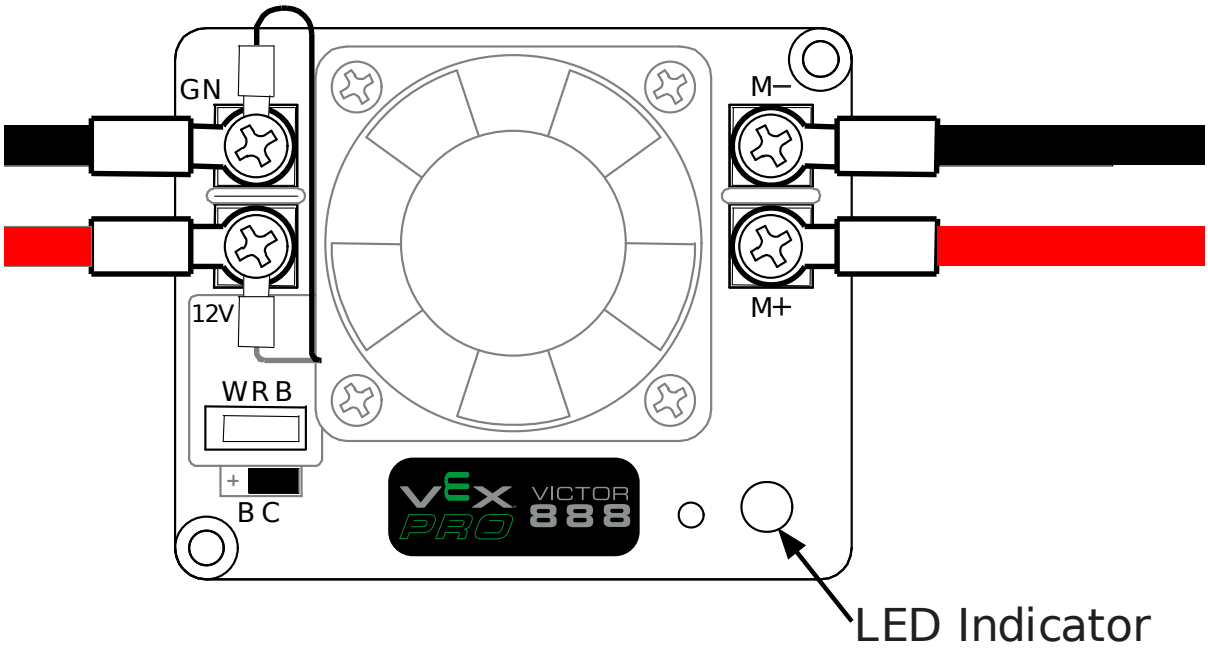
Verde	Sólido	Salida hacia adelante completa
	Parpadeando	Hacia adelante proporcional al voltaje de salida
Rojo	Sólido	Salida en reversa completa
	Parpadeando	Hacia adelante proporcional al voltaje de salida
Naranja	Sólido	Estado desactivado, señal PWM perdida, robot FRC desactivado, o señal en rango muerto (+/- 4% de salida)
Rojo/Verd	Parpadeando	Listo para la calibración. Varios parpadeos verdes indican una calibración exitosa, y rojo varias veces indica una calibración fallida.

36.5.14 Controlador de Motor Talon



Verde	Sólido	Salida hacia adelante completa
	Parpadeando	Hacia adelante proporcional al voltaje de salida
Rojo	Sólido	Salida en reversa completa
	Parpadeando	En reversa proporcional al voltaje de salida
Naranja	Sólido	No hay dispositivos CAN conectados
	Parpadeando	Estado desactivado, señal PWM perdida, robot FRC desactivado, o señal en rango muerto (+/- 4% de salida)
Apagado		No hay energía de entrada al Talon
Rojo/Verde	Parpadeando	Listo para la calibración. Varios parpadeos verdes indican una calibración exitosa, y rojo varias veces indica una calibración fallida.

36.5.15 Controlador de Motor Victor888

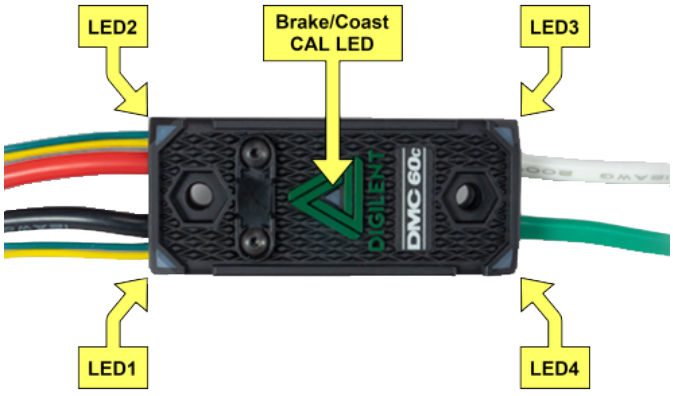


Verde	Sólido	Salida hacia adelante completa
	Parpadeando	Calibración exitosa
Rojo	Sólido	Salida en reversa completa
	Parpadeando	Calibración no exitosa
Naranja	Sólido	Neutral/brake
Rojo/Verde	Parpadeando	Modo de calibración

36.5.16 Controlador de Motor Jaguar

Estado del LED	Estado del módulo
Condiciones normales de operación	
Amarillo	Neutral (velocidad fijada en 0)
Verde intermitente rápido	Adelante
Rojo intermitente rápido	Reversa
Verde	A toda velocidad
Rojo	Reversa a toda velocidad
Condiciones de falla	
Amarillo intermitente lento	Pérdida del servo o del enlace de la red
Amarillo intermitente rápido	ID del CAN inválido
Rojo intermitente lento	Voltaje, temperatura o condición de falla del interruptor de límite
Rojo y Amarillo intermitente lento	Condición de falla de corriente
Calibración o condiciones CAN	
Rojo y Verde intermitente	Modo de calibración activo
Rojo y Amarillo intermitente	Fallo en el modo de calibración
Verde y Amarillo intermitente	Éxito en el modo de calibración
Verde intermitente lento	Modo de asignación del ID del CAN
Amarillo intermitente rápido	ID de CAN actual (cuenta parpadeos para determinar el ID)
Amarillo intermitente	ID de CAN inválido (es decir, puesto a 0) en espera de una asignación de ID válida

36.5.17 Digilent DMC-60



El DMC60C contiene cuatro LEDs RGB (Rojo, Verde y Azul) y un Brake/Coast CAL LED. Los cuatro RGB LEDs están localizados en las esquinas y se usan para indicar el estado durante la fase normal de operación, así como cuando se produce un fallo. El Brake/Coast CAL LED está localizado en el centro del triángulo, que se encuentra en el centro de la cubierta, y es usado para indicar la configuración actual del Brake/Coast. Cuando el LED central está apagado, el dispositivo está operando en el modo coast. Cuando el LED central está iluminado, el dispositivo está operando en el modo brake. El modo Brake/Coast se puede cambiar presionando

el centro del triángulo y después soltando el botón.

Al encenderse, los LEDs RGB se iluminan de color de azul, cada vez más brillantes. Esto dura aproximadamente 5 segundos. Durante este tiempo, el controlador del motor no responderá a una entrada, ni se activarán los controladores de salida. Después de que se haya completado el encendido inicial, el dispositivo comienza a funcionar de manera normal y lo que se muestra en los LEDs RGB es una función de la señal de entrada que se aplica, así como el estado actual de la falla. Asumiendo que no se han producido fallos, los LEDs funcionan de la siguiente manera:

Señal PWM aplicada	Estado del LED
No hay señal de entrada o el ancho del pulso de entrada es inválido	Alternar entre los LEDs superiores (LED1 y LED2) y los inferiores (LED3 y LED4) se iluminan en rojo y apagados.
Ancho del pulso de entrada neutral	Los 4 LEDs iluminados de naranja.
Ancho del pulso de entrada positivo	Los LEDs parpadean verde en un patrón circular en el sentido de las manecillas del reloj (LED1 → LED2 → LED3 → LED4 → LED1). La frecuencia de actualización del LED es proporcional al ciclo de trabajo de la salida y aumenta con el crecimiento del ciclo de trabajo. Al 100% del ciclo de trabajo, los 4 LEDs se iluminan en verde.
Ancho del pulso de entrada negativo	Los LEDs parpadean rojo en un patrón circular en sentido contrario a las manecillas del reloj (LED1 → LED4 → LED3 → LED2 → LED1). La frecuencia de actualización del LED es proporcional al ciclo de trabajo de la salida y aumenta con el crecimiento del ciclo de trabajo. Al 100% del ciclo de trabajo, los 4 LEDs se iluminan rojo.

Estado de Control del Bus CAN	Estado del LED
No se ha detectado ninguna señal de entrada o error en el Bus CAN	Alternar entre los LEDs superiores (LED1 y LED2) y los inferiores (LED3 y LED4) se iluminan en rojo y apagados.
No se ha recibido ningún cuadro de control CAN en los últimos 100 ms o el último cuadro de control especificado mode-NoDrive (Salida Desactivada)	Alternan entre la parte superior (LED1 y LED2) y la parte inferior (LED3 y LED4) los LEDs se iluminan en naranja y apagados.
Cuadro de control CAN válido recibido en los últimos 100 ms. El modo de control especificado dio como resultado a que se aplicara un ciclo de trabajo neutro a la salida del motor.	Los 4 LEDs se iluminan de naranja
Cuadro de control CAN válido recibido en los últimos 100 ms. El modo de control especificado dio como resultado un ciclo de trabajo positivo en la salida del motor.	Los LEDs parpadean verde en un patrón circular en el sentido de las manecillas del reloj (LED1 → LED2 → LED3 → LED4 → LED1). La frecuencia de actualización del LED es proporcional al ciclo de trabajo de la salida y aumenta con el crecimiento del ciclo de trabajo. Al 100% del ciclo de trabajo, los 4 LEDs se iluminan en verde.
Cuadro de control CAN válido recibido en los últimos 100 ms. El modo de control especificado dio como resultado un ciclo de trabajo negativo en la salida del motor.	Los LEDs parpadean rojo en un patrón circular en sentido contrario a las manecillas del reloj (LED1 → LED4 → LED3 → LED2 → LED1). La frecuencia de actualización del LED es proporcional al ciclo de trabajo de la salida y aumenta con el crecimiento del ciclo de trabajo. Al 100% del ciclo de trabajo, los 4 LEDs se iluminan rojo.

Indicadores de color de la falla

Cuando se detecta una condición de fallo, el ciclo de trabajo de salida se reduce al 0% y una falla es señalada. La salida entonces permanece desactivada durante 3 segundos. Durante este tiempo los LEDs (LED1-4) a bordo se usan para indicar la condición de fallo. La condición de fallo se indica alternando entre los LEDs superiores (LED1 y LED2) e inferiores (LED3 y LED4) que se iluminan en rojo y se apagan. El color de los LEDs inferiores depende de qué fallas están actualmente activas. La siguiente tabla describe cómo el color de los LEDs inferiores corresponde con las fallas actualmente activas.

Color	Sobre Temperatura	Bajo voltaje
Verde	Encendido	Apagado
Azul	Apagado	Encendido
Cyan / Aqua	Encendido	Encendido

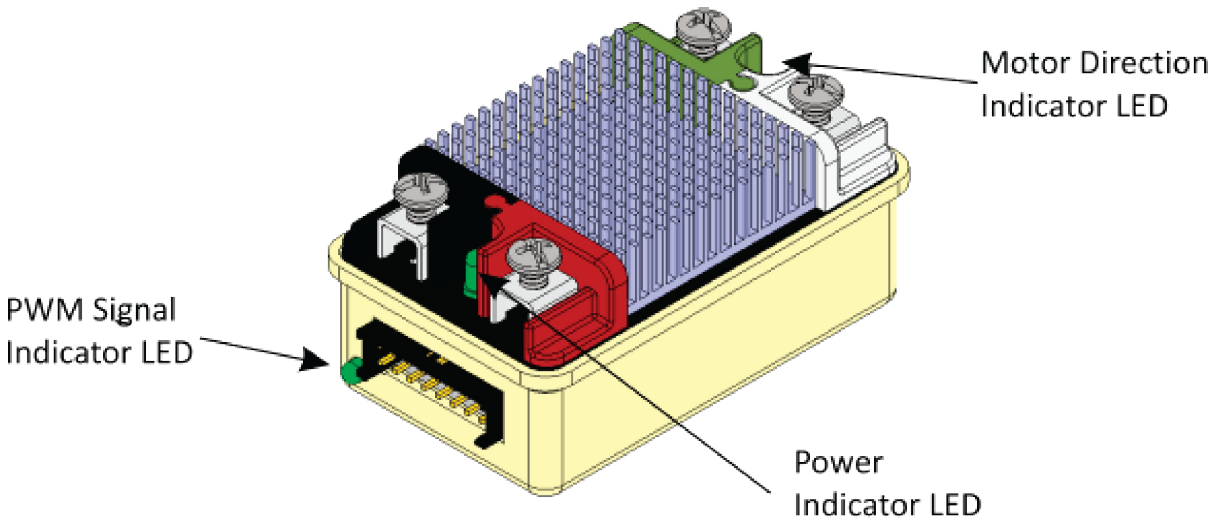
Modo Break/Coast

Cuando el LED central está apagado el dispositivo está operando en el modo coast. Cuando el LED central está iluminado el dispositivo está operando en el modo brake. El modo Brake / Coast se puede cambiar presionando el centro del triángulo y después soltando el botón.

36.5.18 Controlador de motor de veneno

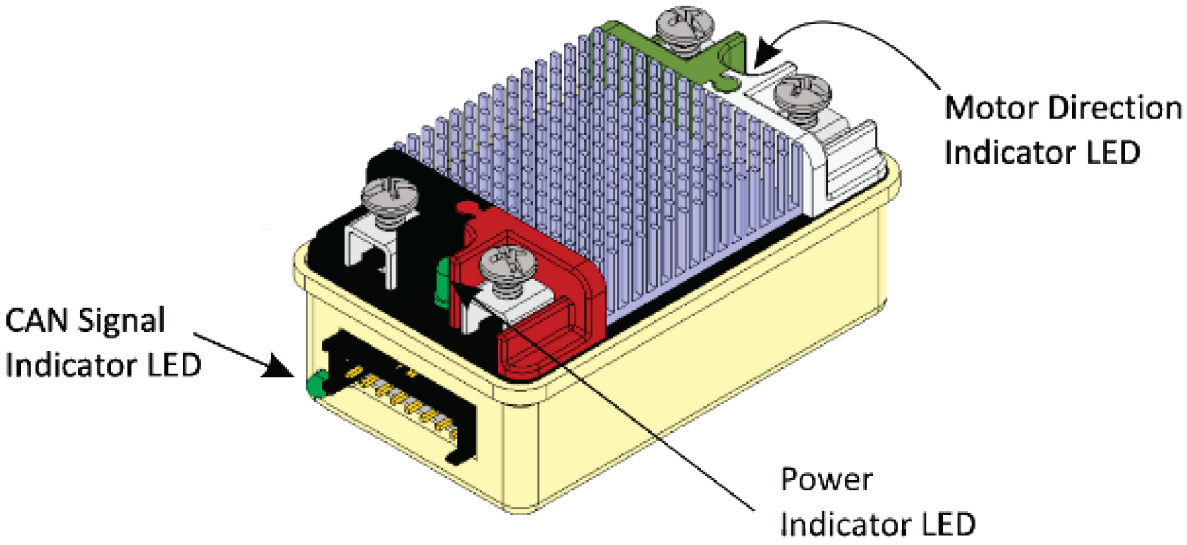
LED Pattern	Description
	Venom is initializing. This state should last less than 40ms after power up.
	The 'Identify Device' feature is active. This pattern in used to locate a particular Playing With Fusion device when multiple are installed on a robot. See the Motor Configuration section for more information
	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)



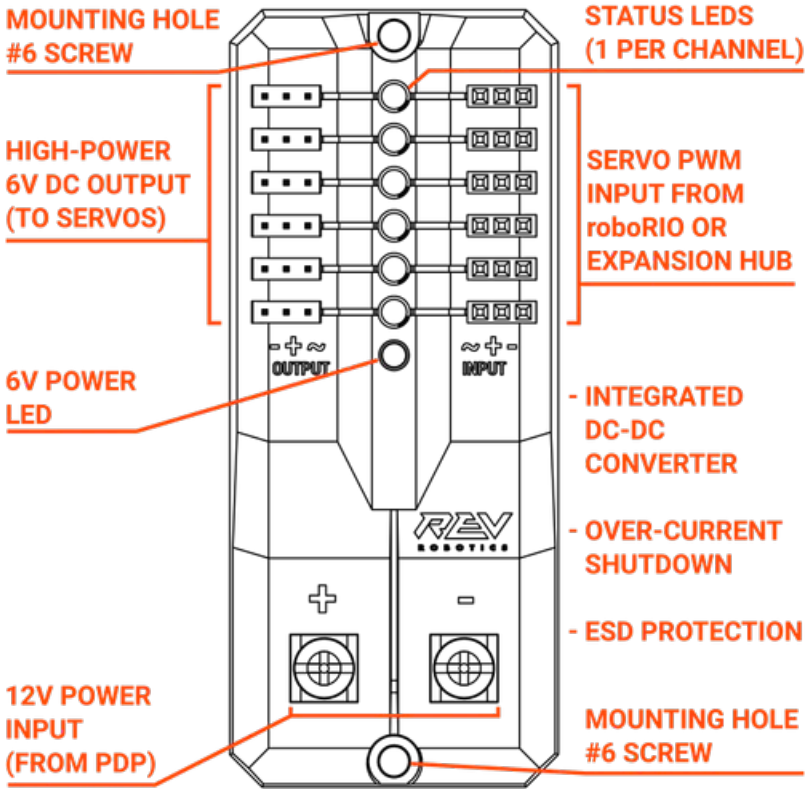
Power LED	Apagado	Energía no suministrada
	Rojo	Energía suministrada
Motor LED	Rojo	Dirección hacia adelante
	Verde	Dirección en reversa
PWM Señal LED	Rojo	No se detecta señal PWM válida
	Verde	Se detecta una señal PWM válida

36.5.20 Mindsensors SD540C (Bus CAN)



Power LED	Apagado	Energía no suministrada
	Rojo	Energía suministrada
Motor LED	Rojo	Dirección hacia adelante
	Verde	Dirección en reversa
LED Señal CAN	Parpadea rápidamente	No hay dispositivos CAN conectados
	Apagado	Conectado a la roboRIO y la driver station está abierta.

36.5.21 REV Robotics Módulo de Energía de Servo



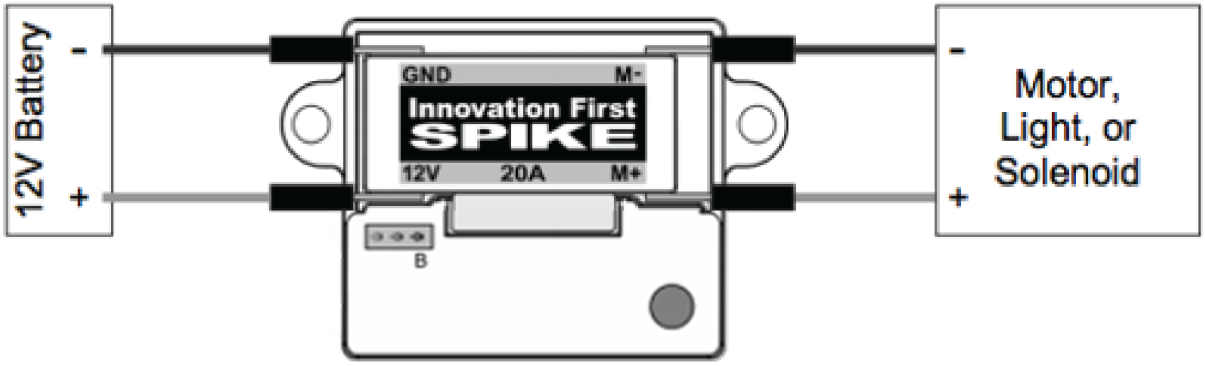
Status LEDs

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.

Estado	Patrón
No hay señal	Ámbar parpadeante
Señal de Izquierda/Reversa	Rojo
Señal Central/Neutral	Ámbar
Señal de Derecha/Adelante	Verde

- 6V Power LED off, dim or flickering with power applied = Over-current shutdown

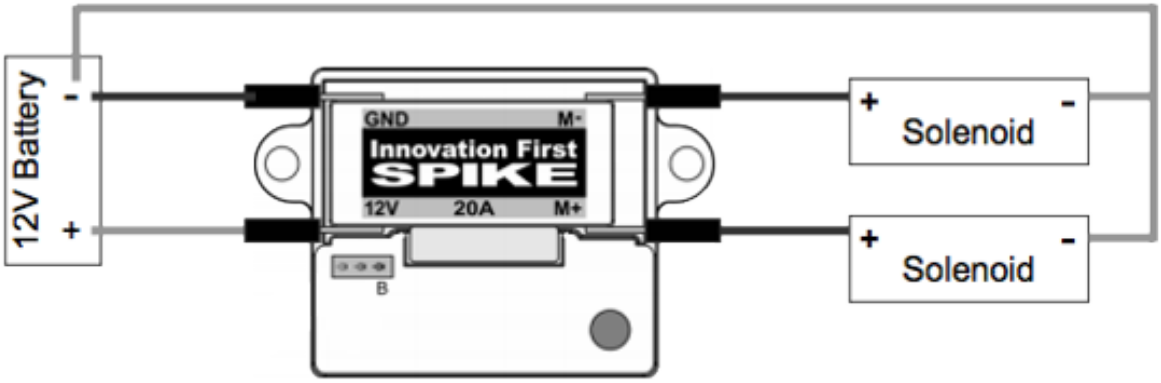
36.5.22 Revelador Spike configurado como motor, luz o interruptor solenoide.



Inputs		Outputs		Indicator	Motor Function
Forward (White)	**Reverse (Red)**	**M+**	**M-**		
Apagado	Apagado	GND	GND	Naranja	Condición de apagado/frenado (por defecto)
Encendido	Apagado	+12v	GND	Verde	El motor gira en una dirección
Apagado	Encendido	GND	+12v	Rojo	El motor gira en dirección opuesta
Encendido	Encendido	+12v	+12v	Apagado	Condición de apagado/frenado

Nota: "Condición de frenado" Se refiere a la parada dinámica del motor debido al cortocircuito de las entradas del motor. Esta condición no es opcional cuando se va a un estado desactivado.

36.5.23 Relevador Spike configurado para uno o dos solenoides.



Inputs		Outputs		Indicator	Motor Function
Forward (White)	**Reverse (Red)**	**M+**	**M-**		
Apagado	Apagado	GND	GND	Naranja	Ambos solenoides apagados (por defecto)
Encendido	Apagado	+12v	GND	Verde	El solenoide conectado a M+ está encendido
Apagado	Encendido	GND	+12v	Rojo	El solenoide conectado a M- está encendido
Encendido	Encendido	+12v	+12v	Apagado	Ambos solenoides encendidos

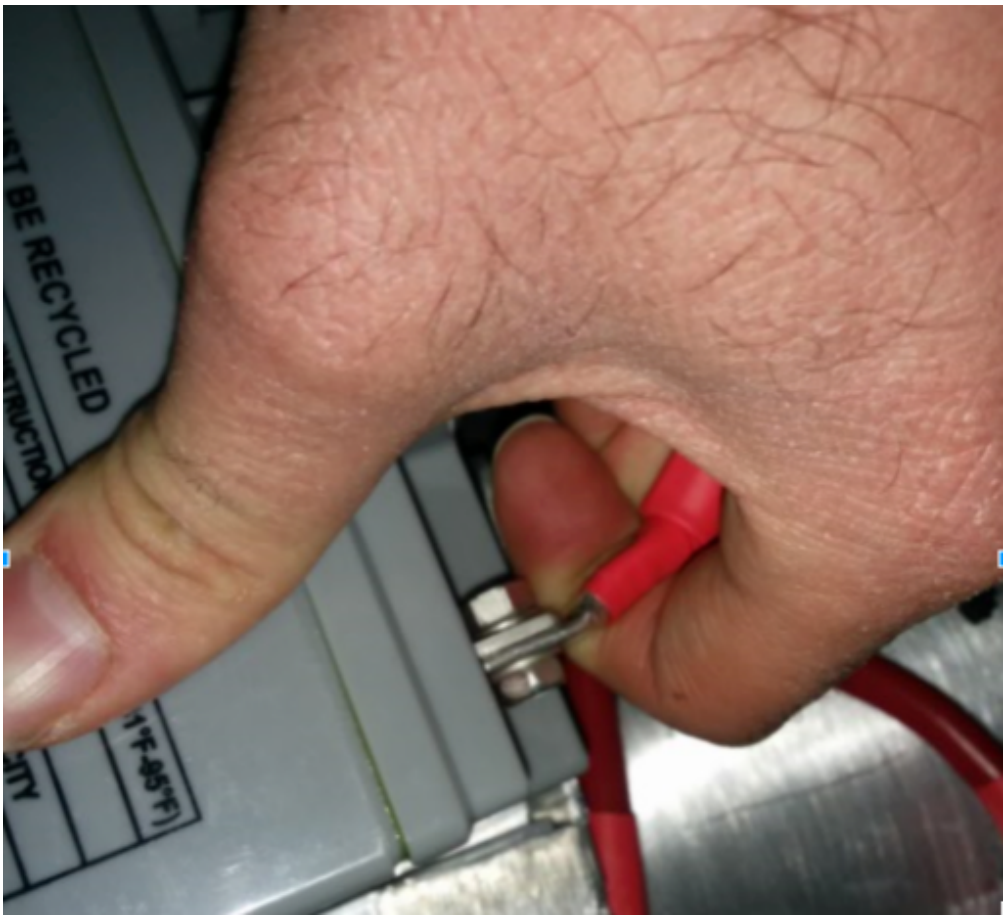
36.5.24 CANCoder Encoder

Color de LED	LED Brightness	CAN Bus detection	Magnet Field Strength	Description
Apagado	Apagado			CANCoder is not powered
Yellow/Gre	Bright			Device is in boot-loader. See user manual for more information.
Slow Red Blink	Bright	CAN bus has been lost		
Rapid Red Blink	Dim	CAN bus never detected since boot	Magnet is out of range (<25mT or >135mT)	
Rapid Yellow Blink			Magnet in range with slightly reduced accuracy (25-45mT or 75-135mT)	
Rapid Green Blink			Magnet in range (between 45mT - 75mT)	
Rapid Red Blink	Bright	CAN bus present	Magnet is out of range (<25mT or >135mT)	
Rapid Yellow Blink			Magnet in range with slightly reduced accuracy (25-45mT or 75-135mT)	
Rapid Green Blink			Magnet in range (between 45mT - 75mT)	

36.6 Soluciones para prevenir problemas del robot

Nota: En *FIRST*® Robotics Competition, los robots soportan mucha presión mientras son conducidos en el campo. Es importante asegurarse de que las conexiones estén bien sujetadas, las piezas estén atornilladas de forma segura en su lugar y que todo está sujetado para que un robot que este en constante movimiento no se rompa.

36.6.1 Revise las conexiones de la batería

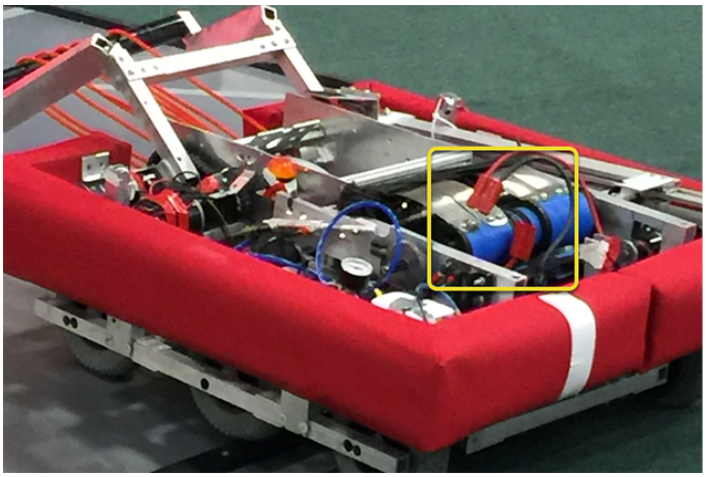


La cinta que debería cubrir la conexión de la batería en estos ejemplos se ha retirado para ilustrar lo que ocurre. En sus robots, las conexiones deberían estar cubiertas.

Mueva el conector del arnés de la batería. A menudo están sueltos porque los tornillos se aflojan, o a veces no está completamente atornillado. Sin embargo, sólo atraparás a los realmente malos porque a menudo la cinta eléctrica endurece la conexión hasta un punto en el que se siente rígida. Usar un voltímetro o un pico de batería ayudará con esto.

Aplique una fuerza considerable sobre el cable de la batería a 90 grados para tratar de mover la dirección del cable que sale de la batería, si tiene éxito la conexión no estaba lo suficientemente apretada para empezar y debe ser rehecha. Este *article* has more detailed battery information.

36.6.2 Asegurar la batería del robot



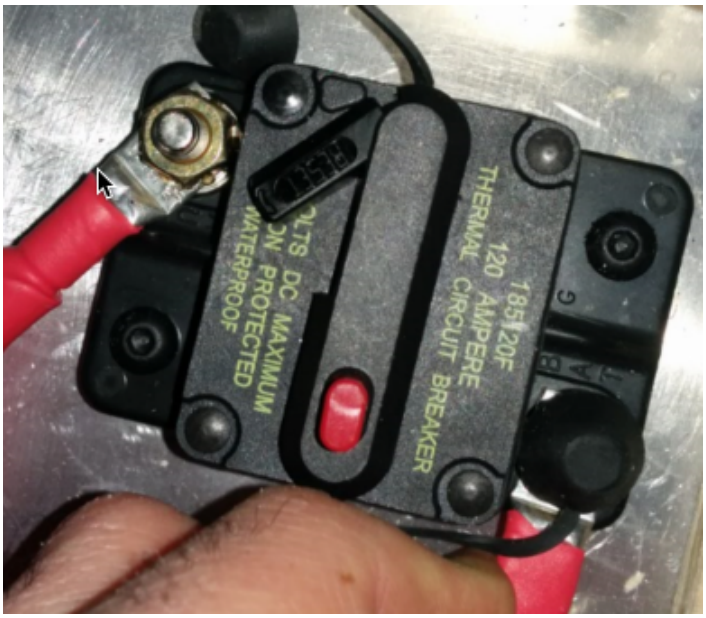
In almost every event we see at least one robot where a not properly secured battery connector (the large Anderson) comes apart and disconnects power from the robot. This has happened in championship matches on the Einstein and everywhere else. Its an easy to ensure that this doesn't happen to you by 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 Cómo asegurar el conector de la batería y los cables de alimentación principales

Un conector de la batería del lado del robot que esté suelto (el Anderson SB grande) puede permitir que los cables de alimentación principal sean tirados cuando se reemplaza la batería. Si los cables de alimentación principal están sueltos, ese «tirón» puede llegar hasta los terminales de engarce conectados al disyuntor de 120 amperios o al panel de distribución de energía (PDP), doblar el terminal y, con el tiempo, hacer que el extremo del terminal se rompa por la fatiga. Colocar un par de cintas de sujeción que unan los cables de alimentación principal al chasis y atornillar el conector de la batería del lado del robot puede evitar esto, así como facilitar la conexión de la batería.

36.6.4 Interruptor principal (disyuntor de 120 amperios)

Nota: Asegúrese de que las tuercas estén firmemente apretadas y que el interruptor esté unido a un elemento rígido.



Aplique una fuerte fuerza de torsión para intentar girar la orejeta engarzada. Si el terminal gira, la tuerca no está suficientemente apretada. Después de apretar la tuerca, vuelva a probar intentando girar el terminal.

La tuerca original tiene un cierre en estrella, que puede desgastarse con el tiempo: es posible que haya que revisarla cada cierto tiempo, especialmente si el conector de la batería del lado del robot no está unido al chasis.

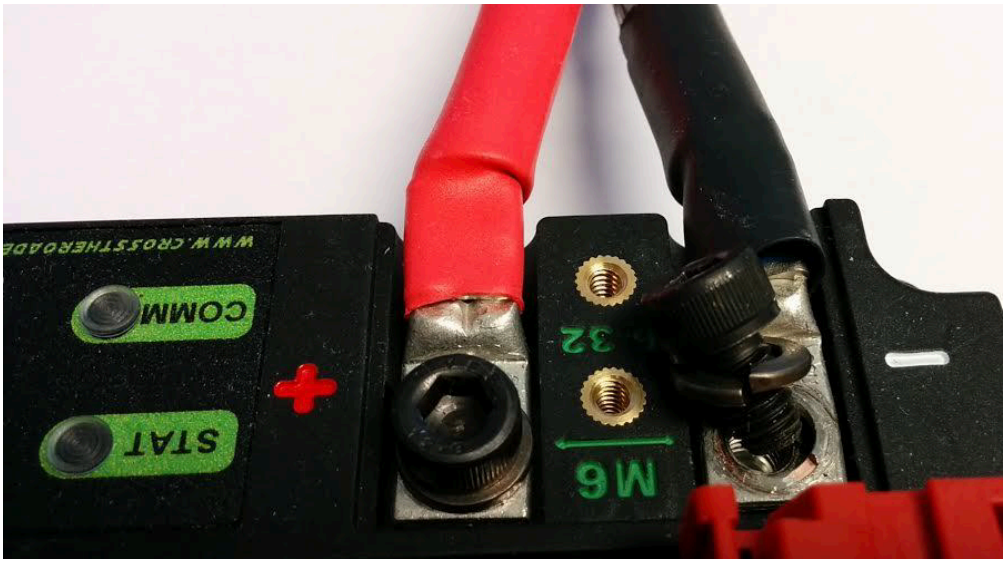
La tuerca suele tener una rosca relativamente poco común de 1/4-28: asegúrese de que es correcta si se sustituye la tuerca.

Debido a que el perno metálico sólo está moldeado en la caja, de vez en cuando se puede romper el perno. No se estrese, simplemente sustituya el conjunto.

Cuando se somete a varias temporadas de competencia, el disyuntor principal es susceptible de sufrir daños por fatiga a causa de las vibraciones y el uso, y puede empezar a abrirse con el impacto. Cada vez que se dispara la función de fusible térmico, puede ser progresivamen-

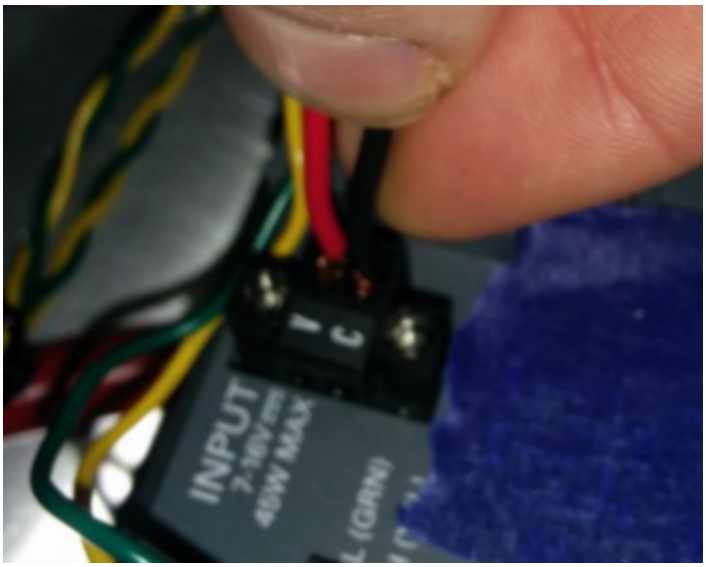
te más fácil que se dispare. Muchos equipos veteranos comienzan cada temporada con un disyuntor principal nuevo y llevan repuestos.

36.6.5 Panel de distribución de energía (PDP)



Asegúrate de que las arandelas de separación se colocaron bajo los tornillos del PDP, pero no es fácil de confirmar visualmente, y a veces no se puede. Puede comprobarlo quitando el estuche. También si aprietas los cables rojos y negros juntos, a veces puedes atrapar las conexiones perdidas.

36.6.6 Prueba de tirón



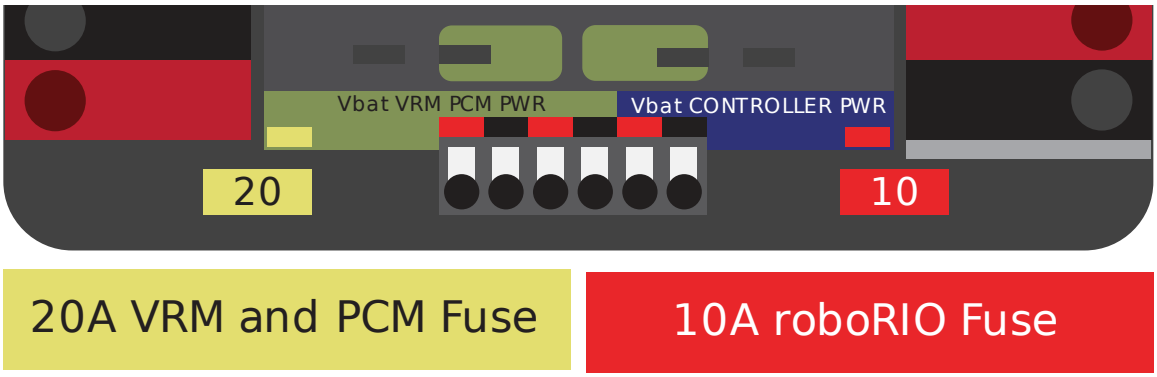
Los contactos de Weidmuller para la energía, la salida del compresor, el conector de energía del roboRIO y la energía de radio son importantes, para verificar estire las conexiones como se muestra. Asegúrese de que ninguna de las conexiones se separe.

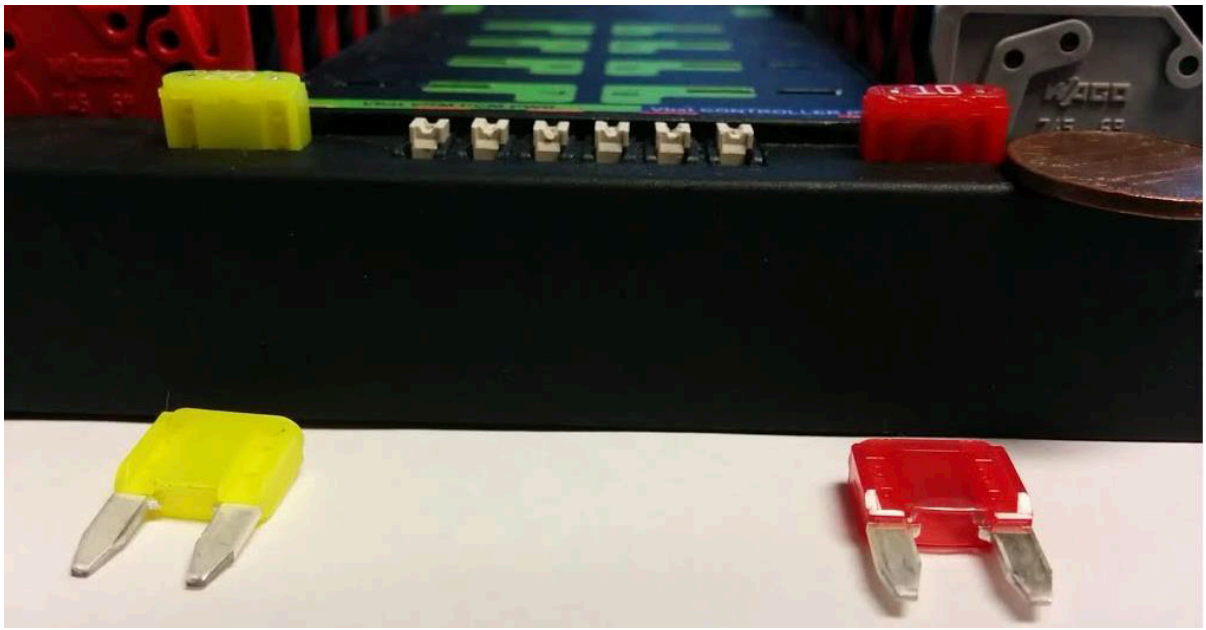
Busque los posibles o inminentes cortos con conexiones Weidmuller que estén cerca una de la otra, y que tengan longitudes de cable demasiado largas (cables que se pelan extra largos).

Los conectores de pala también pueden fallar debido a ajustes inadecuados, así que también hay que probarlos.

36.6.7 Fusibles de cuchilla

Asegúrate de colocar el fusible 20 A (amarillo) a la izquierda y el fusible 10 A (rojo) a la derecha.





Advertencia: Asegúrese de que los fusibles estén completamente colocados en los portafusibles. Los fusibles deben llegar al menos hasta la figura que se muestra a continuación (diferentes marcas de fusibles tienen diferentes longitudes de plomo). Debería ser casi imposible quitar el fusible con las manos (sin el uso de pinzas). Sí no se hace correctamente, el robot / radio puede presentar intermitentes problemas de conectividad.

Si usted puede quitar los fusibles a mano, entonces no están completamente dentro. Asegúrese de que estén completamente colocados en el PDP para que no se salgan durante la operación del robot.

36.6.8 Viruta de la roboRIO

La viruta son chips finos o limadura fina de piedra, metal u otro material producido por una operación de mecanizado. A menudo se deben realizar modificaciones en un robot mientras las piezas del sistema de control están en su lugar. La placa de circuito para el roboRIO está recubierta de forma conforme, pero eso no garantiza absolutamente que las virutas de metal no provoquen un cortocircuito en los rastros o componentes dentro de la caja. En este caso debe tener cuidado de que ninguno de los chips termine en el roboRIO o en cualquiera de los otros componentes. En particular, los 3 cabezales de alfiler expuestos son un lugar donde los chips pueden entrar en la caja. Un rápido vistazo a cada uno de los cuatro lados con una linterna suele ser suficiente para encontrar las áreas realmente malas de infiltración.

36.6.9 Radio Barrel Jack

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 Cable Ethernet

Si al cable RIO que va al cable de radio Ethernet le falta el clip que bloquea al conector, consiga otro cable. Este es un problema común que ocurrirá varias veces en cada competencia. Asegúrese de que los cables estén seguros. El clip a menudo se rompe, especialmente cuando se tira de el a través de un camino estrecho, se engancha en algo y luego se rompe.

36.6.11 Cables sueltos

Los cables deben apretarse, en particular el radio power y el cable Ethernet. Los cables radio power no tienen mucha fuerza de fricción y se caerán (incluso si es el barrel correcto) si se permite que el peso del cable flojo oscile libremente.

El cable de Ethernet también es bastante pesado, si se permite que oscile libremente, el clip de plástico pude no ser suficiente para sostener los conectores de pines Ethernet en el circuito.

36.6.12 Problemas de transmisión en el Pit

Más allá de la sacudida normal mientras que el robot está encendido y atado, se sugiere que se levante y se deje caer un lado del robot. Conducir en el campo, especialmente contra los defensores, será a menudo muy violento, y esto ayuda a asegurar que nada se caiga. Es mejor que el robot falle en los pits que en medio de un partido.

Cuando se hace esta prueba es importante estar conectado a un Ethernet y no a un USB, de lo contrario no se están probando todas las trayectorias.

36.6.13 Revise el Firmware y las versiones

Los inspectores de robot hacen esto, pero usted también debería hacerlo, ayude a los inspectores de robot y ellos lo apreciarán. Y garantiza que está corriendo con el código más reciente, corregido de errores. No querrá perder una partida por un software de sistema de control anticuado en su robot.

36.6.14 Revisión del Driver Station

A menudo se observan problemas con la Driver Station. Usted debe:

- SIEMPRE llevar el cargador de la laptop al campo, no importa lo buena que sea la batería, puede enchufarlo en el campo.
- Check the power and sleep settings, turn off sleep and hibernate, screen savers, etc.
- Turn off power management for USB devices (dev manager)
- Apagar la distribución de energía para los puertos Ethernet (dev manager).
- Apagar Windows defender.
- Apagar el firewall
- Cerrar todas las aplicaciones excepto DS/Dashboard cuando se esté en el campo.
- Verifique que no haya nada innecesario corriendo en la bandeja de aplicaciones en el menú de inicio (abajo a la derecha).

36.6.15 Herramientas útiles

Parece que nunca hay suficiente luz dentro de los robots, al menos no la suficiente para inspeccionar los puntos de conexión críticos, así que considere usar una linterna LED de mano para inspeccionar las conexiones de su robot. Están disponibles en Home Depot o en cualquier ferretería o tienda de autoservicio.

Una herramienta de WAGO es una buena herramienta para rehacer las conexiones de Weidmuller con cables trenzados. A menudo hago una para mostrar al equipo, y luego hago que hagan el resto con la herramienta de WAGO para presionar el botón blanco mientras insertan el cable trenzado. El ángulo de la herramienta WAGO hace que esto sea particularmente útil.

36.7 Conceptos básicos de la batería del robot

The power supply for an FRC® robot is a single 12V 18Ah SLA (Sealed Lead Acid) non-spillable battery, capable of briefly supplying over 180A and arcing over 500A when fully charged. The Robot Battery assembly includes the *COTS* battery, lead cables with contacts, and Anderson SB connector. Teams are encouraged to have multiple Robot Batteries.

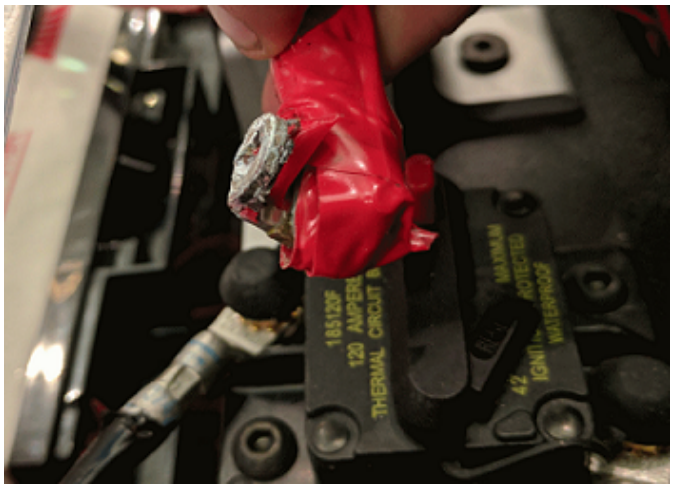
36.7.1 Batería COTS

The Robot Rules in the Game Manual specify a COTS non-spillable sealed lead acid battery meeting specific criteria, and gives examples of legal part numbers from a variety of vendors.

36.7.2 Seguridad y manejo de la batería

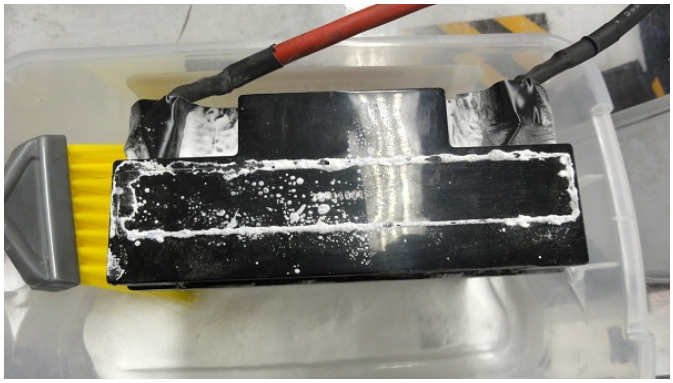
Una batería sana está **siempre** «encendida» y los terminales están **siempre** energizados. Si las polaridades se cortocircuitan -por ejemplo, si una llave inglesa o una lata de aerosol se caen y puentean el espacio entre dos terminales desnudos- toda la energía almacenada se liberará en un arco peligroso. Este riesgo impulsa una amplia gama de buenas prácticas, como cubrir los terminales en el almacenamiento, sólo descubrir y trabajar en un terminal o polaridad a la vez, mantener los contactos SB completamente insertados en los conectores, etc.

Do *NOT* carry a battery assembly by the cables, and always avoid pulling by them. Pulling on batteries by the cables will begin to damage the lugs, tabs, and the internal connection of the tab. Over time, fatigue damage can add up until the entire tab tears out of the housing! Even if it isn't clearly broken, internal fatigue damage can increase the battery internal resistance, prematurely wearing out the battery. The battery will not be able to provide the same amount of current with increased internal resistance or if the *connectors are loose*.



La caída de las baterías puede doblar las placas internas y causar problemas de rendimiento, crear abultamientos o incluso romper la carcasa de la batería. Mientras que la mayoría de las baterías FRC utilizan tecnología de vidrio absorbente [AGM] o gel para la seguridad y el rendimiento, cuando una célula se perfora todavía puede perder una pequeña cantidad de ácido de la batería. Esta es una de las razones por las que FIRST recomienda a los equipos tener un kit de derrame de baterías disponible.

Por último, algunos cargadores de baterías más antiguos sin funciones de «modo de mantenimiento» pueden *sobrecargar* la batería, lo que provoca la ebullición de parte del ácido de la misma.



Las baterías dañadas deben eliminarse de forma segura lo antes posible. Todas las tiendas que venden baterías SLA grandes, como las de los coches, deberían poder deshacerse de ellas por usted. Es posible que le cobren una pequeña cuota, o que le ofrezcan un pequeño «reembolso de la carga del núcleo», dependiendo de la ley de su estado.

Peligro: **NO** intente «reparar» las baterías dañadas o que no funcionen.

36.7.3 Construcción de la batería y herramientas

Cables de la batería

Los cables de la batería deben ser de cobre, con un tamaño mínimo (sección transversal) de 6 AWG (16mm2, 7 SWG) y una longitud máxima de 12», con código de colores para la polaridad, con un conector Anderson SB. Los cables de cobre estándar de 6AWG con cables de batería SB50 rosa/rojo suelen venir en el kit de piezas y los ofrecen los vendedores de FRC.

Cables conductores

Se permite el uso de cobre estañado, recocido o recubierto. No utilice CCA (aluminio revestido de cobre), aluminio u otro metal base que no sea cobre. El metal del conductor está normalmente impreso en el exterior del aislamiento con las otras clasificaciones del cable.

El tamaño del cable 6AWG es suficiente para casi todos los robots y se ajusta a los contactos estándar SB50. Un pequeño número de equipos adoptan tamaños de cable más grandes para obtener beneficios marginales de rendimiento.

Los cables de mayor número de hebras (que a veces se venden como «flexibles» o «para soldar») tienen un radio de curvatura más pequeño, lo que hace que sean más fáciles de enrutar, y un límite de fatiga más alto. No hay ningún requisito de número de hebras, pero tanto el 84/25 (cable de enganche «flexible» de 84 hebras) como el 259/30 (cable de «soldadura» de 259 hebras) serán *mucho* más fáciles de trabajar que el 19/0,0372 (cable de enganche de 19 hebras).

El aislamiento debe estar codificado por colores según el Manual del Juego: a partir de 2021, el cable de +12Vdc debe ser rojo, blanco, marrón, amarillo o negro con raya y el cable de tierra

(cable de retorno) debe ser negro o azul. No hay ningún requisito explícito de temperatura de aislamiento, pero cualquier aislamiento ennegrecido o dañado significa que el cable debe ser reemplazado: a mano, 105C es suficiente y más bajo funcionará para casi todos los robots. No hay ningún requisito de tensión de aislamiento, pero si el aislamiento es más fino, es mejor.

Conector SB

El conector Anderson SB puede ser el estándar SB50 Rosa/Rojo, u otro conector Anderson SB. Se recomienda encarecidamente a los equipos que utilicen el SB50 rosa/rojo para la interoperabilidad: los otros colores y tamaños de carcasas no se interpondrán, y no podrá tomar prestadas las baterías ni los cargadores.

Siga las instrucciones del fabricante para engarzar los contactos y montar los cables en los conectores Anderson SB. Un pequeño destornillador de cabeza plana puede ayudar a insertar los contactos (empuje sobre el contacto, no sobre el aislamiento del cable), o puede ayudar a desenganchar el pestillo interno si el contacto está en la ranura equivocada o al revés.

Tapones de la batería

Los terminales de compresión («crimp lugs») para lengüetas de batería de perno #10 (o M5) (~0,2» o ~5mm de diámetro de orificio) están disponibles en línea y a través de casas de suministros eléctricos, vendidos por los tamaños de cable aceptados en AWG (o mm2) y el diámetro del poste («tamaño del perno», «diámetro del orificio»). Los proveedores de gama alta también distinguen en sus catálogos de terminales entre el número de hilos estándar (~19) y el número de hilos flexibles (>80). Algunos proveedores también ofrecen terminales en ángulo recto, además de los estilos rectos más comunes. Siga las instrucciones del fabricante para engarzar los terminales.

Los terminales de tornillo son legales, pero no se recomiendan. Si utiliza terminales de tornillo, utilice un destornillador del tamaño adecuado para apretar el terminal. Compruebe con frecuencia el apriete de los terminales porque pueden aflojarse con el tiempo.

Conexión de los bornes de la batería al poste

Una tuerca y un tornillo #10 o M5 conectan el terminal del cable de la batería a la lengüeta de la batería.

Advertencia: La oreja y la lengüeta deben estar en contacto directo, cobre con cobre: no ponga ningún tipo de arandela que los separe.

Algunas baterías vienen con pernos de lengüeta en el paquete: pueden utilizarse, o sustituirse por pernos de aleación de acero más resistentes. Es una buena idea añadir una arandela de seguridad funcional, como una arandela de estrella #10 o un sistema de arandela nordlock, además de una tuerca de bloqueo de nylon («nylock»). Utilice sólo un tipo de arandela de seguridad en cada conexión. Aunque el fabricante proporcione arandelas de seguridad de anillo dividido en el paquete, no está obligado a utilizarlas.

Estas conexiones deben estar muy apretadas para que sean fiables. Cualquier movimiento de la lengüeta mientras está en funcionamiento puede interrumpir la alimentación del robot, lo que provoca reinicios del mismo y desconexiones de campo que duran 30 segundos o más.

Esta conexión también debe estar completamente cubierta para la seguridad eléctrica; la cinta eléctrica funcionará, pero se recomienda un termorretráctil que se ajuste a toda la conexión. Unas relaciones de contracción elevadas (mínimo 3:1, se recomienda 4:1) facilitarán la aplicación del termorretráctil. Se permite el uso de termorretráctiles con adhesivo. Asegúrese de cubrir *todo* el cobre. El termorretráctil debe ser «retocado» con cinta aislante si se ve algo de cobre.

Cargadores de baterías

Hay muchos buenos cargadores de baterías COTS «inteligentes» diseñados para baterías SLA de 12V, con una capacidad de 6A o menos por batería, con características de «modo de mantenimiento». Los cargadores de más de 6A no están permitidos en los pits de FRC.

Los cargadores utilizados en la competencia deben utilizar conectores Anderson SB. Conectar un cable de batería con conector SB COTS a los cables del cargador utilizando tuercas de cable de tamaño adecuado o terminales de tornillo es rápido y sencillo (asegúrese de cubrir cualquier cobre expuesto con termorretracción o cinta eléctrica). Los contactos del conector SB también están disponibles para tamaños de cable más pequeños, si el equipo tiene capacidad de engarce.

> **Advertencia:** Después de conectar el SB, compruebe dos veces las polaridades del cargador con un multímetro antes de enchufar la primera batería.

Algunos vendedores de FRC venden cargadores con conectores rojos SB50 preinstalados.

Herramientas de evaluación de baterías

Cargador de baterías

Si su cargador de baterías tiene un indicador de modo de mantenimiento, como un LED VERDE, puede utilizar ese indicador para saber si está LISTO. Algunos cargadores alternan periódicamente entre «CHARGING» y «READY». Este es un comportamiento de «mantenimiento», a veces asociado a que la batería se enfría y puede aceptar más carga.

Pantalla y registro de la Driver Station

Cuando el robot está enchufado y conectado a la Driver Station, el voltaje de la batería se muestra en el software de la NI Driver Station.

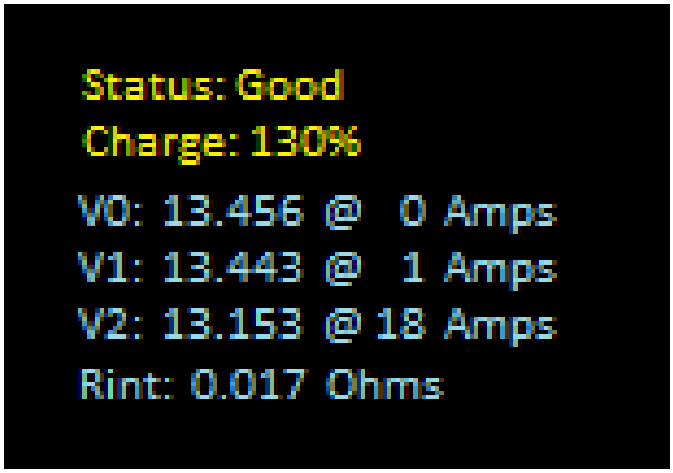
Después de terminar una sesión de conducción, puedes *revisar el voltaje de la batería en el Visor de Registros.*.

Voltímetro o **Multímetro** de mano

Una lectura de voltaje de las sondas en el conector SB de una batería desconectada le dará una instantánea de lo que es el Voc (voltaje en circuito abierto, o «voltaje de flotación») en el estado «sin carga». En general, el Voc no es un método recomendado para entender la salud de la batería: el voltaje en circuito abierto no es tan útil como la combinación de la resistencia interna y los voltajes en cargas específicas que proporciona un Probador de Carga (o Analizador de Baterías).

Prueba de carga

Un comprobador de carga de baterías puede utilizarse como una forma rápida de determinar el estado detallado de una batería. Puede proporcionar información como: tensión en carga abierta, tensión bajo carga, resistencia interna y estado de carga. Estas métricas pueden utilizarse para confirmar rápidamente que una batería está lista para un partido e incluso ayudar a identificar algunos problemas a largo plazo con la batería.



La resistencia interna ideal debe ser inferior a 0.015 Ohms. La especificación del fabricante para la mayoría de las baterías es de 0.011 Ohms. Si una batería supera los 0.020 Ohms, es una buena idea considerar la posibilidad de no utilizar esa batería para los partidos de competencia.

Si una batería muestra voltajes significativamente más bajos en las cargas de corriente de prueba más altas, es posible que no termine de cargarse o que deba retirarse.

36.7.4 Entendiendo los voltajes de las baterías

Una «batería de 12V» es cualquier cosa menos 12.0V.

Completamente cargada, una batería puede tener entre 12,7 y 13,5 voltios en circuito abierto (Voc). La tensión en circuito abierto se mide con *nada* conectado.

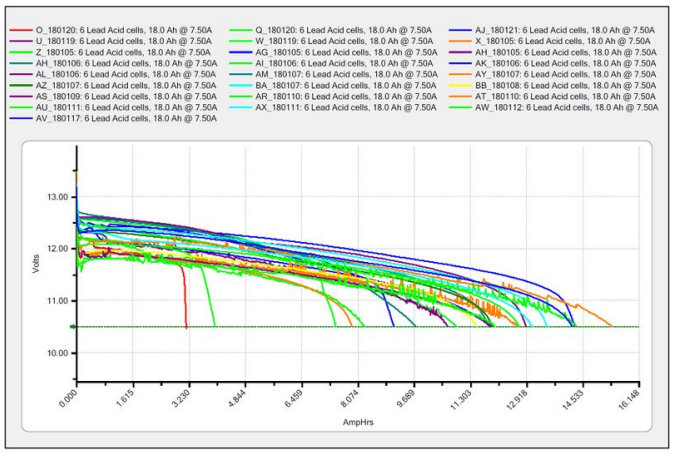
Una vez que se conecta una carga (como un robot), y fluye cualquier cantidad de corriente, el voltaje de la batería caerá. Por lo tanto, si comprueba una batería con un voltímetro y marca 13,2, y luego la conecta a su robot y la enciende, marcará un valor inferior, quizás 12,9 en la pantalla de la Driver Station. Estos números variarán con cada batería y robot específico, véase la caracterización más adelante. Una vez que el robot empiece a funcionar, tomará más corriente y el voltaje bajará más.

Las baterías que indican 12,5 V en un robot inactivo deben cambiarse y cargarse antes de un partido. Cambie siempre las baterías antes de que el robot empiece a alcanzar los umbrales de seguridad de caída de tensión (que aparecen en la pantalla de la Driver Station), ya que al entrar frecuentemente en rangos de baja tensión se corre el riesgo de dañar la batería de forma permanente; este comportamiento puede producirse en una variedad de estados de la voz dependiendo de la salud de la batería, del fabricante de la misma y del diseño del robot. El estado de carga de la batería debe mantenerse por encima del 50% para su longevidad.

El voltaje y la corriente de la batería también dependen de la temperatura: las baterías frías son baterías felices.

Caracterización de la batería

Se puede utilizar un analizador de baterías para realizar una inspección detallada y una comparación del rendimiento de la batería.



Proporcionará gráficos del rendimiento de la batería a lo largo del tiempo. Esta prueba requiere un tiempo considerable (aproximadamente dos horas), por lo que es menos adecuada para realizar pruebas durante la competición. Se recomienda realizar esta prueba en cada batería cada año para controlar y seguir su rendimiento. Esto determinará cómo debe utilizarse: en los partidos, en los entrenamientos, en las pruebas o en la eliminación.

Con la carga de prueba estándar de 7,5 amperios, las baterías de competencia deberían tener una capacidad de 11,5 amperios hora como mínimo. Todo lo que sea menos que eso solo debería usarse para prácticas u otros casos de uso menos exigentes.

Longevidad de la batería

Una batería tiene una vida útil de unos 1200 ciclos de carga/recarga normales. Las altas corrientes necesarias para un partido de FRC reducen esa vida útil a unos 400 ciclos. Estos ciclos están pensados para una descarga relativamente baja, de unos 13,5 a 12 o 12,5 voltios. Los ciclos profundos de la batería (que se descargue por completo) la dañarán.

Batteries last the longest if they are kept fully charged when not in use, either by charging regularly or by use of a maintenance charger. Batteries drop roughly 0.1V every month of non-use.

Las pilas deben mantenerse alejadas del calor y el frío extremos. Por lo general, esto significa almacenar las baterías en un área con clima controlado: un armario de clase suele estar bien, un contenedor de transporte en un aparcamiento es más arriesgado.

36.7.5 Mejores prácticas para las baterías

- Utilice sólo una batería cargada para los partidos de competencia. Si se encuentra en una situación en la que se ha quedado sin baterías cargadas, ¡pide ayuda a un equipo veterano! Nadie quiere ver un robot muerto en el campo (*brownout*) debido a una batería mala o sin cargar.
- Se recomienda encarecidamente a los equipos que utilicen herramientas adecuadas y prácticas estrictas de control de calidad para los procesos de engaste (pida ayuda a los equipos veteranos locales o a un electricista comercial), o que utilicen cables de batería fabricados por el proveedor.
- Espere a que las baterías se enfríen después del partido antes de recargarlas: la carcasa no debe estar caliente al tacto, quince minutos suelen ser suficientes.
- Los equipos deberían considerar la posibilidad de comprar varias pilas nuevas cada año para ayudar a mantener sus baterías frescas. Los partidos de eliminación pueden requerir muchas pilas y puede que no haya tiempo suficiente para recargarlas.

- Un cargador de baterías de varios bancos permite cargar más de una batería a la vez. Muchos equipos construyen un carro de robot para sus baterías y cargadores, lo que permite un fácil transporte y almacenamiento.
- Es una buena idea identificar permanentemente cada batería con al menos: número de equipo, año y un identificador único.
- Teams may also want to use something removable (stickers, labeling machine etc.) to identify what that battery should be used for based on its performance data and when the last analyzer test was run.

- El uso de banderas de batería (un trozo de plástico colocado en el conector de la batería) es una forma habitual de indicar que una batería se ha cargado. Las banderas para baterías también pueden imprimirse fácilmente en 3D.
- Se pueden adquirir asas para los contactos del SB50 o imprimirlas en 3D para evitar que se tire de los cables al conectar o desconectar las baterías. No utilice estas asas para soportar el peso de la batería.

- Algunos equipos cosen correas para el transporte de la batería a partir de viejos cinturones de seguridad u otro tipo de nylon plano que se ajusta alrededor de la batería para ayudar a evitar el transporte por los cables.

- Las abrazaderas de borde de las bridas se pueden utilizar con terminales de engarce de 90 grados para aliviar la tensión de los cables de la batería.

37 Tutoriales sobre hardware

Nota: Las páginas de esta sección de la documentación contienen vídeos que solo se pueden ver desde la versión web de la documentación.

37.1 Motores para aplicaciones de robótica

Una de las decisiones de diseño más importantes con las que los equipos tienen que lidiar es con la selección y el diseño de los sistemas motorizados de su robot. A menudo se elige el motor incorrecto para un diseño particular, lo que hace que se reduzca el rendimiento y, a veces incluso peor, los motores fallan por un consumo excesivo de corriente. En esta serie de videos, el profesor Ken Stafford del WPI muestra cómo funcionan los motores, cómo diseñar sistemas para que funcionen al máximo rendimiento y un diseño de muestra para un sistema del robot.

37.2 Detección y Sensores

Sin sensores y detección los robots son solamente vehículos controlados por radio. Los sensores permiten a los robots comprender el funcionamiento interno de los sistemas mecánicos de los robots, así como la capacidad de interactuar con el entorno que los rodea. En estos videos el profesor Craig Putnam del WPI describe una serie de clases de sensores, cómo se utilizan y proporciona consejos sobre qué sensores son los mejores para sus aplicaciones.

37.3 Neumática

La neumática es un dispositivo de accionamiento a menudo infrautilizado que puede ser utilizado en los robots. La neumática tiene muchas ventajas sobre el uso de motores. En este vídeo el Profesor Ken Sttafford describe las características de la neumática, las aplicaciones con los robots y el cálculo del sistema de tamaño adecuado para una aplicación.

37.4 Transmisión de Potencia

Hand in hand with choosing the correct motors for an application is transmitting that motor power to the place it's needed. Using gears or chains and sprockets are two effective ways of matching the motor power to the application being driven. In this video, WPI Robotics Engineering PhD student Michael Delph talks about power transmission, including choosing correct gear or chain and sprocket ratios to get the maximum performance from your robot design.

38 Sensores

38.1 Descripción general del sensor - Hardware

Nota: This section covers sensor hardware, not the use of sensors in code. For a software sensor guide, see *Descripción general del sensor- software*.

Para que esto sea efectivo, con frecuencia es vital que los robots puedan recopilar información sobre su entorno. Los dispositivos que proporcionan información al robot sobre el estado de su entorno se denominan «sensores». Hay una gran variedad de sensores disponibles para los equipos de FRC®, para medir todo, desde el posicionamiento en el campo hasta la orientación del robot y el posicionamiento del motor/mecanismo. Hacer uso de sensores es una habilidad absolutamente crucial para el éxito en el campo; si bien la mayoría de los juegos de FRC tienen tareas que puede realizar un robot «ciego», los mejores robots dependen en gran medida de los sensores para realizar las tareas del juego de la manera más rápida y confiable posible.

Además, los sensores pueden ser extremadamente importantes para la seguridad de los robots - muchos mecanismos de los robots son capaces de romperse a sí mismos si se usan incorrectamente. Los sensores proporcionan una protección contra esto, permitiendo a los robots, por ejemplo, deshabilitar un motor si un mecanismo está contra un dura parada.

38.1.1 Tipos de sensores

Los sensores utilizados en el FRC pueden clasificarse en general de dos maneras diferentes: por su función y por su protocolo de comunicación. La primera categorización es relevante para el diseño del robot; la segunda para el cableado y la programación.

Sensores por función

Los sensores pueden proporcionar información sobre una variedad de aspectos diferentes del estado del robot. Las funciones de los sensores comunes al FRC incluyen:

- *Proximity switches*
  - Interruptores de proximidad mecánicos («interruptores de límite»)
  - Los interruptores magnéticos de proximidad
  - Interruptores de proximidad inductivos
  - Interruptores de proximidad fotoeléctricos
- Sensores de distancia
  - *Ultrasonic sensors*
  - *Triangulating rangefinders*
  - *LIDAR*
- Sensores de rotación del eje
  - *Encoders*
  - *Potentiometers*
- *Accelerometers*
- *Gyroscopes*

Sensores por Protocolo de Comunicación

Para que un sensor sea útil, debe ser capaz de «hablar» con el roboRIO. Hay varios métodos principales por los cuales los sensores pueden comunicar sus lecturas al roboRIO:

- *Analog input*
- *Digital input*
- *Serial bus*

En general, el apoyo a los sensores que se comunican a través de entradas analógicas y digitales es sencillo, mientras que la comunicación a través de un bus serial puede ser más complicada.

38.2 Entradas analógicas - Hardware

Nota: This section covers analog input hardware. For a software guide to analog inputs, see *Entradas análogas - Software*.

Una señal análoga es una señal cuyo valor puede encontrarse en cualquier lugar de un intervalo continuo. Esto se encuentra marcado en un contraste a *digital signal*, que puede tomar solo uno de varios valores discretos. Los puertos de entrada analógica del roboRIO permiten la medición de señales analógicas con valores de 0 V a 5 V.

En la práctica, no hay forma de medir una señal analógica «true» con un dispositivo digital como una computadora (como el roboRIO). En consecuencia, las entradas analógicas se miden en realidad como una señal digital de 12-bits - sin embargo, esta es una resolución bastante alta [1] _.

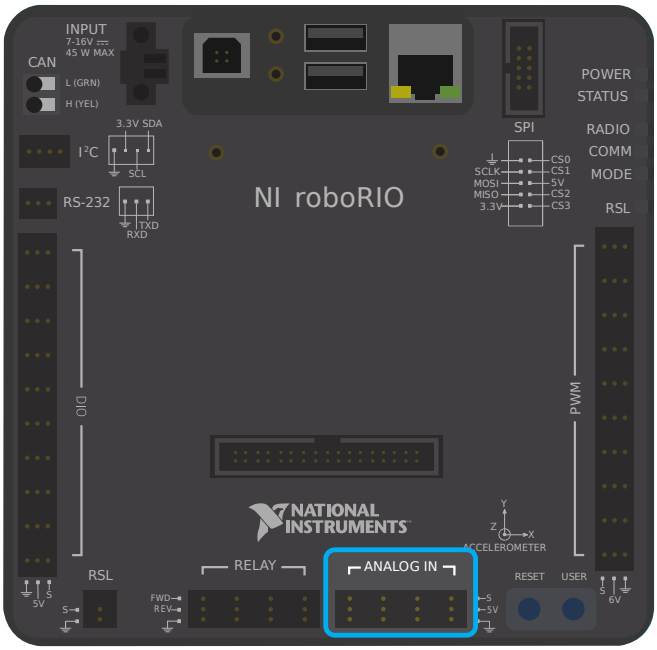
Las entradas analógicas se utilizan normalmente (¡pero no siempre!) Para sensores cuyas medidas varían continuamente en un rango, como *ultrasonic rangefinders* y *potentiometers*, ya que pueden comunicarse emitiendo un voltaje proporcional a sus medidas.

38.2.1 Conexión a puertos de entrada analógica roboRIO

Nota: Hay cuatro entradas analógicas adicionales disponibles a través del puerto de expansión «MXP». Para usarlas, se necesita una placa de ruptura de algún tipo que se conecte al MXP.

Advertencia: Consulte siempre las especificaciones técnicas del sensor que está utilizando *antes* de cablear el sensor, para asegurarse de que se está conectando el cable correcto a cada pin. Si no lo hace, puede dañar el sensor o el RIO.

Advertencia: *Nunca* conectes directamente el pin de energía a el pin de tierra en ningún puerto del roboRIO! Esto activará las características de protección del roboRIO y puede resultar en un comportamiento inesperado.



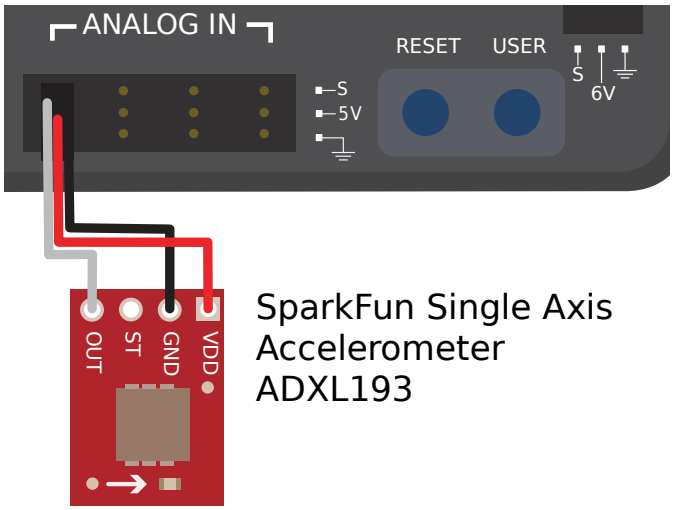
El roboRIO tiene 4 puertos de entrada analógicos incorporados (numerados 0-3), como se ve en la imagen de arriba. Cada puerto tiene tres pines - señal («S»), potencia («V»), y tierra (»□»). Los pines de «potencia» y «tierra» se utilizan para alimentar los sensores periféricos que se conectan a los puertos de entrada analógica - hay una diferencia de potencial constante de 5V entre los pines de «potencia» y «tierra»[2]. El pin de la señal es el pin en el que se mide realmente la señal.

Conectar un sensor a un solo puerto de entrada analógica

Nota: Algunos sensores (como *potentiometers*) pueden tener conexiones de energía y de tierra intercambiables.

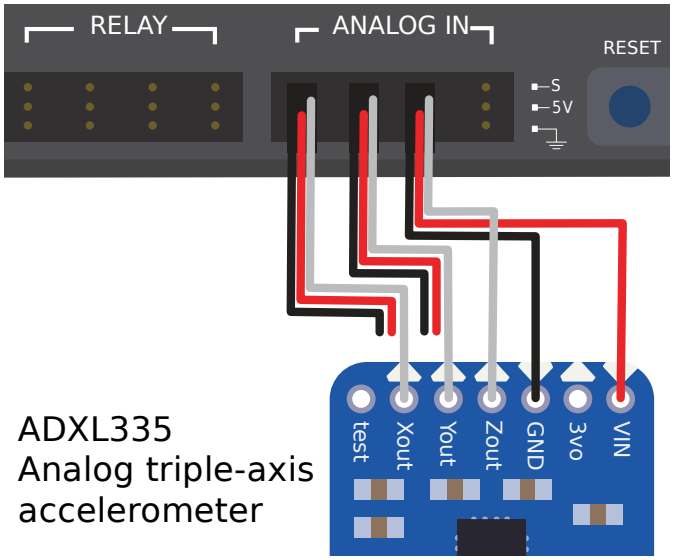
La mayoría de los sensores que se conectan a los puertos de entrada analógicos tendrán tres cables -señal, potencia y tierra- que se corresponden precisamente con los tres pines de los puertos de entrada analógicos. Deben ser conectados en concordancia.

[2] Todos los pines de energía están en realidad conectados a un solo carril, al igual que todos los pines de tierra - no hay necesidad de utilizar los pines de energía/tierra correspondientes a un pin de señal determinado.



Conectar un sensor a múltiples puertos de entrada analógica

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.



38.2.2 Pie de nota

38.3 Potenciómetros análogos - Hardware

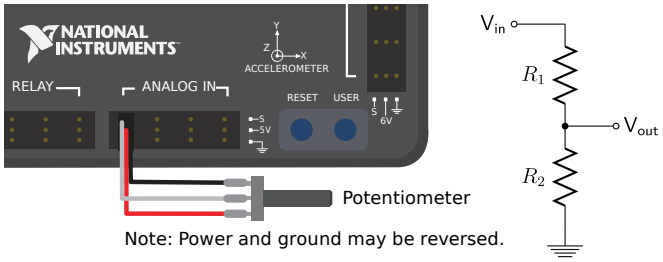
Nota: This section covers analog potentiometer hardware. For a software guide to analog potentiometers, see *Potenciómetros analógicos - Software*.

Advertencia: Los potenciómetros generalmente tienen un rango de recorrido limitado mecánicamente. Los usuarios deben tener cuidado de que sus mecanismos no giren sus potenciómetros más allá de su recorrido máximo, ya que esto dañará o destruirá el potenciómetro.

Aparte de *quadrature encoders*, otra forma común de medir la rotación en los robots FRC® es con potenciómetros analógicos. Un potenciómetro es simplemente una resistencia variable: a medida que gira el eje del potenciómetro, la resistencia cambia (generalmente de forma lineal). Colocando esta resistencia en un divisor de voltaje permite al usuario medir fácilmente la resistencia midiendo el voltaje a través del potenciómetro, que luego se puede usar para calcular la posición de rotación del eje.

38.3.1 Cableado de un potenciómetro analógico

Como lo sugieren los nombres, los potenciómetros analógicos se conectan a los puertos *analog input* de la roboRIO. Sin embargo, para entender exactamente cómo cablear potenciómetros, es importante entender sus circuitos internos.



La imagen de arriba a la izquierda muestra un potenciómetro típico. Hay tres pines, al igual que en las entradas analógicas de RIO. El pin del medio es el pin de señal, mientras que los pines exteriores pueden ser *el que sea* de alimentación o de tierra.

Como se mencionó anteriormente, un potenciómetro es un divisor de voltaje, como se muestra en el diagrama de circuito de la derecha. A medida que gira el eje del potenciómetro, las resistencias R1 y R2 cambian; sin embargo, su suma permanece constante [1] _. Por lo tanto, el voltaje en todo el potenciómetro permanece constante (para el roboRIO, esto sería 5 voltios), pero el voltaje entre el pin de señal y el pin de voltaje o tierra varía linealmente a medida que gira el eje.

Dado que el circuito es simétrico, es reversible; esto permite al usuario elegir en qué extremo del recorrido el voltaje medido es cero y en qué extremo es de 5 voltios. Para invertir la direccionalidad del sensor, ¡simplemente se puede conectar al revés! Asegúrese de verificar la direccionalidad de su potenciómetro con un multímetro para asegurarse de que esté en la orientación deseada antes de soldar los cables a los contactos.

38.3.2 Pie de nota

38.4 Entradas digitales - Hardware

Nota: This section covers digital input hardware. For a software guide to digital inputs, see *Entradas digitales - Software*.

Una señal digital es una señal que puede estar en uno de varios estados discretos. En la gran mayoría de los casos, la señal es el voltaje en un cable y solo hay dos estados para una señal digital - alto o bajo (también denotado 1 y 0, o verdadero y falso, respectivamente).

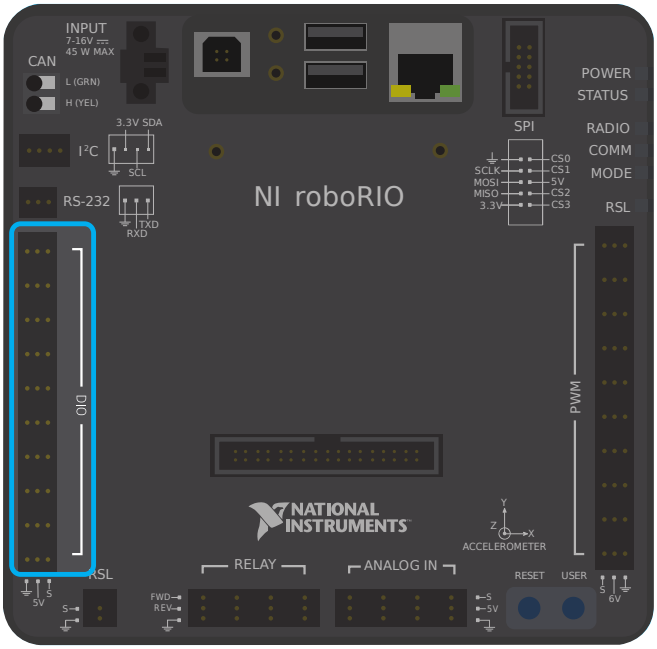
Los puertos de entrada y salida digitales integrados de roboRIO (o puertos «DIO») funcionan en 5V, por lo que «alto» corresponde a una señal de 5V y «bajo» a una señal de 0V [1] _ [2] _.

38.4.1 Conexión a los puertos DIO de roboRIO

Nota: Los puertos DIO adicionales están disponibles a través del puerto de expansión «MXP». Para usarlos, se necesita una placa de conexión de algún tipo que se conecte al MXP.

Advertencia: Consulte siempre las especificaciones técnicas del sensor que está utilizando *antes* de cablear el sensor, para asegurarse de que se está conectando el cable correcto a cada pin. Si no lo hace, puede dañar el dispositivo.

Advertencia: ¡**Nunca** conecte directamente el pin de alimentación al pin de tierra en cualquier puerto del roboRIO! Esto activará funciones de protección en roboRIO y puede resultar en un comportamiento inesperado.



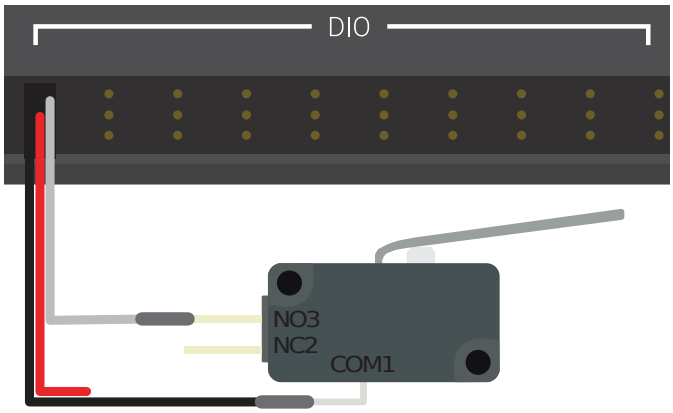
El roboRIO tiene 10 puertos DIO integrados (numerados del 0-9), como se ve en la imagen de arriba.Cada puerto tiene tres pines - señal («S»), potencia («V») y tierra (»**|tierra|**»). Los pines de «poder» y «tierra» se utilizan para alimentar los sensores periféricos que se conectan a los puertos DIO - hay una diferencia de potencia constante de 5 V entre los pines de «poder» y «tierra» [3] _ - el pin de «poder» corresponde al estado «alto» (5 V) y el de «tierra» a «bajo» (0 V). El pin de señal es el pin en el que se mide realmente la señal (o, cuando se usa como salida, el pin que envía la señal).

All DIO ports have built-in «pull-up» resistors between the power pins and the signal pins - these ensure that when the signal pin is «floating» (i.e. is not connected to any circuit), they consistently remain in a «high» state.

Conexión de un conmutador simple a un puerto DIO

El dispositivo más simple que se puede conectar a un puerto DIO es un conmutador (como un *limit switch*). Cuando un interruptor está conectado correctamente a un puerto DIO, el puerto leerá «alto» cuando el circuito está abierto y «bajo» cuando el circuito está cerrado.

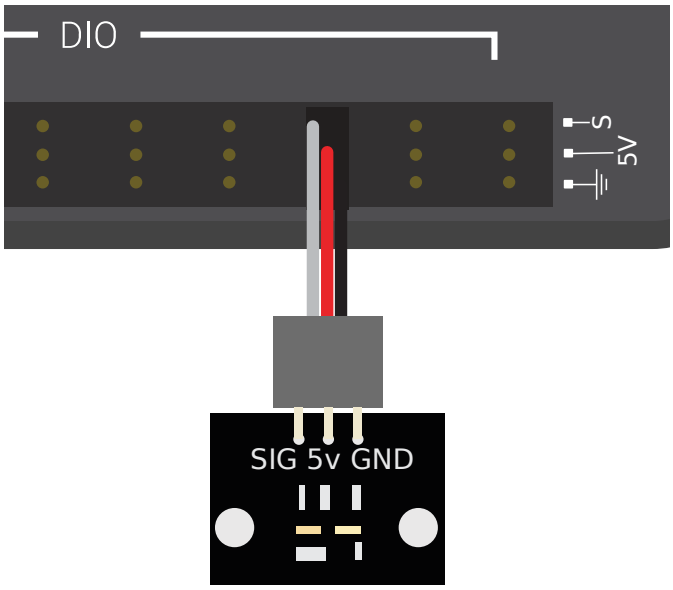
Un simple interruptor no necesita poder y, por lo tanto, solo tiene dos cables. Los interruptores deben cablearse entre la *señal* y los pines de *tierra* del puerto DIO. Cuando el circuito del interruptor está abierto, el pin de señal flotará y la resistencia pull-up se asegurará de que se lea «alto». Cuando el circuito del interruptor está cerrado, se conectará directamente al riel de tierra y, por lo tanto, leerá «bajo».



Limit Switch or Micro Switch

Conectando un sensor energizado a un puerto DIO

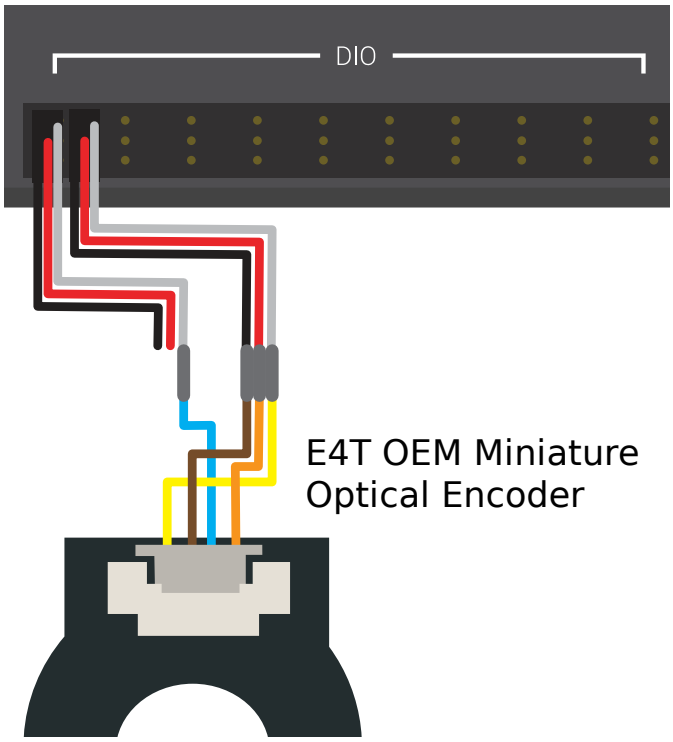
Muchos sensores digitales (como la mayoría de los interruptores de proximidad sin contacto) requieren energía para funcionar. Un sensor energizado generalmente tendrá tres cables - señal, poder y tierra. Estos deben estar conectados a los pines correspondientes del puerto DIO.



WCP Hall Effect Sensor

Conexión de un sensor que utiliza varios puertos DIO

Algunos sensores (así como *quadrature encoders*) es posible que deba conectarse a varios puertos DIO para funcionar. En general, estos sensores solo requerirán un solo poder y un solo pin de tierra - solo se necesitará el pin de señal de los puerto(s) adicionales.



38.4.2 Notas al pie

38.5 Interruptores de proximidad: hardware

Nota: This section covers proximity switch hardware. For a guide to using proximity switches in software, see *Entradas digitales - Software*.

Una de las tareas de detección más comunes en un robot es detectar cuando un objeto (ya sea un mecanismo, una pieza de juego o un elemento de campo) se encuentra a una cierta distancia de un punto conocido del robot. Este tipo de detección se logra mediante un «interruptor de proximidad».

38.5.1 Operación del interruptor de proximidad

Los interruptores de proximidad son interruptores: operan un circuito entre un estado «abierto» (en el que *no hay* conectividad a través del circuito) y uno «cerrado» (en el que *hay*). Por lo tanto, los interruptores de proximidad generan una señal digital y, en consecuencia, casi siempre están conectados a los puertos *digital input* del roboRIO.

Los interruptores de proximidad pueden ser ya sea «abiertos normalmente», en el que al activar el interruptor se cierra el circuito, o «cerrados normalmente», en el que al activar el interruptor se abre el circuito. Algunos interruptores ofrecen ambos los AB y los CN circuito conectado al mismo interruptor. En la práctica, la diferencia efectiva entre un AB y los CN interruptor es el comportamiento del sistema en caso en que el cableado del interruptor falle, ya que un fallo en el cableado casi siempre provocará un circuito abierto. Los interruptores AB suelen ser «más seguros», ya que un fallo en el cableado hace que el sistema se comporte como si el interruptor estuviera presionado - como los interruptores se utilizan a menudo para evitar que un mecanismo se dañe a sí mismo, esto mitiga la posibilidad de que se dañe el mecanismo en el caso de un fallo de cableado.

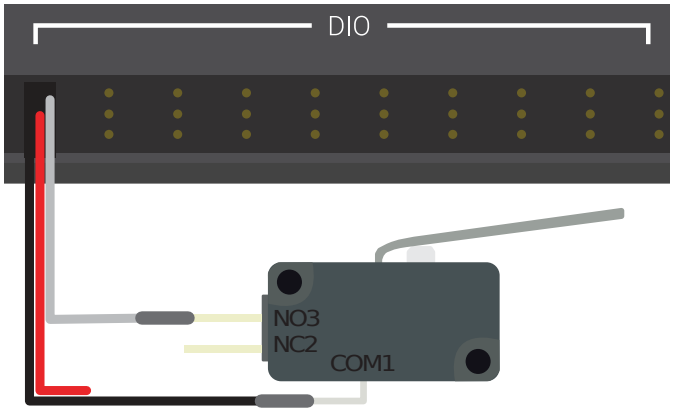
Las entradas digitales en el roboRIO tienen resistencias pull-up que harán que la entrada sea alta (valor 1) cuando el interruptor está abierto, pero cuando el interruptor se cierra, el valor pasa a 0 ya que la entrada ahora está conectada a tierra.

38.5.2 Tipos de interruptores de proximidad

There are several types of proximity switches that are commonly used in FRC®:

- *Interruptores mecánicos de proximidad («interruptores de límite») _*
- *Interruptores de proximidad magnéticos*
- 'Interruptores de proximidad inductivos`_
- *Interruptores de proximidad fotoeléctricos*
- *Interruptores de proximidad de tiempo de vuelo*

Interruptor de Proximidad Mecánica («Interruptor de Límite»)



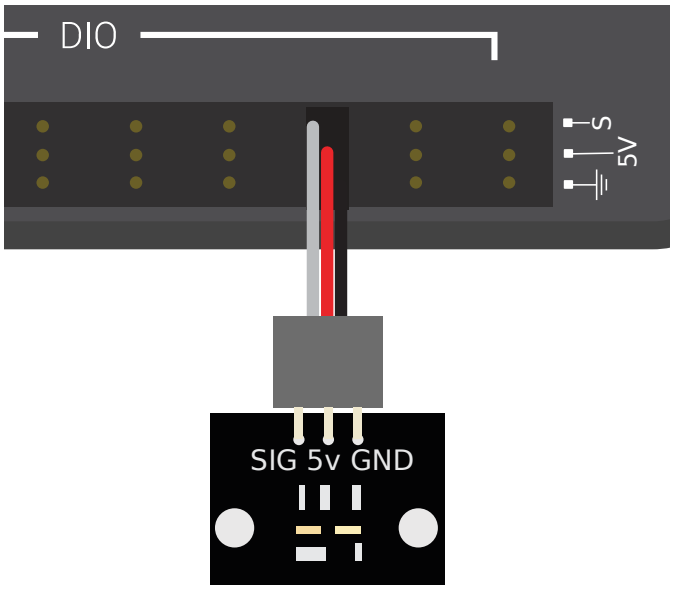
Limit Switch or Micro Switch

Mechanical proximity switches (more commonly known as «limit switches») are probably the most commonly used proximity switch in FRC, due to their simplicity, ease-of-use, and low cost. A limit switch is quite simply a switch attached to a mechanical arm, usually at the limits of travel. The switch is activated when an object pushes against the switch arm, actuating the switch.

Los interruptores de límite varían en tamaño, la geometría de del **interruptor-brazo**, y en la cantidad de "fuerza" requerida para activar el interruptor. Mientras los interruptores de límite son más baratos, la actuación mecánica es por lo regular menos confiable que las alternativas sin contacto. Sin embargo, también son extremadamente versátiles, tanto que pueden ser **activadas** por cualquier objeto físico capaz de mover el brazo del interruptor.

See this *article* for writing the software for Limit Switches.

Interruptor de Proximidad Magnética



WCP Hall Effect Sensor

Los interruptores de proximidad magnética son activados cuando un imán se acerca a un cierto rango del sensor. En resultado, son interruptores de no contacto - no requieren contacto con el objeto detectado.

There are two major types of magnetic proximity switches - reed switches and hall-effect sensors. In a reed switch, the magnetic field causes a pair of flexible metal contacts (the «reeds») to touch each other, closing the circuit. A hall-effect sensor, on the other hand, detects the induced voltage transversely across a current-carrying conductor. Hall-effect sensors are generally the cheaper and more reliable of the two. Pictured above is the Hall effect sensor from West Coast Products.

Los interruptores de proximidad magnética podrían ser unipolar,» «bipolar,» or «omnipolar.» Un interruptor unipolar se activa y desactiva dependiendo de la presencia del polo magnético otorgado (ya sea norte o sur, dependiendo del interruptor). Un interruptor bipolar se activa desde la proximidad de un solo polo y se desactiva con la proximidad del polo opuesto. Un interruptor omnipolar se activará con la presencia de cualquier polo y se desactiva cuando la presencia del imán desaparece.

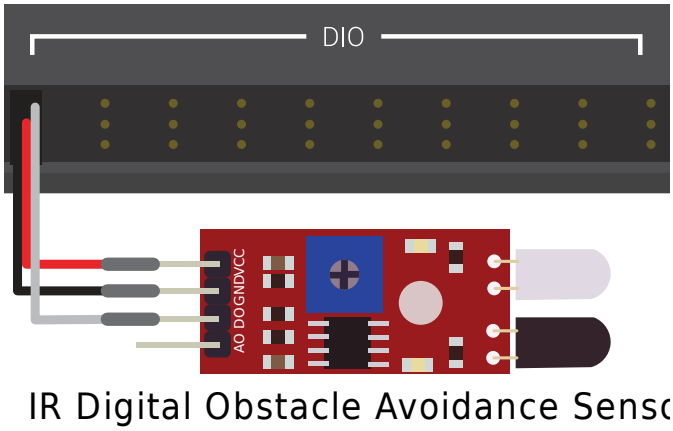
Mientras los interruptores de proximidad magnética son más confiables que sus contra partes mecánicas, requieren al usuario para montar un imán en el objeto para ser detectado - así son principalmente usados para detectar la ubicación de un mecanismo.

Interruptores de Proximidad Inductiva

Los interruptores de proximidad inductiva son activados cuando un conductor de cualquier tipo es dentro de un cierto rango del sensor. Como los interruptores de proximidad magnética, son interruptores de no contacto.

Los interruptores de proximidad inductiva son usados por algunos de los mismos propósitos que los switches de proximidad magnética. Su naturaleza más general (activándose en la presencia de cualquier conductor, en vez que solo un imán) puede ser una ayuda o un obstáculo, dependiendo de la naturaleza de la aplicación.

Interruptores de Proximidad Fotoeléctrica

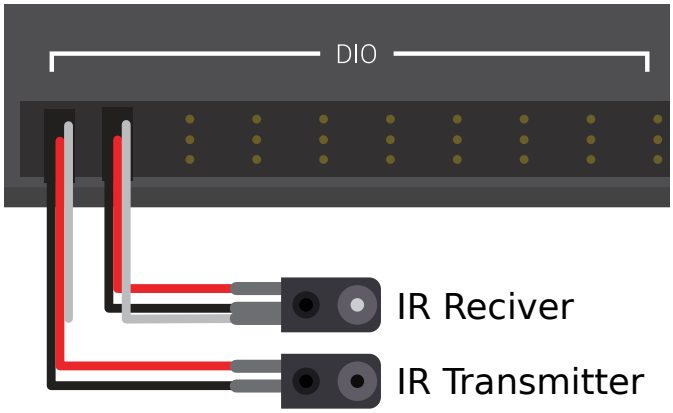


IR Digital Obstacle Avoidance Sensc

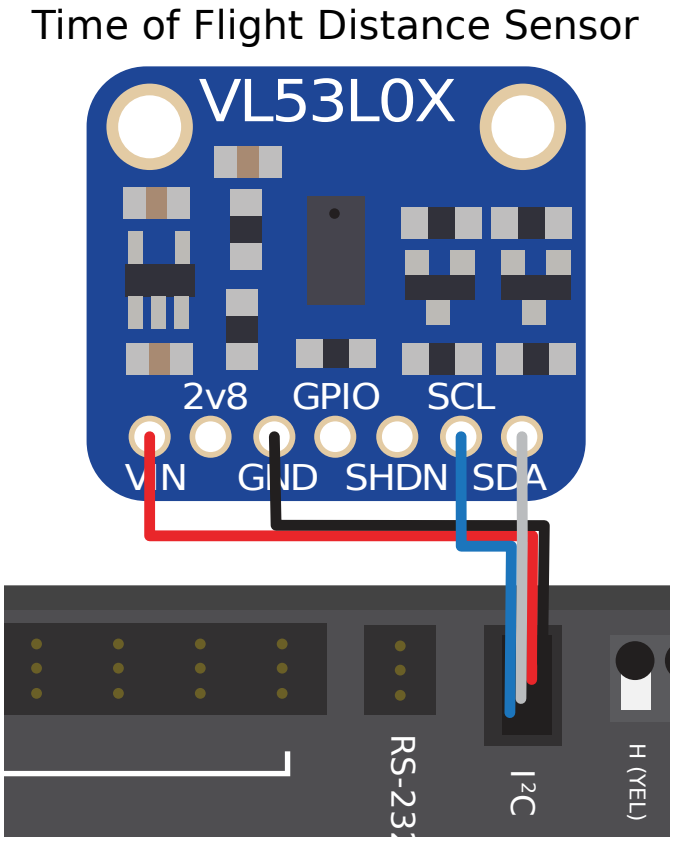
Photoelectric proximity switches are another type of no-contact proximity switch in widespread use in FRC. Photoelectric proximity switches contain a light source (usually an IR laser) and a photoelectric sensor that activates the switch when the detected light (which bounces off of the sensor target) exceeds a given threshold. One such sensor is the IR Obstacle Avoidance Module pictured above.

Since photoelectric proximity switches rely on measuring the amount of reflected light, they are often inconsistent in their triggering range between different materials - accordingly, most photoelectric sensors have an adjustable activation point (typically controlled by turning a screw somewhere on the sensor body). On the other hand, photoelectric sensors are also extremely versatile, as they can detect a greater variety of objects than the other types of no-contact switches.

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.



Interruptores de Proximidad de Tiempo de Vuelo



Time-of-flight Proximity Switches are newer to the market and are not commonly found in FRC. They use a concentrated light source, such as a small laser, and measure the time between the emission of light and when the receiver detects it. Using the speed of light, it can produce a very accurate distance measurement for a very small target area. Range on this type of sensor can range greatly, between 30mm to around 1000mm for the VL53L0X sensor pictured above. There are also longer range versions available. More information about time of flight sensors can be found in this article and more about the circuitry can be found in this article.

38.6 Codificadores - Hardware

Nota: This section covers encoder hardware. For a software guide to encoders, see *Codificadores - Software*.

Encoders are by far the most common method for measuring rotational motion in FRC®, and for good reason - they are cheap, easy-to-use, and reliable. As they produce digital signals, they are less-prone to noise and interference than analog devices (such as *potentiometers*).

38.6.1 Tipos de codificadores

There are three main ways encoders connect physically that are typically used in FRC:

- *Codificadores de eje*
- *Codificadores en el eje*
- *Codificadores magnéticos*

Estos codificadores varían en cómo son montados en el mecanismo en cuestión. En adición a este tipo de codificadores, algunos mecanismos de FRC vienen con codificadores de cuadratura integrados en el diseño.

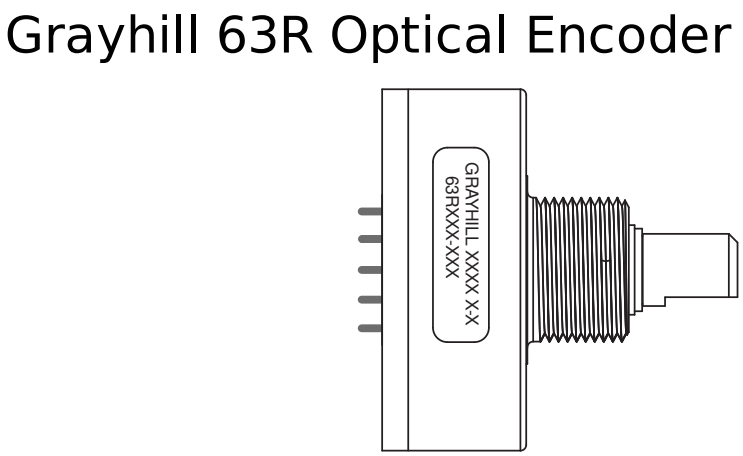
There are also three main ways the encoder data is communicated that are typically used in FRC:

- *Quadrature encoders*
- *Duty Cycle encoders*
- *Analog encoders*

Nota: Some encoders may support more then one communication method

---

Shafted Encoders



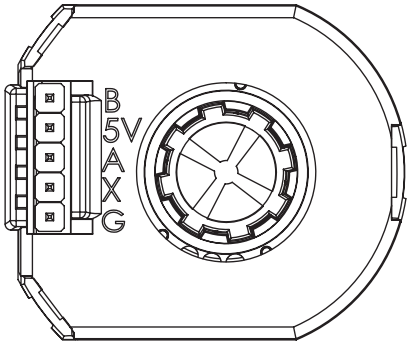
Shafted encoders have a sealed body with a shaft protruding out of it that must be coupled rotationally to a mechanism. This is often done with a helical beam coupling, or, more cheaply, with a length of flexible tubing (such as surgical tubing or pneumatic tubing), fastened with cable ties and/or adhesive at either end. Many commercial off-the-shelf FRC gearboxes have purpose-built mounting points for shafted encoders.

Examples of shafted encoders:

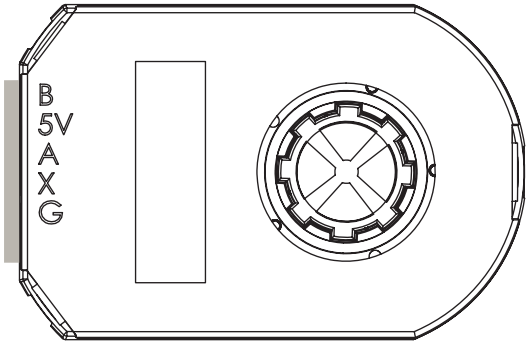
- Grayhill 63r
- US Digital MA3

On-shaft Encoders

AM10 Series Modular Incremental Encoder



AMT103



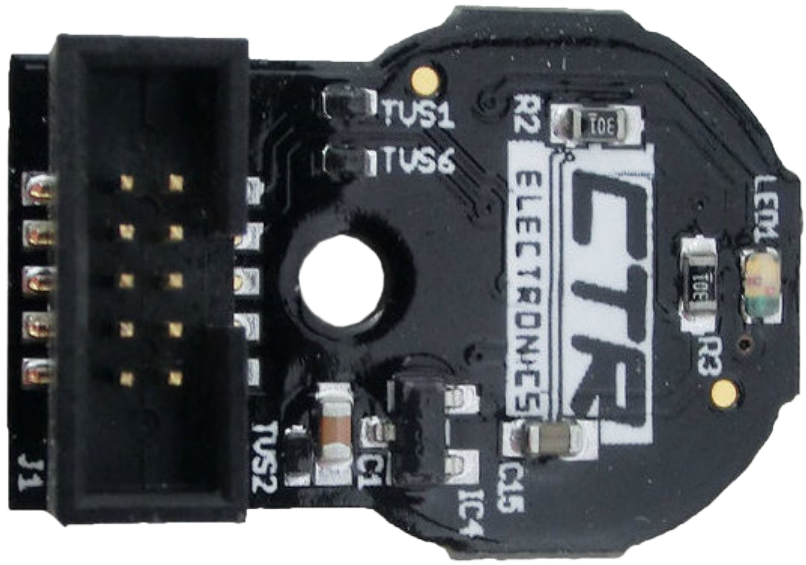
AMT102

On-shaft encoders couple to a shaft by fitting *around* it, forming a friction coupling between the shaft and a rotating hub inside the encoder.

Examples of On-shaft encoders:

- AMT103-V available through FIRST Choice
- CIMcoder
- REV Through Bore Encoder
- US Digital E4T

Magnetic Encoders



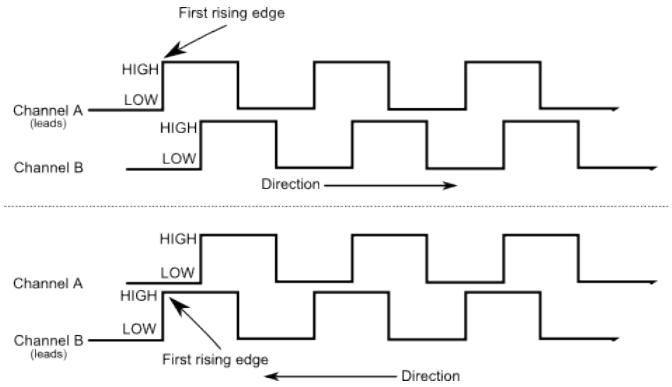
Magnetic encoders require no mechanical coupling to the shaft at all; rather, they track the orientation of a magnet fixed to the shaft. While the no-contact nature of magnetic encoders can be handy, they often require precise construction in order to ensure that the magnet is positioned correctly with respect to the encoder.

Examples of magnetic encoders:

- CTRE Mag Encoder
- Thrifty Absolute Magnetic Encoder
- Team 221 Lamprey2

Quadrature Encoders

El término cuadratura se refiere al método por el cual el movimiento es medido/codificado. Un codificador de cuadratura produce 2 olas cuadradas, pulsos de 90 grados fuera de fase entre ellos como se ve en la imagen debajo:



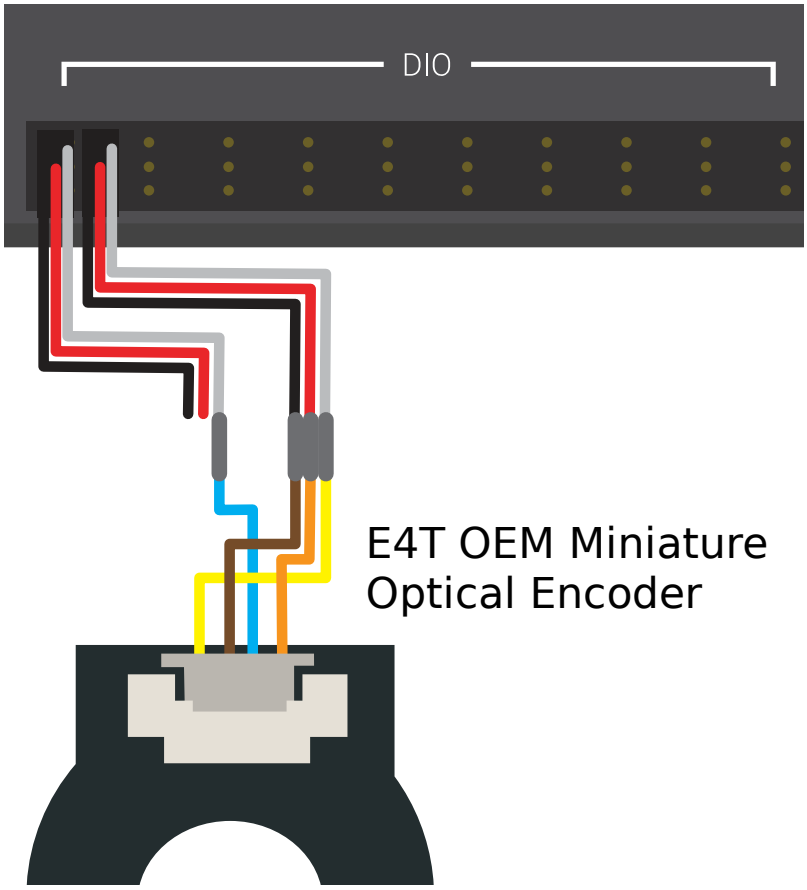
Así, a través de los 2 canales, son 4 bordes en total por periodo (por lo tanto “cuadrante”). El uso de 2 pulsos fuera de fase permite la dirección de movimiento para ser inequívocamente determinada por cuál pulso lidera a otro.

As each square wave pulse is a digital signal, quadrature encoders connect to the *digital input* ports on the roboRIO.

Examples of quadrature encoders:

- AMT103-V available through FIRST Choice
- CIMcoder
- CTRE Mag Encoder
- Grayhill 63r
- REV Through Bore Encoder
- US Digital E4T

Quadrature Encoder Wiring



Quadrature Encoders, such as the E4T OEM Miniature Optical Encoder, can be wired to two digital input ports as shown above.

Index

Some quadrature encoders have a third index pin which pulses when the encoder completes a revolution.

Quaderature Encoder Resolution

Advertencia: Los acrónimos “CPR” y “PPR” son usados por fuentes variadas para denotar ambos bordes por revolución y ciclos por revolución, entonces el acrónimo solo no es suficiente para decir cuál de los 2 se entiende cuando se da un valor. Si hay dudas, consulta el manual técnico del codificador específico.

Como codificadores de medida rotatoria con pulsos digitales, la exactitud de la medida es limitada por el número de pulsaciones por las cantidades dadas de los movimientos rotatorios. Esto es conocido como la “resolución” de los codificadores y es tradicionalmente medido en una de las 2 formas diferentes: bordes por revolución o ciclos por revolución.

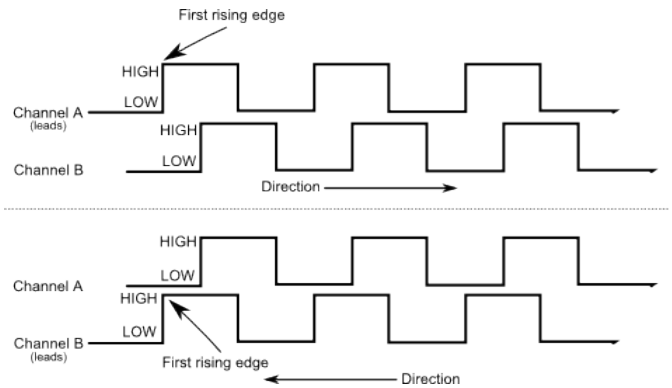
Los *bordes por revolución* se refieren al número total de transmisiones de alto a bajo o de bajo a alto a través de ambos canales por revolución del codificador de eje. Un periodo completo contiene *cuatro* bordes.

Ciclos por revolución se refiere al número total del periodo completo de ambos canales por revolución del codificador por eje. Un periodo completo es igual a un ciclo.

Así, una resolución expresada en bordes por revolución tiene un valor 4 veces mayor que la misma resolución expresada en ciclos por revolución.

En general, la resolución en su codificador en bordes por revolución debería ser un poco más fina que su más pequeño aceptable error en posicionamiento. Así, si usted quiere conocer el mecanismo más o menos un grado, debería tener un codificador con resolución algo más alta que 360 bordes por revolución.

Duty Cycle Encoders



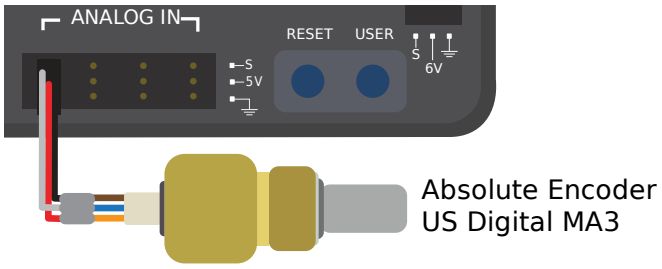
Duty cycle encoders connect to a single digital input on the roboRIO. They output a pulse where the length of a pulse is proportional to the absolute position of the encoder.

Examples of duty cycle encoders:

- AndyMark Mag Encoder

- CTRE Mag Encoder
- REV Through Bore Encoder
- Team 221 Lamprey2
- US Digital MA3

Analog Encoders



Analog encoders connect to a analog input on the roboRIO. They output a voltage proportional to the absolute position of the encoder.

Examples of analog encoders:

- Team 221 Lamprey2
- Thrifty Absolute Magnetic Encoder
- US Digital MA3

38.7 Giroscopio - Hardware

Nota: This section covers gyro hardware. For a software guide to gyros, see *Giroscopios - Software*.

Los giroscopios (o «gyros») son aparatos que miden la velocidad de rotación. Estos son particularmente útiles para estabilizar la conducción del robot, o para medir el camino o la inclinación mediante la integración (suma) de las medidas para obtener la medida del desplazamiento angular total.

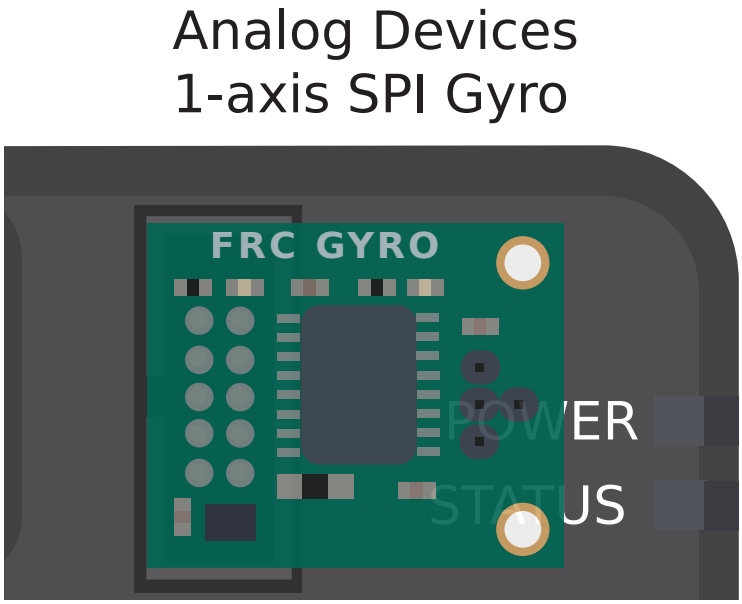
Varios dispositivos populares de FRC® conocidos como *IMUs* (Unidades de Medición de Inercia) combinan giroscopios de 3 ejes, acelerómetros y otros sensores de posición en un solo dispositivo. Algunos ejemplos son:

- Dispositivos Análogos ADIS16448 and ADIS 16470 IMUs
- CTRE Pigeon IMU
- Kauai Labs NavX

38.7.1 Tipos de Giroscopios

Existen dos tipos de giroscopios comúnmente usados en FRC; giroscopios de eje simple, giroscopios de 3 ejes y los IMUs, los cuales incluyen también giroscopios de 3 ejes.

Giroscopios de Eje Simple



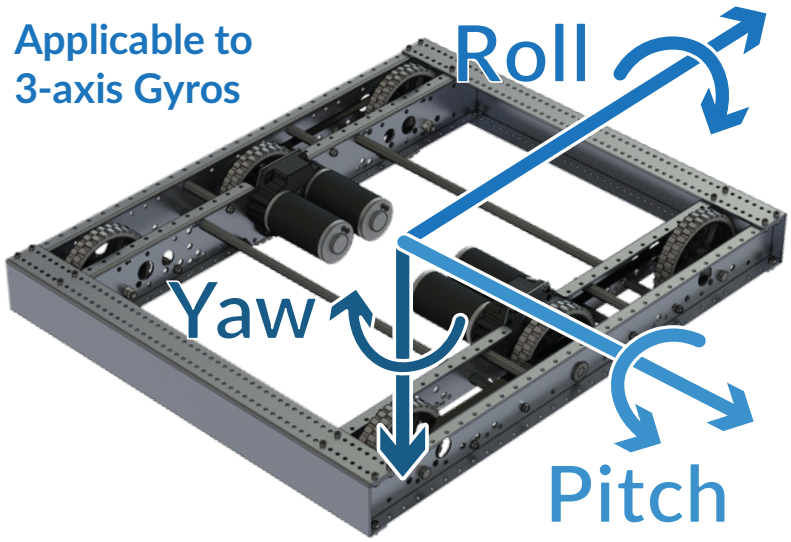
As per their name, single-axis gyros measure rotation rate around a single axis. This axis is generally specified on the physical device, and mounting the device in the proper orientation so that the desired axis is measured is highly important. Some single-axis gyros can output an analog voltage corresponding to the measured rate of rotation, and so connect to the roboRIO's *analog input* ports. Other single-axis gyros, such as the ADXRS450 pictured above, use the *SPI port* on the roboRIO instead.

El Dispositivo Análogo ADXRS450 FRC Gyro Board el cual ha estado en FIRST Choice en los años recientes es un giroscopio de eje simple usado comunmente.

Giroscopio de 3 Ejes

El giroscopio de 3 ejes mide la tasa de rotación alrededor de los 3 ejes espaciales (normalmente etiquetado como x, y, z). El movimiento alrededor de estos ejes se llama cabeceo, guiñada y alabeo.

The Analog Devices ADIS16470 IMU Board for FIRST Robotics that has been in FIRST Choice in recent years is a commonly used three-axis gyro.



Nota: The coordinate system shown above is often used for three axis gyros, as it is a convention in avionics. Note that other coordinate systems are used in mathematics and referenced throughout WPILib. Please refer to the *Drive class axis diagram* for axis referenced in software.

El giroscopio periférico de 3 ejes simplemente puede emitir tres voltajes analógicos (y así conectar al *analog input ports*, o (mas comúnmente) se pueden comunicar con uno de los *buses seriales* del roboRIO.

38.8 Ultrasónicos - Hardware

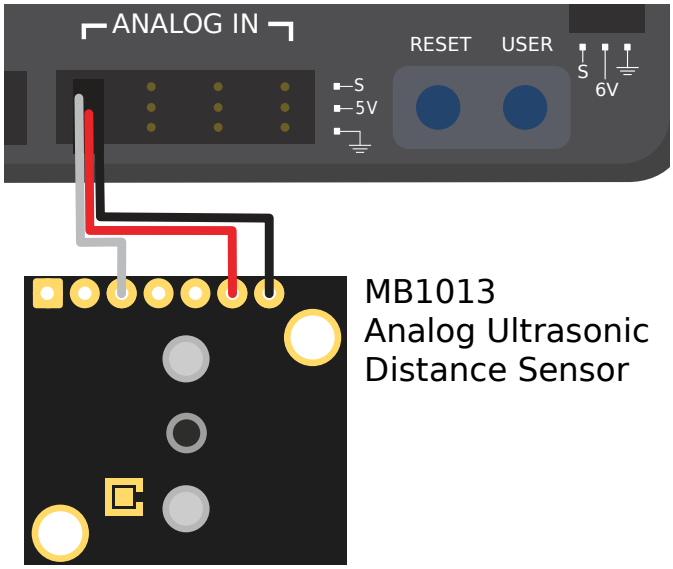
Nota: This section covers ultrasonic sensor hardware. For a software guide to ultrasonics, see *Ultrasónicos - Software*.

Los telémetros ultrasónicos son algunos de los telémetros más comunes utilizados en FRC®. Son baratos, fáciles de usar y bastante seguros. Los telémetros ultrasónicos funcionan emitiendo un pulso de sonido de alta frecuencia, y después midiendo cuánto tiempo tarda el eco en llegar al sensor después de rebotar en el objetivo. A partir del tiempo medido y la velocidad del sonido en el aire, es posible calcular la distancia al objetivo.

38.8.1 Tipos de ultrasónicos.

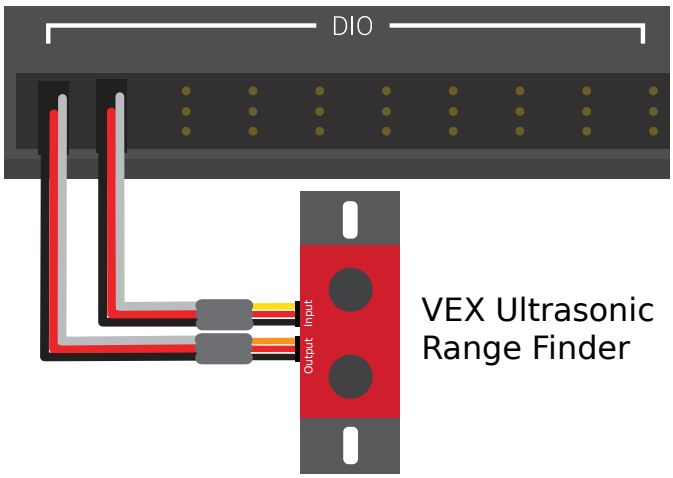
Mientras todos los telemetros ultrasónicos operan en “respuesta ping”, principio utilizado anteriormente, puede variar la forma en que se comunica con la roboRIO.

Ultrasónicos análogos



Analog ultrasonics output a simple analog voltage corresponding to the distance to the target, and thus connect to an *analog input* port. The user will need to calibrate the voltage-to-distance conversion in *software*.

Ultrasónicos de respuesta ping



Mientras todos los ultrasónicos son dispositivos de respuesta ping funcionales, un ultrasónico de “respuesta ping“ está configurado para ser conectado en a una entrada y salida digital. Ver en <docs/hardware/sensors/digital-inputs-hardware:Connecting a sensor that uses multiple DIO ports>`. La salida digital es usada para mandar el ping mientras que la entrada para mandar la respuesta.

Ultrasónicos de serie

Algunos sensores ultrasónicos más complicados pueden comunicarse con la RÍO sobre unos de los *serial buses* , tales como RS-232.

38.8.2 Advertencias

Los sensores ultrasónicos son generalmente bastante fáciles de usar, sin embargo, hay algunas advertencias. Como los ultrasonidos funcionan midiendo el tiempo entre el pulso y su eco, generalmente miden la distancia solo al objetivo *más cercano* en su rango. Por lo tanto, es extremadamente importante elegir el sensor adecuado para el trabajo. La documentación de los sensores ultrasónicos generalmente incluirá una imagen del «patrón de haz» que muestra la forma de la «ventana» en la que el ultrasonido detectará un objetivo; preste mucha atención a esto cuando seleccione su sensor.

Igualmente estos sensores son susceptibles a tener interferencia con otros sensores iguales. Para disminuir esto, la roboRIO puede correr un ultrasonido de respuesta ping en un «round-robin» - sin embargo en las competencias no existe un modo seguro de anular esta interferencia en los sensores montados en otros robots.

Finalmente, estos sensores probablemente no lleguen a detectar objetos que absorben las ondas de sonido o que las lleguen a redireccionar a extrañas vías. Así los sensores trabajan mejor detectando arduamente objetos planos.

38.9 Acelerómetros - Hardware

Estos sensores son comúnmente usados para medir la aceleración.

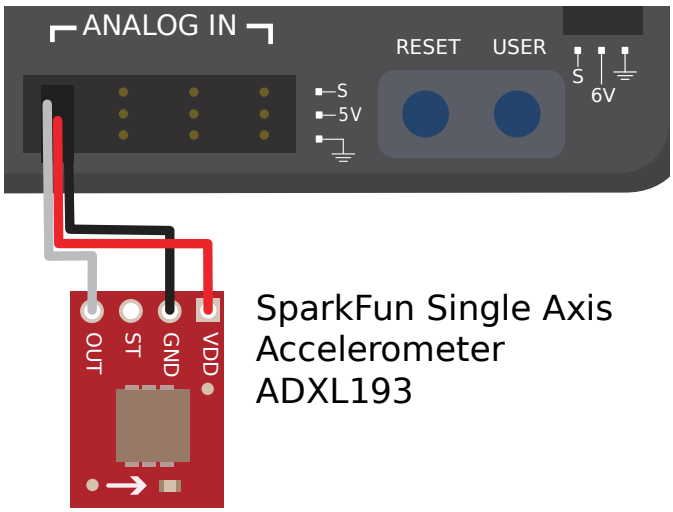
En principio, las medidas precisas de aceleración pueden ser doblemente integradas y usadas para rastrear posiciones (similar a como las medidas de velocidad del giroscopio pueden ser integradas para determinar el rumbo), sin embargo, en la práctica, los acelerómetros que están aceptados dentro del rango de precios legal de FRC® no son tan precisos para este uso. Sin embargo, estos sensores son usados en diferentes tareas de FRC.

The roboRIO comes with a *built-in three-axis accelerometer* that all teams can use, however teams seeking more-precise measurements may purchase and use a peripheral accelerometer, as well.

38.9.1 Tipos de acelerómetros

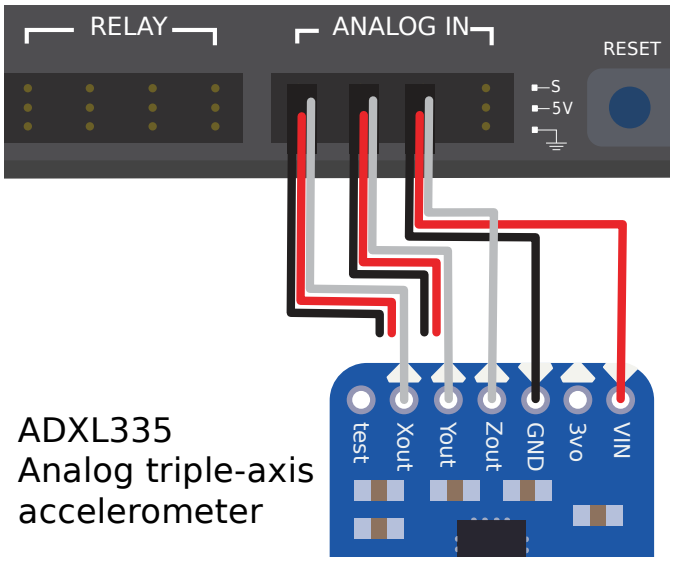
Existen 3 tipos de acelerómetros comúnmente usados en FRC: acelerómetros de eje simple, acelerómetros de eje múltiple e IMUs.

Acelerómetros de eje simple



Como lo dice su nombre estos acelerómetros pueden medir la aceleración a lo largo de ese simple eje. Estos ejes son generalmente especificados en los dispositivos físicos, y montar el dispositivo de manera apropiada para que el eje del dispositivo sea medido es muy importante. Estos acelerómetros generalmente sacan un voltaje análogo que corresponde a la medición de la aceleración, para que este sea conectado al puerto de entrada análoga en la roboRIO. Ver *analog input*.

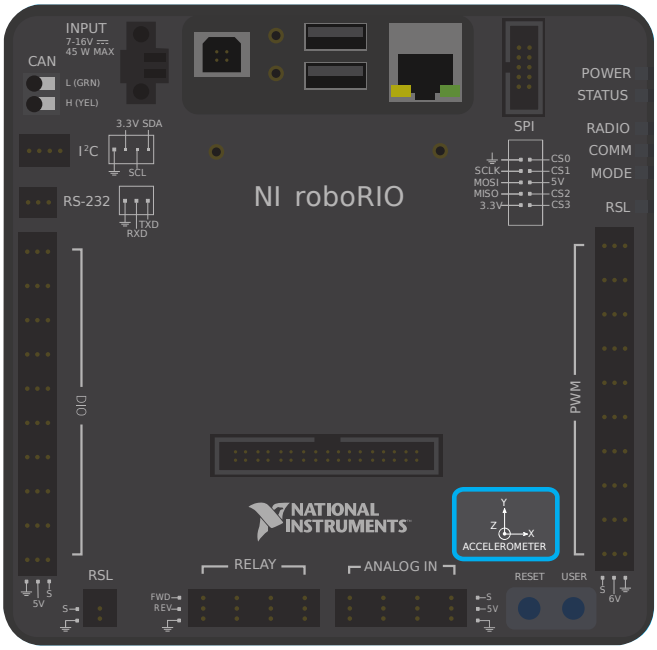
Aceletrómetros de eje múltiple



Estos acelerómetros miden la aceleración a lo largo de los múltiple ejes espaciales. El acelerómetro ya integrado en la roboRIO es unos de 3 ejes.

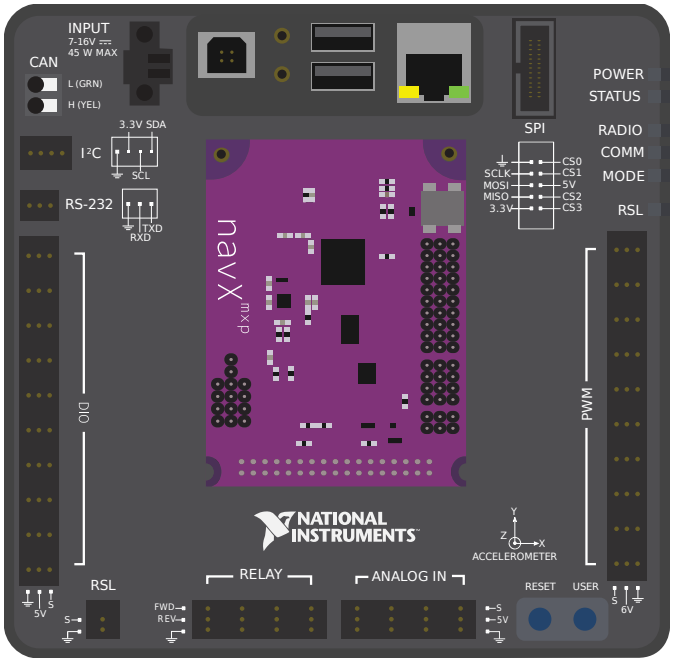
Los acelerómetros periféricos de eje múltiple pueden sacar múltiple voltajes análogos (y así ser conectados al puerto de entrada análoga, ver *analog input ports*, o (comúnmente) pueden comunicarse con uno de los serial buses de la roboRIO *serial buses*.

Acelerómetro integrado en la roboRIO.



The roboRIO has a built-in accelerometer, which does not need any external connections. You can find more details about how to use it in the *Built-in Accelerometer section* of the software documentation.

IMUs (Unidades de medida inercial, por sus siglas en inglés)



Varios de los más populares dispositivos de FRC (conocidos como IMUs) que combinan acelerómetros y giroscopios son:

- Dispositivos Análogos IMUs ADIS 16448 y ADIS 16470 <https://www.analog.com/en/landing-pages/001/first.html>`__
- CTRE Pigeon IMU
- Kauai Labs NavX. <https://pdocs.kauailabs.com/navx-mxp/>`__

38.10 LIDAR - Hardware

Los sensores LIDAR (detección y alcance de luz) son una variedad de telémetros que se usan cada vez más en FRC®.

Trabajan casi igual que los ultrasónicos *ultrasonics*, pero usan luz en vez de sonido. Al láser ser pulsado, el sensor mide el tiempo en que este rebota de regreso.

38.10.1 Tipos de LIDAR

Existen 2 tipos muy comúnmente usados en FRC: 1-LIDAR dimensional y 2-LIDAR dimensional.

LIDAR 1-Dimensión

Estos sensores (1D) trabajan como un sensor ultrasónico - mide la distancia del objeto mas cercano mas o menos a lo largo de una línea en frente de este. Estos sensores llegan a ser más confiables que los ultrasónicos ya que son menos propensos a interferencia y tienen unos

perfiles de viga más estrechos. La fotografía mostrada arriba es el Sensor de Distancia Óptica Garmin LIDAR-Lite <https://www.andymark.com/products/lidar-lite-3>`__.

También, estos sensores 1D generalmente sacan un voltaje análogo proporcional a la distancia de la medida, para así ser conectado al puerto de entrada análoga de la roboRIO *analog input* o a unos de los buses seriales de la roboRIO *roboRIO's serial buses*.

LIDAR 2-Dimensiones

Estos sensores (2D) miden distancias a todas direcciones sobre un plano. Generalmente esto es logrado (más o menos) simplemente colocando un 1D LIDAR en una placa giratoria que gira a una cierta velocidad.

Desde que los sensores de la 2D LIDAR naturalmente necesitan mandar una gran cantidad de información a la roboRIO, este se conecta al menos a uno de los buses seriales de la roboRIO *serial buses*.

38.10.2 Advertencias

Los sensores LIDAR sufren de algunos inconvenientes comunes:

Igual que los ultrasónicos, estos se basan en la reflexión emitida con el pulso que regresa al sensor. Así que los LIDAR dependen fuertemente en la reflexión del material en la longitud de onda del láser. La barrera de la cancha de FRC es hecha de policarbonato, la cual tiende a ser transparente en la longitud de onda infrarroja (la cual es generalmente legal para el uso en FRC). Así, las LIDAR tienden a batallar para detectar la barrera de la cancha.

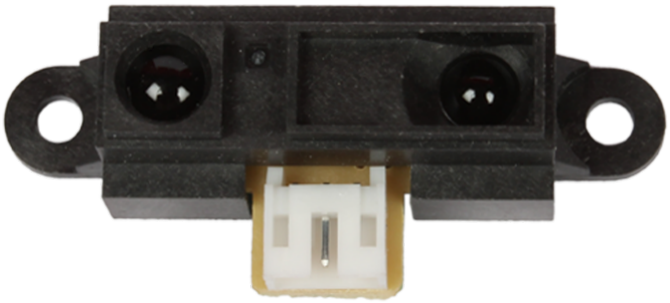
Los sensores 2D LIDAR (en el rango de precio legal para el uso en FRC) tienden a ser un poco ruidosos y el proceso de medición de la información (conocido como "punto de nube") puede estar envuelto en un software muy complejo. Adicionalmente, existen muy pocos sensores 2D LIDAR echos especialmente para FRC, así que el soporte del software tiende a ser escaso.

Como los sensores 2D LIDAR dependen de un plato giratorio para trabajar, las velocidades son limitadas a la velocidad por la que gira el plato giratorio. Por sensores que se encuentran

en el rango de precios legal en FRC, esto también significa que no se podrán actualizar los valores particularmente rápido, el cual puede ser una limitación cuando el robot (o el objetivo) está en movimiento.

Adicionalmente , como los sensores 2D LIDAR son limitados en su resolución angular, la resolución espacial del punto de nube es peor cuando el objetivo se encuentra lejos.

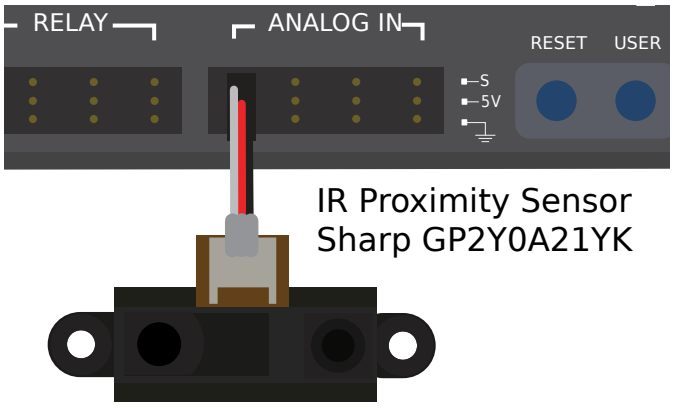
38.11 Telémetros triangulados



Los telémetros triangulados (con frecuencia llamados «telémetros de infrarrojos», ya que normalmente funcionan en la banda de longitud de onda infrarroja) son otro tipo común de telémetros que se utilizan en FRC®. El sensor que se muestra arriba es un sensor Sharp-brand común

A diferencia de las LIDAR *LIDAR*, estos telémetros no miden el tiempo entre la emisión del pulso y la reflexión recibida. Más bien, la mayoría de telémetros IR trabajan emitiendo un haz constante en un ligero ángulo y miden la posición del haz reflejado. Cuanto más cerca el punto de contacto del haz reflejado está del emisor, más cerca estará el sensor del objetivo.

38.11.1 Uso de los telémetros IR



Los telémetro IR por lo regular sacan un voltaje análogo proporcional con la distancia del objetivo para así ser conectado al puerto de entrada análoga de la roboRIO *analog input*.

38.11.2 Advertencias

Estos telemetros sufren inconvenientes similares a los sensores 1D LIDAR - son muy sensibles al reflejo del objetivo en la longitud de onda emitida por el láser.

Adicionalmente, mientras el telémetro IR tiende a ofrecer mejor resolución que los sensores LIDAR cuando se miden distancias cortas, son usualmente más sensibles para diferenciar la orientación del objetivo, especialmente si el objetivo es muy reflectivo (tal como un espejo).

38.12 Buses seriales

En adición a las entradas digitales *digital* y análogas *analog*, la roboRIO también ofrece varios métodos de comunicación en serie con dispositivos periféricos.

Both the digital and analog inputs are highly limited in the amount of data that can be sent over them. Serial buses allow users to make use of far more-robust and higher-bandwidth communications protocols with sensors that collect large amounts of data, such as inertial measurement units (IMUs) or 2D LIDAR sensors.

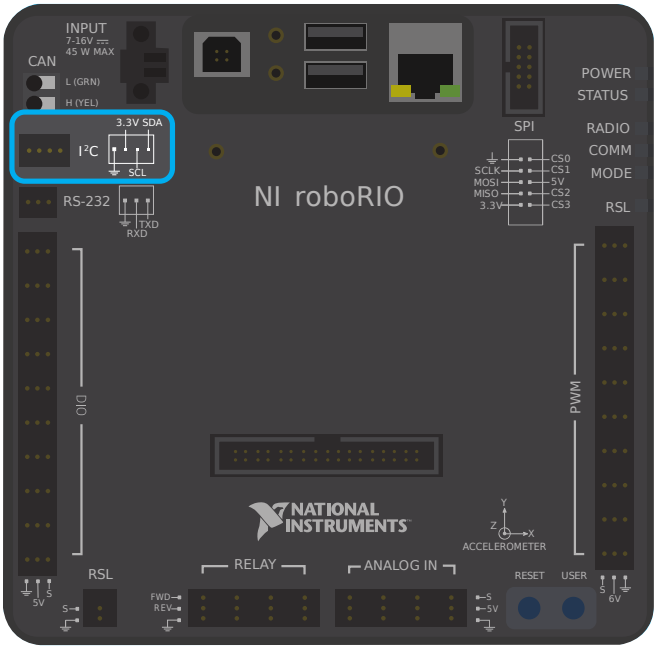
38.12.1 Tipos de buses seriales soportados

La roboRio soporta varios tipos de comunicaciones en serie:

- *I2C*
- *SPI*
- *RS-232*
- *USB Host*
- *CAN Bus*

Additionally, the roboRIO supports communications with peripheral devices over the *CAN* bus. However, as the FRC® CAN protocol is quite idiosyncratic, relatively few peripheral sensors support it (though it is heavily used for motor controllers).

38.12.2 I2C



I²C Port

Figure 6 and Table 5 describe the I²C port pins and signals.

Figure 6. I²C Port Pinout

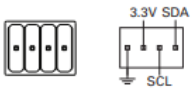


Table 5. I²C Port Signal Descriptions

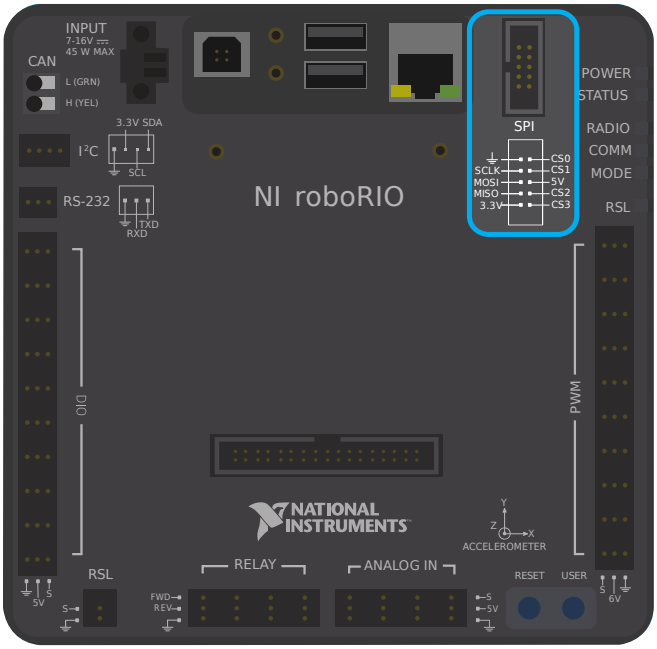
Signal Name	Direction	Description
GND	—	Reference for digital lines and +3.3 V power output.
3.3V	Output	+3.3 V power output.
SCL	Input or Output	I²C lines with 3.3 V output, 3.3 V/ 5 V-compatible input. Refer to the *I²C Lines* section for more information.
SDA	Input or Output	

To communicate to peripheral devices over *I2C*, each pin should be wired to its corresponding pin on the device. I2C allows users to wire a «chain» of slave devices to a single port, so long as those devices have separate IDs set.

The I2C bus can also be used through the *MXP expansion port*. The I2C bus on the *MXP* is independent. For example, a device on the main bus can have the same ID as a device on the MXP bus.

Advertencia: Be sure to familiarize yourself on the following known issue before using the onboard I2C port: *Onboard I2C Causing System Lockups*

38.12.3 SPI



SPI Port

Figure 13 and Table 12 describe the SPI port pins and signals.

Figure 13. SPI Port Pinout

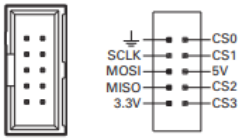


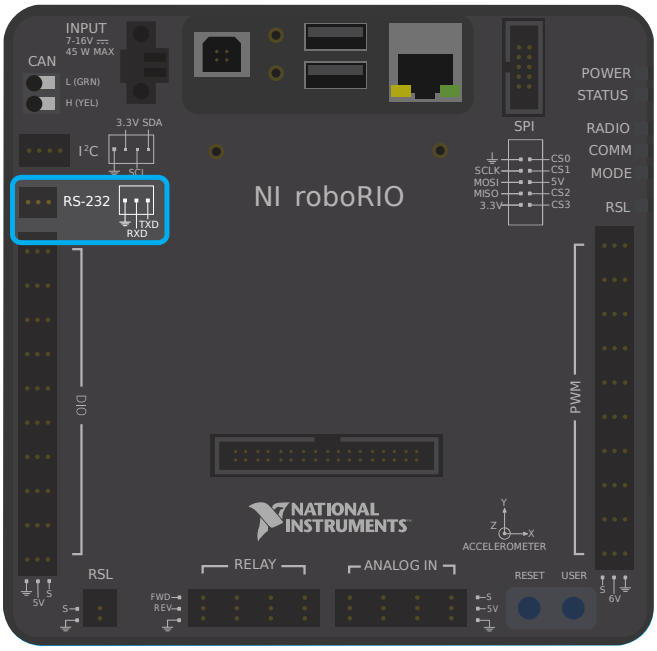
Table 12. SPI Port Signal Descriptions

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.

To communicate to peripheral devices over *SPI*, each pin should be wired to its corresponding pin on the device. The SPI port supports communications to up to four devices (corresponding to the Chip Select (CS) 0-3 pins on the diagram above).

El bus SPI también puede ser usado mediante *MXP expansion port*. El puerto MXP proporciona un reloj independiente y líneas de entrada/salida y un CS adicional.

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

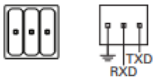


Table 6. RS-232 Serial Port Signal Descriptions

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.

Para comunicar con los dispositivos periféricos sobre el RS-232, cada pin debe ser cableado con su correspondiente pin en el dispositivo.

El bus RS-232 también puede ser usado mediante el puerto de expansión MXP *MXP expansion port*.

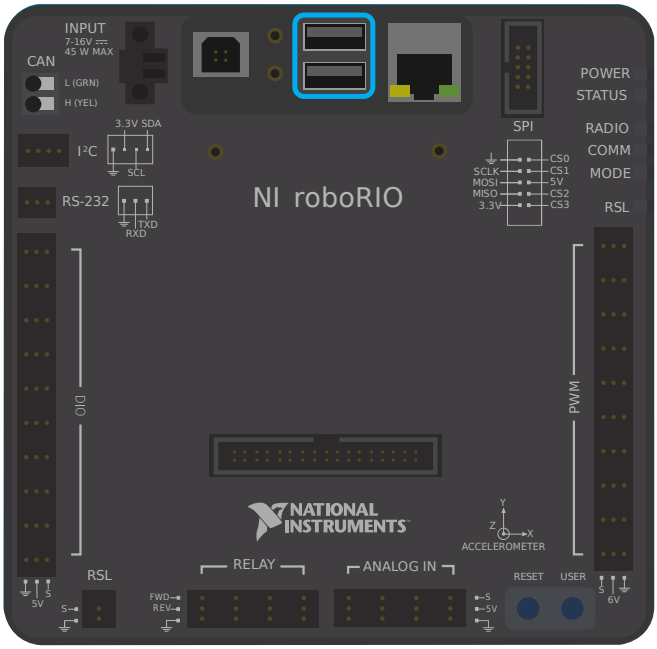
The roboRIO RS-232 serial port uses RS-232 signaling levels (+/- 15v). The MXP serial port uses CMOS signaling levels (+/- 3.3v).

Nota: By default, the onboard RS-232 port is utilized by the roboRIO's serial console. In order to use it for an external device, the serial console must be disabled using the *Imaging Tool* or *RoboRIO Web DashBoard*.

38.12.5 USB Client

Uno de los puertos USB de la roboRIO es un USB-B o un puerto de cliente USB. Este puede ser conectado a dispositivos, tales como la Driver Station con un cable USB estándar.

38.12.6 USB Host



Dos de los puertos USB de la roboRIO es un USB-A o un puerto USB Host. Este puede ser conectado a diferentes dispositivos, tales como cámaras o sensores usando un cable USB estándar.

38.12.7 Puerto de expansión MXP

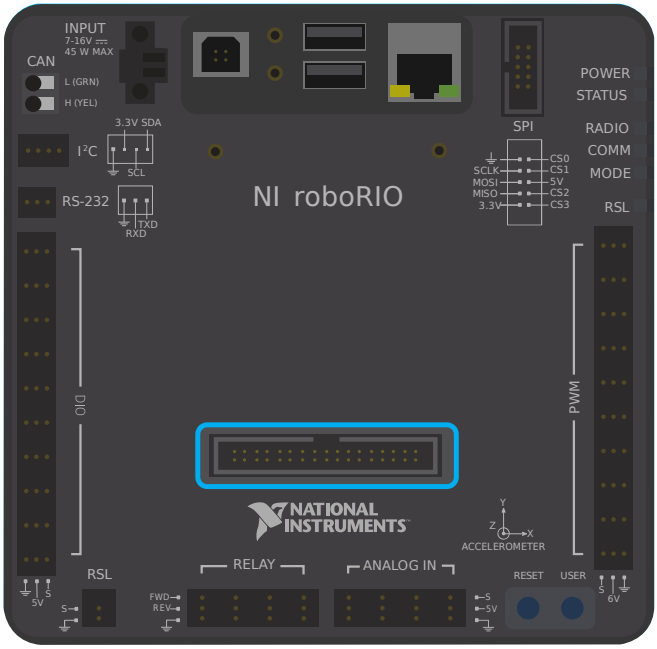
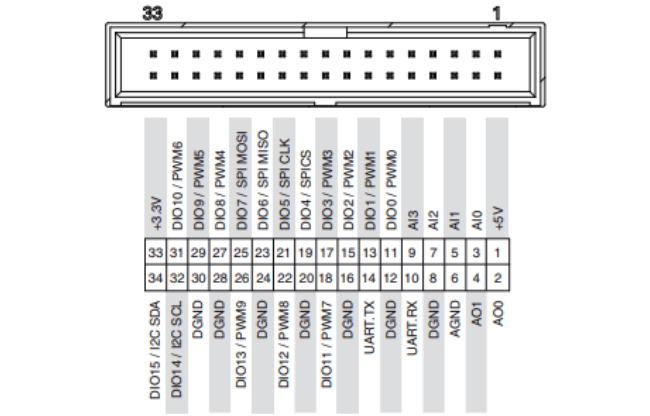


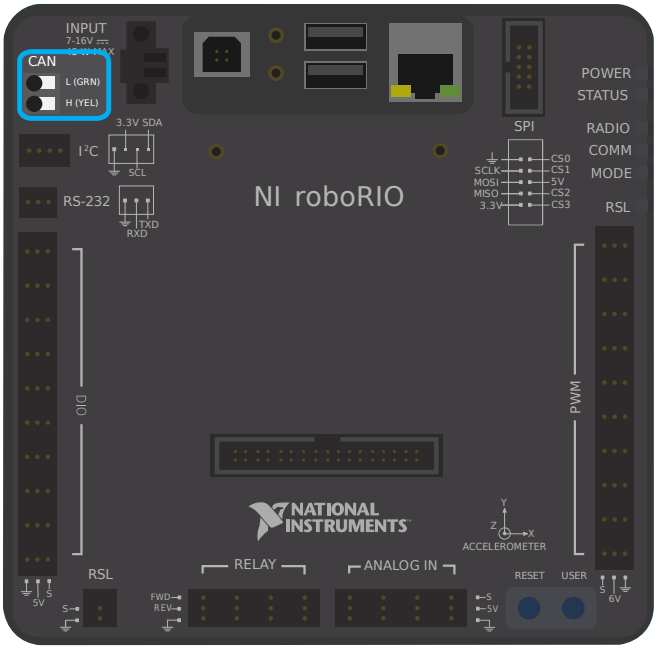
Figure 4. MXP Pinout



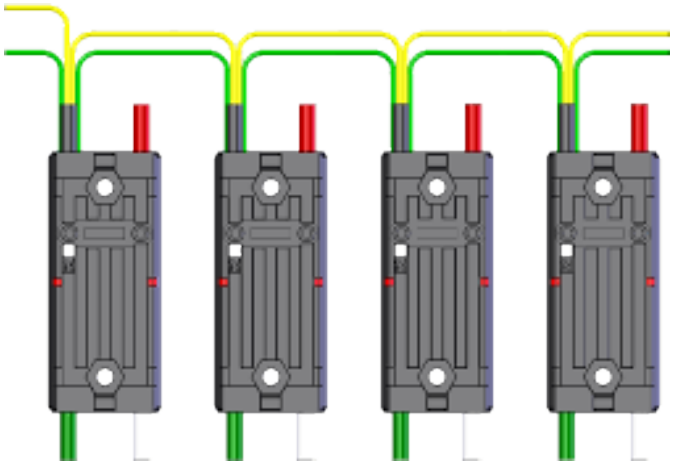
Several of the serial buses are also available for use through the roboRIO's *MXP* Expansion Port. This port allows users to make use of many additional *digital* and *analog* inputs, as well as the various serial buses.

Algunos dispositivos periféricos se adjuntan directamente al puerto MXP por conveniencia, no requieren cableado por parte del usuario.

38.12.8 Bus CAN



Adicionalmente la roboRIO soporta comunicaciones con dispositivos periféricos sobre el Bus CAN. Sin embargo como el protocolo CAN de FRC es bastante peculiar, relativamente algunos sensores periféricos lo soportan (aunque es más pesado usarlo para controles de motor). Una de las ventajas de usar el protocolo Bus CAN es que los dispositivos pueden ser encadenados, como se muestra arriba. Si la energía es removida de algún dispositivo en la cadena, las señales de información seguirán disponibles seguirán disponibles para llegar a todos los dispositivos de la cadena.



Algunos sensores son usados primordialmente por el bus CAN. Algunos ejemplos son:

- CAN basado en el rango de tiempo de vuelo/ sensor de distancia de playingwithfusion.com <https://www.playingwithfusion.com/productview.php?pdid=96&catid=1009>`__
- TalonSRX-based sensors, such as the Gadgeteer Pigeon IMU and the SRX MAG Encoder
- CANifier

- Power monitoring sensors built into the *CTRE Power Distribution Panel (PDP)* and the *REV Power Distribution Hub (PDH)*

Más información sobre cómo usar dispositivos conectados al Bus CAN, Ver *using can devices*.

39

Empezar a trabajar con 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.

Truco: A course that teaches programming using the Romi Robot is available via Thinkscape. Information on this course is available here

---

39.1 Romi Hardware y montaje

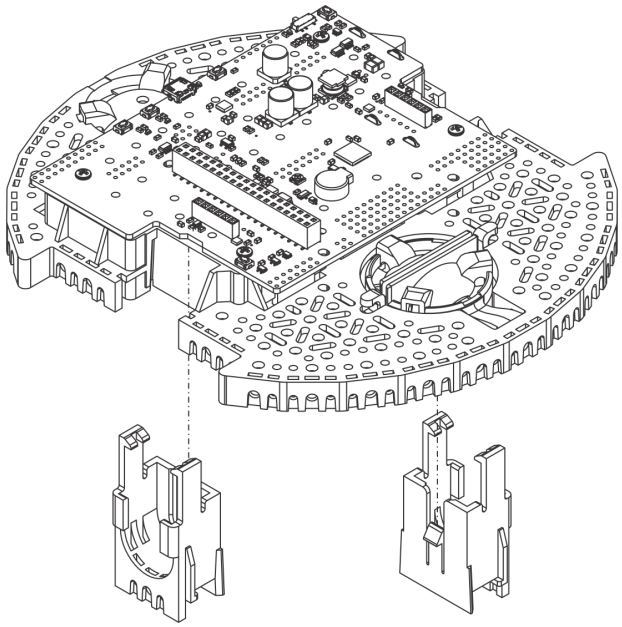
Para comenzar con Romi, deberá tener el hardware necesario.

1. Kit Romi de Pololu – El pedido tiene derecho a un envío gratis
2. *Raspberry Pi <https://www.amazon.com/gp/product/B07BFH96M3/>* __ - 3B + o 4
3. 8GB (o más grande) tarjeta Micro SD
4. Lector de tarjeta Micro SD - si no tiene uno ya
5. 6 pilas AA – Mejor si son recargables (no olvides el cargador)

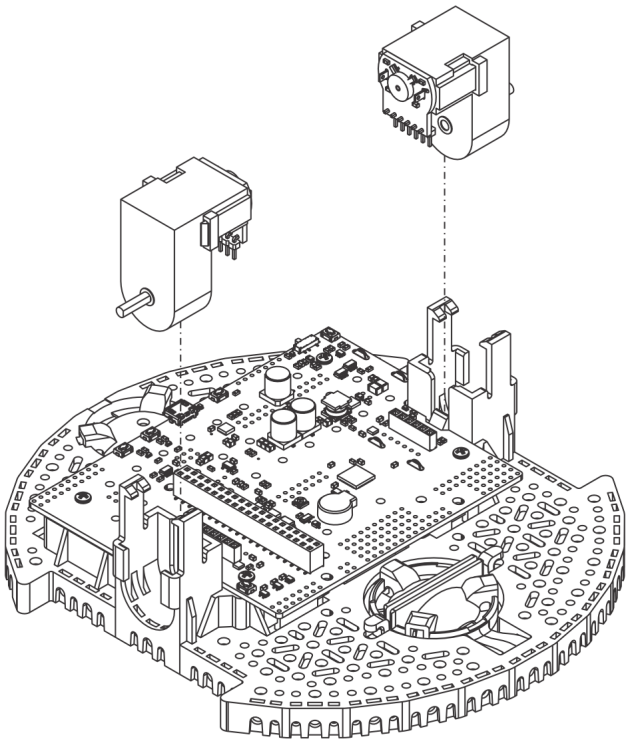
39.1.1 Ensamblaje

El kit de robot Romi para FIRST viene pre-soldado y solo tiene que ensamblarse antes de poder usarse. Una vez que haya reunido todos los materiales, puede comenzar a ensamblar:

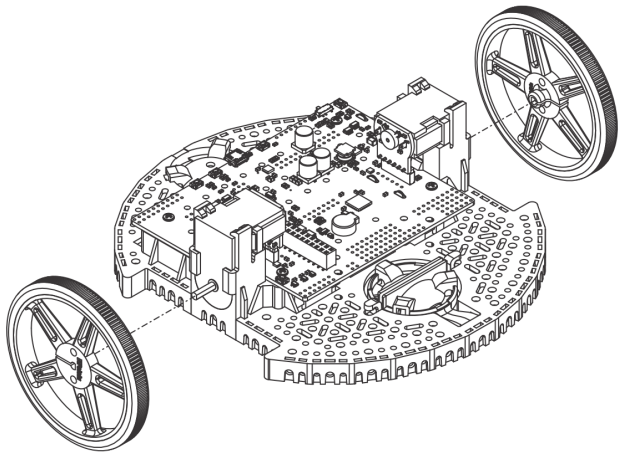
1. Alinee los clips del motor con el chasis como se indica y presiónelos firmemente en el chasis hasta que la parte inferior de los clips esté nivelada con la parte inferior del chasis (puede que escuche varios clics).



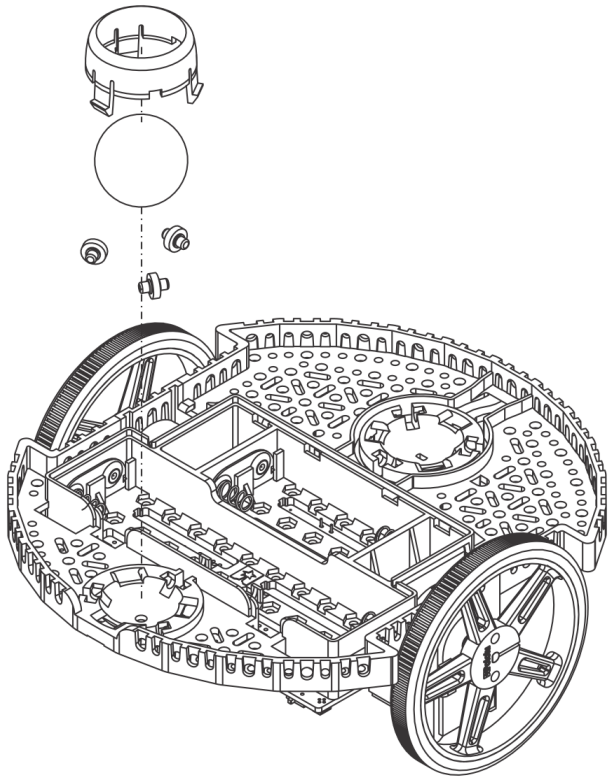
2. Empuje los mini motorreductores de plástico en las abrazaderas del motor hasta que encajen en su lugar. Tenga en cuenta que el motor bloquea la liberación de la abrazadera, por lo que si necesita quitar un soporte de motor más tarde, primero tendrá que quitar el motor. Los mini motorreductores de plástico que vienen con el kit tienen ejes de motor extendidos para permitir que los codificadores de cuadratura puedan retroalimentar la posición.



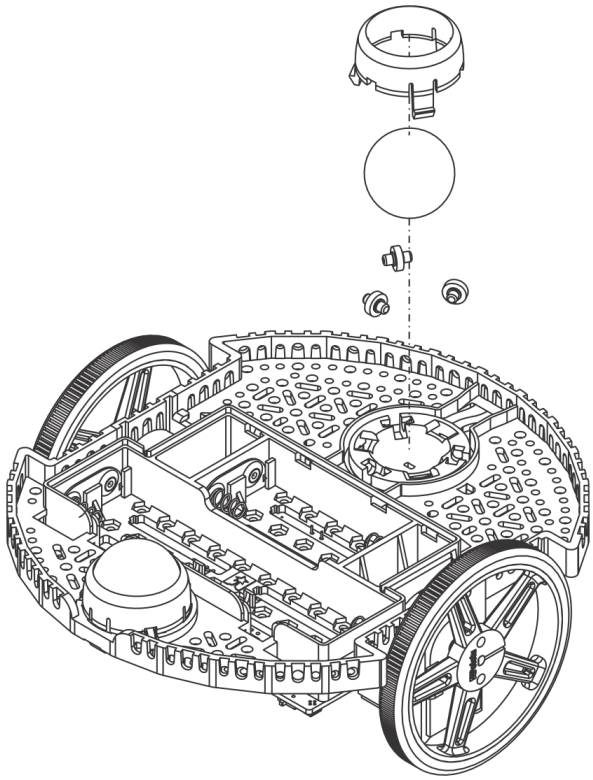
3. Presione las ruedas sobre los ejes de salida de los motores hasta que el eje del motor esté al ras de la cara exterior de la rueda. Una forma de hacerlo es colocar la rueda en una superficie plana y alinear el chasis con ella para que la parte plana del eje D del motor se alinee correctamente con la rueda. Luego, baje el chasis, presionando el eje del motor en la rueda hasta que entre en contacto con la superficie.



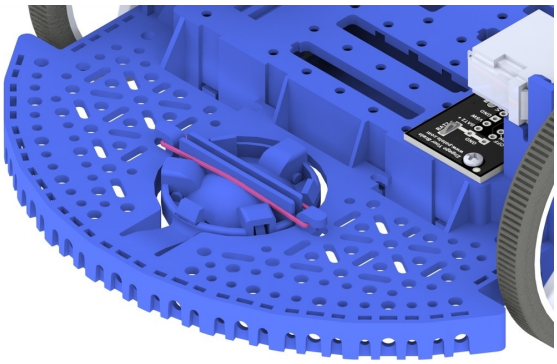
4. Ponga el chasis al revés y coloque los tres rodillos para el lanzador de bolas trasero en los recortes del chasis. Coloque la bola de plástico 1″ encima de los tres rodillos. Luego empuje el clip de retención del lanzador de pelotas sobre la pelota y dentro del chasis para que las tres patas encajen en sus respectivos agujeros.



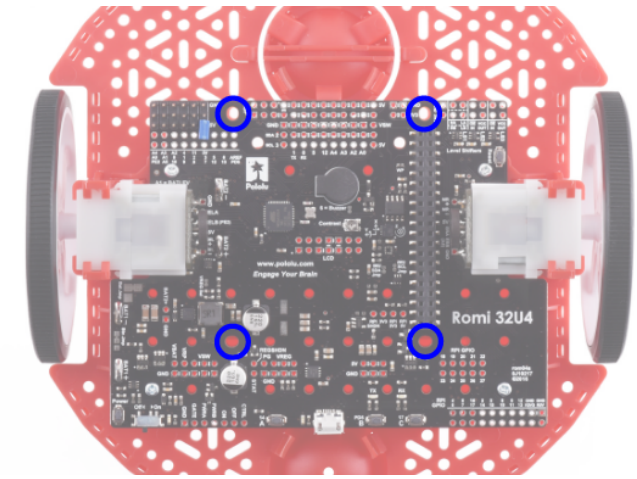
5. Repita para el lanzador de bolas frontal para que haya un lanzador en la parte delantera y trasera del robot.



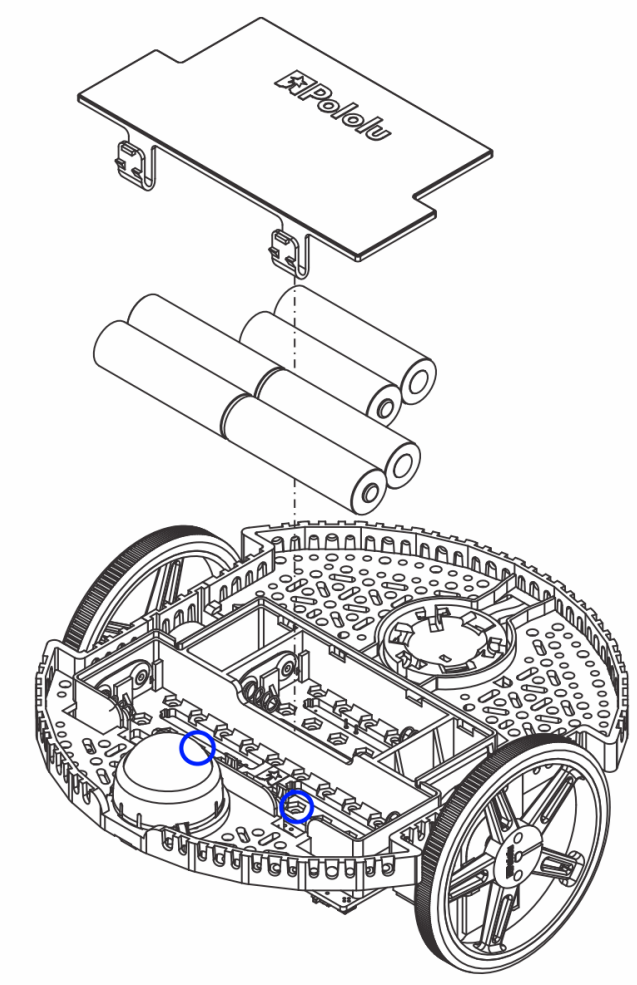
6. Opcional: El lanzador de bolas delantero está apoyado por un brazo flexible que actúa como sistema de suspensión. Si quiere hacerlo más rígido, puede envolver una banda de goma alrededor de los dos ganchos situados a cada lado del lanzador de pelotas en la parte superior del chasis.



7. Instale los separadores para apoyar la placa Raspberry Pi. Dos separadores (con la rosca hacia abajo) se montan en los agujeros del lado de la placa Romi más cercano a la etiqueta «Romi 32U4» como se muestra en la imagen. Las tuercas para estos separadores están dentro del compartimiento de la batería. Los otros dos separadores van en los agujeros del lado opuesto de la placa. Para colocarlos, necesitarás unos alicates de punta para sujetar la tuerca mientras atornillas los separadores. Los agujeros marcados con un círculo en la imagen de abajo muestran dónde deben ir los separadores.



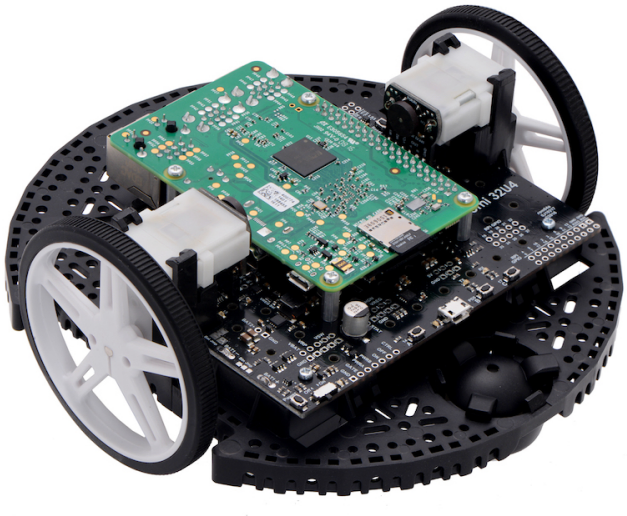
8. El chasis funciona con cuatro o seis pilas AA (recomendamos utilizar pilas AA NiMH recargables). La orientación correcta de las baterías se indica mediante los orificios en forma de batería en el chasis Romi, así como los indicadores + y - en el propio chasis.



9. Coloque la placa Raspberry Pi boca abajo, alineando con cuidado el conector de 2x20 pines en el Pi con el enchufe de 2x20 pines en el Romi. Empuje con presión uniforme teniendo cuidado de no doblar ninguno de los pasadores. Una vez insertado, use los torni-

llos suministrados para sujetar la placa Raspberry Pi a los separadores que se instalaron en un paso anterior.

Nota: Dos de los tornillos requerirán la colocación de una tuerca en un agujero hexagonal dentro del compartimento de la batería. Las ubicaciones se muestran con los círculos azules en la imagen de arriba.



¡El ensamblaje de su chasis Romi ahora está completo!

39.2 Configuración de su Romi

La Romi tiene 2 placas de microprocesador:

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.

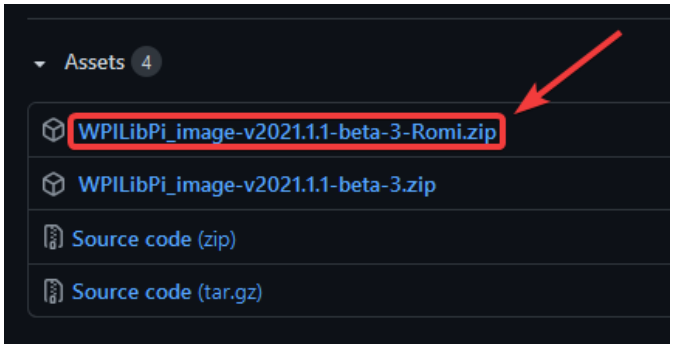
39.2.1 Raspberry Pi

Descarga

El firmware de Raspberry Pi está basado en WPILibPi (anteriormente FRCVision) y debe ser descargado y escrito en la tarjeta micro SD de Raspberry Pi. Haga clic en `Assets` en la parte inferior de la descripción para ver los archivos de imagen disponibles:

Romi WPILibPi <https://github.com/wpilibsuite/WPILibPi/releases> __

Asegúrese de descargar la versión Romi y no la versión estándar de WPILibPi. La versión Romi tiene el sufijo `-Romi`. Vea la imagen de abajo para un ejemplo.



Configuración

El procedimiento para instalar la imagen se describe aquí : *WPILibPi Installation*.

Configuración de la red inalámbrica

Realice los siguientes pasos para que su Raspberry Pi esté lista para ser utilizada con la Romi:

1. Encienda la Romi deslizando el interruptor de encendido de la placa Romi 32U4 a la posición de encendido. La primera vez que se inicie con una nueva imagen tardará aproximadamente 2-3 minutos en arrancar mientras redimensiona el sistema de archivos y se reinicia. Las siguientes veces arrancará en menos de un minuto.
2. Usando su computadora, conéctese a la red Romi WiFi usando el SSID `WPILibPi-<number>` (donde``<number>está basado en el número de serie de la Raspberry Pi) con la frase de contraseña WPA2 `WPILib2021!`.

Nota: Si se enciende la Raspberry Pi en un entorno con múltiples WPILibPi Raspberry Pis, el SSID para una Raspberry Pi en particular también se anuncia de forma audible a través del puerto de auriculares. El SSID por defecto también se escribe en el archivo `/boot/default-ssid.txt` , que se puede leer insertando la tarjeta SD (a través de un lector) en un ordenador y abriendo `boot`

---

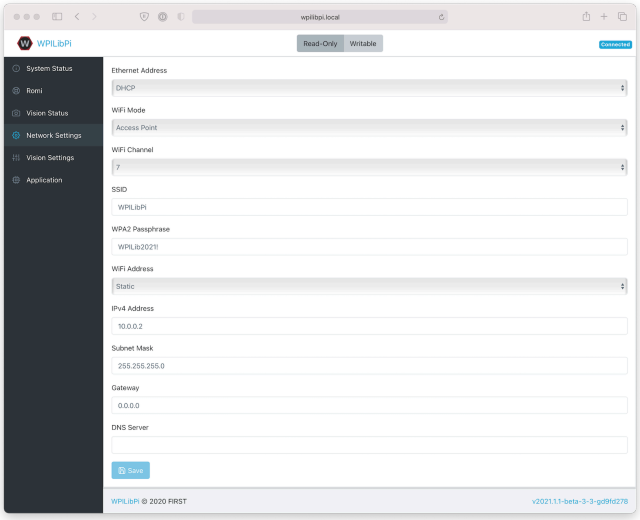
3. Abra un navegador web y conéctese a la dashboard de la Raspberry Pi en `http://10.0.0.2/` o `http://wpilibpi.local/`.

Nota: La imagen se inicia en modo de solo lectura de forma predeterminada, por lo que es necesario hacer clic en el botón `Escribir` para realizar cambios. Una vez que haya hecho los cambios, haga clic en el botón `Solo lectura` para evitar daños en la memoria.

---

4. Seleccione *Writable* en la parte superior de la dashboard de la página web.
5. Cambie la contraseña por defecto de su Romi estableciendo una nueva contraseña en el campo `WPA2 Passphrase`.
6. Pulse el botón *Save* en la parte inferior de la página para guardar los cambios.
7. Change the network SSID to a unique name if you plan on operating your Romi on a wireless network with other Romis.
8. Vuelva a conectarse a la red WiFi de la Romi con la nueva contraseña que estableció.

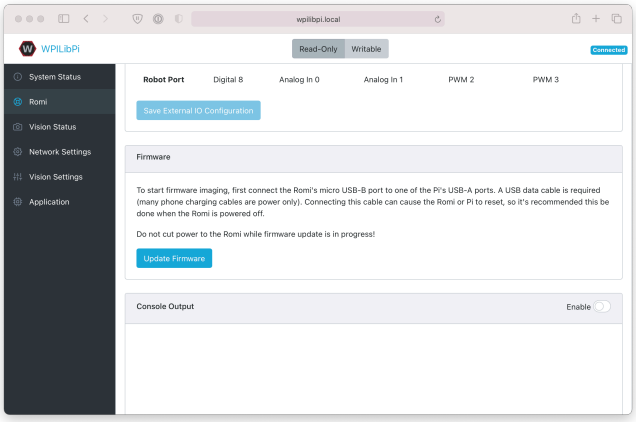
Asegúrese de poner la Dashboard en `Read-only` cuando todos los cambios hayan sido completados.



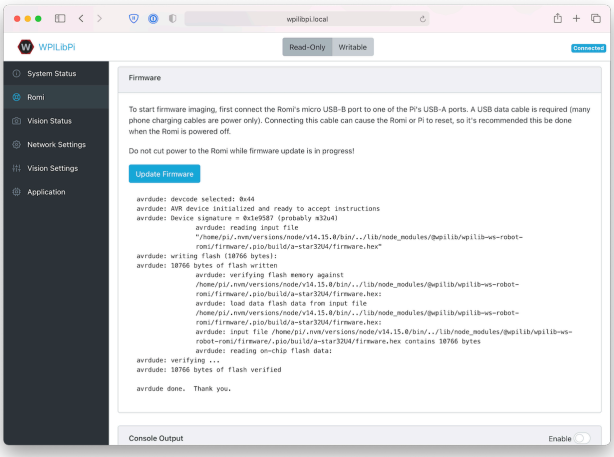
39.2.2 Tablero de control 32U4

La Raspberry Pi puede utilizarse ahora para escribir la imagen del firmware en la placa de control 32U4.

1. Apague la Romi
2. Conecte un cable USB A a micro-B desde uno de los puertos USB de la Raspberry Pi al puerto micro USB de la placa de control 32U4.
3. Encienda la Romi y conéctese a su red Wifi y conéctese a la dashboard como en los pasos anteriores.
4. En la página de configuración de `Romi`, pulse el botón *Update Firmware*.



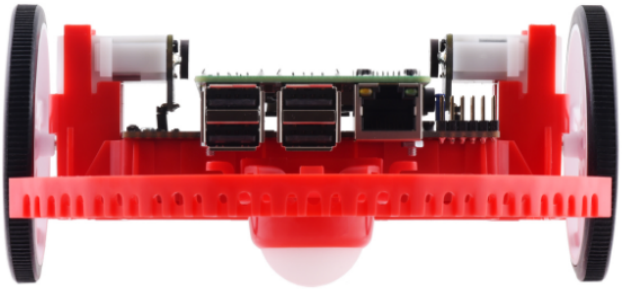
Aparecerá una consola que muestra un registro del proceso de despliegue del firmware. Una vez que el firmware ha sido desplegado a la Junta de Control de 32U4, aparecerá el mensaje `avrdude done. Thank you.`



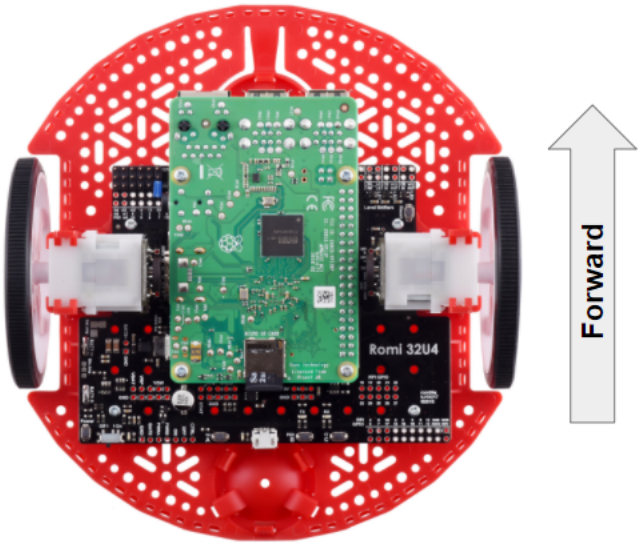
39.3 Conociendo a tu Romi

39.3.1 Convenciones sobre direcciones

El frente del Romi es donde se encuentran los puertos USB de la Raspberry Pi, los pines GPIO y la rueda giratoria.



En toda la documentación del Romi, las referencias a conducir hacia adelante utilizan la definición de arriba de «frente».



39.3.2 Hardware, Sensors, and GPIO

El Romi cuenta con los siguientes accesorios incorporados:

- 2x motorreductores con encoders
- 1x Unidad de Medición Inercial (IMU)
- 3x LEDs (verde, amarillo, rojo)
- 3x pulsadores (marcados A, B y C)
- 5x configurable GPIO channels (EXT)
- Buzzer

Nota: The Buzzer is currently not supported by WPILib.

Motors, Wheels, and Encoders

Los motores usados en el Romi tienen una reducción de 120:1, y una velocidad de salida sin carga de 150 RPM a 4.5V. La corriente libre que consumen es de 0.13 amperes y la corriente de motor en parada es de 1.25 amperes. El torque en parada es de 25 oz-in (0.1765 N-m) pero puede que el embrague de seguridad incorporado se resbale a torques más bajos.

Las llantas tienen un diámetro de 70mm (2.75»). Tienen un ancho de pista de 141mm (5.55»)

Los encoders están conectados directamente al eje de salida del motor y tienen 12 Cuentas Por Revolución (CPR). Con la reducción, esto da 1440 cuentas por revolución de la llanta.

The motor *PWM* channels are listed in the table below.

Channel	Componente de hardware del Romi
PWM 0	Left Motor
PWM 1	Right Motor

Nota: The right motor will spin in a backward direction when positive output is applied. Thus, the corresponding motor controller needs to be inverted in robot code.

The encoder channels are listed in the table below.

Channel	Componente de hardware del Romi
DIO 4	Canal A de cuadratura del encoder izquierdo
DIO 5	Canal B de cuadratura del encoder izquierdo
DIO 6	Canal A de cuadratura del encoder derecho
DIO 7	Canal B de cuadratura del encoder derecho

Nota: Por default, los encoders cuentan de manera positiva cuando el Romi se mueve hacia adelante.

Unidad de Medición Inercial

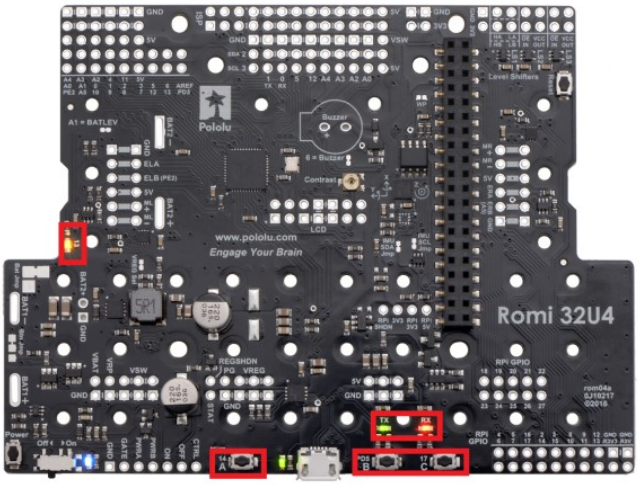
El Romi incluye una Unidad de Medición Inercial (IMU) STMicroelectronics LSM6DS33 que contiene un giroscopio de 3 ejes y un acelerómetro de 3 ejes.

The accelerometer has selectable sensitivity of 2G, 4G, 8G, and 16G. The gyro has selectable sensitivity of 125 Degrees Per Second (DPS), 250 DPS, 500 DPS, 1000 DPS, and 2000 DPS.

La interfaz de usuario web de Romi también proporciona un medio para calibrar el giroscopio y medir sus desplazamientos de cero antes de usarlo con el código del robot.

Onboard LEDs and Push Buttons

The Romi 32U4 control board has 3 push buttons and 3 LEDs onboard that are exposed as Digital IO (DIO) channels to robot code.

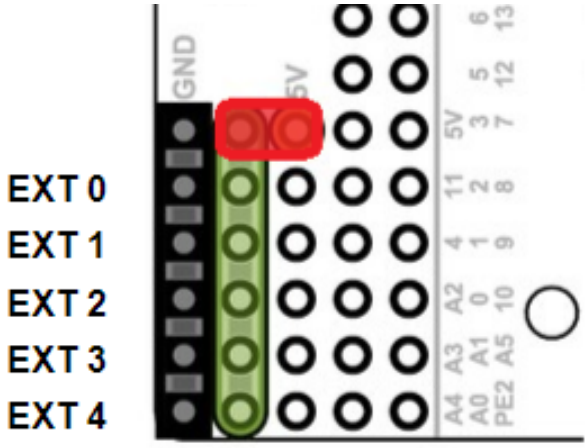


Canal DIO	Componente de hardware del Romi
DIO 0	Botón A (solo entrada)
DIO 1	Botón B (entrada), LED verde (salida)
DIO 2	Botón C (entrada), LED rojo (salida)
DIO 3	LED amarillo (solo salida)

Writes to DIO 0, 4, 5, 6 and 7 will result in a *no-op*.

Configurable GPIO Pins

The control board has 5 configurable GPIO pins (named EXT0 through EXT4) that allow a user to connect external sensors and actuators to the Romi.



All 5 pins support the following modes: Digital IO, Analog In, and PWM (with the exception of EXT 0, which only supports Digital IO and PWM). The mode of the ports can be configured with *The Romi Web UI*.

Los canales GPIO están expuestos a través de una interfaz de estilo servo de 3 pines, con conexiones para tierra, alimentación y señal (con la conexión a tierra más cercana al borde de la placa y la señal la más cercana al interior de la placa).

The power connections for the GPIO pins are initially left unconnected but can be hooked into the Romi's on-board 5V supply by using a jumper to connect the 5V pin to the power bus (as seen in the image above). Additionally, if more power than the Romi can provide is needed, the user can provide their own 5V power supply and connect it directly to power bus and ground pins.

GPIO Default Configuration

The table below shows the default configuration of the GPIO pins (EXT0 through EXT4). *The Romi Web UI* allows the user to customize the functions of the 5 configurable GPIO pins. The UI will also provide the appropriate WPILib channel/device mappings on screen once the IO configuration is complete.

Channel	Ext Pin
DIO 8	EXT0
Analog In 0	EXT1
Analog In 1	EXT2
PWM 2	EXT3
PWM 3	EXT4

39.4 Romi Hardware Support

The Romi robot, having a different hardware architecture than a roboRIO, is compatible with a subset of commonly used FRC control system components.

39.4.1 Compatible Hardware

In general, the Romi is compatible with the following:

- Simple Digital Input/Output devices (e.g. bumper switches, single LEDs)
- Standard RC-style *PWM* output devices (e.g. servos, PWM based motor controllers)
- Analog Input sensors (e.g distance sensors that report distance as a voltage)

39.4.2 Incompatible Hardware

Due to hardware limitations, the Romi Robot is not compatible with the following:

- Encoders other than the Romi-integrated encoders
- «Ping» style ultrasonic sensors (which require 2 DIO channels)
- Timing based sensors
- CAN based devices
- Romi built-in buzzer

39.4.3 Compatible Classes

All classes listed here are supported by the Romi Robot. If a class is not listed here, assume that it is not supported and *will not* work.

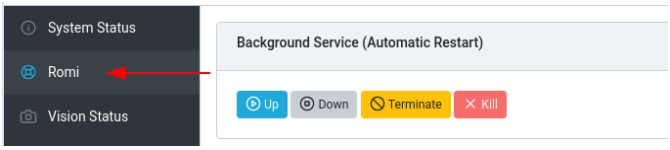
- PWM Motor Controllers (i.e. `Spark`)
- `Encoder`
- `AnalogInput`
- `DigitalInput`
- `DigitalOutput`
- `Servo`
- `BuiltInAccelerometer`

The following classes are provided by the Romi Vendordep.

- `RomiGyro`
- `RomiMotor`
- `OnboardIO`

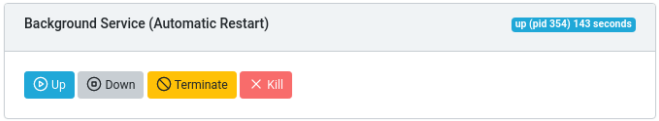
39.5 La Interfaz de Usuario web de Romi

La interfaz de usuario web de Romi viene instalada como parte de la imagen WPILibPi Raspberry Pi. Se puede acceder haciendo clic en la pestaña Romi en la barra de navegación de la interfaz de usuario web principal de WPILibPi.



El resto de esta sección abordará las distintas partes de la interfaz de usuario web de Romi y describirá su funcionalidad relevante.

39.5.1 Estado del servicio en segundo plano



Esta sección de la interfaz de usuario web de Romi proporciona información sobre el servicio web de Romi que se está ejecutando actualmente (que es lo que permite a WPILib hablar con Romi). La interfaz de usuario proporciona controles para activar o desactivar el servicio y muestra el tiempo de actividad actual del servicio web.

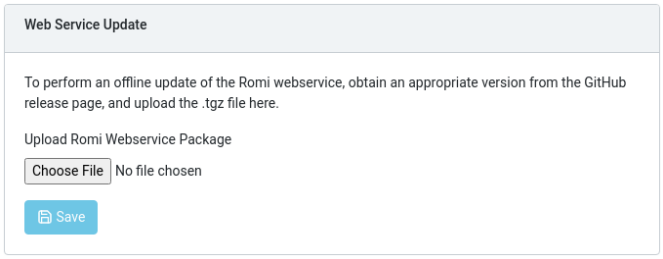
Nota: Los usuarios no necesitarán utilizar la funcionalidad de esta sección con frecuencia, pero puede ser útil para solucionar problemas.

39.5.2 Estado de Romi

Esta sección provee información sobre Romi, incluyendo la versión de servicio, voltaje de batería, y si el firmware instalado al momento en el tablero 32U4 de Romi es compatible con la versión actualizada del servicio web.

Nota: Si el firmware no es compatible, vea la sección en *Imaging your Romi*

39.5.3 Actualización del servicio web

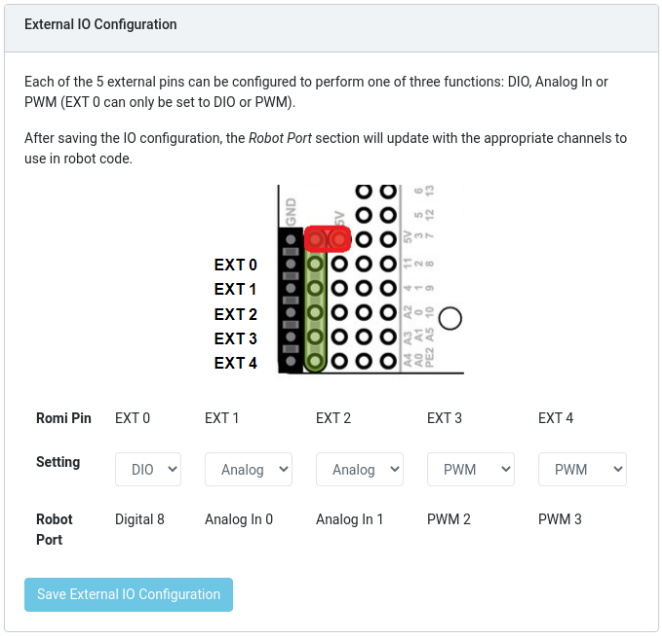


Nota: La Raspberry Pi debe estar en modo **Writable** para que esta sección funcione.

La imagen Romi WPILibPi se envía con la última versión (en el momento de la publicación) del servicio web Romi. Para admitir la actualización a versiones más nuevas del servicio web Romi, esta sección permite a los usuarios cargar un paquete prediseñado que se puede obtener a través del servicio web Romi *GitHub releases page <https://github.com/wpilibsuite/wpilibws-robot-romi/releases> __*.

Para realizar una actualización, descargue el archivo .tgz apropiado de la página de versiones de GitHub. A continuación, seleccione el archivo .tgz descargado y haga clic en :guilabel: *Guardar*. El paquete de servicio web actualizado se cargará en Raspberry Pi y se instalará. Después de un breve momento, la sección Romi Status debería actualizarse con la información de la última versión.

39.5.4 Configuración IO externa



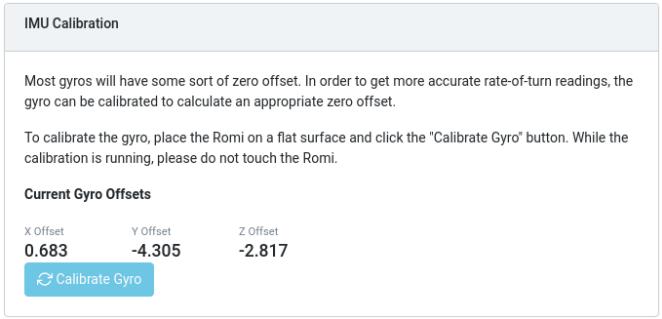
Esta sección permite a los usuarios configurar los 5 canales GPIO externos en la Romi.

Nota: La Raspberry Pi debe estar en modo **Writable** para que esta sección funcione.

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 fila «Robot Port» proporciona el mapeo WPILib apropiado para cada canal GPIO configurado. Por ejemplo, EXT 0 está configurado como un canal de IO digital y será accesible en WPILib como un canal 8 de entrada digital (o salida digital).

39.5.5 Calibración IMU



Nota: La Raspberry Pi debe estar en modo **Writable** para que esta sección funcione.

Esta sección permite a los usuarios calibrar el giróscopo en el Romi. Los giroscopios suelen tener algún tipo de error de compensación cero, y la calibración permite a Romi calcular la compensación y utilizarla en los cálculos.

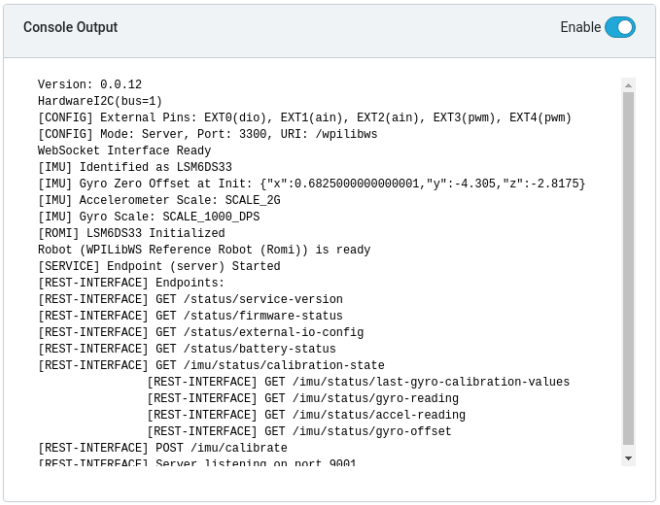
Para comenzar la calibración, coloque el Romi en una superficie plana y estable. Luego, haga clic en el botón :guilabel: *Calibrate Gyro*. Aparecerá una barra de progreso que muestra el proceso de calibración actual. Una vez que se completa la calibración, los últimos valores de compensación se mostrarán en la pantalla y se registrarán con el servicio web Romi.

Estos valores de compensación se guardan en el disco y persisten entre reinicios.

39.5.6 Firmware

Nota: Consulte la sección sobre :doc: *Imaging your Romi </docs/romi-robot/imaging-romi>*

39.5.7 Salida de consola



Cuando está habilitada, esta sección permite a los usuarios ver la salida de consola sin procesar que el servicio web de Romi proporciona. Esto es útil para solucionar problemas con Romi, o solamente para tener más información sobre qué más sucede detrás de escenas,

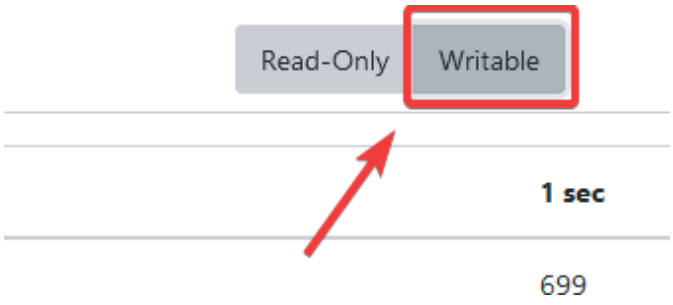
39.5.8 Modo Bridge

El modo Bridge permite que su robot Romi se conecte a una red WiFi, en lugar de actuar como un Punto de Acceso (AP). Esto es especialmente útil en entornos de aprendizaje remotos, ya que puede utilizar Internet mientras usa el Romi sin necesidad de hardware adicional.

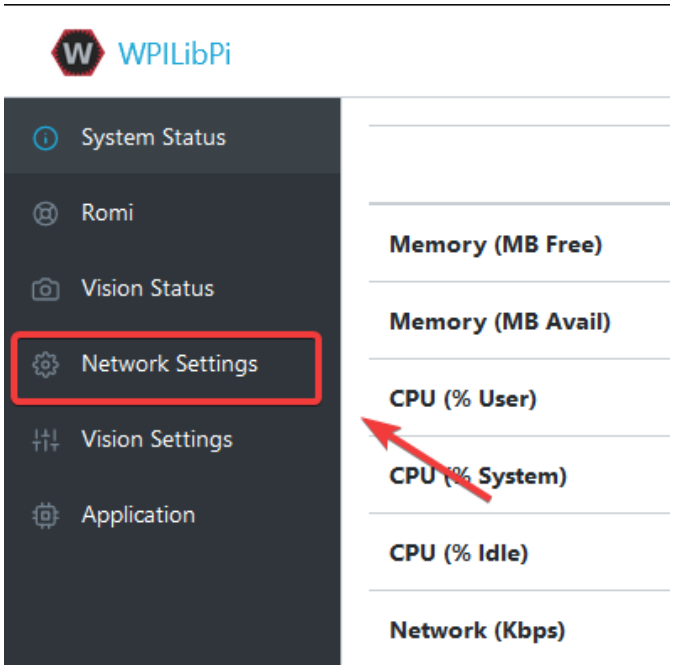
Nota: Es probable que el modo Bridge no funcione correctamente en entornos de red restringidos (instituciones educativas).

---

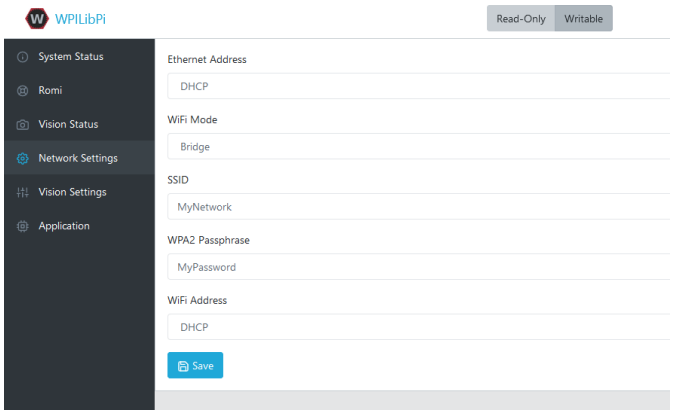
1. Active *Writable* en el menú superior.



2. Haga clic en *Network Settings*.



3. Deben aplicarse los siguientes ajustes de red:



- **Ethernet**: DHCP
- **Modo WiFi**: Bridge
- **SSID**: SSID (nombre) de su red
- **WPA2 Frase de Acceso**: Contraseña de su red wifi

- **Dirección WiFi**: DHCP

Una vez aplicada la configuración, reinicie la Romi. Ahora debería poder navegar a `wpilibpi.local` en su navegador web mientras está conectado a su red especificada.

No se puede acceder a Romi

Si la Romi tiene la configuración correcta del puente y usted no puede acceder a ella, tenemos algunas soluciones.

- Ethernet en la Romi
- Reimaginar la Romi

Algunas redes restringidas pueden interferir con la resolución del nombre de host del Romi, usted puede solucionar esto usando Angry IP Scanner para encontrar la dirección IP.

> **Advertencia:** Angry IP Scanner es marcado por algunos antivirus como software espía, ya que hace pings a los dispositivos de su red. Es una aplicación segura.

39.6 Programando el Romi

Escribir un programa para Romi es muy parecido a escribir un programa para un robot regular de FRC. De hecho, todas las mismas herramientas (Visual Studio Code, Driver Station, SmartDashboard, etc) pueden usarse con Romi.

39.6.1 Crear un programa Romi

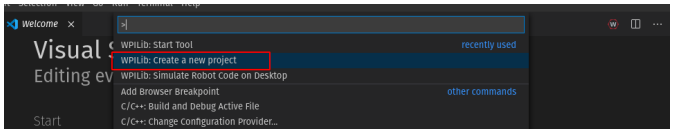
Creating a new program for a Romi is like creating a normal FRC program, similar to the *Zero To Robot* programming steps.

WPILib comes with two templates for Romi projects, including one based on TimedRobot, and a Command-Based project template. Additionally, an example project is provided which showcases some of the built-in functionality of the Romi. This article will walk through creating a project from this example.

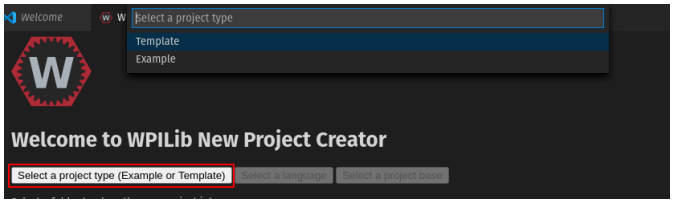
Nota: In order to program the Romi using C++, a compatible C++ desktop compiler must be installed. See *Robot Simulation - Additional C++ Dependency*.

Crear un nuevo proyecto WPILib Romi

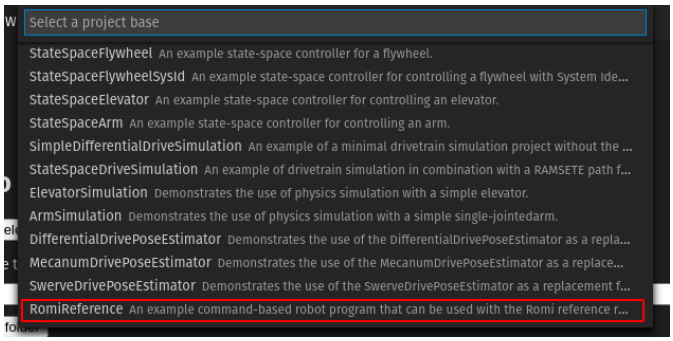
Abra la paleta de comandos de Visual Studio Code con `Ctrl+Shift+P` y escriba «Nuevo proyecto» en el indicador. Seleccione el comando «Crear un nuevo proyecto»:



Esto abrirá la «Ventana del Creador de nuevos proyectos». Desde aquí, haga clic en «Seleccionar un tipo de proyecto (ejemplo o plantilla) y elija «Ejemplo» en el mensaje que aparece:



A continuación, aparecerá una lista de ejemplos. Desplácese por la lista para encontrar el ejemplo «RomiReference»:



Complete el resto de los campos en el «Creador de nuevo proyecto» y haga clic en «Generar proyecto» para crear el nuevo proyecto de robot.

Ejecución de un programa Romi

Once the robot project is generated, it is essentially ready to run. The project has a pre-built `Drivetrain` class and associated default command that lets you drive the Romi around using a joystick.

One aspect where a Romi project differs from a regular FRC robot project is that the code is not deployed directly to the Romi. Instead, a Romi project runs on your development computer and leverages the WPILib simulation framework to communicate with the Romi robot.

To run a Romi program, first, ensure that your Romi is powered on. Next, connect to the `WPILibPi-<number>` WiFi network broadcast by the Romi. If you changed the Romi network settings (for example, to connect it to your own WiFi network) you may change the IP address that your program uses to connect to the Romi. To do this, open the `build.gradle` file and update the `wpi.sim.envVar` line to the appropriate IP address.

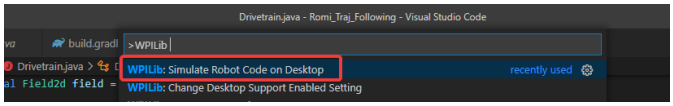
```
43 //Sets the websocket client remote host.
44 wpi.sim.envVar("HALSIMWS_HOST", "10.0.0.2")
```

(continúe en la próxima página)

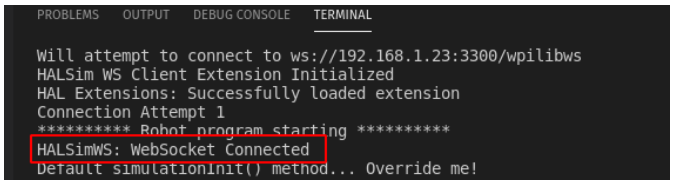
(proviene de la página anterior)

```
wpi.sim.addWebsocketsServer().defaultEnabled = true
wpi.sim.addWebsocketsClient().defaultEnabled = true
```

Now to start your Romi robot code, open the WPILib Command Palette (type `Ctrl+Shift+P`) and select «Simulate Robot Code», or press `F5`.



Si todo va bien, debería ver una línea en la salida de la consola que dice «HALSimWS: WebSocket Connected»:



¡Tu código Romi ya se está ejecutando!

39.7 Programming the Romi (LabVIEW)

Writing a LabVIEW program for the Romi is very similar to writing a program for a regular roboRIO based robot. In fact, all the same tools can be used with the Romi.

39.7.1 Creating a Romi Project

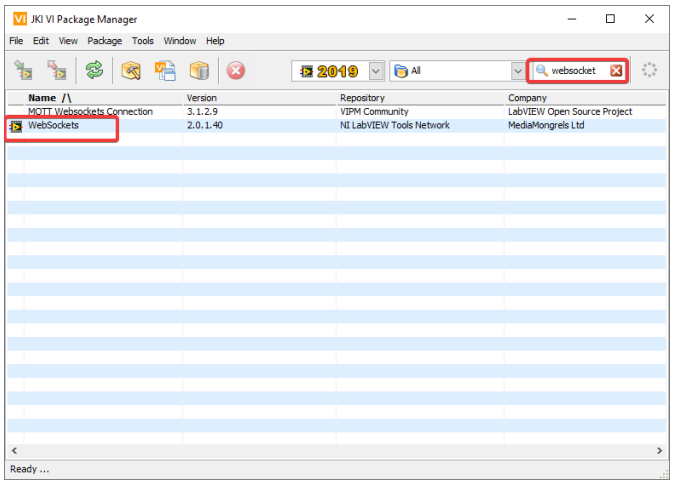
Creating a new program for a Romi is no different than creating a normal FRC |reg| program, similar to the *Zero To Robot* programming steps. Initially, you may wish to create a separate project for use on just the Romi as the Romi hardware may be connected to different ports than on your roboRIO robot.

The Romi Robot used *PWM* ports 0 and 1 for left and right side respectively.

Installing the WebSockets VI

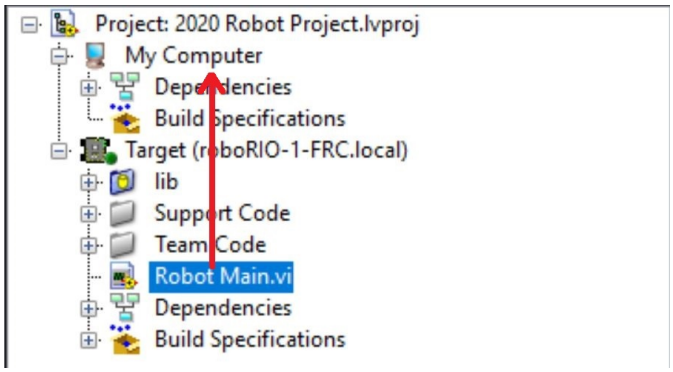
One aspect where a Romi project differs from a regular FRC |reg| robot project is that the code is not deployed directly to the Romi. Instead, a Romi project runs on your development computer, and leverages the WPILib simulation framework to communicate with the Romi robot. WebSockets is the protocol that LabVIEW uses to converse with the Romi.

Open the *VI Package Manager* application. Type `websockets` into the search box in the top right. Select the VI by *LabVIEW Tools Network*.



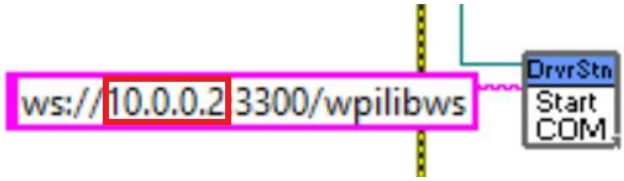
Changing the Project Target

The primary step needed to run your LabVIEW program on the Romi is to change the target to the Desktop. To change the project target, locate the Robot Main VI in the Project Explorer and click and drag it from the `Target` section to the `My Computer` section.



Setting the Target IP

By default, your LabVIEW program will attempt to connect to a Romi with the IP address of `10.0.0.2`. If you wish to use a different IP, you can specify it as an input to the `Driver Station Start Communication VI` inside `Robot Main`. Locate the pink input terminal for `Simulation URL` then right-click and select *Create Constant* to create a constant pre-filled with the default value. You can then modify the IP address portion of the text, taking care to leave the protocol section (at the beginning) and port and suffix (at the end) the same.



Ejecución de un programa Romi

To run a Romi program, first, ensure that your Romi is powered on. Once you connect to the `WPILibPi-<number>` network broadcast by the Romi, press the white *Run* arrow to start running the Romi program on your computer.

Your Romi code is now running! The program will automatically attempt to connect to either the IP you have specified, or the default if you have not specified an IP.

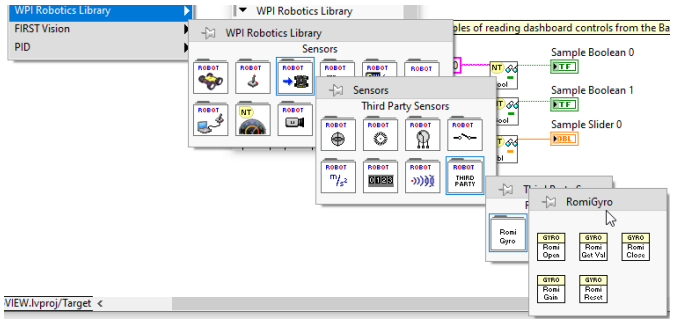
It is recommended to run the Driver Station software on the same computer as the LabVIEW code. Once your program successfully connects to the Driver Station, it will automatically notify the Driver Station that the code is running on the Desktop, allowing the Driver Station to connect without you changing any information inside the Driver Station. Next, you'll need to point the Driver Station to your Romi. This is done by setting the team number to `127.0.0.1`. You can then use the controls in the Driver Station to set the robot mode and enable/disable as normal.

Nota: If your robot code is unable to connect to the Romi, the Driver Station will also show no connectivity.

Using the Gyro or Encoder

The gyro that is available on the Romi is available using the RomiGyro functions. This is located under

```
- WPI Robotics Library
   - Sensors
      - Third Party Libraries
         - RomiGyro
```



The encoders can be used using the standard encoder function. The DIO ports are:

- Left (4, 5)
- Right (6, 7)

40

Comenzando con 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.

El XRP provee un caso de uso similar al *Romi* con funcionalidad similar, aunque utiliza un procesador de menor potencia y es en general más económico.

40.1 XRP Hardware, Assembly and Imaging

To get started with the XRP, you will need to have the necessary hardware.

1. XRP Kit from SparkFun or from DigiKey - Available at a discount for educational institutions or FIRST teams. See individual vendors for details.
2. Micro-USB cable - Ensure that this is a data cable
3. 4 AA batteries - Rechargeable (example) is best (don't forget the charger)

40.1.1 Assembly

Nota: See the assembly instructions on the XRP User Guide.

You should follow the instructions up to and including the point where the XRP arm is mounted to the servo.

40.1.2 Imaging your XRP

The XRP uses a Raspberry Pi Pico W as its main processor. A special firmware will need to be installed so that the robot operates properly.

Download

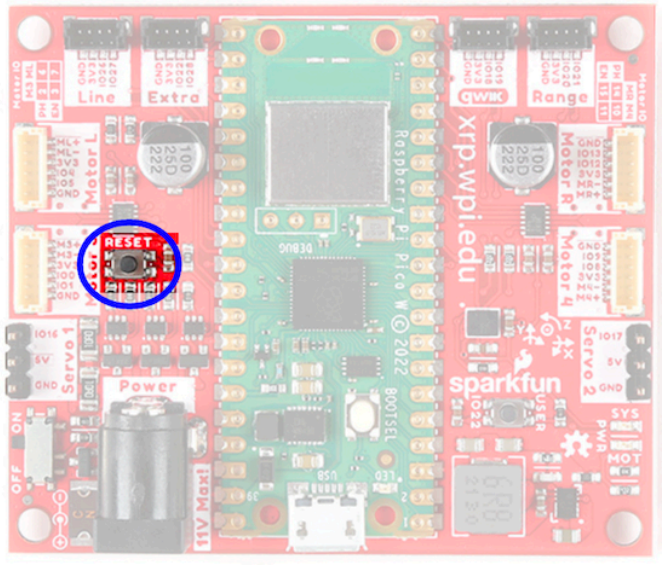
The XRP firmware must be downloaded and written to the Pico W. Click on `Assets` at the bottom of the description to see the available image files:

XRP-WPILib Firmware

Imaging

To image the XRP, perform the following steps:

1. Extract the contents of the firmware ZIP file. You should end up with a `.uf2` file.
2. Plug the XRP into your computer with a Micro-USB cable. You should see a red power LED that lights up.
3. While holding the `BOOTSEL` button (the white button on the green Pico W, near the USB connector), quickly press the reset button (circled below), and then release the `BOOTSEL` button.

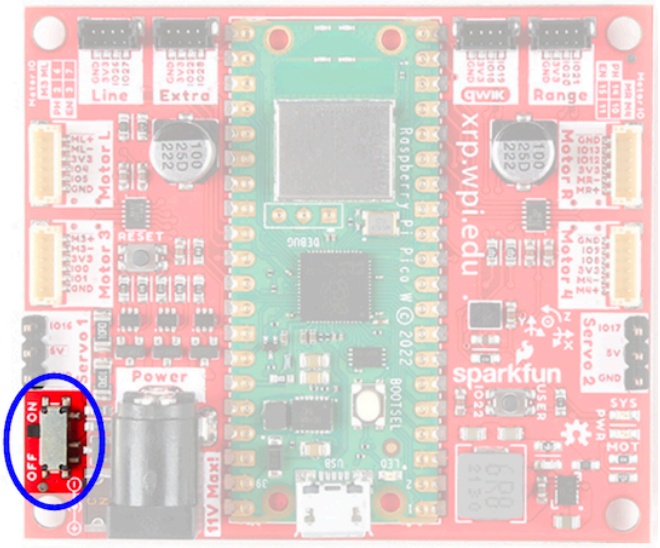


4. The board will temporarily disconnect from your computer, and then reconnect as a USB storage device named `RPI-RP2`.
5. Drag the `.uf2` firmware file into the `RPI-RP2` drive, and it will automatically update the firmware.
6. Once complete, the `RPI-RP2` USB storage device will disconnect. At this point, you can disconnect the XRP board from your computer and run it off battery power.

First Boot

Perform the following steps to get your XRP ready for use:

1. Ensure that you have 4 AA batteries installed
2. Turn the XRP on by sliding the power switch (circled below) on the XRP board to the on position. A red power LED will turn on.



3. Using your computer, connect to the XRP WiFi network using the SSID `XRP-<IDENT>` (where `<IDENT>` is based on the unique ID of the Pico W) with the WPA2 passphrase `xrp-wpilib`.

Nota: If powering on the XRP in an environment with multiple other XRPs, the SSID can also be found by connecting the XRP to a computer, navigating to the USB storage device (`PICODISK`) that appears and opening the `xrp-status.txt` file.

---

4. Open a web browser and connect to the web UI at `http://192.168.42.1:5000`. If the page loads, you have established connectivity with the XRP.

Nota: More information about the Web UI and configuration can be found in the *Web UI section*.

40.2 Getting to know your XRP

40.2.1 Booting up the XRP

Upon start up (when power is applied to the XRP either via battery or USB), the following will happen:

1. The IMU will calibrate itself. This lasts approximately 3-5 seconds, and will be indicated by the green LED blinking rapidly. Ideally, the XRP should be placed on a flat surface prior to power up, and if necessary, users can hit the reset button to restart the firmware and IMU calibration process.
2. The network will be configured, depending on the configuration settings. See the section on *the Web UI* for more information on how to configure the network settings. By default the XRP will broadcast its own WiFi Access Point.
3. After this, the XRP is ready for use.

40.2.2 Hardware, Sensors and GPIO

The XRP has the following built-in hardware/peripherals:

- 2x geared drive motors with encoders
- 2x additional geared motor connectors with encoder support (marked Motor3 and Motor4)
- 2x Servo connectors (marked Servo1 and Servo2)
- 1x Inertial Measurement Unit (IMU)
- 1x LED (green)
- 1x pushbutton (marked USER)
- 1x Line following sensor (exposed as 2 Analog inputs)
- 1x Ultrasonic PING style rangefinder (uses 2 digital IO pins, exposed as an analog input)

Motors, Wheels, and Encoders

The motors used on the XRP have a 48.75:1 gear reduction and a no-load output speed of 90 RPM at 4.5V.

The wheels have a diameter of 60mm (2.3622»). They have a trackwidth of 155mm (6.1»).

The encoders are connected directly to the motor output shaft and have 12 Counts Per Revolution (CPR). With the provided gear ratio, this nets 585 counts per wheel revolution.

The motor channels are listed in the table below.

Nota: We use «motor channels» here instead of «PWM channels» as the XRP requires the use of a special `XRPMotor` object in WPILib code to interact with the hardware.

Channel	XRP Hardware Component
XRPMotor 0	Left Motor
XRPMotor 1	Right Motor
XRPMotor 2	Motor 3
XRPMotor 3	Motor 4

Nota: The right motor will spin in a backward direction when positive output is applied. Thus the corresponding motor controller needs to be inverted in robot code.

The servo channels are listed in the table below.

Nota: We use «servo channels» here instead of «PWM channels» as the XRP requires the use of a special `XRPServo` object in WPILib code to interact with the hardware.

Channel	XRP Hardware Component
XRPServo 4	Servo 1
XRPServo 5	Servo 2

The encoder channels are listed in the table below.

Channel	XRP Hardware Component
DIO 4	Left Encoder Quadrature Channel A
DIO 5	Left Encoder Quadrature Channel B
DIO 6	Right Encoder Quadrature Channel A
DIO 7	Right Encoder Quadrature Channel B
DIO 8	Motor3 Encoder Quadrature Channel A
DIO 9	Motor3 Encoder Quadrature Channel B
DIO 10	Motor4 Encoder Quadrature Channel A
DIO 11	Motor4 Encoder Quadrature Channel B

Nota: By default, the encoders count up when the XRP moves forward.

Inertial Measurement Unit

The XRP includes an STMicroelectronics LSM6DSOX Inertial Measurement Unit (IMU) which contains a 3-axis gyro and a 3-axis accelerometer.

The XRP will calibrate the gyro and accelerometer upon each boot (the onboard LED will quickly flash for about 3-5 seconds at startup time).

Onboard LED and Push Button

The XRP has a push button (labeled USER) and a green LED onboard that are exposed as Digital IO (DIO) channels to robot code.

DIO Channel	XRP Hardware Component
DIO 0	USER Button
DIO 1	Green LED

Nota: DIO 2 and 3 are reserved for future system use.

Line Following (Reflectance) Sensor

When assembled according to the instructions, the XRP supports a line following sensor with 2 sensing elements. Each sensing element measures reflectance exposes these as AnalogInput channels to robot code. The returned values range from 0V (pure white) to 5V (pure black).

AnalogInput Channel	XRP Hardware Component
AnalogInput 0	Left Reflectance Sensor
AnalogInput 1	Right Reflectance Sensor

Ultrasonic Rangefinder

When assembled according to the instructions, the XRP supports an ultrasonic, PING style, rangefinder. This is exposed as an AnalogInput channel to robot code. The returned values range from 0V (20mm) to 5V (4000mm).

AnalogInput Channel	XRP Hardware Component
AnalogInput 2	Ultrasonic Rangefinder

40.3 XRP Hardware Support

The XRP robot, having a different hardware architecture than a roboRIO, is compatible with a subset of commonly used FRC control system components.

40.3.1 Compatible Hardware

In general, the XRP is compatible with the following:

- Hobby DC motors with built-in encoders (6-pin connector)
- Standard RC-style *PWM* output devices (e.g. servos, PWM based motor controllers)
- «Ping» style ultrasonic sensors (only when connected to the RANGE port)

40.3.2 Incompatible Hardware

Due to hardware limitations, the XRP is not compatible with the following:

- Encoders other than those already integrated into hobby motors
- Timing based sensors
- CAN based devices

40.3.3 Compatible Classes

All classes listed here are supported by the XRP. If a class is not listed here, assume that it is not supported and *will not* work.

- `Encoder`
- `AnalogInput`
- `DigitalInput`
- `DigitalOutput`
- `BuiltInAccelerometer`

Nota: The PWM motor controller classes (e.g. `Spark`) and `Servo` are not supported. The XRP requires use of specialized `XRPMotor` and `XRPServo` classes.

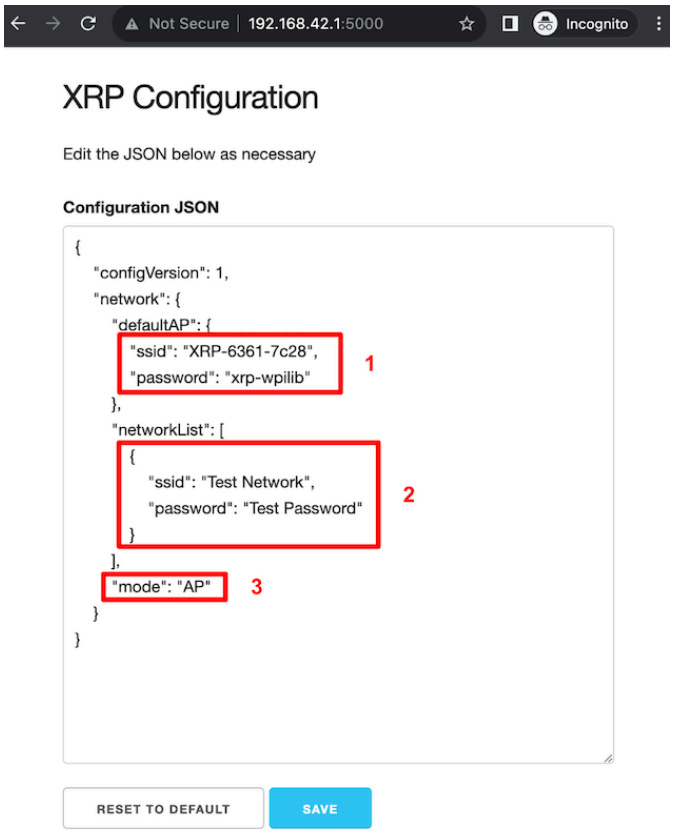
The following classes are provided by the XRP Vendordep (built-in to WPILib).

- `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.

Nota: 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 Programming the XRP

Writing a program for the XRP is very similar to writing a program for a regular FRC robot. In fact, all the same tools (Visual Studio Code, Driver Station, SmartDashboard, etc) can be used with the XRP.

40.5.1 Creating an XRP Program

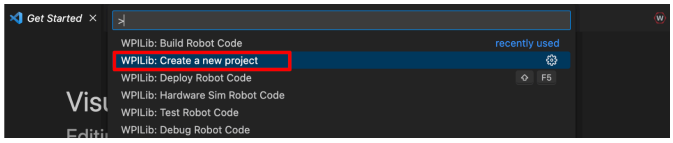
Creating a new program for an XRP is like creating a normal FRC program, similar to the *Zero To Robot* programming steps.

WPILib comes with two templates for XRP projects, including one based on TimedRobot, and a Command-Based project template. Additionally, an example project is provided which showcases some of the built-in functionality of the XRP, and shows how to use the vendordep exposed XRP classes. This article will walk through creating a project from this example.

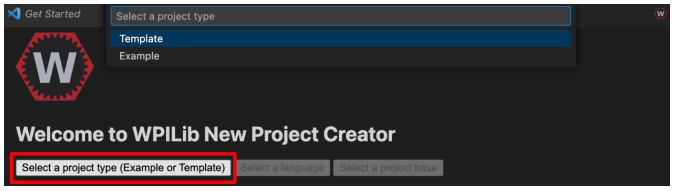
Nota: In order to program the XRP using C++, a compatible C++ desktop compiler must be installed. See *Robot Simulation - Additional C++ Dependency*.

Creating a New WPILib XRP Project

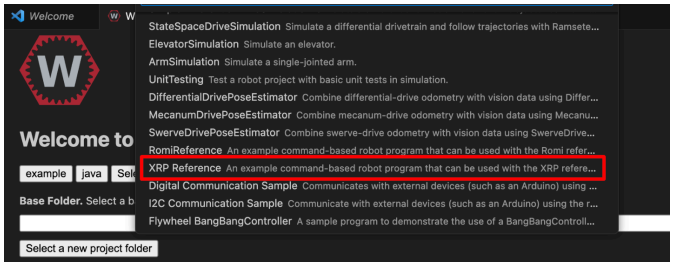
Bring up the Visual Studio Code command palette with `Ctrl+Shift+P`, and type «New project» into the prompt. Select the «Create a new project» command:



This will bring up the «New Project Creator Window». From here, click on «Select a project type (Example or Template)», and pick «Example» from the prompt that appears:



Next, a list of examples will appear. Scroll through the list to find the «XRP Reference» example:



Fill out the rest of the fields in the «New Project Creator» and click «Generate Project» to create the new robot project.

Running an XRP Program

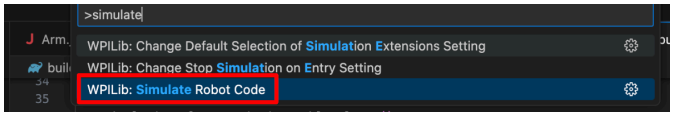
Once the robot project is generated, it is essentially ready to run. The project has a pre-built `Drivetrain` class and associated default command that lets you drive the XRP around using a joystick.

One aspect where an XRP project differs from a regular FRC robot project is that the code is not deployed directly to the XRP. Instead, an XRP project runs on your development computer and leverages the WPILib simulation framework to communicate with the XRP.

To run an XRP program, first, ensure that your XRP is powered on. Next, connect to `XRP-<IDENT>` WiFi network broadcast by the XRP. If you changed the XRP network settings (for example, to connect it to your own network), you may change the IP address that your program uses to connect to the XRP. To do this, open the `build.gradle` file and update the `wpi.sim.envVar` line to the appropriate IP address.

```
43 //Sets the XRP Client Host
44 wpi.sim.envVar("HALSIMXRP_HOST", "192.168.42.1")
45 wpi.sim.addXRPClient().defaultEnabled = true
```

Now to start your XRP robot code, open the WPILib Command Palette (type `Ctrl+Shift+P`) and select «Simulate Robot Code», or press `F5`.



If all goes well, you should see the simulation GUI pop up and see the gyro and accelerometer values updating.

Your XRP code is now running!

41

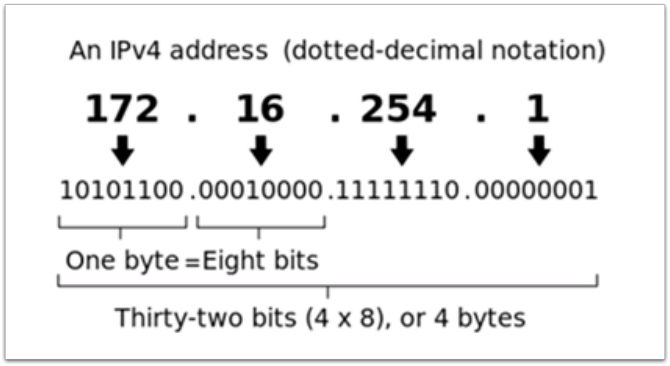
Introducción a las Redes

Esta sección describe la configuración básica del robot y el uso relacionado con la comunicación entre la driver station y la roboRIO.

41.1 Conceptos Básicos de Redes

41.1.1 ¿Qué es una Dirección IP?

Una dirección IP es una cadena única de números, separados por periodos que identifican cada dispositivo en la red. Cada dirección IP está dividida en 4 secciones (octetos) desde 0-255.



Como se muestra anteriormente, esto significa que cada dirección IP es una dirección de 32-bits, esto quiere decir que hay 2^{32} direcciones, o cerca de 4,300,000,000 direcciones posibles. Como sea, la mayoría de éstas son usadas públicamente para cosas como servidores web.

Esto trae nuestro **primer punto clave** de direccionamiento de IP: Cada dispositivo en la red tiene que tener una dirección IP única. Dos dispositivos no pueden tener la misma dirección IP, de otra manera pueden ocurrir colisiones.

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.

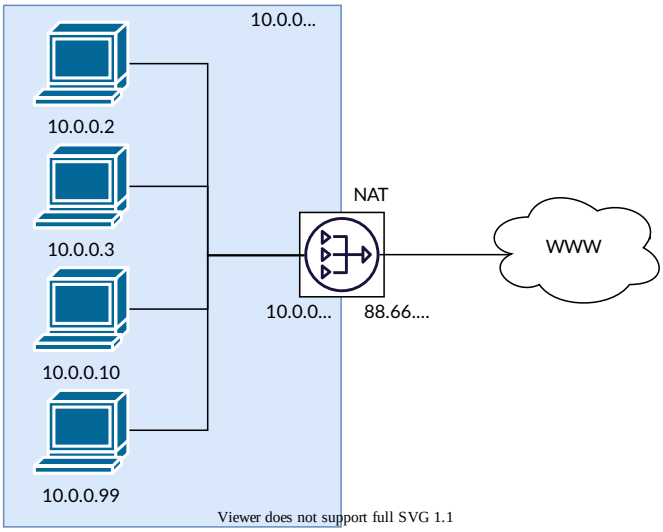
41.1.2 Direcciones Públicas vs Privadas

Para ser eficientes con el uso de las direcciones IP, la idea de «Rangos de IP reservados» fue implementada. En resumen, esto significa que hay rangos de direcciones IP que no serán asignados a servidores web, y serán solamente usados por redes locales, como las de su casa.

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:

Clase	Bits	Inicio de Dirección	Fin de Dirección	Cantidad de Direcciones
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

Estos rangos reservados nos permiten asignar una «dirección IP no reservada» a toda una casa, y luego utilizar múltiples direcciones en un rango reservado para conectar más de un ordenador a Internet. Un proceso en el router de internet de la casa conocido como **NAT** (Network Address Translation), maneja el proceso de mantener un registro de qué IP privada está solicitando datos, usando la IP pública para solicitar esos datos desde internet, y luego pasando los datos devueltos a la IP privada que los solicitó. Esto nos permite utilizar las mismas direcciones IP reservadas para muchas redes locales, sin causar ningún conflicto. A continuación se presenta una imagen de este proceso.



Nota: For the FRC® networks, we will use the `10.0.0.0` range. This range allows us to use the `10.TE.AM.xx` format for IP addresses, whereas using the Class B or C networks would only allow a subset of teams to follow the format (*TE.AM IP Notation*).

41.1.3 ¿Cómo se asignan éstas direcciones?

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).

Dinámicamente

Asignar una dirección IP dinámicamente quiere decir que estamos dejando un dispositivo en la red manejar las asignaciones de la dirección IP. Esto se hace mediante el Protocolo de Configuración de Huésped Dinámico (DHCP, por sus siglas en inglés). DHCP tiene muchos componentes a el, pero para el alcance de éste documento, vamos a pensar en el como un servicio que automáticamente maneja la red. Cuando se conecta a un nuevo dispositivo a la red, el servicio DCHP ve el nuevo dispositivo, después le proporciona con una dirección IP disponible y la otra configuración de red necesaria para que se comunique el dispositivo. Esto puede significar que hay veces que no sabemos la IP exacta de cada dispositivo.

¿Qué es un servidor 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.

Esto significa que si alguna vez está en una situación donde necesita tener un servidor DHCP asignando direcciones IP a sus dispositivos de red, es simple como encontrar el enrutador doméstico más cercano, y conectarlo.

Estáticamente

Asignar direcciones IP estáticamente quiere decir que hay que decir manualmente a cada dispositivo en la red cual es la dirección IP que queremos que tenga. Esta configuración ocurre por un ajuste en cada dispositivo. Deshabilitando DHCP en la red y asignando las direcciones manualmente, tenemos el beneficio de saber la dirección IP de cada dispositivo en la red, pero porque nosotros establecemos cada uno manualmente y no hay servicio dando seguimiento de la dirección IP utilizada, nosotros tenemos que dar seguimiento de esto nosotros mismos. Mientras establecemos direcciones IP estáticamente, tenemos que ser cuidadosos con no asignar direcciones duplicadas, y debemos estar seguros que estamos estableciendo otro ajuste de red (como mascara de subconjunto y puerta de enlace predeterminada) correctamente en cada dispositivo.

41.1.4 ¿Qué es un enlace local?

Si un dispositivo no tiene dirección IP, entonces no se puede comunicar a la red. Esto se puede convertir en un problema si tenemos un dispositivo que está ajustado de manera dinámica adquiere su dirección de un servidor DHCP, pero no hay algún servidor DHCP en la red. Un ejemplo de esto puede ser cuando se tiene una laptop directamente conectada a una roboRIO y ambas están ajustadas para adquirir una dirección IP dinámicamente. Ningún dispositivo es un servidor DHCP, y a pesar de ser los únicos dos dispositivos conectados en la red, no serán asignadas automáticamente las direcciones IP.

Las direcciones link-local nos dan un conjunto estándar de direcciones al que podemos recurrir si un dispositivo configurado para adquirir dinámicamente no es capaz de adquirir una dirección. Si esto ocurre, el dispositivo se asignará a sí mismo una dirección IP en el rango de direcciones `169.254.xx.yy`; esto es una dirección link-local. En nuestro ejemplo del roboRIO y el ordenador, ambos dispositivos se darán cuenta de que no se les ha asignado una dirección IP y se asignarán una dirección link-local. Una vez que se les hayan asignado direcciones en el rango `169.254.xx.yy`, estarán en la misma red y podrán comunicarse, aunque hayan sido configurados como dinámicos y un servidor DHCP no les haya asignado direcciones.

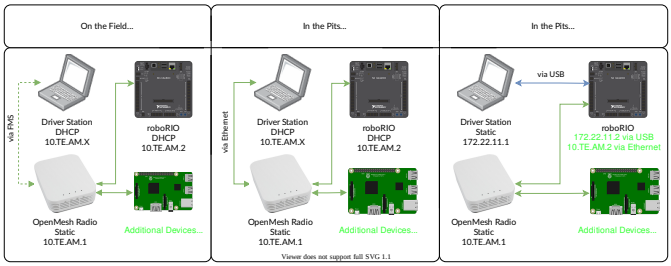
41.1.5 Direccionando IP para FRC

Vea *IP Networking Article* para más información

Combinando Configuraciones Dinámicas y Estáticas

Mientras se está en la cancha, el equipo no debe ver algún problema con tener los dispositivos en modo estático en el rango `10.TE.AM.xx`, y la cancha tener que asignar direcciones IP mientras que no existan conflictos de direcciones IP referentes a la sección de arriba.

En los pits, el equipo puede encontrarse con problemas con mezclar dispositivos estáticos y DHCP por la siguiente razón. Como se mencionó arriba, los dispositivos DHCP pueden retroceder a una dirección de enlace local (`169.254.xx.yy`) si el servidor no está presente. Para dispositivos estáticos, la dirección IP va a ser siempre la misma. Si el servidor DHCP no está presente y la roboRIO, driver station y la laptop retroceden a una dirección de enlace local, los dispositivos ajustados estáticamente en el rango 10.TE.AM.xx van a estar en diferente red y no estarán visibles en esas direcciones de enlace local. Una explicación visual se muestra abajo:



Advertencia: Cuando se esta conectado vía USB a la roboRio, una configuaración *Redirecccionamiento de puertos* es requerida para entrar a los dispositivos conectados a la radio OpenMesh (en la red verde mostrada abajo).

Available Network Ports

Please see R704 of the 2024 Game Manual for information regarding available network ports.

41.1.6 mDNS

mDNS, o Sistema de Nombres de Dominio de multidifusión es un protocolo que nos permite el beneficio de las características de un DNS, sin tener un servidor DNS en la red. Para hacer esto más claro, vamos a dar un paso para atrás y hablar que es un DNS.

¿Qué es un DNS?

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.

Ejemplo 2: En la red de cada, tiene un servidor llamado `MYCOMPUTER` que quiere conectarlo desde su computadora. Su red usa DHCP para que no sepa la dirección IP de `MYCOMPUTER`, pero el DNS permite que se conecte por solo usar el nombre `MYCOMPUTER`. Adicionalmente, cuando las asignaciones del DHCP se vuelven a cargar, `MYCOMPUTER` puede terminar con una dirección diferente, pero porque se está conectando usando el nombre `MYCOMPUTER` en vez de una dirección IP específica, el registro del DNS fue actualizado y esta disponible aún para conectar.

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 para 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.

mDNS nos da los mismos beneficios que una DNS tradicional, pero ha implementado una manera que no requiere de servidor. Cuando el usuario quiere conectarse a un dispositivo usando un nombre amigable, mDNS manda un mensaje preguntando el dispositivo con ese nombre para poder identificarse. El dispositivo con el nombre después manda un mensaje de regreso incluyendo su dirección IP para que todos los dispositivos en la red puedan tener la nueva información. mDNS es lo que nos permite atribuir a nuestra roboRIO como `roboRIO-TEAM-FRC.local` y tenerla conectada en una red DHCP.

Nota: Si el dispositivo utilizado para FRC no soporta mDNS, entonces se le asignará una Dirección IP en el rango 10.TE.AM.20 - 10.TE.AM.255, pero no sabrá la dirección exacta a conectar y no seremos capaces de usar un nombre amigable como antes. En este caso, el dispositivo tendrá que tener ins dirección IP estática.

mDNS - Principios

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 - Proveedores

Para usar mDNS, se necesita una implementación de mDNS para ser instalada en su PC. Aquí hay algunas implementaciones comunes de mDNS para cada plataforma:

Windows:

- **NI mDNS Responder:** Instalado con las herramientas de juego de NI FRC
- **Apple Bonjour:** Instalado con iTunes

OSX:

- **Apple Bonjour:** Instalado por defecto

Linux:

- **nss-mDNS/Avahi/Zeroconf:** Instalado y habilitada por defecto en algunas variaciones de Linux (como Ubuntu o Mint). Puede necesitar que se instale o habilitada en otras (como Arch)

mDNS - Firewalls

Nota: Dependiendo en la configuración de la PC, no se requieren cambios, esta sección está para ayudar con la solución de problemas.

Para que mDNS trabaje bien debe dejar pasar a través de la firewall. Porque el tráfico de red viene de la implementación de mDNS y no directamente de la Driver Station o IDE, dejando esas aplicaciones pasar no puede ser suficiente. Hay dos maneras posibles de resolver los problemas de firewall de mDNS:

- Añadir una excepción de aplicación/servicio para la implementación de mDNS (NI mDNS Responder es `C:\Program Files\National Instruments\Shared\mDNS Responder\nimdnsResponder.exe`)
- Añadir una excepción de puerto para tráfico de/para rangos IP UDP 5353.:
  - `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 - Ayuda de Navegador

La mayorpia de los navegadores deben de estar disponibles para usar la dirección mDNS para acceder al servidor we de la roboRIO mientras que el proveedor de mDNS esté instalado. Los navegadores incluyen Microsoft Edge, Firefox, y Google Chrome.

41.1.7 USB

Si usa la interface USB, no se necesita ajustes de red (necesita *Instalando FRC Game Tools* instalado para proporcionar el roboRIO USB Driver). El driver de la roboRIO va a configurar automáticamente la dirección IP del usuario (de la computadora) y la roboRIO y el software enumerado arriba deben de estar disponible para localizar y utilizar su roboRIO.

41.1.8 Ethernet/Inalámbrica

The *Programando su 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 Resumen

Las Direcciones IP son las que nos permiten comunicarnos con dispositivos en la red. Para FRC, estas direcciones estarán en el rango 10.TE.AM.xx si estamos conectados a un servidor DHCP o si están asignadas estáticamente, o en el rango de enlace local `169.254.xx.yy` si los dispositivos están ajustados a DHCP, pero no hay servidor presente. Para más información de cómo funcionan las direcciones IP, vea éste artículo de Microsoft aquí.

Si todos los dispositivos en la red soportan mDNS, entonces todos los dispositivos pueden ser ajustados a DHCP y referidos a usar su nombre amigable (ejemplo, `roboRIO-TEAM-FRC.local`). Si algún dispositivo no soporta mDNS, se necesitará ajustar a direcciones estáticas.

Si todos los dispositivos están ajustados para utilizar DHCP o asignaciones estáticas de IP (con los ajustes correctos estáticos), la comunicación debe funcionar ya sea en el pit o en la cancha sin ningún cambio necesario. Si existe alguna mezcla de dispositivos estáticos y DHCP, entonces los dispositivos estáticos se conectarán en la cancha pero no en el pit. Ésto se puede resolver ya sea ajustando todos los dispositivos a ajuste estático, o dejando los actuales y proporcionando un servidor DHCP en el pit.

41.2 Configuraciones IP

Nota: Este documento describe la configuración de IP utilizada en los eventos, tanto en los campos como en las fosas, los problemas potenciales y las configuraciones de solución.

41.2.1 Notación TE.AM IP

The notation TE.AM is used as part of IPs in numerous places in this document. This notation refers to splitting your five digit team number into digits for the IP address octets. Where AM is the last two digits of the team number, and TE is the first three digits. Leading zeros are optional. This scheme supports team numbers up to 25599.

Ejemplo: `10.TE.AM.2`

Team 1 - `10.0.1.2`

Equipo 12 - `10.0.12.2`

Equipo 122 - `10.1.22.2`

Team 1002 - `10.10.2.2`

Equipo 1212 - `10.12.12.2`

Team 1202 - `10.12.2.2`

Team 1220 - `10.12.20.2`

Equipo 3456 - `10.34.56.2`

Team 10000 - `10.100.0.2`

Team 12345 - `10.123.45.2`

41.2.2 En el campo

Esta sección describe el trabajo en red cuando se conecta a la Red de Campo para los partidos

En el campo Configuración DHCP

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.

- Radio OM5P-AN u OM5P-AC de malla abierta - Estática «10.TE.AM.1» programada por Kiosk.
- roboRIO - DHCP `10.TE.AM.2` asignado por la Radio Robot
- Driver Station - DHCP («Obtener una dirección IP automáticamente») 10.TE.AM.X asignado por campo

- Cámara IP (si se usa) - DHCP `10.TE.AM.Y` asignada por Radio Robot
- Otros dispositivos (si se usan) - DHCP `10.TE.AM.Z` asignado por Radio Robot

En la configuración estática del campo

It is also possible to configure static IPs on your devices to accommodate devices or software which do not support mDNS. When doing so you want to make sure to avoid addresses that will be in use when the robot is on the field network. These addresses are `10.TE.AM.1` for the OpenMesh radio, `10.TE.AM.4` for the field router, and anything greater than `10.TE.AM.20` which may be assigned to a device configured for DHCP or else reserved. The roboRIO network configuration can be set from the webdashboard.

- Radio OpenMesh - Estática «10.TE.AM.1» programada por Kiosk
- roboRIO - Estática `10.TE.AM.2` sería una elección razonable, máscara de subred de `255.255.255.0` (por defecto)
- 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`.
- Cámara IP (si se usa) - Estática «10.TE.AM.11» sería una elección razonable, la subred «255.255.255.0» estaría bien.
- Otros dispositivos - Estática `10.TE.AM.6-.10` o `.12-.19` (.11 si la cámara no está presente) subred `255.255.255.0`

41.2.3 En los pits

Nota: Nuevo para 2018: Ahora hay un servidor DHCP funcionando en el lado cableado de la Radio del Robot en la configuración de eventos.

En los pits configuración de DCHP

- Radio OpenMesh - Estática «10.TE.AM.1» programada por Kiosk.
- roboRIO - `10.TE.AM.2`, asignado por Radio Robot
- Driver Station - DHCP («Obtener una dirección IP automáticamente»), `10.TE.AM.X`, asignado por Robot Radio
- Cámara IP (si se usa) - DHCP, `10.TE.AM.Y`, asignada por Radio Robot
- Otros dispositivos (si se usan) - DHCP, `10.TE.AM.Z`, asignado por Radio Robot

En los pits configuración estática

It is also possible to configure static IPs on your devices to accommodate devices or software which do not support mDNS. When doing so you want to make sure to avoid addresses that will be in use when the robot is on the field network. These addresses are `10.TE.AM.1` for the OpenMesh radio and `10.TE.AM.4` for the field router.

41.3 Solución de problemas de la red en la 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 Fijar la roboRIO usando mDNS

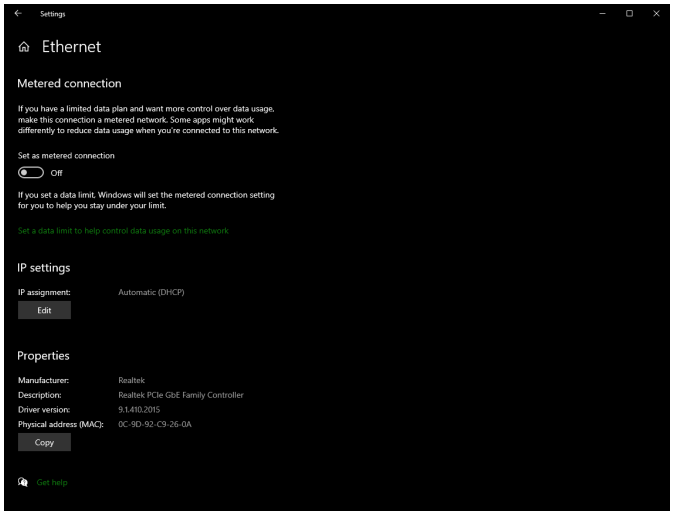
The first step to identifying roboRIO networking issues is to isolate if it is an application issue or a general network issue. To do this, click **Start -> type cmd -> press Enter** to open the command prompt. Type `ping roboRIO-#####-FRC.local` where `#####` is your team number (with no leading zeroes) and press enter. If the ping succeeds, the issue is likely with the specific application, verify your team number configuration in the application, and check your firewall configuration.

41.3.2 Fijar la dirección IP de la roboRIO

Si no hay respuesta, intente fijar «10.TE.AM.2». (*TE.AM IP Notation*) utilizando el símbolo del sistema como se ha descrito anteriormente. Si esto funciona, tiene un problema para resolver la dirección mDNS en tu PC. Las dos causas más comunes son no tener un solucionador de mDNS instalado en el sistema y un servidor DNS en la red que está tratando de resolver la dirección .local usando DNS regulares.

- Verify that you have an mDNS resolver installed on your system. On Windows, this is typically fulfilled by the NI FRC Game Tools. For more information on mDNS resolvers, see the *Network Basics article*.
- Desconecte su ordenador de cualquier otra red y asegúrese de que tiene el OM5P-AN configurado como punto de acceso, utilizando el *FRC Radio Configuration Utility*. Elimine cualquier otro router del sistema ayudará a verificar que no hay un servidor DNS que cause el problema.

41.3.3 Fije las fallas



Si el ping de la dirección IP falla, se debe tener un problema con la configuración de la conexión de la PC. La PC debe tener configurado **automático**. Para verificar eso, click en *Start -> Settings -> Network & Internet*. Dependiendo de la conexión, seleccione *Wifi* or *Ethernet*. Después haga click en la conexión deseada. Baje por la pantalla a **IP settings** y de click en *Edit* y verifique que la opción *Automatic (DHCP)* esté seleccionada.

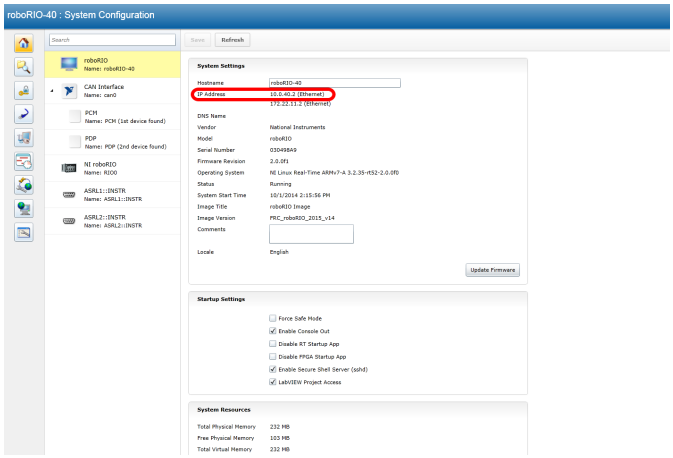
41.3.4 Solución de problemas de conexión USB

Si está intentando solucionar un problema de conexión USB, intente hacer ping a la dirección IP de roboRIO. Siempre y cuando solo haya un roboRIO conectado a la PC, debe configurarse como 172.22.11.2. Si este ping falla, asegúrese de tener el roboRIO conectado y encendido, y de haber instalado NI FRC Game Tools. Las herramientas del juego instalan los controladores roboRIO necesarios para la conexión USB.

Si este ping tiene éxito, pero el .ping local falla, es probable que el nombre de host de roboRIO esté configurado incorrectamente o que esté conectado a un servidor DNS que está intentando resolver la .dirección local.

- Verify that your roboRIO has been imaged for your team number: *roboRIO 1 roboRIO 2*. This sets the hostname used by mDNS.
- *Desactive cualquier adapatador de red*

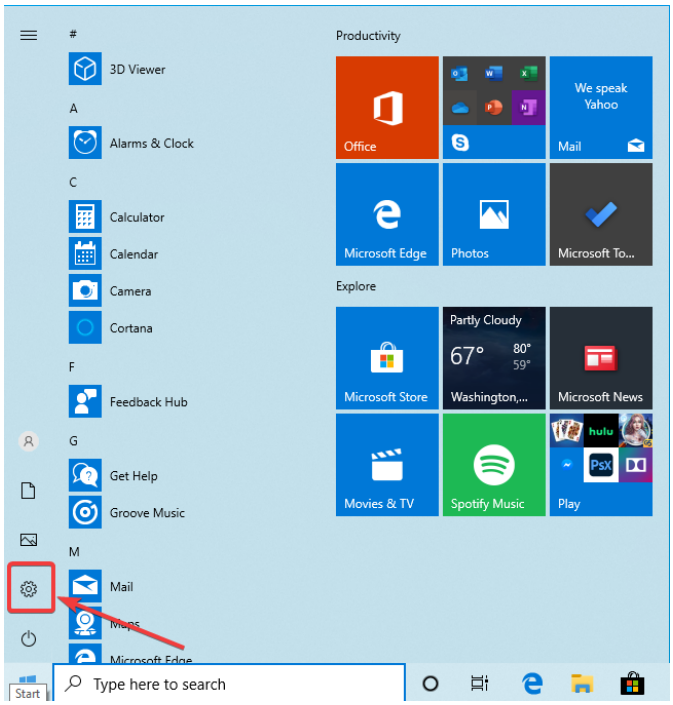
41.3.5 Conexión ethernet



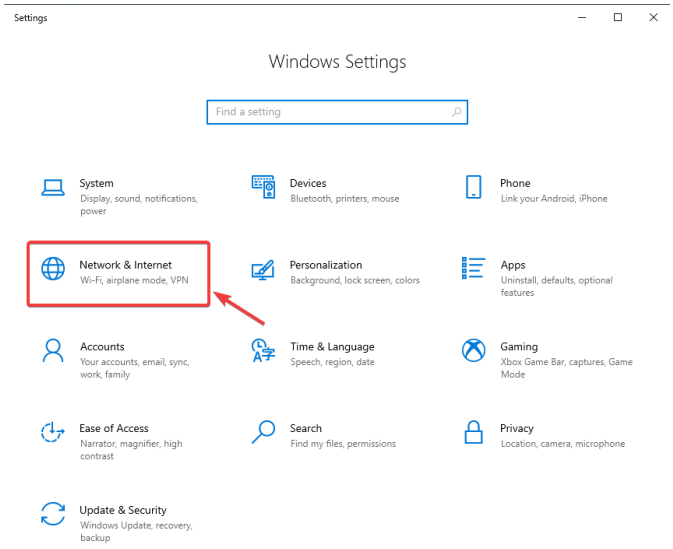
If you are troubleshooting an Ethernet connection, it may be helpful to first make sure that you can connect to the roboRIO using the USB connection. Using the USB connection, open the *roboRIO webdashboard* and verify that the roboRIO has an IP address on the ethernet interface. If you are tethering to the roboRIO directly this should be a self-assigned `169.*.*.*` address, if you are connected to the OM5P-AN radio, it should be an address of the form `10.TE.AM.XX` where TEAM is your five digit FRC team number(*TE.AM IP Notation*). If the only IP address here is the USB address, verify the physical roboRIO ethernet connection.

41.3.6 Desactivar los adaptadores de red

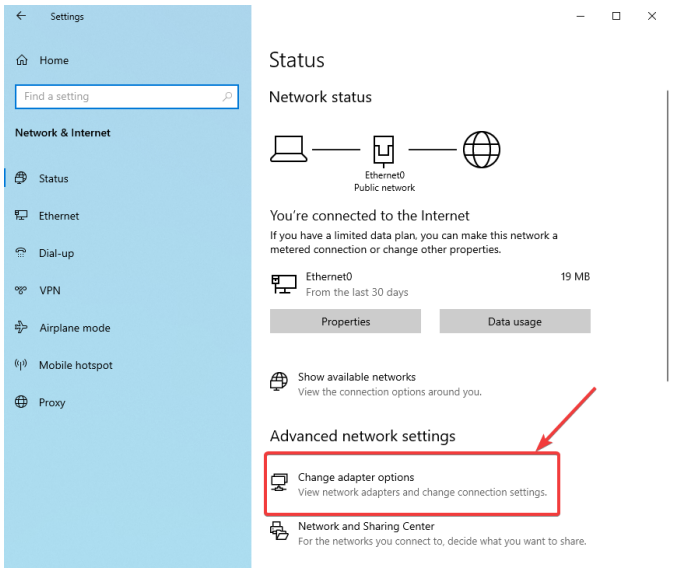
No siempre es lo mismo que apagar los adaptadores con un botón físico o poner el PC en modo avión. Los siguientes pasos proporcionan más detalles sobre cómo desactivar los adaptadores.



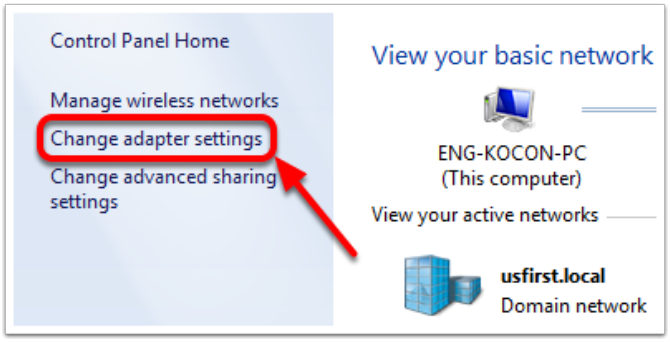
Abra la aplicación Configuración haciendo clic en el icono de configuración.



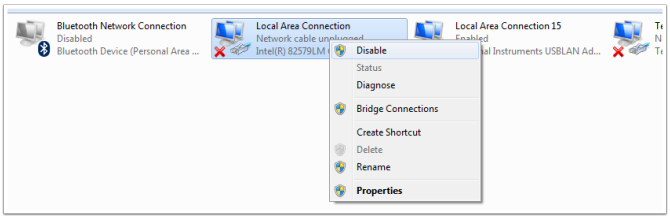
Elija la categoria *Network & Internet*



Haga click en *Change adapter options*.



En el panel izquierdo, haga click en *Change Adapter Settings*.



Para cada adaptador que no sea el que está conectado a la radio, haga clic con el botón derecho del ratón en el adaptador y seleccione *Disable* from the menu.

41.3.7 Proxies

- Proxies. Tener un proxy habilitado puede causar problemas con la red roboRIO.

41.4 Configuración de Firewall de Windows

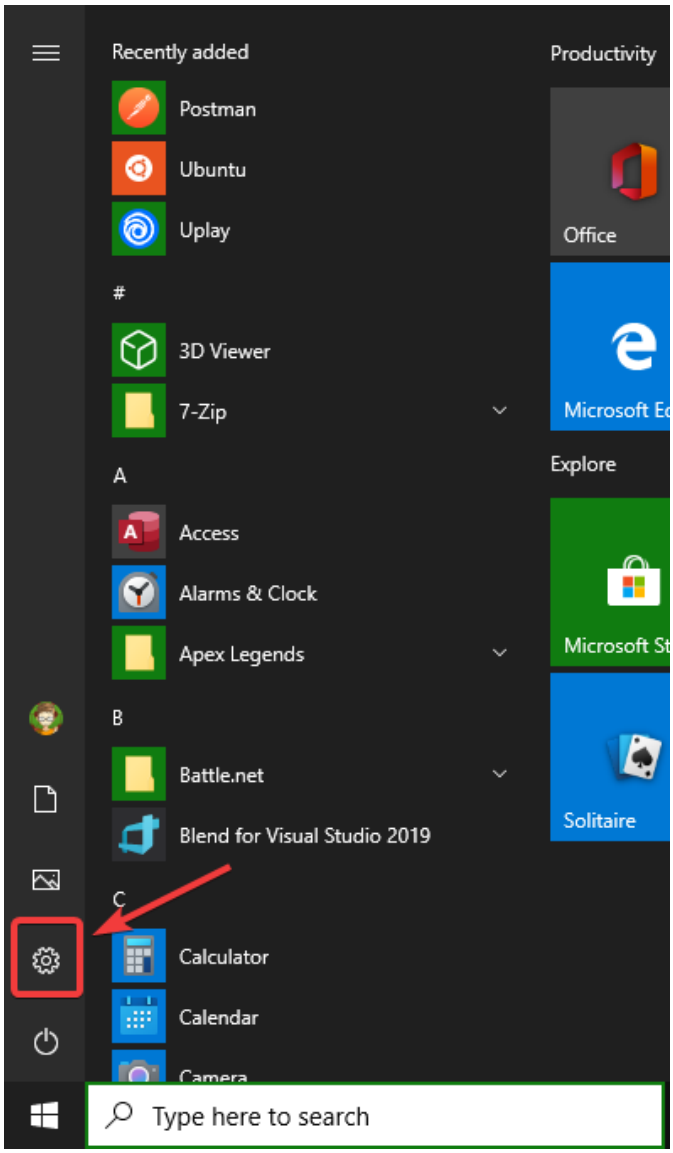
Muchas de las herramientas de programación utilizadas en FRC® necesitan acceso a la red por varias razones. Dependiendo de la configuración exacta, el Firewall de Windows puede interferir potencialmente con este acceso para uno o más de estos programas.

41.4.1 Deshabilitar el Firewall de Windows

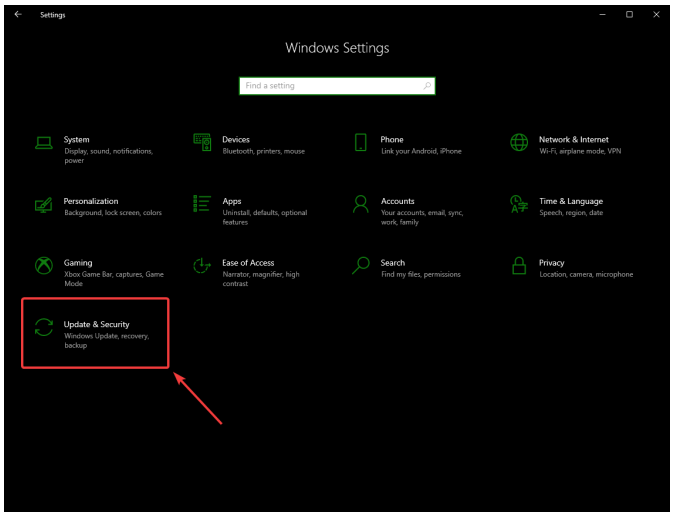
Importante: La desactivación de su firewall requiere privilegios de administrador en la PC. Además, tenga en cuenta que no se recomienda deshabilitar el firewall para computadoras que se conectan a Internet.

La solución más sencilla es desactivar el Firewall de Windows. Los equipos deben tener en cuenta que esto hace que la PC sea potencialmente más vulnerable a los ataques de malware si se conecta a Internet.

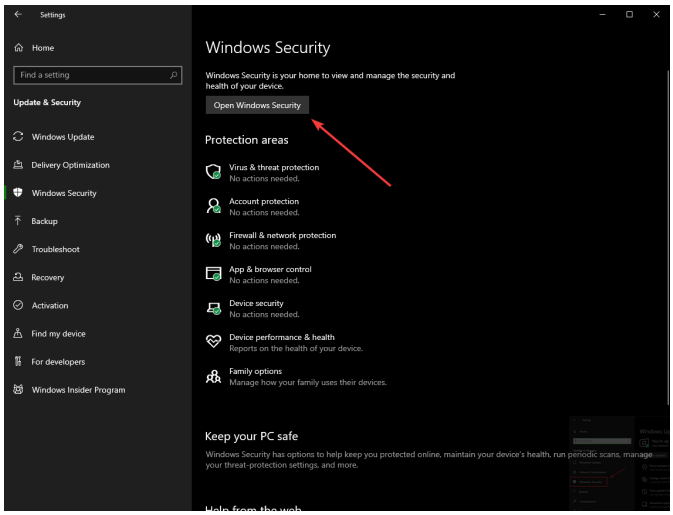
Haga clic en *Start -> Settings*



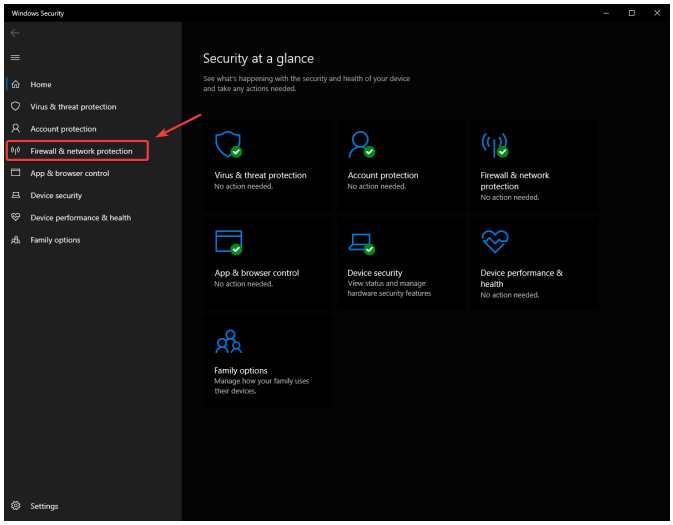
Haga clic en: guilabel:*Update & Security*



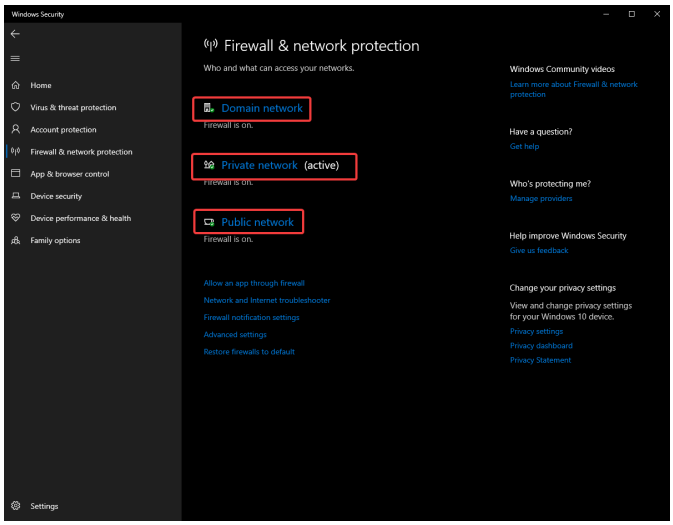
En el panel derecho, seleccione *Abrir seguridad de Windows*.



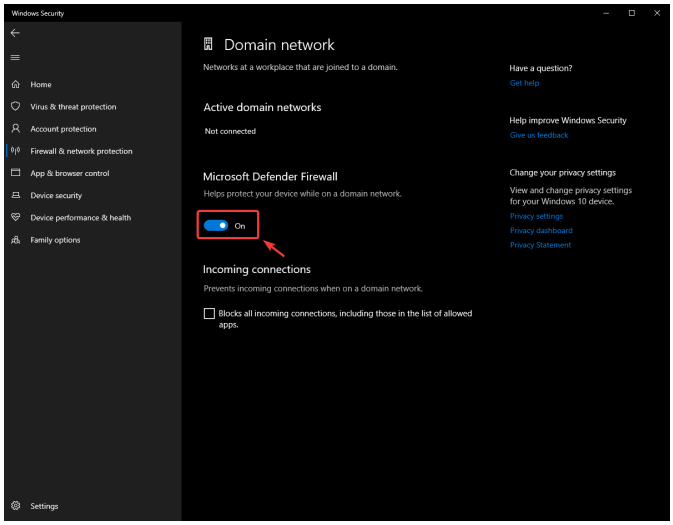
En el panel izquierdo, seleccione *Firewall y network protection*



Haga clic en **cada una** de las opciones resaltadas



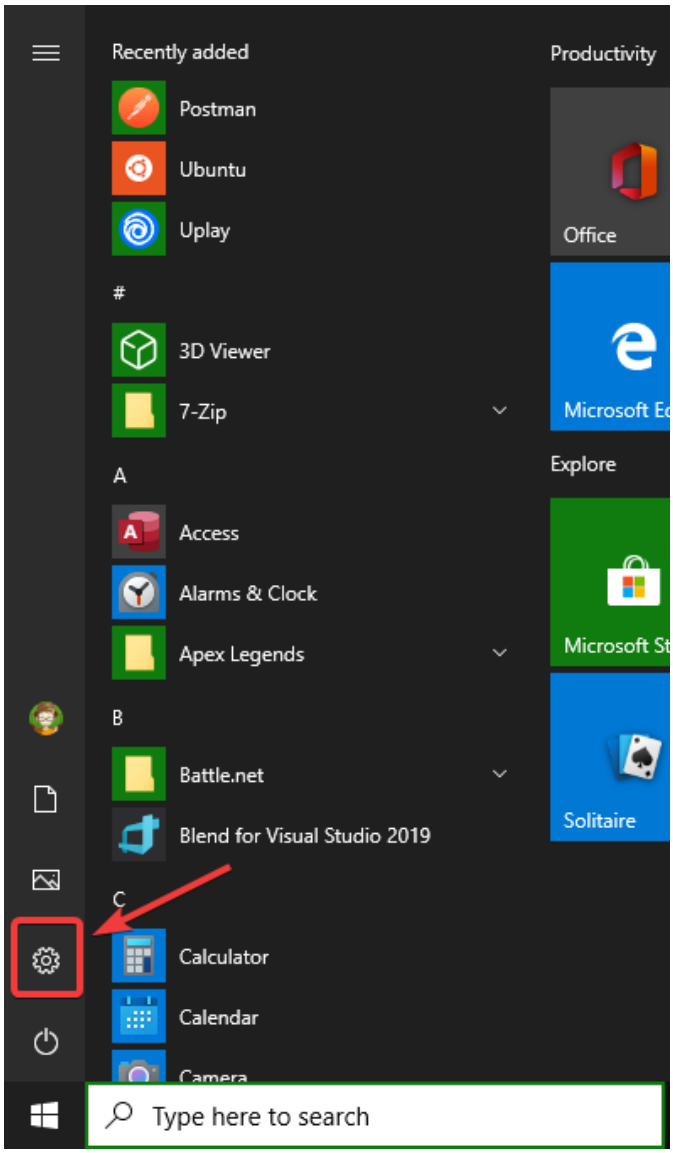
Luego, haga clic en el botón **On** para apagarlo.



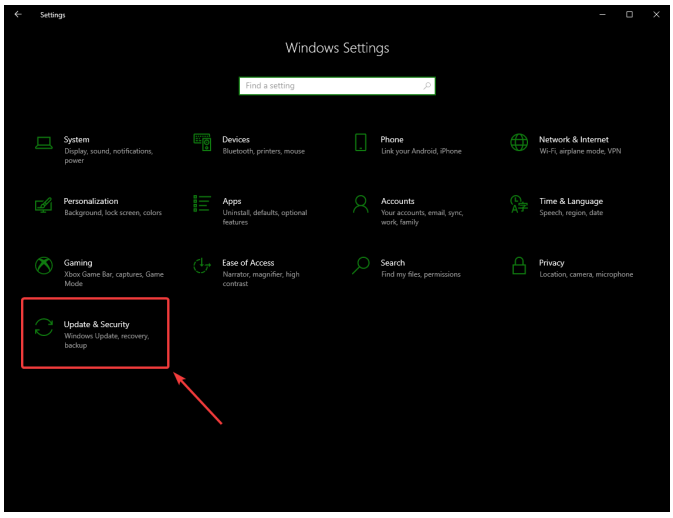
41.4.2 Aplicaciones de la lista blanca

Alternativamente, puede agregar excepciones al Firewall para cualquier programa de FRC con el que tenga problemas.

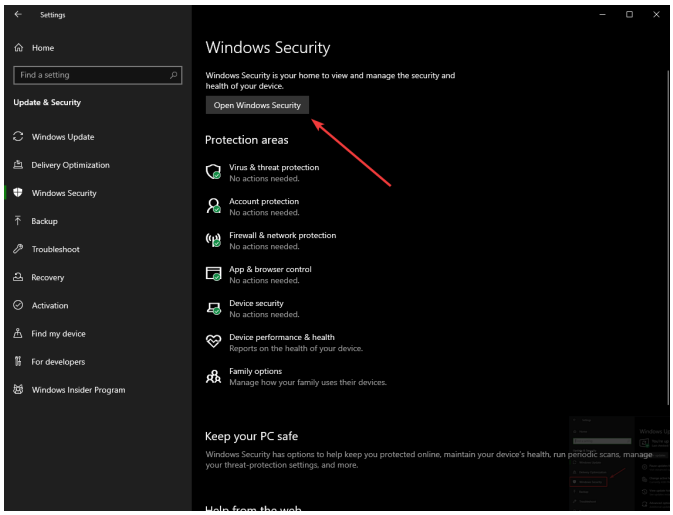
Haga clic en *Start -> Settings*



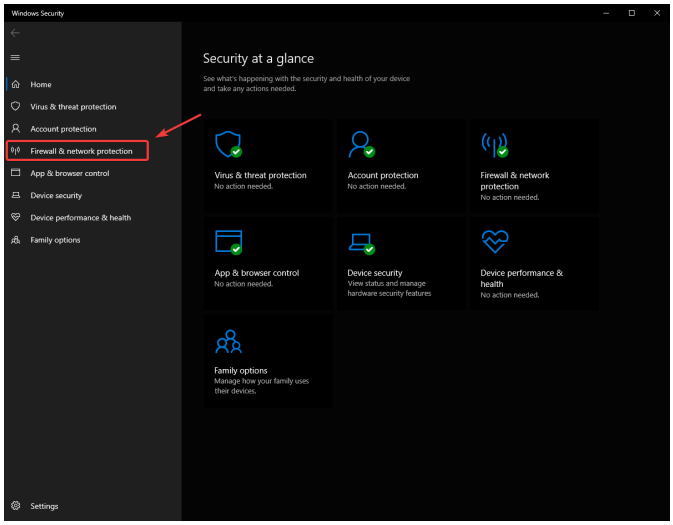
Haga clic en: guilabel:*Update & Security*



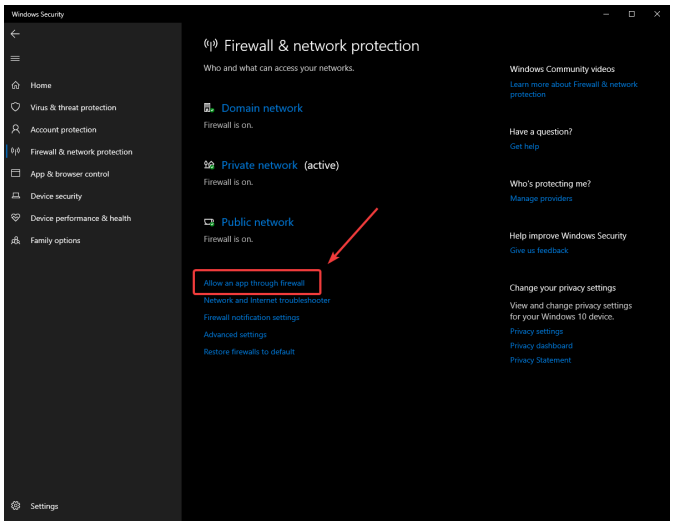
En el panel derecho, seleccione *Abrir seguridad de Windows*.



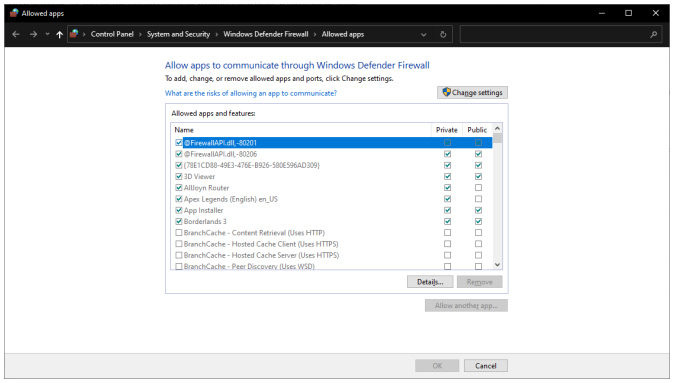
En el panel izquierdo, seleccione *Firewall y network protection*



En la parte inferior de la ventana, seleccione *Allow an app through firewall*



Para cada programa de FRC con el que tenga un problema, asegúrese de que aparezca en la lista y de que tenga una marca en cada una de las 3 columnas. Si necesita cambiar una configuración, debe hacer clic en el botón *Change settings* en la parte superior derecha antes de cambiar la configuración. Si el programa no está en la lista, haga clic en el botón *Allow another program...* y busque la ubicación del programa para agregarlo.



41.5 Medición del uso de banda ancha

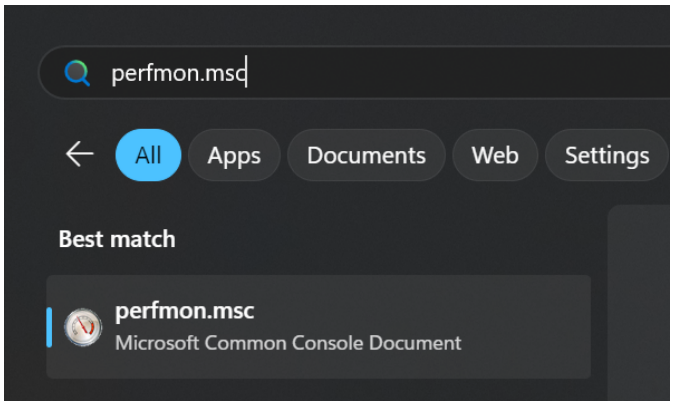
On the FRC® Field each team is allocated limited network bandwidth (see R704 in the 2024 manual). Some teams may wish to measure their overall bandwidth consumption. This document details how to make that measurement.

Nota: Teams can simulate the bandwidth throttling at home using the FRC Bridge Configuration Utility with the bandwidth checkbox checked.

41.5.1 Measuring Bandwidth Using the Performance Monitor (Win 7/10)

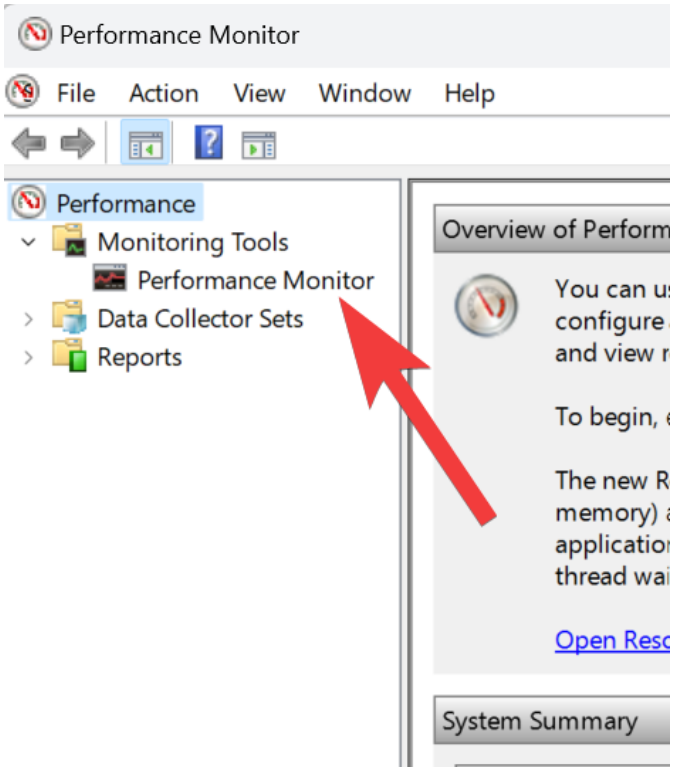
Windows contains a built-in tool called the Performance Monitor that can be used to monitor the bandwidth usage over a network interface.

Lanzamiento del Monitor de Rendimiento



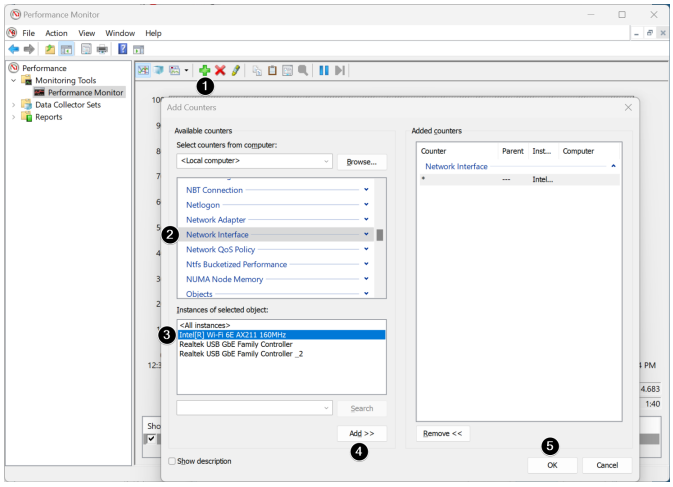
Open the Start menu and in the search box, type `perfmon.msc` and press Enter.

Abrir el monitor en tiempo real



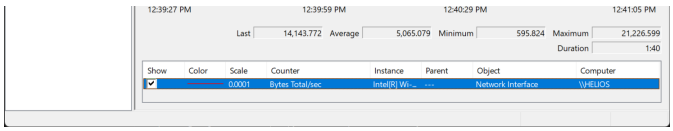
In the left pane, click *Performance Monitor* to display the real-time monitor.

Agregar contador de red



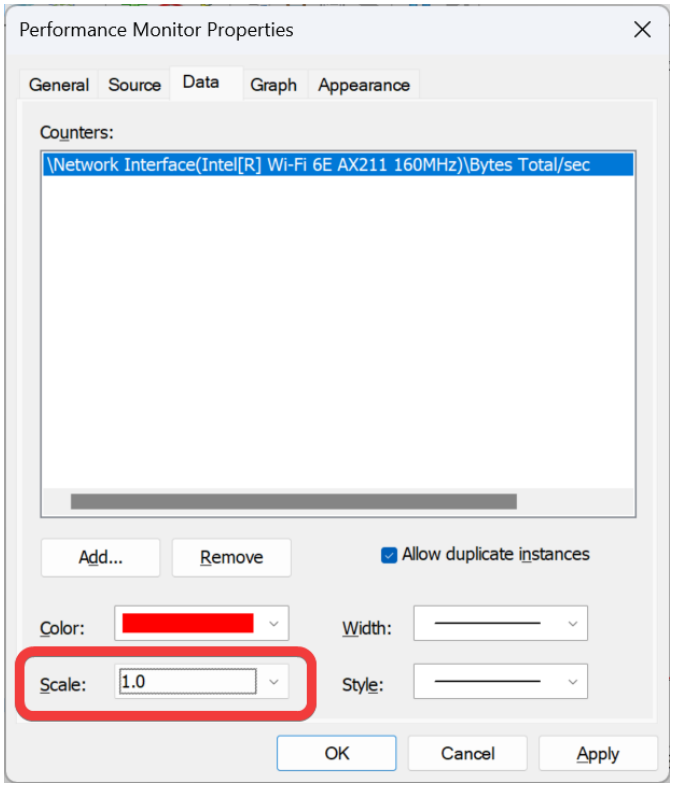
1. Haga clic en el plus verde cerca de la parte superior de la pantalla para añadir un contador
2. En el panel superior izquierdo, localice y haga clic en «Interfaz de red» para seleccionarlo.
3. En el panel inferior izquierdo, localice la interfaz de red deseada (o utilice Todas las instancias para supervisar todas las interfaces)
4. Click *Add >>* to add the counter to the right pane.
5. Click *OK* to add the counters to the graph.

Eliminar los contadores adicionales



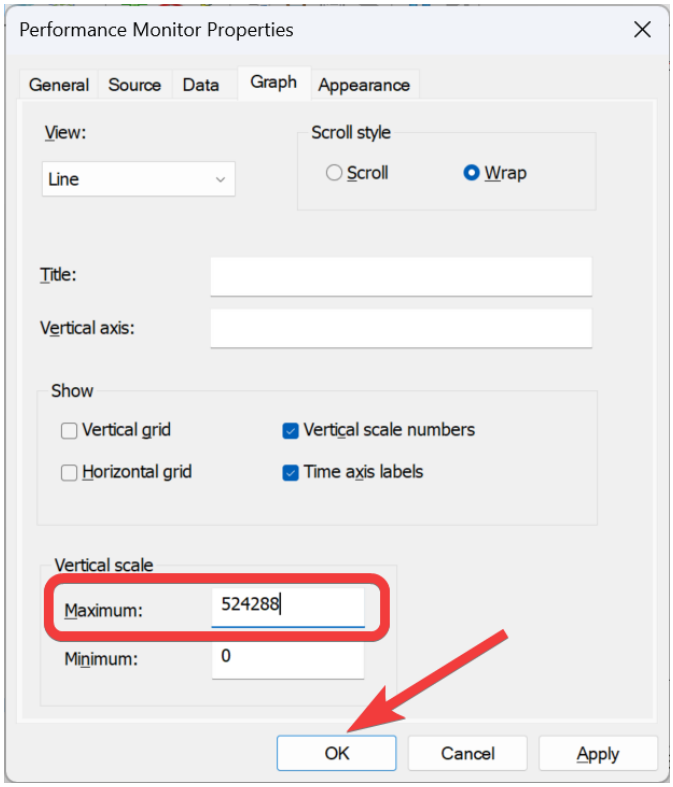
In the bottom pane, select each counter other than `Bytes Total/sec` and press the `Delete` key. The `Bytes Total/sec` entry should be the only entry remaining in the pane.

Configurar las propiedades de los datos



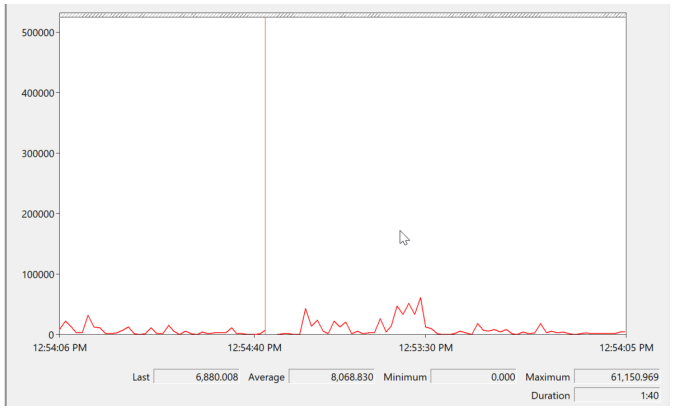
Press `Ctrl+Q` to bring up the Properties window. Click on the dropdown next to `Scale` and select `1.0`. Then click on the *Graph* tab.

Configurar las propiedades de los gráficos



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.

Ver el uso del ancho de banda

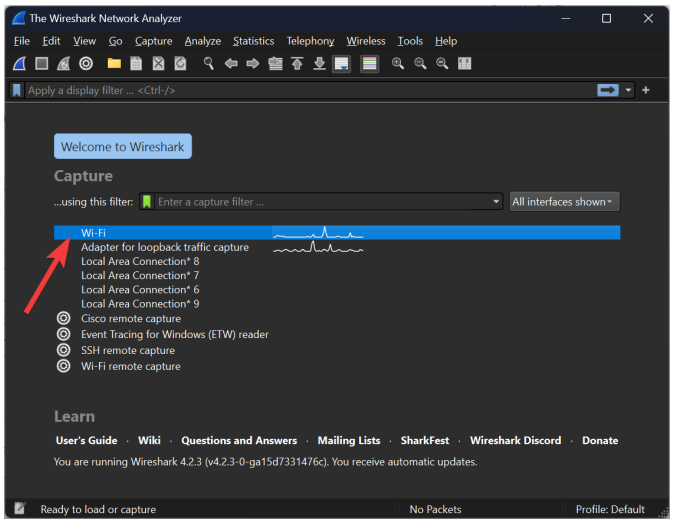


You may now connect to your robot as normal over the selected interface (if you haven't done so already). The graph will show the total bandwidth usage of the connection, with the bandwidth cap at the top of the graph. The Last, Average, Min and Max values are also displayed at the bottom of the graph. 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 Medición del uso de banda ancha usando Wireshark

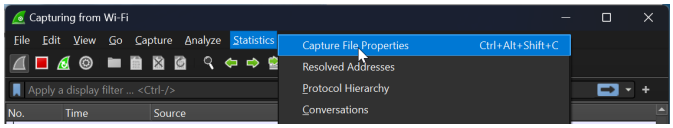
If you can not use performance monitor, you will need to install a 3rd party program to monitor bandwidth usage. One program that can be used for this purpose is Wireshark. Download and install the latest version of Wireshark for your version of Windows from this page. After installation is complete, locate and open Wireshark. Connect your computer to your robot, open the Driver Station and any Dashboard or custom programs you may be using.

Seleccione la interfaz e inicie la captura



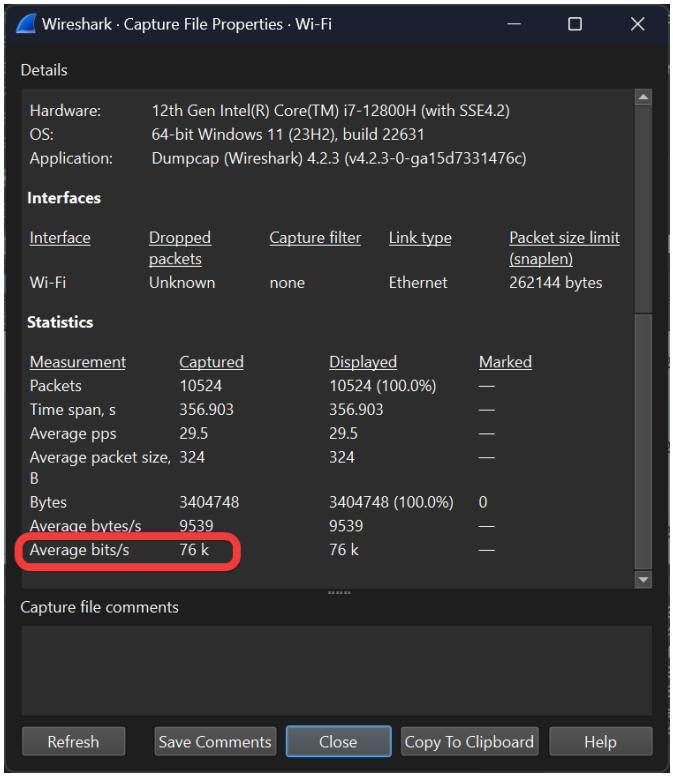
In the Wireshark program on the left side, double click the interface you are using to connect to the robot.

Open Capture File Properties



Let the capture run for at least 1 minute, then click *Statistics* then *Capture File Properties*.

Ver el uso de la banda ancha



Average bandwidth usage, in bits/second is displayed near the bottom of the window.

41.6 Modificación de radio OM5P-AC

El caso de uso previsto para la radio OM5P-AC no lo somete a los mismos golpes y fuerzas que ve en el entorno FRC®. Si la radio se somete a una presión significativa en la parte inferior de la carcasa, es posible provocar un reinicio de la radio poniendo en cortocircuito un blindaje metálico en la parte inferior de la radio con algunos cables metálicos expuestos en la parte inferior de la placa. Este artículo detalla una modificación a la radio para evitar este escenario.

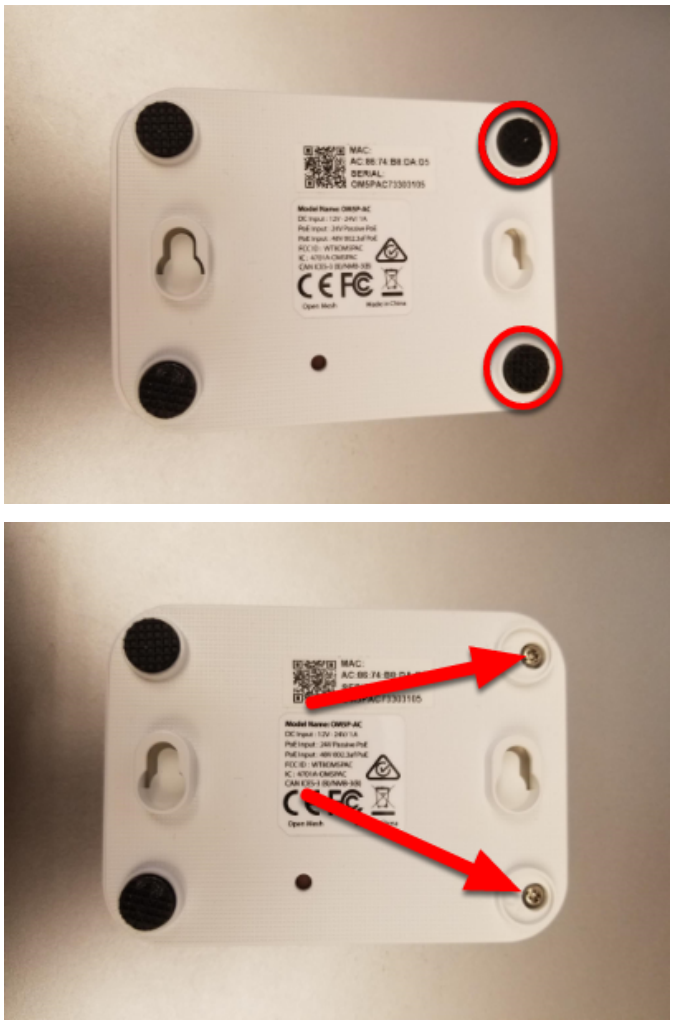
Advertencia: Se necesita una presión significativa aplicada a la parte inferior de la carcasa para provocar un reinicio de esta manera. La mayoría de los problemas de reinicio de la radio FRC se pueden rastrear hasta la ruta de alimentación de alguna forma. Recomendamos mitigar este riesgo mediante el montaje estratégico de la radio en lugar de abrir y modificar la radio (y arriesgarse a dañar los delicados componentes internos):

- Evite utilizar las funciones de «pestaña de montaje» en la parte inferior de la radio.
- Es posible que desee montar la radio para permitir cierta absorción de impactos. Un poco puede ser muy útil, montar la radio usando un gancho o un velcro o en una superficie de robot con una pequeña cantidad de flex (plástico o chapa metálica, etc.) puede reducir significativamente las fuerzas experimentadas por la radio.

41.6.1 Abrir la radio

Nota: El OpenMesh OM5P-AC no está diseñado para ser un dispositivo reparable por el usuario. Los usuarios realizan esta modificación bajo su propio riesgo. Asegúrese de trabajar lenta y cuidadosamente para evitar dañar los componentes internos, como los cables de la antena de radio.

Tornillos de caja



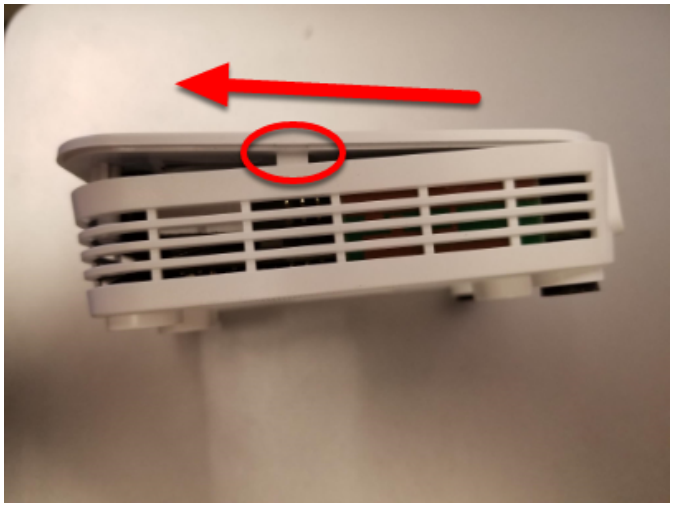
Ubique las dos patas de goma en la parte frontal de la radio y luego sáquelas de la radio con las uñas, un destornillador plano pequeño, etc. Con un destornillador Phillips pequeño, retire los dos tornillos debajo de las patas.

Pestillos laterales

Hay un pequeño pestillo en la tapa de la radio cerca del medio de cada borde largo (puede ver estos pestillos con mayor claridad en la siguiente imagen). Con una uña o una herramienta muy delgada, deslice a lo largo del espacio entre la tapa y la caja de adelante hacia atrás hacia el centro de la radio. Debería escuchar un pequeño estallido cuando se acerque a la mitad de la radio. Repita en el otro lado (nota: no es difícil volver a enganchar accidentalmente el primer lado mientras hace esto, asegúrese de que ambos lados estén desbloqueados antes de continuar). La tapa de la radio ahora debe estar ligeramente abierta en la parte frontal como se muestra en la imagen de arriba.

Quitar la tapa

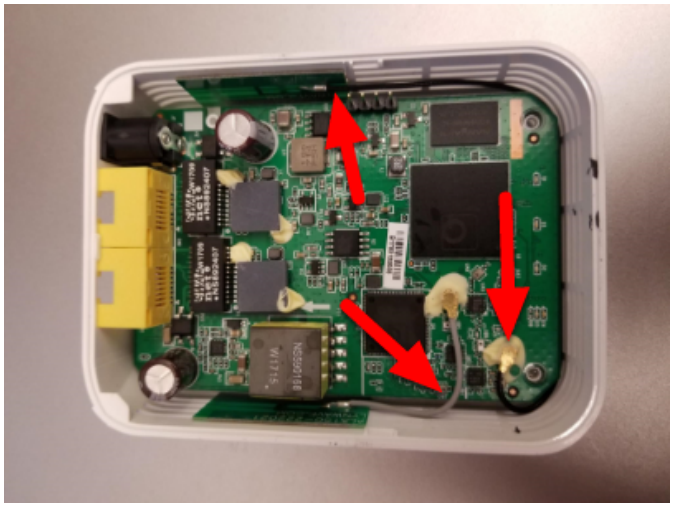
Advertencia: La placa puede adherirse a la tapa al retirarla debido a las almohadillas del disipador de calor. Mire a través de las rejillas de ventilación de la radio mientras retira la tapa para ver si la placa viene con ella, si es así, es posible que deba insertar una pequeña herramienta para sujetar la placa hacia abajo para separarla de la tapa. Recomendamos un destornillador pequeño o una herramienta similar que se ajuste a través de las rejillas de ventilación, aplicado a través de la esquina frontal del lado del gato de barril, justo encima del orificio del tornillo. Puede desplazarse hacia abajo hasta la imagen con la tapa retirada para ver cómo se ve el tablero en esta área.



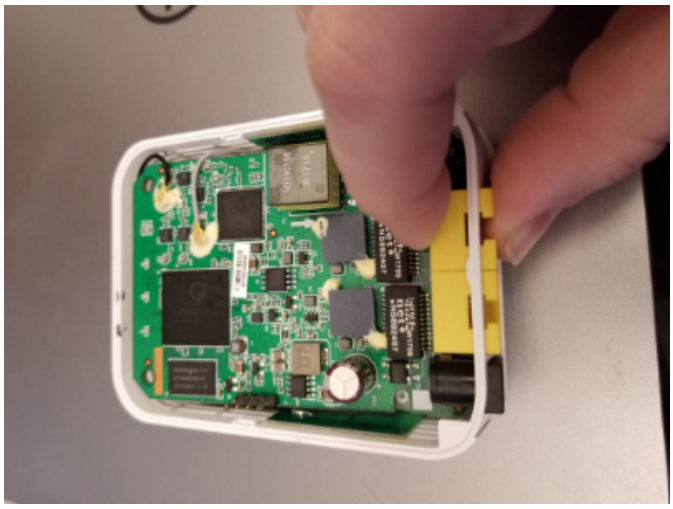
Para comenzar a quitar la tapa, deslícela hacia adelante (levantándola ligeramente) hasta que los soportes de los tornillos golpeen el frente de la caja (es posible que deba aplicar presión en las áreas del pestillo mientras hace esto.

A continuación, comience a girar la tapa ligeramente alejándola del lado del gato de barril, como se muestra, mientras continúa levantando. Esto desenganchará la tapa del pequeño triángulo visible en la esquina superior derecha. Continúe girando ligeramente en esta dirección mientras empuja la esquina superior izquierda hacia el barrel jack(no intente levantar más en este paso) para desenganchar una característica similar en la esquina superior izquierda. Luego levante la tapa completamente alejándola del cuerpo.

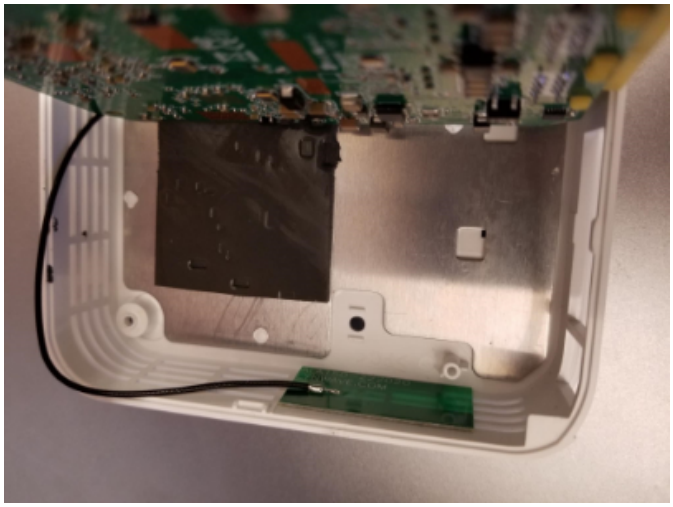
Quitar el tablero



Advertencia: Tenga en cuenta los cables de la antena que se muestran en la imagen de arriba. Estos cables, y sus conectores, son frágiles, tenga cuidado de no dañarlos mientras realiza los siguientes pasos.



Para quitar la placa, recomendamos sujetar uno o ambos puertos de red con los dedos (como se muestra) y empujar hacia adentro (hacia la parte frontal de la radio) y hacia arriba hasta que los puertos de red y el barrel jack estén libres de la carcasa.



Incline la placa hacia arriba (hacia el cable de antena gris corto) para exponer el blindaje de metal que se encuentra debajo.

Nota: Al realizar este paso, es posible que observe que hay un pequeño botón de reinicio en la parte inferior de la placa que es más grande que el orificio de la caja. Tenga en cuenta que presionar el botón de reinicio con el firmware FRC instalado no tiene ningún efecto y que perforar la carcasa del radio no es una modificación permitida.

41.6.2 Aplicar cinta

Aplique un trozo de cinta aislante al blindaje de metal en el área justo dentro de las aberturas del conector de barril / puerto de red. Esto evitará que los cables expuestos en la parte inferior de la placa entren en cortocircuito en esta placa.

41.6.3 Volver a montar la radio

Vuelva a ensamblar la radio invirtiendo las instrucciones para abrirla:

- Vuelva a colocar la tabla, asegurándose de que se alinee con los orificios de los tornillos cerca del frente y se asiente de forma segura
- Deslice la tapa sobre la función de retención trasera izquierda moviéndola hacia adentro de derecha a izquierda. Cuida el condensador en esta área
- Gire la tapa, presione hacia abajo y deslice la pieza de retención trasera derecha hacia dentro
- Presione firmemente hacia abajo en la parte delantera / central de la tapa para asentar los pestillos
- Reemplace 2 tornillos en las patas delanteras
- Reemplazar pies delanteros

42 Utilidades de red

42.1 Redirecccionamiento de puertos

Esta clase proporciona una manera fácil de redireccionar puertos locales a otro host / puerto. Esto es útil para proporcionar una manera de acceder a dispositivos conectados a Ethernet desde una computadora conectada al puerto USB roboRIO. Esta clase actúa como un redireccionador de puertos TCP sin procesar, esto significa que puede redireccionar conexiones como SSH.

42.1.1 Redireccionamiento de un puerto remoto

Often teams may wish to connect directly to the roboRIO for controlling their robot. The PortForwarding class (Java, C++) can be used to forward the Raspberry Pi connection for usage during these times. The PortForwarding class establishes a bridge between the remote and the client. To forward a port in Java, simply do `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)
```

Importante: **No se puede** usar un puerto menor que 1024 como su puerto de reenviado local. También es importante tener en cuenta que **no se puede** usar URL completas (`http://wpilibpi.local`) y solo debe usar direcciones IP o nombres DNS.

42.1.2 Removiendo un redireccionamiento de puertos

Para detener el reenvío en un puerto específico, simplemente llame a `remove(int port)`, siendo el puerto el número de puerto. Si llama a `remove()` en un puerto que no se está reenviando, no sucederá nada.

JAVA

```
@Override
public void robotInit() {
   PortForwarder.remove(8888);
}
```

C++

```
void Robot::RobotInit {
   wpi::PortForwarder::GetInstance().Remove(8888);
}
```

PYTHON

```
wpinet.PortForwarder.getInstance().remove(8888)
```

43 Para Contribuir a frc-docs

43.1 Guías de Contribución

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.

Importante: *FIRST*® conserva todos los derechos sobre la documentación y las imágenes proporcionadas. El crédito por artículos / actualizaciones estará en GitHub commit history.

43.1.1 Misión

La misión de WPILib es hace posible a los equipos de *FIRST* Robotics concentrarse en el desarrollo de software específico para el juego en vez de centrar su atención en detalles de hardware. Trabajamos para ayudar a los equipos con conocimientos de programación limitados y/o experiencia de sus mentores a ser tan exitosos como sea posible, sin obstaculizar las habilidades de los equipos con capacidades mayores de programación. Tenemos soporte directo en nuestras librerías de los componentes del sistema de control del Kit of Parts. También nos esforzamos al máximo para mantener una paridad entre las características principales de cada lenguaje (Java, C++ y LabVIEW), para que los equipos no tengan desventaja alguna al escoger su lenguaje de programación.

Estos documentos sirven para proporcionar un campo de aprendizaje para todos los equipos de *FIRST* Robotics Competition. Las contribuciones al proyecto deben seguir estos principios básicos.

- Documentación dirigida por la comunidad. Las fuentes de la documentación se alojan públicamente y la comunidad puede hacer contribuciones.
- Documentación estructurada, con buen formato, y limpia. La documentación debe ser limpia y fácil de leer, tanto desde el punto de vista de la fuente como de la versión.
- Relevante. La documentación debe centrarse en el *FIRST* Robotics Competition.

Consulte *el Guía de Estilo* para obtener información sobre cómo diseñar su documentación.

43.1.2 Proceso de liberación

frc-docs utiliza un proceso de liberación especial para manejar el sitio principal `/stable/` y el sitio de desarrollo `/latest/`. Este flujo se detalla a continuación.

Durante la temporada:

- Commit hecho en la rama `main`
  - Actualiza `/stable/` y `/latest/` en el sitio web

Fin de la temporada:

- El repositorio está etiquetado con el año, para propósitos de archivo.

Off-Season:

- La rama `stable` se bloquea con el ultimo commit hecho en temporada
- Commit hecho en la rama `main`
  - Sólo se actualiza `/latest/` en el sitio de documentación

43.1.3 Creando un PR

Los PRs deben hacerse en el repositorio frc-docs en GitHub. Deben apuntar a la rama `main` y *no* a `stable`.

43.1.4 Crear Contenido Nuevo

¡Gracias por contribuir al proyecto frc-docs ! ¡Hay algunas cosas que debería saber antes de iniciar!

¿Dónde colocar los artículos?

La ubicación de los nuevos artículos puede ser un tema bastante obstinado. Los artículos independientes que caen bien en una categoría de tema ya deben colocarse en la categoría de tema mencionada (la documentación sobre algo sobre simulación debe colocarse en la sección de simulación). Sin embargo, las cosas pueden complicarse bastante cuando un artículo combina o hace referencia a dos secciones existentes separadas. En esta situación, recomendamos al autor que abra un número en el repositorio para iniciar la discusión antes de abrir el PR.

Nota: Todos los artículos nuevos se someterán a un proceso de revisión antes de fusionarse en el repositorio. Este proceso de revisión será realizado por miembros del equipo de WPILib. Los nuevos artículos deben ser sobre software y hardware oficial compatible con *FIRST*. La documentación sobre bibliotecas o sensores no oficiales *no* será aceptada. Este proceso puede tomar tiempo de revisar, por favor sea paciente.

¿Dónde colocar las secciones?

Las secciones son bastante complicadas, ya que contienen una gran cantidad de contenido. Aconsejamos al autor que abra un issue para entablar una discusión antes de abrir un PR.

Vincular Otros Artículos

En el caso de que el artículo haga referencia a contenido que se describe en otro artículo, el autor debe hacer todo lo posible por vincular a ese artículo en la primera referencia.

Imagina que tenemos el siguiente contenido en un tutorial de transmisión:

```
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.
```

Observe cómo solo está vinculada la primera instancia de Simulación. Esta es la estructura que debe seguir el autor. Hay ocasiones en las que un artículo vinculado tiene diferentes temas de contenido. Si hace referencia a los diferentes tipos de contenido en el artículo, debe vincular a cada nueva referencia una vez (excepto en situaciones en las que el autor haya considerado apropiado lo contrario).

43.2 Guía de Estilo

Este documento contiene las diversas pautas específicas de RST/Sphinx para el proyecto frc-docs. Para conocer las pautas relacionadas con los diversos proyectos de código WPILib, consulte the WPILib GitHub

43.2.1 Nombres de Archivo

Use solo caracteres alfanuméricos en minúsculas y el guion (signo «-«).

Para documentos que tendrán un nombre de software/hardware idéntico, agregue «hardware» o «software» al final del nombre del documento. p.ej. `ultrasonics-hardware.rst`

Añada la extensión `.rst` como sufijo a los nombres de archivo.

Nota: If you are having issues editing files with the `.rst` extension, the recommended text editor is VS Code with the reStructuredText extension.

43.2.2 Texto

All text content should be on the same line. If you need readability, use the word-wrap function of your editor.

Use el siguiente caso para estos términos:

- roboRIO (no RoboRIO, roboRio, o RoboRio)
- LabVIEW (no labview o LabView)
- Visual Studio Code (VS Code) (no vscode, VScode, vs code, etc)
- macOS (no Mac OS, Mac OSX, Mac OS X, Mac, Mac OS, etc.)
- GitHub (no github, Github, etc.)
- PowerShell (no PowerShell, Powershell, etc.)
- Linux (no linux)
- Java (no Java)

Utilice el conjunto de caracteres ASCII para el texto en inglés. Para caracteres especiales (p.ej., Símbolos griegos) use los conjuntos de entidades de caracteres estándar.

Use `.. math::` para ecuaciones independientes y `:math:` para ecuaciones en línea. Puede encontrar una útil hoja de referencia de ecuaciones de LaTeX aquí.

Use literales para nombres de archivos, funciones y nombres de variables.

El uso de las marcas registradas *FIRST*® y FRC® debe seguir la Política de esta página. Específicamente, cuando sea posible (es decir, no anidado dentro de otro marcado o en el título de un documento), el primer uso de las marcas registradas debe tener el ® símbolo y todas las instancias de *FIRST* deben estar en cursiva. El ® símbolo se puede agregar usando `.. include:: <isonum.txt>` al principio del documento y luego usando `*FIRST* \ |reg|` o `FRC\ |reg|`.

Commonly used terms should be added to the *Glosario de FRC*. You can reference items in the glossary by using ``:term:`deprecated```.

43.2.3 Espacios en Blanco

Sangría

La sangría debe *siempre* coincidir con el nivel anterior de sangría *a menos que* esté creando un nuevo bloque de contenido.

La sangría de las directivas de contenido como nueva línea `.. toctree::` debe ser *3* espacios.

Líneas En Blanco

There should be 1 blank line separating basic text blocks and section titles. There *should* be 1 blank line separating text blocks *and* content directives.

Espacio blanco interior

Use solo uno espacio entre frases.

43.2.4 Títulos

Los títulos deben estar en la siguiente estructura. Los subrayados de títulos deben coincidir con el mismo número de caracteres que hay en el título.

1. `` = `` para títulos de documentos. *No* use esto más de *una vez* por artículo.
2. `` -`` para secciones de documentos
3. ^ para subsecciones de documentos
4. ~ para sub-subsecciones de documentos
5. Si necesita usar niveles más inferiores para la estructura, está haciéndolo mal.

Escriba las primeras letras de cada palabra en mayúsculas para los títulos.

43.2.5 Listas

Las listas deben tener una nueva línea entre cada nivel de sangría. El mayor nivel de sangría debe tener una sangría de «0», y las subsiguientes sublistas deben tener una sangría que comience en el primer carácter de la sangría anterior.

```
- Block one
- Block two
- Block three

  - Sub 1
  - Sub 2

- Block four
```

43.2.6 Bloques de código

Todos los bloques de código deben tener un lenguaje especificado.

1. Excepción: Contenido donde el formato debe ser preservado y no tiene lenguaje. En lugar de eso, usa «texto».

Siga el Guía de estilo de WPILib para el código de ejemplo de C++ y Java. Por ejemplo, usa dos espacios para la indentación en C++ y Java.

43.2.7 RLI (Remote Literal Include)

When possible, instead of using code blocks, an RLI should be used. This pulls code lines directly from GitHub, most commonly using the example programs. This automatically keeps the code up to date with any changes that are made. The format of an RLI is:

```
.. 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
```

Make sure to link to the raw version of the file on GitHub. There is a handy Raw button in the top right corner of the page.

43.2.8 Tabs

To create code tabs in an article, you can use the `.. tab-set-code::` directive. You can use `code-block` and `rli` directives inside. The format is:

```
.. 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);
```

If you need to use more than one tab per language, multiple RLIs per language, or text tabs, you can use the `.. tab-set::` and `.. tab-item::` directive. The format is:

```
.. tab-set::

   .. tab-item:: Title
      :sync: sync-id

         Content
```

This example uses the sync argument to allow all of the tabs with the same key to be synced together. This means that when you click on a tab, all of the tabs with the same key will open.

If you have a mix of `tab-set` and `tab-set-code` directives on a page, you can sync them by setting the sync id on the `tab-item` directives to `tabcode-LANGUAGE`. For example, a java tab would have a sync id of `tabcode-java`.

43.2.9 Amonestaciones

Admonitions (list here) should have their text on the same line as the admonition itself. There are exceptions to this rule, however, when having multiple sections of content inside of an admonition. Generally having multiple sections of content inside of an admonition is not recommended.

Uso

```
.. warning:: This is a warning!
```

NO

```
.. warning::
   This is a warning!
```

43.2.10 Links

Internal Links

Internal Links will be auto-generated based on the ReStructuredText filename and section title.

For example, here are several ways to link to sections and documents.

Use this format to reference a document section. You must use the absolute path of the document. `:ref:`docs/software/hardware-apis/sensors/ultrasonics-software:Analog ultrasonics`` renders to *Ultrasónicos analógicos*.

Use this format to reference a section of the same document. Note the single underscore. `` `Images`_ `` renders to *Images*.

Use this format to reference the top-level of a document. You can use relative paths `:doc:`build-instructions`` renders to *Instrucciones de Construcción* Or to use absolute paths, put a forward slash at the beginning of the path `:doc:`/docs/software/hardware-apis/sensors/ultrasonics-software`` renders to *Ultrasónicos - Software*. Note that the text rendered is the main section title of the target page regardless of the target filename.

When using `:ref:` or `:doc:` you may customize the displayed text by surrounding the actual link with angle brackets `<>` and adding the custom text between the first backtick `` ` `` and the

first angle bracket <. For example `:ref:`custom text <docs/software/hardware-apis/sensors/ultrasonics-software:Analog ultrasonics>`` renders to *custom text*.

External Links

It is preferred to format external links as anonymous hyperlinks. The important thing to note is the **two** underscores appending the text. In the situation that only one underscore is used, issues may arise when compiling the document.

```
Hi there, `this is a link <https://example.com>`__ and it's pretty cool!
```

Sin embargo, en algunos casos en que el mismo enlace debe ser referenciado varias veces, se acepta la sintaxis que figura a continuación.

```
Hi there, `this is a link`_ and it's pretty cool!

..  _this is a link: https://example.com
```

43.2.11 Imágenes

Las imágenes deben ser creadas con 1 línea que separe el contenido y la directiva.

Todas las imágenes (incluyendo los vectores) deben tener un tamaño inferior a «500» kilobytes. Por favor, utilice una resolución más pequeña y algoritmos de compresión más eficientes.

```
.. image:: images/my-article/my-image.png
   :alt: Always add alt text here describing the image.
```

Archivos de imágenes

Image files should be stored in the document directory, sub-directory of `document-name/images`.

Deben seguir el esquema de nomenclatura de `descripción-corta.png`, donde el nombre de la imagen es una breve descripción de lo que muestra la imagen. Debe tener menos de 24 caracteres.

Deben tener la extensión de imagen `.png` or `.jpg`. `.gif` es inaceptable debido a problemas de almacenamiento y accesibilidad.

Nota: ¡La accesibilidad es importante! Las imágenes deben estar marcadas con una directiva `:alt:`.

```
.. image:: images/my-document/my-image.png
   :alt: An example image
```

Imágenes vectoriales

Los archivos SVG son soportados a través de la extensión Sphinx de `svg2pdfconverter`.

Simplemente úsalos como lo harías con cualquier otra imagen.

Nota: Asegúrate de que las imágenes incrustadas en el vector no inflen el vector hasta superar el límite de 500KB.

```
.. image:: images/my-document/my-image.svg
   :alt: Always add alt text here describing the image.
```

Diagramas Draw.io

Draw.io (también conocidos como diagrams.net) los diagramas son soportados a través de archivos `svg` con metadatos incrustados `.drawio`, permitiendo que el archivo `svg` actúe como un archivo fuente de los diagramas, y que se renderice como un archivo normal de gráficos vectoriales.

Simplemente úsalos como lo harías con cualquier otra imagen vectorial, o cualquier otra imagen.

```
.. image:: diagrams/my-document/diagram-1.drawio.svg
   :alt: Always add alt text here describing the image.
```

Archivos de Draw.io

Los archivos Draw.io siguen casi el mismo esquema de nombres que las imágenes normales. Para llevar un registro de los archivos que tienen los metadatos `.drawio` incrustados, agrega un `.drawio` al final del nombre del archivo, antes de la extensión, lo que significa que el nombre del archivo debe ser `document-title-1.drawio.svg` y así sucesivamente. Además, los diagramas deben ser almacenados en el directorio de documentos en una subcarpeta llamada `diagramas`.

Para los detalles de guardar un diagrama como un `.svg` con metadatos, echa un vistazo a *Draw.io Instrucciones de Guardado*.

Advertencia: Asegúrate de que no modificas ningún archivo que esté en una carpeta `diagramas`, o que termine en `.drawio.svg` en cualquier otro programa que no sea draw.io, de lo contrario podrías arriesgarte a romper los metadatos del archivo, haciéndolo ineditable.

43.2.12 Extensiones de archivo

Las extensiones de archivo deben usar el formato de código. Por ejemplo, use:

```
``.png``
```

en lugar de:

```
.png
".png"
"``.png``"
```

43.2.13 Tablas de contenido (TDC)

Cada categoría debe contener un «índice.rst». Este archivo de índice debe contener una «profundidad máxima» de «1». Las subcategorías son aceptables, con una `profundidad máxima` de 1.

The category `index.rst` file can then be added to the root index file located at `source/index.rst`.

43.2.14 Ejemplos

```
Title
=====
This is an example article

.. code-block:: java

   System.out.println("Hello World");

Section
-------
This is a section!
```

43.2.15 ¡Nota importante!

Esta lista no es exhaustiva y los administradores se reservan el derecho de hacer cambios. Los cambios se reflejarán en este documento.

43.3 Instrucciones de Construcción

Este documento contiene información sobre cómo construir las versiones de HTML, PDF, y EPUB del sitio frc-docs. frc-docs usa Sphinx como generador de documentación. Éste documento también asume que usted tiene los conocimientos básicos de Git y los comandos de consola.

43.3.1 Prerrequisitos

Asegúrese de que Git esté instalado y que el repositorio de frc-docs se clone utilizando `git clone https://github.com/wpilibsuite/frc-docs.git`.

Editores de texto / IDE

Para desarrollo, recomendamos que uses VS Code junto con la extensión reStructuredText. Sin embargo, cualquier editor de texto funcionará.

Por defecto, la extensión reStructuredText habilita el linting con todas las características de doc8 habilitadas. Como frc-docs no sigue el lint de longitud de línea, agrega lo siguiente a tu archivo `settings.json` de VS Code para deshabilitar el linting de longitud de línea.

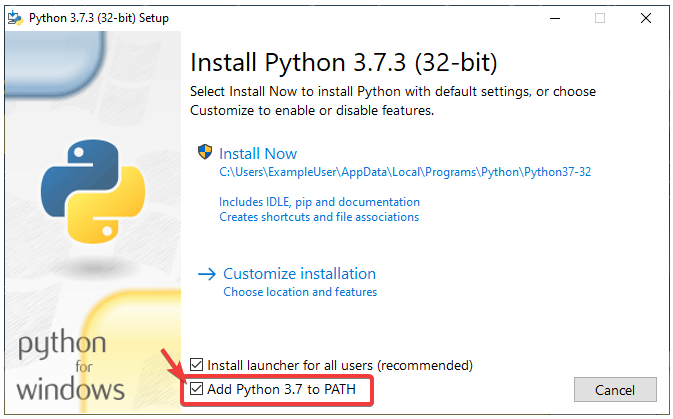
```
"restructuredtext.linter.doc8.extraArgs": [
   "--ignore D001"
]
```

Windows

Nota: MikTeX y `rsvg-convert` no son necesarios para compilaciones de HTML, sólo son necesarios para PDF en Windows.

- Python 3.9
- Perl
- *MiKTeX <https://miktex.org/download>* __ (Solo necesario para compilaciones PDF)
- rsvg-convert (Solo necesario para generar PDF)

Asegúrese de que Python esté en su PATH, marcando la casilla **Agregar Python a PATH** al instalar Python.



Una vez que Python esté instalado, abra Powershell. Luego navegue al directorio frc-docs. Ejecute el siguiente comando: `pip install -r source/requirements.txt`

Instala los paquetes faltantes de MikTex navegando al directorio frc-docs, luego ejecuta el siguiente comando desde 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 Compilando

Abra una ventana o terminal de Powershell y navegue hasta el directorio frc-docs que fue clonado.

```
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>
```

Lint Check

Nota: Lint Check no marcará los bordes en Windows debido al error de los bordes. Vea este problema para más información.

Se recomienda verificar cualquier cambio que realice con el linter. Esto **podría** fallar el buildbot si no pasa. Para verificar, ejecute `.\make lint`

Verificar el enlace

El verificador de enlace asegura que todos los enlaces/links en la documentación estén resueltos. Esto **podría** fallar el buildbot si no pasa. Para verificar, ejecute `.\make linkcheck`

Verificar el tamaño de imagen

Ejecute `.\make sizecheck` para verificar que todas las imágenes tengan menos de 500 KB. Esta verificación **fallará** CI si falla. Las exclusiones se permiten caso por caso y se agregan a la lista `IMAGE_SIZE_EXCLUSIONS` en el archivo de configuración.

Verificar redirección

Los archivos que se han movido o cambiado de nombre deben tener su nueva ubicación (o reemplazarse con 404) en el archivo `redirects.txt` en `source`.

El escritor de redireccionamiento agregará automáticamente archivos renombrados / movidos al archivo de redireccionamientos. Ejecute `.\make rediraffewritediff`.

Nota: Si un archivo se mueve y se modifica sustancialmente, el escritor de redirecciones no lo agregará al archivo `redirects.txt` y el archivo `redirects.txt` deberá actualizarse manualmente.

El comprobador de redireccionamiento se asegura de que haya redireccionamientos válidos para todos los archivos. Esto **fallará** al buildbot si no pasa. Para comprobarlo, ejecute `.\make rediraffecheckdiff` para verificar que todos los archivos estén redirigidos. Además, es posible que sea necesario ejecutar una compilación HTML para garantizar que todos los archivos se redireccionen correctamente.

Construyendo HTML

Escriba el comando `.\make html` para generar un contenido HTML. Éste contenido esta ubicado en el directorio `build/html` en la raíz del repositorio.

43.3.3 Construyendo un PDF

Advertencia: Por favor note que un PDF hecho en Windows puede resultar en imágenes distorsionadas por el contenido SVG. Esto se debe a la falta de soporte de librsvg2-bin en Windows.

Escriba el comando `.\make latexpdf` para generar el contenido PDF. El PDF está localizado en el directorio `build/latex` en la raíz del repositorio.

43.3.4 Construyendo EPUB

Escriba el comando `.\make epub` para generar un contenido EPUB. El EPUB está ubicado en el directorio `build/epub` en la raíz del repositorio.

43.3.5 Agregar bibliotecas de terceros de Python

Importante: Después de modificar las dependencias de frc-docs de cualquier manera, `requirements.txt` debe regenerarse ejecutando `poetry export -f requirements.txt --output source/requirements.txt --without-hashes` desde la raíz de el repositorio.

---

frc-docs usa Poetry para administrar sus dependencias para asegurarse que contrucciones sean reproducibles.

Nota: Poetry **no** es necesaria para crear y contribuir al contenido de frc-docs. *Solo* se utiliza para la gestión de dependencias.

Instalar Poetry

Asegúrese de que Poetry esté instalado. Ejecute el siguiente comando: `pip install poet`.

Agregar una dependencia

Agregue la dependencia a la sección `[tool.poetry.dependencies]` de `pyproject.toml`. Asegúrese de especificar una versión exacta. Luego, ejecute el siguiente comando: `poetry lock --no-update`.

Actualizar una dependencia de nivel superior

Actualice la versión de la dependencia en la sección `[tool.poetry.dependencies]` de `pyproject.toml`. Luego, ejecute el siguiente comando: `poetry lock --no-update`.

Actualizar dependencias ocultas

Ejecute el siguiente comando: `poetry lock`.

43.4 Draw.io Instrucciones de Guardado

Advertencia: Make sure you don't modify any file that is in a `diagrams` folder, or ends in `.drawio.svg` in any program other than draw.io; otherwise you might risk breaking the metadata of the file, making it uneditable.

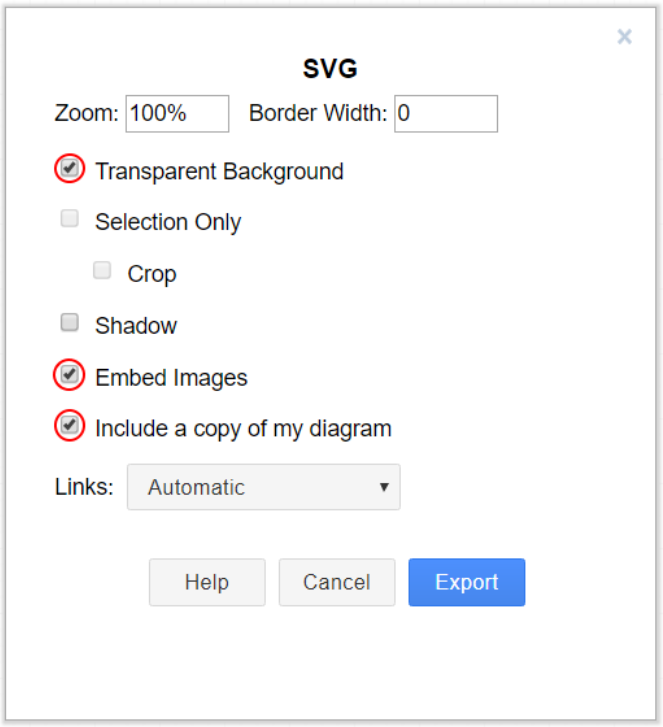
Draw.io (también conocido como diagrams.net) son compatibles cuando se guardan como archivos `svg`, con metadatos XML incrustados para el archivo fuente draw.io (normalmente almacenado como `.drawio`). Esto permite que estas imágenes actúen como archivos fuente para los diagramas que se pueden editar en el futuro y que se rendericen como archivos `svg` normales.

There are a few methods to save a diagram with the embedded metadata, but using the export menu is preferred because it allows us to embed any images in the diagram; otherwise they might not render properly on the docs.

Este método es aplicable tanto al escritorio de draw.io como a la versión web en diagrams.net.

Para exportar, vaya a `Archivo - Exportar como - SVG ...`. Asegúrese de que esté habilitado `Incluir una copia de mi diagrama` para incrustar los metadatos del diagrama, y que `Insertar imágenes` esté habilitado para que los archivos de imagen en el diagrama estén incrustados para que se muestren en los documentos. Además, marque la opción `Fondo transparente` para asegurarse de que el fondo se muestre correctamente.

El menú de exportación debería verse así:



Luego simplemente haga clic en `Exportar`, luego seleccione dónde desea guardar el archivo y guárdelo.

Nota: Al guardar, asegúrese de seguir la guía de estilo en *Archivos de Draw.io*

43.5 Traducciones

frc-docs admite traducciones mediante la utilidad basada en web Transifex. frc-docs ha sido traducido al español - México (es_MX), francés - Canadá (fr_CA) y turco - Turquía (tr_TR). Chino - China (zh_CN), hebreo - Israel (he_IL) y portugués - Brasil (pt_BR) tienen traducciones en curso. Los traductores que dominan *tanto* inglés como uno de los idiomas especificados serán muy apreciados para contribuir a las traducciones. Incluso una vez que se completa una traducción, debe actualizarse para mantenerse al día con los cambios en frc-docs.

43.5.1 Flujo de Trabajo

Aquí hay algunas instrucciones para traducir frc-docs.

1. Elabora una cuenta en Transifex y pide unirte a proyecto frc-docs, y pide acceso a contribuir al idioma preferido.
2. Únase a GitHub *discusiones <https://github.com/wpilibsuite/allwpilib/discussions>* __! Este es un medio de comunicación directo con el equipo de WPILib. Puede usar esto para hacernos preguntas de una manera rápida y ágil.
3. Es posible que te contactemos y te hagamos preguntas relacionadas con la contribución a las traducciones antes de obtener acceso al proyecto de traducción frc-docs.
4. ¡Traduce tu idioma!

43.5.2 Enlaces

Los enlaces deben conservarse en su sintaxis original. Para traducir un enlace, puede reemplazar el texto TRANSLATE ME (este será reemplazado por el título en inglés) con la traducción apropiada.

Un ejemplo del texto original podría ser

```
For complete wiring instructions/diagrams, please see the :doc:`Wiring the FRC␣
↪Control System Document <Wiring the FRC Control System document>`.
```

en cuyo caso, «Wiring the FRC Control System Document» se traduce.

```
For complete wiring instructions/diagrams, please see the :doc:`TRANSLATED TEXT
↪<Wiring the FRC Control System document>`.
```

Otro ejemplo está a continuación

```
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 Publicar las Traducciones

Las traducciones se extraen de Transifex y se publican automáticamente cada día.

43.5.4 Fidelidad

Las traducciones deben ser fieles al texto original. Si el texto original tiene áreas de oportunidad, abre un «Pull Request» o «Issue» en frc-dics. Puedes traducir el texto nuevo después de la actualización.

43.6 Traductores Principales

43.6.1 Chino

- 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 Francés

- 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 Portugués

- 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 Español

- 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 Turco

- 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 Hebreo

- Aric Radzin
- Dalton Smith
- Itay Ziv
- Ofek Ashery
- Shai Grossman
- Starlight220
- Yotam Shlomi

44 Desarrollando con allwpilib

Importante: Este documento contiene información de desarrolladores *de* WPILib. Esto no es para programar robots FRC®

Esta es una lista de enlaces a la diversa documentación para el repositorio allwpilib .

44.1 Quick Start

Below is a list of instructions that guide you through cloning, building, publishing and using local allwpilib binaries in a robot project. This quick start is not intended as a replacement for the information that is further listed in this document.

- Clone the repository with `git clone https://github.com/wpilibsuite/allwpilib.git`
- Build the repository with `./gradlew build` or `./gradlew build --build-cache` if you have an internet connection
- Publish the artifacts locally by running `./gradlew publish`
- Update your robot project's `build.gradle` to use the artifacts

44.2 Repositorio principal

44.3 NetworkTables

Índice

J

K

L

M

N

O

P

R

S

T

V

W

X