FIRST Robotics Competition

WPILib

2024 年 04 月 25 日

零到机器人

Welcome to the *FIRST*® Robotics Competition Control System Documentation! This site contains everything you need to know for programming a competition robot!

Community translations can be found in a variety of languages in the bottom-left menu.

Returning Teams

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

Changelog

Known Issues

Quick Start

New Teams

The Zero-to-Robot tutorial will guide you through preparation, wiring and programming a basic robot!

Go to Zero-to-Robot

Hardware Overview

An overview of the hardware components available to teams.

Go to Hardware Overview

Software Overview

An overview of the software components and tools available to teams.

Go to Software Overview

Programming Basics

Documentation that is useful throughout a team's programming process.

View articles

Advanced Programming

Documentation that is suited toward veteran teams. This includes content such as Path Planning and Kinematics.

View articles

Hardware

Hardware tutorials and content available for teams.

View articles

Romi and XRP Robots

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

View Romi articles

View XRP articles

API Documentation

Java, C++, and Python class documentation.

Java

C++

Python

Software Tools

Essential tooling such as FRC Driver Station, Dashboards, roboRIO Imaging Tool and more.

View articles

Example Projects

This section showcases the available example projects that teams can reference in VS Code.

View articles

Status Light Quick Reference

Quick reference guide for the status lights on a variety of FRC hardware.

View article

3rd Party libraries

Tutorial on adding 3rd party libraries such as CTRE and REV to your robot project.

View article

1 简介

欢迎来到 *FIRST*® Robotics Competition 控制系统和 WPILib 软件包的官方文档主页。此网页是记录 FRC® 控制系统 (包括排线、设置与软件) 和 WPILib 库和工具使用方法的一手资源。

1.1 不熟悉编程？

这些页面涵盖了 WPILib 库和 FRC 控制系统的细节，但没有讲述如何使用支持的基础编程语言。如果您想获得学习支持的编程语言的资源，请查看以下建议：

备注： 您可以继续执行“零到机器人”部分，以在不了解编程语言的情况下获得正常运行的基本机器人。除此之外，您将需要熟悉选择的编程语言。

1.1.1 Java

- Code Academy
- Head First Java 2nd Edition 是一个对新手极其友好的 Java 编程教程 (ISBN-10: 0596009208).

1.1.2 C++

- LearnCPP
- ‘编程：使用 C ++ 的原理和实践第二版 <https://www.amazon.com/dp/B00KPTEH8C>‘__ 是该语言的创建者自己对 C ++ 的介绍（ISBN-10：0321992784）。
- C ++ Primer Plus 6th Edition <https://www.amazon.com/dp/0321928423/>‘__（ISBN-10：0321776402）。

1.1.3 LabVIEW 软件

- NI 学习 LabVIEW

1.1.4 Python

- List of various guides to learn Python

1.2 零到机器人

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

第 1 步：搭建机器人

可以在:doc:‘此处 </docs/controls-overviews/control-system-hardware>‘找到的可用的控制系统硬件的概述。

2.1 Introduction to FRC Robot Wiring

备注： 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*)

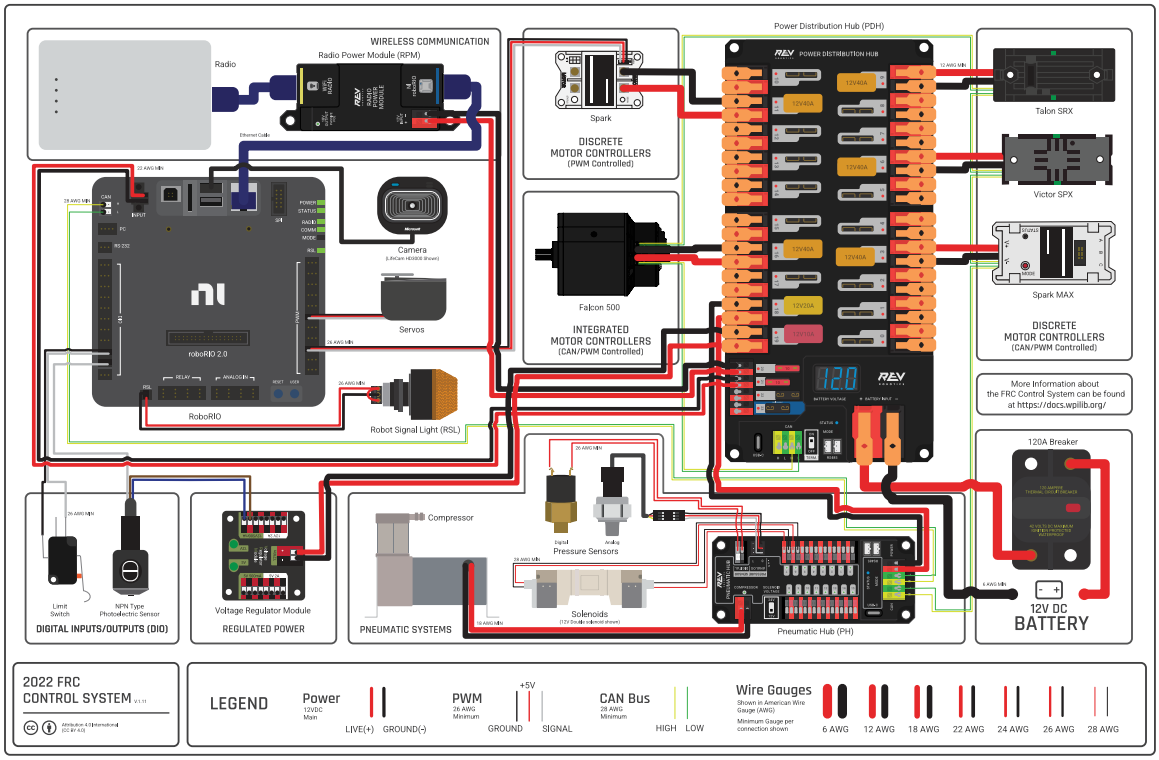


图 1: Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

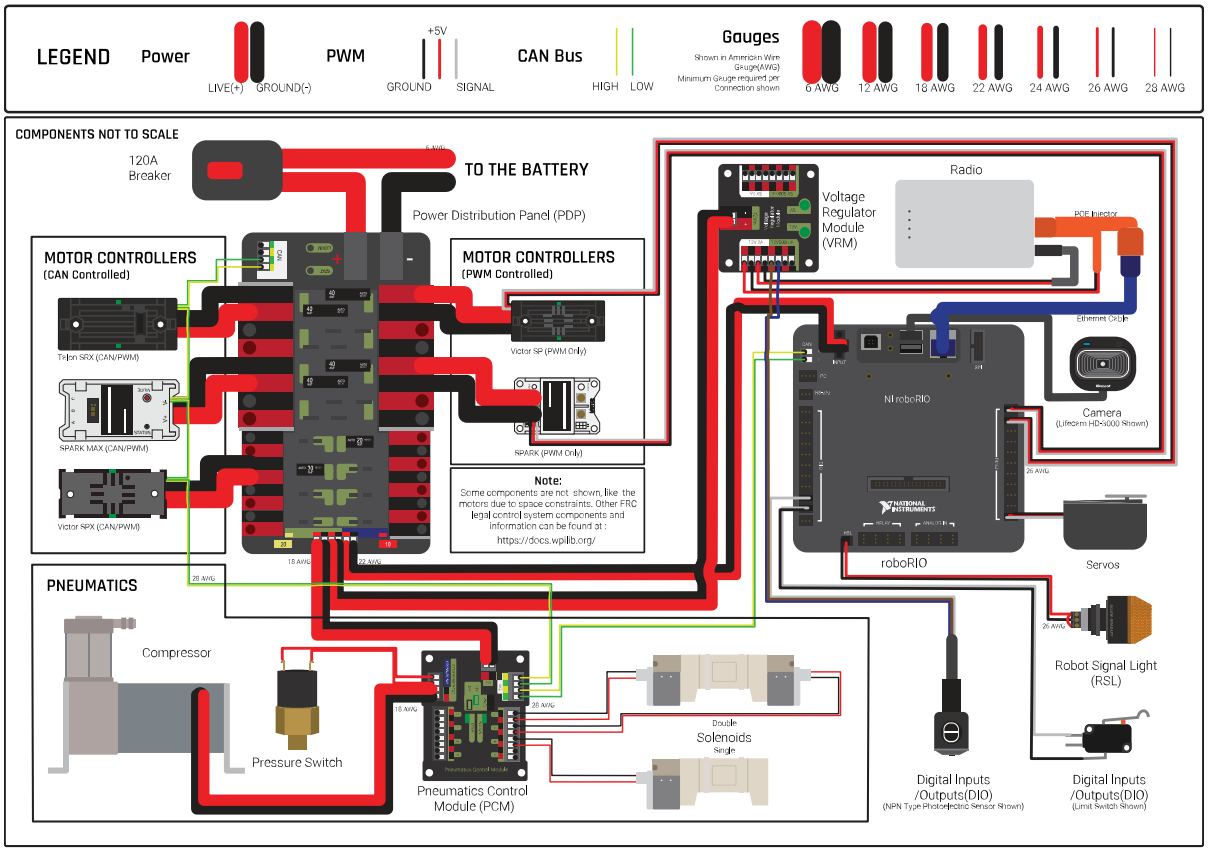


图 2: Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

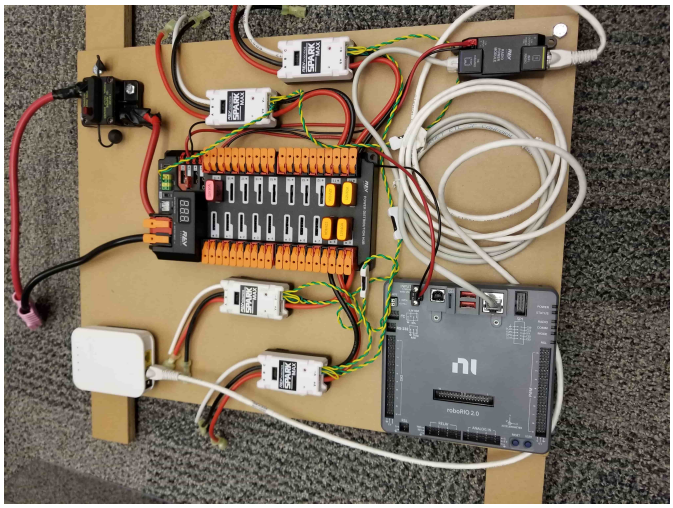
  - OpenMesh radio (with power cable and Ethernet cable)
  - Robot Signal Light (*RSL*)
  - 4x SPARK MAX or other motor controllers
  - 2x *PWM* y-cables
  - 120A Circuit breaker
  - 4x 40A Circuit breaker
  - 6 AWG (16 mm^2) Red wire
  - 10 AWG (6 mm^2) Red/Black wire
  - 18 AWG (1 mm^2) Red/Black wire
  - 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

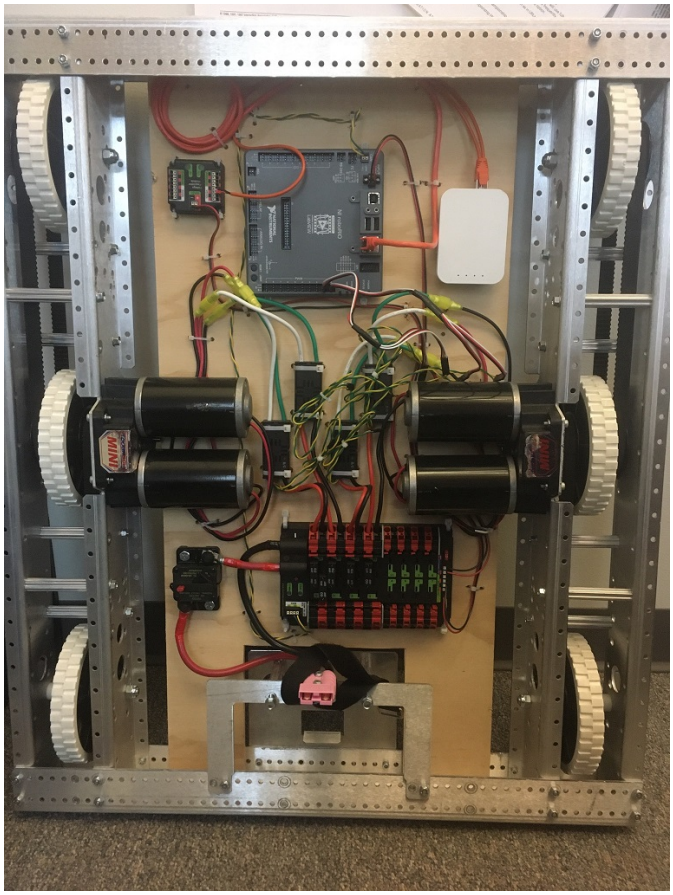
For a test board, cut piece of 1/4" or 1/2" (6-12 mm) material (wood or plastic) approximately 24" x 16" (60 x 40 cm). For a Robot Quick Build control board see the supporting documentation for the proper size board for the chosen chassis configuration.

2.1.4 Layout the Core Control System Components

REV

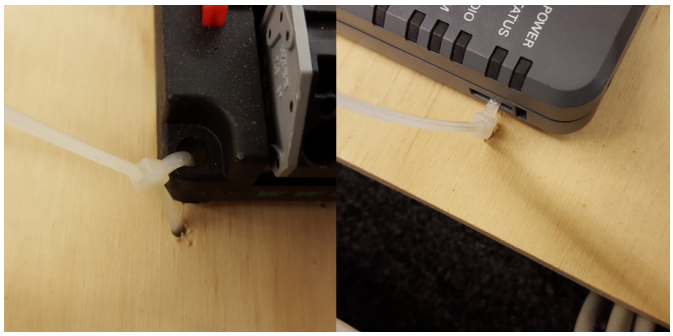


CTR



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.

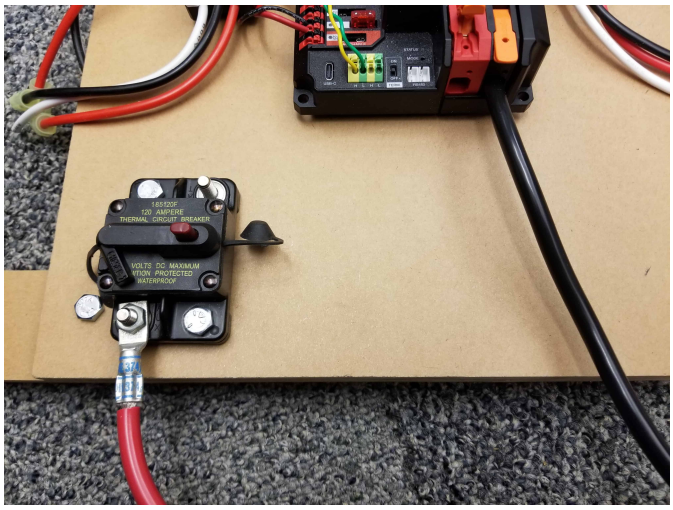
2.1.6 Attach Robot Side Battery Connector

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.



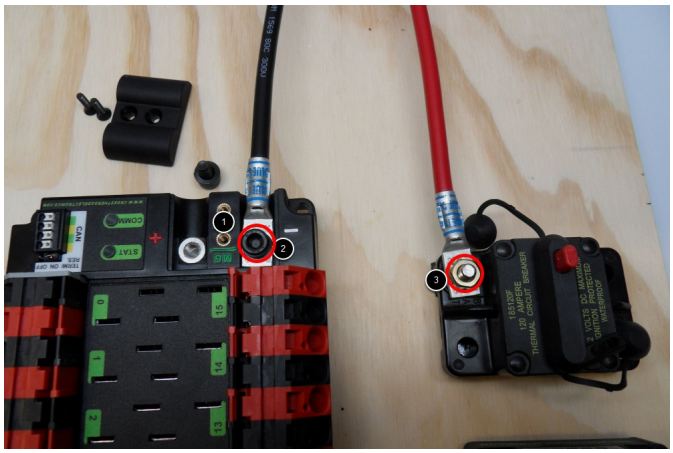
Requires: Battery Connector, 6 AWG (16 mm^2) terminal lugs, 7/16" (11 mm) Box end

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



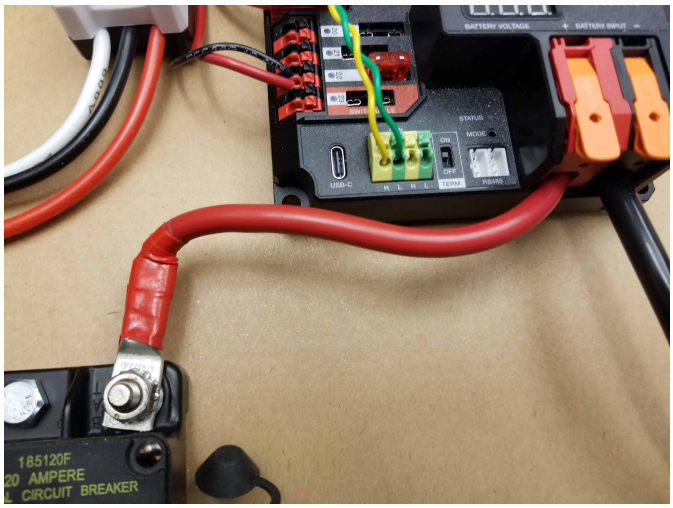
Requires: Battery Connector, 6 AWG (16 mm^2) terminal lugs, 1/16" Allen, 5 mm Allen, 7/16" (11 mm) Box end

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

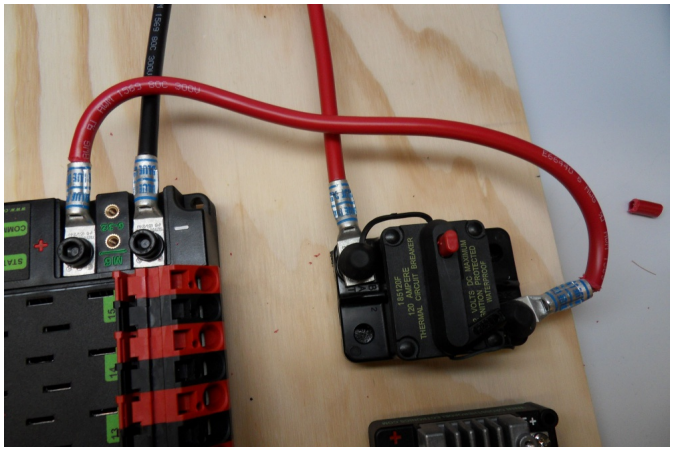


Requires: 6 AWG (16 mm^2) red wire, 1x 6 AWG (16 mm^2) terminal lugs, 7/16" (11 mm) wrench

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.

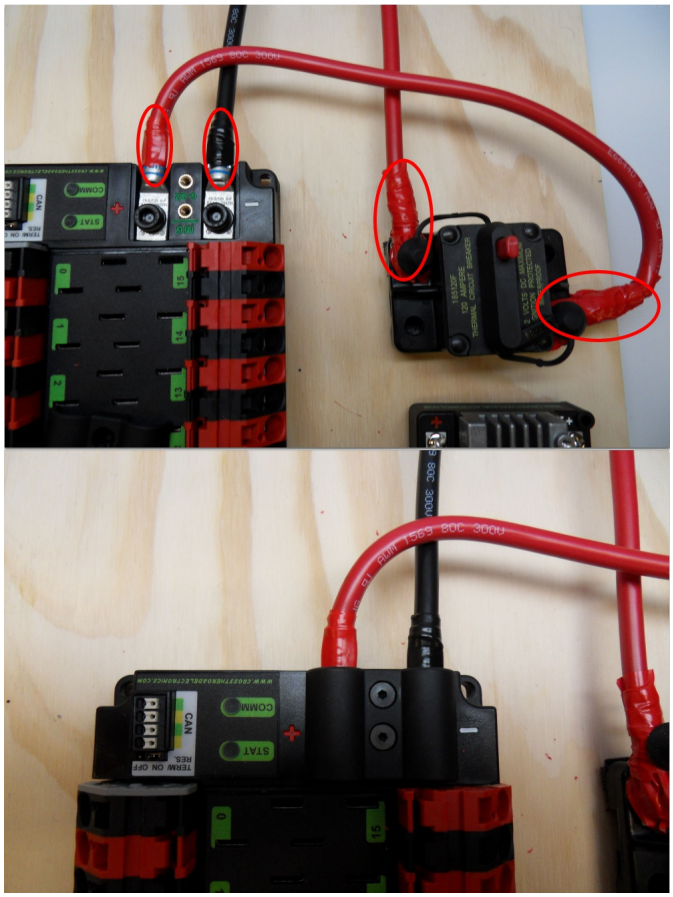
2.1.8 Insulate power connections

REV

Requires: Electrical tape

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

CTR

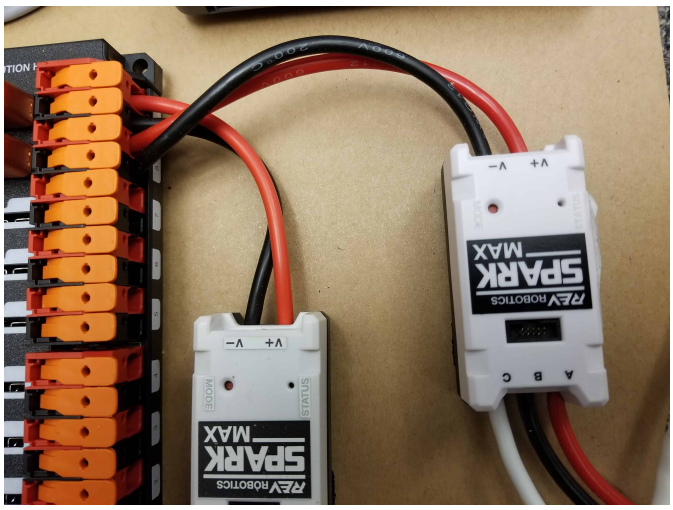


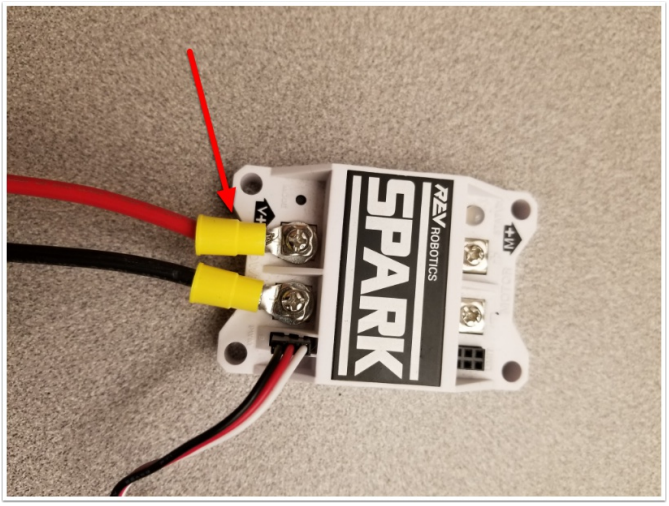
Requires: 1/16” Allen, Electrical tape

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):

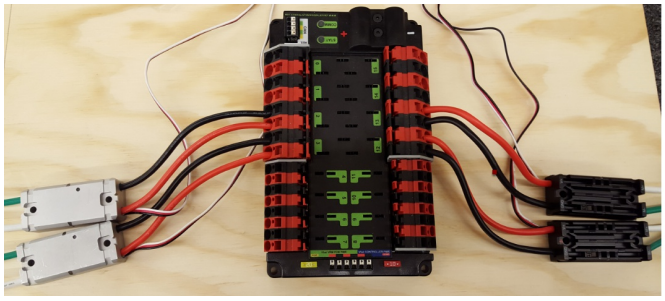
1. Cut red and black wire to appropriate length to reach from one of the Wago terminal pairs to the input side of the motor controller (with a little extra for the length that will be inserted into the terminals on each end)
2. Strip one end of each of the wires, then insert into the Wago terminals.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

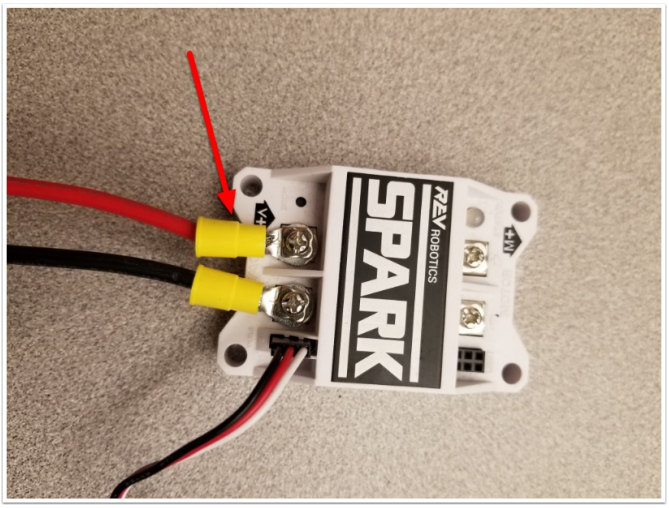
CTR

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:

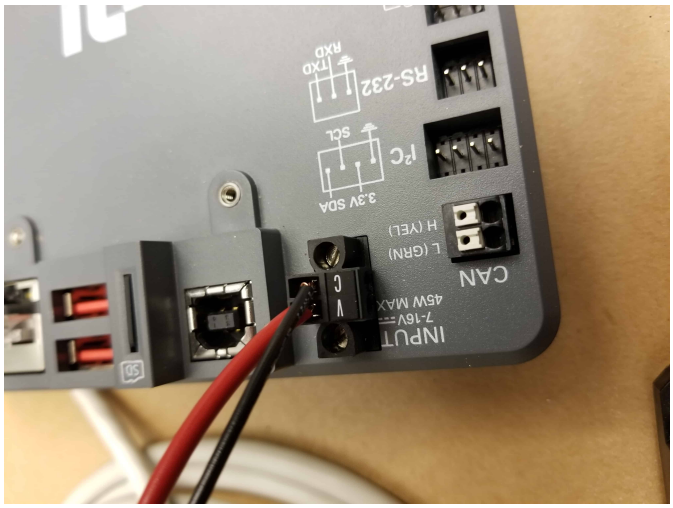
- Wire should be 16 AWG (1.5 mm^2) to 24 AWG (0.25 mm^2) (consult rules to verify required gauge for power wiring)
- Wire ends should be stripped approximately 5/16 (~8 mm)"
- To insert or remove the wire, press down on the corresponding "button" to open the terminal

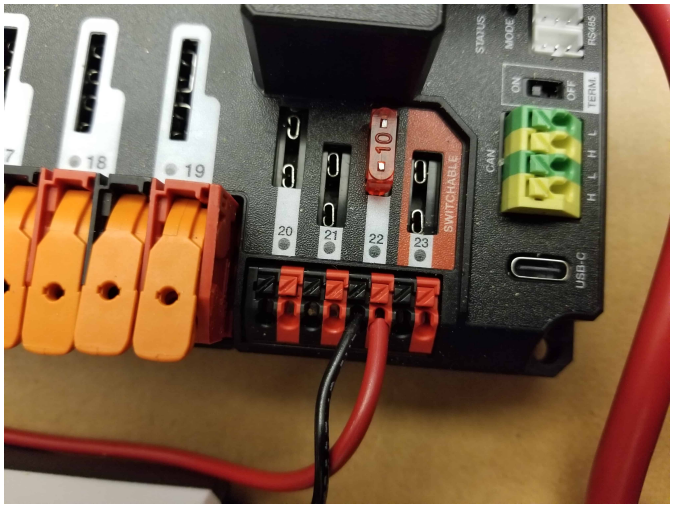
After making the connection check to be sure that it is clean and secure:

- 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

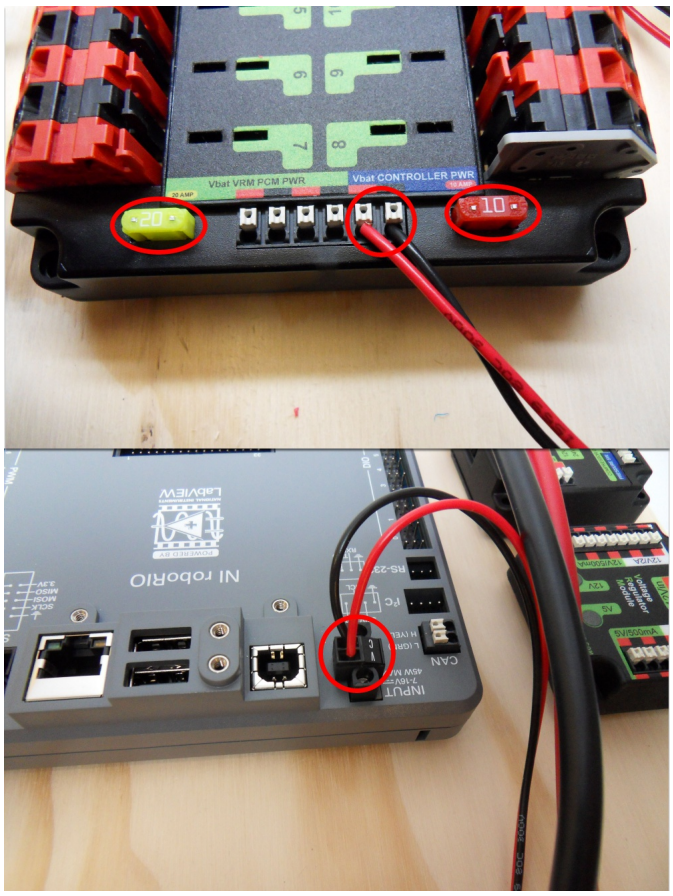




Requires: 10A mini fuse, Wire stripper, very small flat screwdriver, 18 AWG (1 mm^2) Red and Black

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

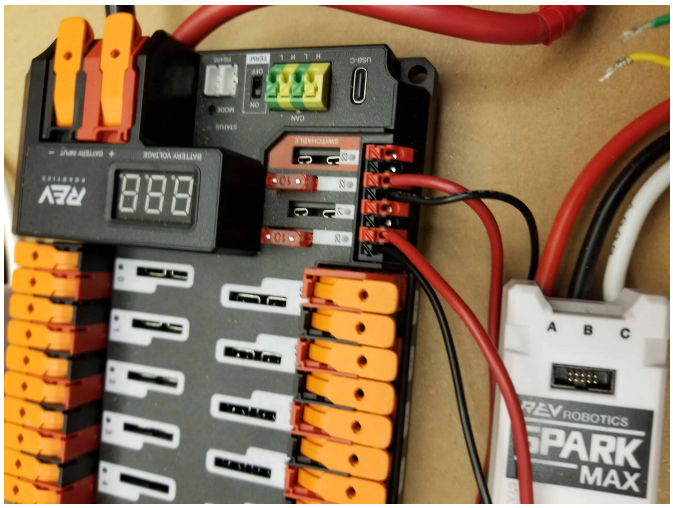


Requires: 10A/20A mini fuses, Wire stripper, very small flat screwdriver, 18 AWG (1 mm^2) Red and Black

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

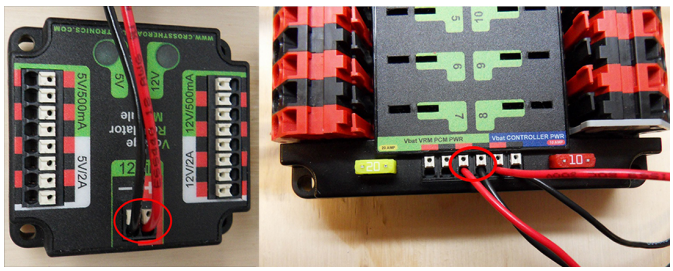
REV



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.

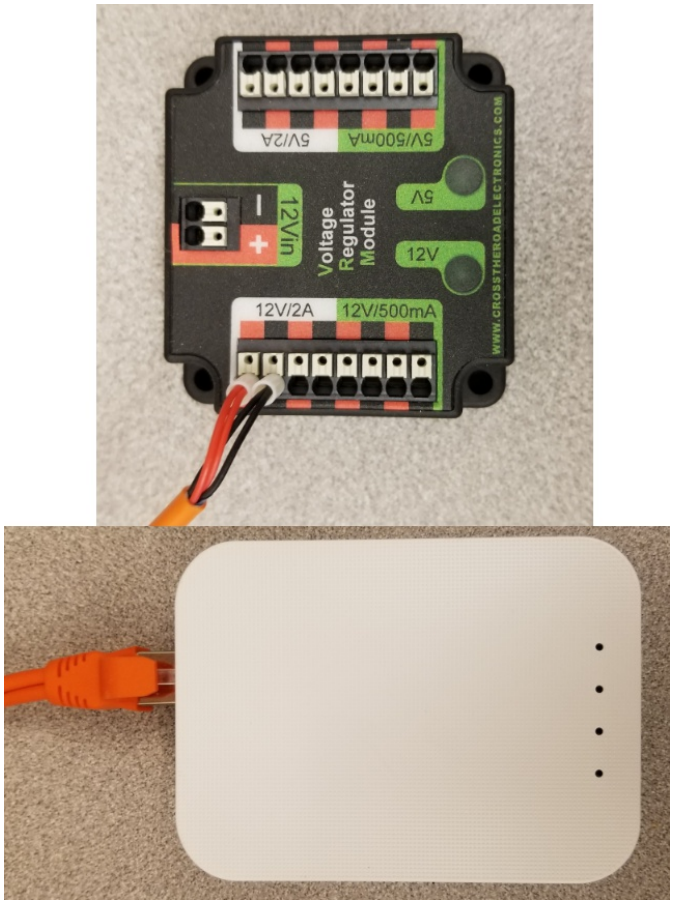
CTR



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.

警告: 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.

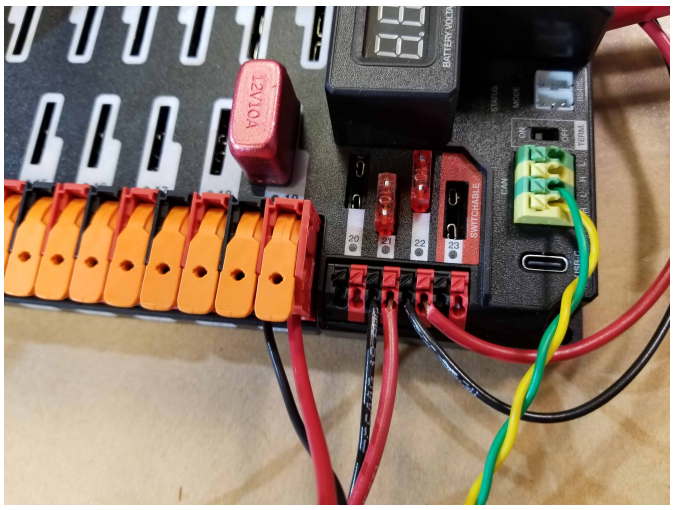


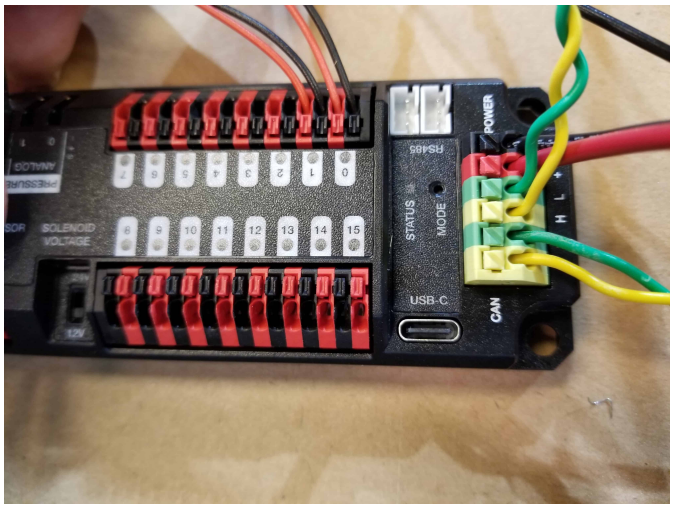
Requires: Small flat screwdriver (optional), Rev radio PoE cable

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

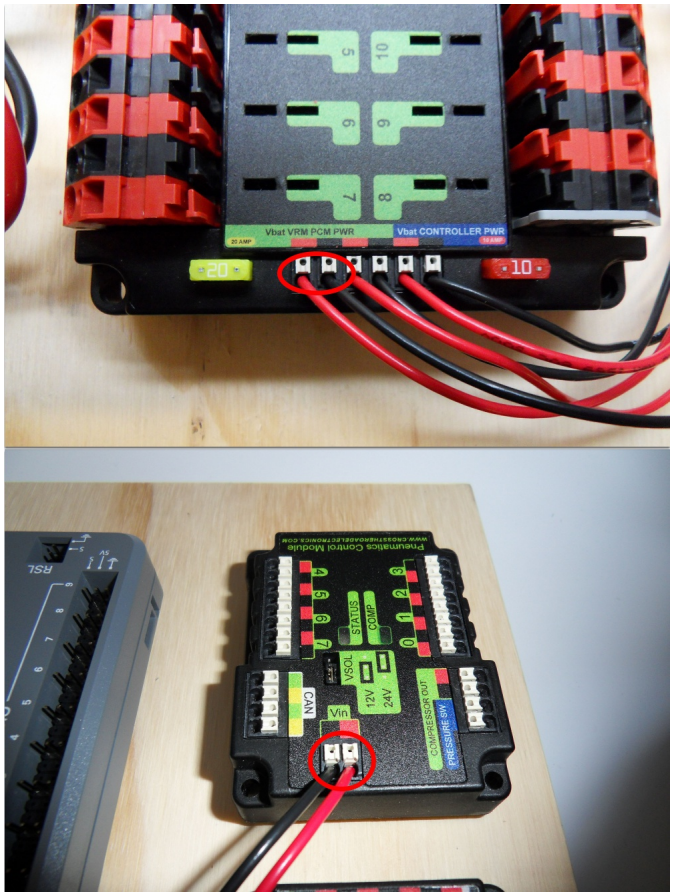
---

备注： 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

备注： 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

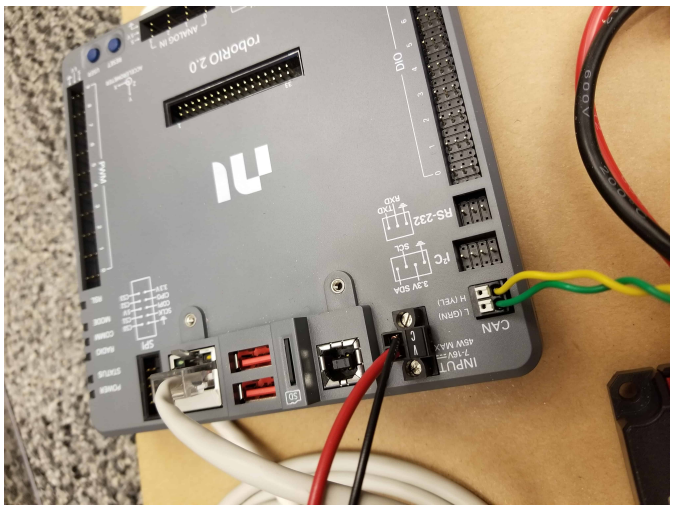
Requires: Ethernet cable

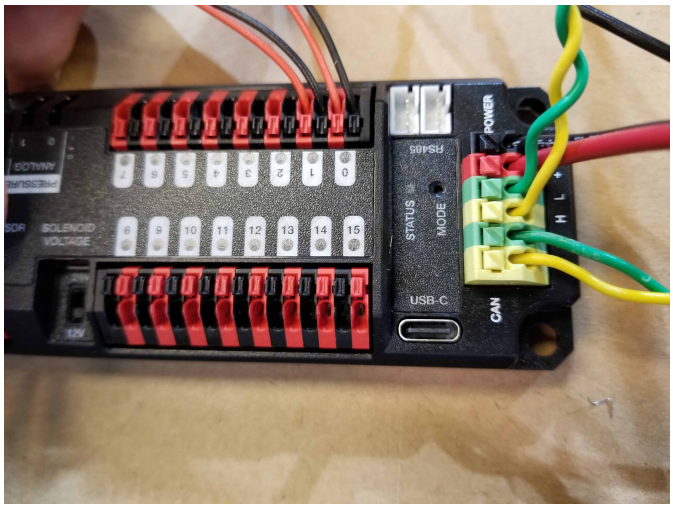
Connect an Ethernet cable from the RJ45 (Ethernet) socket of the Rev Passive POE cable to the RJ45 (Ethernet) port on the roboRIO.

2.1.15 CAN Devices

roboRIO to Pneumatics CAN

REV





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

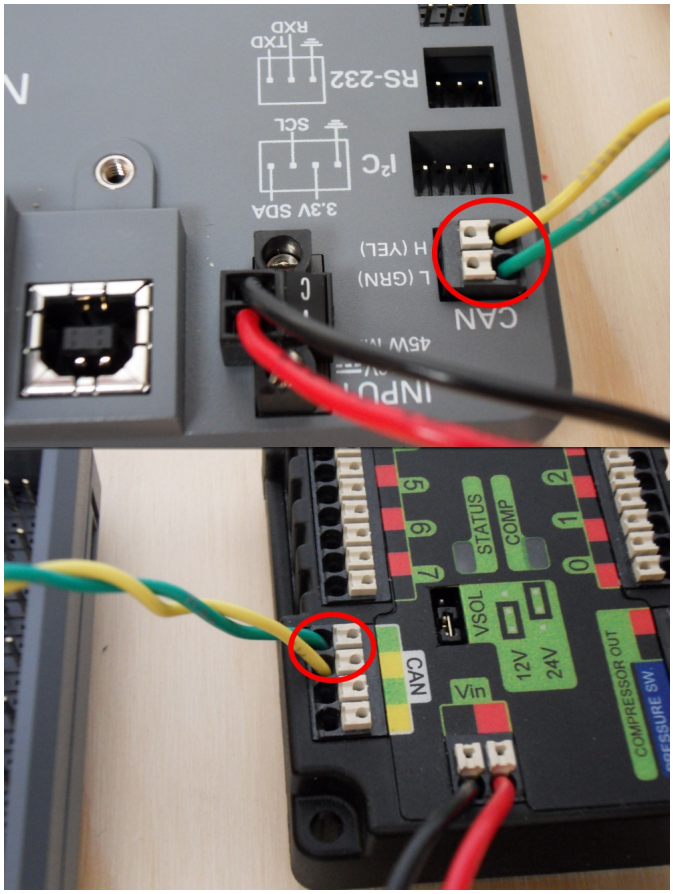
---

备注: 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

---

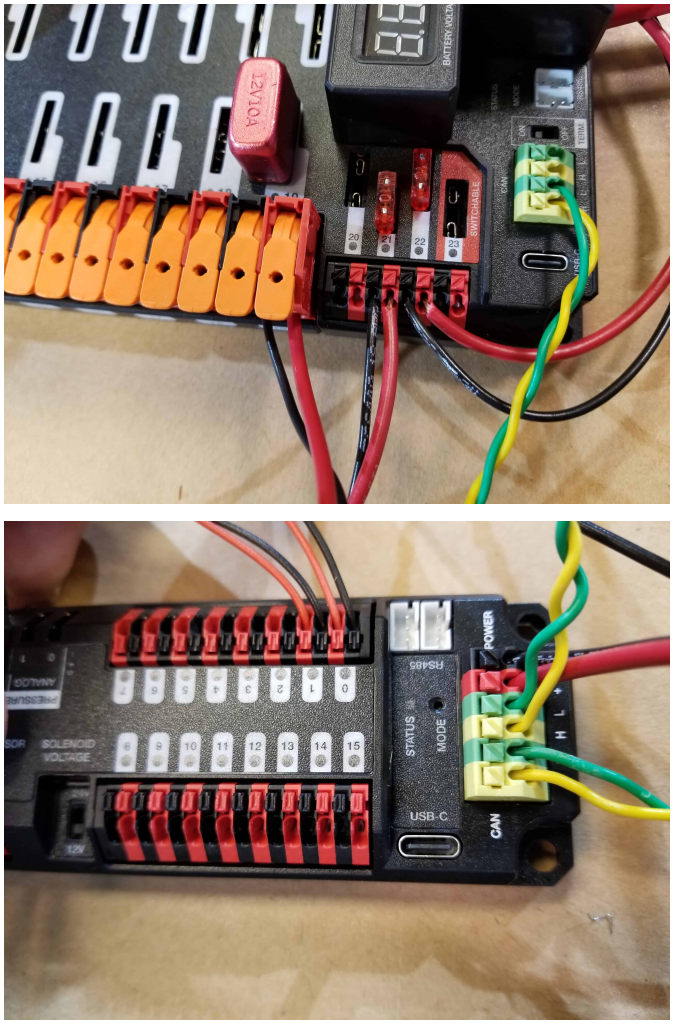
备注： The PCM is an optional component used for controlling pneumatics on the robot. If you are not using the PCM, wire the CAN connection directly from the roboRIO (shown in this step) to the PDP (shown in the next step).

---

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

---

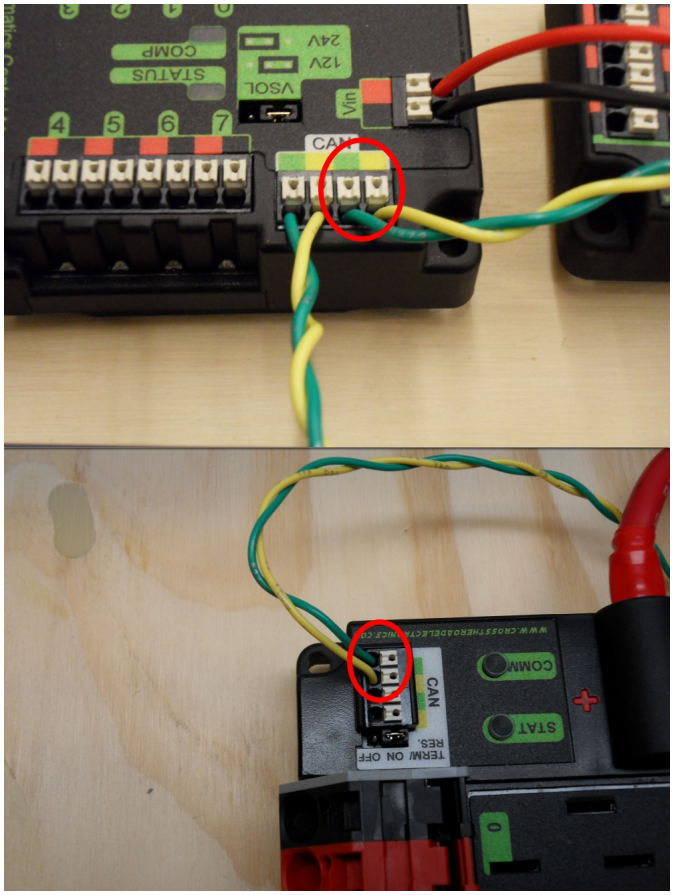
备注: The PH is an optional component used for controlling pneumatics on the robot. If you are not using the PH, wire the CAN connection directly from the roboRIO (shown in the above step) to the PDH (shown in this step).

---

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.

备注： 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

备注： 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).

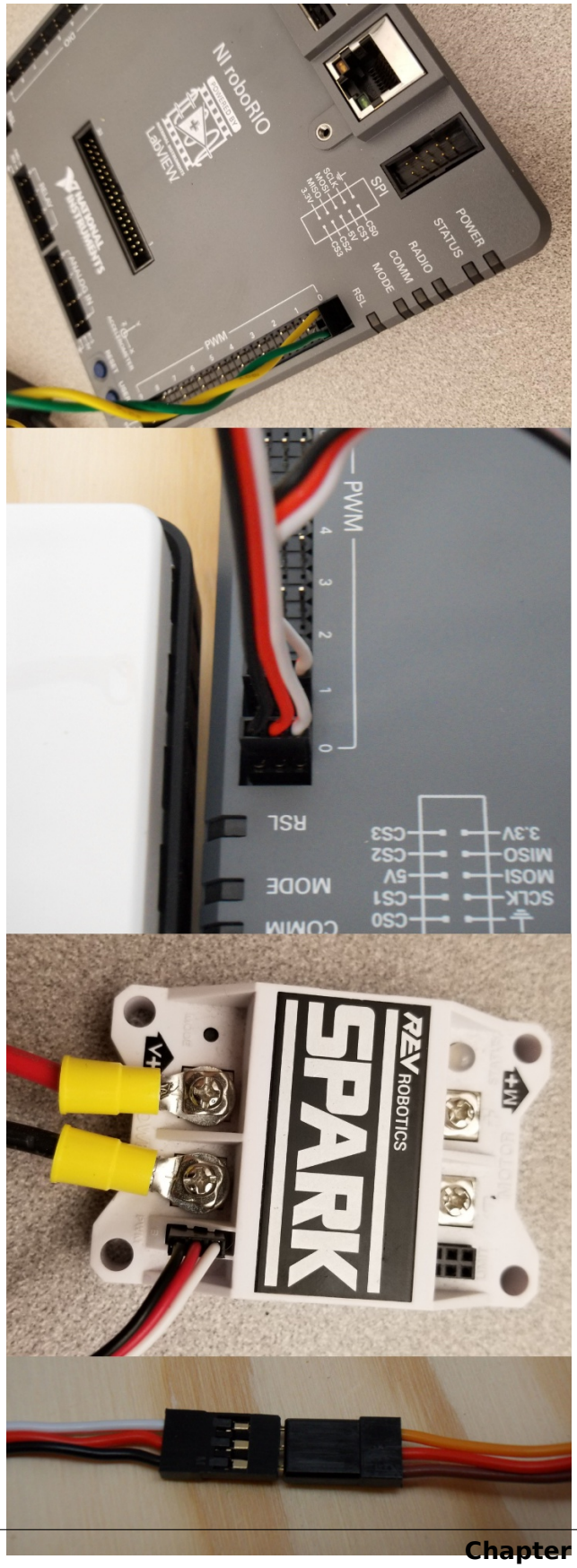
1. Strip ~5/16" (~8 mm) off of each of the CAN wires.
2. Insert the wires into the appropriate CAN terminals on the PCM.
3. Measure the length required to reach the CAN terminals of the PDP (either of the two available pairs). Cut and strip ~5/16" (~8 mm) off this end of the wires.
4. Insert the wires 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.

备注： 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
SCLK
CS3
CS2
5V
CS1
CS0
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.

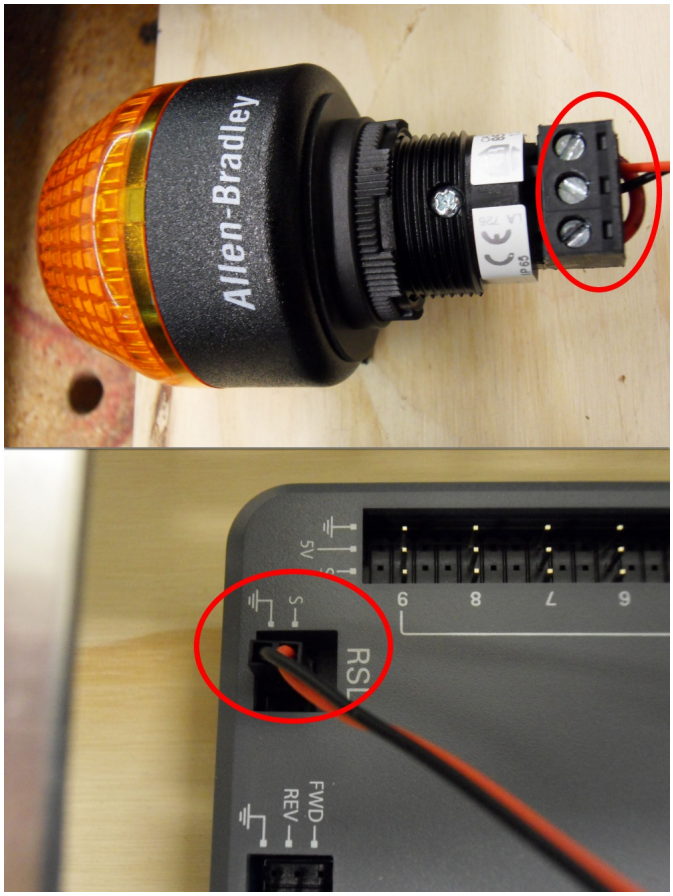
CAN

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.

小技巧: 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

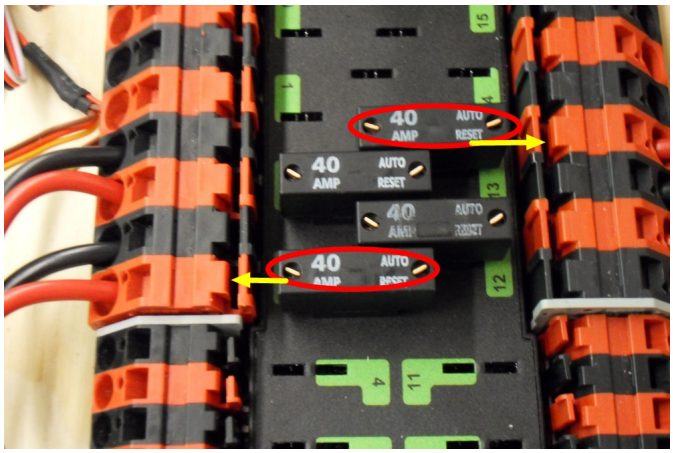
REV

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



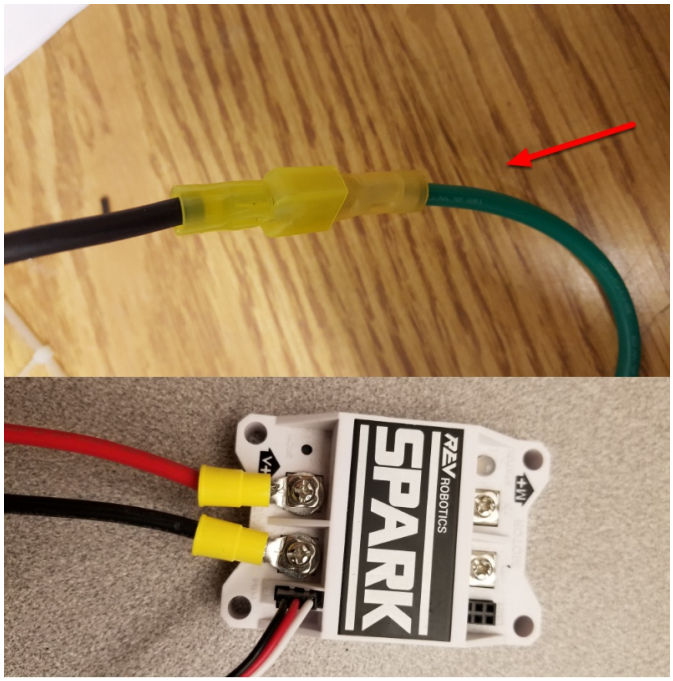
Requires: 4x 40A circuit breakers

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

危险: 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!

小技巧: 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

小技巧: 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.

小技巧: 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

步骤 2：安装软件

可用的控制系统软件的概述可在:doc:‘此处 </docs/controls-overviews/control-system-software>‘找到

3.1 离线安装准备

本文包含有关要脱机安装 FRC® 控制系统软件时要收集的组件的说明/链接。

小技巧： 本文档编译了以下文档中的所有下载链接，以使其更易于在脱机计算机或多台计算机上安装。如果要在连接到互联网的单台计算机上安装，则可以跳过此页面。

备注： 对于 Java 和 C ++ 团队，这些工具的安装顺序无关紧要。LabVIEW 应该在 FRC 游戏工具或第三方库之前安装。

3.1.1 文档

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

3.1.2 安装人员

所有队伍

- 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

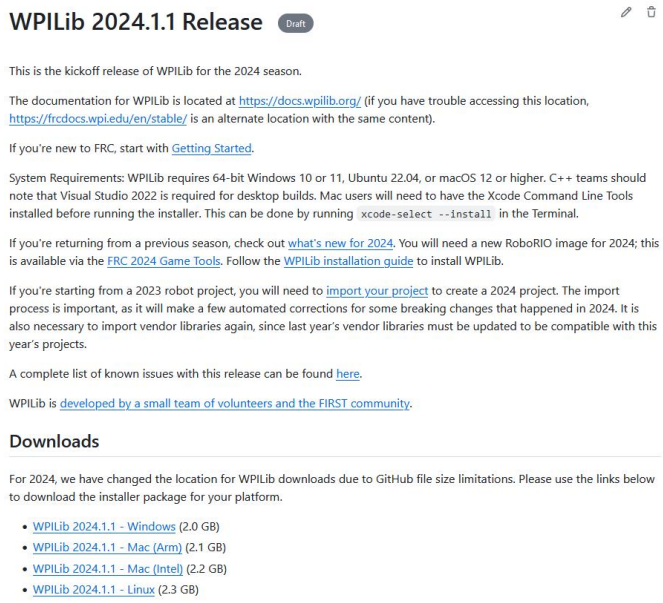
LabVIEW 团队

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

Java / C ++ 团队

- Java/C++ WPILib Installer

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



WPILib 2024.1.1 Release Draft

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

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

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

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

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

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

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

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

Downloads

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

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

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

备注： 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 第三方库/软件

在第三方库中可以找到可插入 WPILib 的可用第三方软件的目录。

3.2 安装 LabVIEW for FRC（仅 LabVIEW）

备注： 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*.

下载和安装时间会因计算机和 Internet 连接规格的不同而有很大差异，但是请注意，此过程涉及大量文件的下载和安装，可能至少需要一个小时才能完成。

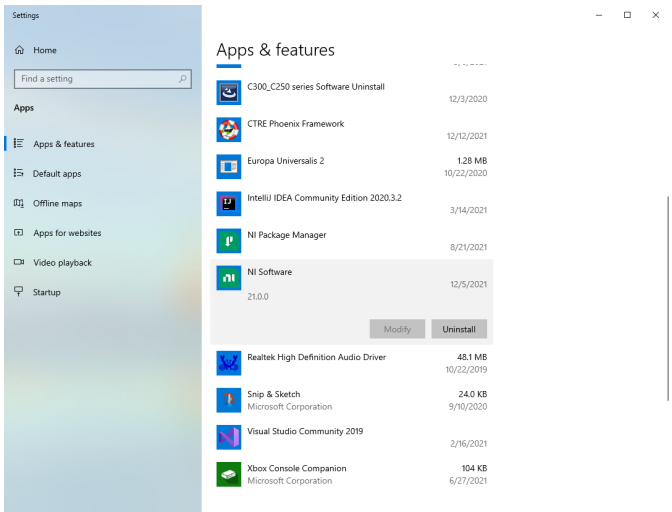
3.2.1 Requirements

- Windows 10 or higher (Windows 10, 11)

3.2.2 卸载旧版本（推荐）

备注： 如果希望继续对 cRIO 进行编程，则需要保持安装 LabVIEW forFRC®2014。LabVIEW for FRC 2014 许可证已得到扩展。虽然这些版本应该可以在单台计算机上共存，但是这不是经过广泛测试的配置。

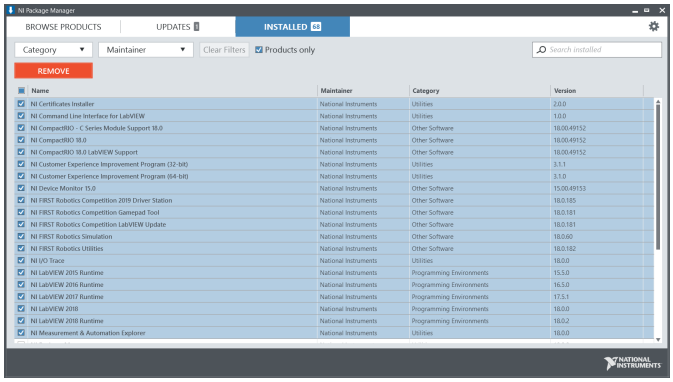
Before installing the new version of LabVIEW it is recommended to remove any old versions. The new version will likely co-exist with the old version, but all testing has been done with FRC 2024 only. Make sure to back up any team code located in the "User\LabVIEW Data" directory before un-installing.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



选择要卸载的组件

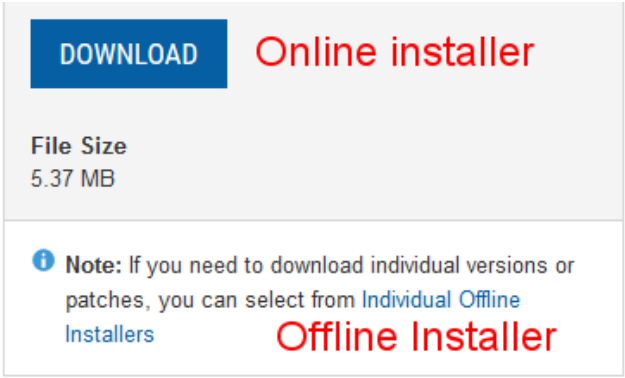
在出现的对话框中，选择所有条目。最简单的方法是取消选中“仅产品”复选框，然后选择“名称”左侧的复选框。单击删除。等待卸载程序完成并根据提示重启。

警告： 这些说明假定未安装其他 NI 软件。如果安装了其他 NI 软件，则必须取消选中不应卸载的软件。



3.2.3 获取 LabVIEW 安装程序

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



如果要脱机安装在其他计算机上，请不要单击“下载”按钮，单击“单个脱机安装程序”，然后单击“下载”以下载完整的安装程序。

备注： 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 安装 LabVIEW

NI LabVIEW 需要许可证。每个季节的许可证在第二年的 1 月 31 日之前有效（例如 2020 季节的许可证在 2021 年 1 月 31 日到期）

根据适用软件随附的限制和许可条款，允许团队根据需要在尽可能多的团队计算机上安装软件，但前提是只有团队成员或指导者才能使用该软件，并且仅用于 FRC。使用 LabVIEW 的权利仅受适用软件安装过程中显示的许可协议条款的约束。

开始安装

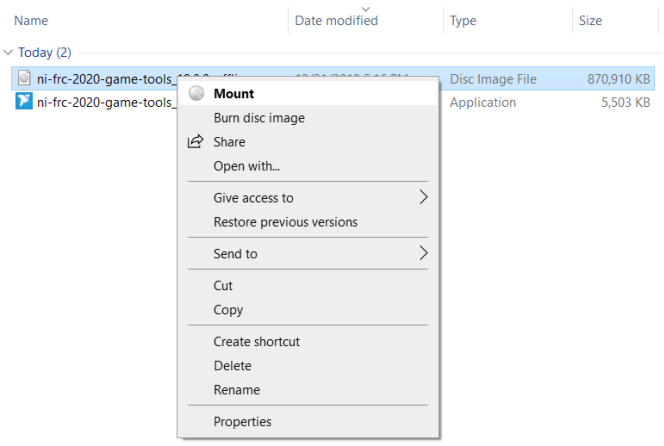
在线安装程序

运行下载的 exe 文件以开始安装过程。如果出现 Windows 安全提示，请单击“是”。

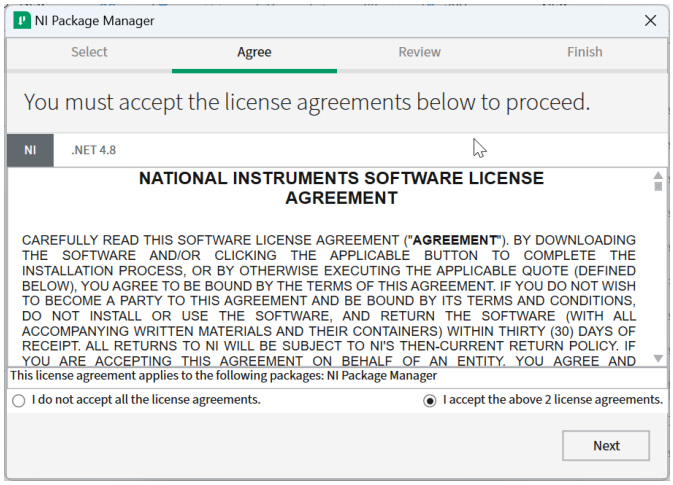
离线安装程序（Windows 10+）

右键单击下载的 iso 文件，然后选择挂载。从已安装的 iso 运行 install.exe。如果出现 Windows 安全提示，请单击“是”

备注： 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.

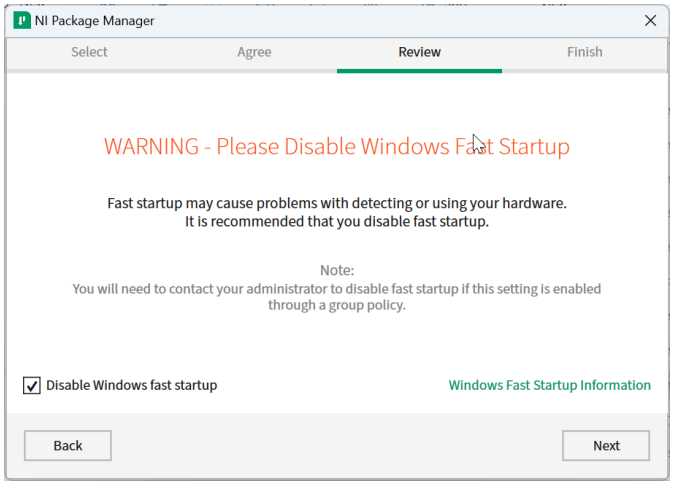


NI Package Manager 许可证



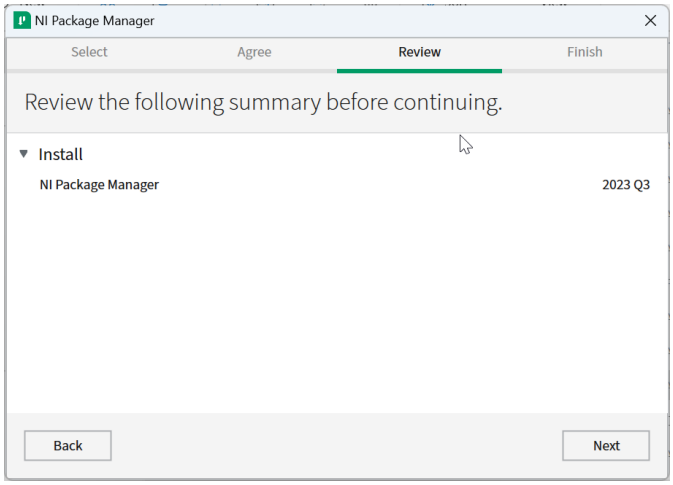
如果看到此屏幕，请单击“下一步”。

禁用 Windows 快速启动



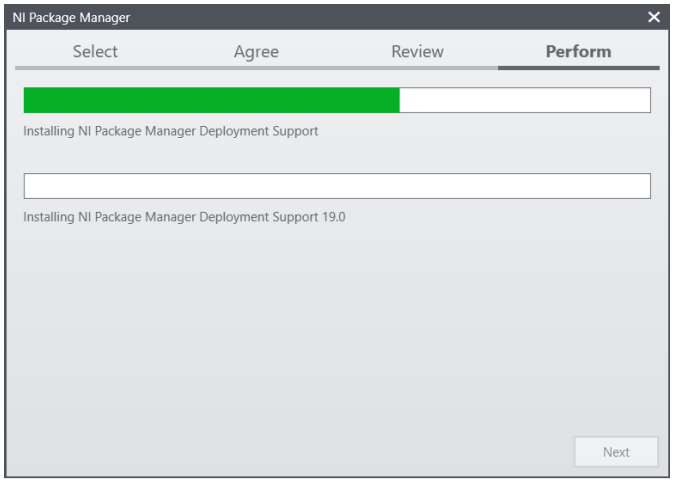
如果看到此屏幕，请单击“下一步”。

NI Package Manager 评估



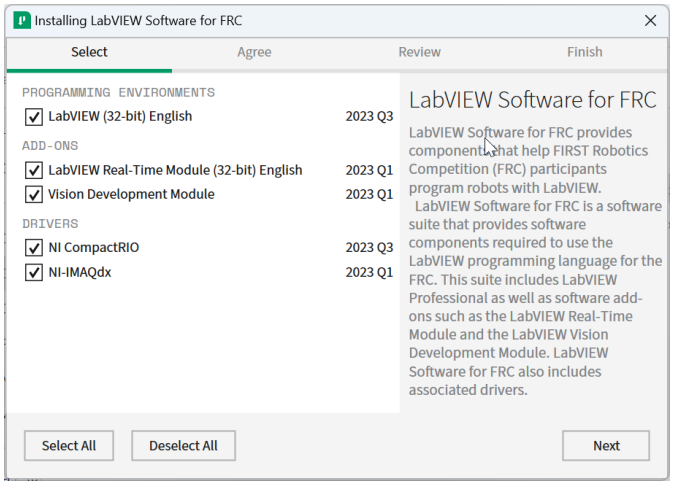
如果看到此屏幕，请单击“下一步”。

NI Package Manager 安装



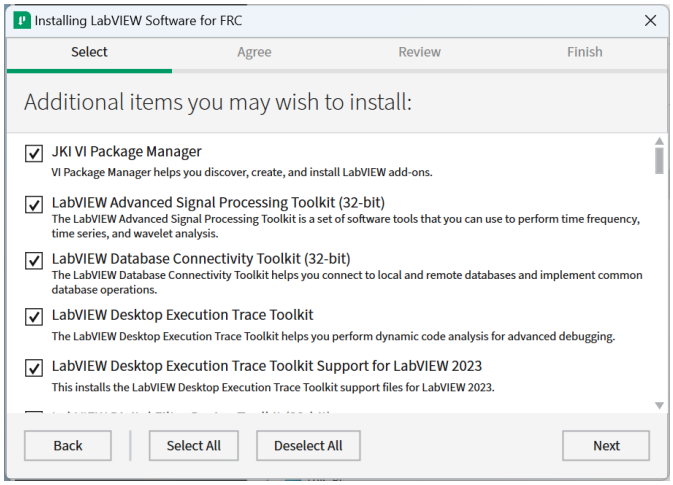
NI Package Manager 的安装进度将在此窗口中跟踪

产品列表



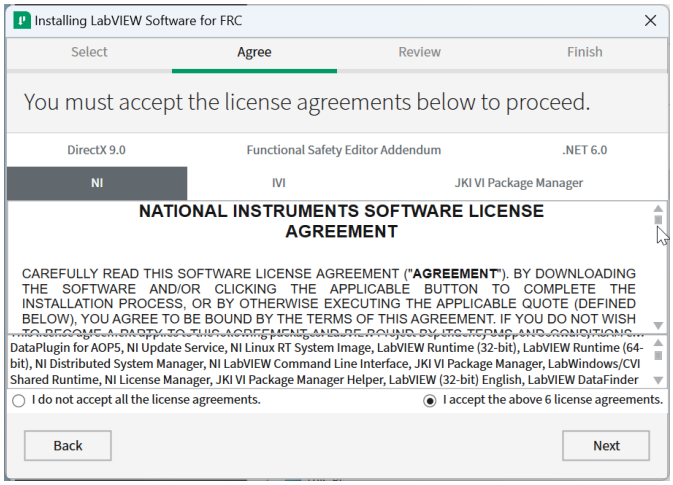
点击：下一步

其他包



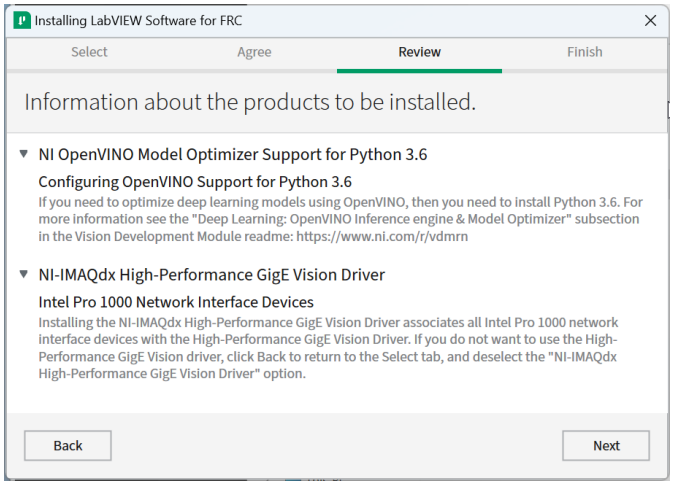
点击：下一步

许可协议



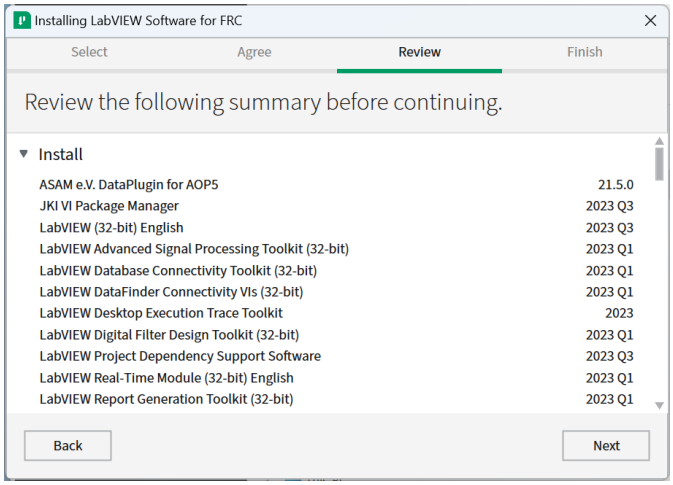
选中“我接受…”，然后单击“下一步”。

产品信息



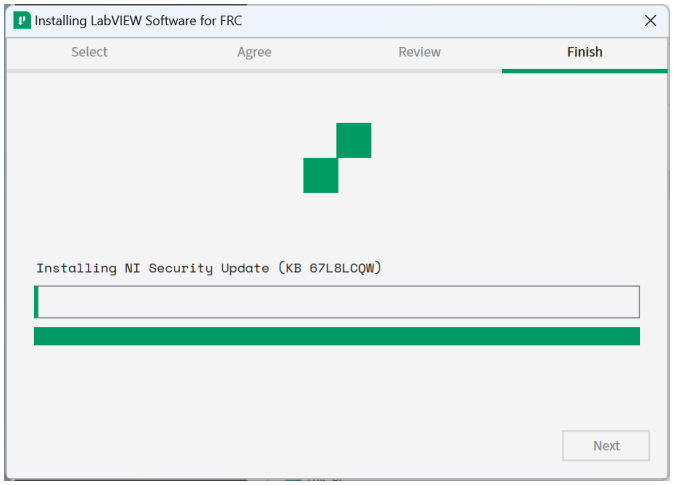
点击：下一步

开始安装



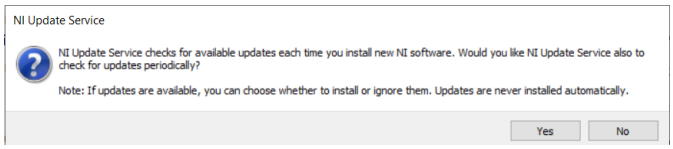
点击：下一步

总体进程



在此窗口中将跟踪总体安装进度

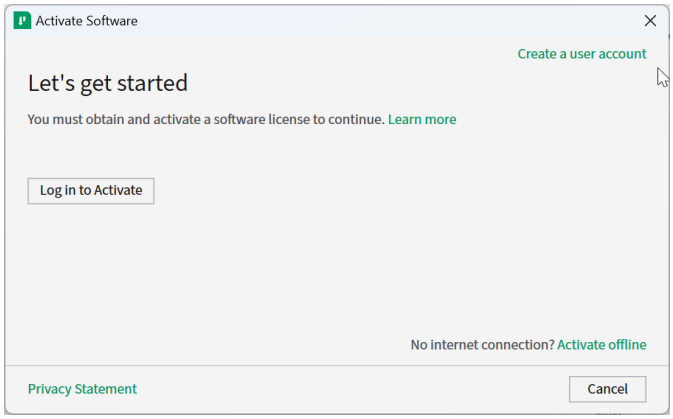
3.2.5 NI 更新服务



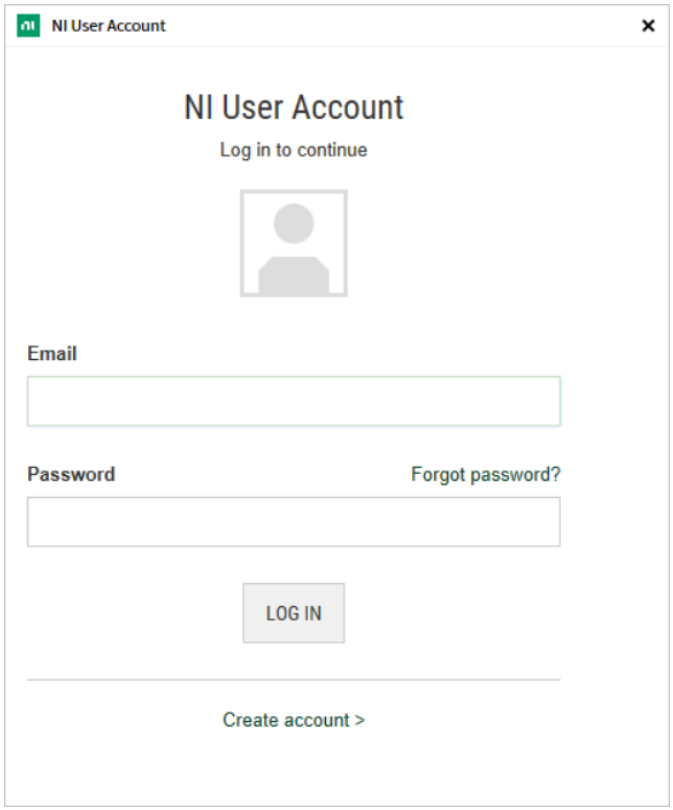
系统将提示您是否启用 NI 更新服务。您可以选择不启用更新服务。

警告： 除非 FRC 通过我们通常的沟通渠道（FRC 博客，团队更新或电子邮件）指导，否则不建议安装这些更新。

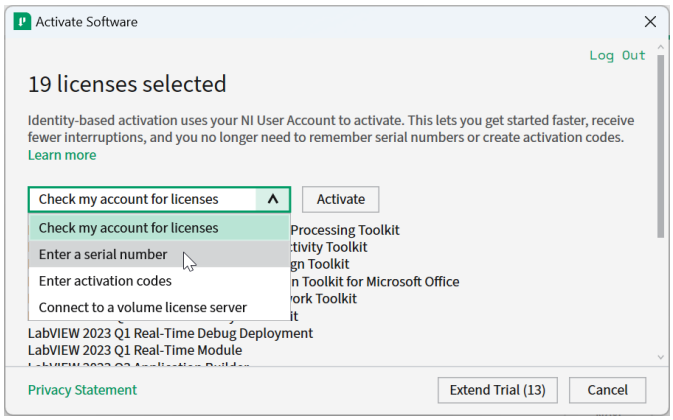
NI 激活向导



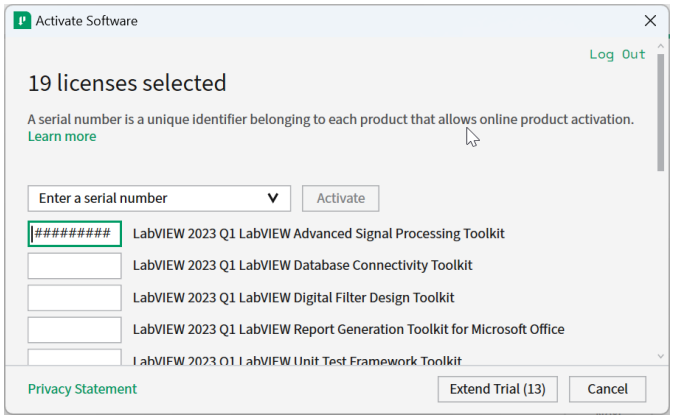
Click the *Log in to Activate* button.



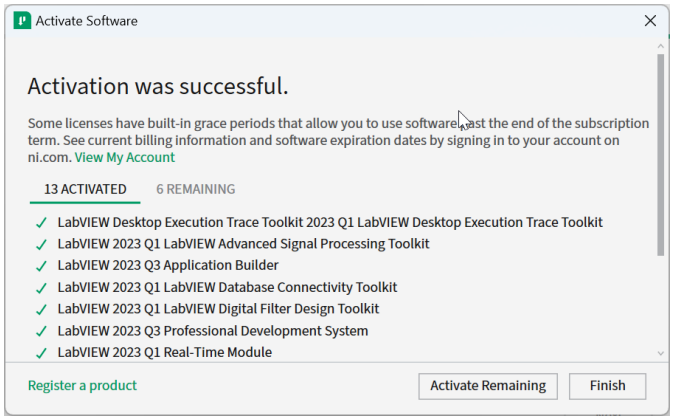
登录您的 ni.com 帐户。如果您没有帐户，请选择 *Create account* 创建一个免费帐户。



From the drop-down, select enter a serial number

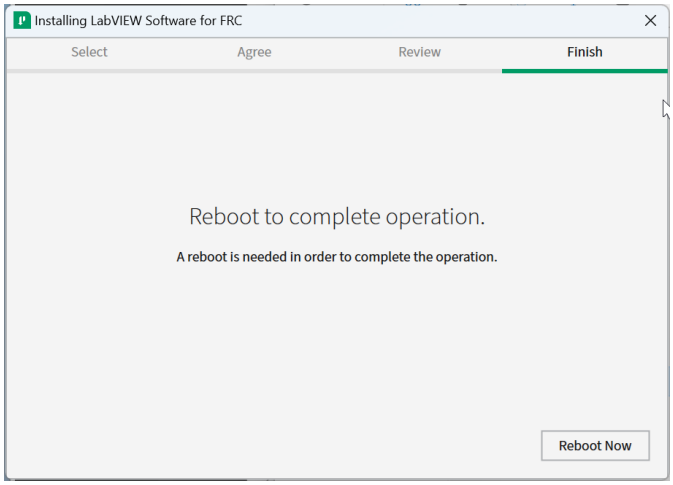


Enter the serial number in all the boxes. Click *Activate*.



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*.

重启



关闭所有打开的程序后，选择“立即重启”。

3.3 安装 FRC Game Tools

The FRC® Game Tools 包含以下组件

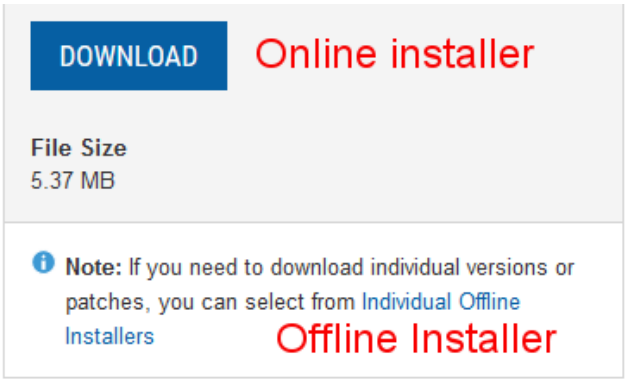
- LabVIEW Update
- FRC Driver Station
- FRC roboRIO Imaging Tool and Images

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

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

3.3.1 要求

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



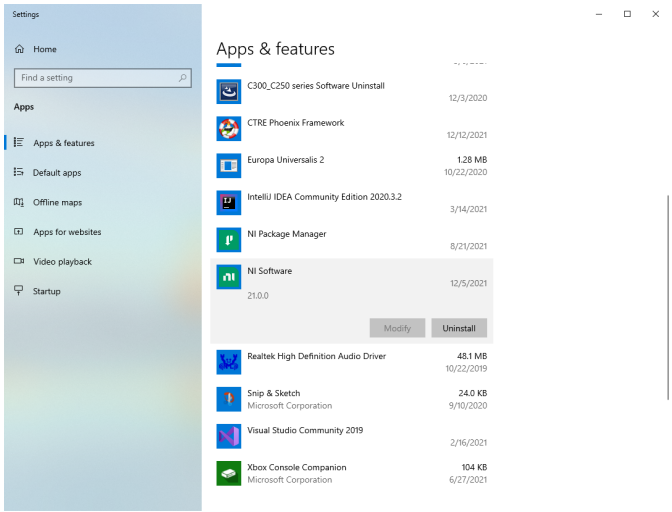
如果您希望脱机安装在其他计算机上，请在单击“下载”以下载完整的安装程序之前，单击“脱机安装程序”。

3.3.2 卸载旧版本（推荐）

重要： LabVIEW 团队已完成此步骤，请勿重复。LabVIEW 团队应跳至:ref:*docs/zero-to-robot/step-2/frc-game-tools:installation* section.

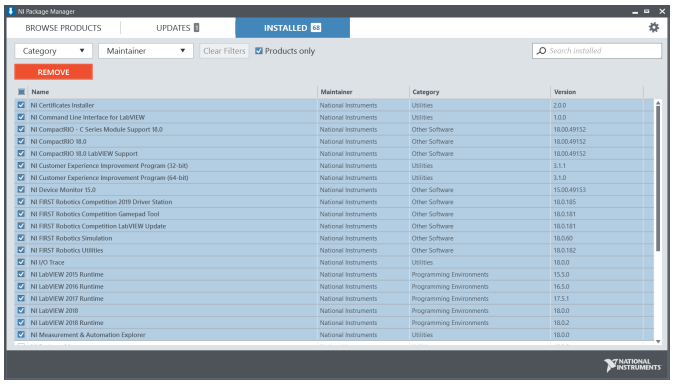
Before installing the new version of the FRC Game Tools it is recommended to remove any old versions. The new version will likely co-exist with the old version (note that the DS will overwrite old versions), but all testing has been done with FRC 2024 only. Then click Start » Add or Remove Programs.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

备注： 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.



选择要卸载的组件

在出现的对话框中，选择所有条目。最简单的方法是取消选中“仅限产品”复选框，然后选中“名称”左侧的复选框。点击“删除”。等待卸载程序完成并根据提示重新启动。



3.3.3 安装

重要： Game Tools 安装程序可能会提示需要更新或安装.NET Framework 4.6.2。按照屏幕上的提示完成安装，包括根据要求重新启动。然后继续安装 FRC Game Tools，并在必要时重新启动安装程序。

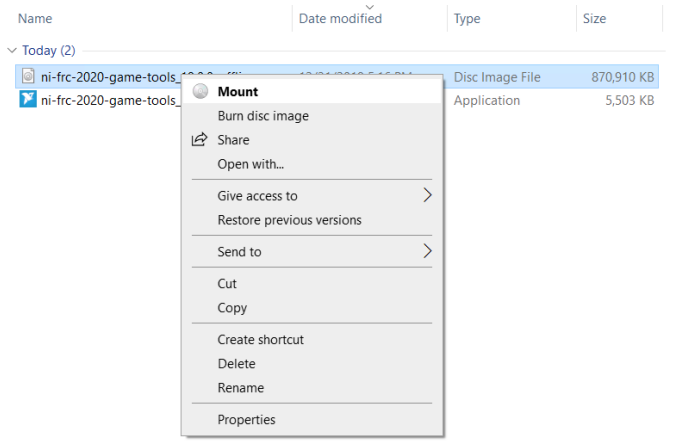
提取

在线

运行下载的可执行文件以开始安装过程。如果出现 Windows 安全提示，请单击“是”。

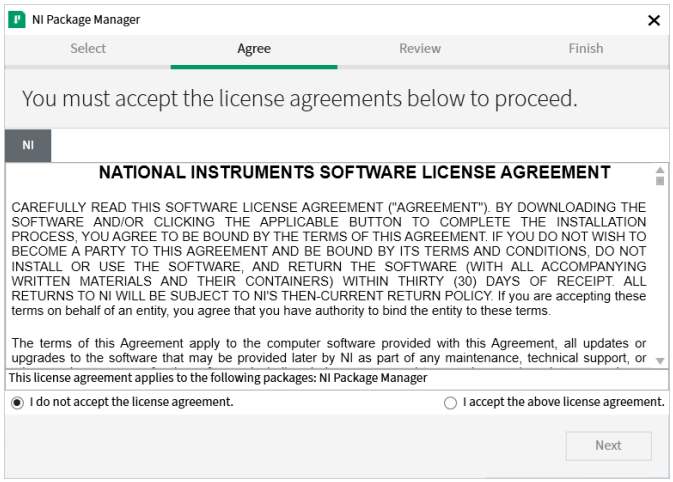
离线（Windows 10+）

右键单击下载的 iso 文件，然后选择“mount”。从已安装的 iso 运行“install.exe’’。如果出现 Windows 安全提示，请单击“是”。



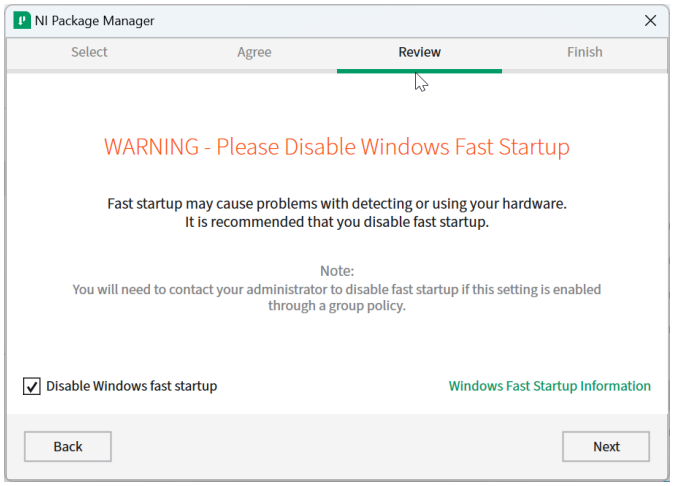
备注： Other installed programs may associate with iso files and the *mount* option may not appear. If that software does not give the option to mount or extract the iso file, then install 7-Zip and use that to extract the iso.

NI Package Manager 许可证



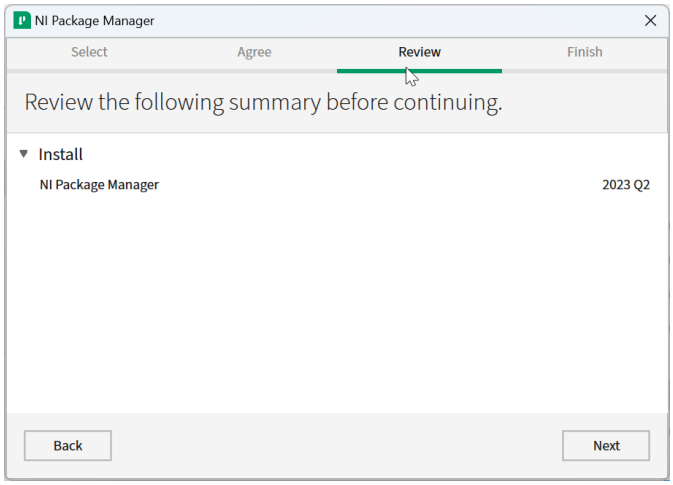
如果您看到此屏幕，请单击“下一步”。该屏幕确认您同意 NI Package Manager 许可协议。

禁用 Windows 快速启动



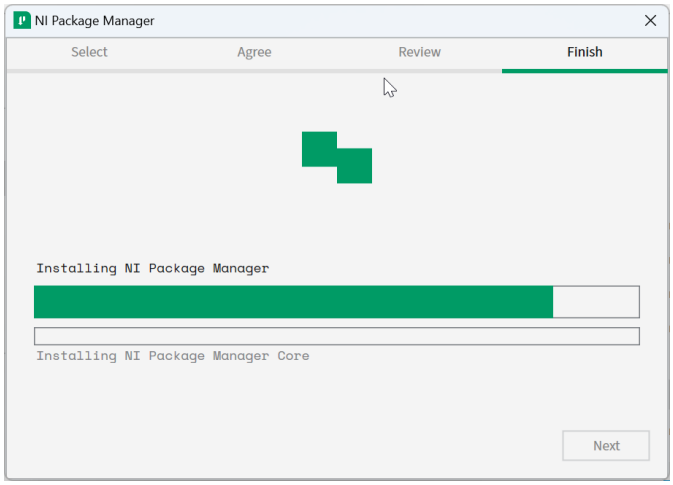
建议保持该屏幕不变，因为 Windows 快速启动会导致镜像 roboRIO 所需的 NI 驱动程序出现问题。继续并单击“下一步”。

NI Package Manager 评估



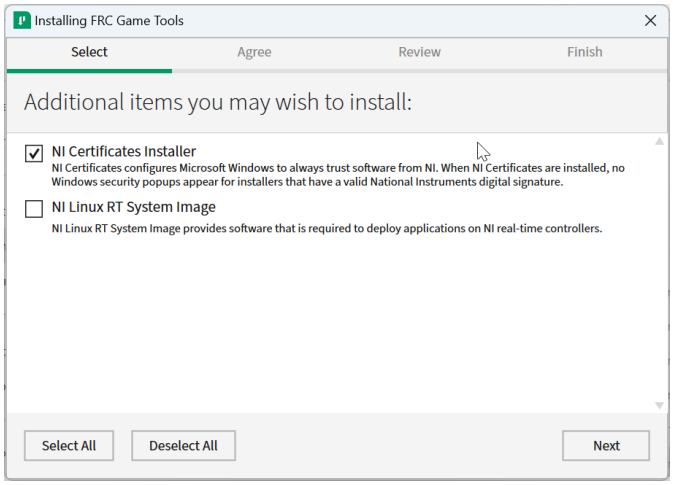
如果您看到此屏幕，请单击“下一步”。

NI Package Manager 安装



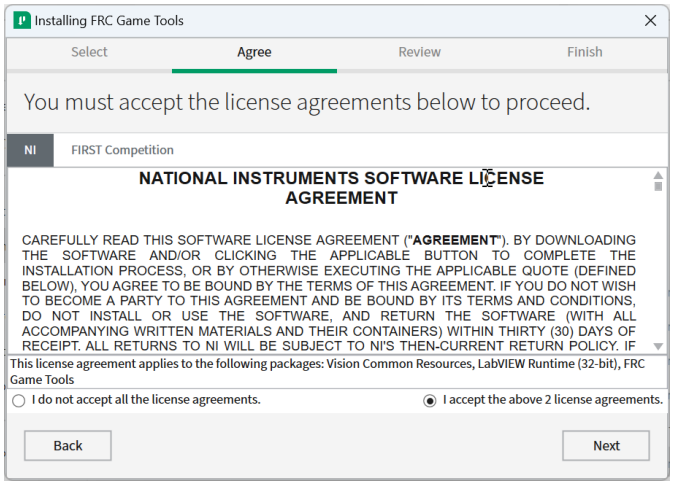
NI Package Manager 的安装进度将在此窗口中跟踪。

附加软件



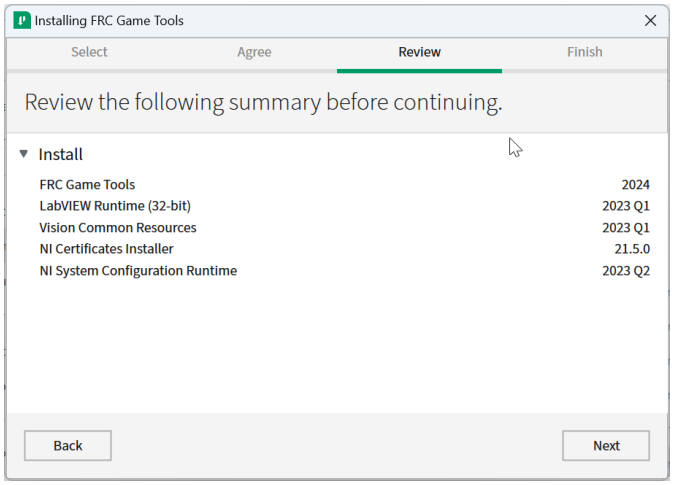
如果您看到此屏幕，请单击“下一步”。

许可协议



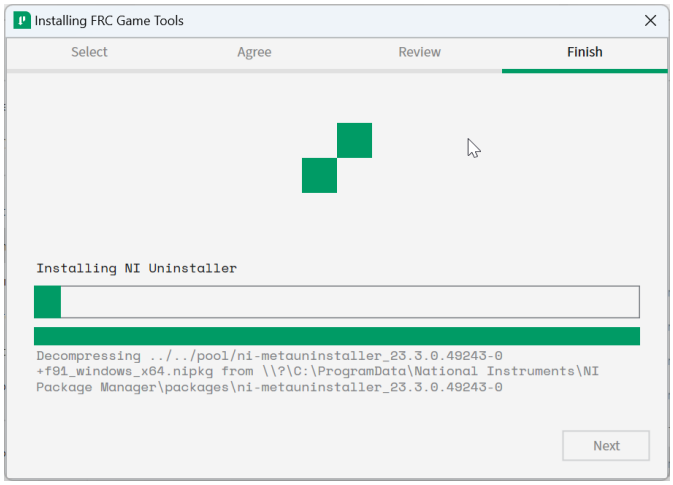
选择“我接受…”，然后单击“下一步”。

总结



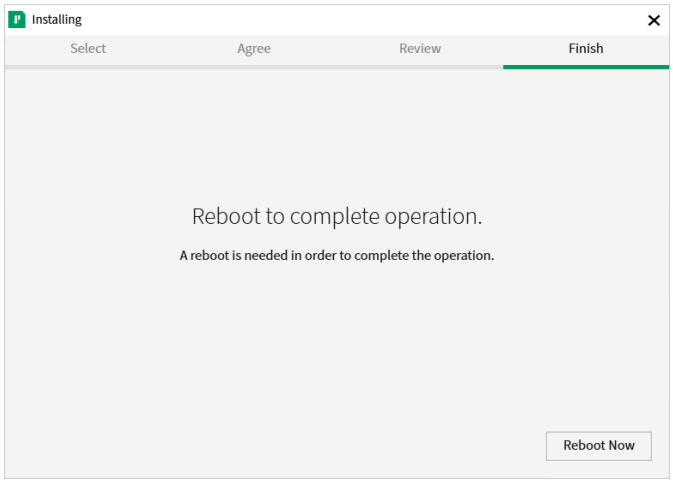
点击“下一步”。

详细进度



该屏幕展示了安装过程，请继续，然后在完成时按：*Next*。

3.3.4 重新启动以完成安装



如果出现提示，请在关闭所有打开的程序后选择“立即重启”。

3.4 WPILib 安装指南

This guide is intended for Java and C++ teams. LabVIEW teams can skip to *安装 LabVIEW for FRC（仅 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 先行准备

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.

警告： 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

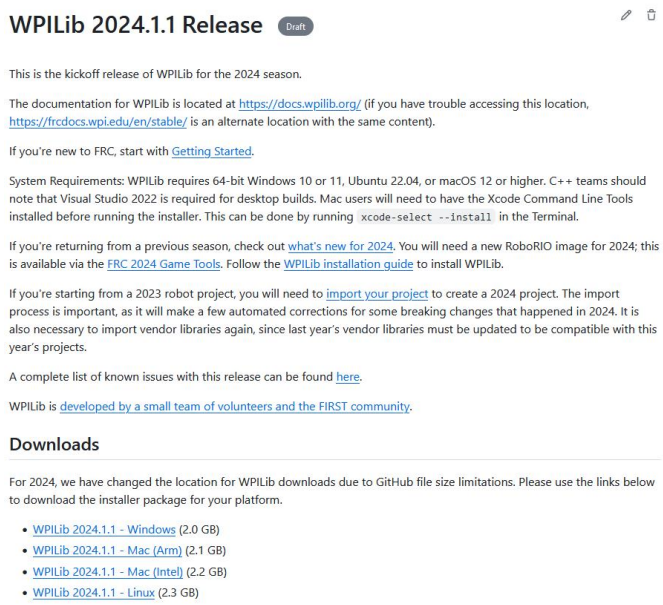
WPILib 2024.3.2 Release - March 14, 2024 Downloads

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)

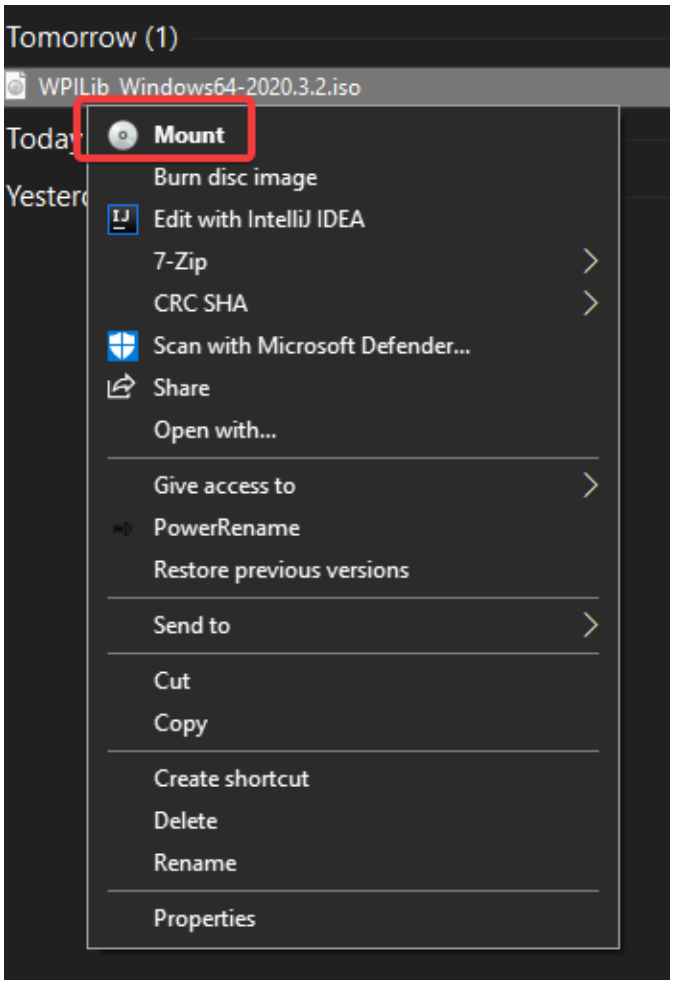
Then click on the correct binary for your OS and architecture to begin the download.

3.4.3 解压缩安装程序

当您下载 WPILib 安装程序时，对于 Windows，它作为磁盘映像文件.iso 分发；对于 Linux，它作为.tar.gz 分发；对于 MacOS，它作为 DMG 分发。

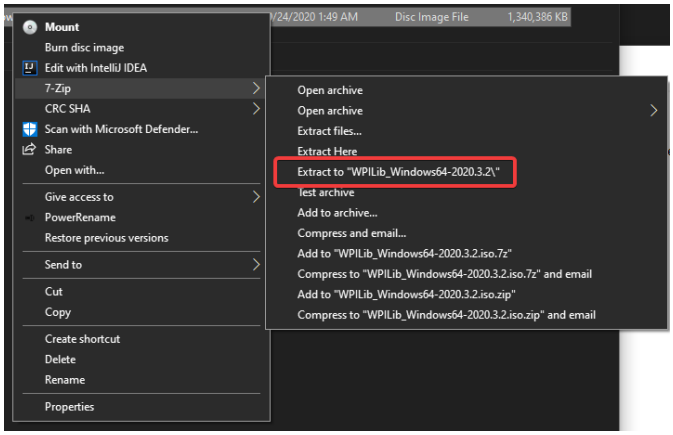
Windows 10+

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



备注: 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`.

备注： 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.

---

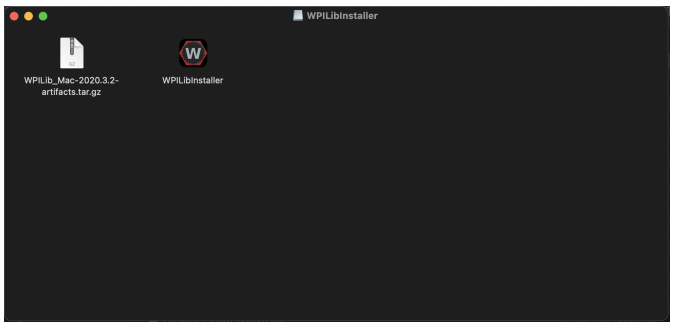
macOS

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.

重要： 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.

---

macOS 用户可以双击下载的 "DMG''，然后选择 "WPILibInstaller'' 以启动应用程序。



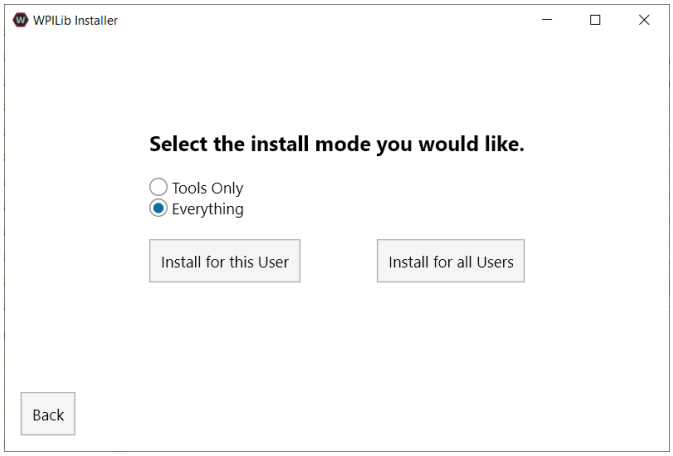
Linux

Linux 用户应解压缩下载的.tar.gz，然后启动 WPILibInstaller。Ubuntu 将文件浏览器中的可执行文件视为共享库，因此双击将不会运行它们。在终端中运行以下命令，而不是将 "<version>" 替换为要安装的版本。

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

3.4.4 运行安装程序

打开安装程序后，将显示以下屏幕。继续并按：Start。



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

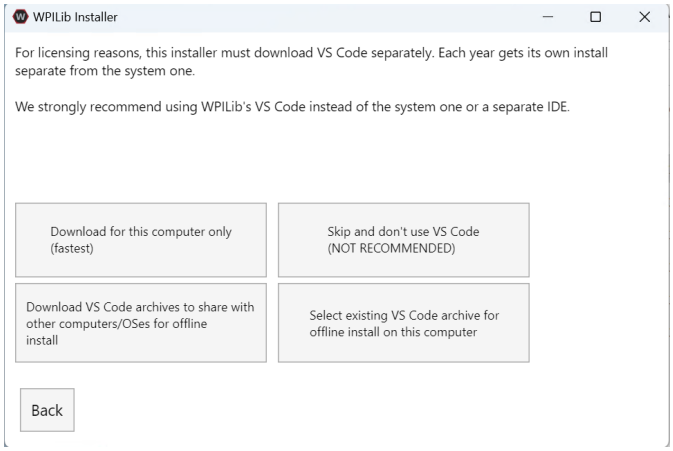
- *Tools Only* installs just the WPILib tools (Pathweaver, Shuffleboard, RobotBuilder, SysId, Glass, and OutlineViewer) and JDK.
- *Everything* installs the full development environment (VS Code, extensions, all dependencies), WPILib tools, and JDK.

您会注意到两个按钮，“为此用户安装”和“为所有用户安装”。为此用户安装仅将其安装在当前用户帐户上，并且不需要管理员特权。然而，为所有用户安装安装工具，所有系统帐户和将需要管理员权限。为所有用户安装不是 macOS 和 Linux 的选项。

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

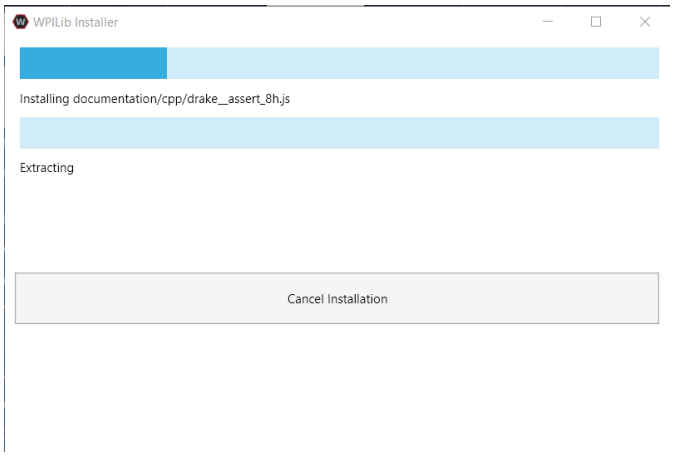
选择适合您的选项，然后将显示以下安装界面。

下一个界面涉及下载 VS Code。不幸的是，由于许可原因，VS Code 无法与安装程序捆绑在一起。

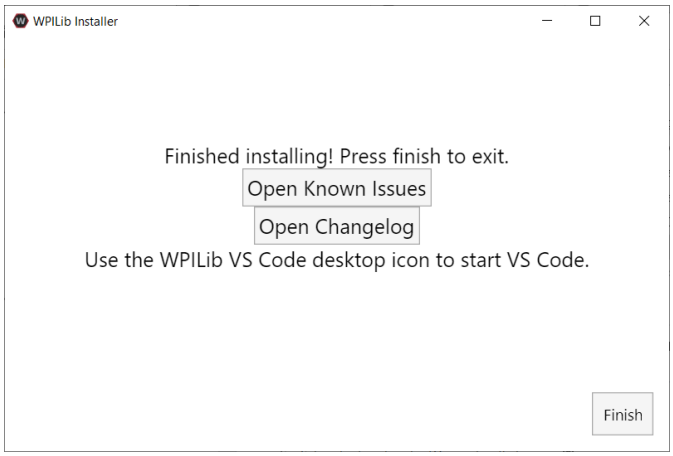


- Download for this computer only
  - 这仅下载当前平台的 VS Code，这也是最小的下载。
- Skip and don’t use VS Code
  - 跳过安装 VS Code。对于高级安装或配置很有用。通常不推荐。
- Select existing VS Code archive for offline install on this computer
  - 选择此选项将弹出提示，允许您选择安装程序先前已下载的 VS Code 压缩文件。该选项不是让您在机器上选择一个已安装 VS 代码的副本。
- Create VS Code archives to share with other computers/OSes for offline install
  - 此选项为所有平台下载并保存 VS Code 的副本，这对于共享安装程序的副本很有用。

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.



安装完成后，将显示完成的界面。



重要： 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!

恭喜，您的计算机上现已安装 WPILib 开发环境和工具！按完成退出安装程序。

3.4.5 安装后

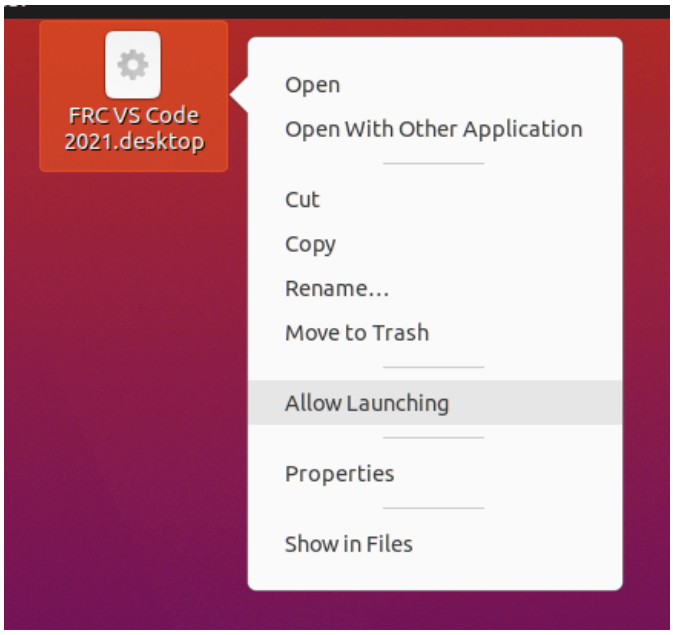
某些操作系统需要某些最终操作才能完成安装。

macOS

安装后，安装程序将打开 WPILib VS Code 文件夹。将 VS Code 应用程序拖到扩展坞上。从桌面弹出 WPILibInstaller 映像。

Linux

某些版本的 Linux（例如 Ubuntu 20.04）要求您赋予桌面快捷方式启动功能。右键单击桌面图标，然后选择允许启动。

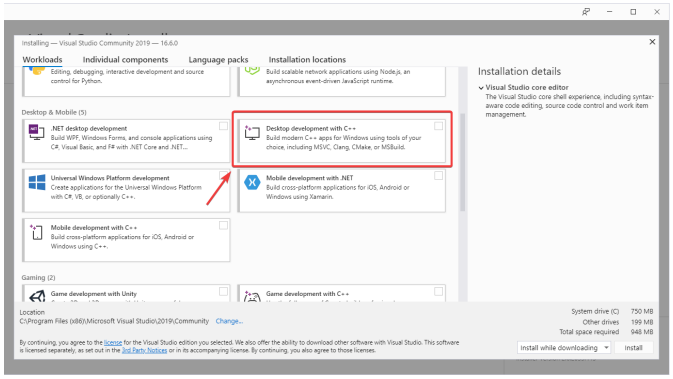


备注： Installing desktop tools and rebooting will create a folder on the desktop called `YYYY WPILib Tools`, where `YYYY` is the current year. Desktop tool shortcuts are not available on Linux and macOS.

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 安装了什么？

脱机安装程序将安装以下组件：

- ** Visual Studio 代码 **-受支持的 IDE，适用于 2019 及更高版本的机器人代码开发。即使您的计算机上已经安装了 VS Code，脱机安装程序也会为 WPILib 开发设置单独的 VS Code 副本。这样做是因为，如果您将 VS Code 用于其他项目，则使 WPILib 安装程序起作用的某些设置可能会破坏现有的工作流程。
- ** C ++ 编译器 **-用于为 roboRIO 构建 C ++ 代码的工具链
- ** Gradle **-用于构建/部署 C ++ 或 Java 机器人代码的 Gradle 的特定版本
- ** Java JDK / JRE **-Java JDK / JRE 的特定版本，用于构建 Java 机器人代码并运行任何基于 Java 的工具（Dashboards 等）。它与任何现有的 JDK 安装并存，并且不会覆盖 JAVA_HOME 变量
- **WPILib Tools** - SmartDashboard, Shuffleboard, RobotBuilder, OutlineViewer, PathWeaver, Glass, SysId, Data Log Tool, roboRIO Team Number Setter, AdvantageScope
- ** WPILib 依赖项 **-OpenCV 等
- **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 卸载

WPILib 旨在在不同的年份安装到不同的文件夹，因此在安装今年的 WPILib 之前不必卸载以前的版本。但是，如果需要，可以使用以下说明卸载 WPILib。

Windows 系统

1. Delete the appropriate wpilib folder (`c:\Users\Public\wpilib\YYYY` where YYYY is the year to uninstall)
2. 删除 “C：Users Public Public Desktop’’ 中的桌面图标

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 故障排除

如果安装程序失败，请在安装程序库中打开反馈问题。‘此处 <https://github.com/wpilibsuite/wpilibinstaller-avalonia>‘__ 提供链接。安装程序在错误时应会报错，请在您的问题说明中提供有关错误原因的报错信息。

3.5 Python Installation Guide

This guide is intended for Python teams. Java and C++ teams can skip to *WPILib 安装指南*. LabVIEW teams can skip to *安装 LabVIEW for FRC（仅 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

备注： 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.

警告： 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

备注： 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

备注： 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.

> **警告:** It may take a very long time to install!

> **警告:** 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.

参见:

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.

参见:

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

---

备注: 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 下一步

恭喜你！您已经完成了第 2 步，现在应该已经可以使用软件开发环境！本教程的第 3 步介绍如何更新硬件，以便对其进行编程，而第 4 步则展示了在 VS Code 集成开发环境（IDE）中对机器人进行编程。有关更多信息，您可以通读:ref:‘VS Code section <docs/software/vscode-overview/index:VS Code Overview>‘以熟悉 IDE。

建议阅读的特定文章是：

- *Visual Studio Code 基础*
- *Visual Studio Code 中的 WPILib 命令*
- 创建机器人程序
- 构建和部署机器人代码
- 安装第三方库

此外，您可能需要进行适用于队伍机器人的额外配置。请利用搜索功能查找必要的文档。

备注： 使用第三方 CAN 电机控制器的团队请务必阅读:ref: *安装第三方库 <docs/software/vscode-overview/3rd-party-libraries:3rd Party Libraries>* 一文，因为为这些设备编写代码需要额外的步骤。

---

4 第 3 步：准备你的机器人

4.1 Imaging your roboRIO 2

备注： 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.

重要： 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.

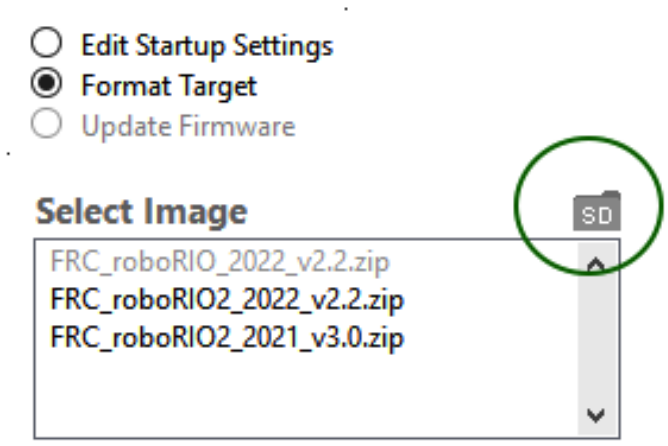
警告: Before imaging your roboRIO, you must have completed installation of the *FRC Game Tools*. You also must have the roboRIO power properly wired to the CTRE Power Distribution Panel or REV Power Distribution Hub. Make sure the power wires to the roboRIO are secure and that the connector is secure firmly 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.

备注: 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*

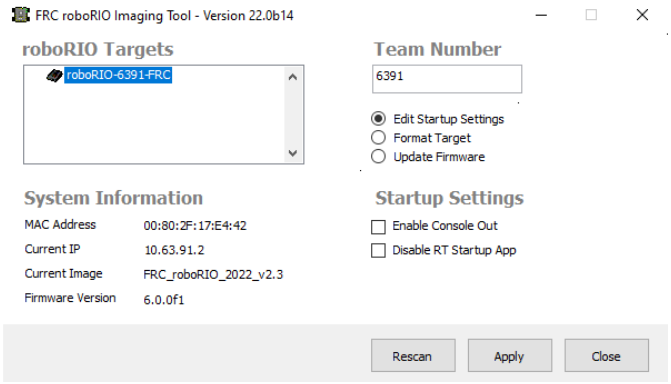
Writing the image with Raspberry Pi Imager

- 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*

警告： 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

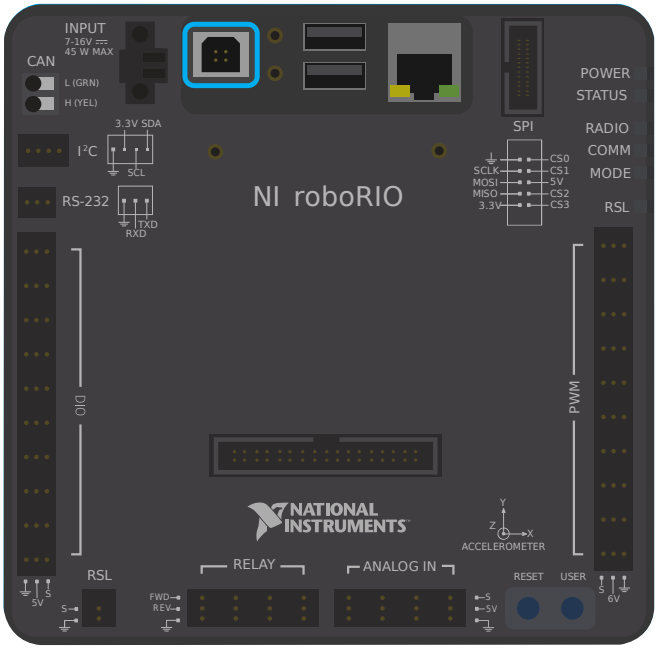
警告： 在为你的 roboRIO 镜像之前，，你必须完成:doc:‘FRC Game Tools</docs/zero-to-robot/step-2/frc-game-tools>‘的安装. 你也必须将 roboRIO 电源正确地连接到配电板上。确保 roboRIO 的电源线连接牢固，并且接口牢固地固定在 roboRIO 上（共检查 4 个螺钉）。

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

4.2.1 配置 roboRIO

roboRIO 镜像工具将被用于升级 roboRIO 的固件。

USB 连接



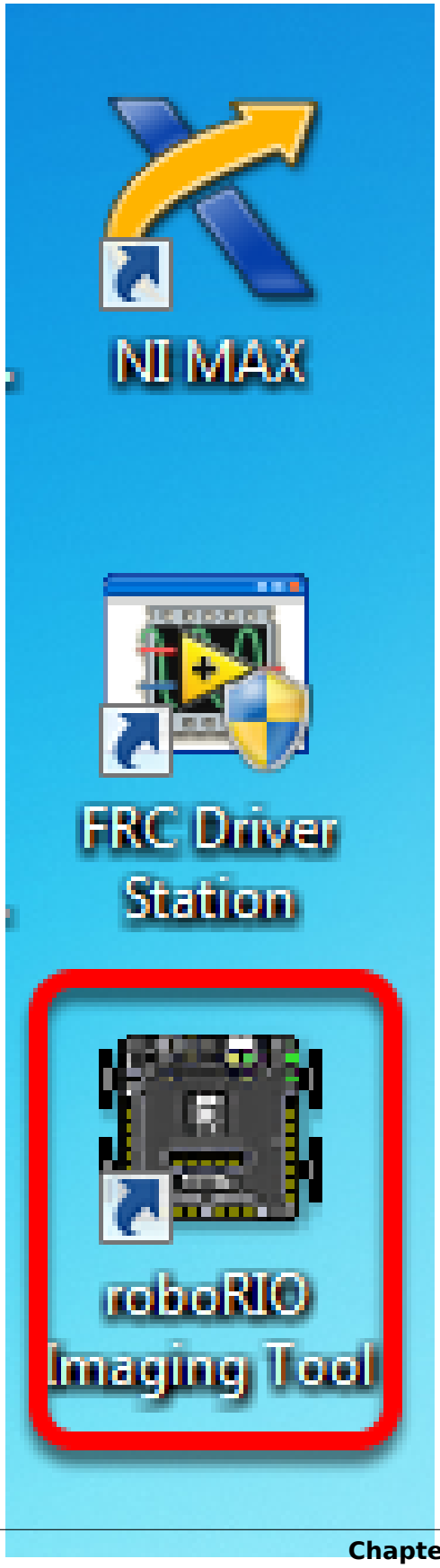
将 USB 电缆从 roboRIO USB 设备端口连接到个人电脑。这需要 USB A 型公头 (标准 PC 端) 到 B 型公头 (形似切去 2 个角的正方形) 电缆。常见的有打印机 USB 电缆等。

备注： roboRIO 应只能通过 USB 连接进行镜像写入。不建议在尝试写入影像时使用以太网连接。

驱动安装

设备驱动程序应自动开始安装。如果您在屏幕的右下方看到一个“新设备”弹窗，等待至驱动程序安装完成后再继续。

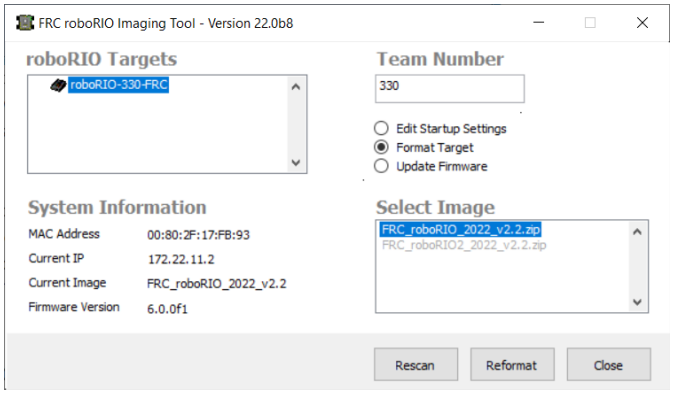
4.2.2 启动镜像工具



roboRIO 映像工具和最新映像已随 NI FRC | reg |一起安装。游戏工具。双击桌面上的快捷方式启动映像工具。如果您在对 roboRIO 进行成像时遇到困难，则可能需要尝试右键单击该图标，然后选择以管理员身份运行。

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

4.2.3 roboRIO 镜像工具

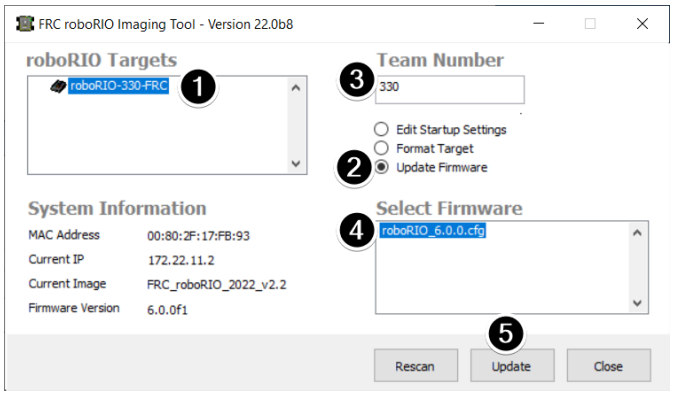


启动后，roboRIO 镜像工具将扫描可用的 roboRIO，并在左上方的框中显示所有找到的 roboRIO。左下方的框将显示当前所选 roboRIO 的信息和设置。右侧面板包含了用于修改 roboRIO 设置的控件：

- **编辑启动设置**-当您要配置 roboRIO 的启动设置（右窗格中的设置）而不使用 roboRIO 的映像时，使用此选项。
- **格式化目标**-当您要在 roboRIO 上加载新图像（或重新刷新现有图像）时使用此选项。这是最常见的选择。
- **更新固件**-此选项用于更新 roboRIO 固件。在这个季节，映像工具将需要 roboRIO 固件版本为 5.0 或更高版本。

更新固件

警告： 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.



roboRIO 固件必须至少为 v5.0 才能使用 2019 或更高版本的映像。如果 roboRIO 的版本至少为 5.0（如映像工具的左下方所示），则无需更新。

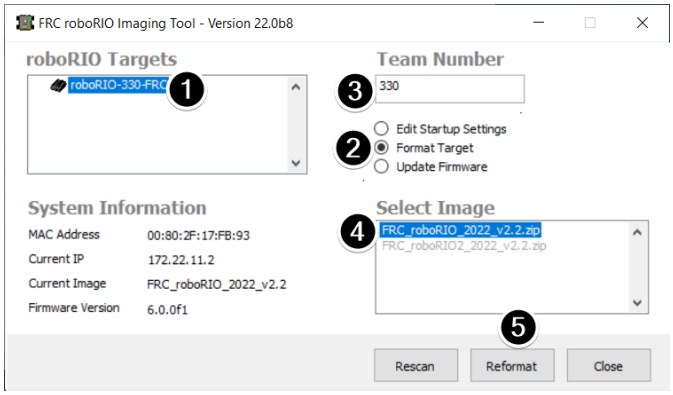
备注： roboRIO firmware has had different version numbering schemes over the years. It isn't necessary to update the firmware if it has version 5, 6, 8, 22.5, 23.5 or variations of those version numbers (e.g. 8.8.0f0 is a variation of 8). The firmware is only utilized in *safe mode*, it is not used in normal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

要更新 roboRIO 固件：

1. 请确保在左上方窗格中选择了您需要的 roboRIO。
2. Select *Update Firmware* in the top right pane
3. Enter a team number in the *Team Number* box
4. 在右下方选择最新的固件文件
5. Click the *Update* button

4.2.4 镜像 roboRIO

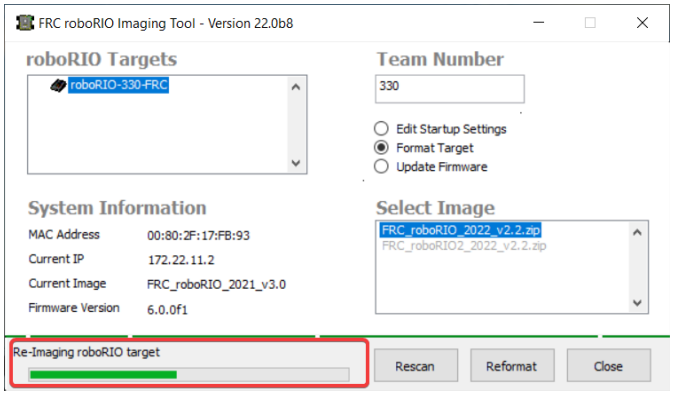
警告： The roboRIO image is different then the firmware, and must be updated yearly.



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

1. 确保在左上方窗格中选择了 roboRIO
2. Select *Format Target* in the right pane
3. Enter a team number in the *Team Number* box
4. 在框中选择最新的镜像版本。
5. Click *Reformat* to begin the imaging process.

4.2.5 写入镜像过程



写入镜像过程大约需要 3-10 分钟。窗口左下方的进度条将指示进度。

4.2.6 写入镜像完成

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 故障排除

如果您无法对 roboRIO 进行镜像，则故障排除步骤包括：

- 尝试以管理员身份运行 roboRIO 镜像工具，方法是右键单击桌面图标以启动它。
- 尝试使用网络浏览器在 http://172.22.11.2/ 访问 roboRIO 网页，并验证 NI 网络适配器是否出现在控制面板的网络适配器列表中。如果未出现，请尝试重新安装 NI FRC Game Tools 或尝试使用其他 PC。
- ：ref：禁用所有其他网络适配器 *<docs/networking/networking-introduction/roborio-network-troubleshooting:Disabling Network Adapters>*
- 确保您的防火墙已关闭。
- 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*
- 按住复位按钮至少 5 秒钟，尝试将 roboRIO 引导至安全模式。
- Try a different USB Cable
- 尝试其他电脑
- 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 给你的路由器编程

本指南将向您展示如何使用 FRC®Radio Configuration Utility 软件来配置机器人的无线连接，来在 FRC 比赛之外使用。

4.3.1 先行准备

FRC 路由器配置工具需要管理员权限才能配置您电脑上的网络设置。该程序会自动请求必要的权限 (如果您使用非管理员帐户登录计算机，则可能需要密码) 。如果遇到问题，尝试用管理员帐户运行。

从以下链接下载最新的 FRC 路由器配置工具安装程序：

FRC Radio Configuration 24.0.1

FRC Radio Configuration 24.0.1 Israel Version

备注： 此 _IL 版本适用于以色列团队。为了使软件能在以色列使用，其包含一个有受限信道的 OM5PAC 固件版本。

在开始使用此软件之前：

1. 禁用所有其他网络适配器
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.

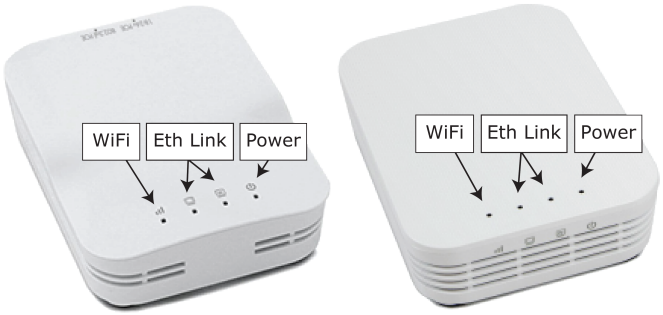
警告： OM5P-AN 和 AC 使用与 D-Link DAP1522 相同的电源插头，但是它们是 12V 路由器。将路由器连接到 VRM 上的 12V 2A 端口 (中心引脚正极) 。

4.3.2 应用须知

默认情况下，the Radio Configuration Utility 程序将对路由器进行编程，以对通过无线接口输出路由器的流量实施 4Mbps 带宽限制。在家庭配置（AP 模式）下，这是总数，而不是每个用户的限制。这意味着将视频流传输到多个用户是不明智的。

该程序已经在 Windows 7、8 和 10 上进行了测试。它可以在其他操作系统上运行，但尚未经过测试。

程控配置



运行时，“路由器配置工具”会通过内置程序将一些配置直接设定到路由器中。这适用于所有模式下的路由器 (包括在比赛时) 。这些配置包括：

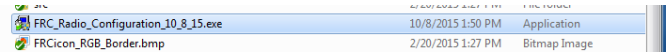
- 将静态 IP 设置为 “10.TE.AM.1’’
- 在有线端设置备用 IP，‘192.168.1.1’’ 以备将来编程
- 桥接有线端口，以便端口互换使用
- 上图中指出了 LED 配置。
- 无线接口输出侧的带宽限制为 4Mb / s（可能禁用以供家庭使用）
- 遵从内部数据包优先的 QoS 规则（影响内部缓冲区且决定如果达到带宽限制则丢弃哪些数据包）。这些规则是：
  - 机器人控制和状态（UDP“1110’’，“1115’’，“1150’’）
  - 机器人 TCP & *NetworkTables* (TCP 1735, 1740)
  - 批量（所有其他流量）。(如果禁用带宽限制，则禁用）
- *DHCP* server enabled. Serves out:
  - 接线侧的 “10.TE.AM.11’’ -“10.TE.AM.111’’
  - 无线侧的 “10.TE.AM.138’’ -“10.TE.AM.237’’
  - 子网掩码 “255.255.255.0’’
  - 广播地址 “10.TE.AM.255’’
- DNS server enabled. DNS server IP and domain suffix (`.lan`) are served as part of the DHCP.

仅在家中：

- SSID 可能在团队编号后附加一个 “机器人名称”，以区分多个网络。
- 可以启用防火墙选项以模仿现场防火墙规则（可以在游戏手册中找到开放的端口）

警告： 无法手动修改配置。

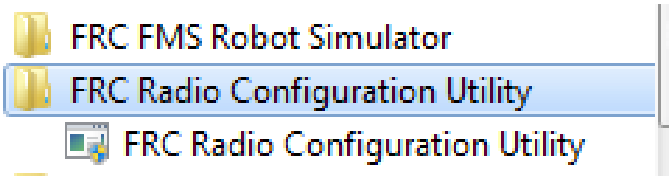
4.3.3 安装此软件



双击“FRC_Radio_Configuration_VERSION.exe’’启动安装程序。按照提示完成安装。

如果尚未安装 Npcap，则部分安装提示将包括安装 Npcap。Npcap 安装程序包含许多用于配置安装的复选框。您应该将这些选项保留为默认值。

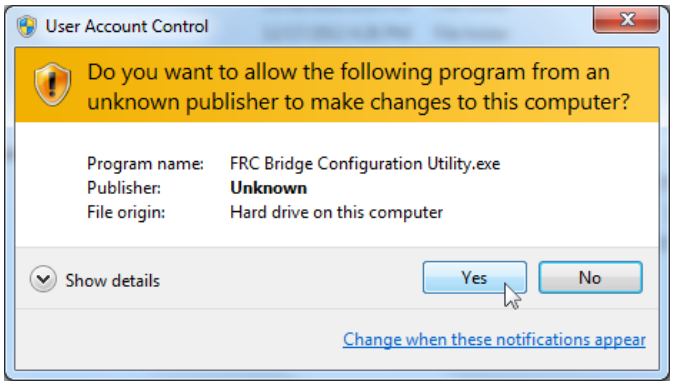
4.3.4 启动软件



使用“开始”菜单或桌面快捷方式启动程序。

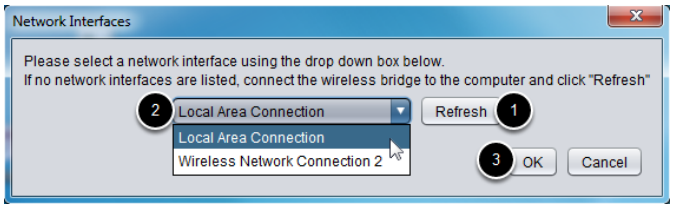
备注： 如果需要找到该程序，则将其安装到“C：Program Files（x86）FRC Radio Configuration Utility’’中。对于 32 位计算机，路径为“C：Program Files FRC Radio Configuration Utility’’

4.3.5 如果出现弹窗提示，请允许程序对系统进行更改



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

4.3.6 选择网络接口



使用弹出窗口选择该程序将用于与路由器通信的以太网接口。在 Windows 计算机上，以太网接口通常称为“本地连接”。该程序无法对通过无线连接的路由器进行编程

1. 如果未列出任何以太网接口，请单击“刷新”以重新扫描可用接口。
2. 从下拉列表中选择要使用的接口。
3. 单击确定。

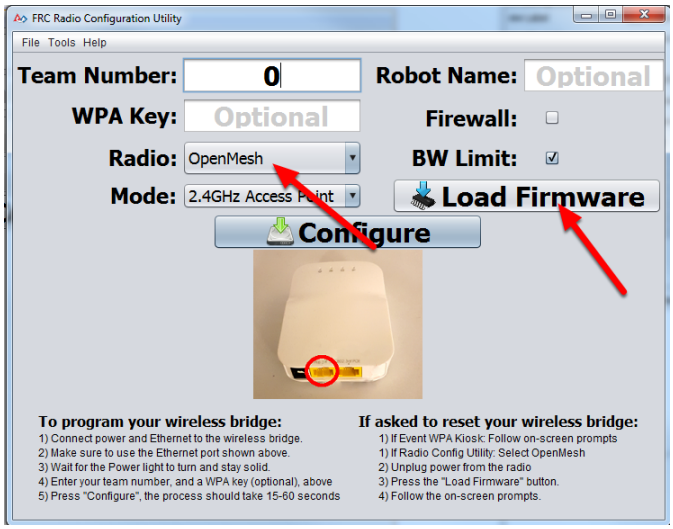
4.3.7 Open Mesh 固件说明

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.

如果不需要更新或重新加载固件，请跳过下一步。

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

4.3.8 将 FRC 固件加载到 Open Mesh Radio



如果您需要加载 FRC 固件（或重置路由器），则可以使用 FRC Radio Configuration Utility 程序进行加载。

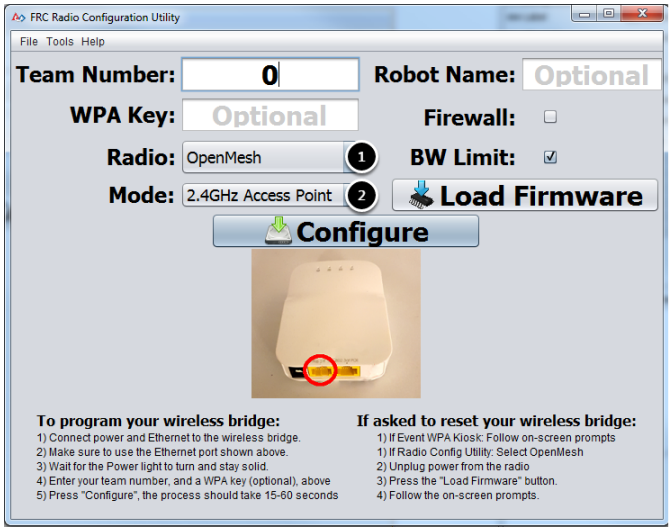
1. 请按照上述说明安装软件，启动程序并选择以太网接口。
2. 确保在“无线电”下拉列表中选择了“Open Mesh radio”。

3. 确保路由器通过以太网连接到 PC。
4. 拔掉路由器的电源。(如果使用 PoE 电缆，也可以将以太网从 PC 上拔下，这没问题)
5. 按“加载固件”按钮
6. 出现提示时，插入路由器电源。该软件应检测路由器、加载固件并在完成时提示您。

警告： 如果看到有关 NPF 名称的错误，请尝试禁用除用于编程路由器的适配器以外的所有适配器。如果仅找到一个适配器，则该工具应尝试使用该适配器。有关更多信息，请参阅禁用网络适配器 。

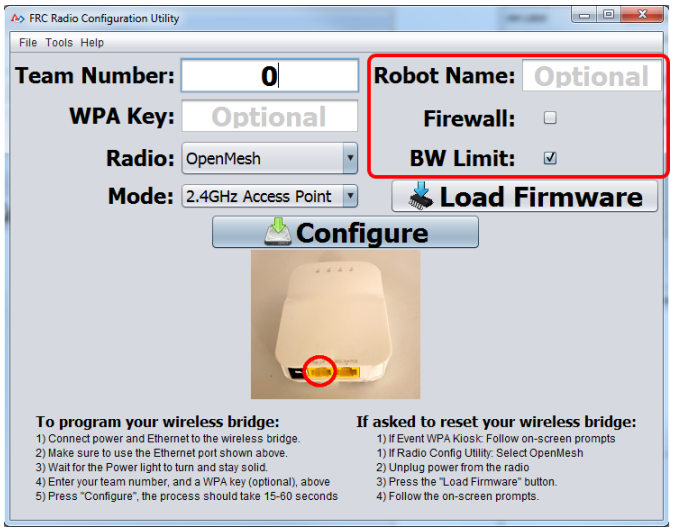
Teams may also see this error with Operating Systems configured for languages other than US English. If you experience issues loading firmware or programming on a foreign language OS, try using an English OS, such as on the KOP provided PC or setting the Locale setting to “en_us” as described on this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

4.3.9 选择无线电和操作模式



1. 使用下拉列表选择要配置的无线电。
2. 选择要配置的操作模式。对于大多数情况，默认选择 2.4GHz 接入点就足够了。如果您的计算机支持的话，则建议使用 5GHz AP 模式，因为在许多环境中 5GHz 不太容易拥堵。

4.3.10 选择选项



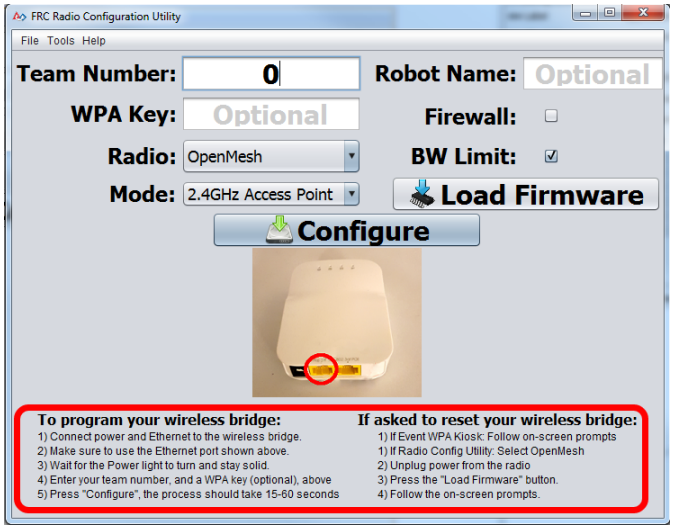
已选项的默认值匹配大多数队伍的使用，但是，您可能希望针对特定方案自定义这些选项：

1. 机器名称：这是一个字符串，附加到路由器使用的 SSID。这使您可以拥有多个具有相同队伍编号的网络，并且仍然能够区分它们。
2. 防火墙：如果选中此框，则将无线电防火墙配置为尝试模仿 FRC 波段上存在的防火墙的端口阻碍。有关开放端口的列表，请参阅 FRC 游戏手册。
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.

备注： 防火墙和带宽限制仅适用于 Open Mesh 路由器。这些选项对 D-Link 无线电无效。

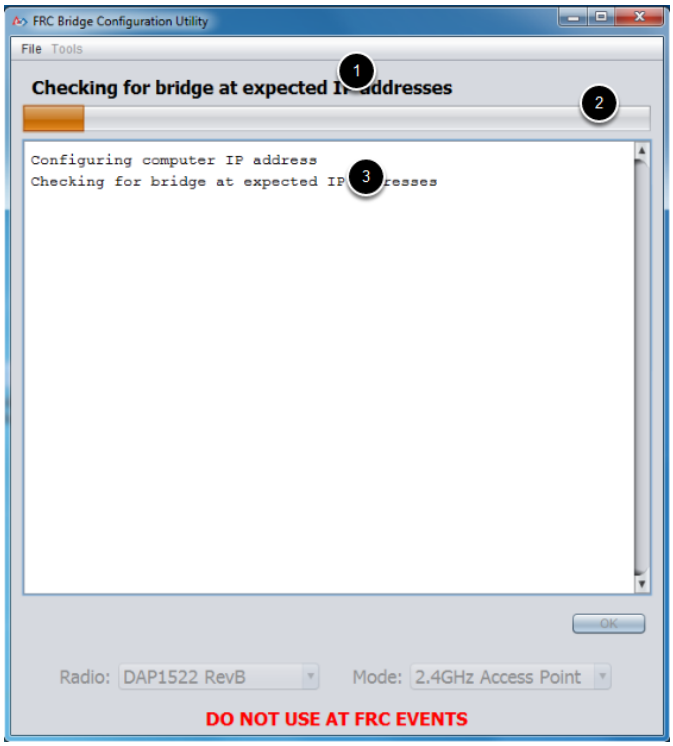
警告： “防火墙”选项将路由器配置为模拟现场防火墙。这意味着启用此选项后，您将无法无线部署代码。这对于模拟比赛中可能存在的端口拥堵很有用。

4.3.11 开始配置过程



按照屏幕上的说明准备无线连接，输入将使用该连接配置的设置，然后开始配置进程。这些屏幕说明会进行更新，以匹配所选的连接类型和操作模式。

4.3.12 配置过程

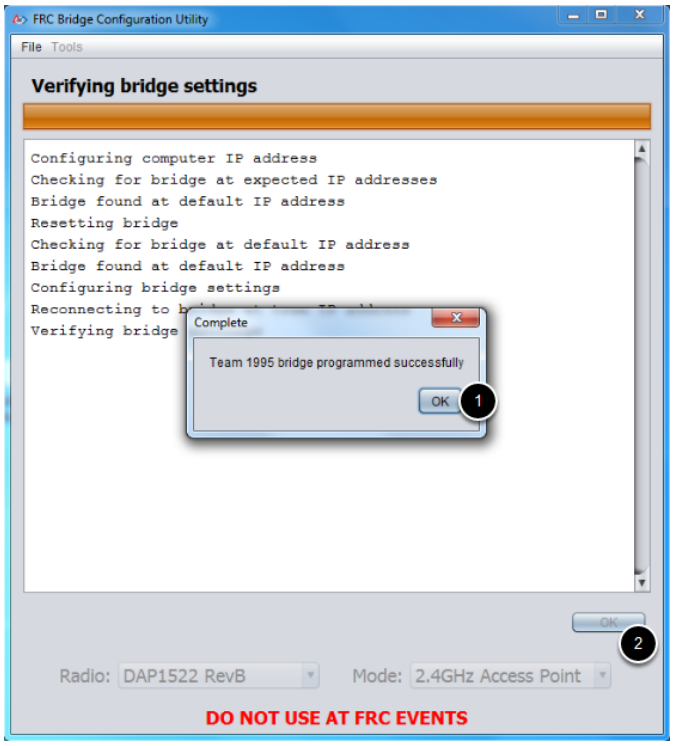


在整个配置过程中，该窗口将指示：

1. 当前正在执行的步骤。
2. 配置过程的总体进度。

3. 到目前为止，所有已执行的步骤。

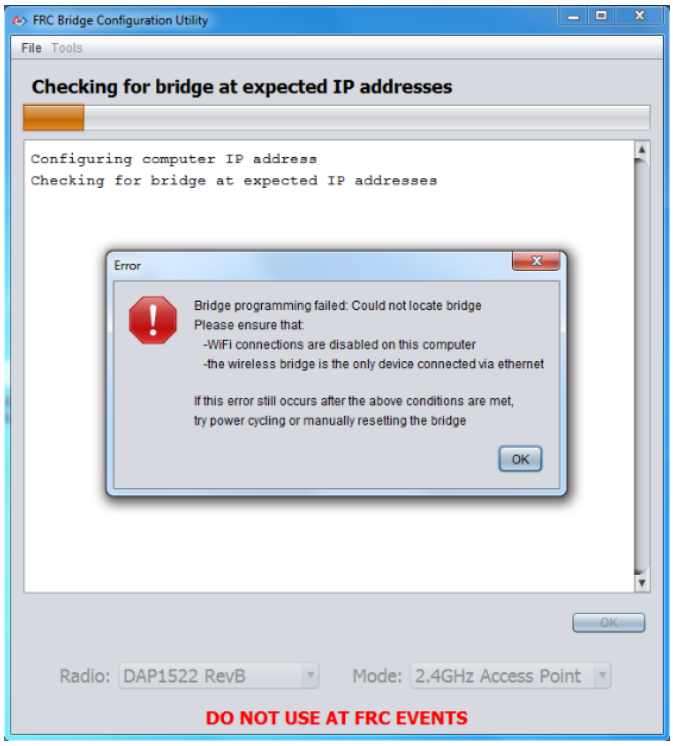
4.3.13 配置完成



配置完成后：

1. 在对话框窗口中按确定。
2. 在主窗口上按 OK 确定以返回到设置屏幕。

4.3.14 配置错误



如果在配置过程中发生错误，请按照错误消息中的说明更正问题。

4.3.15 故障排除

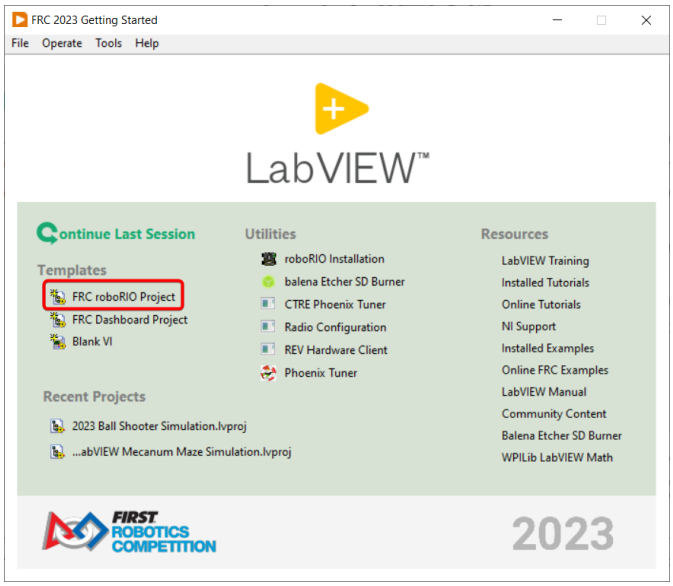
- 禁用所有其他网络适配器。
- 确保等待足够长的时间，使电源指示灯保持常亮 10 秒钟。
- 确保您拥有正确的网络接口，并且下拉列表中只列出了一个接口。
- Make sure your firewall is turned off.
- 直接从计算机插入无线连接，并确保没有其他设备通过以太网连接到计算机。
- 确保以太网已插入最靠近无线桥接器电源插孔的端口。
- If using an Operating System configured for languages other than US English, try using an English OS, such as on the 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 第 4 步：给你的机器人编程

5.1 Creating your Test Drivetrain Program (LabVIEW)

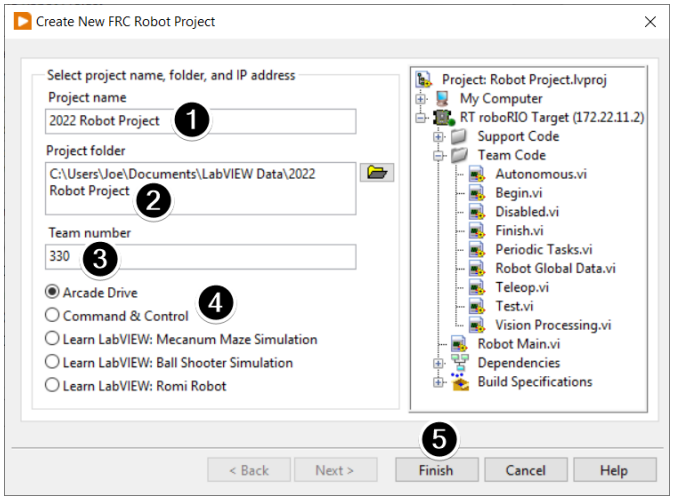
备注： This document covers how to create, build and load a basic FRC® LabVIEW program for a drivetrain onto a roboRIO. Before beginning, make sure that you have installed LabVIEW for FRC and the FRC Game Tools and that you have configured and imaged your roboRIO as described in the *Zero-to-Robot tutorial*.

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.

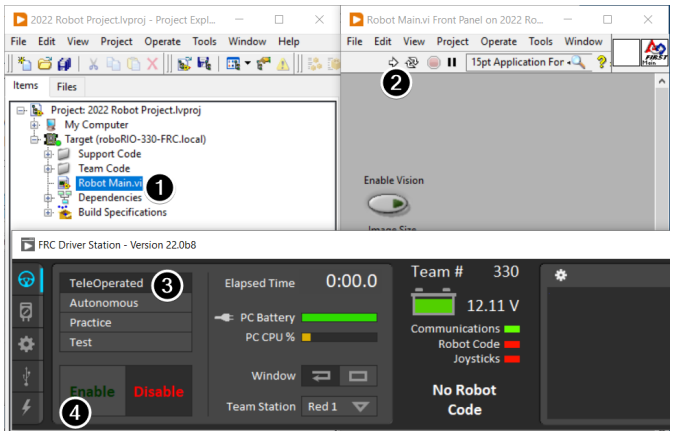
5.1.2 Configuring Project



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

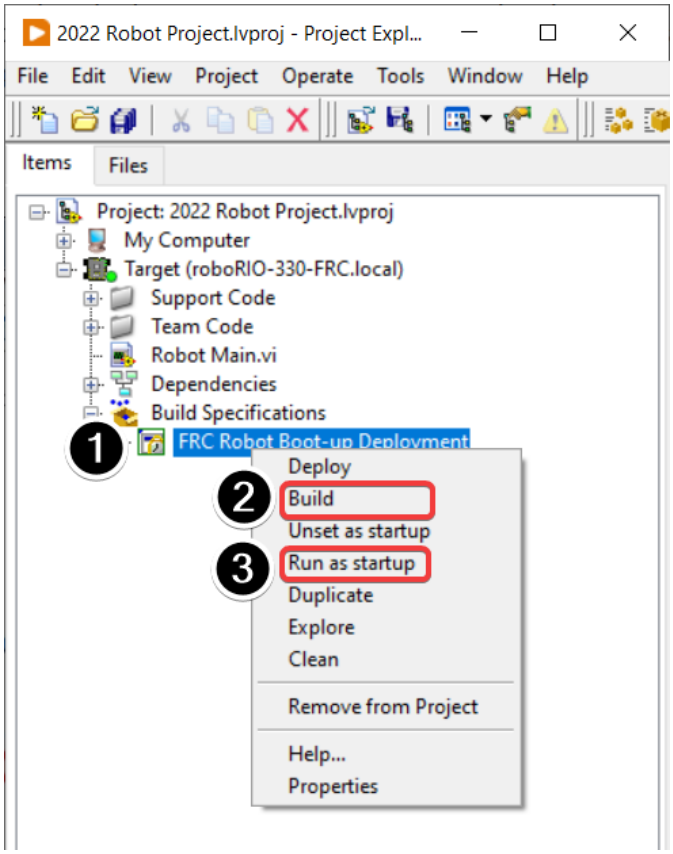


备注： Note that a program deployed in this manner will not remain on the roboRIO after a power cycle. To deploy a program to run every time the roboRIO starts follow the next step, Deploying the program.

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)

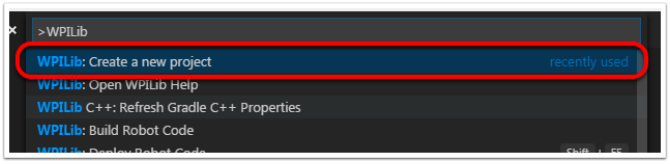
Once everything is installed, we're ready to create a robot program. WPILib comes with several templates for robot programs. Use of these templates is highly recommended for new users; however, advanced users are free to write their own robot code from scratch. This article walks through creating a project 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)*

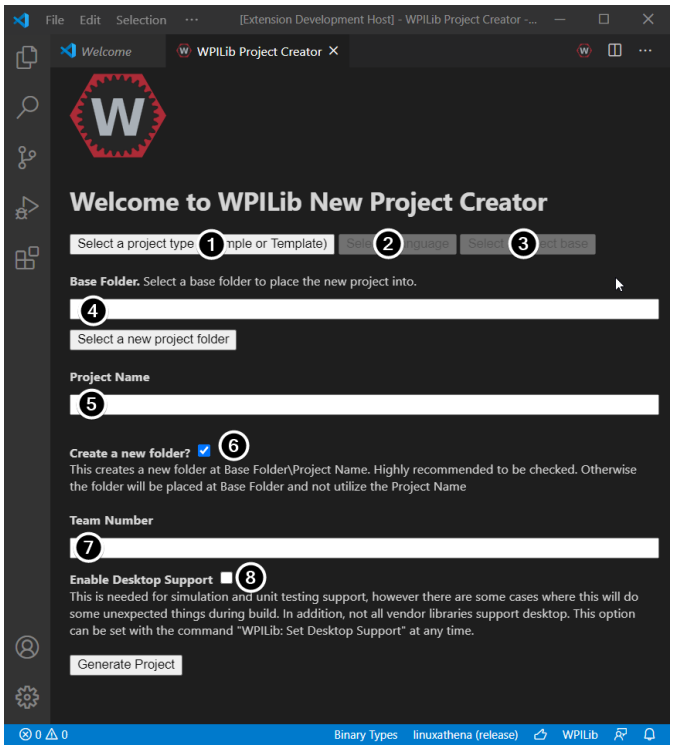
重要: This guide includes code examples that involve vendor hardware for the convenience of the user. In this document, *PWM* refers to the motor controller included in the KOP. The CTRE tab references the Talon FX motor controller (Falcon 500 motor), but usage is similar for TalonSRX and VictorSPX. The 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.

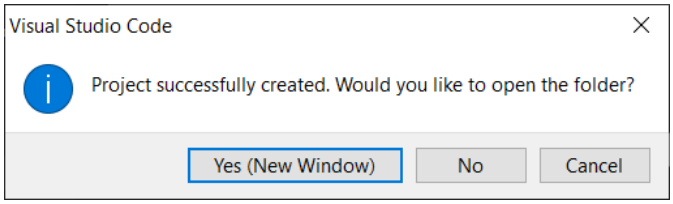
Once all the above have been configured, click "Generate Project" and the robot project will be created.

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

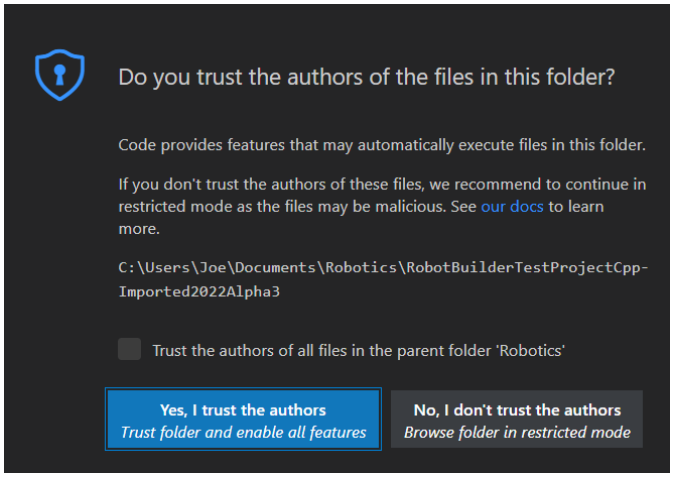
screen.

警告： 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

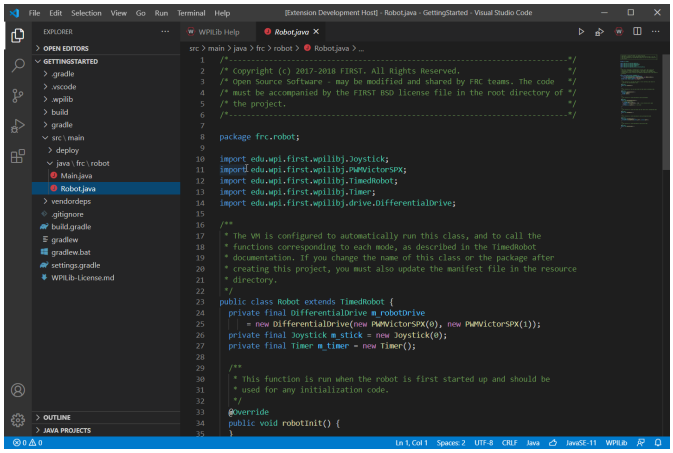


After successfully creating your project, VS Code will give the option of opening the project as shown above. We can choose to do that now or later by typing `Ctrl+K` then `Ctrl+O` (or just `Command+O` on macOS) and select the folder where we saved our project.



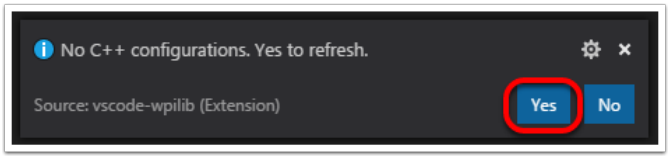
Click *Yes I trust the authors*.

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.

参见:

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).

备注: 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);
```

(续下页)

(接上页)

```
  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++

```cpp
// 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); },
```

(续下页)

(接上页)

```
      [&](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
```

(续下页)

(接上页)

```
            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);
```

(续下页)

(接上页)

```
  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.

备注： 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.

备注： 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. */
```

(续下页)

(接上页)

```
  @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++

```
  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

```
    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() {}

```

(续下页)

(接上页)

```
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

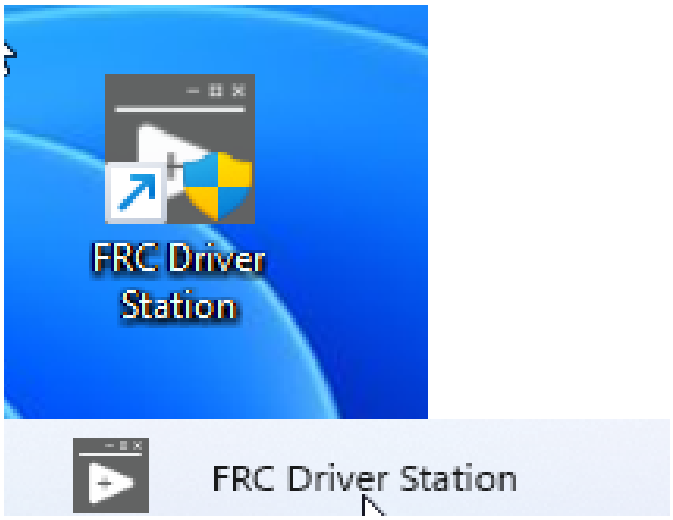
LabVIEW

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.

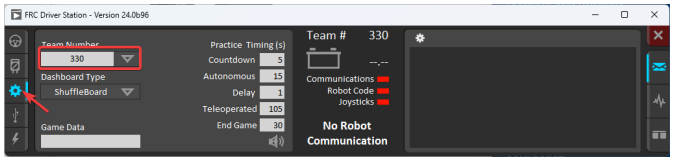
The roboRIO should be powered on and connected to the PC over Ethernet or USB.

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

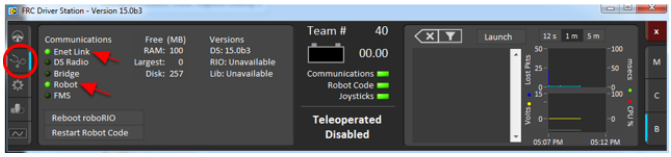


图 1: Tethered

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.

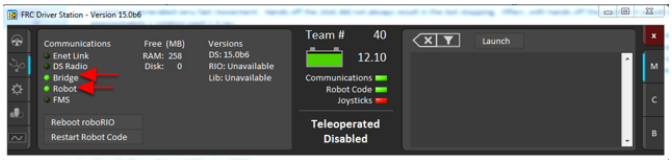
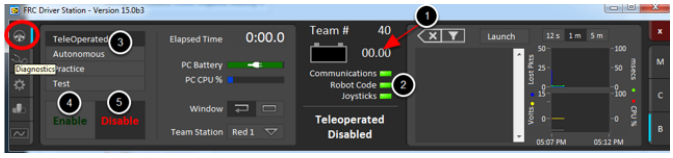


图 2: Wireless

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 硬件组件概述

本文档的目的是简要概述构成 FRC® 控制系统的硬件组件。每个组件都将包含对该组件功能的简短描述以及指向更多文档的链接。

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

6.1 控制系统概述

REV

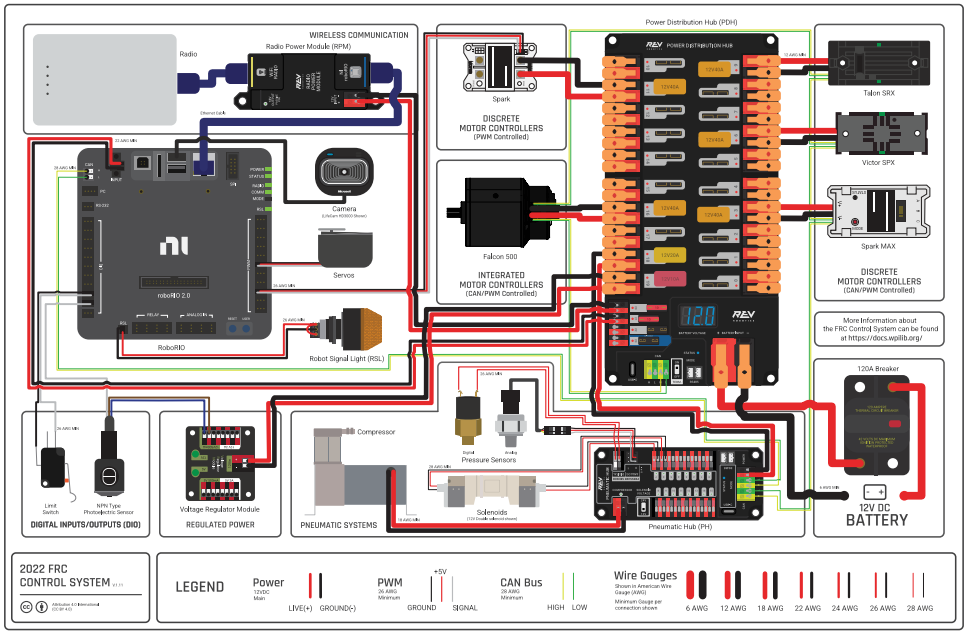


Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

CTRE

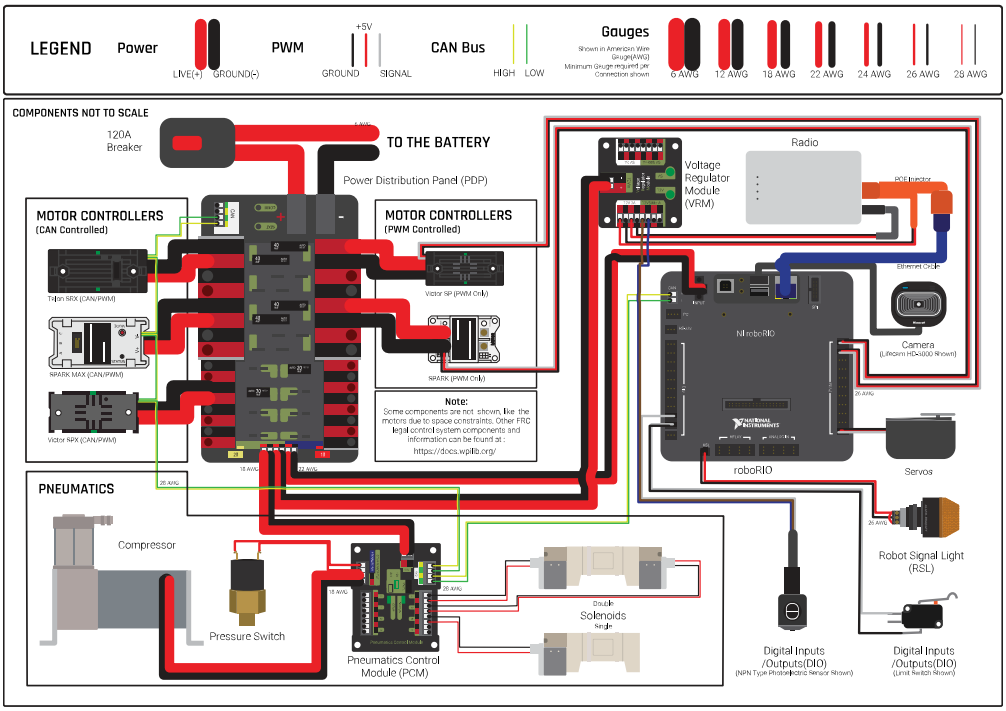
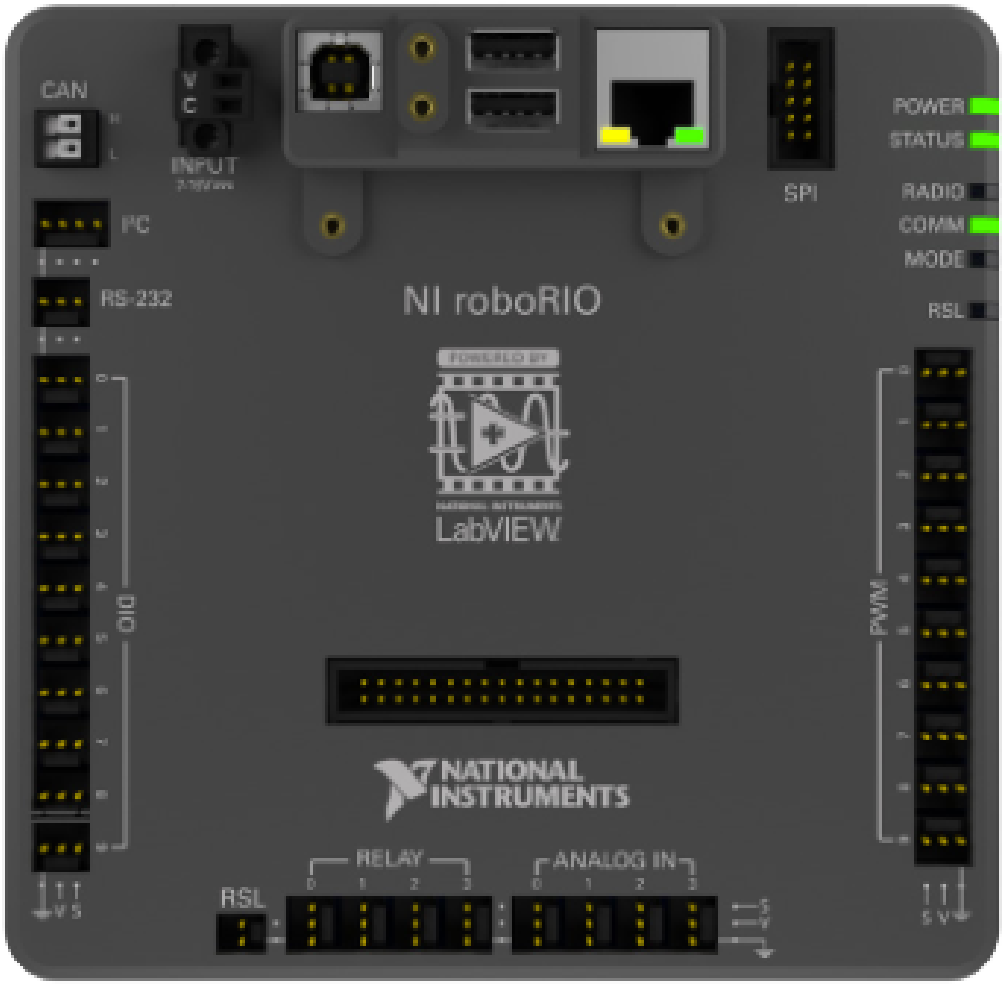


Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

6.2 NI roboRIO

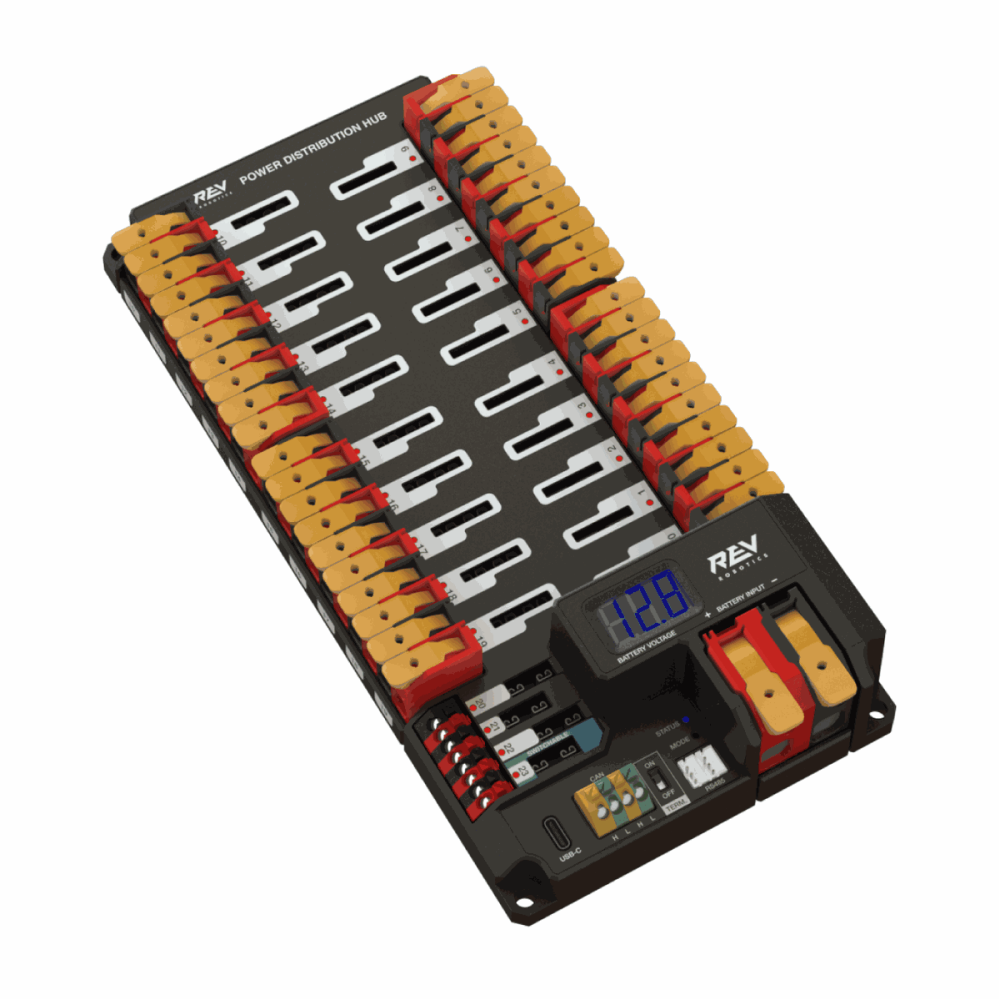


:ref:‘NI-roboRIO <docs/software/roborio-info/roborio-introduction:roboRIO Introduction>‘是用于 FRC 的主要机器人控制器。roboRIO 充当机器人运行团队生成的代码的“大脑”，该代码命令所有其他硬件。

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 OpenMesh OM5P-AN 或 OM5P-AC 无线路由器

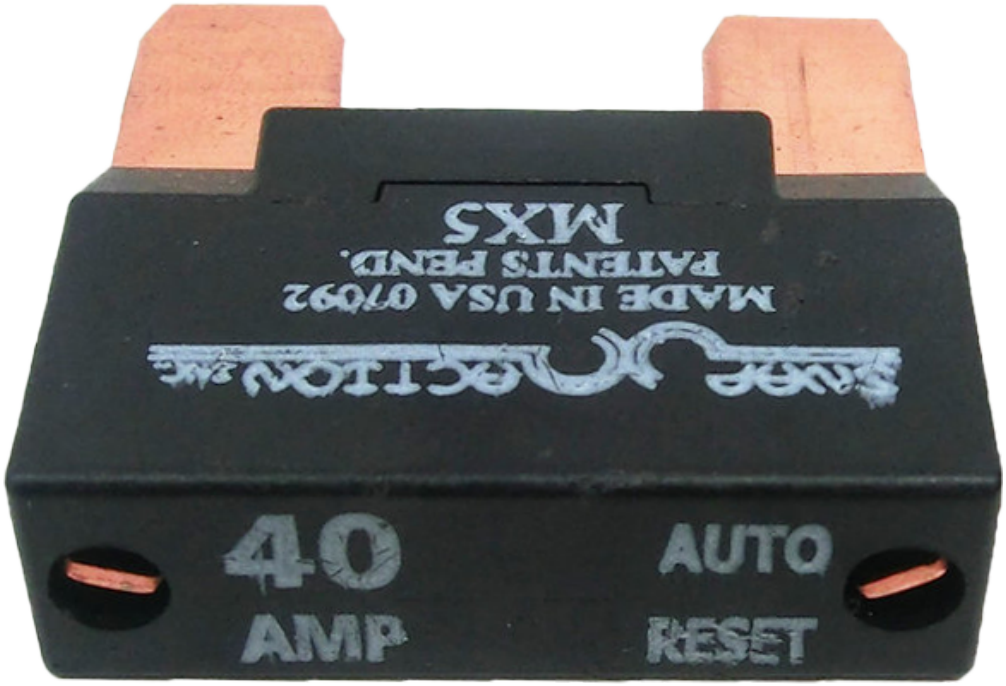
Either the OpenMesh OM5P-AN or OpenMesh OM5P-AC wireless radio is used as the robot radio to provide wireless communication functionality to the robot. The device can be configured as an Access Point for direct connection of a laptop for use at home. It can also be configured as a bridge for use on the 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*.

OM5P-AN‘不再可用于购买 <https://www.firstinspires.org/robotics/frc/blog/radio-silence>‘__。与 OM5P-AN 相比，OM5P-AC 稍重，具有更多的冷却格栅，并且具有粗糙的表面纹理。

6.8 120A 断路器

120A 主断路器在机器人上起着两个作用：主机器人电源开关和用于下游机器人接线和组件的保护装置。120A 断路器已连接至机器人电池和配电板的正极。有关更多信息，请参见 'Cooper Bussmann 18X 系列数据表（PN：185120F）<https://www.mouser.com/datasheet/2/87/BUS_Tns_DS_18X_CIRCUITBREAKER-515519.pdf>'__

6.9 速动断路器



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 机器人电池

FRC 机器人的电源是一个 12V 18Ah 密封铅酸（SLA）电池，能够满足 FRC 机器人的高电流需求。有关更多信息，请参见“机器人电池”页面。<docs/hardware/hardware-basics/robot-battery:Robot Battery Basics>‘

备注： 多个电池部件号可能是合法的，有关完整列表，请参考 ‘FRC 手册 <https://www.firstinspires.org/resource-library/frc/competition-manual-qa-system>‘__。

6.11 机器人信号灯

Allen-Bradley

图 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.

图 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.

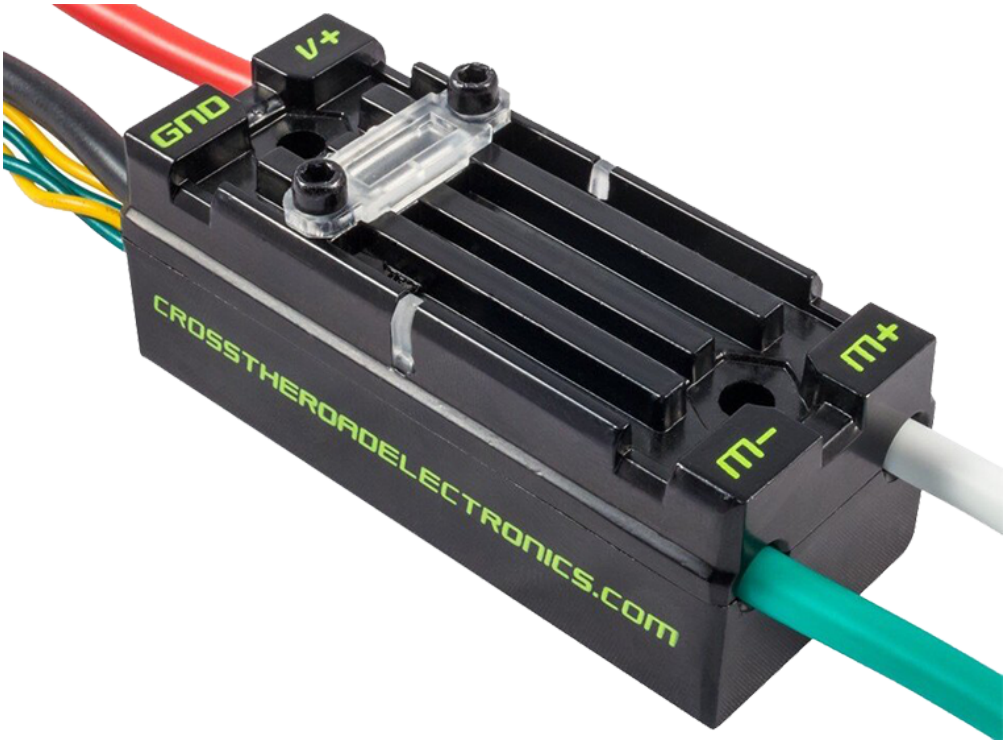
Digital and analog pressure sensor ports are built into the device, increasing the flexibility and feedback functionality of the pneumatic system. The USB-C connection on the Hub works with the REV Hardware Client, allowing users to test pneumatic systems without a need for an additional robot controller.

6.14 电机控制器

There are a variety of different *motor controllers* which work with the FRC Control System and are approved for use. These devices are used to provide variable voltage control of the brushed and brushless DC motors used in FRC. They are listed here in order of usage.

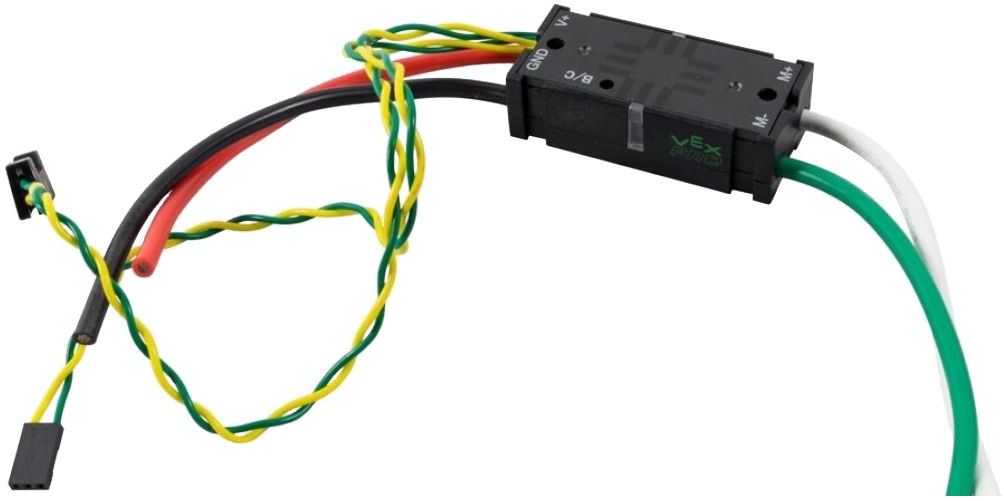
备注： WPILib 不支持第三方 CAN 控制。有关更多信息，请参见 docs / software / can-devices / third-party-devices：第三方 CAN 设备 '上的此部分。

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 Victor 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 SPARK MAX 电机控制器

SPARK MAX 电动机控制器 <https://www.revrobotics.com/rev-11-2158/>‘__ 是 REV Robotics 提供的高级有刷和无刷直流电动机控制器。当使用 CAN 总线或 USB 控制时，SPARK MAX 使用限位开关，编码器和其他传感器（包括 REV NEO 无刷电机的集成编码器）的输入来执行高级控制模式。SPARK MAX 可以通过 PWM，CAN 或 USB 进行控制（仅用于配置/测试）。有关更多信息，请参见 ‘SPARK MAX 用户手册 <https://docs.revrobotics.com/sparkmax/>‘__。

6.14.4 TalonFX 电机控制器

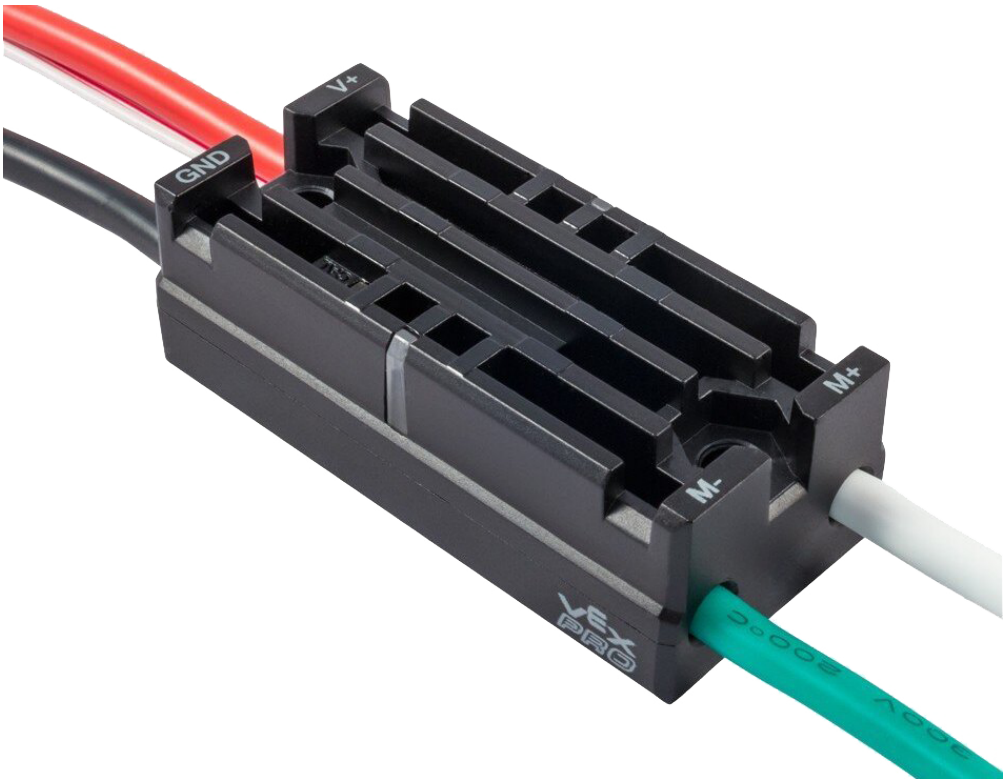
The TalonFX Motor Controller is integrated into the Falcon 500 brushless motor. It features an integrated encoder and all of the smart features of the Talon SRX and more! For more information see the Falcon 500 User Guide.

6.14.5 SPARK 电机控制器

警告： 尽管此电机控制器仍可用于 FRC，但制造商已停止使用该产品。

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 Victor SP



警告： 尽管此电机控制器仍可用于 FRC，但制造商已停止使用该产品。

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 Talon 电机控制器

警告： 尽管此电机控制器仍可用于 FRC，但制造商已停止使用该产品。

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

6.14.8 Victor 888 电机控制器/ Victor 884 电机控制器

警告： 尽管此电机控制器仍可用于 FRC，但制造商已停止使用该产品。

VEX Robotics 的 'Victor 884 <https://content.vexrobotics.com/docs/ifi-v884-users-manual-9-25-06.pdf>'__ 和 'Victor 888 <https://content.vexrobotics.com/docs/217-2769-Victor888UserManual.pdf>'__ 电动机控制器是用于 FRC 的变速 PWM 电动机控制器。Victor888 替代了也在 FRC 中使用的 Victor884。

6.14.9 Jaguar 电机控制器

警告： 尽管此电机控制器仍可用于 FRC，但制造商已停止使用该产品。

VEX Robotics（以前由 Luminary Micro 和 Texas Instruments 生产）的 'Jaguar Motor Controller <https://www.ti.com/lit/an/spma033a/spma033a.pdf?ts=1607574399581>'__ 是用于 FRC 的变速电动机控制器。对于 FRC，Jaguar 只能使用 PWM 接口进行控制。

6.14.10 DMC-60 和 DMC-60C 电机控制器

警告： 尽管此电机控制器仍可用于 FRC，但制造商已停止使用该产品。

DMC-60 是 Digilent 的 PWM 电机控制器。DMC-60 具有集成的热感应和保护功能，包括电流折返功能以防止过热和损坏，以及四个多色 LED 指示速度，方向和状态，以便于调试。有关更多信息，请参见 'DMC-60 参考手册 <https://reference.digilentinc.com/_media/dmc-60/dmc60_rm.pdf>'__

DMC-60C 在 DMC-60 控制器中增加了 CAN 智能控制器功能。由于制造商停止生产该产品，因此 DMC-60C 仅可与 PWM 一起使用。有关更多信息，请参见 'DMC-60C 产品页面 <https://reference.digilentinc.com/dmc-60c/start/>'__

6.14.11 Venom 电机控制器

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 带控制器的 Nidec Dynamo BLDC 电机

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 SD540B 和 SD540C 电机控制器

Mindsensors 的 SD540B 和 SD540C 电机控制器使用 PWM 控制。由于缺乏制造商的支持，SD540C 不再可以使用 CAN 控制。限位开关可以直接连接到 SD540，以限制电机在一个或两个方向上的行程。有关更多信息，请参见 'Mindsensors FRC 页面 <http://www.mindsensors.com/68-frc>'__

6.15 Spike H-Bridge 继电器

警告： 尽管该继电器仍可用于 FRC，但制造商已停止使用该产品。

VEX Robotics 的 Spike H 桥继电器是一种用于控制电动机或其他自定义机器人电子设备电源的设备。当连接到电动机时，Spike 可以在正向和反向上提供开/关控制。Spike 输出是独立控制的，因此它也可以用于为最多 2 个定制电子电路提供电源。Spike H 桥继电器应连接到 roboRIO 的继电器输出，并由配电盘供电。有关更多信息，请参见**'Spike 用户指南 <https://content.vexrobotics.com/docs/spike-blue-guide-sep05.pdf>'__**。

6.16 伺服电源模块

Rev Robotics 的伺服电源模块能够将伺服器可用的功率扩展到 roboRIO 集成电源所不具备的能力。伺服电源模块可通过 6 个通道提供高达 90W 的 6V 电源。所有控制信号都直接从 roboRIO 传递。有关更多信息，请参见 '伺服电源模块网页 <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 图片积分

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 软件组件概述

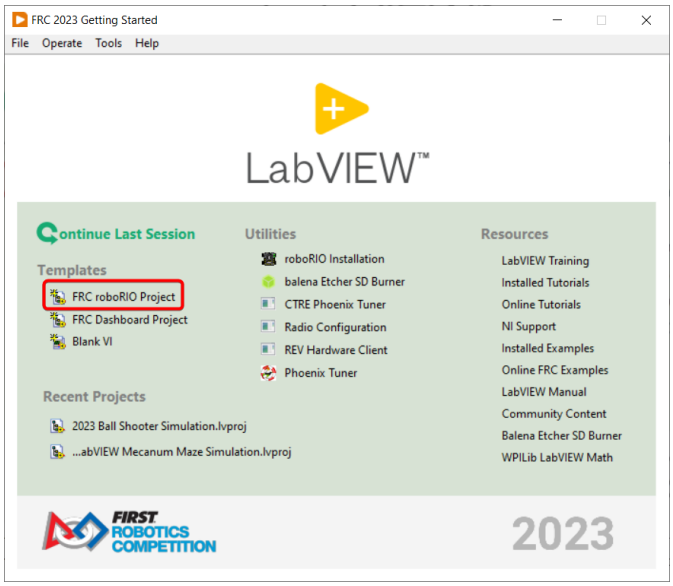
FRC® 软件包含各种强制性和可选性组件。它们旨在帮助您设计，开发和调试机器人代码、您控制机器人，并在排除故障时提供反馈。本文档将简要概述每个软件组件的用途，提供软件包下载链接以及其他文档的链接。

7.1 操作系统兼容性

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

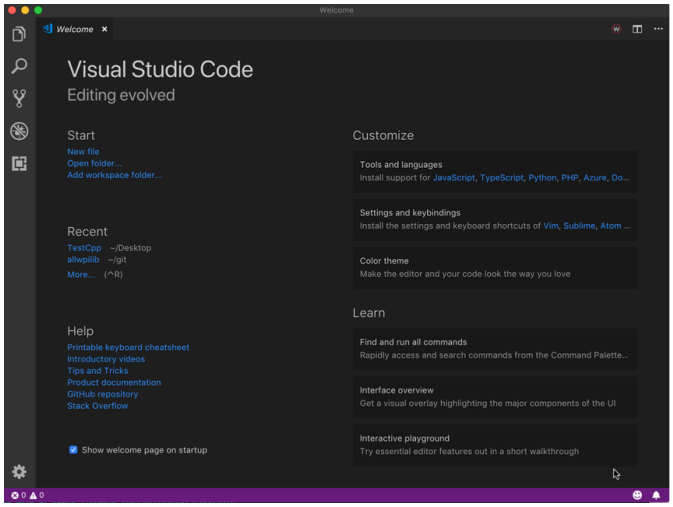
Many of the tools for C++/Java/Python programming are also supported and tested on macOS and Linux. Teams programming in C++/Java/Python should be able to develop using these systems, using a Windows system for the Windows-only operations such as the Driver Station, Radio Configuration Utility, and roboRIO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

7.2 LabVIEW FRC（仅支持 Windows）



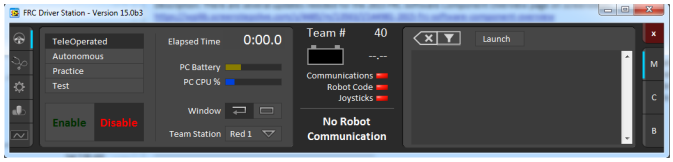
基于最新版本的 LabVIEW Professional 语言的 LabVIEW FRC 语言，是得到官方支持的用于 FRC 机器人编程的三种的语言之一。LabVIEW 是一种图形化、数据流驱动的语言。LabVIEW 程序由称为 VI 的图标集合组成，这些图标通过在 VI 之间传递数据的导线连接在一起。LabVIEW FRC 安装程序在 Kickoff 套件里的 DVD 中，也可以从网上下载。LabVIEW FRC 软件入门指南，与安装说明，可在:ref:‘此处 <docs/zero-to-robot/step-2/labview-setup:Installing LabVIEW for FRC (LabVIEW only)>‘找到。

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*.

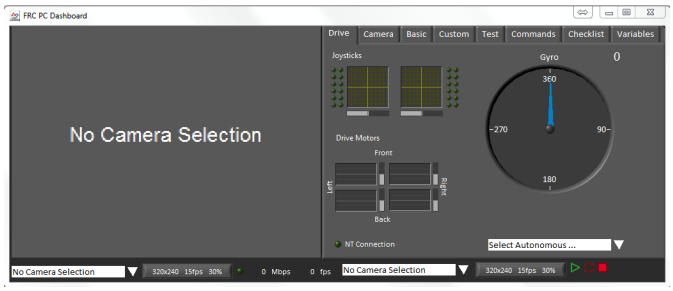
7.4 NI LabVIEW 支持的 FRC Driver Station（仅支持 Windows）



这是唯一一个能用来在比赛期间控制机器人状态的软件。它可通过各种输入设备将数据发送到您的机器人。它还包含许多工具，用于帮助对机器人问题进行故障排除。可以:ref:‘在此<docs/software/driverstation/driver-station:FRC Driver Station Powered by NI LabVIEW>‘找到有关它的更多信息。

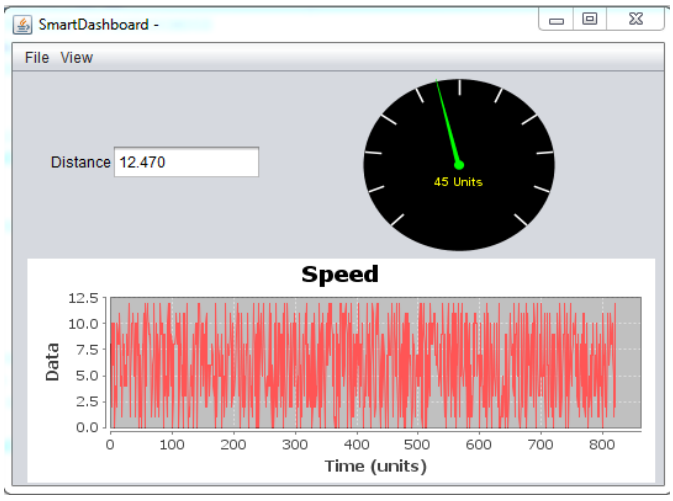
7.5 Dashboard 选项

7.5.1 LabVIEW Dashboard（仅支持 Windows）



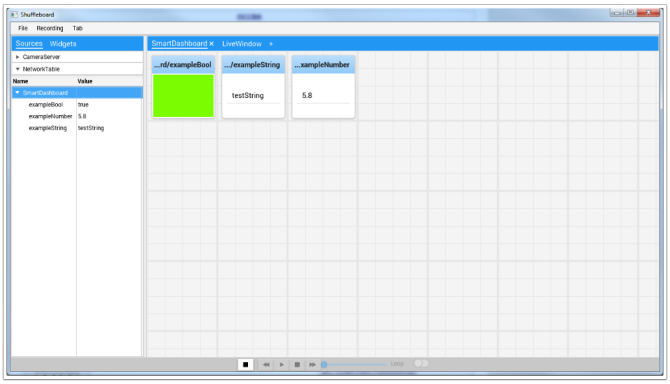
LabVIEW Dashboard 默认由 FRC 机器操控台自动启动。Dashboard 的目的是使用带有各种内置功能的选项卡显示提供关于机器人操作的反馈。更多关于 FRC 默认仪表盘软件的信息可以在这里找到:ref: ‘<docs/software/dashboards/labview-dashboard/driver-station-labview-dashboard:FRC LabVIEW Dashboard>‘.

7.5.2 智能仪表盘



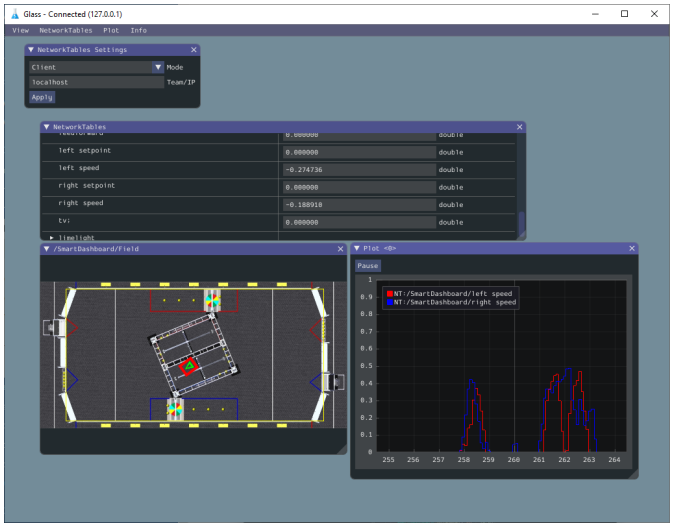
SmartDashboard 允许您通过为机器人发送的每条数据自动创建可定制的指标来查看机器人数据。SmartDashboard 的其他文档可以找到:ref:*here <docs/software/dashboards/smartdashboard/index:SmartDashboard>*.

7.5.3 模块化仪表盘



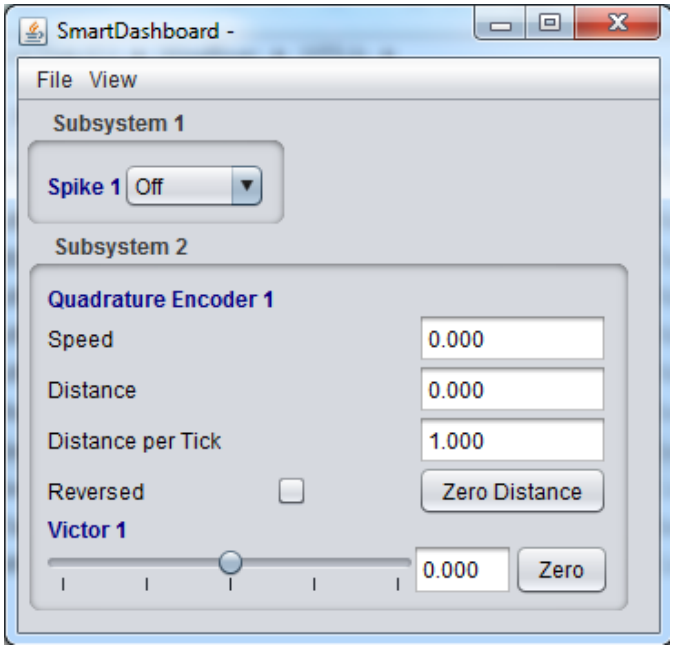
模块化仪表盘和 SmartDashboard 具有相同的功能。它还通过新特性和现代设计改进了数据的设置和可视化，但降低了资源效率。额外的文件沙狐球可以找到:ref:*here <docs/software/dashboards/shuffleboard/index:Shuffleboard>*.

7.5.4 Glass



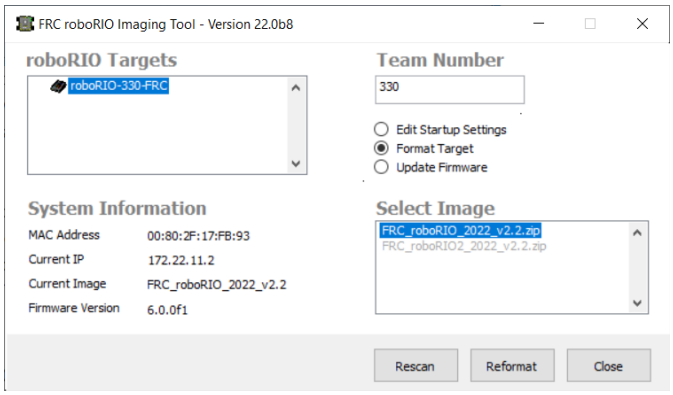
Glass 是一个 Dashboard，专注于成为程序员的调试工具。主要优点是现场视图，姿态可视化和先进的信号绘图工具。

7.6 LiveWindow



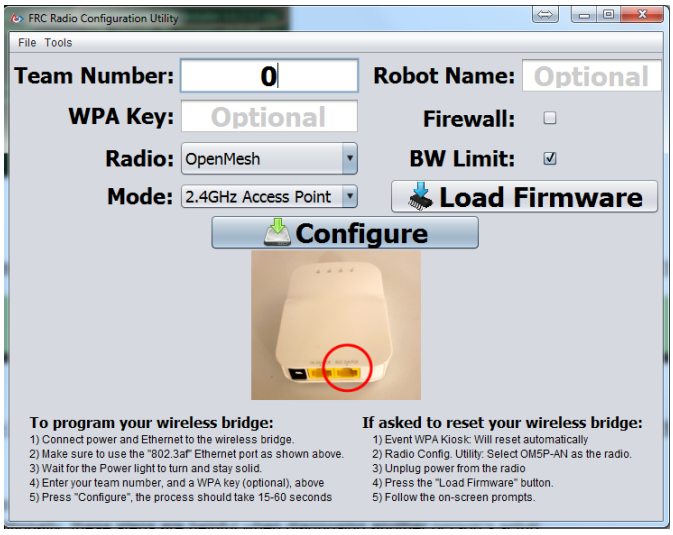
LiveWindow 是 SmartDashboard 和 Shuffleboard 的一个功能，专为机器操控台的测试模式而设计。LiveWindow 允许用户看到来自机器人传感器和控制执行器的反馈，而无需编写用户代码。可以找到更多关于 LiveWindow 的信息*here*.

7.7 FRC roboRIO Imaging Tool (仅支持 Windows)



此工具用于格式化和设置 roboRIO 以在 FRC 中使用。可以在:ref:‘此处 <docs/zero-to-robot/step-2/frc-game-tools:Installing the FRC Game Tools>‘找到安装说明。您可以在:doc:‘此处 </docs/zero-to-robot/step-3/imaging-your-roborio>‘找到有关使用此工具对 roboRIO 进行成像的其他说明。

7.8 FRC Radio Configuration Utility（仅支持 Windows）

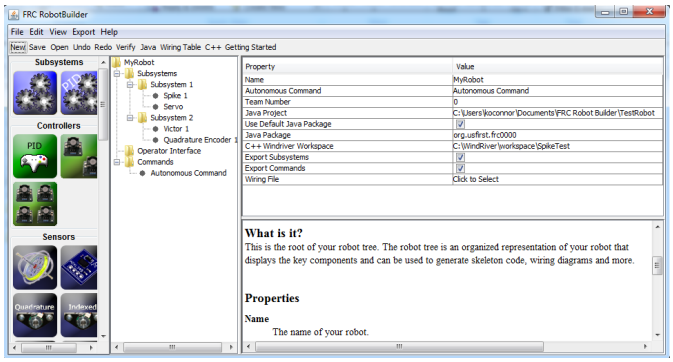


FRC Radio Configuration Utility 是用于配置标准无线连接以在家中练习的工具。该工具设置适当的网络设置，以模仿 FRC 赛场的体验。它由独立安装程序安装，可以在:ref:‘这里 <docs/zero-to-robot/step-3/radio-programming:Programming your Radio>‘找到。

7.9 FRC Driver Station Log Viewer (仅支持 Windows)

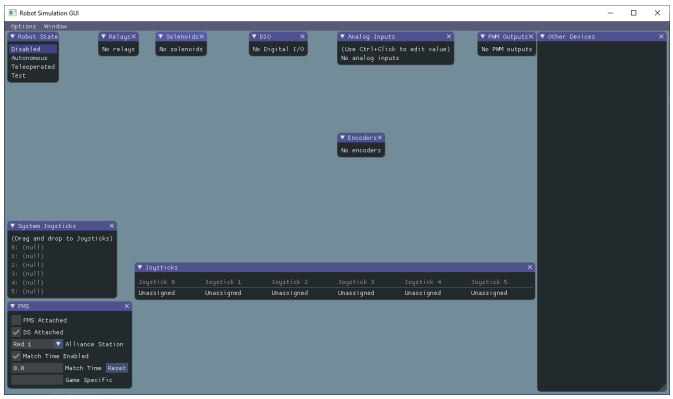
FRC Driver Station 日志查看器可用于查看 FRC Driver Station 创建的日志。这些日志包含各种信息，对于理解练习或 FRC 比赛期间发生的事情非常重要。有关 FRC Driver Station Log Viewer 和了解日志的更多信息，请参见此处

7.10 RobotBuilder 软件



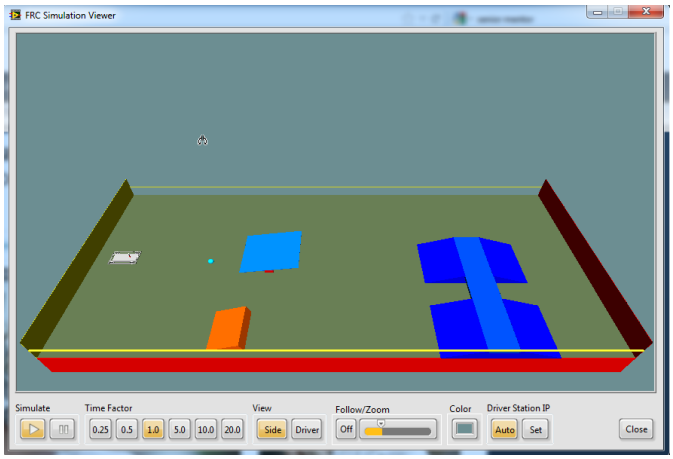
RobotBuilder is a tool designed to aid in setup and structuring of a Command Based robot project for C++ or Java (Python not currently supported). RobotBuilder allows you to enter in the various components of your robot subsystems and operator interface and define what your commands are in a graphical 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 Robot Simulation 软件



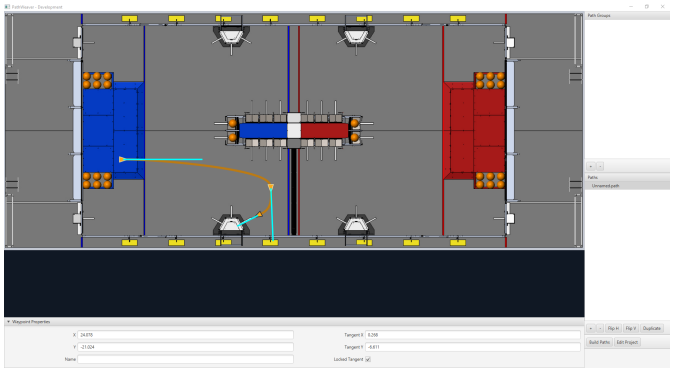
Robot Simulation offers a way for Java, C++, and Python teams to verify their actual robot code is working in a simulated environment. This simulation can be launched directly from VS Code and includes a 2D field that users can visualize their robot's movement on. For more information see the *Robot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

7.12 FRC LabVIEW Robot Simulator (仅支持 Windows)



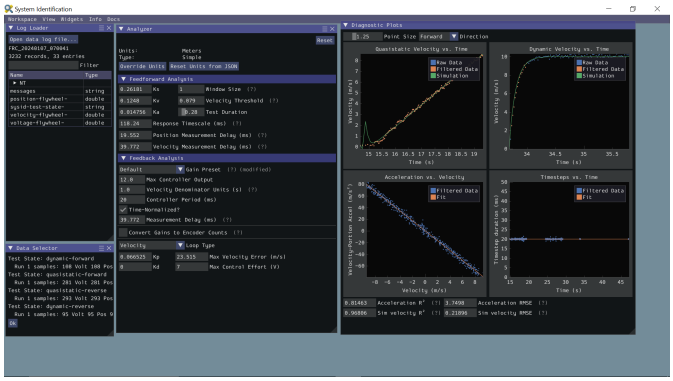
FRC LabVIEW Robot Simulator 是 LabVIEW 编程环境的组件，允许您在模拟环境中操作预先设定好的机器人以测试代码和/或 driver station 的功能。有关使用 FRC LabVIEW Robot Simulator 的信息，可在 '这里 <https://forums.ni.com/t5/FIRST-Robotics-Competition/LabVIEW-Tutorial-10-Robot-Simulation/ta-p/3739702?profile.language=en>'找到，或在 LabVIEW 项目浏览器中打开 Robot Simulation Readme.html 文件。

7.13 PathWeaver 软件



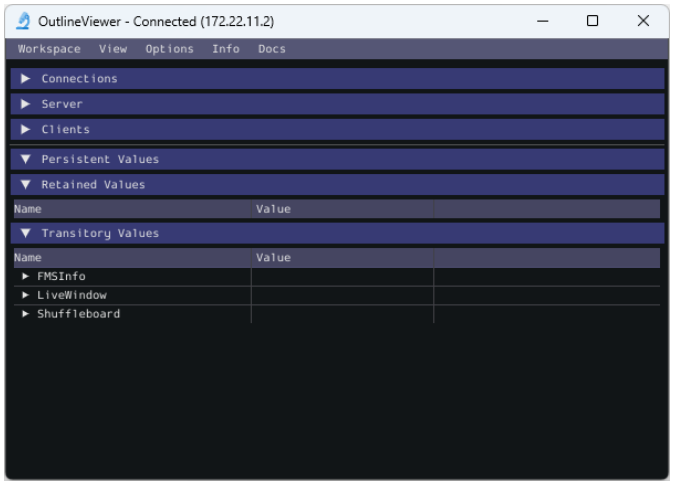
PathWeaver allows teams to quickly generate and configure paths for advanced autonomous routines. These paths have smooth curves allowing the team to quickly navigate their robot between points on the field. For more information see the *PathWeaver section*.

7.14 系统识别



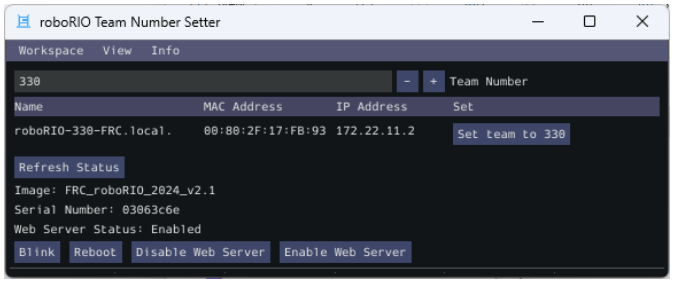
This tool helps teams automatically calculate constants that can be used to describe the physical properties of your robot for use in features like robot simulation, trajectory following, and PID control. For more information see the *System Identification section*.

7.15 OutlineViewer



OutlineViewer 是一个实用工具，用于查看、修改和添加到 NetworkTables 的所有内容，以进行调试。LabVIEW 团队可以使用 LabVIEW 仪表盘的 Variables 选项卡来完成这个功能。有关更多信息，请参见 ref:*Outline Viewer section <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 什么是 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 支持的语言

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 源代码和文档

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:

可以在 WPILibJ 和 WPILibC 源目录中找到 Java 和 C++ 源代码：

- **‘Java 源代码 <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibj/src/main/java/edu/wpi**
- **‘C++ 源代码 <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibc/src/main/native/cpp>**

- Python source code

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

- Java documentation
- C++ documentation
- Python documentation

9 2024 Overview

9.1 已知问题

本文详细介绍了 FRC|reg| 控制系统软件的已知问题（和解决方法）。

9.1.1 公开的问题

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.

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

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

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

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

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

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

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

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

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

Driver Station Reports Less Free RAM then is Available

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

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

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

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

Driver Station Reporting No Code

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

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

> **备注:** 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.

由于缺少 GradleRIO 导致构建无效

问题：用户的 Gradle 缓存很少会损坏，并且会显示类似于以下内容的错误：

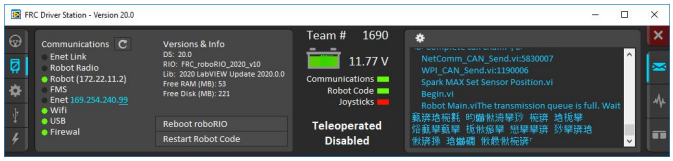
```
Could not apply requested plugin [id: 'edu.wpi.first.GradleRIO', version: '2020.3.
↪2'] as it does not provide a plugin with id 'edu.wpi.first.GradleRIO'
```

解决方法：

删除位于 "~$USER_HOME/.gradle" 下的 Gradle 缓存。Windows 可能需要启用查看隐藏文件的能力 <https://support.microsoft.com/en-us/windows/view-hidden-files-and-folders-in-windows-10-97fbc472-c603-9d90-91d0-1166d1d9f4b5>'__。到目前为止，此问题仅出现在 Windows 上。如果您在其他操作系统上遇到此问题，请向 <https://github.com/wpilibsuite/frc-docs/issues/new>'__ 报告这个问题。

Driver Station 日志中的汉字

问题：极少情况下，driver station 日志会显示中文字符而不是英文文本。仅当 Windows 设置为英语以外的其他语言时，才会出现这种情况。



解决方法：有两种已知的解决方法：

1. 将中文字符复制并粘贴到记事本中，将显示英文文本。
2. 暂时将 Windows 语言更改为英语。

C ++ Intellisense-启动时打开的文件无法正常工作

问题：在 C++ 中，当 VS Code 启动时打开的文件将出现“Intellisense”功能异常，显示来自编译单元的所有选项的建议而不仅仅是适当的选项，或者找不到头文件。这是 VS Code 中的一个 bug。

解决方法：

1. 关闭 VS Code 中的所有文件，但保持 VS Code 处于打开状态
2. 如果存在，删除.vscode 文件夹中的 c_cpp_properties.json 文件
3. Run the “Refresh C++ Intellisense” command in VS Code.
4. 在右下角，您应该会看到类似于平台的东西（linuxathna 或 windowsx86-64 等）。如果不是 linuxastina，请单击它并将其设置为 linuxastina(release)。
5. 等待🄴1 分钟
6. 打开主 cpp 文件（而不是头文件）。Intellisense 现在应该可以工作了

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

解决方案：安装‘媒体功能包 <https://www.microsoft.com/en-us/software-download/mediafeaturepack>‘__

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 Major Changes (Java/C++)

These changes contain *some* of the major changes to the library that it's important for the user to recognize. This does not include all of the breaking changes, see the other sections of this document for more changes.

- 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.

警告: 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

综合库

- 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); }};`

重大变化

- 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 模拟

- 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

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

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

9.2.6 Glass / OutlineViewer / Simulation GUI

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

9.2.7 GradleRIO

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

9.2.8 WPILib All in One Installer

- 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 扩展

- 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

VS Code 概述

10.1 VS Code 基础以及 WPILib 扩展

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 欢迎页面

| 欢迎页面 |

当你第一次打开 Visual Studio Code，你会看见一个 “欢迎” 页面。在这个界面上你可以找到一些设定 VS Code 的链接，你还可以找到一些指引到帮助文档和视频的链接，这些或许可以帮助你学到一些这个 IDE 的基础和窍门。

你可能会注意到在右上角有个小的 WPILib 的标识。这可以帮助你获取那些 WPILib 插件提供的功能（下文将介绍）。

10.1.2 用户界面

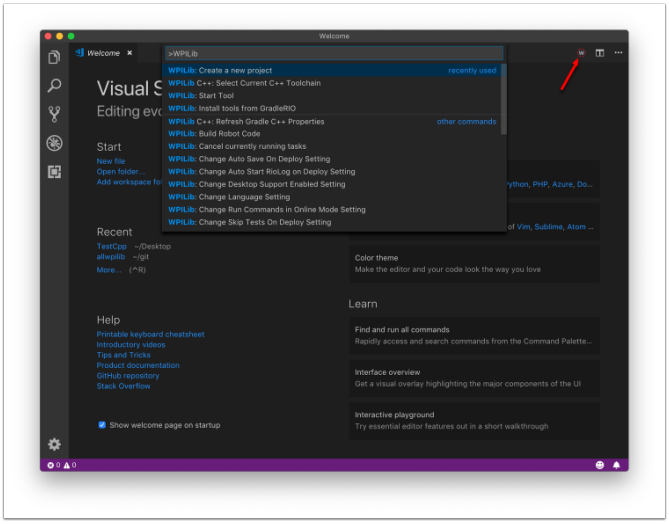
对你来说最重要的链接或许会是有关用户界面基础的文档。这篇文档介绍了使用用户界面 (UI) 的许多基础知识，并提供了使用 FRC VS Code 之前你所需的大多数信息。

10.1.3 指令面板

指令面板可用于访问或运行 Visual Studio Code 中的几乎所有功能（包括 WPILib 扩展中的功能）。可以从查看菜单或快捷键 `Ctrl+Shift+P`（在 macOS 上为 `:kbd:`Cmd+Shift+P`）来访问命令面板。在指令面板的窗口中键入文本将缩小搜索范围，相关命令会动态地显示在下拉菜单中。

比如，激活 “指令面板” 后在搜索框中键入 “wpilib”，显示的命令范围将缩小到 WPILib 包含的功能。

10.1.4 WPILib 扩展



WPILib 插件提供了 FRC® 特有的功能，包括创建项目和项目组件，编译代码并下载到 roboRIO 等。您可以通过以下两种方式之一访问 WPILib 命令：

- 在指令面板中键入“WPILib”
- 单击大多数窗口右上方的 WPILib 图标。这将打开预先输入“WPILib”的指令面板

备注： **不 ** 建议在 FRC 的 VS Code 上安装 Visual Studio IntelliCode <https://marketplace.visualstudio.com/items?itemName=VisualStudioExptTeam.vscodeintellicode>‘_ 插件，IntelliSense 会有玄学问题。

请参阅本章中的其他文章来了解有关特定 WPILib 扩展命令的更多信息。

10.2 Visual Studio Code 中的 WPILib 命令

本文档包含 VS Code 的 WPILib 扩展提供的命令及其作用的完整列表。

要使用这些指令，请先按 Ctrl + Shift + P 打开“指令面板”，然后开始输入命令名称（如下所示）以过滤命令列表。单击命令名称以执行它。

- **WPILib：Build Robot Code**-使用 GradleRIO 构建打开的项目
- **WPILib：Create a new project**-创建一个新的机器人项目
- **WPILib C ++：Refresh C ++ Intellisense**-强制更新 C ++ Intellisense 的配置。
- **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**-停止 WPILib 扩展正在运行的所有任务

- **WPILib: Change Auto Save On Deploy Setting**-更改在执行部署时是否自动保存文件。默认为启用。
- **WPILib: Change Auto Start RioLog on Deploy Setting**-更改 RioLog 是否在部署时自动启动。默认为启用。
- **WPILib: Change Desktop Support Enabled Setting**-更改是否启用在电脑上构建机器人代码。在测试和模拟时可以启用此功能。默认为关闭。
- **WPILib: Change Language Setting**-更改当前打开的是 C ++ 还是 Java 项目。
- **WPILib: Change Run Commands Except Deploy/Debug in Offline Mode Setting**-更改 GradleRIO 是否在在线模式下运行除部署/调试以外的命令（将尝试从联机状态自动提取依赖项）。默认为启用（在线模式）。
- **WPILib: Change Run Deploy/Debug Command in Offline Mode Setting**-更改 GradleRIO 是否在在线模式下进行部署/调试（将尝试从在线状态自动提取依赖项）。默认为禁用（离线模式）。
- **WPILib: Change Select Default Simulate Extension Setting**-更改默认情况下是否启用模拟扩展（"build.gradle"中定义的所有模拟扩展都将被启用）
- **WPILib: Change Skip Tests on Delpoy Setting**-设定在部署时是否跳过测试。默认为禁用（在部署时运行测试）
- **WPILib: Change Stop SImulation on Entry Setting** –更改在开始运行模拟时是否停止机器人代码。默认为禁用（不停止输入）。
- **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**-构建机器人代码，将其配置到 roboRIO，以调试模式开始调试
- **WPILib: Deploy Robot Code**-构建机器人代码并将其部署到 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** -安装 WPILib Java 工具（例如 SmartDashboard，Shuffleboard 等）。请注意，默认情况下，此操作由离线安装程序完成
- **WPILib: Manage Vendor Libraries**-安装/更新第三方库
- **WPILib: Open API Documentation**-打开 WPILib Javadocs 或 C ++ Doxygen 文档
- **WPILib: Open Project Information**-打开带有项目信息（项目版本，扩展版本等）的 "小帮手"
- **WPILib: Open WPILib Command Palette**-此指令用于打开 WPILib 指令面板（等同于按 Ctrl + Shift + P 并输入" WPILib''）
- **WPILib: Open WPILib Help**-这将打开一个链接到 WPILib 文档（此网站）的页面
- **WPILib: Reset Ask for WPILib Updates Flag**-这将清除当前项目上的标志，如果您先前选择暂不更新，这使得您可以将项目更新为最新的 WPILib 版本。

- **WPILib: Run a command in Gradle**-这使您可以在 GradleRIO 环境中运行任意指令
- **WPILib: Set Team Number**-用于修改与项目关联的团队编号。您只会在创建项目时更改最初指定的团队号时用到。
- **WPILib: Set VS Code Java Home to FRC Home*-设置 VS Code Java Home 变量以指向 FRC 扩展发现的 Java Home。如果不使用离线安装程序，您需要这样做以确保 intellisense 设置与 WPILib 构建设置同步。
- **WPILib: Show Log Folder**-显示 WPILib 扩展存储内部日志的文件夹。在调试或报告扩展问题给 WPILib 开发人员时，这或许很有用。
- **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**-这将启动用于查看机器人程序的控制台输出的 RioLog 显示。
- **WPILib: Start Tool**-这使您可以从 VS Code 内部启动 WPILib 工具（例如 SmartDashboard，Shuffleboard 等）。
- **WPILib: Test Robot Code**-这将构建当前的机器人代码项目并运行任何创建的测试。这需要将“Desktop Support”设置为已启用。

10.3 创建机器人程序

安装完所有内容后，我们就可以创建机器人程序了。WPILib 随附了多个机器人程序模板。强烈建议新用户使用这些模板。但是，高级用户可以自由地从头开始编写自己的机器人代码。

10.3.1 选择基类

要使用 WPILib 机器人程序模板之一启动项目，用户必须首先为其机器人选择基类。用户将这些基类子类化以创建其主 Robot 类，该主类控制机器人程序的主要流程。基类有三种选择：

TimedRobot

Documentation: Java - C++

源: 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++

备注： 提供了“TimedRobot Skeleton”模板，该模板删除了一些有用的注释和自动阶段示例。如果您已经熟悉‘TimedRobot‘，则可以使用它。下面显示的示例是“TimedRobot Skeleton”。

JAVA

```java
import edu.wpi.first.wpilibj.TimedRobot;

/**
 * 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 build.gradle file in the
 * project.
 */
public class Robot extends TimedRobot {
  /**
   * This function is run when the robot is first started up and should be used for any
   * initialization code.
   */
  @Override
  public void robotInit() {}

  @Override
  public void robotPeriodic() {}

  @Override
  public void autonomousInit() {}

  @Override
  public void autonomousPeriodic() {}

  @Override
  public void teleopInit() {}

  @Override
  public void teleopPeriodic() {}

  @Override
  public void disabledInit() {}

  @Override
  public void disabledPeriodic() {}

  @Override
  public void testInit() {}

  @Override
  public void testPeriodic() {}

  @Override
  public void simulationInit() {}

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

C++

```
#include "Robot.h"

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

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

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

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

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

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

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

默认情况下，周期方法每 20 毫秒调用一次。这可以通过使用新的所需更新率调用超类构造函数来更改。

> **危险：** 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++

源: Java - C++

RobotBase 类是提供最基本的基类，一般不建议直接使用。没有为用户处理机器人控制流程。一切都必须从头开始写在 startCompetition() 方法内部。默认情况下，模板展示了如何处理不同的操作模式（遥控，自动等）。

备注: RobotBase Skeleton 模板提供了一个空白的方法 startCompetition()。

Command Robot

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.

使用“Command Robot”的队伍需要查看:ref:*Command-Based Programming Tutorial <docs/software/commandbased/index:Command-Based Programming>*.

Romi

使用*Romi* 的团队应使用“Romi-Timed”或“Romi-Command Bot”模板。

Romi - Timed

Romi - Timed 模板提供了一个 RomiDrivetrain 类，该类公开了一个 arcadeDrive(double xaxisSpeed, double zaxisRotate) 方法。用户可以为此 arcadeDrive 函数提供参数。

该类还提供用于检索和重置 Romi 板载编码器的功能。

Romi - Command Bot

Romi - Command Bot 模板提供了一个 RomiDrivetrain 子系统，该子系统公开了一个 arcadeDrive(double xaxisSpeed, double zaxisRotate) 方法。用户可以给此 arcadeDrive 函数提供参数。

该子系统还提供用于检索和重置 Romi 板载编码器的功能。

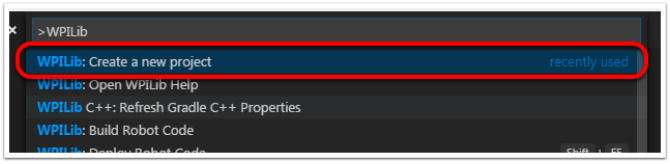
不使用基类

如果需要，用户可以完全省略基类，而只需使用一种 main() 方法编写其程序即可，就像处理其他任何程序一样。这非常不推荐-写机器人的代码时，用户不应该“另起炉灶”-但还是支持那些希望有他们的程序的绝对控制权的人。

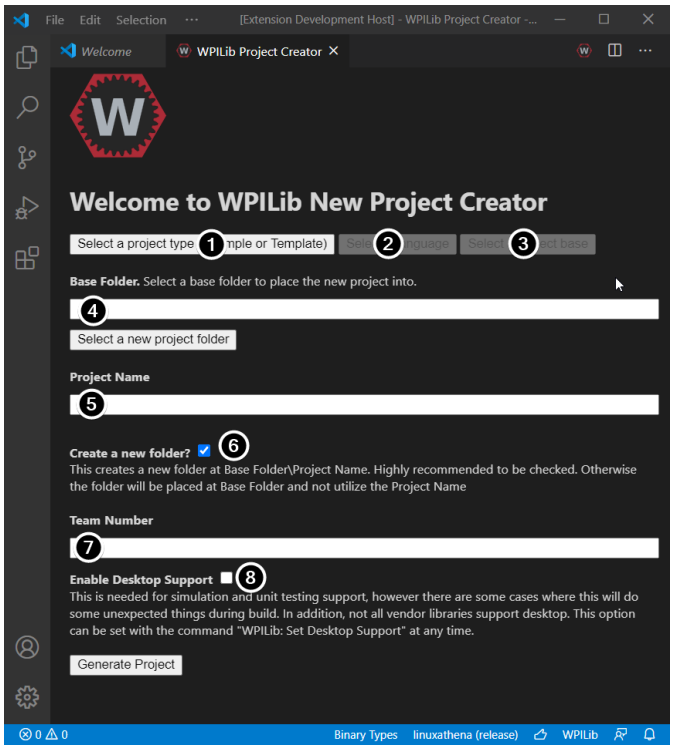
> **警告：** 用户不应该修改机器人程序的 main() 方法，除非他们绝对确定自己在做什么。

10.3.2 创建新的 WPILib 项目

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:



这将打开“新项目创建者窗口”：



创建新建项目窗口的元素说明如下：

1. **项目类型**：我们希望创建的项目类型。这可以是示例项目，也可以是 WPILib 提供的项目模板之一。每个机器人基本类都有模板。此外，还有:ref:‘Command-based <docs/software/commandbased/what-is-command-based:What is “command-based” programming?>‘项目的模板，这些模板基于 TimedRobot 基类构建，但包含许多其他功能-强烈建议新团队使用这种类型的机器人程序。
2. **语言**：这是将用于该项目的语言（C ++ 或 Java）。
3. **基础文件夹**：如果这是模板项目，则指定将使用的模板类型。
4. **项目位置**：确定机器人项目所在的文件夹。
5. **项目名称**：机器人项目的名称。如果选中了“创建新文件夹”框，这还将指定项目文件夹的名称。
6. **创建新文件夹**：如果选中此选项，将创建一个新文件夹，以将项目保存在先前指定的文件夹中。如果未选中，则项目将直接位于先前指定的文件夹中。如果文件夹不为空且未选中，将引发错误。
7. **团队编号**：项目的团队编号，将用于项目中的程序包名称并在部署代码时定位机器人。
8. **启用桌面支持**：启用单元测试和模拟。尽管 WPILib 支持此功能，但第三方软件库可能不支持。如果库不支持桌面，则您的代码可能无法编译或崩溃。除非需要进行单元测试或仿真并且所有库都支持它，否则应将其保持选中状态。

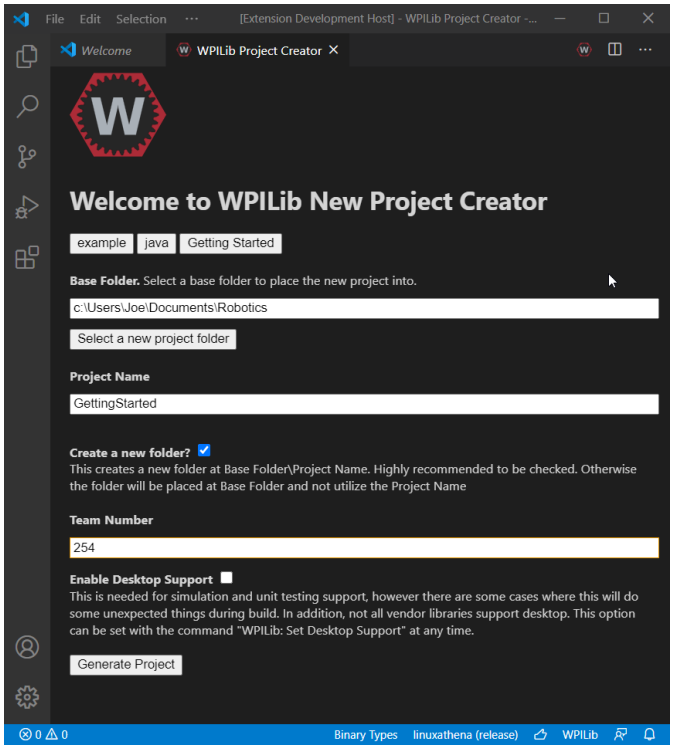
完成以上所有配置后，单击“生成项目”，将创建机器人项目。

备注： 项目生成中的任何错误将显示在屏幕的右下角。

警告： 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

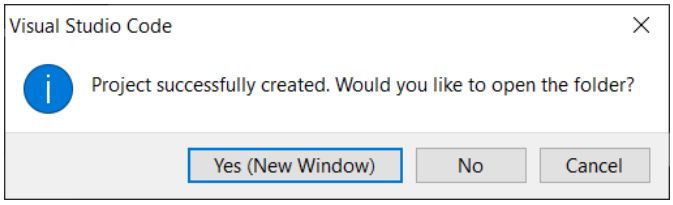
on OneDrive by default.

选择所有选项后的示例如下所示。

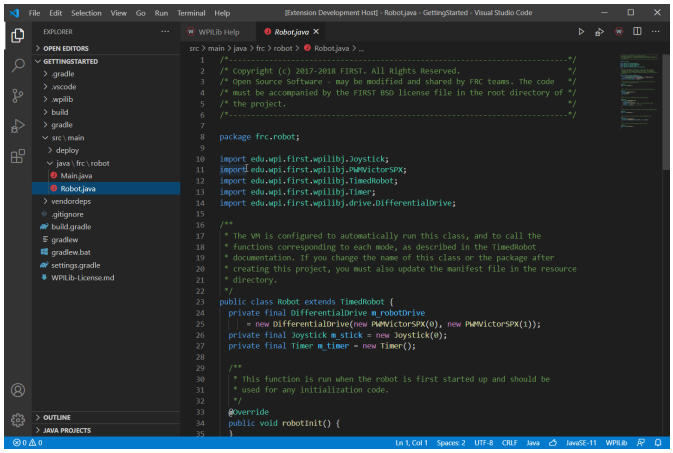


10.3.3 打开新项目

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.

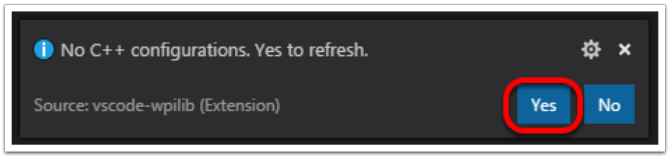


打开后，我们将在左侧看到项目层次结构。双击文件将在编辑器中打开该文件。



10.3.4 C ++ 配置（仅 C ++）

对于 C ++ 项目，还有另外一步来设置 IntelliSense。每当我们打开一个项目时，我们都应该在右下角出现一个弹出窗口，要求刷新 C ++ 配置。单击“是”以设置 IntelliSense。



10.4 第三方库

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.

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

备注: 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.

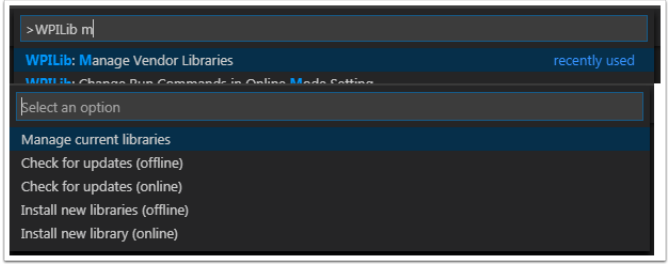
How Does It Work? - Python

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.

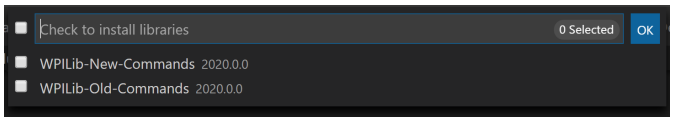
Installing Libraries

Java/C++

VS Code



要添加由脱机安装程序安装的供应商库，请按 Ctrl + Shift + P 并输入 WPILib/单击右上角的 WPILib 图标，以打开 WPILib 命令面板。然后，输入 “Manage Vendor Libraries”，然后从下拉菜单中选择它。选择 “Install new libraries (offline)” 的选项。



选中每个库旁边的框，选择要添加到项目中的所需库，然后单击 “确定”。这样，JSON 文件将被复制到项目中的 vendordeps 文件夹中，并将库添加为项目的依赖项。

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)

供应商可以选择填充一部分 JSON 文件为在线更新。如果库指定了适当的位置，则运行 “Check for updates (online)” 可检查远程位置是否有可用的较新版本的库。

Removing a Library Dependency

要从项目中删除库依赖性，请从“Manage Vendor Libraries”菜单中选择“Manage Current Libraries”，选中要卸载的任何库的复选框，然后单击 “确定”。这些库将从项目中删除。

Command-Line

还可以通过 gradle 任务通过命令行从供应商 URL 添加供应商库依赖项。在项目根目录下打开一个命令行，然后输入 “gradlew vendordep –url = <url>”，其中 <url> 是供应商 JSON URL。这会将供应商库依赖项 JSON 文件添加到项目的 vendordeps 文件夹中。供应商库可以用相同的方式更新。

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`

参见:

pyproject.toml usage

10.4.3 库

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 `RomiReference` 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`

备注: 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 - 包括 Venom motor/controller 在内的所有 PWF 设备的库

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 - NavX-MXP, NavX-Micro, 与 Sensor Fusion 的库

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"]`

社区库

PhotonVision-PhotonVision CV 软件库

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

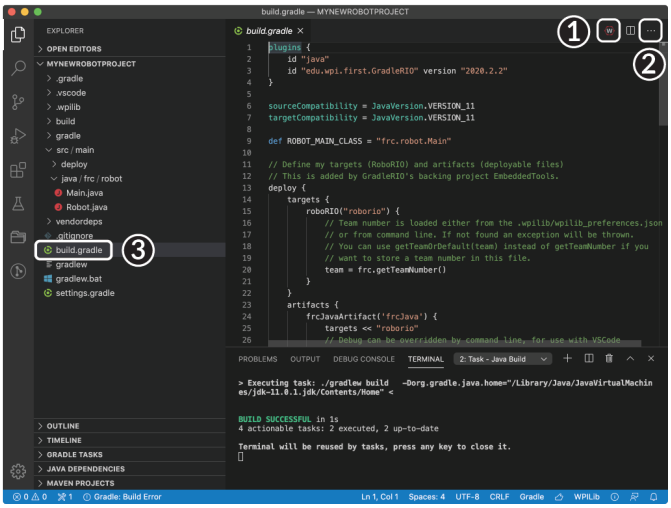
Not available

10.5 编译并部署机器人代码

机器人项目必须在编译（构建）和部署之后才能在 roboRIO 上运行。由于代码不是在机器人控制器上编译的，这个过程被称为“交叉编译”。

要编译并部署机器人项目，请执行以下一项操作：

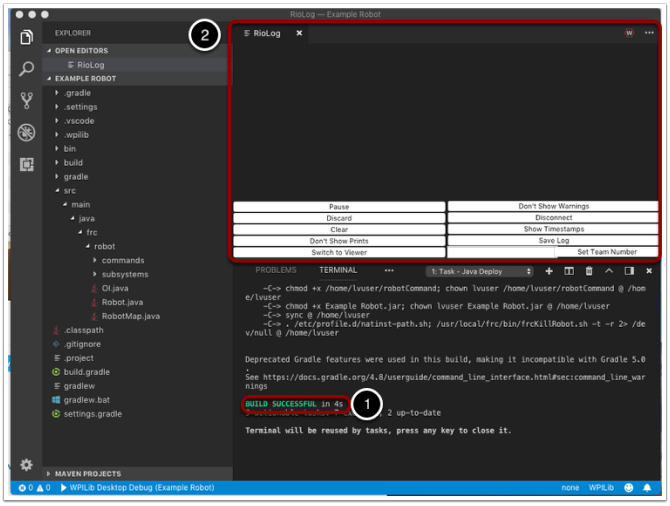
1. 打开命令面板，然后输入或选择“Build Robot Code”
2. 打开 VS Code 窗口右上角椭圆所指示的快捷菜单，然后选择“Build Robot Code”
3. 右键单击项目层次结构中的 build.gradle 文件，然后选择“Build Robot Code”



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.

警告： 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 查看控制台输出

为了查看基于文本的程序的控制台输出，roboRIO 实例化了一个 NetConsole。查看 roboRIO 的 NetConsole 输出有两种主要方法：FRC Driver Station 中的 Console Viewer 和 VS Code 中的 Riolog 插件。

备注： 在 roboRIO 上，NetConsole 仅用于程序输出。如果要与系统控制台进行交互，则需要使用 SSH 或串行控制台。

10.6.1 控制台查看器

打开控制台查看器

| 打开控制台查看器 |

要打开控制台查看器，请首先打开 FRC® Driver Station。然后，单击消息查看器窗口顶部的齿轮（1），然后选择 “View Console”。

控制台查看器窗口

| 控制台查看器窗口 |

Console Viewer 窗口以绿色显示我们的机器人程序的输出。右上方的齿轮可以清除窗口并设置显示的消息级别。

10.6.2 Riolog VS Code 插件

Riolog 插件是一个 VS Code 视图，可用于查看 VS Code 中的 NetConsole 输出（原始 Eclipse 版本的授权：Manuel Stoeckl，FRC1511）。

打开 RioLog 视图

| 打开 Riolog 视图 |

默认情况下，在每次 roboRIO 配置结束时，RioLog 视图将自动打开。要手动启动 RioLog 视图，请按 Ctrl+Shift+P 打开指令选项板并键入 “RioLog”，然后选择 WPILib：“启动 RioLog” 选项。

Riolog 窗口

RioLog 视图应出现在顶部窗格中。Riolog 包含许多用于操纵控制台的控件：

- **暂停/继续显示**-这将暂停/继续显示。在后台，新数据包仍将被接收，并在单击恢复按钮时显示。
- **丢弃/接受传入**-这将切换是否接受新数据包。当数据包被丢弃时，显示将暂停，所有接收到的数据包将被丢弃。再次单击该按钮将恢复接收数据包。
- **清除**-这将清除显示的当前内容。
- **不显示/不显示打印内容**-显示或隐藏归类为打印语句的信息
- **切换到查看器**-切换到查看器以保存日志文件
- **不显示/显示警告**-显示或隐藏归类为警告的消息
- **断开/重新连接**-这断开或重新连接到控制台流
- **显示/不显示时间戳**-在窗口中显示或隐藏消息的时间戳
- **保存日志**-将日志内容复制到一个文件中，您可以保存和查看该文件，或者以后再使用 RioLog 查看器打开（请参见上方的“切换到查看器”）
- **设置组号**-设置连接到控制台流的 roboRIO 的组号，如果在部署过程中启动 RioLog，则自动设置

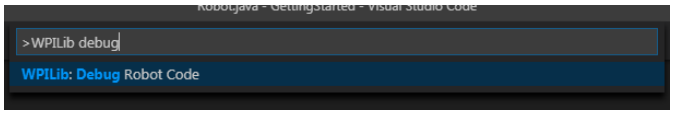
10.7 调试机器人程序

程序不可避免地不会像我们期望的那样运行。发生这种情况时，有必要弄清楚该程序为什么会执行其所执行的操作，以便我们可以使它执行我们想要执行的操作。这种期望外的程序行为称为“错误”，而此过程称为“调试”。

调试器是用于控制程序流和监视变量以帮助调试程序的工具。本节将描述如何为 FRC|reg| 机器人程序设置调试会话。

备注： 对于需要调试程序但不知道/没有时间学习如何使用调试器的入门用户，通常只需将相关程序状态打印到控制台即可调试程序。但是，强烈建议学生最终学习使用调试器。

10.7.1 运行调试器



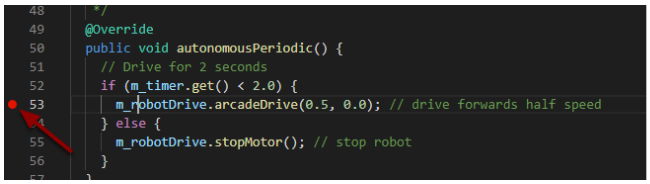
按 Ctrl+Shift+P 并输入 WPILib 或单击 *WPILib Menu Item* 打开带有预填充的 WPILib 的命令面板。键入 Debug 并选择 Debug Robot Code menu 项以开始调试。该代码将下载到 roboRIO 并开始调试。

10.7.2 断点

“断点”是一行代码，调试器将在该代码行处暂停程序执行，以便用户可以检查程序状态。这在调试时非常有用，因为它允许用户在有问题的代码中的特定位置暂停程序，以确定程序在哪些地方偏离了预期的行为。

调试器将在遇到的第一个断点处自动暂停。

设定断点



在源代码窗口的左边距中（在行号的左边）单击以在用户程序中设置一个断点：红色小圆圈表示已在相应行上设置了断点。

10.7.3 使用打印语句进行调试

调试程序的另一种方法是在代码中使用 print 语句，并使用 Visual Studio Code 或 Driver Station 中的 RioLog 查看它们。打印语句应谨慎添加，因为它们效率不高，尤其是在大量使用时。应将它们及时移除，因为它们可能导致循环超限。

JAVA

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

C++

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

10.7.4 使用网络表进行调试

:doc:‘网络表 </docs/software/networktables/networktables-intro>‘可用于与调试计算机共享机器人信息。可以使用您喜欢的仪表板或:ref:‘OutlineViewer <docs/software/wpilib-tools/outlineviewer/index:OutlineViewer>‘查看网络表。网络表的优点之一是可以使用:doc:‘Shuffleboard </docs/software/dashboards/shuffleboard/getting-started/shuffleboard-tour>‘之类的工具来图形化分析数据。然后，这些相同的工具可以与相同的数据一起使用，以在以后为驱动程序提供操作员界面。

10.7.5 了解更多

- 要了解有关使用 VS Code 进行调试的更多信息，请参见 ‘链接 <https://code.visualstudio.com/docs/editor/debugging>‘__。
- VS Code‘文章 <https://code.visualstudio.com/docs/editor/editingevolved>‘__ 中提到的某些功能将帮助您了解和诊断代码问题。快速修复（黄色灯泡）功能对包括导入内容在内的各种问题很有帮助。
- 避免调试很多问题的最佳方法之一是进行单元测试。
- 验证您的机器人是否可以在:doc: *模拟 </docs/software/wpilib-tools/robot-simulation/introduction>* 中运行，也是避免在实际机器人上进行复杂调试的一种好方法。

10.8 Importing Last Year's Robot Code

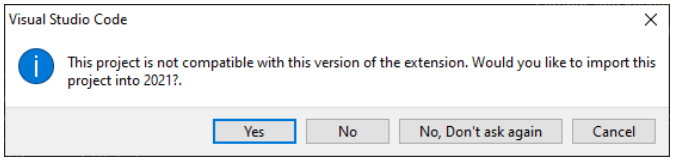
Due to changes in the project, it is necessary to update the build files for a previous years Gradle project. It is also necessary to import vendor libraries again, since last year's vendor libraries must be updated to be compatible with this year's projects.

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.

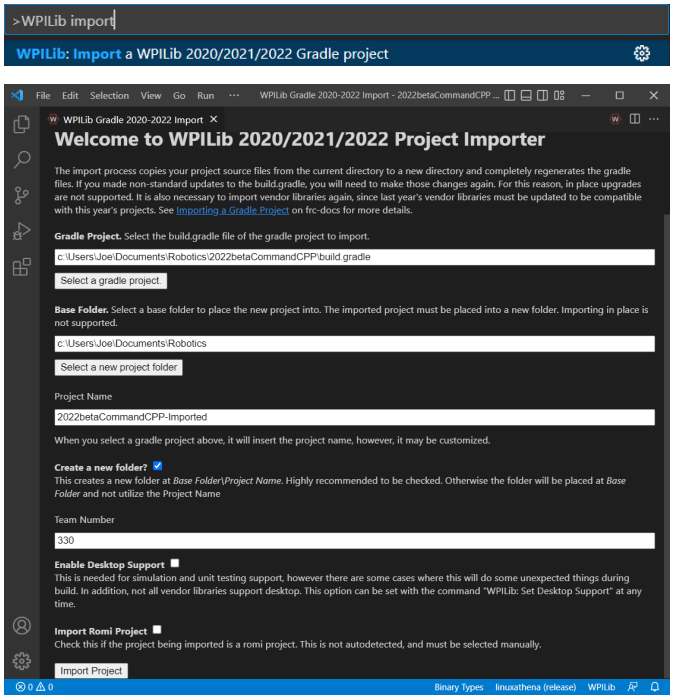
重要： The import process copies your project source files from the current directory to a new directory and completely regenerates the gradle files. Additionally, it updates the code for the package changes made in 2023. If you made non-standard updates to the `build.gradle`, you will need to make those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

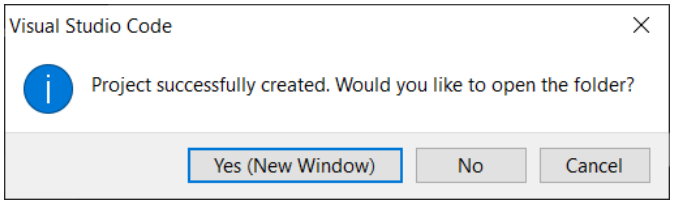
1. **Gradle Project**: Selects the project to be imported. Users should select the build.gradle file in the root directory of the gradle project.
2. **Project Location**: This determines the folder in which the robot project will be located.
3. **Project Name**: The name of the robot project. This also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

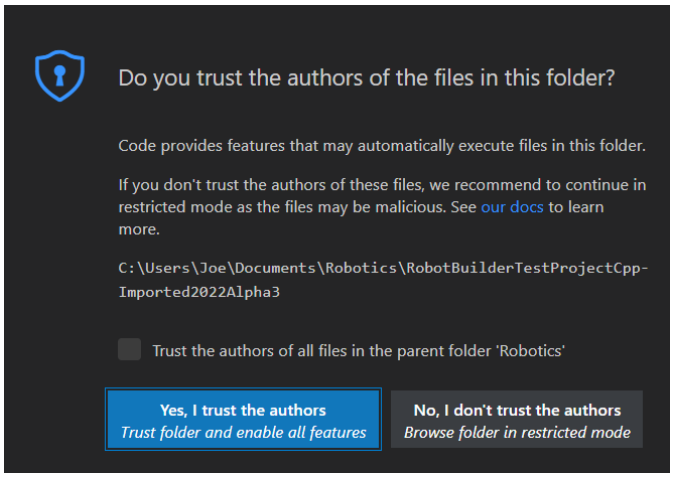
6. **Enable Desktop Support**: If this is checked, simulation and unit test support is enabled. However, there are some cases where this will do some unexpected things. In addition, all vendor libraries need desktop support which not all libraries do.
7. **Import Romi Project**: If this is checked, the project is imported using the Romi gradle template. This should only be checked for Romi projects.

> **警告:** 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.

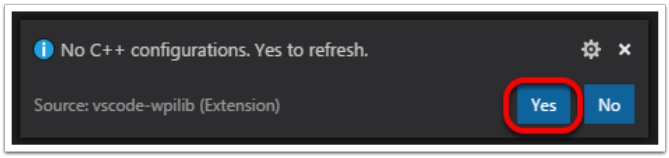




Click *Yes I trust the authors*.

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)

备注： 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 模块化仪表盘

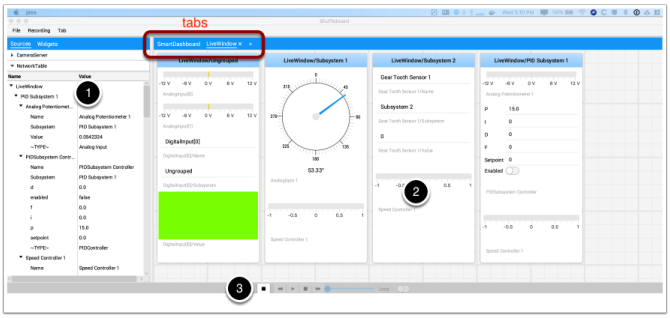
Shuffleboard is a straightforward and easily customizable driveteam focused dashboard. It displays network tables data using a variety of widgets that can be positioned and controlled with robot code. It includes many extra features like: tabs, recording / playback, and advanced custom widgets.

11.2.1 “Shuffleboard”-开始

浏览 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:

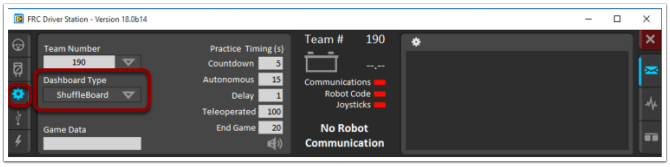
- 图形基于 Java 图形标准 ** JavaFX **。每个组件都有一个关联的样式表，因此有可能为 Shuffleboard 使用不同的“皮肤”或“主题”。我们提供默认的浅色和深色主题。
- Shuffleboard 支持 ** 多页显示数据 **。实际上，您可以创建一个新工作表（在 Shuffleboard 窗口中显示为选项卡），并指示是否以及应该在哪些上自动填充哪些数据。默认情况下，有一个“测试”选项卡和一个“智能仪表板”选项卡，它们会在数据到达时自动填充。其他选项卡可能用于机器人调试与驾驶。
- 图形化的 **display elements (widgets) are laid out on a grid**，以保持界面清洁并易于阅读。您可以更改网格大小以在布局中获得或多或少的分辨率，并且提供了视觉提示来帮助您使用拖放来更改布局。或者，即使保留了网格布局，也可以选择关闭网格线。
- 当您再次运行 shuffleboard 时，默认情况下会保存布局并实例化先前的布局。
- 有一项 **record and playback** 功能，可让您在机器人程序完成后查看其发送的数据。这样，如果出现问题，您可以仔细检查机器人的动作。
- **Graph widgets are available for numeric data**，您可以将数据拖动到图形上，以同时查看相同大小的多个点。
- 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. **Sources area:**这是数据源，您可以从 NetworkTables 或其他源中选择值，方法是将值拖到选项卡之一中以显示

2. **Tab panes:**. 这是您从机器人或其他来源显示数据的地方。在本示例中，此处显示在 LiveWindow 选项卡中的是测试模式子系统。该区域可以显示任意数量的选项卡式窗口，并且每个窗口都有其自己的属性集，例如网格大小和自动填充。
3. **Record/playback controls:**一组类似媒体的控件，您可以在其中播放当前会话以查看历史数据

启动沙狐球



您可以通过以下四种方式之一启动 Shuffleboard：

1. 您可以通过在设置标签中将”Dashboard Type”设置为 Shuffleboard 来在 Driver Station 启动时自动启动它，如上图所示。
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. 您可以通过双击🄴/ WPILib / YYYY / tools /中的 shuffleboard.XXX 文件（Windows 上的 XXX 是.vbs，Linux 或 macOS 上的.py）来运行它。‘（其中 YYYY 是年份，在 Windows 上🄴是 “C: Users Public’’）。这对于未安装 Driver Station 的开发系统例如 macOS 或 Linux 系统很有用。
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.

备注： .vbs（Windows）和.py（macOS / Linux）脚本有助于使用正确的 JDK 启动工具。

将机器人数据放到仪表板上

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());
```

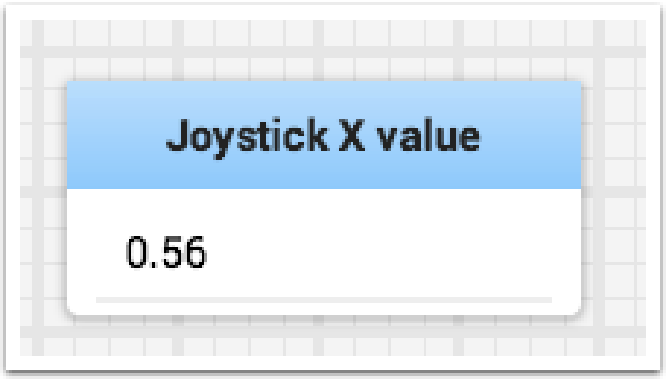
C++

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

PYTHON

```
from wpilib import SmartDashboard

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

看到带有标签“操纵杆 X 值”和操纵杆 X 值的显示的字段。每次执行此行代码时，都会将一个新的操纵杆值发送到 Shuffleboard。请记住：每当要查看更新的值时，都必须写入操纵杆值。在程序开始时执行该行一次将仅在执行代码行时显示一次该值。



显示来自机器人的数据

您的机器人可以在诸如遥测阶段和自动阶段模式之类的常规操作模式下显示数据，但是当机器人切换到“测试”模式时，您还可以显示状态并操作所有机器人子系统。默认情况下，启动 Shuffleboard 时，您会看到两个标签，一个用于遥测姐阶段 / 自动阶段，另一个用于测试模式。如下面的图片所示，当前选择的选项卡带有下划线。

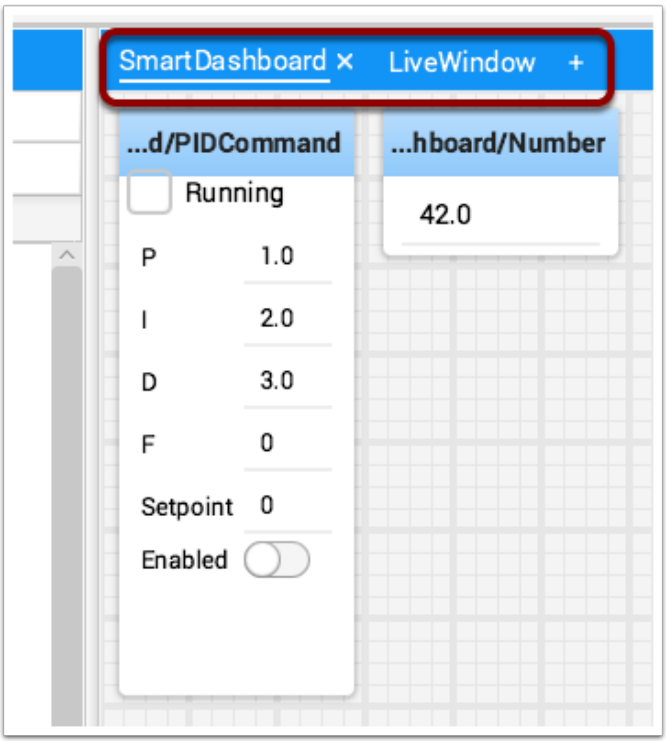
通常，调试或监视机器的状态需要将许多数值写入控制台，并观察它们的流向。使用 Shuffleboard，您可以将数值放到基于程序自动构建的 GUI 中。随着数值的更新，相应的 GUI 元素也会更改值-无需尝试捕获屏幕上流式传输的数字。

在正常操作模式下显示数值（自动阶段或遥测阶段）

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());
```

(续下页)



(接上页)

```
   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());
```

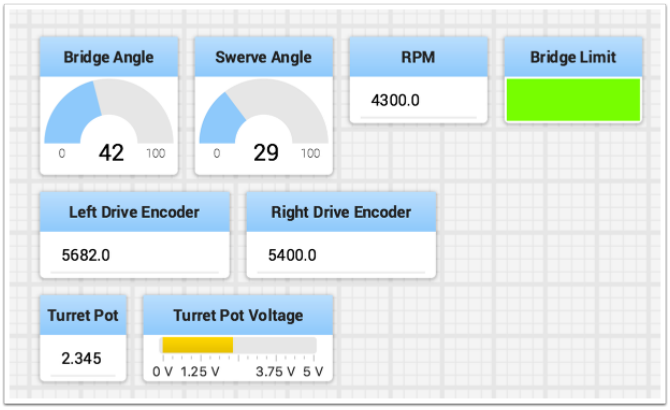
PYTHON

```
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())
```

(续下页)

(接上页)

```
SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage())
SmartDashboard.putNumber("RPM", shooter.getRPM())
```

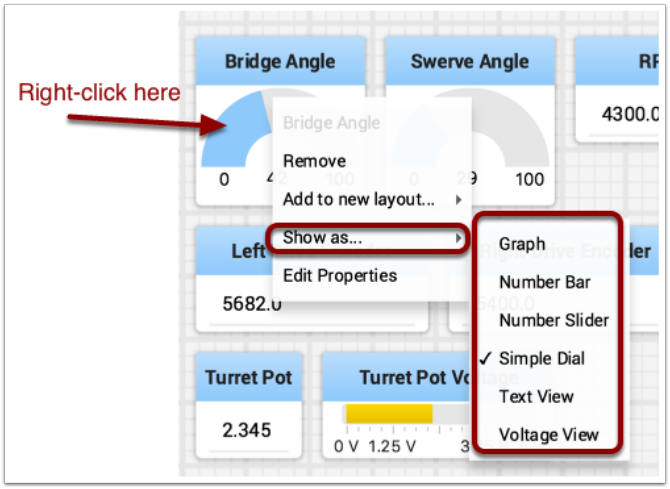


您可以通过简单地为类型调用正确的方法，包括数据的名称和值，将布尔值，数字值或字符串值写入 Shuffleboard，而无需其他代码。

- 字符类型，例如 char，int，long，float 或 double 调用 SmartDashboard.putNumber（“dashboard-name”，value）。
- 字符串类型调用 SmartDashboard.putString（“dashboard-name”，value）
- 布尔类型调用 SmartDashboard.putBoolean（“dashboard-name”，value）

更改数据的显示类型

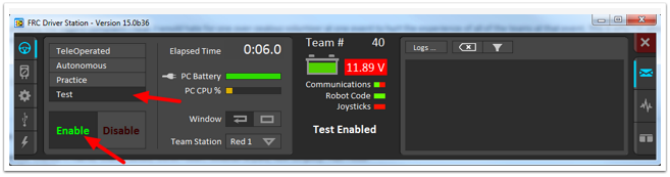
根据发送到 Shuffleboard 的值的数据类型，您通常可以更改显示格式。在前面的示例中，您可以看到数字值显示为十进制数字，更好地表示角度的刻度盘以及转塔电位计的电压视图。要设置显示类型，请在图块上单击鼠标右键，然后选择“显示为…”。您可以从弹出菜单的列表中选择显示类型。



在测试模式下显示数据

您可以将代码添加到程序中，让机器人在处于“测试”模式时显示传感器和执行器的值。只要机器人不在现场，都可以从机器操控台中进行选择。显示这些值的代码由 RobotBuilder 自动生成或手动添加到您的程序中，并在下一篇文章中进行介绍。测试模式旨在验证机器人上传感器和执行器的正确操作。此外，它还可以用于从电位计等传感器获取设定值，以及在代码中调整 PID 回路。

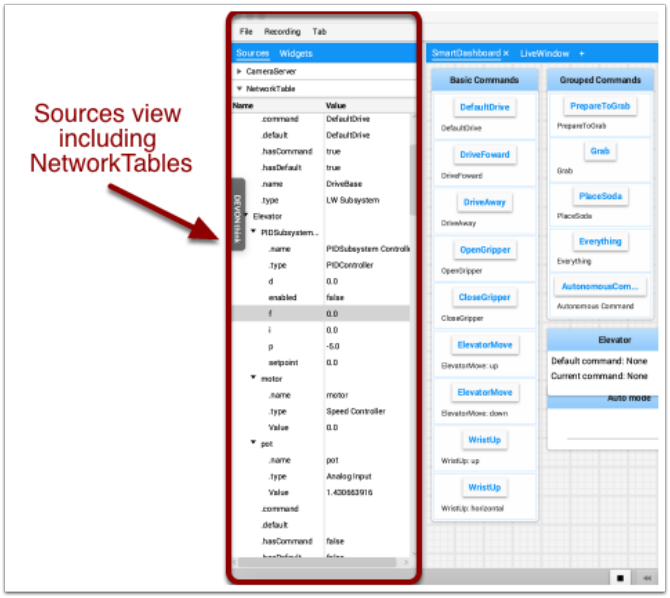
设定测试模式



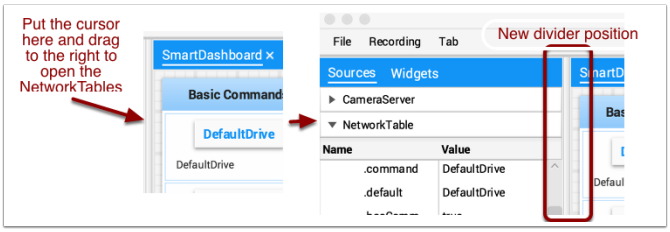
通过单击“测试”按钮并在机械手上设置“启用”，在机器操控台中启用测试模式。执行此操作时，Shuffleboard 将显示由子系统组织的程序所使用的任何执行器和传感器的状态。

从 Sources 中获取数据

通常*NetworkTables* 数据会自动出现在其中一个选项卡上，您只需重新排列和使用该数据。有时您可能想要恢复从选项卡中意外删除的值或显示不属于 SmartDashboard / NetworkTables 键的值。对于这些情况，可以将值从窗口左侧的 Sources 下的 NetworkTables 视图拖到窗格上。选择要显示的值并将其拖到窗格中，它将使用该数据类型的默认小部件类型自动创建。



备注： 有时左边的 Sources 视图不可见——可以拖动选项卡窗格和源之间的分隔符，这样源就不可见了。如果发生这种情况，将光标移到左边边缘，寻找调整大小的分隔符光标，然后左键单击并拖出视图。在下面的两张图片中，你可以看到点击和拖动的位置，完成后的分割线如图 2 所示。

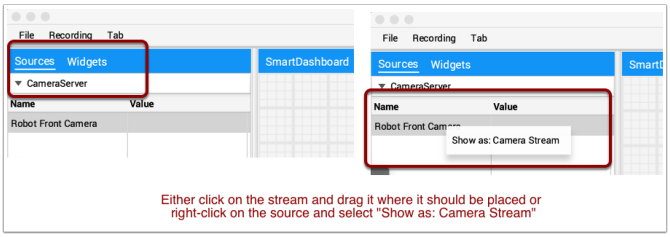


显示相机流

可以在 Shuffleboard 中的选项卡上查看来自机器人的摄像机流。这对于查看机器人所看到的东西很有用，为操作员提供拥有较少阻碍的视野，或有助于机器操控台中计算机或机器人上的协处理器中视觉算法的可视化输出。可以在相机流小部件中查看使用 CameraServer API 运行的任何信息流。

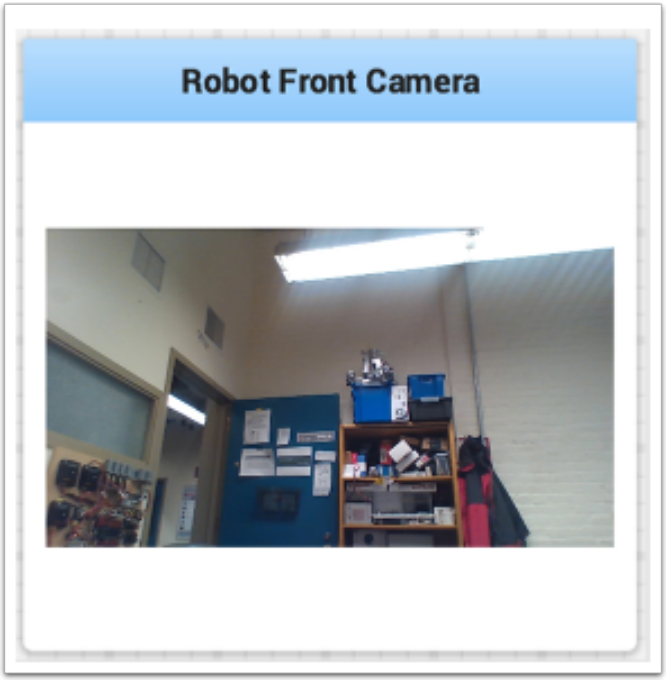
添加相机流

要将摄像机添加到仪表板，请选择“源”，然后在 Shuffleboard 窗口的左侧面板中查看“CameraServer”源，如下例所示。摄像机流列表将会显示，在这个例子中，只有一个被称为“Robot Front Camera”的摄像机。将其拖到应该显示它的选项卡上。或者，也可以通过右键单击“源”列表中的流，然后选择“显示为：摄像机流”，将该流放置在仪表板上。



添加摄像机流后，它将显示在窗口中。它可以被调整大小并移动到所需位置。

备注： 请注意，以太高分辨率或太高的帧速率发送太多数据将导致 roboRIO 和笔记本电脑上的 CPU 使用率过高。

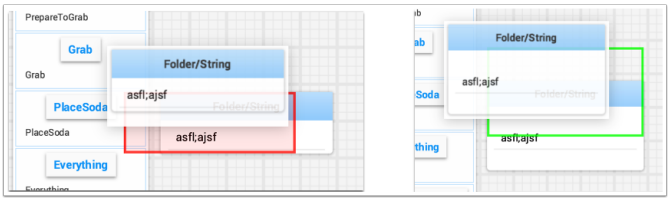


与小部件一起工作

在 Shuffleboard 屏幕上操作的可视化显示称为 widgets。Widgets 通常是由机器人程序通过 NetworkTables 发布的值自动显示的。

移动小部件

只需拖放即可移动小部件。只需将光标移到小部件上，单击鼠标左键并将其拖动到新位置。拖动时，你只能将小部件放置在网格正方形上，并且网格的大小会影响显示的分辨率。拖动时，将显示红色或绿色轮廓。绿色通常表示当前位置有足够的空间来放置小部件，而红色通常表示它将重叠或太大而无法放置。在下面的示例中，窗口小部件被移动到不合适的位置。



调整小部件的大小

可以通过单击并拖动窗口小部件图像的边缘或角落来调整窗口小部件的大小。当光标位于调整小部件大小的正确位置时，它将变为调整大小光标。与移动窗口小部件一样，将绘制绿色或红色轮廓，指示窗口小部件可以调整大小。下面的示例显示了一个窗口小部件已调整为更大的区域，绿色轮廓指示周围的窗口小部件没有重叠。

更改小部件的显示类型

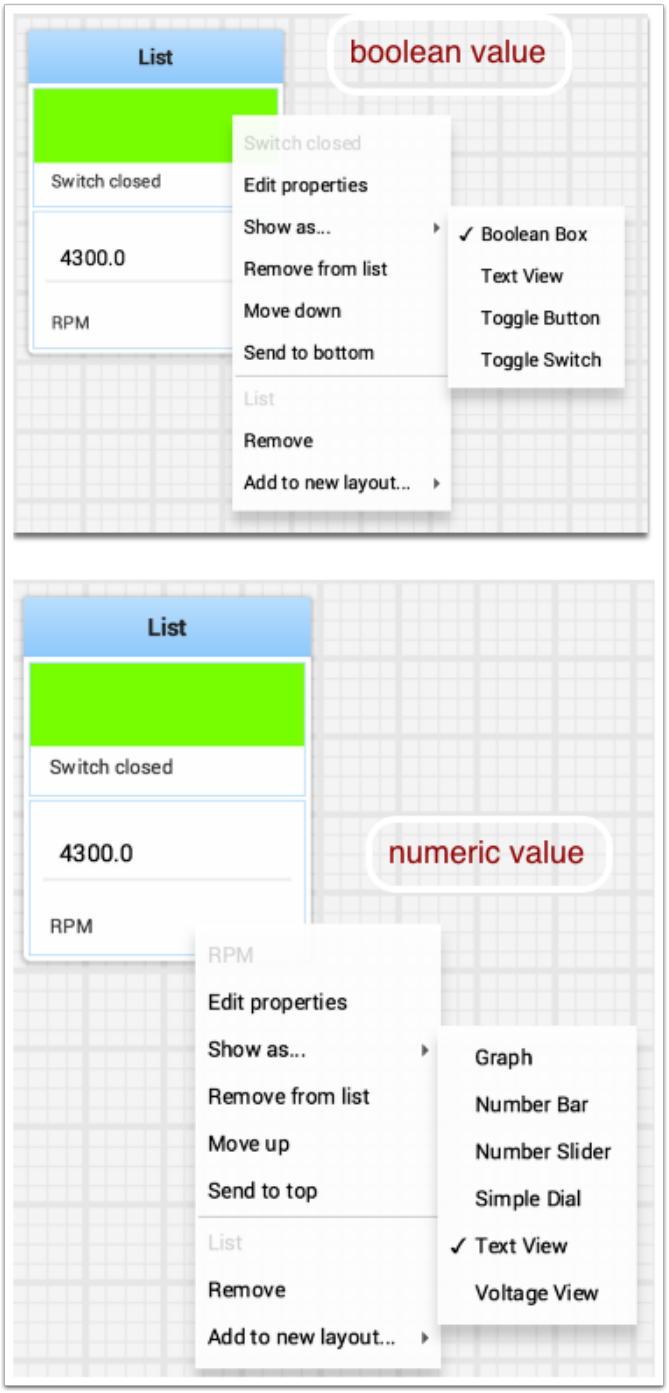
Shuffleboard 的显示类型非常丰富，具体取决于机器人发布的数据。它将自动选择默认显示类型，但是你可能需要根据应用程序进行更改。要查看任何窗口小部件可能显示的内容，请右键单击窗口小部件，然后选择“显示为…”，然后从弹出菜单中选择所需的类型。在下面的示例中，有两个数据值，一个为数字，另一个为布尔值。你可以看到每种显示选项可用的不同类型。布尔值只有两个可能的值（是/否），可以显示为布尔框（红色/绿色），文本，切换按钮或切换开关。的。数字值可以显示为图形，数字栏，数字滑块，刻度盘，文本或电压视图，具体取决于该值的上下文。

更改小部件的标题

你可以通过双击小部件的标题栏并将其标题编辑为新值来更改小部件的标题。如果布局中包含窗口小部件，则右键单击窗口小部件并选择属性。从那里您可以更改显示的窗口小部件标题。

更改小部件属性

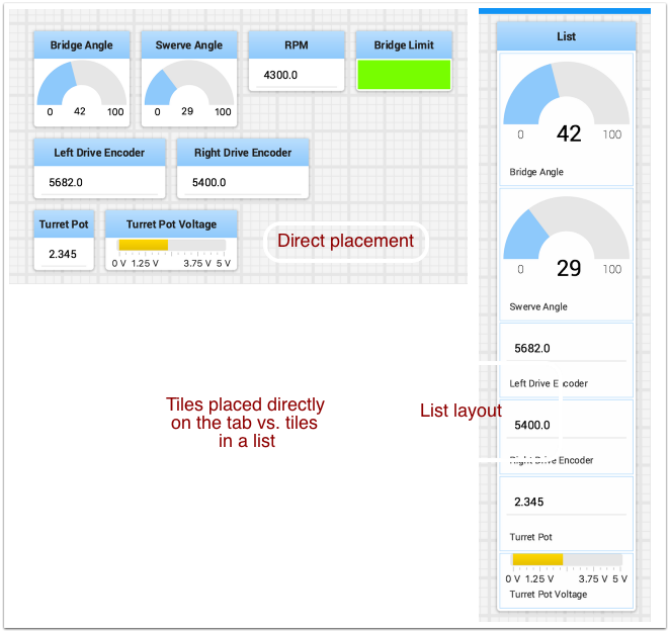
你可以更改小部件的外观，例如所表示的值的范围，颜色或其他视觉元素。在可能的情况下，右键单击窗口小部件，然后从弹出菜单中选择“编辑属性”。在下面显示的布尔值小部件中，小部件标题，真彩色和假彩色都可以编辑。


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

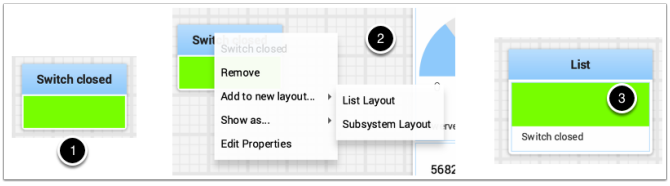
使用清单

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.



建立清单



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.

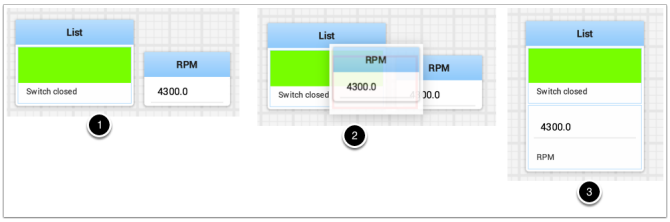
Adding tiles to/removing tiles from a list

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:

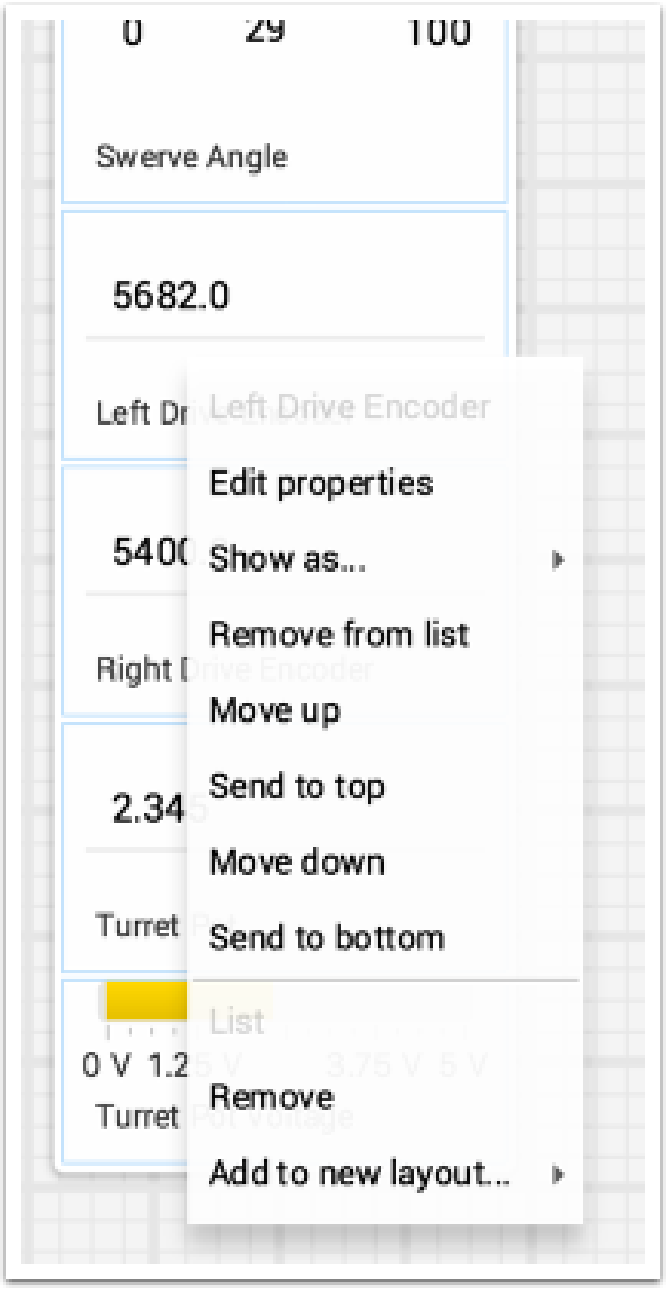
1. Identify the list and the tile within it to be removed.
2. Drag the tile out of the list and place it anywhere with free space.
3. The tile will be removed from the list and placed at that location.



重新排列列表中的图块

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

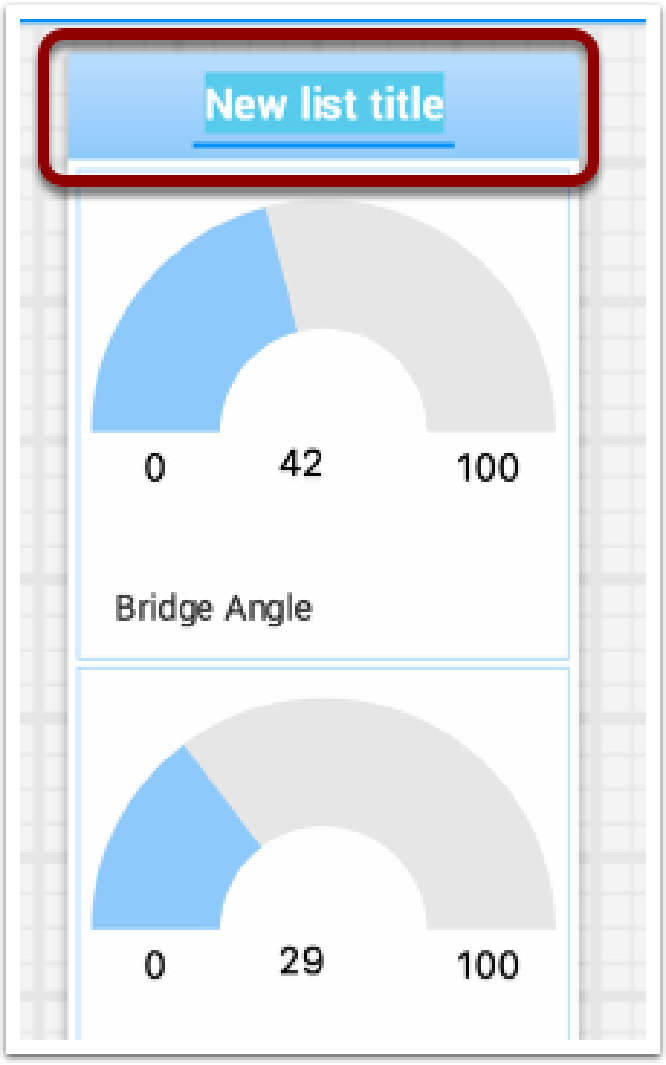
- *Move up* moves the tile **towards** the *top* of the list.
- *Move down* moves the tile **towards** the *bottom* of the list.
- *Send to top* moves the tile **to** the *top* of a list.
- *Send to bottom* moves the tile **to** the *bottom* of a list.


0
29
100
Swerve Angle
5682.0
Left Dr
Left Drive Encoder
Edit properties
540(
Show as...
Remove from list
Right
Move up
Send to top
2.34
Move down
Turret
Send to bottom
List
0 V 1.2
Remove
Turret
Add to new layout...

- 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*.

重命名清单

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

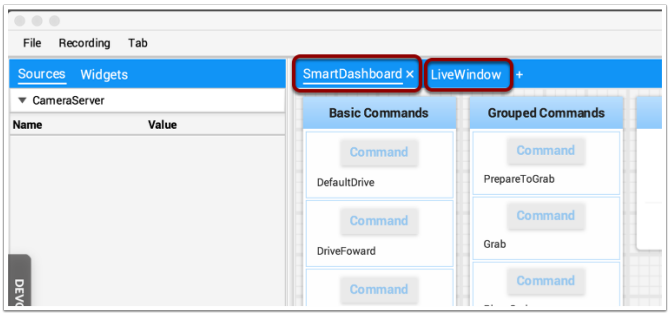


创建和操作标签

Shuffleboard 使用的选项卡式布局可帮助分隔机器人数据的不同“视图”并使显示更加有用。您可能有一个选项卡，其中有用于帮助调试机器人程序的显示，还有一个用于比赛的其他选项卡。有许多选项使选项卡非常强大。您可以使用本文后面描述的自动填充选项来控制每个选项卡中出现的来自 NetworkTables 或其他来源的数据。

默认标签

首次打开 Shuffleboard 时，有两个选项卡，分别标记为 SmartDashboard 和 LiveWindow。这些对应于 SmartDashboard 的两个视图，具体取决于您的机器人是在自主/远程运行还是在测试模式下运行。在沙狐球中，这两个视图随时都可用。



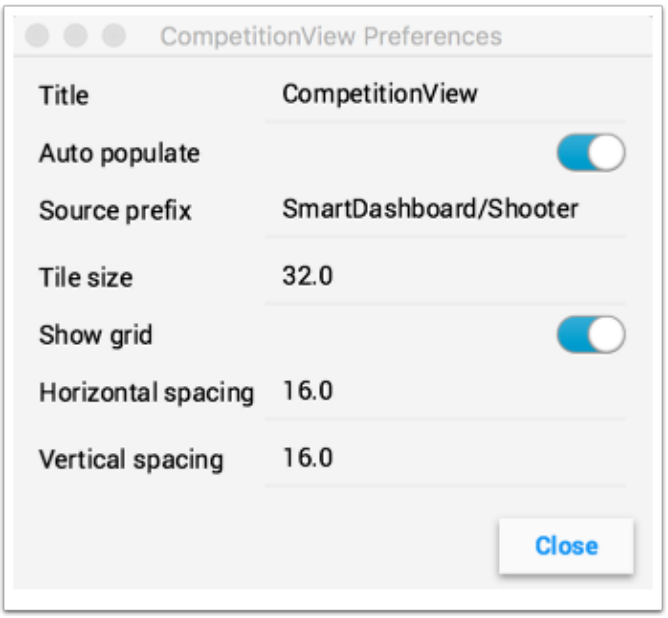
在 SmartDashboard 选项卡上，使用 SmartDashboard.putType () 方法集写入的所有值。在“LiveWindow”选项卡上，将显示所有自动生成的调试值。

在标签之间切换

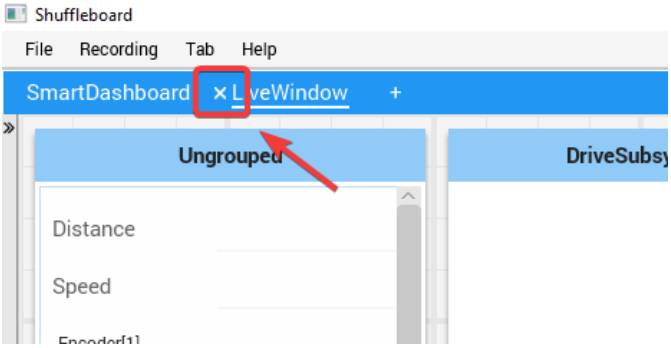
单击窗口顶部的标签标签，可以在标签之间切换。在上述情况下，只需单击 SmartDashboard 或 LiveWindow 即可查看与每个选项卡关联的值。

添加和隐藏选项卡

您可以通过单击最后一个选项卡右侧的加号（+）来添加其他选项卡。创建新选项卡后，可以通过双击选项卡中的标签并进行编辑来设置标签。您也可以右键单击选项卡，或使用选项卡菜单调出选项卡首选项，然后在该窗口中通过编辑标题字段来更改名称。



您可以通过单击所选选项卡名称左侧的减号（-）来隐藏选项卡。由于选项卡是基于相关的:term:*NetworkTable* 生成的，所以不可能在不删除表的情况下永久删除它们。



将标签设置为自动填充

选项卡最强大的功能之一是让它们根据选项卡“首选项”窗格中提供的源前缀自动填充新值。在上面的示例中，“首选项”窗格的“源”前缀为“SmartDashboard / Shooter”，并且“自动填充”处于打开状态。使用 SmartDashboard 类编写的任何指定 Shooter 子项的值都将自动显示在该选项卡上。注意：与多个 Source 前缀匹配的键将显示在两个选项卡中。由于这些键也以 SmartDashboard /开头，并且是默认 SmartDashboard 选项卡的 Source 前缀，因此这些小部件将同时出现在两个窗格中。若要仅在一个窗格中显示值，则可以使用 NetworkTables 编写标签和值，并使用不在 SmartDashboard 下的其他路径。另外，您可以让所有内容都显示在 SmartDashboard 选项卡中，使其变得很混乱，但是可以根据需要设置特定的选项卡，以便更好地进行过滤。

使用标签格和间距

每个选项卡都可以有自己的 Tile 大小 (每个大方块的像素个数)。因此，一些选项卡可能有更粗的分辨率，以便更容易的布局，另一些选项卡可能有一个精细的网格。Tab 偏好中的 Tile 大小复盖了 Shuffleboard 偏好中的任何全局设置。此外，还可以指定小部件中绘图与小部件边缘之间的填充。如果编写用户界面，这些参数通常被称为水平和垂直间隙 (hgap，vgap)。

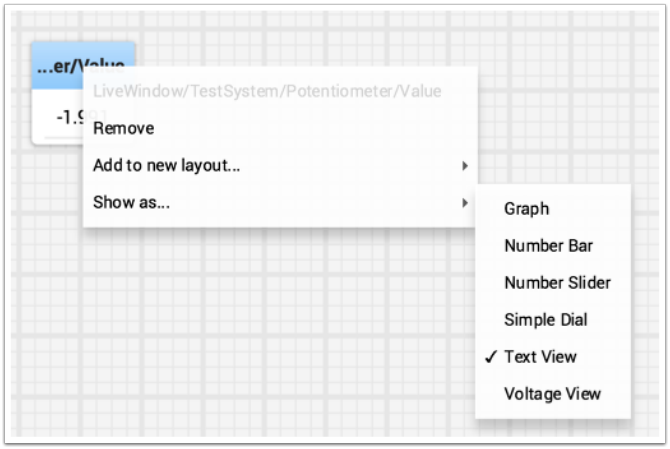
在选项卡之间移动小部件

当前，没有一种方法可以轻松地在各个选项卡之间移动小部件，而无需将其从一个选项卡中删除并将该字段从左侧的源层次结构拖动到新窗格中。我们希望在以后的更新中具有该功能。

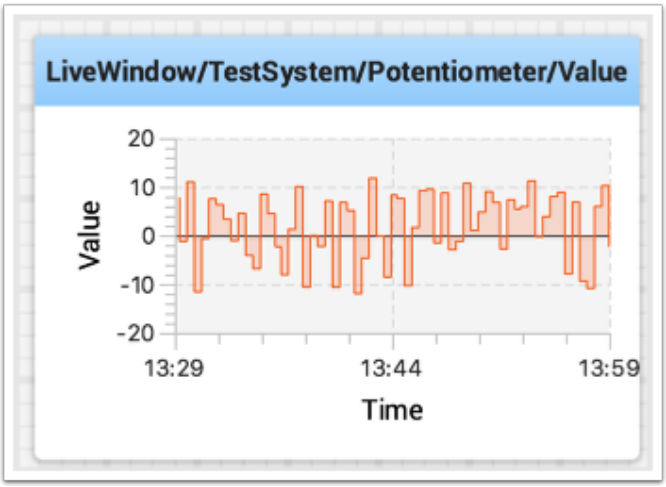
使用图像

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.

要创建图形，选择一个数值，右键单击标题，选择“显示为…”，然后选择“图形”

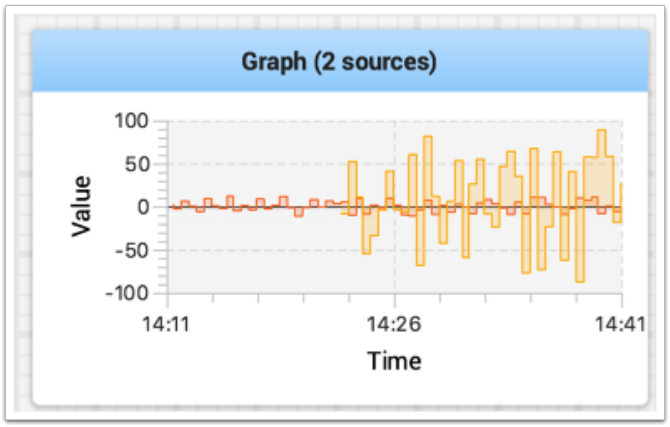


图像小部件显示您选择的值的线图。它将自动设置比例，图形将显示的默认时间间隔将为 30 秒。您可以在图的设置中更改这一点 (见下面)。

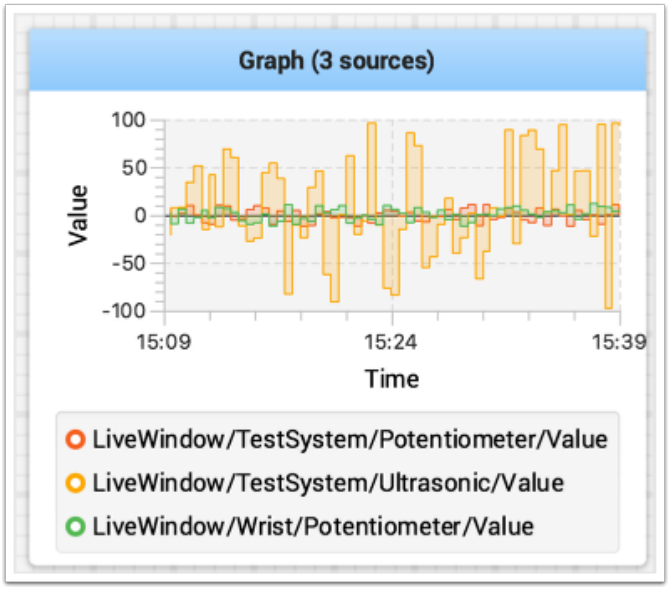


添加额外数值

对于相关值，通常需要在同一图表上显示多个值。为此，只需从 NetworkTables 源视图（Shuffleboard 窗口的左侧）拖动其他值并将其放到图表上，该值将如下所示添加。您可以继续将其他值拖到图形上。



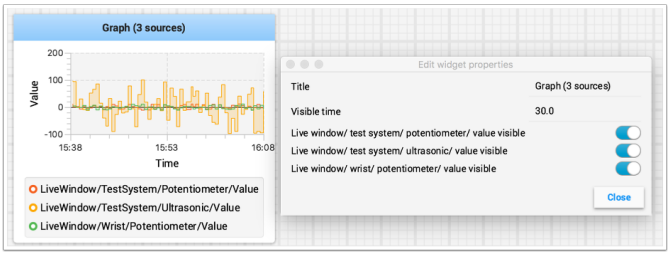
如果图例没有如下图所示显示，您可以垂直调整图形的大小以查看图例。图例显示了情节中使用的所有来源。



设置图像性质

您可以通过更改图像小部件属性中的“可见时间”来设置在图像中显示的秒数。要访问属性，右键单击图形并选择“Edit properties”。

除了设置可见时间之外，图还可以通过打开和关闭属性窗口中显示的每个源的开关，有选择地打开和关闭源(见下文)。

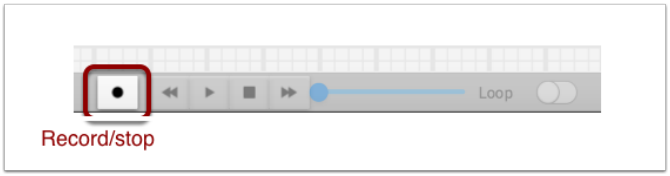


录音和播放

Shuffleboard 可以记录会话期间的所有小部件更新。之后，可以“播放”日志文件，以查看比赛或练习运行期间发生的情况。如果在比赛期间某些东西未按预期运行并且您想查看发生了什么，这将特别有用。每个记录都记录在一个记录文件中。

创建录音

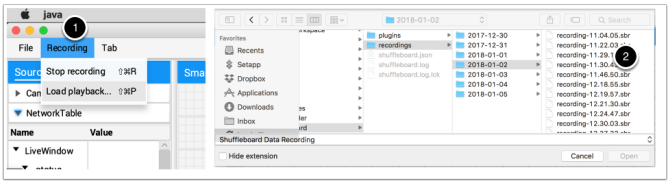
沙狐球启动时，它开始记录所有的 NetworkTables 值，并通过按下记录器控件中的“记录/停止”按钮继续直至停止，如下所示。如果需要新的记录，例如正在测试新的代码段或机械系统时，请停止当前记录（如果正在运行），然后单击“记录”按钮。再次单击该按钮可停止录制并关闭录制文件。如果按钮是圆形的（如图所示），则单击它开始录制。如果该按钮是正方形，则表示当前正在运行记录，因此单击它可以停止录制。



播放录音

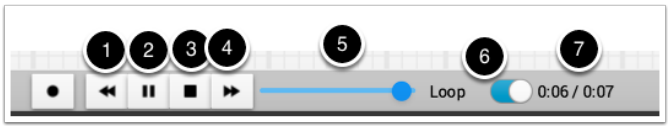
以前的录音可以通过以下方式播放：

1. 选择“录制”菜单，然后单击“加载播放”。
2. 从显示的目录中选择一个录音。记录按日期分组，文件名是记录时间，以帮助识别正确的记录。从列表中选择正确的记录。



控制播放

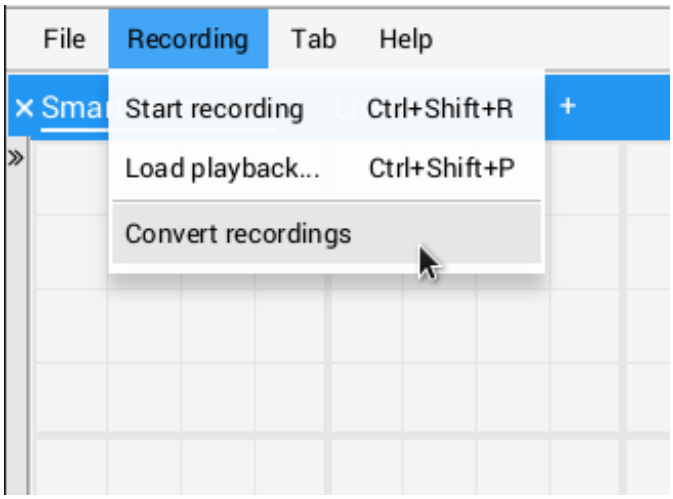
选择记录文件将开始播放该文件。在播放录音时，录音控件将显示录音中的当前时间，以及在观看时循环播放录音的选项。回放录音时，“传输”控件将允许对回放进行控制。



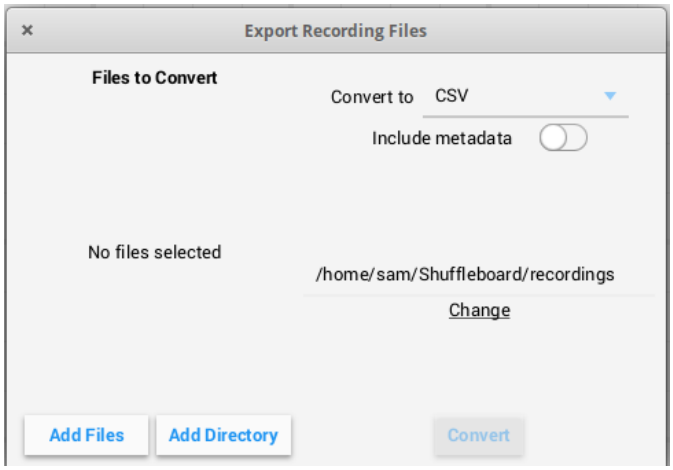
控件的工作方式如下：

1. 左箭头键将回放备份到最后更改的数据点
2. 播放/暂停控件开始和停止播放
3. 方形停止按钮停止播放并恢复显示当前机器人值
4. 右侧的双箭头向前跳到下一个更改的数据值
5. 滑块允许直接定位到任何时间点以查看记录的不同部分
6. 循环开关打开播放循环，即播放将一遍又一遍直到停止
7. 时间显示记录中的当前时间点和记录的总时间

转换为不同的文件格式



沙狐球录像采用自定义二进制格式以提高效率。要分析记录的数据而无需通过应用程序回放，Shuffleboard 支持数据转换器将记录转换为任意格式。该应用程序仅随附一个简单的 CSV 转换器，但团队可以编写自定义转换器并将其包含在 Shuffleboard 插件中。



多个记录可以一次转换。可以使用“添加文件”按钮选择单个文件，或者可以通过“添加目录”按钮一次选择目录中的所有录制文件。

转换后的录音将在“🄴/ Shuffleboard / recordings”目录中生成，但可以使用“更改”按钮手动选择。

可以使用右上角的下拉菜单选择不同的转换器。默认情况下，仅 CSV 转换器可用。来自插件的自定义转换器将在下拉列表中显示为选项。

补充说明

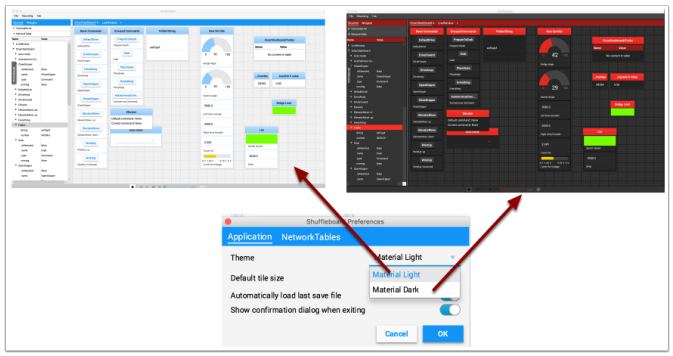
清理时间轴时，图形将无法正确显示，但是如果正在播放的时间通过图形可以捕获图形历史记录，则它们将以原始运行方式显示。

设置 Shuffleboard 的全局首选项

有许多设置可设定 Shuffleboard 的外观和行为方式。这些设置位于“Shuffleboard Preference”窗格中，可以从“文件”菜单访问。

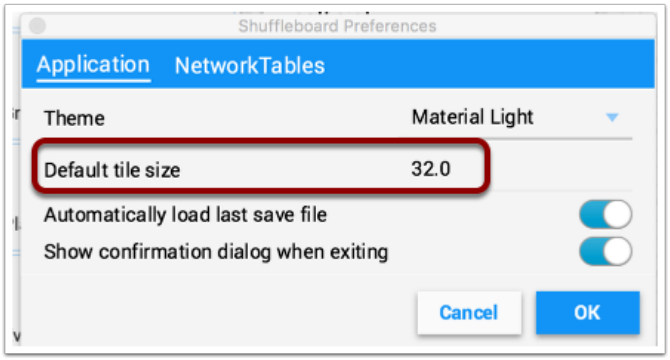
设定主题

Shuffleboard supports two themes, Material Dark and Material Light and the setting depends on your preferences. This uses css styles that apply to the entire application and can be changed any time.



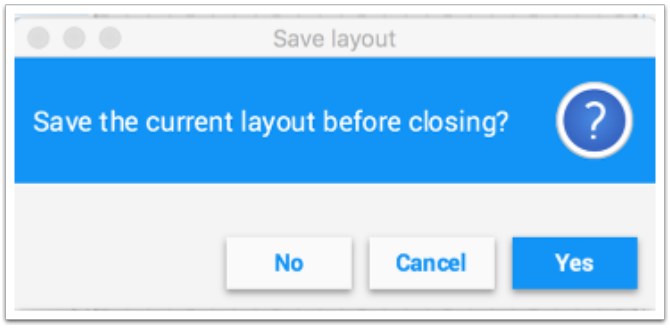
设置默认图块大小

当您自己添加或移动它们或自动填充磁贴时，Shuffleboard 会将它们放置在网格上。您可以为每个选项卡设置默认磁贴大小，也可以更改默认设置后为创建的所有选项卡设置默认磁贴大小。网格中更精细的分辨率可以更好地控制瓷砖的放置。这可以在 Shuffleboard 首选项窗口中进行设置，如下所示。

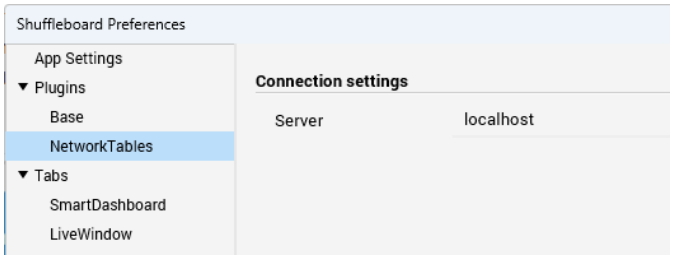


使用布局保存文件

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.



设定团队编号



为了使 Shuffleboard 能够在机器人上找到 NetworkTables 服务器，请在"首选项"窗格的"NetworkTables"选项卡中指定团队编号。如果您用在机器操控台上运行 Shuffleboard，则 Server 字段将自动

填充正确的信息。如果您在没有机器操控台的计算机上运行 Shuffleboard ，则可以手动输入团队编号或 robotRIO 网络地址。

Shuffleboard 常见问题，问题和错误

> **警告：** Shuffleboard 以及大多数其他控制系统组件是用 Java 11 开发的，不会在 Java 8 中工作。在报告问题之前，请确保您的计算机安装了 Java 11，并将其设置为默认 Java 环境。

常见问题

我如何用 Shuffleboard 报告问题，错误或功能请求？

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.

我如何添加自己的小部件或其他扩展到 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.

我如何从源代码构造 Shuffleboard？

你可以在 GitHub 网站上下载、克隆或分叉存储库来获得源代码。要从源代码构建和运行 Shuffleboard，请确保当前目录是顶级源代码，并使用以下命令之一:

应用	指令 (对于 Windows 系统运行 gradlew.bat 文件)
运行 Shuffleboard	./gradlew :app:run
为插件创建构建 api 和实用程序类	./gradlew :api:shadowJar
构建完整的应用程序 jar 文件	./gradlew :app:shadowJar

11.2.2 布局与代码

使用标签

沙狐球, 推移板游戏是一个选项卡式界面。每个选项卡将小部件按逻辑分组进行组织。默认情况下，Shuffleboard 具有用于旧版 SmartDashboard 和 LiveWindow 的选项卡-但是现在可以直接从机器人程序中在 Shuffleboard 中创建新的选项卡，以实现更好的组织。

创建一个新标签

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")
```

创建新选项卡就像在 Shuffleboard 类上调用单个方法一样简单，它将在 Shuffleboard 上创建一个新的选项卡，并返回一个将数据添加到选项卡中的句柄。多次调用具有相同标签标题的 getTab 将每次返回相同的句柄。

选择一个标签

JAVA

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

C++

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

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

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

此方法允许按标题选择选项卡。这是区分大小写的 (so “Tab Title” and “Tab title” are two individual tabs)，并且仅当在调用该方法时存在具有该标题的选项卡时才起作用，因此调用 “selectTab（“Example”）“仅在先前定义了名为 “Example” 的选项卡时才有效。

此方法可用于选择 Shuffleboard 中的任何选项卡，而不仅仅是由机器人程序创建的选项卡。

注意事项

从机器人程序中创建的选项卡与从仪表板中创建的正常选项卡有几个重要的区别:

- 未保存在 Shuffleboard 保存文件中
- 不支持自动填充
- 用户需要在他们的机器人程序中指定标签内容
- 具有特殊的颜色以区别于普通标签

传送资料

与 SmartDashboard 不同，数据不能直接发送到 Shuffleboard，而不需要首先指定数据应该放在什么选项卡。

发送简单数据

发送简单数据（数字，字符串，布尔值和它们的数组）是通过在选项卡上调用“add”来完成的。如果尚不存在，则此方法将设置该值，但不会覆盖现有值。

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)
```

如果数据需要更新 (例如，在机器人上做一些计算的输出)，在定义值后调用”getEntry ( ) “，然后在需要时或在”周期性”函数中时更新

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)
```

重新启动之间的选择仍然存在

从仪表板配置机械人时，某些设置可能希望在机械人或驱动程序重新启动之间保留下来，而不是让驱动程序在每次匹配之前记住（或忘记）配置设置。

简单地使用“addPersistent”而不是“add”将使值保存在 roboRIO 上，并在机器人程序启动时加载。

备注： 这不适用于可发送的数据，例如选择器或电动机控制器。

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))
```

发送传感器，电机等

类似于“SmartDashboard．putData”，任何“Sendable”对象 (大多数传感器、电机控制器和 SendableChoosers) 都可以添加到任何标签中

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))
```

检索数据

与”SmartDashboard．getNumber “和朋友不同，从 Shuffleboard 检索数据也是通过我们在前一篇文章中提到的 NetworkTableEntries 完成的。

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)
```

这个基本示例有一个明显的缺陷：可以在仪表板上将最大速度设置为 [0，1] 之外的值-这可能导致输入饱和（始终以最大速度），甚至颠倒！幸运的是，有一种方法可以避免此问题-下一篇文章将对此进行介绍。

配置小部件

机器人程序可以准确指定要用于显示数据点的窗口小部件，以及应如何配置该窗口小部件。由于此处列出的小部件太多，请查阅文档以获取详细信息。

指定小部件

Call `withWidget` after `add` in the call chain:

JAVA

```
Shuffleboard.getTab("Drive")
   .add("Max Speed", 1)
   .withWidget(BuiltInWidgets.kNumberSlider) // specify the widget here
   .getEntry();
```

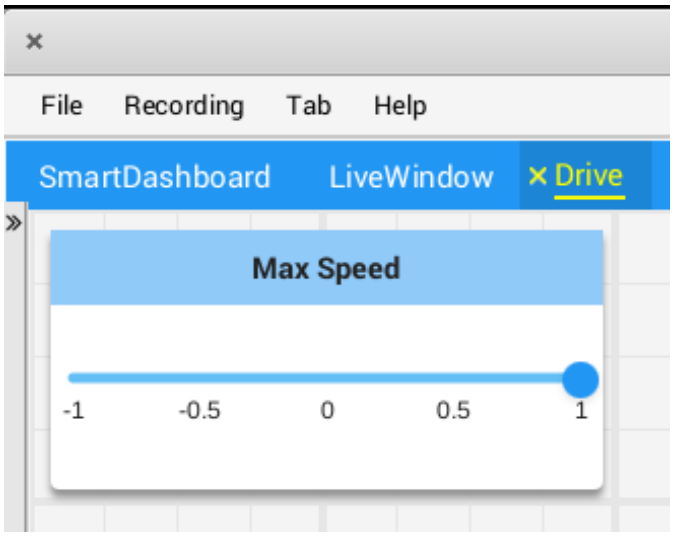
C++

```
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())
```

在此示例中，我们将“最大速度”窗口小部件配置为使用滑块而不是基本文本字段来修改值。



设置小部件属性

由于最大速度只有在 0 到 1（全停到全速）的一个值才有意义，因此如果值降至零以下，则从-1 到 1 的滑块可能会引起问题。幸运的是，我们可以使用 withProperties 方法修改它

JAVA

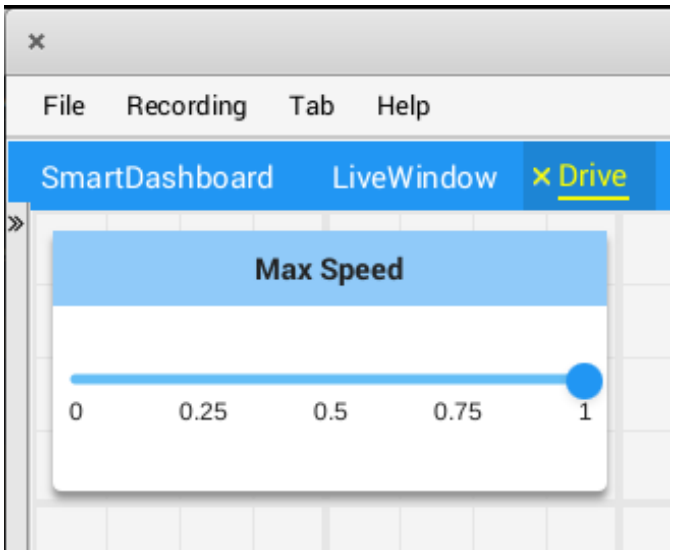
```
Shuffleboard.getTab("Drive")
   .add("Max Speed", 1)
   .withWidget(BuiltInWidgets.kNumberSlider)
   .withProperties(Map.of("min", 0, "max", 1)) // specify widget properties here
   .getEntry();
```

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())
```



笔记

小部件可以通过名称指定；但是，名称区分大小写和空格（“” Number Slider” 与 “Number slider” 和 NumberSlider” 不同）。为此，建议使用构建在 widget 类中的小部件来指定小部件，而不是使用原始名称。但是，只能通过名称或通过为该小部件创建自定义 “WidgetType’’ 来指定自定义小部件。

窗口小部件属性名称既不区分大小写也不区分空格（“Max” 和 “max” 相同）。有关该窗口小部件属性的详细信息，请参阅 BuiltInWidgets 类中的窗口小部件文档。

组织小部件

设置小部件的大小和位置

调用 “withSize” 和 “withPosition” 来设置窗口小部件在选项卡中的大小和位置。

“withSize” 设置小部件应有的列宽和行高的数量。例如，调用 ‘ withSize (1，1) ‘使得小部件在网格中占据一个单元格。注意一些小部件的最小尺寸可能大于指定尺寸，在这种情况下小部件将使用最小的支持尺寸。

withPosition 设置窗口小部件左上角的行和列。行和列都索引为 0，因此最上面的行是数字 0，最左边的列也是数字 0。如果指定了选项卡中任何窗口小部件的位置，则每个窗口小部件的位置也应设置为避免窗口小部件重叠。

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
```

将小部件添加到布局

如果选项卡中包含相关数据的小部件很多，那么将它们放置在较小的子组中而不是将其散布在选项卡中会很有用。就像使用 Shuffleboard.getTab 检索选项卡的句柄一样，可以使用 ShuffleboardTab.getLayout 检索选项卡内的布局（甚至在另一个布局中）。

JAVA

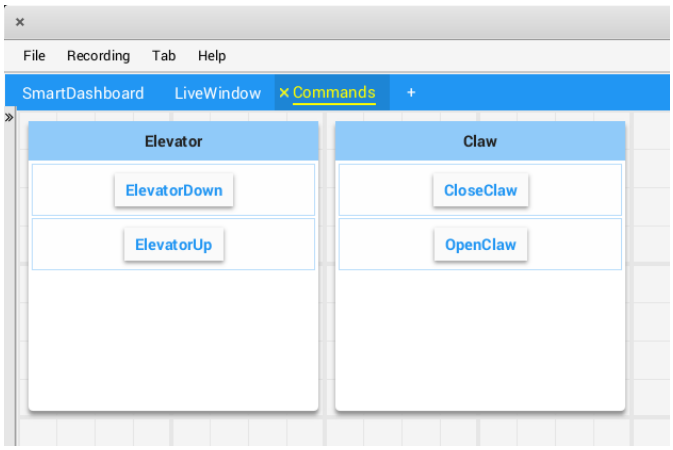
```
ShuffleboardLayout elevatorCommands = Shuffleboard.getTab("Commands")
  .getLayout("Elevator", BuiltInLayouts.kList)
  .withSize(2, 2)
  .withProperties(Map.of("Label position", "HIDDEN")); // hide labels for commands

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”-高级用法

指令和子系统

使用 command-based 架构时,” Shuffleboard” 可以通过实时显示各种命令和子系统的状态来帮助人们更轻松地了解机器人在做什么。

显示子系统

如果要在机器人以自动或遥控模式运行时查看其子系统的状态（即其默认命令是什么，以及当前正在使用该子系统的命令），请将子系统实例发送到 Shuffleboard：

JAVA

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

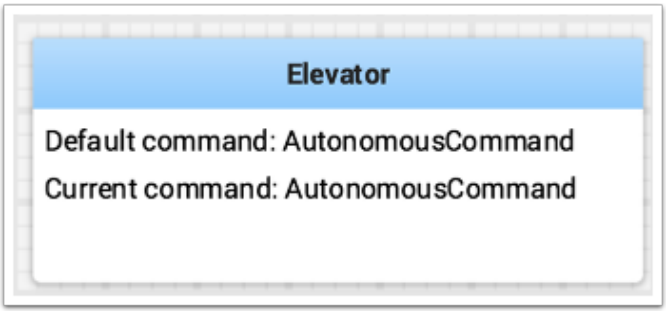
C++

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

PYTHON

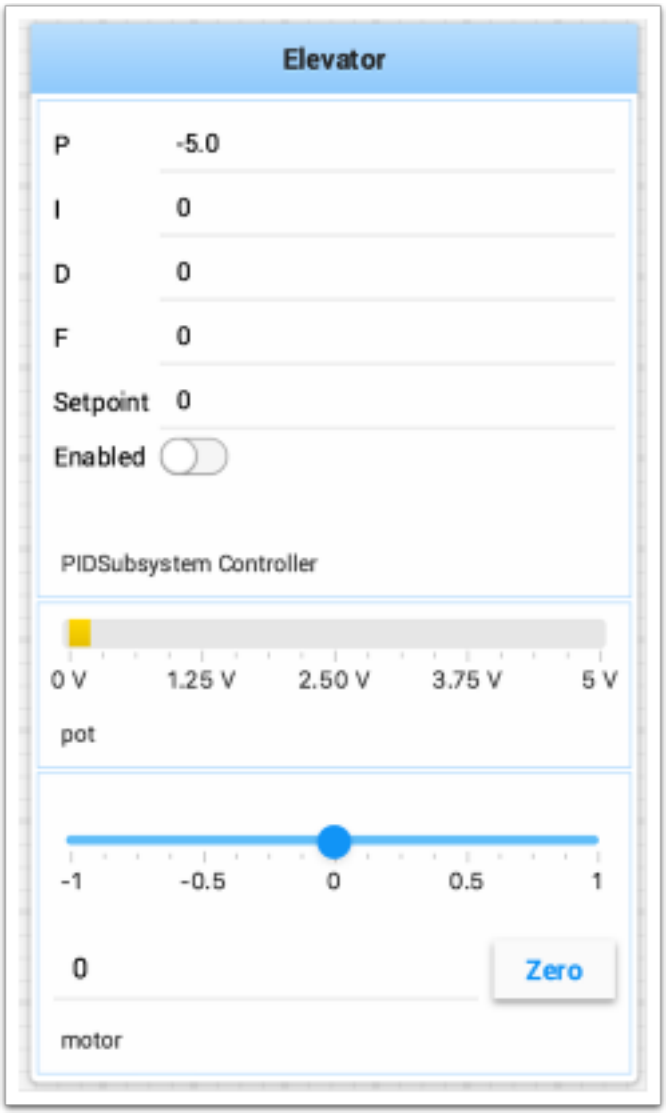
```
from wpilib import SmartDashboard

SmartDashboard.putData(subsystem-reference)
```

Shuffleboard 将显示子系统名称，与此子系统关联的默认命令以及当前正在运行的命令。在此示例中，Elevator 子系统的默认命令称为 “AutonomousCommand’’，它也是使用 Elevator 子系统的当前命令。



测试模式下的子系统

在“测试”模式（driver station 中的“Test/Enabled”）中，子系统可以与子系统的传感器和执行器一起显示在 LiveWindow 选项卡中。这有助于通过查看传感器返回的值来判断传感器是否正常工作。另外，可以使用 actuators。例如，可以使用滑块操作马达以设置其移动速度和方向。对于 PID 子系统，将显示 P，I，D 和 F 常数以及他们的设定值和启用控制器。这对于通过调整常量，输入设定值并启用嵌入式 PIDController 来调整 PIDSubsystems 来说很有用。这样就可以观察到该机制的响应。可以重复此循环（更改参数，启用和观察），直到找到合理的参数集为止。



有关 PIDSubsystems 的更多信息可以在这里找到：<shuffleboard-tuning-pid> 使用 RobotBuilder 将自动生成代码，以便在 Test 模式下显示子系统。需要显示子系统的代码如下所示，其中子系统名称是包含子系统名称的字符串：

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

显示命令

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));
```

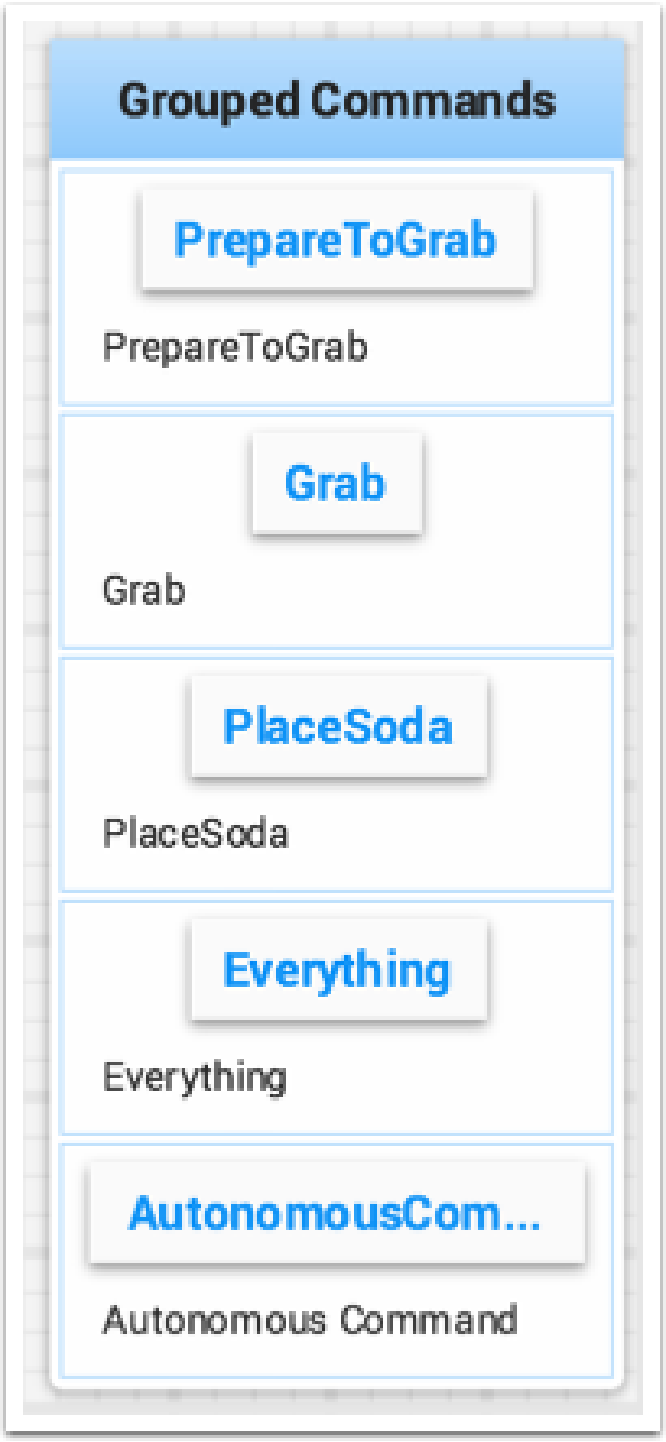
C++

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

PYTHON

```
from wpilib import SmartDashboard

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

Shuffleboard 将显示命令名称和一个用于执行命令的按钮。这样，可以轻松地测试单个命令和命令组，而无需在机器人程序中使用特殊的测试代码。在下图中，Shuffleboard 列表中包含许多命令。一次按下按钮将运行命令，再次按下将停止命令。要使用此功能，机器人必须开启遥控模式。


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

测试和调整 PID 循环

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:

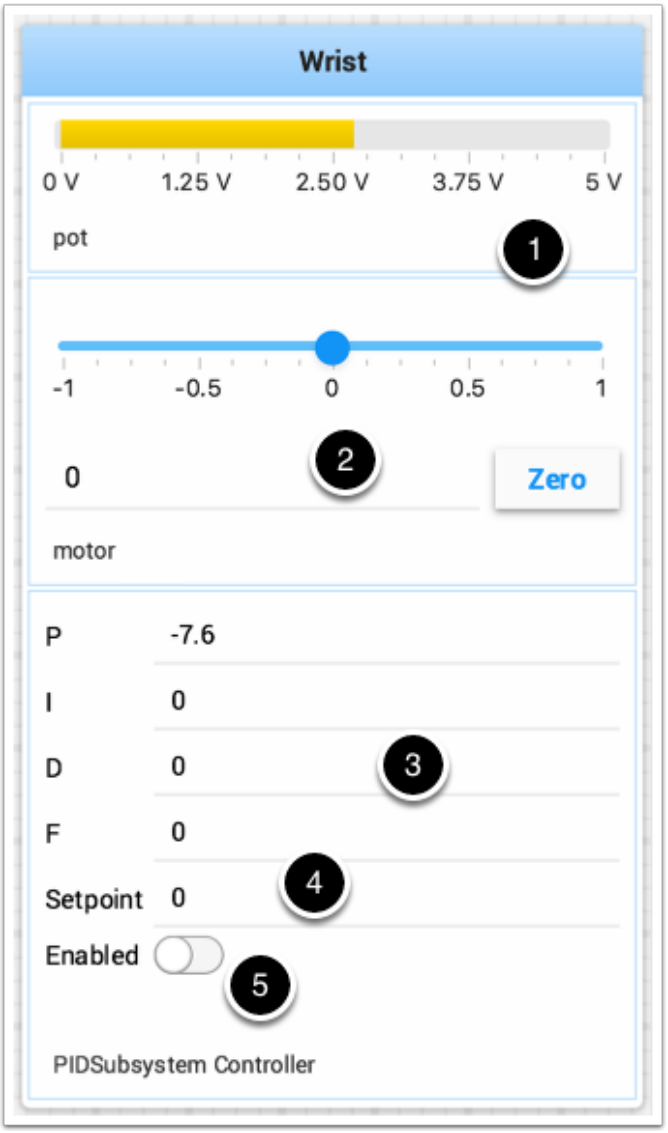
1. 读取传感器值以确定机器人或机械装置距所需设定值的距离。设定值是与预期目标相对应的传感器值。例如，带有腕关节的机械臂应该能够非常快速地移动到指定角度并停止在传感器指示的那个角度。电位计是可以测量的旋转角度的传感器，通过将其连接到模拟输入，程序可以获得与角度成正比的电压值。
2. 计算误差（传感器值与所需值之差）。错误值的符号指示腕关节位于设定点的哪一侧。例如，负值可能表示所测量的腕部角度大于所需的腕部角度。误差的大小是所测得的手腕角度与实际手腕角度之间的距离。如果误差为零，则测量角度与所需角度完全匹配。该误差可用作 PID 算法的输入，以计算电动机速度。
3. The resultant motor 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 具有一个 PIDController 类，该类实现 PID 算法并接受与 Kp，Ki 和 Kd 值相对应的常量。PID 算法具有三个部分，这些部分有助于根据误差计算电动机速度。

1. P（Proportional）- 这是一个术语，当某个值乘以常数（Kp）时将产生的电动机速度，从而有助于沿正确的方向和速度移动马达。
2. I（Integral）- 此术语所指的是连续错误的总和。错误存在的时间越长，积分贡献将越大。这只是一段时间内所有错误的总和。如果手腕由于要移动的负载而不能完全达到设定点，则积分项将继续增加（误差之和），直到它对电动机速度做出足够的贡献以使其移动至设定点。我们使用误差的总和乘以常数（Ki），以缩放系统的积分项。
3. D（Differential）- 此值是误差的变化率。如果电动机速度太快，则可以用其减慢电动机速度。我们通过计算当前误差值和先前误差值之间的差来得到此值。我们用其乘以常数（kd）来缩放它以匹配系统的其余部分。

调整 PID 控制器

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).



这是腕部子系统的测试模式图片，腕部子系统具有一个电位计作为传感器，以及一个与电机连接的电机控制器。它具有许多与 PID 子系统对应的子系统。

1. 电位计的模拟输入电压值。这是传感器输入值。
2. 一个滑动器以 0 为停止方向沿任一方向移动手腕马达。正值和负值将对应于向上或向下移动。
3. 如上所述的 PID 常数（F 是用于速度 PID 回路的前馈值）
4. 当手腕达到所需值时，对应于电位计值的设定值
5. 启用 PID 控制器-如果不起作用，请参见下文。

尝试各种 PID 增益以获得所需的电动机性能。您可以查看本文开头链接的视频或互联网上的其他资料来源，以获得所需的效果。

重要： 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.

在新的 PIDController 中启用功能

以下示例演示了如何在仪表板上创建一个按钮，该按钮将启用/禁用 PIDController。

JAVA

```
ShuffleboardTab tab = Shuffleboard.getTab("Shooter");
GenericEntry shooterEnable = tab.add("Shooter Enable", false).getEntry();

// Command Example assumed to be in a PIDSubsystem
new NetworkButton(shooterEnable).onTrue(new InstantCommand(m_shooter::enable));

// Timed Robot Example
if (shooterEnable.getBoolean()) {
  // Calculates the output of the PID algorithm based on the sensor reading
  // and sends it to a motor
  motor.set(pid.calculate(encoder.getDistance(), setpoint));
}
```

C++

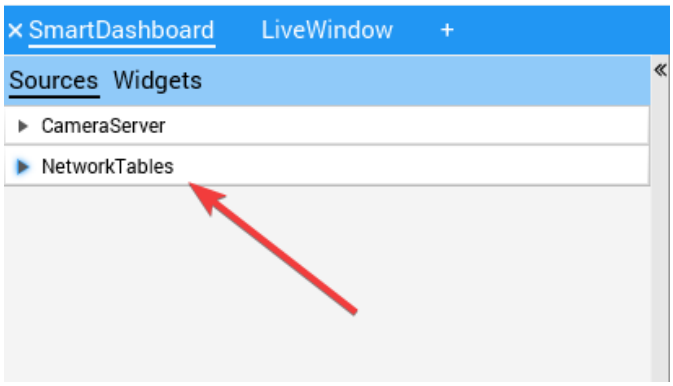
```
frc::ShuffleboardTab& tab = frc::Shuffleboard::GetTab("Shooter");
nt::GenericEntry& shooterEnable = *tab.Add("Shooter Enable", false).GetEntry();

// Command-based assumed to be in a PIDSubsystem
frc2::NetworkButton(shooterEnable).OnTrue(frc2::InstantCommand([&] { m_shooter.
↪Enable(); }));

// Timed Robot Example
if (shooterEnable.GetBoolean()) {
  // Calculates the output of the PID algorithm based on the sensor reading
  // and sends it to a motor
  motor.Set(pid.Calculate(encoder.GetDistance(), setpoint));
}
```

PYTHON

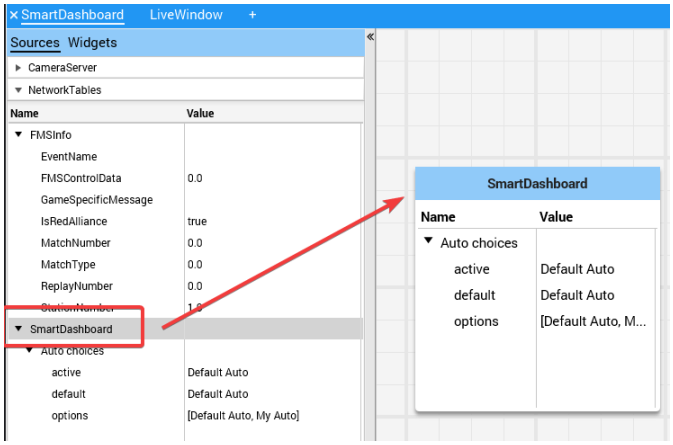
```
from wpilib.shuffleboard import Shuffleboard

tab = Shuffleboard.getTab("Shooter")
shooterEnable = tab.add("Shooter Enable", false).getEntry()

# Command Example assumed to be in a PIDSubsystem
NetworkButton(shooterEnable).onTrue(InstantCommand(m_shooter.enable()))

# Timed Robot Example
if (shooterEnable.getBoolean()):
  # Calculates the output of the PID algorithm based on the sensor reading
  # and sends it to a motor
  motor.set(pid.calculate(encoder.getDistance(), setpoint))
```

查看数据层次结构

将一个键及其下的其他键（层次结构中的更深层）拖动到树状结构中，类似于左侧的 NetworkTables 源。选择数据源：



单击并将 NetworkTables 键拖到首选选项卡中。



11.2.4 “Shuffleboard”-定制小部件

内置插件

Shuffleboard 提供了许多内置插件来处理 FRC® 的常见任务。使用，例如相机流、所有小部件和 NetworkTables 连接。

基本插件

基本插件定义了 FRC 使用所需的所有数据类型，小部件和布局。它 * 不 * 定义任何源类型，或这些源类型所需的任何特殊数据类型或小部件。这些由*NetworkTables Plugin* 和 ‘CameraServer Plugin‘_ 处理。关注点的分离使团队更容易为 FRC 数据的自定义源类型或协议（例如 HTTP，ZeroMQ）创建插件，而无需 NetworkTables 客户端。

CameraServer 插件

Camera Server 插件提供了源和小部件，用于查看“CameraServer” WPILib 类中的摄像机数据流。

这个插件依赖于*NetworkTables Plugin* 来发现可用的摄像机数据流。

数据流发现

通过查看“/ CameraPublisher“ NetworkTable 自动发现 CameraServer 源数据。

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

例如，名为“Camera”且服务器位于“roborio-0000-frc.local”的相机将具有以下表格布局：

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

此设置将通过 WPILib 中的 CameraServer 自动发现 roboRIO 上托管的所有摄像机数据流。希望在 Shuffle Board 上显示摄像机数据流的所有非 WPILib 项目都必须设置摄像机服务器的 Stream Entry。

NetworkTables 插件

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.

该插件自动处理与 NetworkTables 的连接和重新连接，shuffleboard 的用户和自定义插件的编写者将不必担心 NetworkTables 协议的复杂性。

创建一个插件

概述

插件提供了创建自定义小部件、布局、数据源/类型和自定义主题的能力。Shuffleboard 提供以下:ref: 内置插件 *<docs/software/dashboards/shuffleboard/custom-widgets/builtin-plugins:Built-in Plugins>*。

- NetworkTables Plugin: 用于连接通过网络传输的数据
- Base Plugin: 用于展示自定义的 FRC® 在自定义小配件里的数据分类
- CameraServer Plugin: 用于观察从照相服务器传来的数据

小技巧: 创建自定义数据类型的示例自定义 Shuffleboard 插件和用于显示它的简单小部件可以在这里找到 <https://github.com/wpilibsuite/shuffleboard/tree/main/example-plugins/custom-data-and-widget>‘__。

创建一个自定义插件

为了定义插件，插件类必须是 ‘edu.wpi.first.shuffleboard.api.Plugin <https://github.com/wpilibsuite/shuffleboard/blob/main/api/src/main/java/edu/wpi/first/shuffleboard/api/plugin/Plugin.java>‘_ 或其子类之一。插件类的示例如下。

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 {

}
```

有关如何使用这些属性（包括版本号）的其他说明，请参见“此处 <https://semver.org/>” _。

请注意，需要 @Description 注释来告知插件加载程序自定义插件类的属性。允许插件类具有默认构造函数，但不能接受任何参数。

建筑插件

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"`

允许插件依赖于其他插件和库，但是，它们必须正确包含在 maven 或 gradle 构建文件中。当一个插件依赖于其他插件时，最好定义这些依赖项，以便在依赖项也未加载时不加载该插件。这可以使用 @Requires 注释完成，如下所示：

```
@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 {

}
```

minVersion 指定可加载的插件的最低允许版本。例如，如果 “minVersion” 为 1.4.5，并且加载了版本为 1.4.7 的插件，则将允许这样做。但是，如果加载了版本 1.2.4 的插件，由于它小于 “minVersion“，因此将不允许这样做。

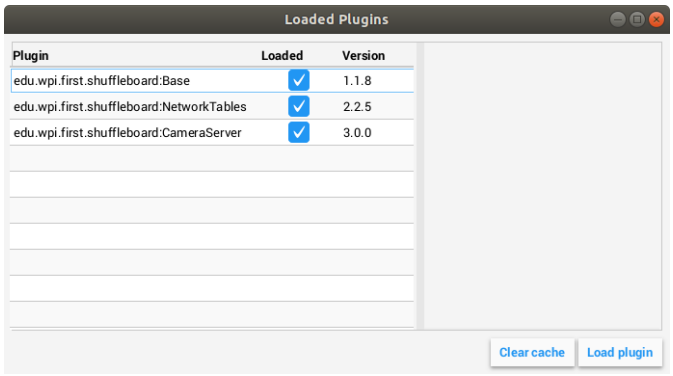
将插件部署到沙狐球

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`

部署后，Shuffleboard 将缓存插件的路径，以便下次 Shuffleboard 加载时可以自动加载。可能需要单击插件菜单下的“清除缓存’’以删除插件或将插件重新加载到 Shuffleboard 中。

手动添加插件

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.



在插件窗口中，选择右下角的“加载插件”按钮，然后选择您的 jar 文件。

创建自定义数据类型

小部件使我们能够控制和可视化不同类型的数据。该数据可以是整数或双精度甚至 Java 对象。为这些类型的数据创建一个容器类会帮助小部件显示它们。如果窗口小部件将处理单个字段数据类型（例如，双精度，数组或字符串），则无需创建自己的数据类。

创建数据类

在此示例中，我们将为 2D 点及其 x 和 y 坐标创建自定义数据类型。为了创建自定义数据类型类，它必须扩展抽象类 ‘ComplexData <https://github.com/wpilibsuite/shuffleboard/blob/main/api/src/main/java/edu/wpi/first/shuffleboard /api/data/ComplexData.java>‘_. 您的自定义数据类还必须实现 `asMap()` 方法，该方法将表示的数据作为简单地图返回，如下所述，带有 `@Override` 批注：

```
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;
```

(续下页)

(接上页)

```
   //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);
   }
}
```

覆盖默认的“equals”和“hashcode”方法也是一种很好的做法，以确保不同的对象在其字段相同时被认为是等效的。`asMap()` 方法应该返回以简单 Map 对象表示的数据，因为它将被映射到它对应的 NetworkTables 条目。在这种情况下，我们可以将点表示为其 X 和 Y 坐标，并返回一个包含它们的“地图”。

```
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);
   }
 }
```

可以添加其他方法来检索或编辑字段和实例变量，但是，优良作法是使这些类不可变以防止更改源数据对象。相反，您可以创建一个新的副本对象，而不是操纵现有对象。例如，如果要更改点的 y 坐标，则可以定义以下方法：

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

这将创建一个新的 MyPoint2D 对象，并以新的 y 坐标返回它。更改 x 坐标时也可以这样做。

创建数据类型

可以创建两种不同的数据类型：仅具有一个字段（即单个数字或字符串）的简单数据类型，以及具有多个数据字段（即多个字符串，多个数字）的复杂数据类型。

为了定义一个简单的数据类型，该类必须用所需的数据类型扩展“SimpleDataType<DataType>”类，并使用“getDefaultValue ()”方法。在此示例中，我们将使用 double 作为我们的简单数据类型。

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

   private static final String NAME = "Double";

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

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

}
```

将类构造函数设置为私密，以确保仅存在数据类型的单个实例。

为了定义复杂的数据类型，该类必须扩展“ComplexDataType”类，并覆盖“fromMap ()”和“getDefaultValue ()”方法。我们将以 MyPoint2D 类为例，以了解复杂数据类型类的外观。

```
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);
   }

}
```

上面的以下代码按规定工作：

`fromMap()` 方法使用它绑定到的 NetworkTables 条目中的值创建一个新的 MyPoint2D。如果无法获取条目值，`getOrDefault` 方法将返回 0.0。如果不存在源，`getDefaultValue` 将返回一个新的“MyPoint2D“ 对象。

将数据类型导出到插件

为了让 Shuffleboard 能够识别数据类型，插件必须通过覆盖 “getDataTypes” 的方法来导出它们。例如，

```
public class MyPlugin extends Plugin {

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

}
```

创建一个小部件

小部件使我们能够查看，更改和与通过不同数据源发布的数据进行交互。CameraServer，NetworkTables 和 Base 插件提供了用于控制基本数据类型（包括特定于 FRC 的数据类型）的小部件。但是，自定义小部件允许我们控制在上一节或 Java 对象中创建的自定义数据类型。

基本的 “Widget” 接口继承自 “Component” 和 “Sourced” 接口。“组件” 是在 Shuffleboard 中显示的最基本的组件构建块。Sourced 是用于处理和显示数据源或修改数据源的接口。不支持数据绑定但仅具有子节点的小部件将不使用 Sourced 接口，而仅使用 Component 接口。两者都是制作小部件的基本构建块，并允许我们修改和显示数据。

一个好的小部件可以让用户自定义窗口小部件以适应他们的需求。一个示例可能是允许用户控制数字滑块的范围，即其最大和最小值或滑块本身的方向。小部件的视图或其外观是使用 FXML 定义的。FXML 是一种基于 XML 的语言，可用于定义小部件（面板，标签和控件）的静态布局。

有关 FXML 的更多信息，请参见此处 https://openjfx.io/javadoc/11/javafx.fxml/javafx/fxml/doc-files/introduction_to_fxml.html。

定义小组件的 FXML

在此示例中，我们将创建两个滑块，以帮助我们控制在上一节中创建的 Point2D 数据类型的 X 和 Y 坐标。将 FXML 文件与 Java Class 放在同一包中会很有帮助。

为了给我们的小部件创建一个空的空白窗口，我们需要创建一个 “Pane”。Pane 窗格是一个父节点，其中包含其他子节点，在本例中为 2 个滑块。有许多不同类型的窗格，如下所示：

- 堆栈窗格
  - 堆栈窗格允许叠加的元素。另外，默认情况下，堆栈窗格默认为中心子节点。
- 网格窗格
  - ” 网格窗格 “通过在窗格上创建行和列的灵活网格来使用坐标系，在定义子元素非常有用。
- 流窗格
  - 流窗格将所有子节点都包装在边界集中。子节点可以垂直流动（包裹在窗格的高度边界处）或水平流动（包裹在窗格的宽度边界处）。
- 锚定窗格
  - 锚定窗格允许把子元素放在上方、下方、左侧、右侧和窗格的中心

布局窗格在用 HBox（https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/layout/HBox.html）把子元素放在一行，或用 VBox（https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/layout/VBox.html）把子元素放在一列时，会非常有用。

使用 FXML 定义窗格的基本语法如下：

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

fx：controller 属性会包含小组件的类的名称。加载 FXML 文件时，将创建此类的实例。为此，“控制器”类必须没有实际参数的构造函数。

创建一个小组件类

现在我们有了一个“窗格”，我们可以将子元素添加到该窗格中。在此示例中，我们可以添加两个滑块对象。记住要在每个元素上添加一个“fx：id”，以便可以在我们稍后将要创建的 Java 类中引用。我们将使用“VBox”将滑块放 · 置在彼此的顶部。

```
<?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>
```

现在我们已经完成了 FXML 文件的创建，现在可以创建一个小部件类了。小部件类应包含一个 @Description 注释，该注释说明小部件支持的数据类型和小部件名称。如果不存在 @Description 注解，则插件类必须实现 get（）方法以返回其小部件。

它还必须包含一个 @ParametrizedController 注释，该注释指向包含小部件布局的 FXML 文件。如果该类仅支持一个数据源，则必须扩展 SimpleAnnotatedWidget 类。如果该类支持多个数据源，则它必须扩展 ComplexAnnotatedWidget 类。有关更多信息，请参见：widget-types。

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

/*
 * If the FXML file and Java file are in the same package, that is the Java file is␣
↪in src/main/java and the
 * FXML file is under src/main/resources or your code equivalent package, the␣
↪relative path will work
 * However, if they are in different packages, an absolute path will be required.
*/

@Description(name = "MyPoint2D", dataTypes = MyPoint2D.class)
@ParametrizedController("Point2DWidget.fxml")
public final class Point2DWidget extends SimpleAnnotatedWidget<MyPoint2D> {

}
```

如果您没有使用自定义数据类型，则可以引用任何 Java 数据类型（即 Double.class），或者如果小部件不需要数据绑定，则可以传递 NoneType.class。

现在我们已经创建了类，我们可以使用 @FXML 批注为在 FXML 文件中声明的小部件创建字段。对于我们的两个滑块，一个示例是：

```
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;
}
```

为了在我们的自定义窗口小部件上显示我们的窗格，我们需要重写 getView () 方法并返回我们的 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;

   @FXML
   private Slider xSlider;

   @FXML
   private Slider ySlider;

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

}
```

绑定元素和添加侦听器

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.

在这种情况下，示例将通过分别更改 xSlider 和 ySlider 的值来更改 2D 点的 X 和 Y 坐标。

一个好的做法是在带有 @FXML 批注的“initialize ()”方法中设置绑定，如果该方法不是“public”，则必须从 FXML 调用该方法。

```
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;
   }

 }
```

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.

当滑块或数据源已更改时，使用侦听器是另一种更改值的方法。例如，滑块的侦听器为：

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

在这种情况下，setData () 方法会将小部件的数据源中的值设置为 newValue。

探索自定义组件

加载插件时不会自动发现窗口小部件；定义插件必须导出它才能使用。采用这种方法可以在同一个 JAR 中定义多个插件。

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

设置默认小组件数据类型

为了将小部件设置为自定义数据类型的默认值，您可以覆盖插件类中的 getDefaultComponents ()，该类存储所有默认小部件的 Map，如下所示：

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

自定义主题

Since shuffleboard is a JavaFX application, it has support for custom themes via Cascading Stylesheets (**CSS** for short). These are commonly used on webpages for making HTML look nice, but JavaFX also has support, albeit for a different language subset (see here for documentation on how to use it).

Shuffleboard 默认带有三个主题：浅色，深色和午夜。这些是同一材料设计样式表上的颜色变化。另外，它们继承自 base.css 样式表，该样式表定义了自定义组件的样式，这些样式在沙狐球板或所使用的库中定义；基础材料设计样式表仅适用于 JavaFX 中内置的 UI 组件。

定义自定义主题有两种方法：将样式表放在“~/Shuffleboard/themes”中主题名称的目录中；例如，理论上的“黄色”主题可以放在

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

目录中的所有样式表将被视为主题的一部分。

通过插件加载主题

自定义主题也可以由插件定义。这使它们更易于与自定义窗口小部件共享和捆绑，但定义起来却稍微困难一些。主题对象将需要对插件中定义的类的引用，以便插件加载器可以确定样式表的位置。如果传递的类在插件所在的 JAR 中不存在，则将无法使用该主题。

```
@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", ...);
```

(续下页)

(接上页)

```
  @Override
  public List<Theme> getThemes() {
    return ImmutableList.of(myTheme);
  }

}
```

修改或扩展 Shuffleboard 的默认主题

Shuffleboard 的 Material Light 和 Material Dark 主题分别为亮色和深色主题提供了很多框架，以及针对 Shuffleboard，ControlsFX 和 Medusa UI 组件的许多特定样式，以适应材料样式设计。

想要修改这些主题的主题需要为这些样式表添加“import”语句：

```
@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 */
```

请注意，base.css 内部会导入 material.css 和 light.css，dark.css 和 midnight.css 都将导入 base.css”，因此导入“light.css”将隐式导入“base.css”和“material.css”。

CSS 文件的源代码

- _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

材质设计颜色色板

材质设计 CSS 对几乎所有内容都使用颜色样本变量。可以从自定义 CSS 文件设置这些变量，从而减少所需的自定义代码量。

“-swatch- <100|200|300|400|500>’’ 变量定义相同原色的逐渐变深的阴影。浅色主题使用在 material.css 中设置的默认蓝色阴影，而深色主题用红色阴影覆盖它们。“-swatch- <|light|dark> -gray’’ 定义了三个灰度等级，用于各种背景或文本颜色。

覆盖色板颜色

用红色代替蓝色（浅色）

```
@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%);
}
```

将红色替换为蓝色（深色）

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

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

小部件类型

虽然“Widget ”对于接口来说非常直观，但是有几个中间实现可以使定义小部件变得更容易。

类	描述
“抽象小部件”	实现“getProperties()”，“getSources()”，和“titleProperty()”
“单一类型小部件”	添加只支持一种数据类型的小部件特性
“带注释的小部件”	给带有“@Description”注释的小部件添加“getName()”和“getDataTypes()”的默认实现
“单一资源小部件”	对于只有一个源的小部件 (默认情况下，小部件支持多个源)
“单一带注释的小部件”	合并“单一类型小部件”，“带注释的小部件”和“单一资源小部件”

还有两个注释可以帮助定义 widget：

名称	描述
“@ 参数化控制器”	允许小部件成为通过 FXML 定义的 JavaFX 视图的 FXML 控制器
“@ 描述”	让名称和支持的数据类型定义在一行中

抽象小部件

此类实现了“getProperties()”、“getSources()”、“addSource()”和“titleProperty()”。它还定义了一个方法 `exportProperties(Property<?>...)` 方法，因此子类可以轻松添加自定义小部件属性，或小部件中 JavaFX 组件的属性。基础插件中的大部分 ‘widgets <https://github.com/wpilibsuite/shuffleboard/tree/main/plugins/base/src/main/java/edu/wpi/first/shuffleboard/plugin/base/widget> ‘_ 使用这个。

单一类别小部件

一种只支持一种数据类型的小部件。这个接口是参数化的，具有设置或获取数据的方法，以及获取小部件 (单个) 数据类型的方法。

带注释的小部件

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 {
  // ...
}
```

单一资源小部件

一种只用一个资源的小部件。

单一带注释的小部件

“SingleTypeWidget1”，“AnnotatedWidget”，和“SingleSourceWidget”的组合。大多数基本插件中的小部件从这个类中拓展而来。这同时也有一个“受保护的”名叫“dataOrDefault”的字段，它在小部件没有资源或资源显示“null”时使子类别们使用没有资源的默认数据。

@ 参数化控制器

这个注释可以放在小部件类上，让 shuffleboard 知道它是通过 FXML 定义的 JavaFX 视图的 FXML 控制器。注释接受一个参数，该参数定义了 FXML 文件 * 相对于它所在的类的位置 *。例如，目录“src/main/java/com/acme”中的一个小部件，它是“src/main/resources/com/acme”中的一个 FXML 文件的 FXML 控制器，可以使用注释作为任何一种

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

或作为

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

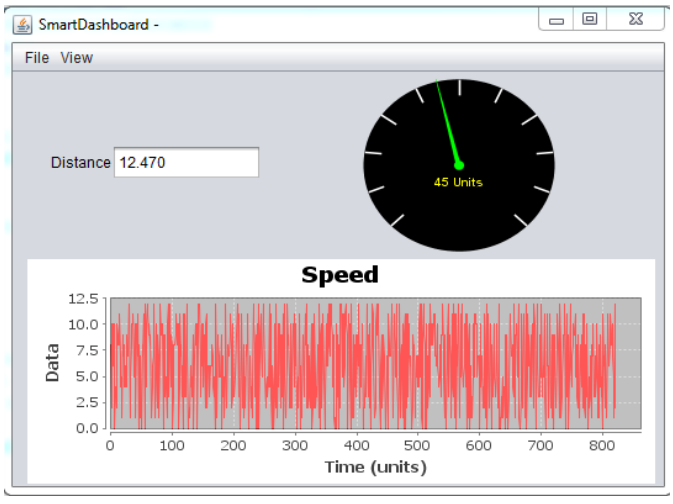
@ 描述

当与’AnnotatedWidget ‘_ 一起使用时，这允许小部件的名称和受支持的数据类型由单个注释定义。

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 SmartDashboard Introduction

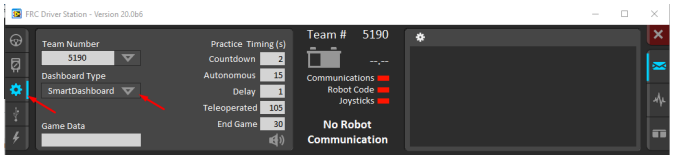


The SmartDashboard is a Java program that will display robot data in real time. The SmartDashboard helps you with these things:

- Displays robot data of your choice while the program is running. It can be displayed as simple text fields or more elaborately in many other display types like graphs, dials, etc.
- Displays the state of the robot program such as the currently executing commands and the status of any subsystems
- Displays buttons that you can press to cause variables to be set on your robot
- Allows you to choose startup options on the dashboard that can be read by the robot program

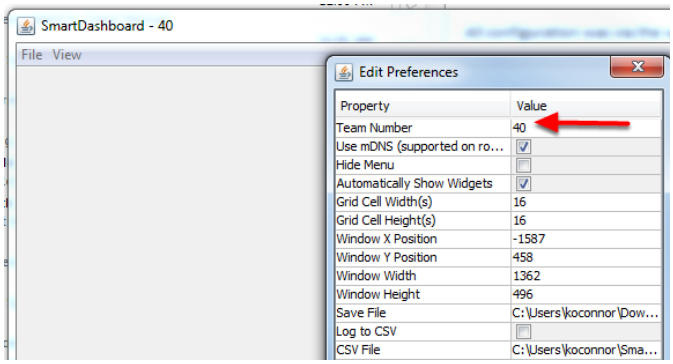
The displayed data is automatically formatted in real-time as the data is sent from the robot, but you can change the format or the display widget types and then save the new screen layouts to be used again later. And with all these options, it is still extremely simple to use. To display some data on the dashboard, simply call one of the SmartDashboard methods with the data and its name and the value will automatically appear on the dashboard screen.

Installing the SmartDashboard



The SmartDashboard is packaged with the WPILib Installer and can be launched directly from the Driver Station by selecting the **SmartDashboard** button on the Setup tab.

Configuring the Team Number



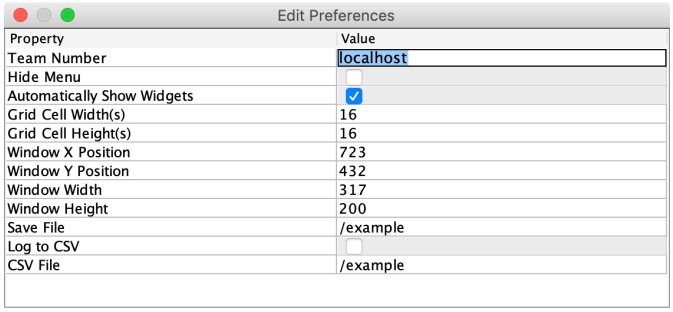
The first time you launch the SmartDashboard you should be prompted for your team number. To change the team number after this: click **File > Preferences** to open the Preferences dialog. Double-click the box to the right of **Team Number** and enter your FRC® Team Number, then click outside the box to save.

备注: SmartDashboard will take a moment to configure itself for the team number, do not be alarmed.

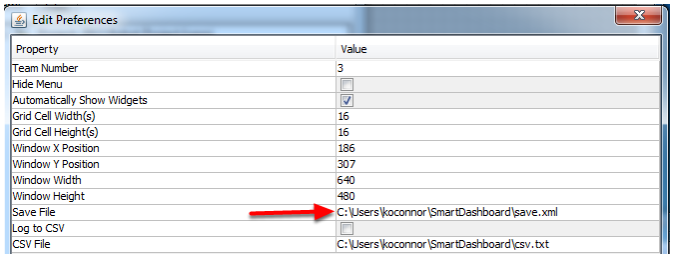
Setting a Custom NetworkTables Server Location

By default, SmartDashboard will look for NetworkTables instances running on a connected RoboRIO, but it's sometimes useful to look for NetworkTables at a different IP address. To connect to SmartDashboard from a host other than the roboRIO, open SmartDashboard preferences under the `File` menu and in the `Team Number` field, enter the IP address or hostname of the NetworkTables host.

This option is incredibly useful for using SmartDashboard with *WPILib simulation*. Simply add `localhost` to the `Team Number` field and SmartDashboard will detect your locally-hosted robot!

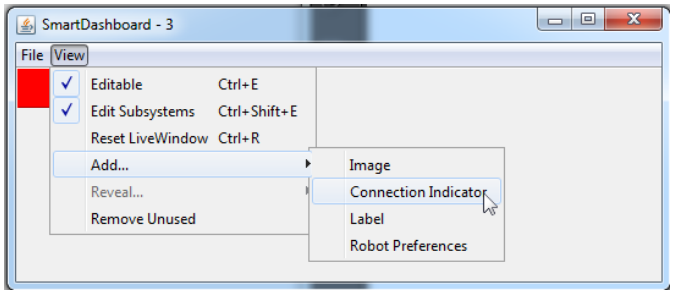


Locating the Save File



Users may wish to customize the save location of the SmartDashboard. To do this click the box next to **Save File** then browse to the folder where you would like to save the configuration. Files saved in the installation directories for the WPILib components will likely be overwritten on updates to the tools.

Adding a Connection Indicator



It is often helpful to see if the SmartDashboard is connected to the robot. To add a connection indicator, select **View > Add > Connection Indicator**. This indicator will be red when

disconnected and green when connected. To move or resize this indicator, select **View > Editable** to toggle the SmartDashboard into editable mode, then drag the center of the indicator to move it or the edges to resize. Select the **Editable** item again to lock it in place.

Adding Widgets to the SmartDashboard

Widgets are automatically added to the SmartDashboard for each "key" sent by the robot code. For instructions on adding robot code to write to the SmartDashboard see *Displaying Expressions from Within the Robot Program*.

11.3.2 从机器人程序显示表达式

备注: 一般在调试或检测机器人的状态时，需要将一些参数传入控制台并观察他们的变化。在智能仪表盘的帮助下，你将可以把参数输进一个基于你得程序而自动生成的图形化用户界面。随着参数的更新，相应图形界面元素的值也将会改变。用户大可不必用肉眼捕捉屏幕上变化的参数。

将值输入智能仪表盘

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())
```

当你需要将布尔值/数值/字符串作为参数输入智能仪表盘中时，你可以直接调用对应类型的方法并输入参数的名称和参数的具体值。你不需要写任何其他的复杂代码。当你的程序中有两个值被赋予了同样的参数名，他们将被置入同一个图形化用户界面元素下并显示在操作台或开发者电脑的屏幕上。正如你所想的那样，这将是一种极佳的调试方法。

在智能仪表盘上创建小工具

窗口小部件会被自动填充在智能仪表盘上而不需要用户干预。需要注意的是，小部件仅会在数值第一次被写入时添加。为了使某些特定项目出现，你可能需要以特定模式启动机器人或触发特定代码例程。当你想要更改小部件的外观时，请参考以下两章的内容：“更改值的显示属性”<changing-display-properties> 和“更改显示值所对应的小工具的类型”<changing-display-widget-type>

过期数据

SmartDashboard 使用:term:*NetworkTables* 在机器人和机器操控台笔记本电脑之间传递值。NetworkTables 充当名称和值对的分布式表。如果将名称/值对添加到客户端（笔记本电脑）或服务器（机器人），则会将其复制到另一个。如果名称/值对从机器人中删除，但 SmartDashboard 或 OutlineViewer 仍在运行，那么当机器人重新启动时，旧值仍将出现在 SmartDashboard 和 OutlineViewer 中，因为它们从未停止运行并继续运行他们表中的那些值。当机器人重新启动时，这些旧值将复制到机器人。

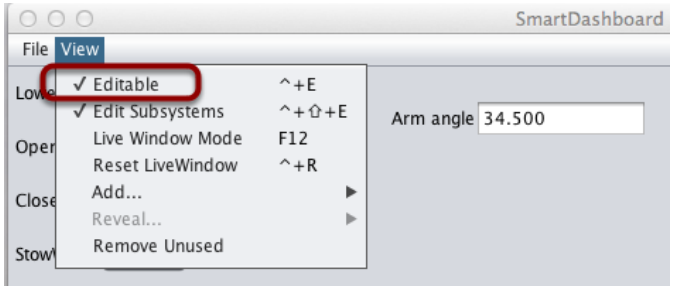
为确保 SmartDashboard 和 OutlineViewer 显示当前值，必须同时重新启动 NetworkTables 客户端和机器人。这样，一个人持有的旧值就不会复制到其他人身上。

如果程序没有在不断变化，这通常不是问题，但是如果程序正在开发中，并且添加到 NetworkTables 的键组在不断变化，那么可能需要重新启动所有程序以准确查看当前是什么。

11.3.3 Changing the display properties of a value

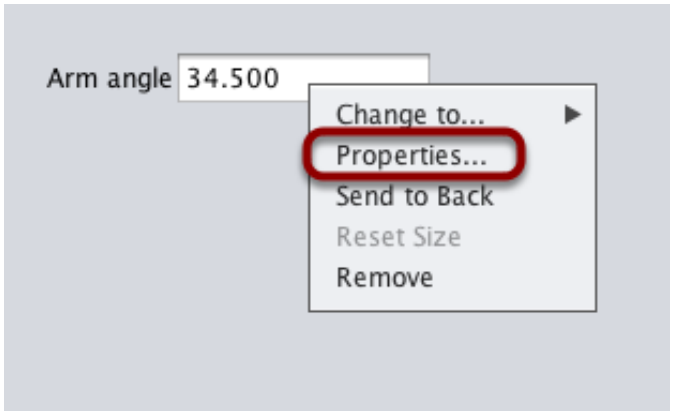
Each value displayed with SmartDashboard has a set of properties that effect the way it’s displayed.

Setting the SmartDashboard display into editing mode



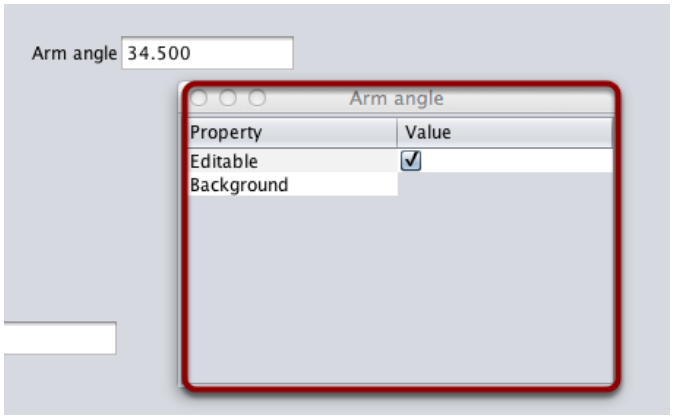
The SmartDashboard has two modes it can operate in, display mode and edit mode. In edit mode you can move around widgets on the screen and edit their properties. To put the SmartDashboard into edit mode, click the "View" menu, then select "Editable" to turn on edit mode.

Getting the properties editor of a widget



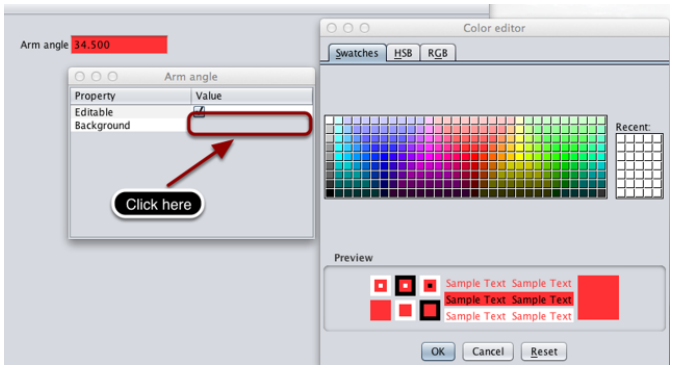
Once in edit mode, you can display and edit the properties for a widget. Right-click on the widget and select "Properties…".

Editing the properties on a field



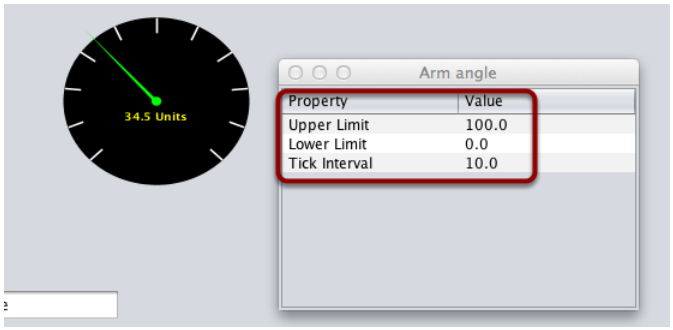
A dialog box will be shown in response to the "Properties…" menu item on the widgets right-click context menu.

Editing the widgets background color



To edit a property value, say, Background color, click the background color shown (in this case grey), and choose a color from the color editor that pops up. This will be used as the widgets background color.

Edit properties of other widgets

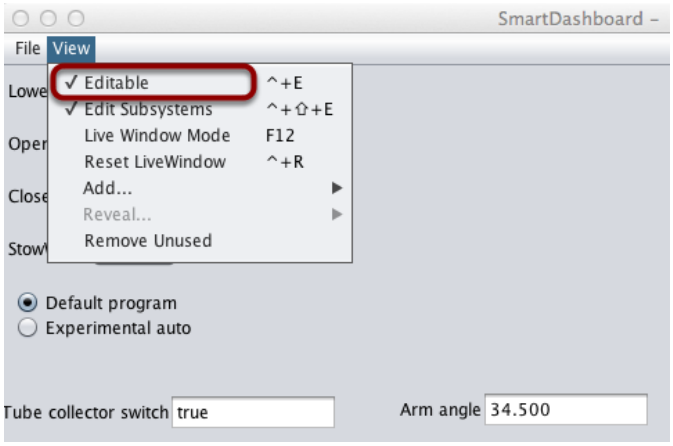


Different widget types have different sets of editable properties to change the appearance. In this example, the upper and lower limits of the dial and the tick interval are changeable parameters.

11.3.4 Changing the Display Widget Type for a Value

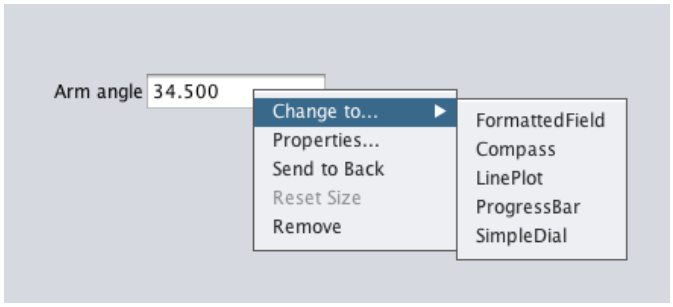
One can change the type of widget that displays values with the SmartDashboard. The allowable widgets depend on the type of the value being displayed.

Setting Edit Mode



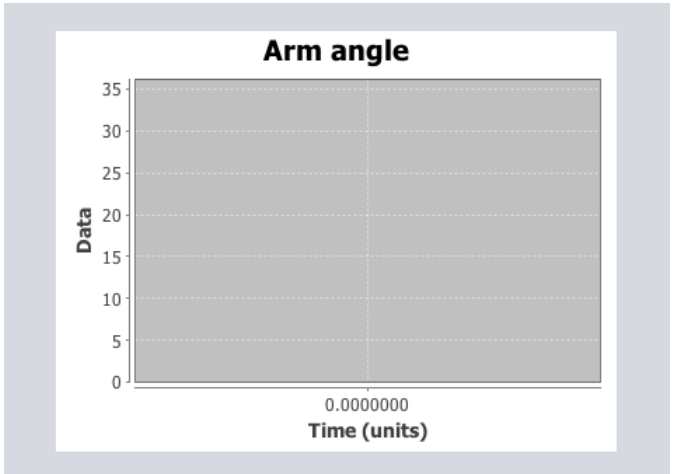
Make sure that the SmartDashboard is in edit mode. This is done by selecting `Editable` from the `View menu`.

Choosing Widget Type



Right-click on the widget and select `Change to...`. Then, pick the type of widget to use for the particular value. In this case we choose `LinePlot`.

Showing New Widget Type



The new widget type is displayed. In this case, a Line Plot, will show the values of the Arm angle value over time. You can set the properties of the graph to make it better fit your data by right-clicking and selecting `Properties...`. See: *Changing the display properties of a value*.

11.3.5 选择自动阶段动程序

出于竞争原因或测试新软件的原因，团队通常拥有多个自动阶段。程序通常会通过添加时间延迟，加入不同的策略等来变化。选择要运行的策略的方法通常包括开关，操纵杆按钮，旋钮或其他基于硬件的输入。

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

备注： 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

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

创建 SendableChooser 对象

在“RobotContainer”中，创建一个变量以保存对“SendableChooser”对象的引用。可以创建两个或更多命令并将其存储在新变量中。使用“SendableChooser”，可以在它们之间进行选择。在此示例中，“SimpleAuto”和“ComplexAuto”显示为选项。

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()
```

设置 SendableChooser

假设你有两个自动阶段的程序可供选择，它们被封装在指令“SimpleAuto”和“ComplexAuto”中。在它们当中进行选择：

在“RobotContainer”中创建一个“SendableChooser”对象，并向其添加两个指令的实例。指令的数量没有限制，并且默认添加的指令（`setDefaultOption`）将成为最初选择的命令。注意，每个指令都包含在“SendableChooser”实例上的“setDefaultOption ()”或“addOption ()”方法调用中。

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)
```

启动自动阶段指令

在“Robot.java”中，当自动阶段开始时，将选取“SendableChooser”对象以获取所选指令，并且必须调度该指令。

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()
```

在自动阶段运行调度程序

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.

备注： 运行调度程序可以在 autonomousPeriodic () 函数或 robotPeriodic () 中进行，两者在自动阶段下的功能相似。

Java

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

C++ (Source)

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

取消自动阶段指令

在“Robot.java”当中，当遥控阶段开始时，自动阶段指令将被终止。

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   }
```

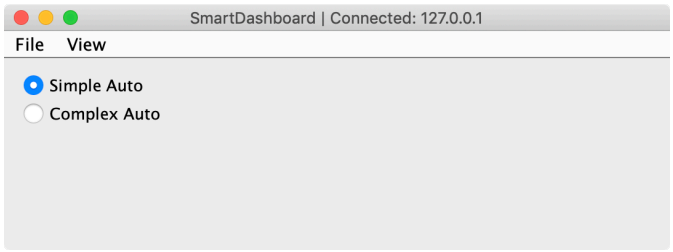
C++ (Source)

```
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()
```

SmartDashboard 显示

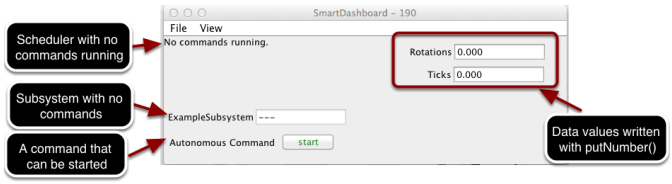


运行 SmartDashboard 时，将自动显示“SendableChooser”中的选项。您可以在自动阶段开始之前选择一个选项，然后相应的指令将运行。

11.3.6 Displaying the Status of Commands and Subsystems

If you are using the command-based programming features of WPILib, you will find that they are very well integrated with SmartDashboard. It can help diagnose what the robot is doing at any time and it gives you control and a view of what's currently running.

Overview of Command and Subsystem Displays



With SmartDashboard you can display the status of the commands and subsystems in your robot program in various ways. The outputs should significantly reduce the debugging time for your programs. In this picture you can see a number of displays that are possible. Displayed here are:

- The Scheduler currently with `No commands running`. In the next example you can see what it looks like with a few commands running showing the status of the robot.
- A subsystem, `ExampleSubsystem` that indicates that there are currently no commands running that are "requiring" it. When commands are running, it will indicate the name of the commands that are using the subsystem.
- A command written to SmartDashboard that shows a `start` button that can be pressed to run the command. This is an excellent way of testing your commands one at a time.
- And a few data values written to the dashboard to help debug the code that's running.

In the following examples, you'll see what the screen would look like when there are commands running, and the code that produces this display.

Displaying the Scheduler Status

JAVA

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

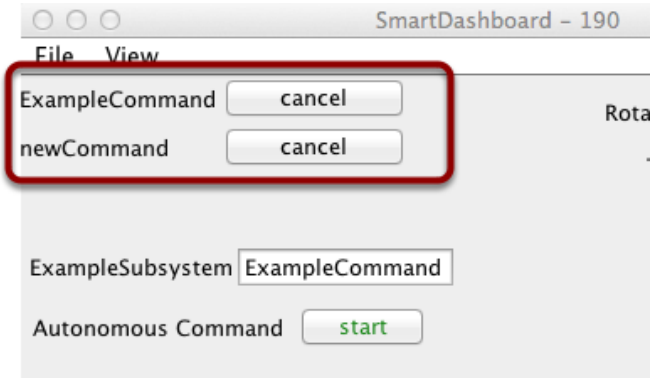
C++

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

PYTHON

```
from wpilib import SmartDashboard
from commands2 import CommandScheduler

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

You can display the status of the Scheduler (the code that schedules your commands to run). This is easily done by adding a single line to the `RobotInit` method in your RobotProgram as shown here. In this example the Scheduler instance is written using the `putData` method to SmartDashboard. This line of code produces the display in the previous image.



This is the scheduler status when there are two commands running, `ExampleCommand` and `newCommand`. This replaces the `No commands running.` message from the previous screen image. You can see commands displayed on the dashboard as the program runs and various commands are triggered.

Displaying Subsystem Status

JAVA

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

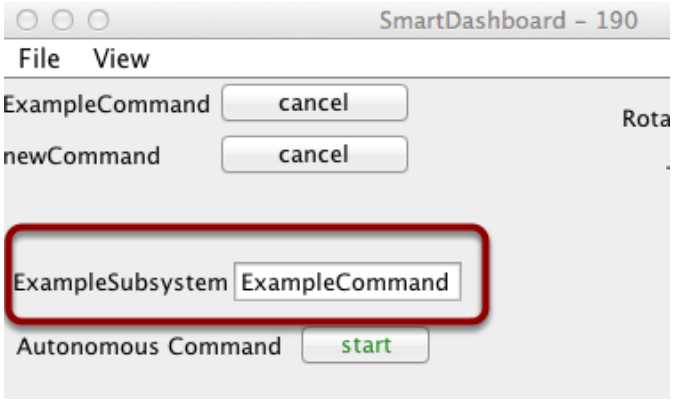
C++

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

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData(exampleSubsystem)
```

In this example we are writing the command instance, `exampleSubsystem` and instance of the `ExampleSubsystem` class to the SmartDashboard. This causes the display shown in the previous image. The text field will either contain a few dashes, `---` indicating that no command is current using this subsystem, or the name of the command currently using this subsystem.



Running commands will "require" subsystems. That is the command is reserving the subsystem for its exclusive use. If you display a subsystem on SmartDashboard, it will display which command is currently using it. In this example, `ExampleSubsystem` is in use by `ExampleCommand`.

Activating Commands with a Button

JAVA

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

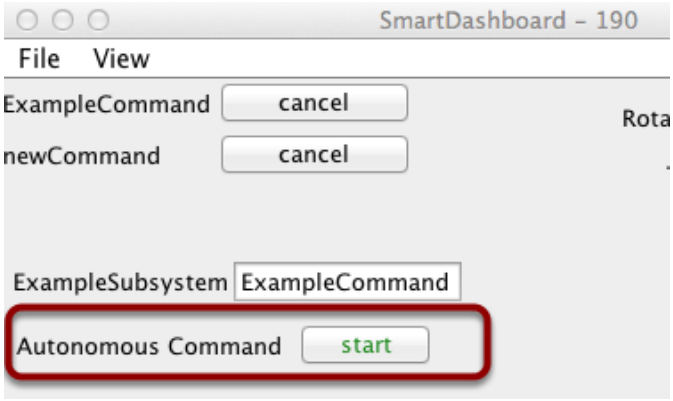
C++

```
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.

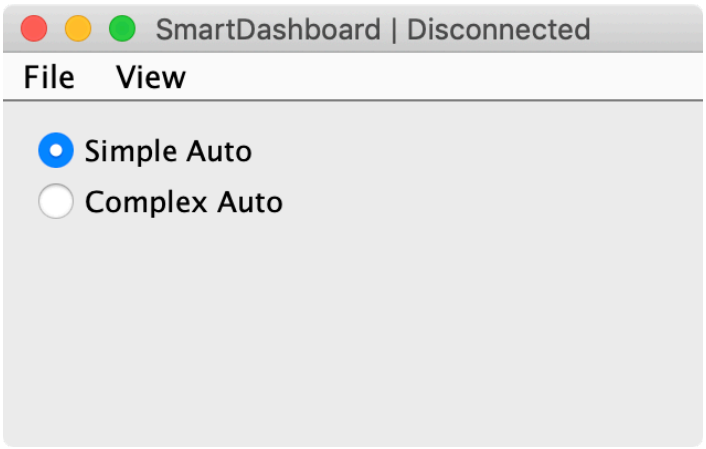


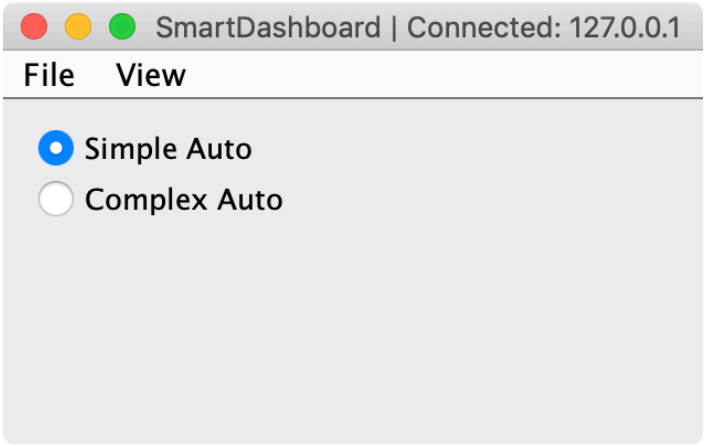
In this example you can see a button labeled `Autonomous Command`. Pressing this button will run the associated command and is an excellent way of testing commands one at a time without having to add throw-away test code to your robot program. Adding buttons for each command makes it simple to test the program, one command at a time.

11.3.7 Verifying SmartDashboard is working

Connection Indicator

SmartDashboard will automatically include the connection status and IP address of the NetworkTables source in the title of the window.





Connection Indicator Widget

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.

Robot Program Example

JAVA

```
public class Robot extends TimedRobot {
   double counter = 0.0;

   public void teleopPeriodic() {
      SmartDashboard.putNumber("Counter", counter++);
   }
}
```

C++

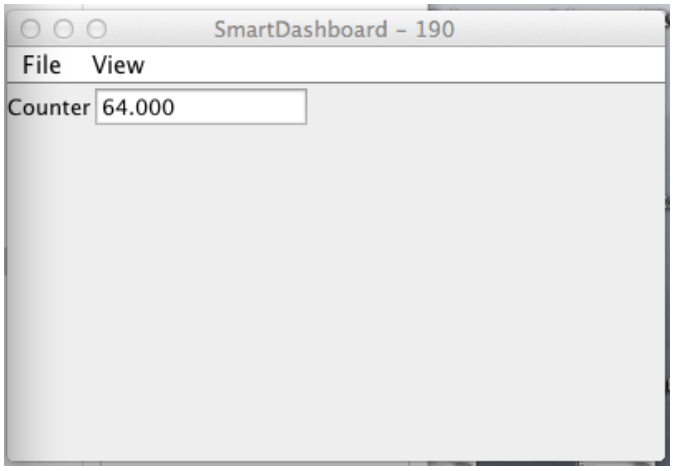
```
#include "Robot.h"
float counter = 0.0;

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)
```

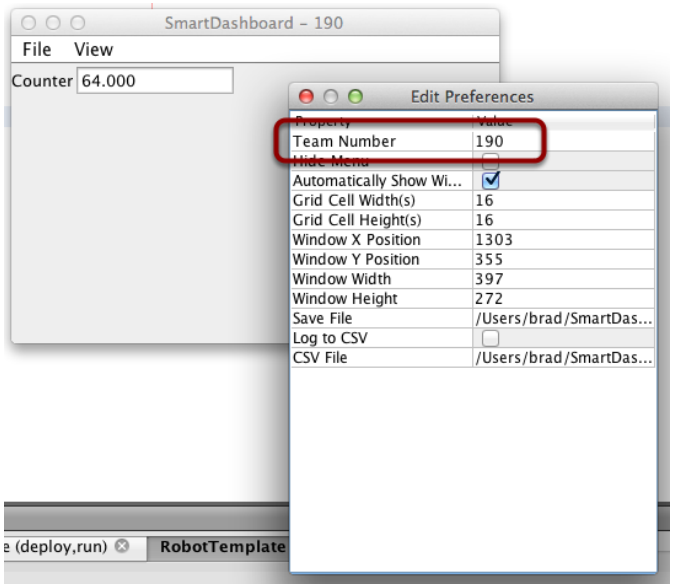
This is a minimal robot program that writes a value to the SmartDashboard. It simply increments a counter 50 times per second to verify that the connection is working. However, to minimize bandwidth usage, NetworkTables by default will throttle the updates to 10 times per second.

SmartDashboard Output for the Sample Program



The SmartDashboard display should look like this after about 6 seconds of the robot being enabled in Teleop mode. If it doesn't, then you need to check that the connection is correctly set up.

Verifying the IP address in SmartDashboard

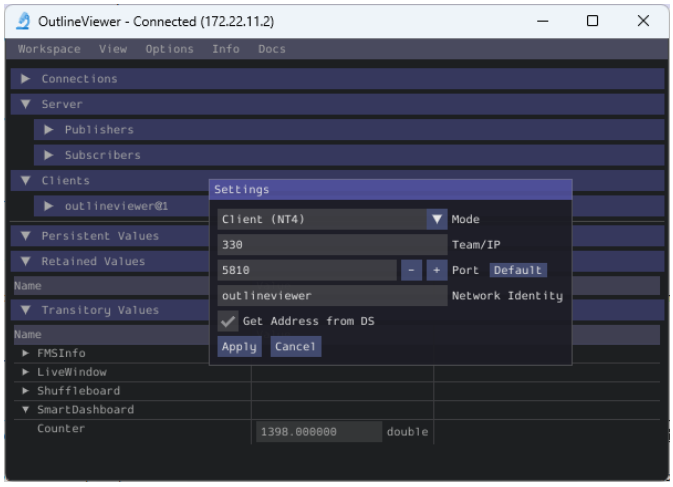


If the display of the value is not appearing, verify that the team number is correctly set as shown in this picture. The preferences dialog can be viewed by selecting `File`, then `Preferences`.

Verifying Program using 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.



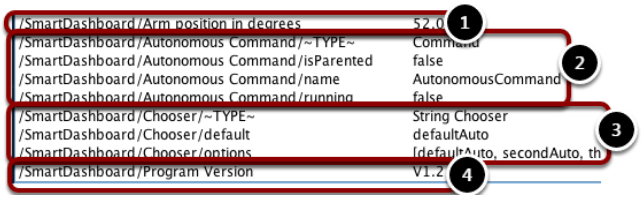
11.3.8 SmartDashboard Namespace

SmartDashboard 使用 NetworkTables 在机器人和仪表板（驱动站）计算机之间发送数据。NetworkTables 以名称、值对的形式发送数据，就像机器人和计算机之间的分布式哈希表。当一个值在一个位置更改时，它的值在另一个位置自动更新。这种机制和一组标准的名称（键）是如何在 SmartDashboard 上显示数据的。

名称空间中有一个层次结构，用于创建一组表和子表。SmartDashboard 数据位于 SmartDashboard 子表中，LiveWindow 数据位于 LiveWindow 子表中，如下所示。

为了提供信息，可以使用安装在与 SmartDashboard 相同位置的 OutlineViewer 应用程序显示名称和值。它将显示更新后的所有 NetworkTables 键和值。

智能仪表盘数据集

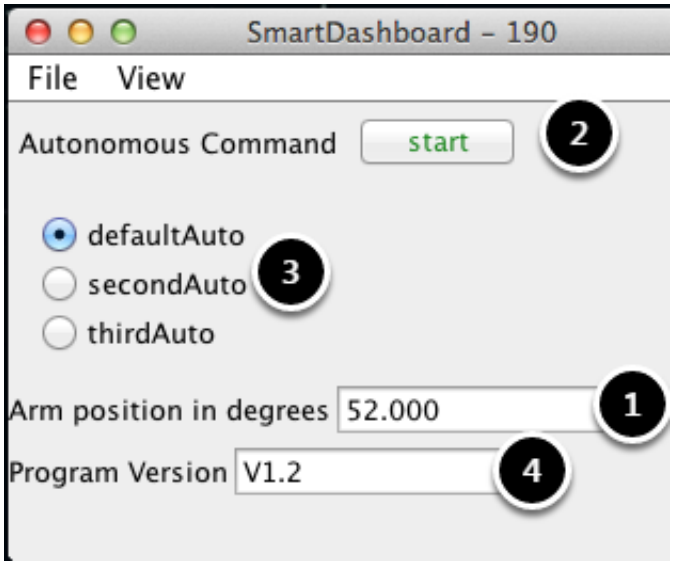


SmartDashboard 值是用以“SmartDashboard/‘ ‘开头的键名创建的。使用 OutlineViewer 查看的上述值与放入 SmartDashboard 的数据相对应，其中包含以下语句：

```
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");
```

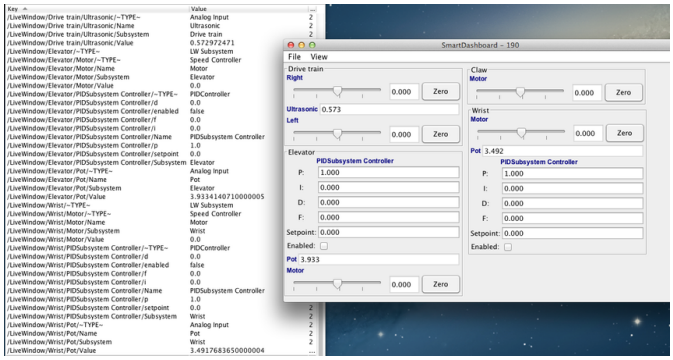
The `Arm position` is created with the `putNumber()` call. The `AutonomousCommand` is written with a `putData("Autonomous Command", command)` that is not shown in the above code fragment. The chooser is created as a `SendableChooser` object and the string value, `Program Version` is created with the `putString()` call.

智能仪表盘显示



上一步的代码生成如下所示的表值和如下所示的 SmartDashboard 显示。这些数字对应于前面步骤中显示的 NetworkTables 变量。

LiveWindow Data Values



LiveWindow 数据按子系统自动分组。当机器人处于测试模式（在驱动站上设置）时，可以在 SmartDashboard 中查看数据。如果您没有编写基于指令的程序，仍然可以通过指定子系统名称将传感器和执行器分组以便于查看。在上面的显示中，您可以在 SmartDashboard 上看到测试模式下的键名称和结果输出。所有字符串以“/LiveWindow‘ ‘开头，然后是子系统名称，然后是用于显示每个元素的一组值。生成此 LiveWindow 显示的代码如下所示：

```
drivetrainLeft = new PWMVictorSPX(1);
drivetrainLeft.setName("Drive train", "Left");

drivetrainRight = new PWMVictorSPX(1);
drivetrainRight.setName("Drive train", "Right";

drivetrainRobotDrive = new DifferentialDrive(drivetrainLeft, drivetrainRight);
drivetrainRobotDrive.setSafetyEnabled(false);
drivetrainRobotDrive.setExpiration(0.1);
```

(续下页)

(接上页)

```
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");
```

不仅显示与执行器对应的值，还可以在 Test 模式下使用 SmartDashboard 中创建的滑块进行设置。

11.3.9 SmartDashboard: Test Mode and Live Window

Displaying LiveWindow Values

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.

Adding the Necessary Code to your Program

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);
```

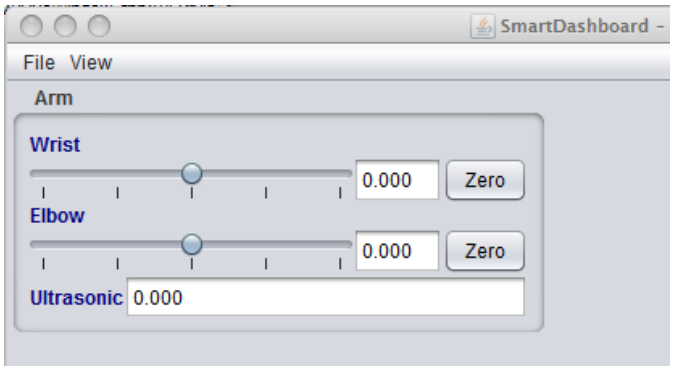
C++

```
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)
```

Viewing the Display in SmartDashboard



The sensors and actuators added to the LiveWindow will be displayed grouped by subsystem. The subsystem name is just an arbitrary grouping the helping to organize the display of the sensors. Actuators can be operated by operating the slider for the two motor controllers.

Enabling Test mode (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.

重要： 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)
```

Explicitly vs. implicit test mode display

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++

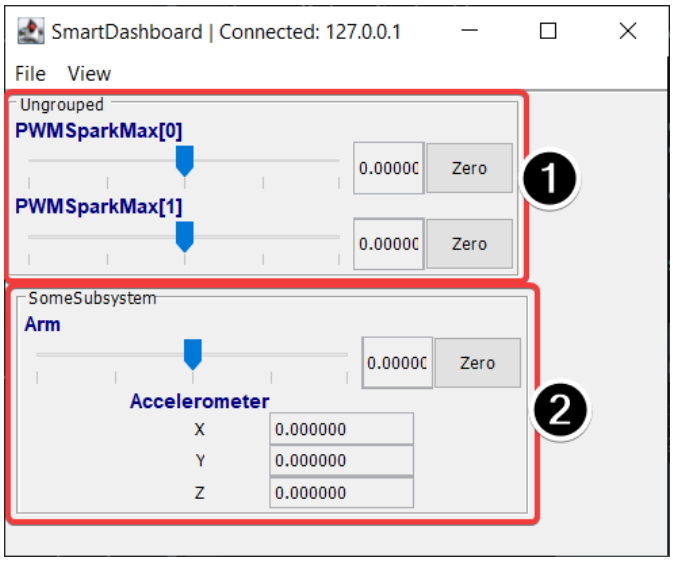
```
frc::PWMSparkMax leftDrive{0};
frc::PWMSparkMax rigthDrive{1};
frc::BuiltInAccelerometer accel{};
frc::PWMVictorSPX arm{3};

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.

Understanding what is displayed in Test mode



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.

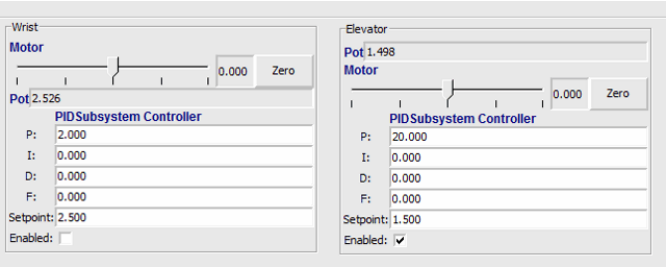
使用智能仪表盘进行 PID 调整

PID（比例，积分，微分）是一种算法，利用根据传感器反馈指定电动机速度，以尽快达到设定值。例如，带有升降机移动到预定位置的机器人应尽可能快地移动到那里，然后停下来而不会产生过多的过冲而导致振荡。使 PID 控制器以这种方式运行称为"调整"。这个想法是要计算一个误差值，该误差值是机械反馈元件的当前值与所需（设定点）值之间的差。对于手臂，可能有一个电位计连接到模拟通道，该电位计提供的电压与手臂的位置成比例。期望值是针对手臂应移动到的位置而预先确定的电压，而当前值是针对手臂的实际位置的电压确定的。

使用"LiveWindow"查找设定值

为每个带有反馈系统的机械结构创建一个 PID 子系统。PID 子系统包含执行器（电机）和反馈传感器（在这种情况下为电位计）。您可以使用"测试"模式显示子系统传感器和执行器。使用滑块手动将执行器调整到每个所需位置。注意每个所需位置的传感器值（2）。这些将成为 PID 控制器的设定值。

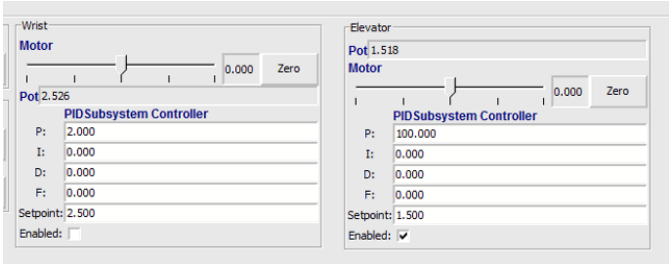
在"LiveWindow"中查看 PID 控制器



在测试模式下，PID 子系统显示其在代码中设置的 P，I 和 D 参数。P，I 和 D 值将被应用于计算计算误差（P）的权重，随时间变化的误差之和（I）和误差的变化率（D）。这些项中的每一项都乘以权重，然后相加在一起即可得出电动机值。选择最佳的 P，I 和 D 值可能很困难，并且需要进行大量的实验。机械手的"测试"模式允许修改值，并观察机械结构的响应状况。

重要： 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.

调整 PID 控制器



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.

一旦确定了 P，I 和 D 值，就可以将它们写入程序中。你可以在 “RobotBuilder” 的 “PIDSubsystem” 的属性中或代码中的 PID 子系统的构造函数中找到它们。

F (feedforward) 项被在 PID 控制器中被用于控制其速度。

更多信息可以参考：” <docs/software/advanced-controls/controllers/pidcontroller:PID Control in WPILib>”

11.4 Glass

Glass is a new dashboard and 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.

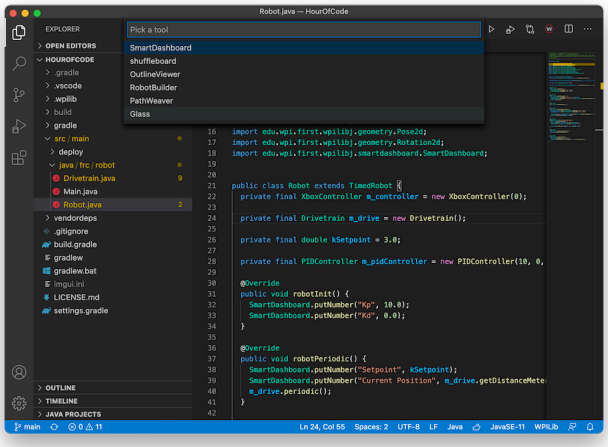
备注： Glass will not be available within the list of dashboards in the NI Driver Station.

11.4.1 Introduction to Glass

Glass is a new dashboard and robot data visualization tool. It supports many of the same *widgets* that the Simulation GUI supports, including robot pose visualization and advanced plotting. In its current state, it is meant to be used as a programmer's tool for debugging and not as a dashboard for competition use.

Opening Glass

Glass can be launched by selecting the ellipsis menu (···) in VS Code, clicking on *Start Tool* and then choosing *Glass*.

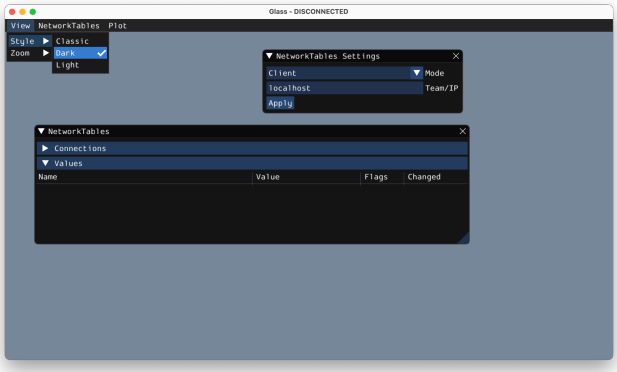


备注: You can also launch Glass directly by navigating to `~/wpilib/YYYY/tools` and running `Glass.py` (Linux and macOS) or by using the shortcut inside the WPILib Tools desktop folder (Windows).

Changing View Settings

The *View* menu item contains *Zoom* and *Style* settings that can be customized. The *Zoom* option dictates the size of the text in the application whereas the *Style* option allows you to select between the `Classic`, `Light`, and `Dark` modes.

An example of the `Dark` style setting is below:



Clearing Application Data

Application data for Glass, including widget sizes and positions as well as other custom information for widgets is stored in a `glass.ini` file. The location of this file varies based on your operating system:

- On Windows, the configuration file is located in `%APPDATA%`.
- On macOS, the configuration file is located in `~/Library/Preferences`.
- On Linux, the configuration file is located in `$XDG_CONFIG_HOME` or `~/.config` if the former does not exist.

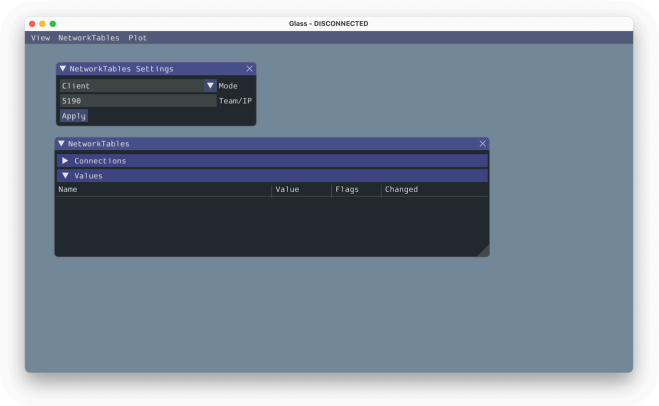
The `glass.ini` configuration file can simply be deleted to restore Glass to a "clean slate".

11.4.2 建立网络表连接

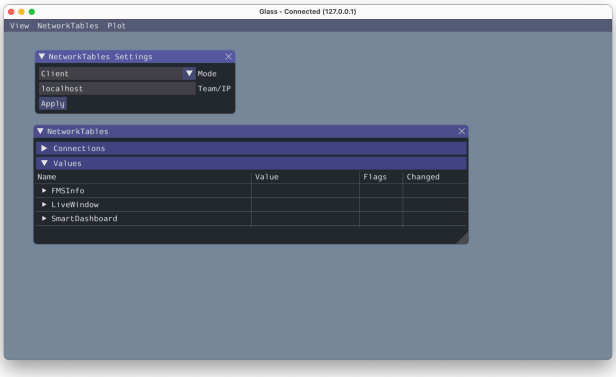
Glass 使用 NetworkTablesP951 协议与您的机器人程序建立连接。它还用于与机器人之间进行数据传输和接收。

连接机器人

首次启动 Glass 时，您将看到两个小部件 - "NetworkTables Settings" 和 "NetworkTables"。要连接到机器人，请在 "NetworkTables Settings" 小组件中的 "Mode" 下选择 "Client"，输入您的队伍编号，然后单击 "Apply"。



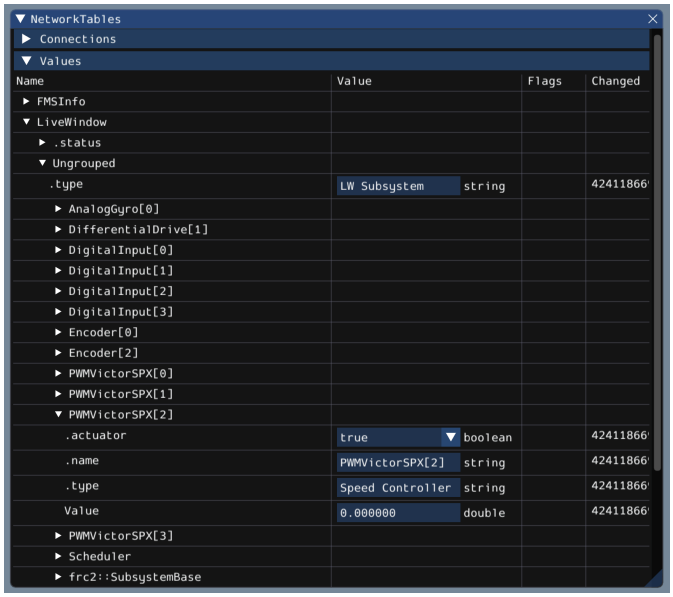
您还可以连接到在计算机上模拟运行的机器人（包括 Romi robots），方法是在“Team / IP ”框中输入 localhost。



重要： 网络表连接状态始终在 Glass 应用程序的标题栏上可见。

查看网络表条目

网络表小部件可用于查看通过网络表发送的所有条目。这些条目按主表，子表等按层次结构排列。



此外，您可以在小部件的“连接”窗格下查看所有已连接的网络表客户端。

11.4.3 Glass Widgets

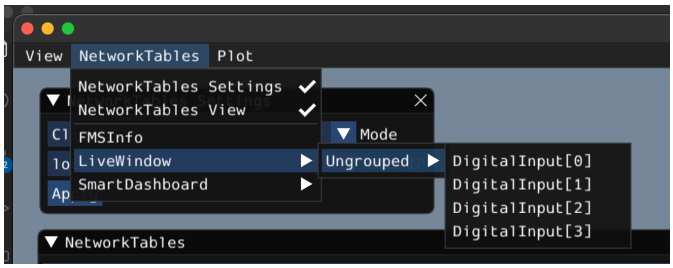
Specialized widgets are available for certain types that exist in robot code. These include objects that are manually sent over NetworkTables such as `SendableChooser` instances, or hardware that is automatically sent over *LiveWindow*.

备注： Widget support in Glass is still in its infancy –therefore, there are only a handful of widgets available. This list will grow as development work continues.

备注： A widget can be renamed by right-clicking on its header and specifying a new name.

Hardware Widgets

Widgets for specific hardware (such as motor controllers) are usually available via LiveWindow. These can be accessed by selecting the *NetworkTables* menu option, clicking on *LiveWindow* and choosing the desired widget.



The list of hardware (sent over LiveWindow automatically) that has widgets is below:

- `DigitalInput`

- `DigitalOutput`
- `SpeedController`
- `Gyro`

Here is an example of the widget for gyroscopes:

Sendable Chooser Widget

The *Sendable Chooser* widget represents a `SendableChooser` instance from robot code. It is often used to select autonomous modes. Like other dashboards, your `SendableChooser` instance simply needs to be sent using a NetworkTables API. The simplest is to use something like `SmartDashboard`:

JAVA

```
SmartDashboard.putData("Auto Selector", m_selector);
```

C++

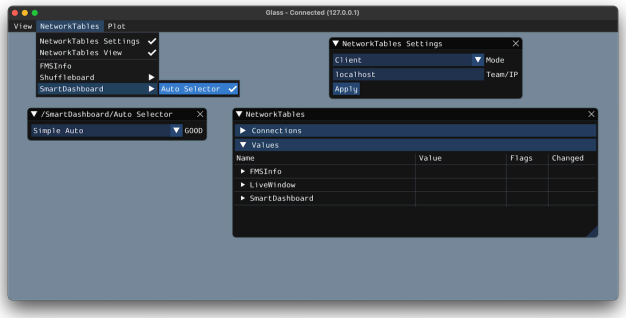
```
frc::SmartDashboard::PutData("Auto Selector", &m_selector);
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putData("Auto Selector", selector)
```

备注: For more information on creating a `SendableChooser`, please see *this document*.

The *Sendable Chooser* widget will appear in the *NetworkTables* menu and underneath the main table name that the instance was sent over. From the example above, the main table name would be *SmartDashboard*.



PID Controller Widget

The *PID Controller* widget allows you to quickly tune PID values for a certain controller. A `PIDController` instance must be sent using a NetworkTables API. The simplest is to use something like `SmartDashboard`:

JAVA

```
SmartDashboard.putData("Elevator PID Controller", m_elevatorPIDController);
```

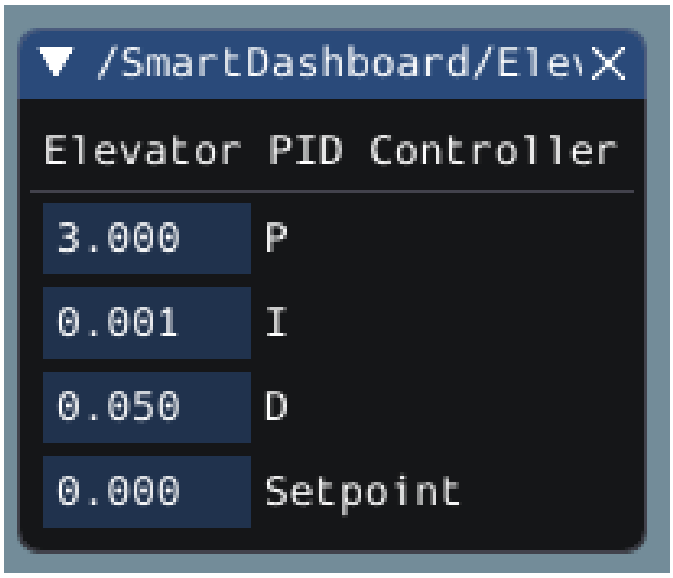
C++

```
frc::SmartDashboard::PutData("Elevator PID Controller", &m_elevatorPIDController);
```

PYTHON

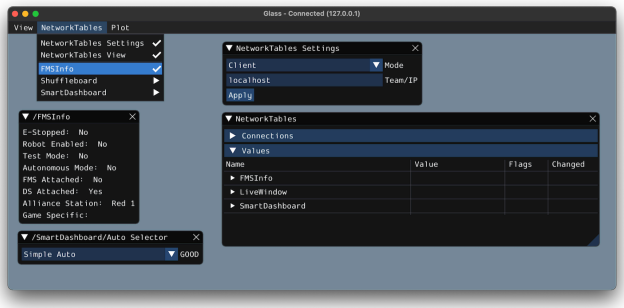
```
from wpilib import SmartDashboard

SmartDashboard.putData("Elevator PID Controller", elevatorPIDController)
```

This allows you to quickly tune P, I, and D values for various setpoints.



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 for the Command-Based Framework

Glass also has several widgets that are specific to the *command-based framework*. These include widgets to schedule commands, view actively running commands on a specific subsystem, or view the state of the *command scheduler*.

Command Selector Widget

The *Command Selector* widget allows you to start and cancel a specific instance of a command (sent over NetworkTables) from Glass. For example, you can create an instance of `MyCommand` and send it to SmartDashboard:

JAVA

```
MyCommand command = new MyCommand(...);
SmartDashboard.putData("My Command", command);
```

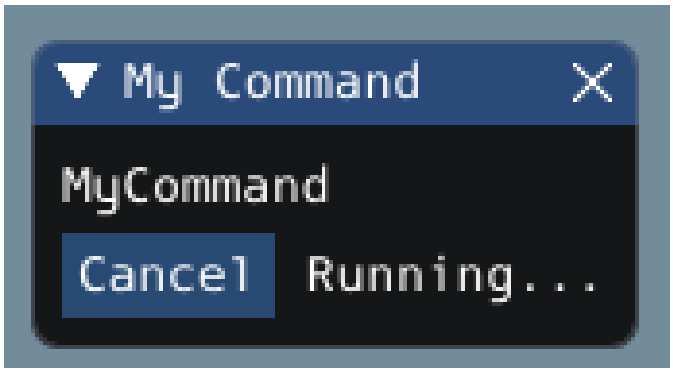
C++

```
#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)
```

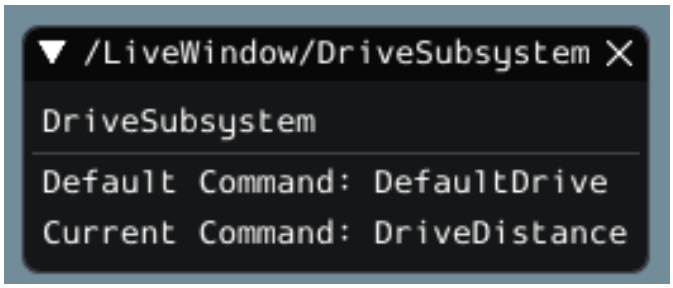
备注: The `MyCommand` instance can also be sent using a lower-level NetworkTables API or using the *Shuffleboard API*. In this case, the `SmartDashboard` API was used, meaning that the *Command Selector* widget will appear under the `SmartDashboard` table name.



The widget has two states. When the command is not running, a *Run* button will appear –clicking it will schedule the command. When the command is running, a *Cancel* button, accompanied by *Running*··· text, will appear (as shown above). This will cancel the command.

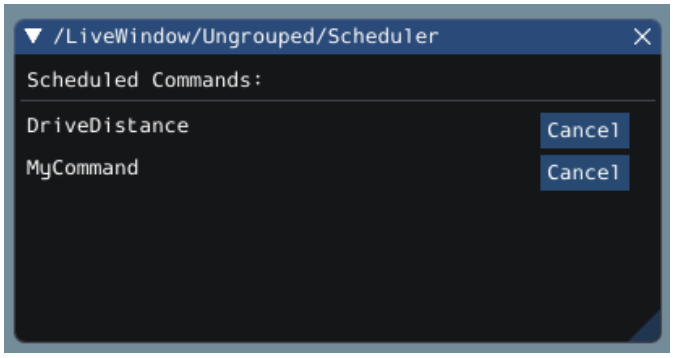
Subsystem Widget

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.



Command Scheduler Widget

The *Command Scheduler* widget allows you to see all currently scheduled commands. In addition, any of these commands can be canceled from the GUI.



The `CommandScheduler` instance is automatically sent to NetworkTables over LiveWindow. To view this widget, look under the *LiveWindow* main table name in the *NetworkTables* menu.

11.4.5 The Field2d Widget

Glass supports displaying your robot's position on the field using the *Field2d* widget. An instance of the `Field2d` class should be created, sent over NetworkTables, and updated periodically with the latest robot pose in your robot code.

Sending Robot Pose from User Code

To send your robot's position (usually obtained by *odometry* or a pose estimator), a `Field2d` instance must be created in robot code and sent over NetworkTables. The instance must then be updated periodically with the latest robot pose.

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())
```

备注: 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++

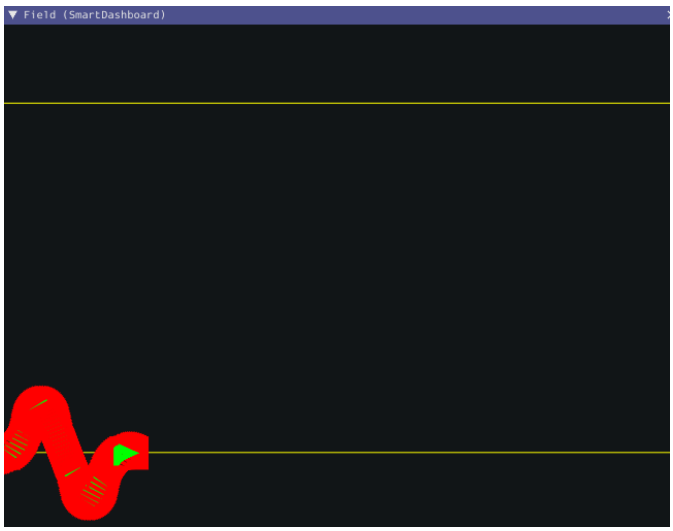
```
18  void AutonomousInit() override {
19    // Start the timer.
20    m_timer.Start();
21
22    // Send Field2d to SmartDashboard.
23    frc::SmartDashboard::PutData(&m_field);
24
25    // Reset the drivetrain's odometry to the starting pose of the trajectory.
26    m_drive.ResetOdometry(m_trajectory.InitialPose());
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

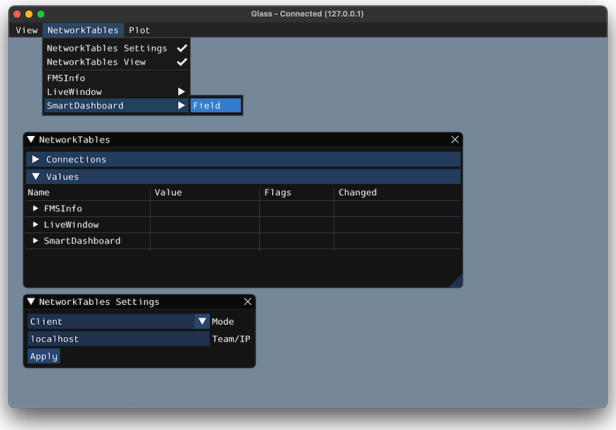
The sent trajectory can be viewed with *Glass* through the dropdown *NetworkTables -> SmartDashboard -> Field2d*.



备注: The above example which uses the RamseteController (Java / C++ / Python) will not show the sent trajectory until autonomous is enabled at least once.

Viewing the Robot Pose in Glass

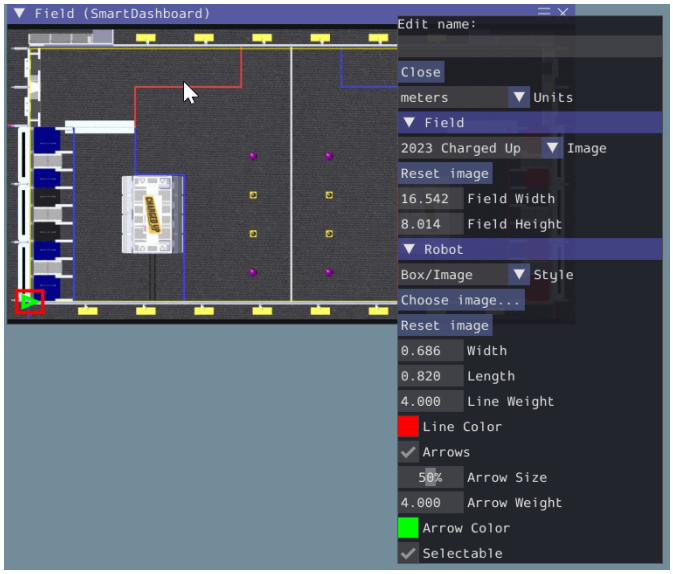
After sending the `Field2d` instance over NetworkTables, the *Field2d* widget can be added to Glass by selecting *NetworkTables* in the menu bar, choosing the table name that the instance was sent over, and then clicking on the *Field* button.



Once the widget appears, 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, select a custom field image, select a custom robot image, and choose the dimensions of the field and robot.

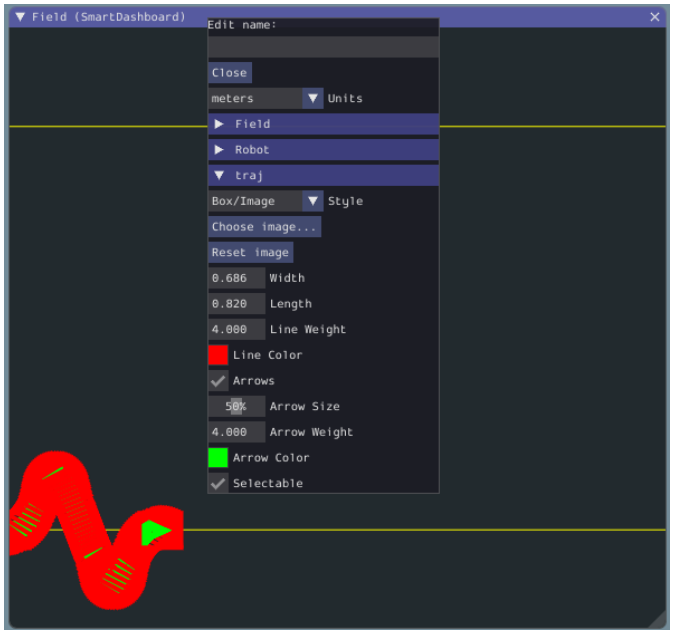
You can choose from an existing field layout using the *Image* drop-down. Or you can select a custom file by setting the *Image* to `Custom` and selecting *Choose image*⋯. You can choose to either select an image file or a PathWeaver JSON file as long as the image file is in the same directory. Choosing the JSON file will automatically import the correct location of the field in the image and the correct size of the field.

备注： 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*.

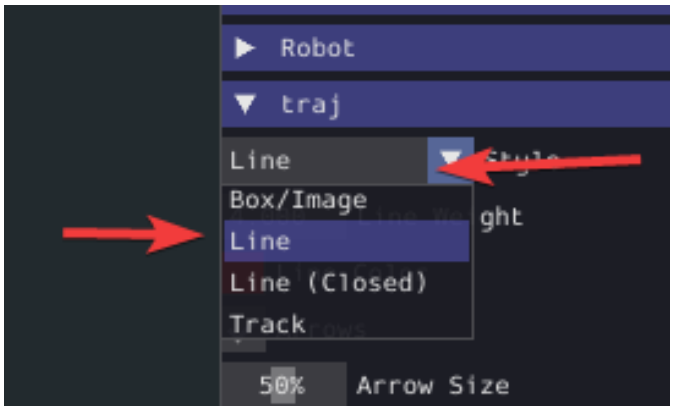


Modifying Pose Style

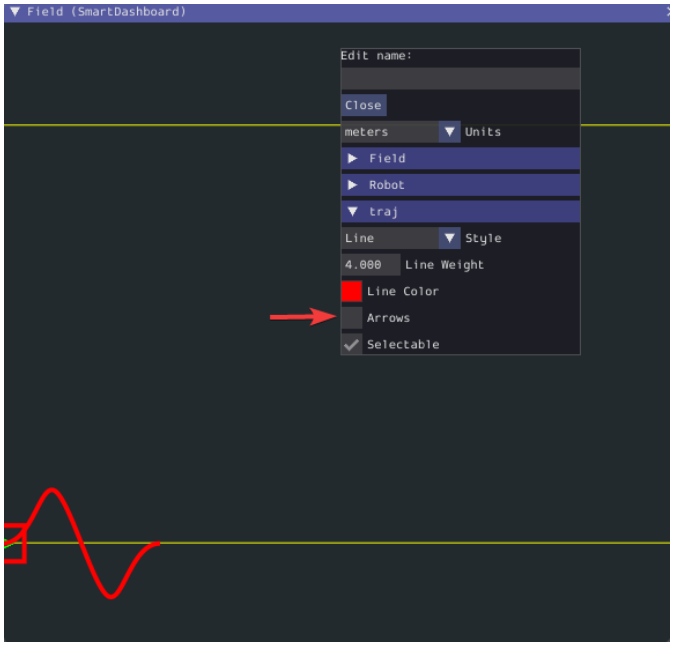
Poses can be customized in a plethora of ways by right clicking on the Field2d menu bar. Examples of customization are: line width, line weight, style, arrow width, arrow weight, color, etc.



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.

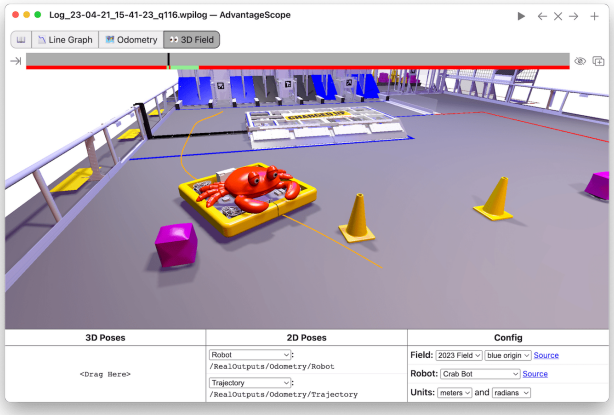


Now, uncheck the *Arrows* checkbox. This will cause our trajectory to look like a nice and fluid line!



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)
```

备注： 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++

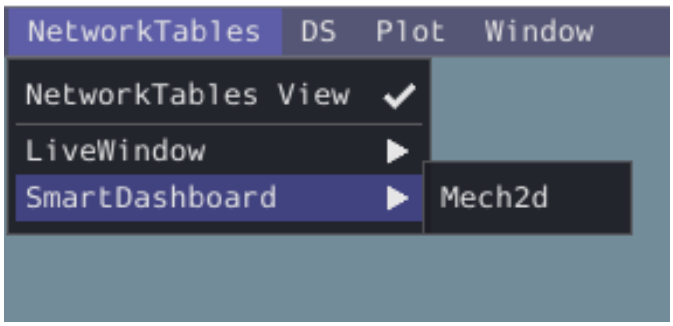
```
  void RobotPeriodic() override {
    // update the dashboard mechanism's state
    m_elevator->SetLength(kElevatorMinimumLength +
                          m_elevatorEncoder.GetDistance());
    m_wrist->SetAngle(units::degree_t{m_wristPotentiometer.Get()});
  }
```

PYTHON

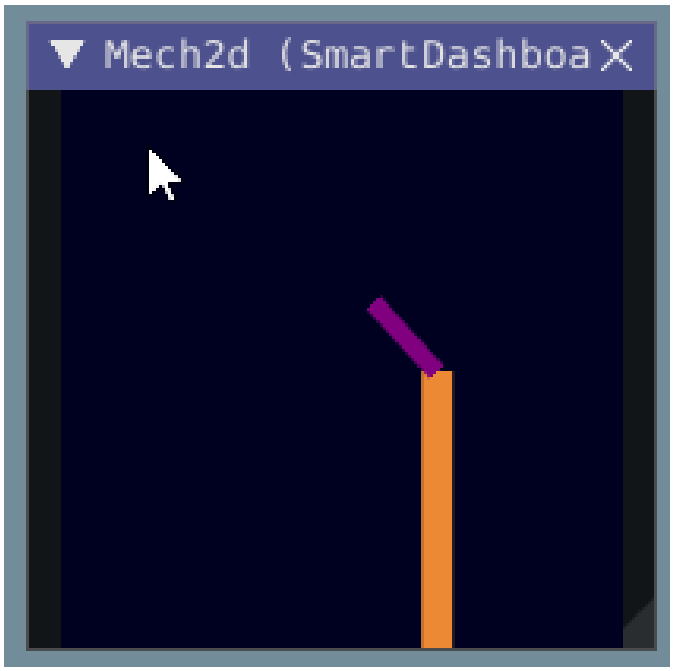
```
49      def robotPeriodic(self):
50          # update the dashboard mechanism's state
51          self.elevator.setLength(
52              self.kElevatorMinimumLength + self.elevatorEncoder.getDistance()
53          )
54          self.wrist.setAngle(self.wristPot.get())
```

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.

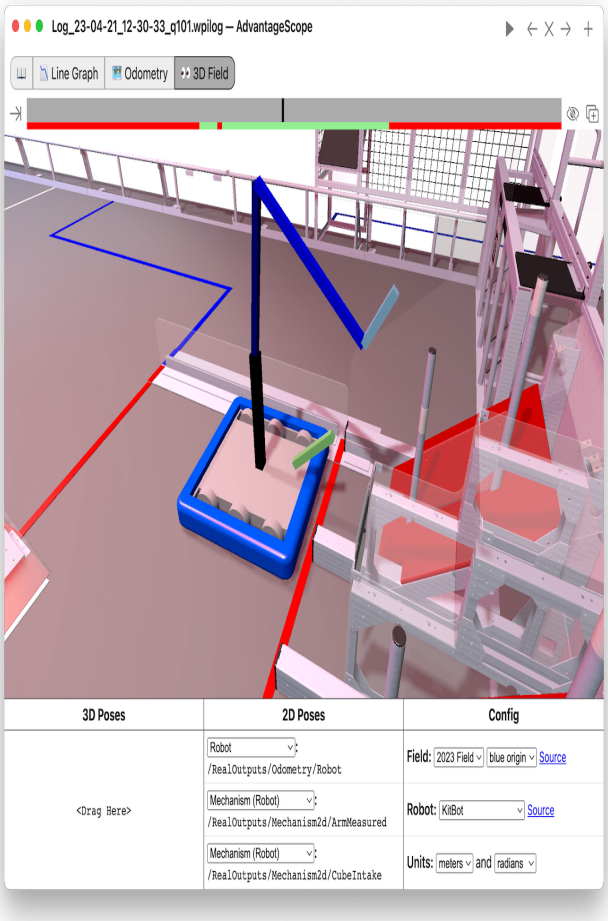


Once the widget appears as shown below, you can resize and place it on the Glass workspace as you desire. Right-clicking the top of the widget will allow you to customize the name of the widget. As the wrist potentiometer and elevator encoder changes, the mechanism will update in the widget.



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.



Next Steps

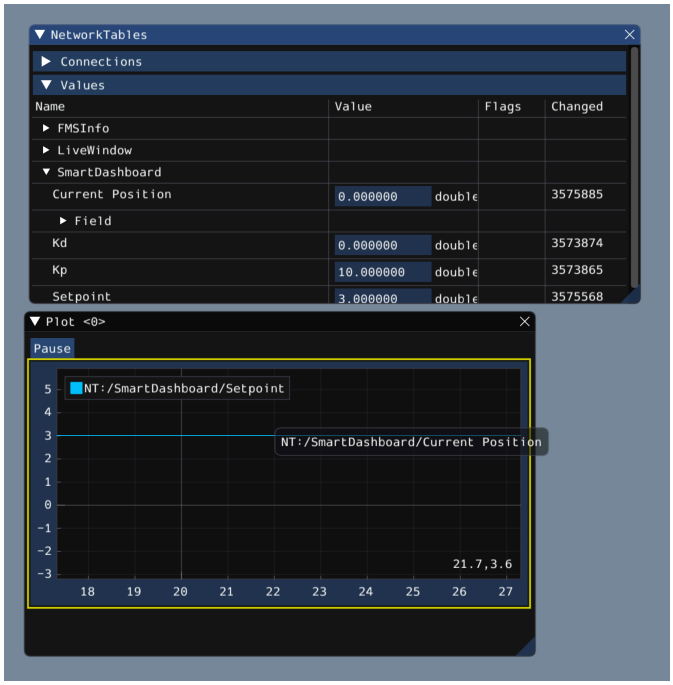
As mentioned above, the Mechanism2d visualization can be combined with *Physics Simulation* to help you program mechanisms before your robot is built. The ArmSimulation (Java / C++ / Python) and ElevatorSimulation (Java / C++ / Python) examples combine physics simulation and Mechanism2d visualization so that you can practice programming a single jointed arm and elevator without a robot.

11.4.7 Plots

Glass excels at high-performance, comprehensive plotting of data from NetworkTables. Some features include resizable plots, plots with multiple y axes and the ability to pause, examine, and resume plots.

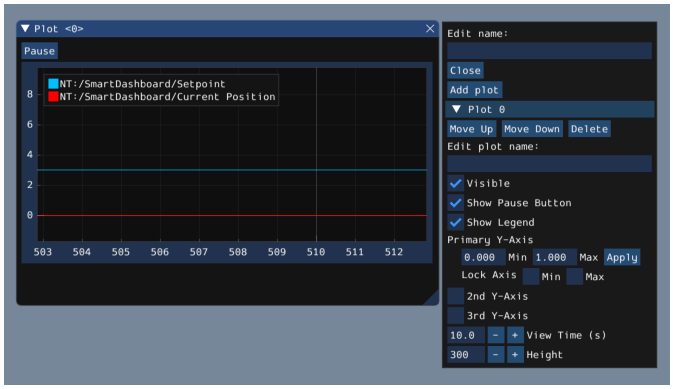
Creating a Plot

A new plot widget can be created by selecting the *Plot* button on the main menu bar and then clicking on *New Plot Window*. Several individual plots can be added to each plot window. To add a plot within a plot window, click the *Add plot* button inside the widget. Then you can drag various sources from the *NetworkTables* widget into the plot:

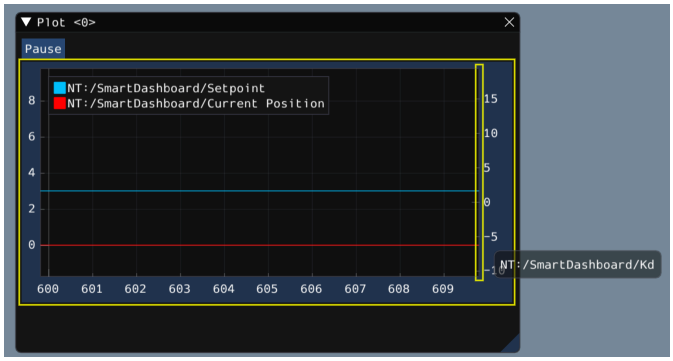


Manipulating Plots

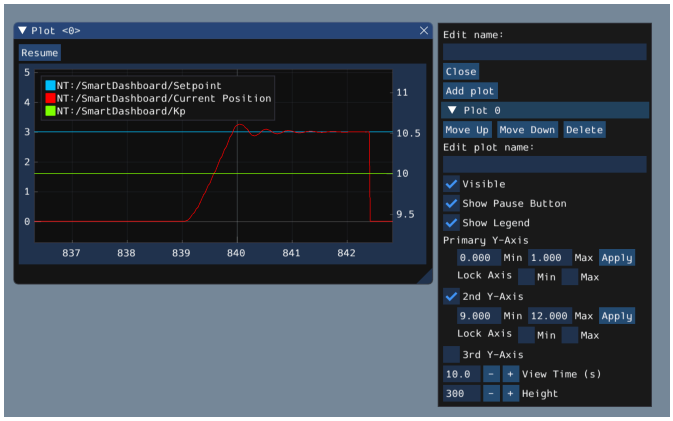
You can click and drag on the plot to move around and scroll on top of the plot to zoom the y axes in and out. Double clicking on the graph will autoscale it so that the zoom and axis limits fit all of the data it is plotting. Furthermore, right-clicking on the plot will present you with a plethora of options, including whether you want to display secondary and tertiary y axes, if you wish to lock certain axes, etc.



If you choose to make secondary and tertiary y axes available, you can drag data sources onto those axes to make their lines correspond with your desired axis:

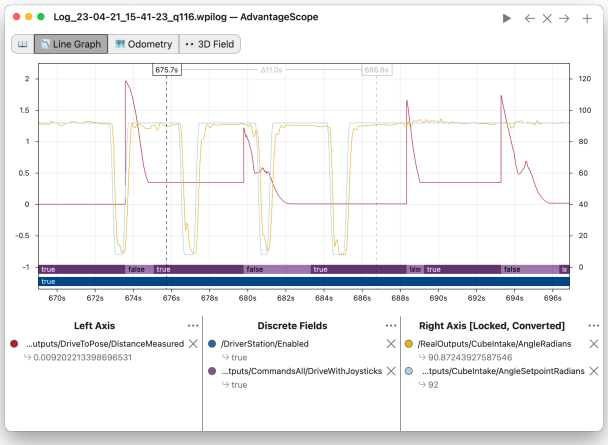


Then, you can lock certain axes so that their range always remains constant, regardless of panning. In this example, the secondary axis range (with the `/SmartDashboard/Kp` entry) was locked between 9 and 12.



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

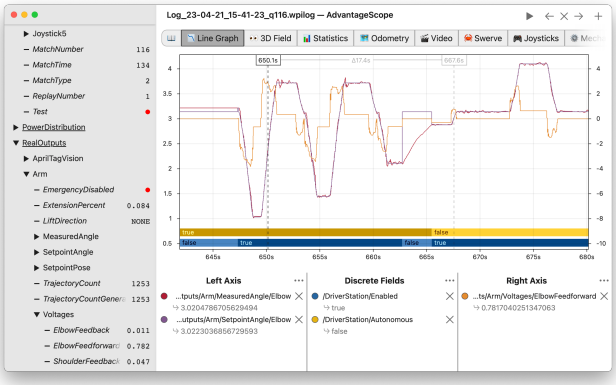
AdvantageScope is a data visualization tool for *NetworkTables*, *WPILib data logs*, and *Driver Station logs*. It is a programmer's tool (rather than a competition dashboard) and can be used to debug real or simulated robot code from a log file or live over the network.

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.

备注: 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



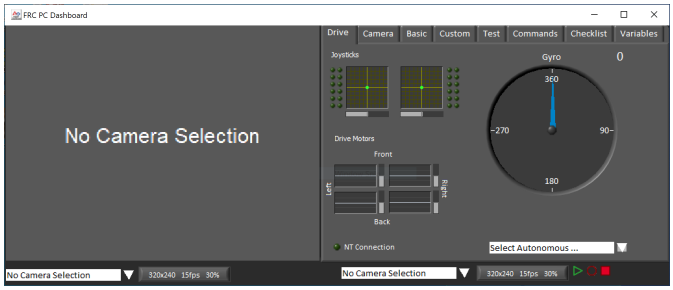
11.6 LabVIEW Dashboard

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 FRC LabVIEW Dashboard

The Dashboard application installed and launched by the FRC® Driver Station is a LabVIEW program designed to provide teams with basic feedback from their robot, with the ability to expand and customize the information to suit their needs. This Dashboard application uses *NetworkTables* and contains a variety of tools that teams may find useful.

LabVIEW Dashboard



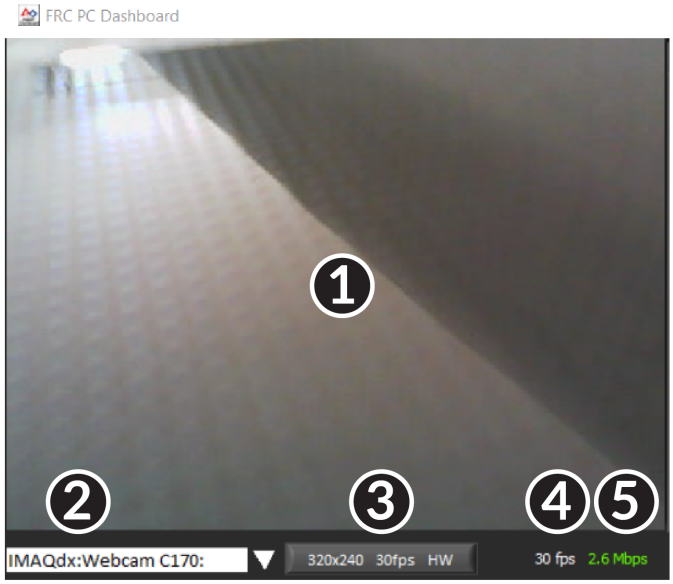
The Dashboard is broken into two main sections. The left pane is for displaying a camera image. The right pane contains:

- Drive tab that contains indicators for joystick and drive motor values (hooked up by default when used with LabVIEW robot code), a gyro indicator, an Autonomous selection text box, a connection indicator and some controls and indicators for the camera
- Basic tab that contains some default controls and indicators

- Camera tab that contains a secondary camera viewer, similar to the viewer in the left pane
- Custom tab for customizing the dashboard using LabVIEW
- Test tab for use with Test Mode in the LabVIEW framework
- Commands tab for use with the new LabVIEW C&C Framework
- Checklist tab that can be used to create task lists to complete before and/or between matches
- Variables tab that displays the raw NetworkTables variables in a tree view format

The LabVIEW Dashboard also includes Record/Playback functionality, located in the bottom right. More detail about this feature is included below under *Record/Playback*.

Camera Image and Controls

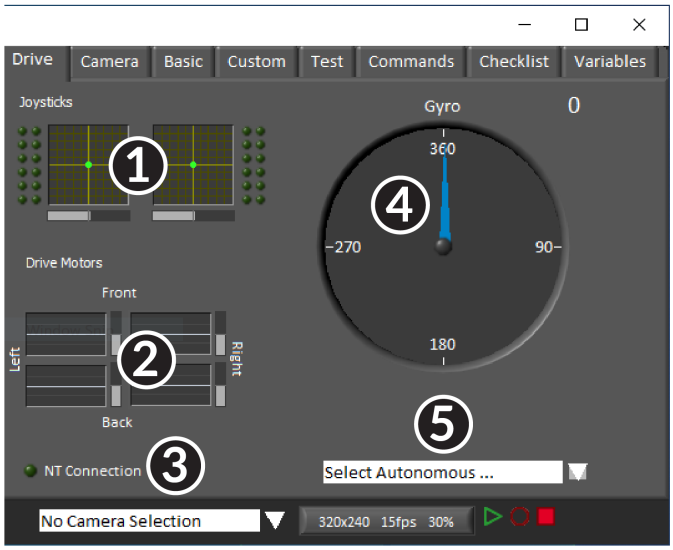


The left pane is used to display a video feed from acamera located on the robot. There are also some controls and indicators related to the camera below the tab area:

1. Camera Image Display
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 (Axis camera). Note that the IP Camera setting will not work when your PC is connected to the roboRIO over USB.
3. Camera Settings - This control allows you to change the resolution, framerate and compression of the image stream to the dashboard, click the control to pop-up the configuration.
4. Bandwidth Indicator - Indicates approximate bandwidth usage of the image stream. The indicator will display green for "safe" bandwidth usage, yellow when teams should use caution and red if the stream bandwidth is beyond levels that will work on the competition field.
5. Framerate - Indicates the approximate received framerate of the image stream.

小技巧: The bandwidth indicator indicates the combined bandwidth for all camera streams open.

Drive



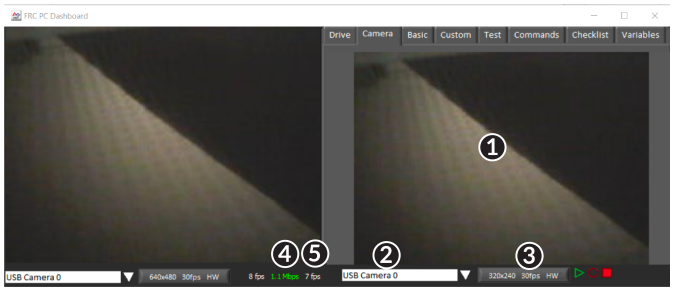
The center pane contains a section that provides feedback on the joysticks and drive commands when used with the LabVIEW framework and a section that displays the NetworkTables status and autonomous selector:

1. Displays X,Y and Throttle information and button values for up to 2 joysticks when using the LabVIEW framework
2. Displays values being sent to motor controllers when using LabVIEW framework
3. Displays a connection indicator for the NetworkTables data from the robot
4. Displays a Gyro value
5. Displays a text box that can be used to select Autonomous modes. Each language's code templates have examples of using this box to select from multiple autonomous programs.

These indicators (other than the Gyro) are hooked up to appropriate values by default when using the LabVIEW framework. For information on using them with C++/Java code see *Using the LabVIEW Dashboard with C++, Java, or Python Code*.

Camera

小技巧: The left pane can only display a single camera output, so use the camera tab on the right pane to display a second camera output if needed.

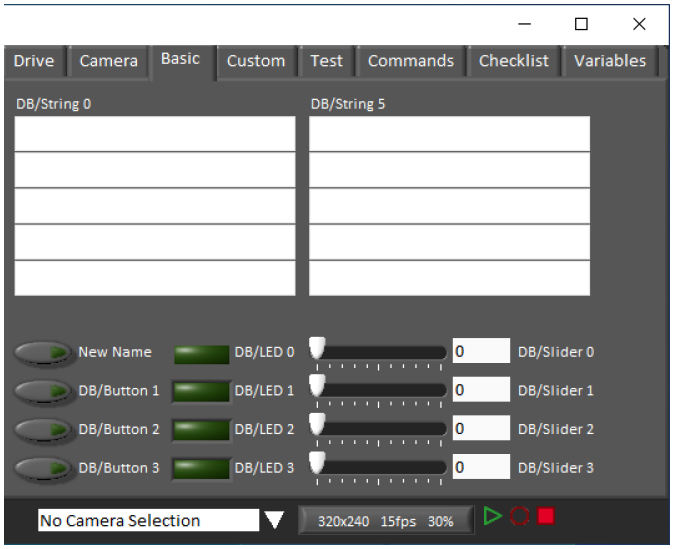


The camera tab is used to display a video feed from a camera located on the robot. There are also some controls and indicators related to the camera below the tab area:

1. Camera Image Display
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 (Axis camera). Note that the IP Camera setting will not work when your PC is connected to the roboRIO over USB.
3. Camera Settings - This control allows you to change the resolution, framerate and compression of the image stream to the dashboard, click the control to pop-up the configuration.
4. Bandwidth Indicator - Indicates approximate bandwidth usage of the image stream. The indicator will display green for "safe" bandwidth usage, yellow when teams should use caution and red if the stream bandwidth is beyond levels that will work on the competition field.
5. Framerate - Indicates the approximate received framerate of the image stream.

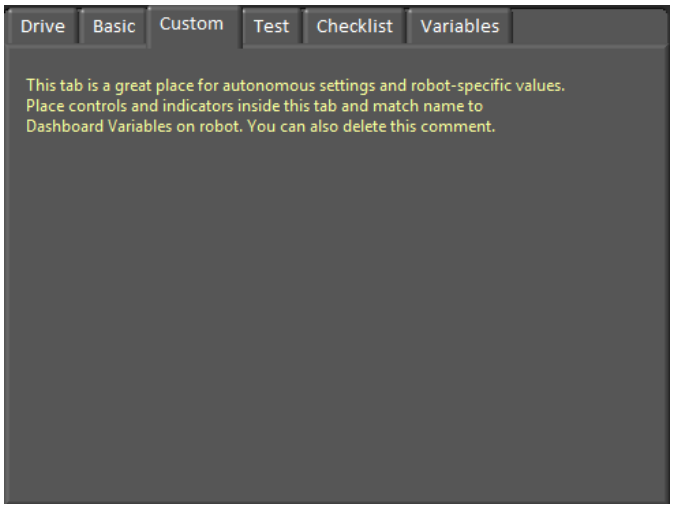
小技巧: The bandwidth indicator indicates the combined bandwidth for all camera streams open.

Basic



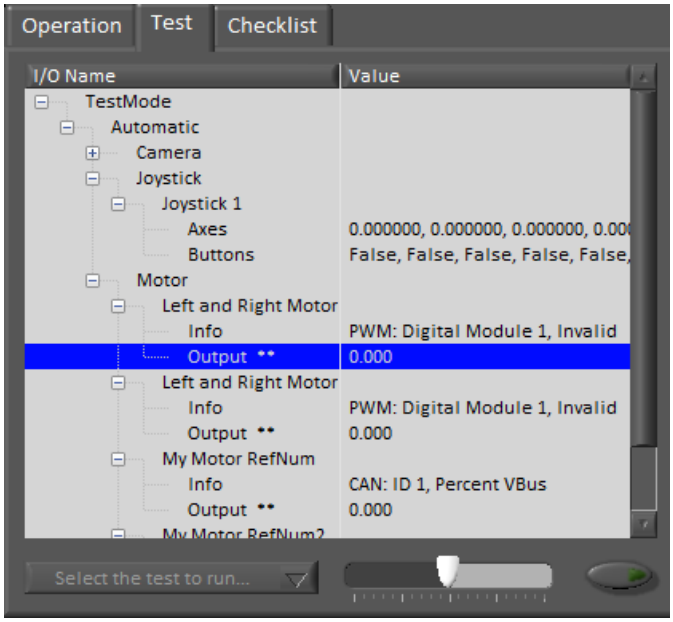
The Basic tab contains a variety of pre-populated bi-directional controls/indicators which can be used to control the robot or display information from the robot. The SmartDashboard key names associated with each item are labeled next to the indicator with the exception of the Strings which follow the same naming pattern and increment from DB/String 0 to DB/String 4 on the left and DB/String 5 to DB/String 9 on the right. The LabVIEW framework contains an example of reading from the Buttons and Sliders in Teleop. It also contains an example of customizing the labels in Begin. For more detail on using this tab with C++Java code, see *Using the LabVIEW Dashboard with C++, Java, or Python Code*.

Custom



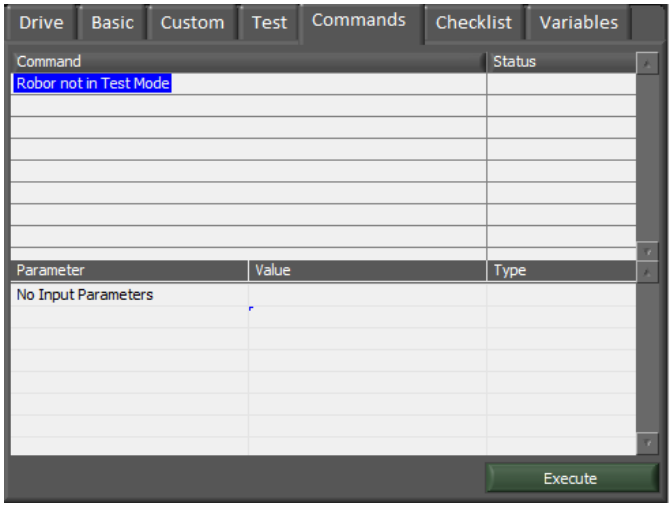
The Custom tab allows you to add additional controls/indicators to the dashboard using LabVIEW without removing any existing functionality. To customize this tab you will need to create a Dashboard project in LabVIEW.

Test



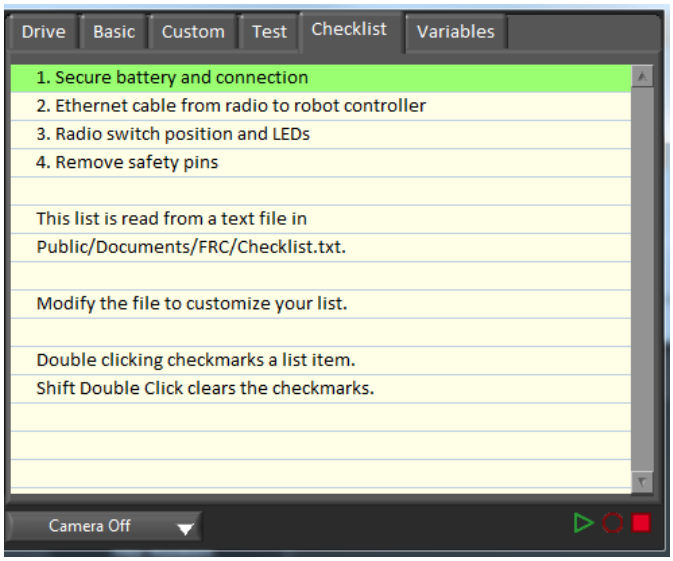
The Test tab is for use with Test mode for teams using LabVIEW (Java and C++ teams should use SmartDashboard or Shuffleboard when using Test Mode). For many items in the libraries, Input/Output info will be populated here automatically. All items which have ** next to them are outputs that can be controlled by the dashboard. To control an output, click on it to select it, drag the slider to set the value then press and hold the green button to enable the output. As soon as the green button is released, the output will be disabled. This tab can also be used to run and monitor tests on the robot. An example test is provided in the LabVIEW framework. Selecting this test from the dropdown box will show the status of the test in place of the slider and enable controls.

Commands



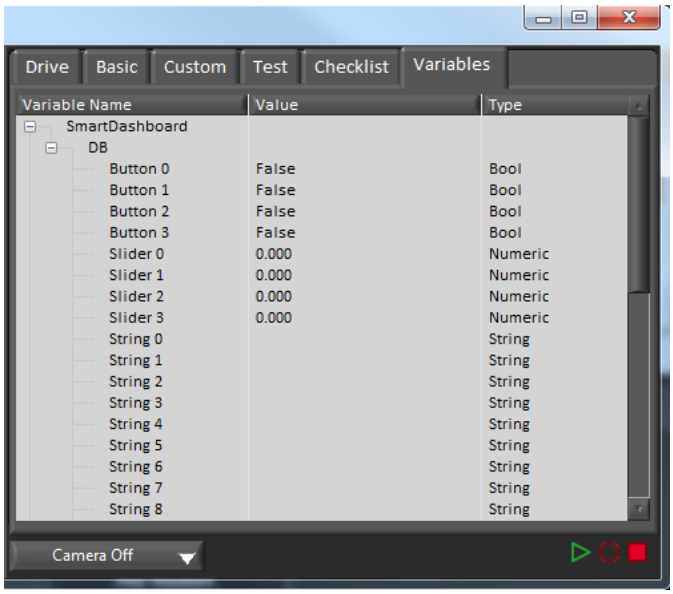
The Commands tab can be used with the Robot in Test mode to see which commands are running and to manually run commands for test purposes.

Checklist



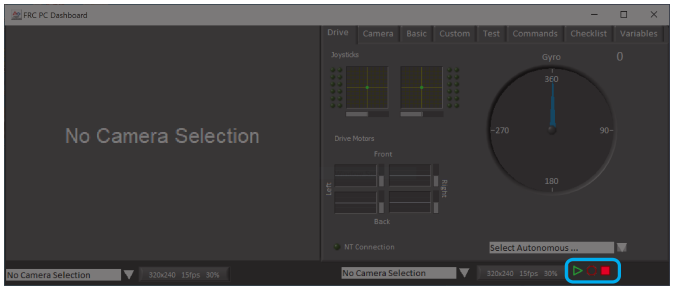
The Checklist tab can be used by teams to create a list of tasks to perform before or between matches. Instructions for using the Checklist tab are pre-populated in the default checklist file.

Variables



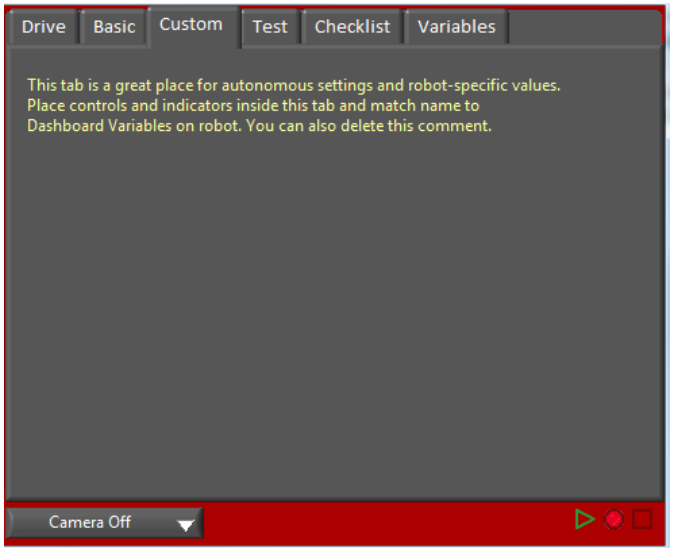
The Variables tab of the left pane shows all NetworkTables variables in a tree display. The Variable Name (Key), Value and data type are shown for each variable. Information about the NetworkTables bandwidth usage is also displayed in this tab. Entries will be shown with black diamonds if they are not currently synced with the robot.

Record/Playback



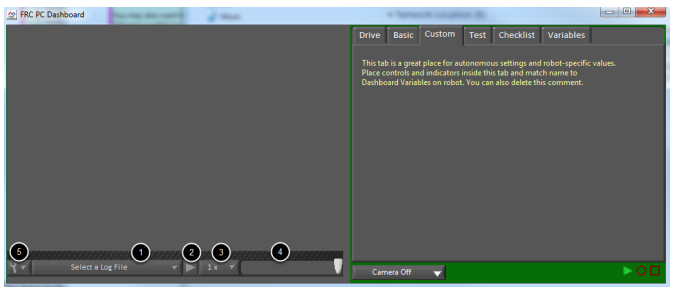
The LabVIEW Dashboard includes a Record/Playback feature that allows you to record video and NetworkTables data (such as the state of your Dashboard indicators) and play it back later.

Recording



To begin recording, click the red circular Record button. The background of the right pane will turn red to indicate you are recording. To stop recording, press the red square Stop button.

Playback



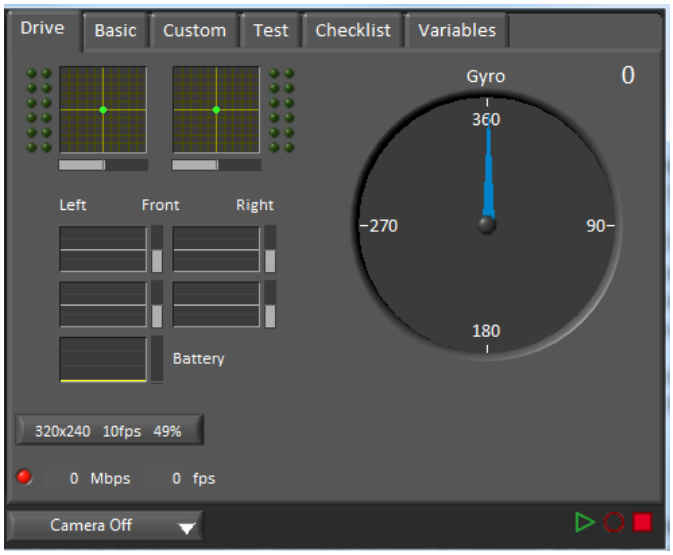
To play a recording back, click the green triangle Play button. The background of the right pane will begin pulsing green and playback controls will appear at the bottom of the camera pane.

1. File Selector - The dropdown allows you to select a log file to play back. The log files are named using the date and time and the dropdown will also indicate the length of the file. Selecting a logfile will immediately begin playing that file.
2. Play/Pause button - This button allows you to pause and resume playback of the log file.
3. Playback Speed - This dropdown allows you to adjust playback speed from 1/10 speed to 10x speed, the default is real-time (1x)
4. Time Control Slider - This slider allows you to fast-forward or rewind through the logfile by clicking on the desired location or dragging the slider.
5. Settings - With a log file selected, this dropdown allows you to rename or delete a file or open the folder containing the logs in Windows Explorer (Typically `C:\Users\Public\Documents\FRC\Log Files\Dashboard`)

11.6.2 Using the LabVIEW Dashboard with C++, Java, or Python Code

The default LabVIEW Dashboard utilizes *NetworkTables* to pass values and is therefore compatible with C++, Java, and Python robot programs. This article covers the keys and value ranges to use to work with the Dashboard.

Drive Tab



The *Select Autonomous⋯* dropdown can be used so show the available autonomous routines and choose one to run for the match.

JAVA

```
SmartDashboard.putStringArray("Auto List", {"Drive Forwards", "Drive Backwards",
↪"Shoot"});

// At the beginning of auto
String autoName = SmartDashboard.getString("Auto Selector", "Drive Forwards") // This␣
↪would make "Drive Forwards the default auto
switch(autoName) {
   case "Drive Forwards":
   // auto here
   case "Drive Backwards":
   // auto here
   case "Shoot":
   // auto here
}
```

C++

```
frc::SmartDashboard::PutStringArray("Auto List", {"Drive Forwards", "Drive Backwards",
↪ "Shoot"});

// At the beginning of auto
String autoName = SmartDashboard.GetString("Auto Selector", "Drive Forwards") // This␣
↪would make "Drive Forwards the default auto
switch(autoName) {
   case "Drive Forwards":
   // auto here
   case "Drive Backwards":
   // 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
```

Sending to the "Gyro" NetworkTables entry will populate the gyro here.

JAVA

```
SmartDashboard.putNumber("Gyro", drivetrain.getHeading());
```

C++

```
frc::SmartDashboard::PutNumber("Gyro", Drivetrain.GetHeading());
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putNumber("Gyro", self.drivetrain.getHeading())
```

There are four outputs that show the motor power to the drivetrain. This is configured for 2 motors per side and a tank style drivetrain. This is done by setting "RobotDrive Motors" like the example below.

JAVA

```
SmartDashboard.putNumberArray("RobotDrive Motors", {drivetrain.getLeftFront(),␣
↪drivetrain.getRightFront(), drivetrain.getLeftBack(), drivetrain.getRightBack()});
```

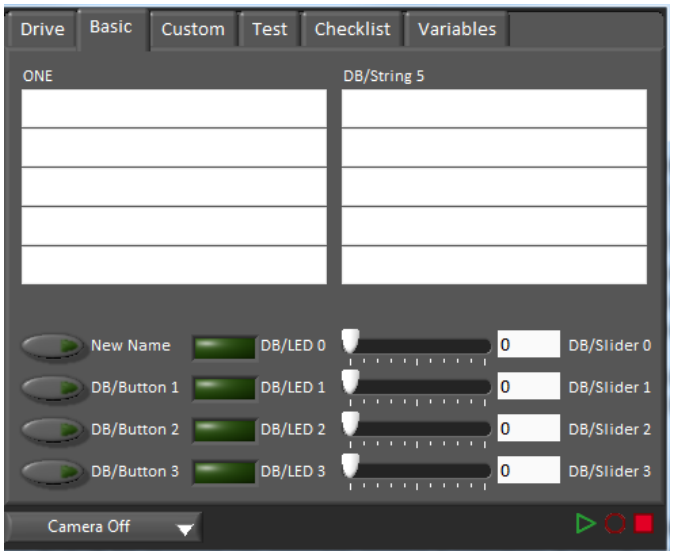
C++

```
frc::SmartDashboard::PutNumberArray("Gyro", {drivetrain.GetLeftFront(), drivetrain.
↪GetRightFront(), drivetrain.GetLeftBack(), drivetrain.GetRightBack()});
```

PYTHON

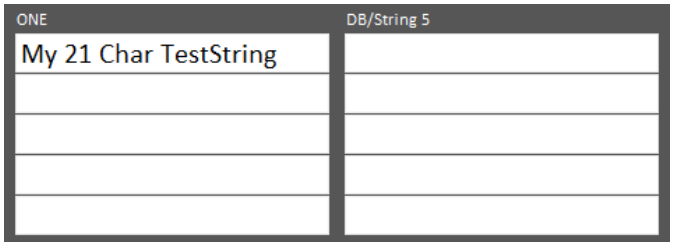
```
from wpilib import SmartDashboard

SmartDashboard.putNumberArray("RobotDrive Motors", [self.drivetrain.getLeftFront(),␣
↪self.drivetrain.getRightFront(), self.drivetrain.getLeftBack(), self.drivetrain.
↪getRightBack()])
```

Basic Tab



The Basic tab uses a number of keys in the a "DB" sub-table to send/receive Dashboard data. The LED's are output only, the other fields are all bi-directional (send or receive).

Strings



The strings are labeled top-to-bottom, left-to-right from "DB/String 0" to "DB/String 9". Each String field can display at least 21 characters (exact number depends on what characters). To write to these strings:

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")
```

To read string data entered on the Dashboard:

JAVA

```
String dashData = SmartDashboard.getString("DB/String 0", "myDefaultData");
```

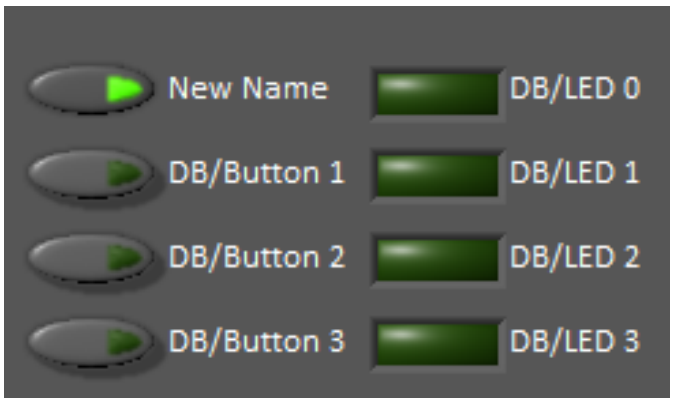
C++

```
std::string dashData = frc::SmartDashboard::GetString("DB/String 0", "myDefaultData");
```

PYTHON

```
from wpilib import SmartDashboard

dashData = SmartDashboard.getString("DB/String 0", "myDefaultData")
```

Buttons and LEDs



The Buttons and LEDs are boolean values and are labeled top-to-bottom from "DB/Button 0" to "DB/Button 3" and "DB/LED 0" to "DB/LED 3". The Buttons are bi-directional, the LEDs are only able to be written from the Robot and read on the Dashboard. To write to the Buttons or 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)
```

To read from the Buttons: (default value is false)

JAVA

```
boolean buttonValue = SmartDashboard.getBoolean("DB/Button 0", false);
```

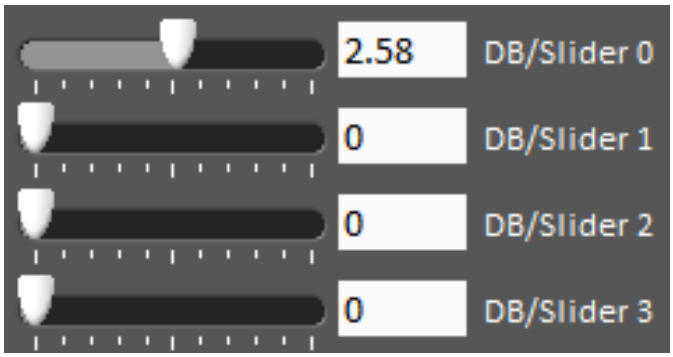
C++

```
bool buttonValue = frc::SmartDashboard::GetBoolean("DB/Button 0", false);
```

PYTHON

```
from wpilib import SmartDashboard

buttonValue = SmartDashboard.getBoolean("DB/Button 0", false)
```

Sliders



The Sliders are bi-directional analog (double) controls/indicators with a range from 0 to 5. To write to these indicators:

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)
```

To read values from the Dashboard into the robot program: (default value of 0.0)

JAVA

```
double dashData = SmartDashboard.getNumber("DB/Slider 0", 0.0);
```

C++

```
double dashData = frc::SmartDashboard::GetNumber("DB/Slider 0", 0.0);
```

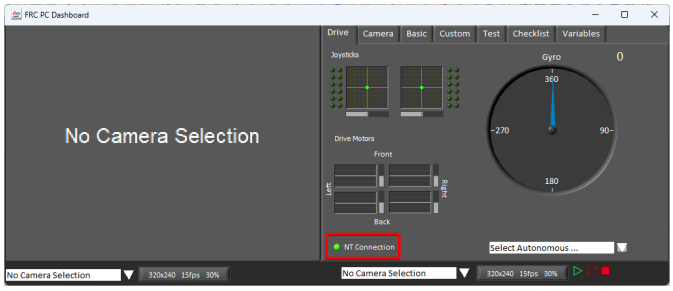
PYTHON

```
from wpilib import SmartDashboard

dashData = SmartDashboard.getNumber("DB/Slider 0", 0.0)
```

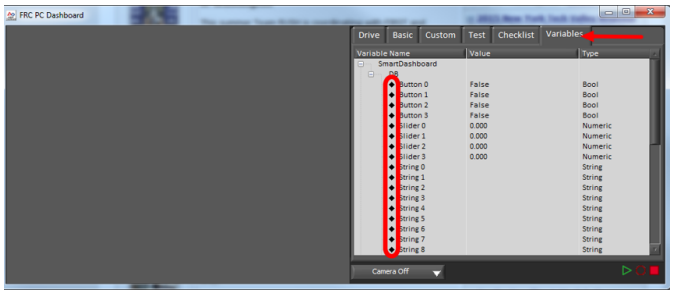
11.6.3 Troubleshooting Dashboard Connectivity

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

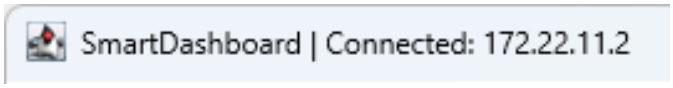


The LabVIEW Dashboard has a NetworkTables Connection indicator on the front panel.

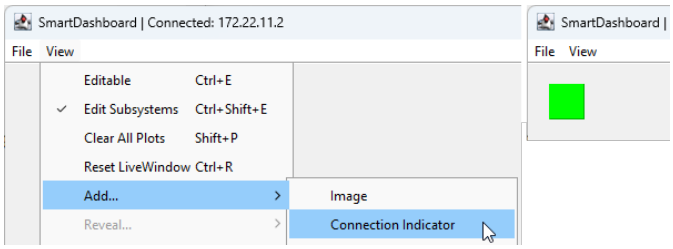


On the Variables tab of the Dashboard, the variables are shown with a black diamond when they are not synced with the robot. Once the Dashboard connects to the robot and these variables are synced, the diamond will disappear.

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.

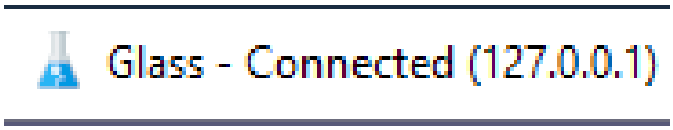


For more visibility, you can also add a Connection Indicator widget. The connection indicator can be moved or re-sized if the Editable checkbox is checked.

Recognizing Shuffleboard Connectivity

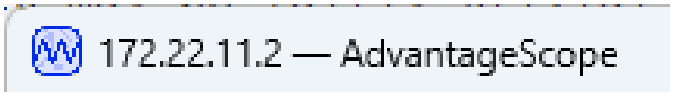
Shuffleboard indicates if it is connected or not in the bottom right corner of the application as shown in the image above. See *page* for more on configuring the connection.

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.

备注: 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

备注： 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 On-Robot Telemetry Recording Into Data Logs

By default, no telemetry data is recorded (saved) on the robot. The `DataLogManager` class provides a convenient wrapper around the lower-level `DataLog` class for on-robot recording of telemetry data into data logs. The WPILib data logs are binary for size and speed reasons. In general, the data log facilities provided by WPILib have minimal overhead to robot code, as all file I/O is performed on a separate thread–the log operation consists of mainly a mutex acquisition and copying the data.

12.3.1 Structure of Data Logs

Similar to NetworkTables, data logs have the concept of entries with string identifiers (keys) with a specified data type. Unlike NetworkTables, the data type cannot be changed after the entry is created, and entries also have metadata–an arbitrary (but typically JSON) string that can be used to convey additional information about the entry such as the data source or data schema. Also unlike NetworkTables, data log operation is unidirectional–the `DataLog` class can only write data logs (it does not support read-back of written values) and the `DataLogReader` class can only read data logs (it does not support changing values in the data log).

Data logs consist of a series of timestamped records. Control records allow starting, finishing, or changing the metadata of entries, and data records record data value changes. Timestamps are stored in integer microseconds; when running on the RoboRIO, the FPGA timestamp is used (the same timestamp returned by `Timer.getFPGATimestamp()`).

备注: For more information on the details of the data log file format, see the WPILib Data Log File Format Specification.

12.3.2 Standard Data Logging using DataLogManager

The `DataLogManager` class (Java, C++, `Python`) provides a centralized data log that provides automatic data log file management. It automatically cleans up old files when disk space is low and renames the file based either on current date/time or (if available) competition match number. The data file will be saved to a USB flash drive in a folder called `logs` if one is attached, or to `/home/lvuser/logs` otherwise.

备注: USB flash drives need to be formatted as FAT32 to work with the roboRIO. NTFS or exFAT formatted drives will not work.

Log files are initially named `FRC_TBD_{random}.wpilog` until the DS connects. After the DS connects, the log file is renamed to `FRC_yyyyMMdd_HHmmss.wpilog` (where the date/time is UTC). If the *FMS* is connected and provides a match number, the log file is renamed to `FRC_yyyyMMdd_HHmmss_{event}_{match}.wpilog`.

On startup, all existing log files where a DS has not been connected will be deleted. If there is less than 50 MB of free space on the target storage, `FRC_` log files are deleted (oldest to newest) until there is 50 MB free OR there are 10 files remaining.

The most basic usage of DataLogManager only requires a single line of code (typically this would be called from `robotInit`). This will record all NetworkTables changes to the data log.

JAVA

```java
import edu.wpi.first.wpilibj.DataLogManager;

// Starts recording to data log
DataLogManager.start();
```

C++

```cpp
#include "frc/DataLogManager.h"

// Starts recording to data log
frc::DataLogManager::Start();
```

PYTHON

```python
from wpilib import DataLogManager

DataLogManager.start()
```

DataLogManager provides a convenience function (`DataLogManager.log()`) for logging of text messages to the `messages` entry in the data log. The message is also printed to standard output, so this can be a replacement for `System.out.println()`.

DataLogManager also records the current roboRIO system time (in UTC) to the data log every ~5 seconds to the `systemTime` entry in the data log. This can be used to (roughly) synchronize the data log with other records such as DS logs or match video.

For custom logging, the managed `DataLog` can be accessed via `DataLogManager.getLog()`.

Logging Joystick Data

DataLogManager by default does not record joystick data. The `DriverStation` class provides support for logging of DS control and joystick data via the `startDataLog()` function:

JAVA

```java
import edu.wpi.first.wpilibj.DataLogManager;
import edu.wpi.first.wpilibj.DriverStation;

// Starts recording to data log
DataLogManager.start();

// Record both DS control and joystick data
DriverStation.startDataLog(DataLogManager.getLog());

// (alternatively) Record only DS control data
DriverStation.startDataLog(DataLogManager.getLog(), false);
```

C++

```
#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.

备注: 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;
```

(续下页)

(接上页)

```
DoubleLogEntry myDoubleLog;
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

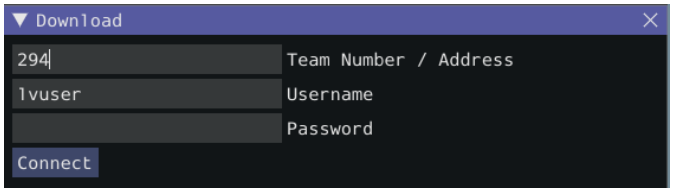
The DataLogTool desktop application integrates a SFTP client for downloading data log files from a network device (e.g. roboRIO or coprocessor) to the local computer.

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

备注: 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.

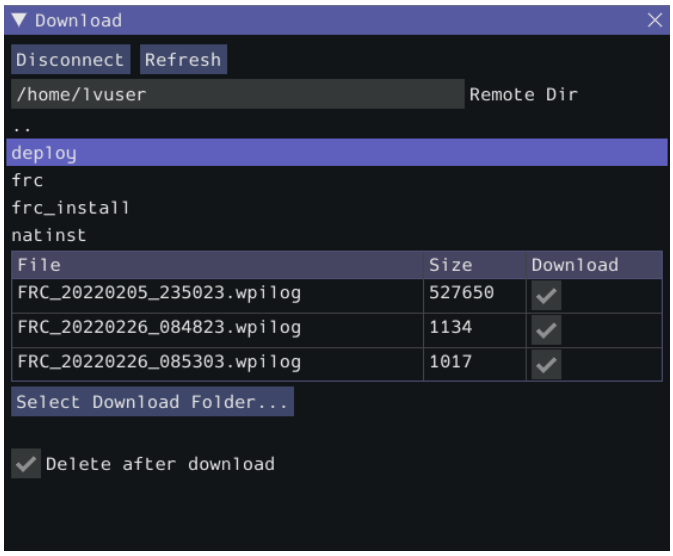
The tool also supports connecting to network devices other than the roboRIO, such as coprocessors, as long as the device supports SFTP password-based authentication.

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.

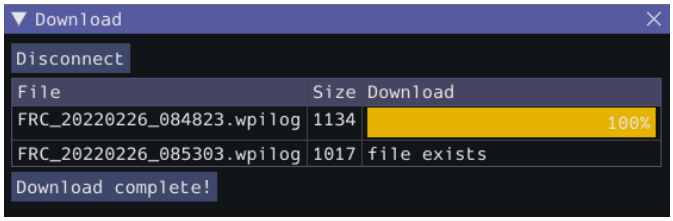
备注： On the roboRIO, log files are typically saved to either `/home/lvuser/logs` or `/u/logs` (USB stick location).



Click *Select Download Folder*··· to bring up a file browser for the local computer.

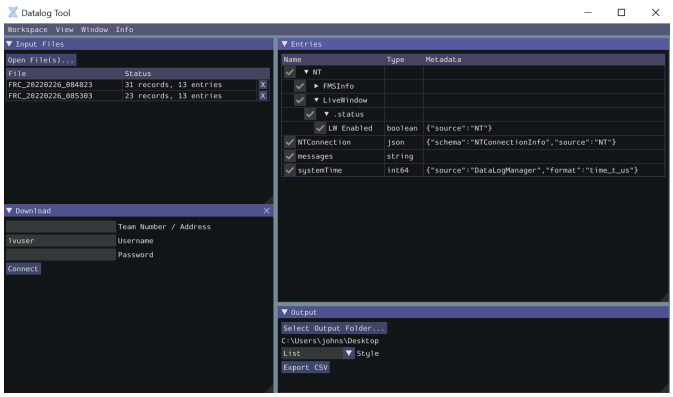
If you want to delete the files from the remote device after they are downloaded, check the *Delete after download* checkbox.

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.

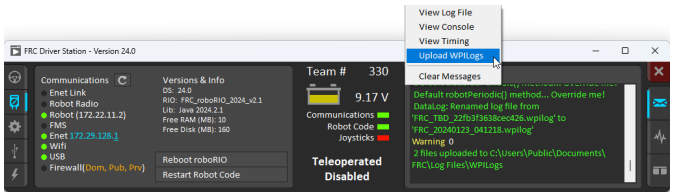


The conversion process is started by opening data log files in the "Input Files" window. Files are opened by clicking *Open File(s)*···. Summary status on each file (e.g. number of records and entries) is displayed. Clicking *X* in the table row closes the file.

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

Since the `Sendable` interface only has one method, writing your own classes that implement `Sendable` (and thus automatically log values to and/or consume values from the dashboard) is extremely easy: just provide an implementation for the overridable `initSendable` method, in which setters and getters for your class's fields are declaratively bound to key values (their display names on the dashboard).

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);
67    builder.AddBooleanProperty(
```

(续下页)

(接上页)

```
68        "atSetpoint", [this] { return AtSetpoint(); }, nullptr);
69  }
```

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

Like all WPILib dashboard code, `Sendable` fields are ultimately transmitted over *NetworkTables*, and thus the databinding methods provided by `SendableBuilder` match the supported NetworkTables data types:

- `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

```
120   @Override
121   public void initSendable(SendableBuilder builder) {
122     builder.setSmartDashboardType("Motor Controller");
123     builder.setActuator(true);
124     builder.setSafeState(this::disable);
125     builder.addDoubleProperty("Value", this::get, this::set);
126   }
```

C++

```
56  void PWMMotorController::InitSendable(wpi::SendableBuilder& builder) {
57    builder.SetSmartDashboardType("Motor Controller");
58    builder.SetActuator(true);
59    builder.SetSafeState([=, this] { Disable(); });
60    builder.AddDoubleProperty(
61        "Value", [=, this] { return Get(); },
62        [=, this](double value) { Set(value); });
```

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

小技巧: 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

FRC LabVIEW 编程

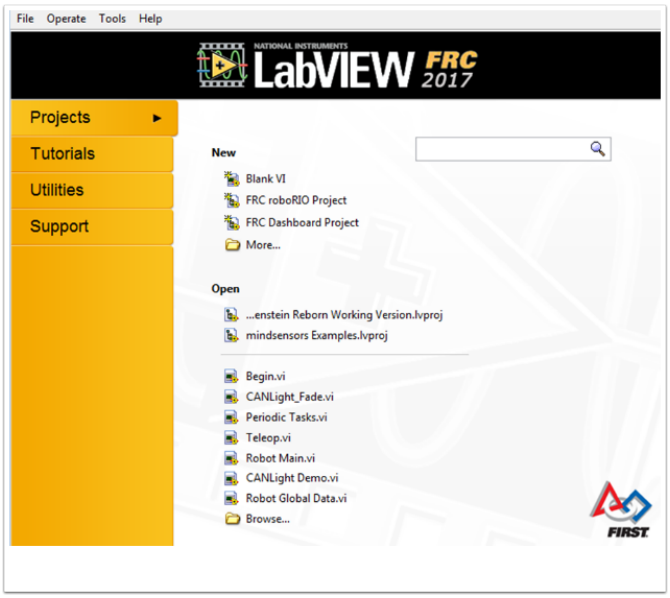
13.1 创建机器人程序

13.1.1 坦克驾驶模式教程

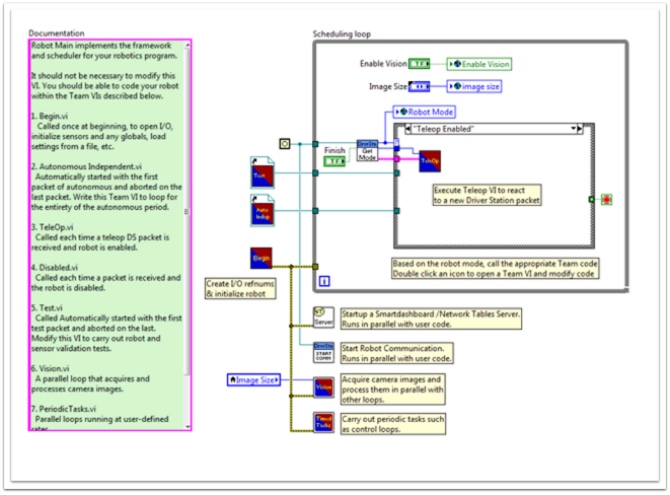
问题：如何使用坦克驾驶模式使我的机器通过两个操纵杆驱动?

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. 打开 LabVIEW 并双击 "FRC roboRIO Project"

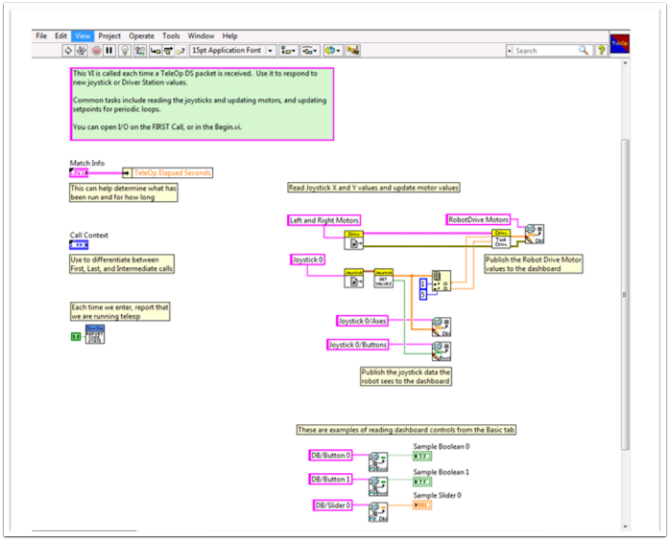


2. 为项目命名，添加团队编号，然后选择 Arcade Drive Robot roboRIO。您可以选择其他选项，但是，本教程将讨论如何为该项目设置坦克驾驶模式。
3. 在“项目资源管理器”窗口中，打开 Robot Main.vi
4. 按下 Ctrl+E 即可查看框图。它应如下图所示：



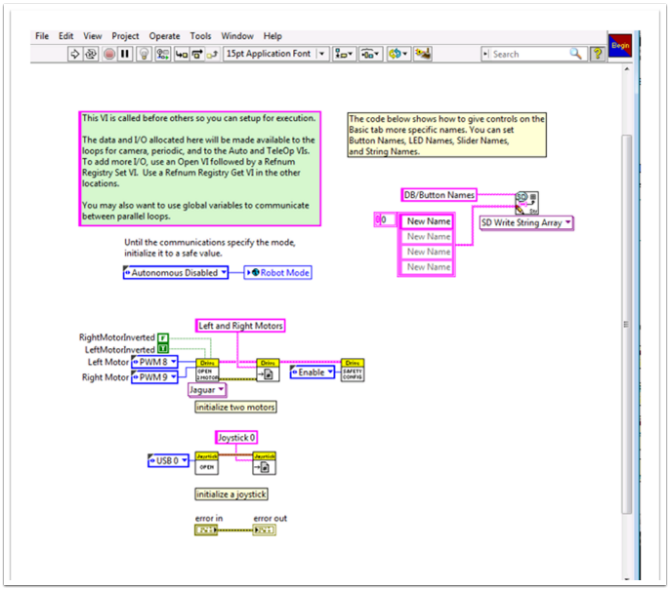
5. 双击启用 Teleop 的案例结构内部的“Teleop” vi。查看其框图。您将要在此处进行两项更改：

- 用 tank drive.vi 替换 Arcade Drive。右键单击程序框图 » WPI 机器库 » 机器驱动器 »，然后单击 Tank Drive VI，可以找到该文件。
- 找到位于 Get Values.vi 之后的索引数组函数。您将需要创建两个数字常量，并将每个常量连接到索引输入之一。您可以通过查看 FRC®Driver Station 中的 USB Devices 选项卡来确定每个索引的值。移动两个操纵杆以确定它们绑定到的数字（索引）。您可能希望对每个操纵杆使用 Y 轴索引。这是因为，当您希望电动机前进时，向上推操纵杆很直观；而当电动机反向时，推下操纵杆很直观。如果为每个轴选择 X 轴索引，则必须向左或向右移动操纵杆（X 轴方向）才能使电机转动。在我的设置中，我为左电机 Y 轴控制选择了索引 1，为右电机 Y 轴控制选择了索引 5。你可以在 LabVIEW 中看到如下图所示的调整：



6. 接下来，您将要返回到“Robot Main.vi”并双击“Begin.vi”。
7. 在该 VI 中首先要确认的是，您的左右电机连接到 LabVIEW 中与 PDP（配电面板）上相同的 PWM 线。
8. 在该 VI 中要确认的第二件事是“Open 2 Motor.vi”已选择了正确的电机控制器（Talon，Jaguar，Victor 等）。

   例如，我正在使用 Jaguar 电动机控制器，并且电动机已连接到 PWM 8 和 9。下图显示了我需要进行的更改：



9. 保存所有已进行调整的 Vi，现在就可以使用坦克驾驶模式来驱动机器了！

13.1.2 指令与控制教程

简介

命令与控制是为 2016 赛季添加的新 LabVIEW 模板，用于将机器代码植入命令和控制器以形成机器特定子系统的集合。每个子系统都有一个独立的控制循环或状态机制，以适当的速率运行，用于更新所需操作和设置节点的方法和高级命令。这使得自动代码非常容易构建命令的同步序列。同时，TeleOp 的好处在于它可以重复执行相同的命令而无需等待完成，从而可以根据驱动团队的输入轻松取消或启动新命令。每个子系统都有一个面板，显示其随时间变化的传感器和控制值以及命令跟踪，以帮助调试。

什么是指令与控制？

指令与控制系统认识到 FRC|reg| 机器倾向于由相对独立的机制组成，例如 Drive，Shooter，Arm 等。每一个机制都称为子系统，并且需要代码来协调子系统的各种传感器和执行器以完成要求的命令或动作，例如“合拢抓手”或“放下手臂”。该框架的主要原理之一是，每个子系统将具有一个独立的控制回路，该回路仅负责更新电机和其他执行器。子系统控制器外部的代码可以发出命令，这些命令可能会更改机器的输出，但不应直接更改任何输出。两者差异非常细微，但这意味着只能从项目中的一个位置更新输出。这通过使您能够查看发送到子系统的命令列表加快了调试行为异常的机器的速度，而不是通过在项目中搜索可能已修改输出的地方。无需在控制器外部修改代码也将添加其他传感器、更改齿轮或禁用机制变得更加容易。

游戏代码（主要由自动程序和 TeleOp 组成）通常需要更新节点并对某些方法的状态做出反应。对于自动程序，常见的做法是将机器的操作定义为一系列操作-前进到这里、拾取、搬运、射出等。可以将命令与附加逻辑顺序连接，以快速构建复杂的例程。对于 teleOp，相同的命令可以异步执行，从而使机器始终能够处理最新的驱动程序输入，如果实施得当，新命令将中断，从而使驱动团队可以快速响应现场条件，同时还利用自动化命令和命令序列。

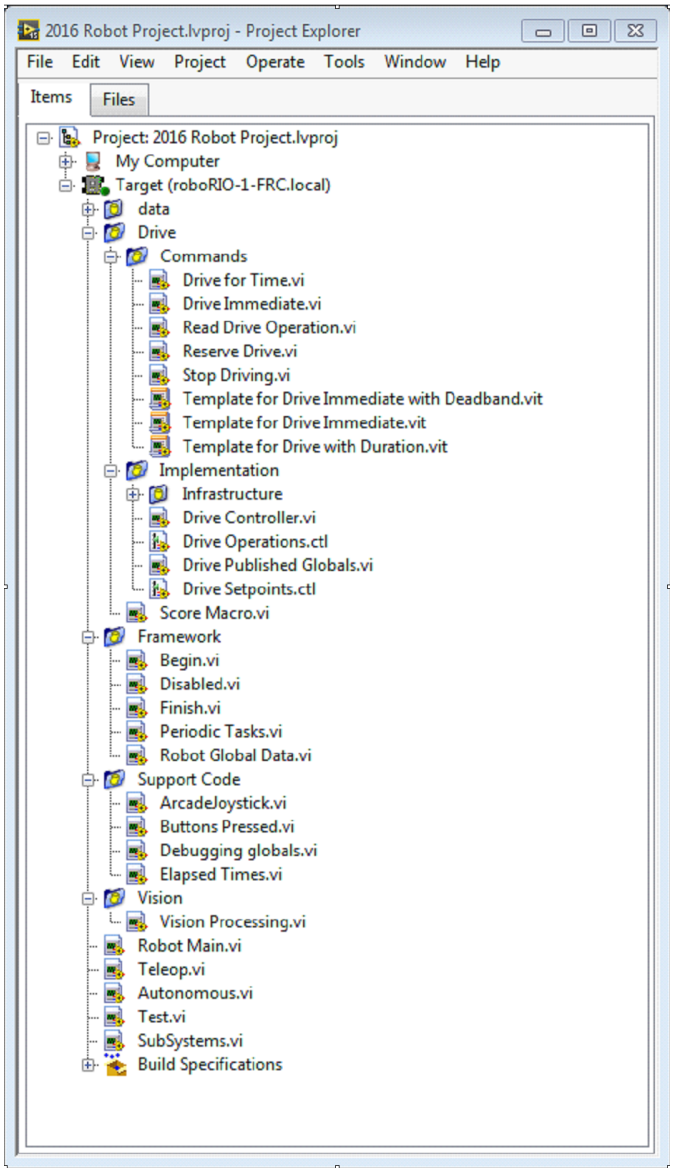
z 为什么要使用指令和控制？

指令与控制功能为现有的 LabVIEW 项目模板增加了功能，使代码可以通过更复杂的机器和机器代码更好地扩展。子系统用于抽象实现细节，游戏代码由高级指令 VI 组成的序列构建而成。这些指令本身是 VI，可以更新节点，在工程单位和机械单位之间执行数值缩放/映射并提供同步选项。如果对机器进行了物理更改，例如更改传动比，则可以仅对几个指令 Vi 进行更改，以在整个代码库中反映此更改。

当发生资源冲突时，I / O 封装可实现更可预测的操作和更快的调试。由于每个指令都是 VI，因此您可以单步执行命令，也可以使用内置的 Trace 功能查看发送给每个子系统的所有命令的列表。该框架使用异步通知和一致的数据传播，可以轻松地对序列进行编程命令或添加简单的逻辑以确定要运行的正确指令。

第 1 部分：项目资源管理器

项目资源管理器将用于为机器系统的所有 Vi 和文件提供组织。以下是项目资源管理器中有助于扩展系统的主要组件的说明。最常用的项目以粗体标出。



我的电脑
定义在项目被加载到的计算机上的操作的项目。对于机器项目，将其用作模拟目标并填充模拟文件。

Sim 支持文件
The folder containing 3D *CAD* models and description files for the simulated robot.

机器人仿真文档.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.

依赖
显示模拟机器代码使用的文件。

构造规范
这将包含有关定义如何为模拟机器目标构建并部署代码的文件。

目标 (roboRIO-TEAM-FRC.local)
定义 roboRIO 上位于（地址）上的操作与项目。

Drive

机器驱动基础的子系统执行和命令。这可以自行替代 WPILib RobotDrive VI。

框架
用于不属于子系统的机器代码的 VI 很少被使用。

开始
机器代码首次启动时调用一次。这对于不属于特定子系统的初始化代码很有用。

禁用
为每个禁用的数据包调用一次，当您不希望机器移动时，可用于调试传感器。

结束
在开发过程中，机器代码完成时可以调用此方法。在中止或电源关闭时不调用。

周期任务
临时循环进行调试或监视的好方法

机器全局数据
用于共享不属于子系统的机器信息。

支持代码
调试和代码开发辅助工具。

视图
相机和图像处理的子系统和命令。

Robot Main.vi
开发代码时将运行的最高级 VI。

Autonomous.vi
在自动阶段内运行的 VI。

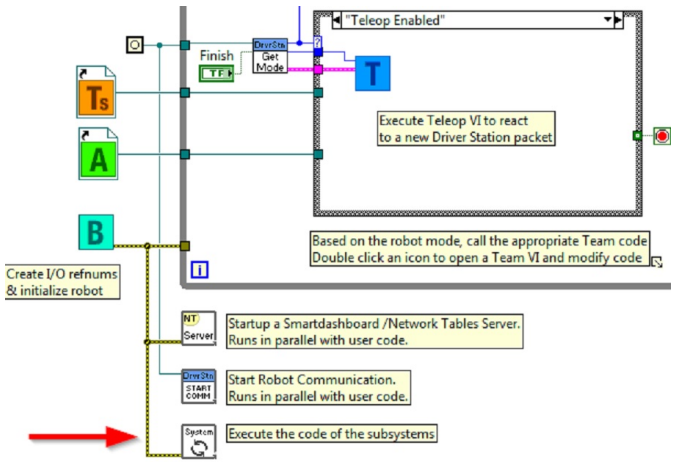
Teleop.vi
被每个 TeleOp 数据包调用的 VI。

Test.vi
在 driver station 处于测试模式时运行的 VI。

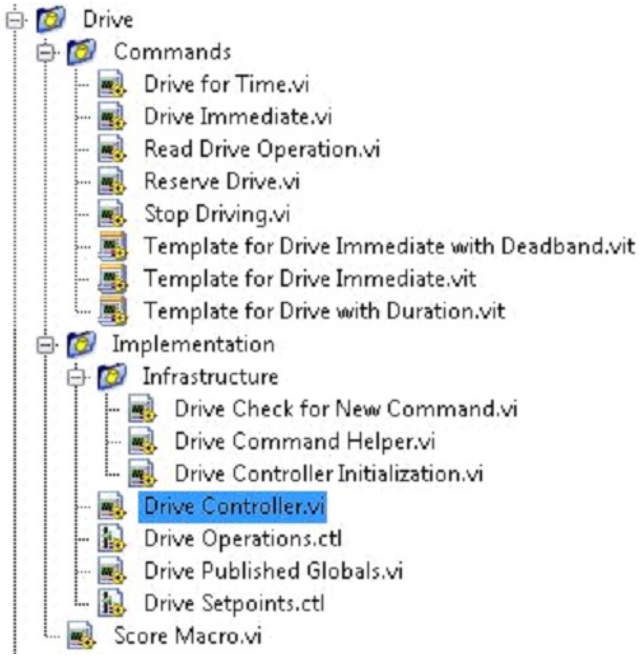
SubSystems.vi
包含并启动所有子系统的 VI。

依赖
显示机器代码使用的文件。

构造规范
一旦代码正常运行，它将被用于将代码作为启动应用程序构建和运行。



驱动器子系统项目资源管理器



命令：

该文件夹包含要求控制器执行操作的指令 VI。它还包含用于创建其他驱动器命令的模板。

备注： 创建新指令后，您可能需要对 `Drive Setpoints.ctl` 进行编辑以添加或更新控制器以定义新操作的字段。您还需要进入 Drive Controller.vi 并修改案例结构以为每个值添加案例。

执行

这些是用于构建子系统的 VI 和控件。

基础架构 VI

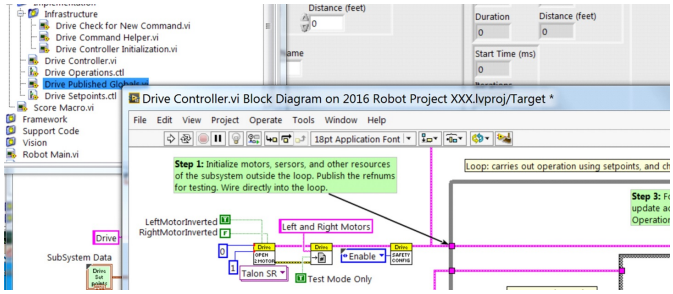
- 驱动器检查新指令：称为控制器循环的每次迭代。它检查新指令，定时更新数据，并在完成后通知等待的指令。
- 驱动器指令帮助.vi：命令调用该 VI 以通知控制器已发出新指令。
- 驱动器控制器初始化.vi：它分配通知程序，并将定时、默认指令和其他信息组合到一条数据线上。

- 驱动器控制器.vi：该 VI 包含控制/状态机制循环。该面板还可能包含对调试有用的显示。
- 驱动器操作.ctl：此 typedef 定义控制器的操作模式。许多命令可以共享一个操作。
- 驱动器设定.ctl：它包含 Drive 子系统的所有操作模式所使用的数据字段。
- 驱动器发布的全局变量.vi：发布有关驱动器子系统的全局信息的有用位置。

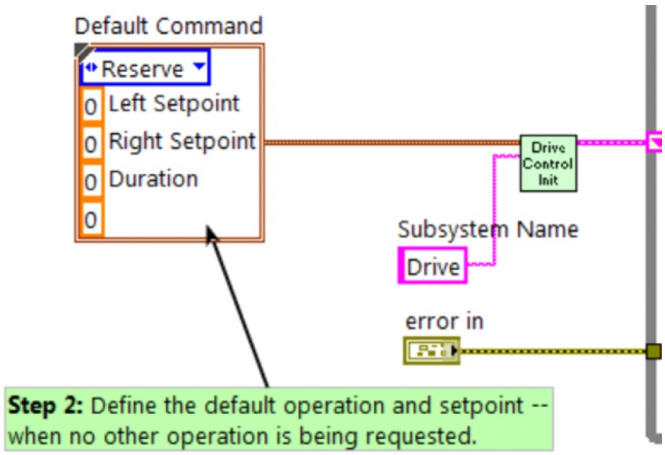
第 2 部分：初始化驱动器子系统

控制器程序框图上有绿色注释，在关键区域指出了如何编辑的方法。

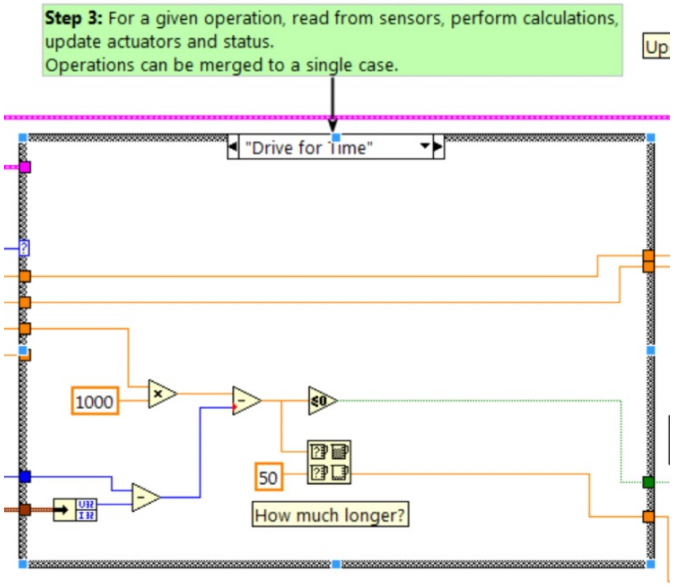
子系统启动时，控制循环左侧的区域将执行一次。通常在这里分配和初始化所有 I / O 和状态数据。您可以发布 I / O 引用句柄，也可以将它们注册为“测试模式”仅是为了使其私有，以便其他代码在不使用指令的情况下无法更新电动机。



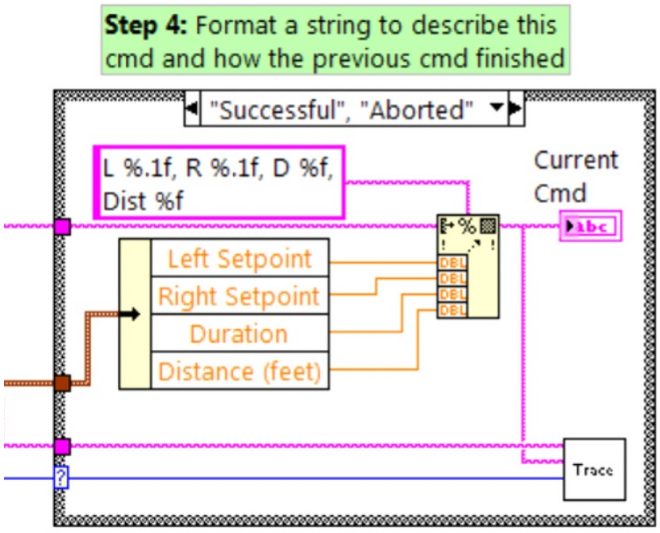
备注： 在各自的 Controller.vi 中而不是 Begin.vi 中初始化每个子系统的资源可以改善 I / O 封装，减少潜在的资源冲突并简化调试。



初始化的一部分是在不处理其他任何操作时选择默认操作和节点值。

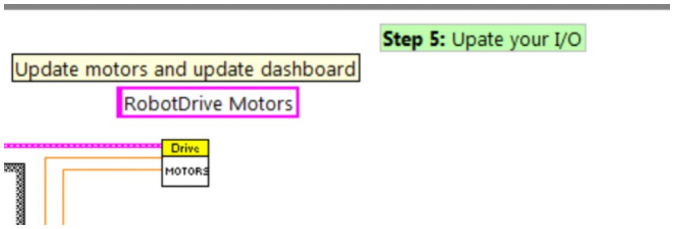


控制循环内部实际执行了操作的是一个 case 语句。设定节点值，迭代延迟，迭代计数和传感器都可能影响子系统的运行方式。此案例结构具有子系统的每个操作状态的值。



控制器循环的每次迭代将可选地更新跟踪 VI。该框架已经包含子系统名称，操作和描述，您可能会发现将其他设置点值格式化为跟踪信息很有用。在机器代码运行到当前设置点和发送到每个子系统的命令时，打开 Trace VI，然后单击“启动”。

控制器的主要目标是更新子系统的执行器。这可以在 case 结构内进行，但是很多时候，在结构的下游进行操作以确保始终在代码中的一个位置始终以正确的值更新值是有益的。



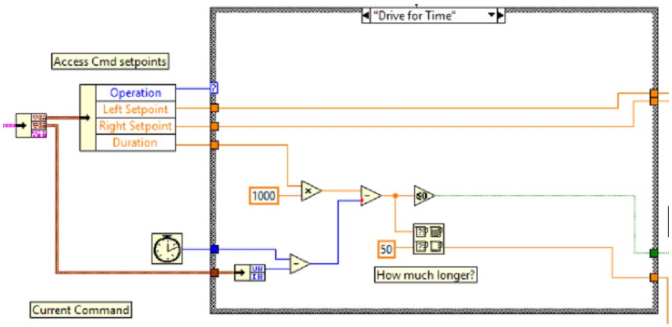
第 3 部分：驱动器子系统附带的命令

每个新子系统都有 3 个附带的示例命令：

Drive For Time.vi

该 VI 将电动机设置为运行给定的秒数。它可以选择与指令的完成同步。

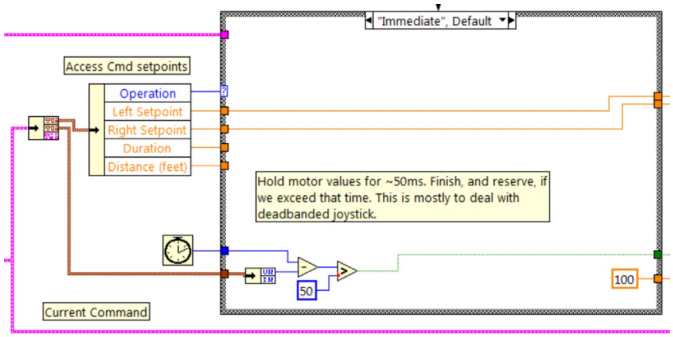
“时间驱动”情况将在设定点操作电动机，直到计时器过去或发出新指令为止。如果电动机启用了安全超时功能，则必须每 100 毫秒至少更新一次电动机。这就是代码等待剩余时间和 50ms 中较小者的原因。



Drive Immediate.vi

获得所需的电动机左右速度，并将电动机立即设置为这些设定点。

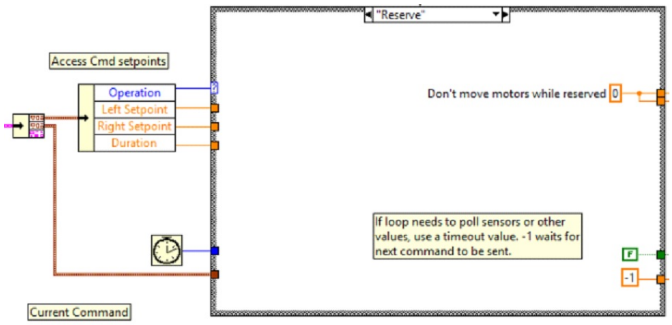
立即将电动机情况更新到指令定义的设定点。该指令未完成，因为您希望电动机在输入新指令或超时值之前保持该值。每当指令包含死区时，超时都是有用的。如果该值小于死区，则不会请求较小的值，且将导致咆哮或爬行除非指令超时。



Stop Driving.vi

将驱动马达调零，使机器静止。

预定命令关闭电动机，并等待新指令。当与命名指令序列一起使用时，即使当前未移动机械臂，预定指令也标识驱动子系统是序列的一部分。这有助于在同时运行的指令之间仲裁子系统资源。

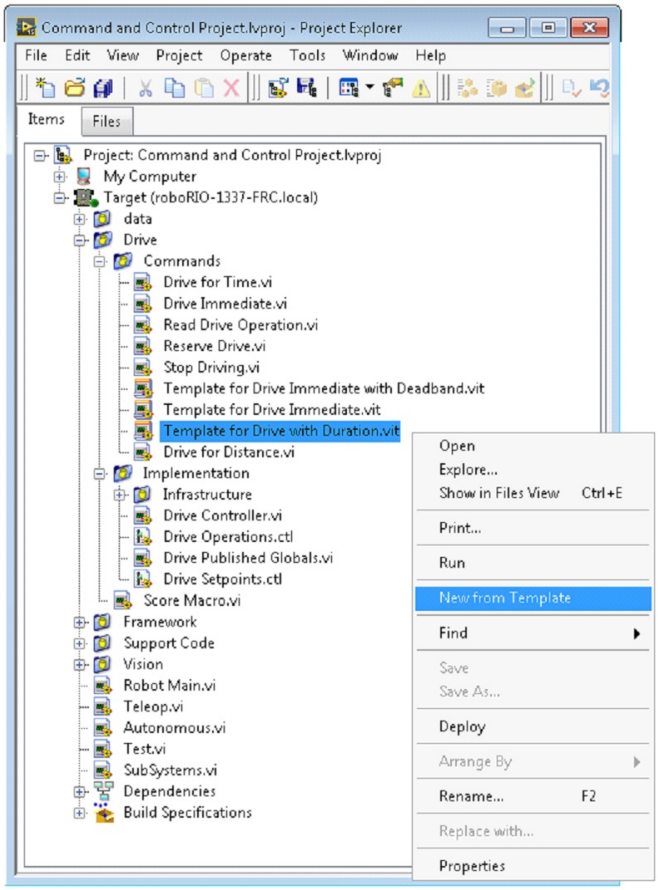


第 4 部分：创建新命令

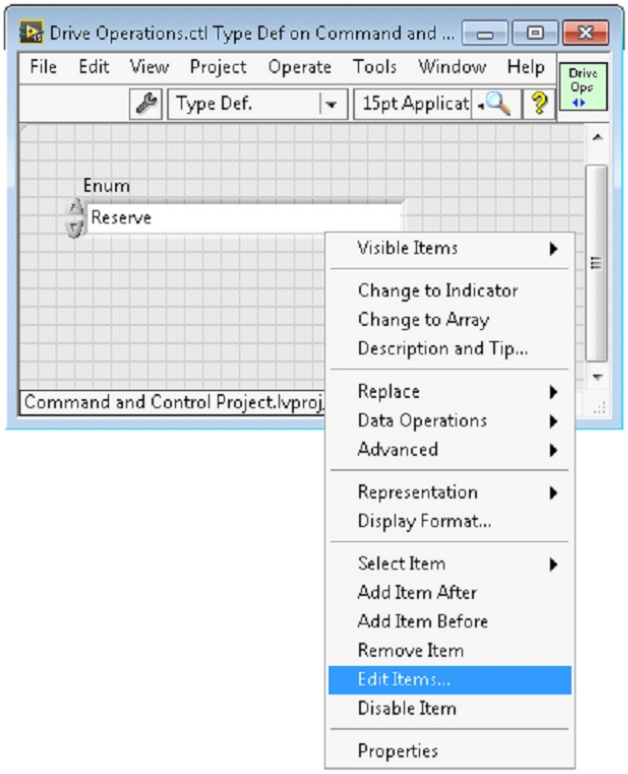
指令和控制框架使用户可以轻松地为子系统创建新指令。要创建新指令，请打开子系统文件夹/命令，在项目浏览器窗口中，选择一个 VI 模板作为新指令的起点，右键单击并选择“从模板新建”。

- **瞬时**：该 VI 将新的设定值通知子系统。
- **带有死区的瞬时**：该 VI 将输入值与死区进行比较，并有选择地将新的设定值通知子系统。当使用操纵杆连续值时，这非常有用。
- **持续时间**：该 VI 通知子系统在给定的持续时间内执行该命令，然后返回默认状态。同步决定该 VI 是启动该操作并立即返回，还是等待该操作完成。第一个选项通常用于 TeleOp，第二个选项用于自动排序。

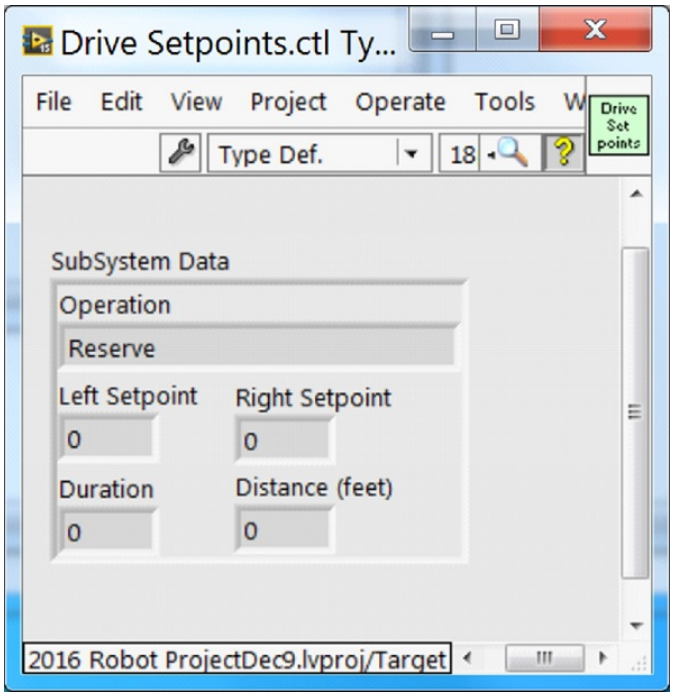
在此示例中，我们将添加新指令“Drive for Distance”。



首先，使用诸如“Drive for Distance”之类的描述性名称保存新的 VI。接下来，确定新指令是否需要添加驱动器操作枚举 typedef 的新值。初始项目代码已经具有枚举 Drive for Distance，但是下图显示了如何在需要时添加一个。

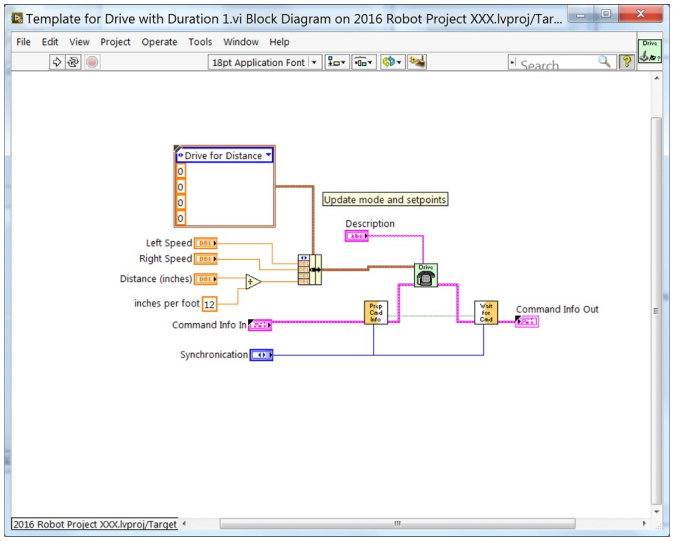


如果指令需要其他信息才能执行，请将其添加到设定控件中。驱动器子系统具有“左设定点”，“右设定点”和“持续时间”字样以及要执行的操作。默认情况下，“行驶距离”指令可以将“持续时间”重新用作距离，但让我们继续向“行驶距离”设置一个数字控件，称为“距离（英尺）”。

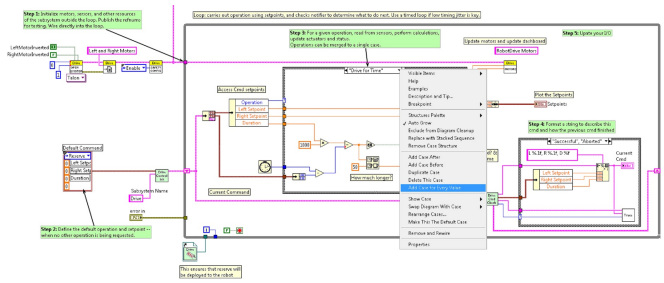


一旦有了指定指令所需的所有字段，便可以修改新创建的 Drive for Distance.vi。如下所示，从枚举的下拉菜单中选择“Drive for Distance”，并添加 VI 参数以指定距离，速度等。如果单位不匹配，则指令 VI

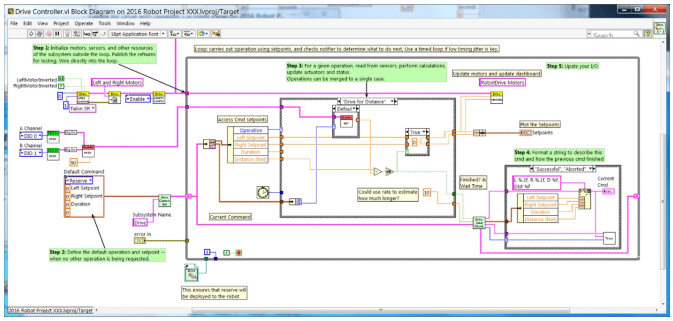
是在单位之间进行映射的好方法。



接下来，将代码添加到“驱动器控制器”中，以定义执行“Drive for Distance”指令时发生的情况。右键单击案例结构，然后为每个值复制或添加案例。这将创建一个新的“Drive for Distance”案例。

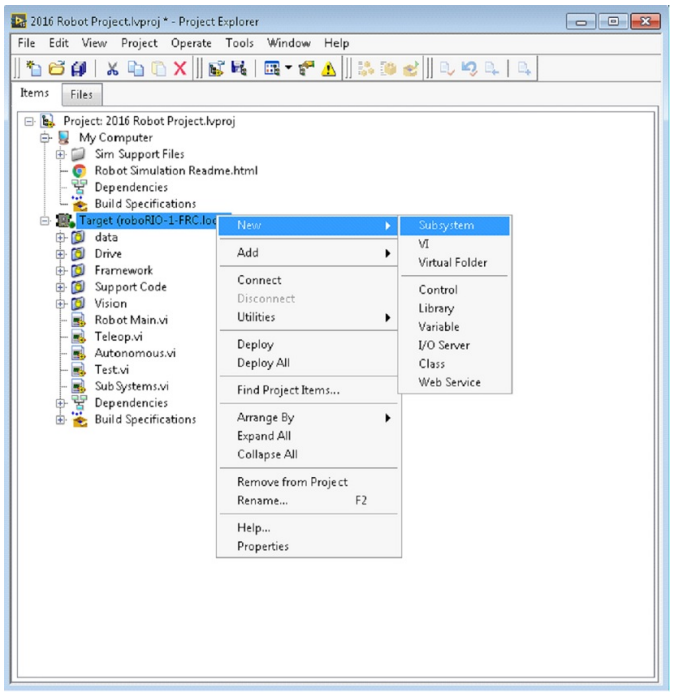


为了访问新的设置点字段，请为“Access Cmd 设置点”取消捆绑。在循环左侧的外部打开编码器。在新的案例结构图中，我们添加了一个调用以在第一次循环迭代时重置编码器，否则将其读取。还有一些简单的代码可以比较编码器值并更新电动机功率。如果将新控件添加到设定点集合中，则还应考虑将它们添加到跟踪中。下图显示了必要的更改。

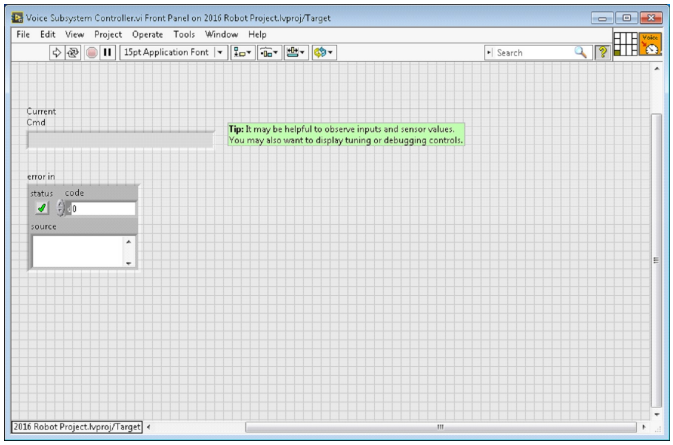


第 5 部分：创建子系统

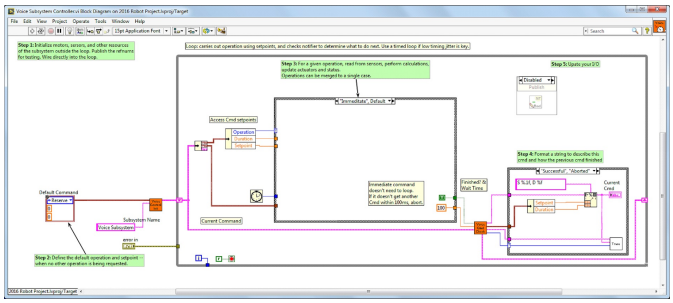
为了创建一个新的子系统，右键单击 roboRIO 目标并选择 New»Subsystem。在弹出的对话框中，输入子系统的名称，列出操作模式，并指定图标的颜色。



单击确定时，将生成子系统文件夹并将其添加到项目磁盘文件夹和项目树中。它将包含构成子系统的 VI 和控件的基本执行操作。对新控制器的调用将插入子系统 VI。控制器 VI 将打开，准备添加 I / O 并实现状态机或控制代码。生成的 VI 图标将使用对话框中提供的颜色和名称。生成的代码将 typedef 用于设置点字样和操作。



下面是新创建的子系统的程序框图。创建子系统时，将自动生成此代码。



13.2 LabVIEW 资源

13.2.1 LabVIEW 资源

备注： 要了解有关在 LabVIEW 中进行编程的更多信息，以及在 LabVIEW 中对 FRC® 机器进行特定编程的更多信息，请查看以下资源。

LabVIEW 基础

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.

NI FRC 教程

NI 还提供了许多 'FRC 特有的教程和演示，从基础到高级。要获得深入的某一方面的资源，请查看页面底部附近的 FRC 基础和高级培训课程。

已安装的教程和示例

LabVIEW 安装包还提供了有关各种任务和组件的教程和示例。要访问教程，请在 LabVIEW Splash 屏幕（程序首次启动时出现的屏幕）上，单击 "Tutorials" 左侧的选项。请注意，这些教程全都在一个文档中，因此一旦打开一个，您就可以自由浏览其他教程，无需返回初始屏幕。

要访问示例，请单击 "Support" 选项，然后在使用程序或进行操作时打开菜单，选择并打开文件夹 "Find FRC ExamplesHelpFind ExamplesFRC Robotics"

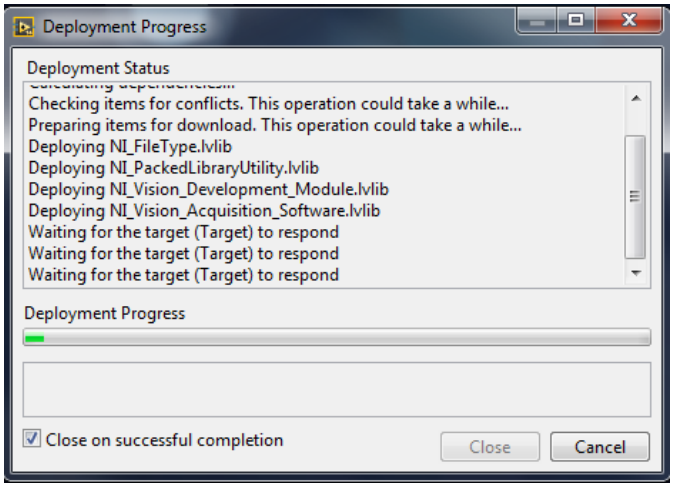
Third Party Resources

- FRC Control and Trajectory Library
- Secret Book Of FRC LabVIEW 2

13.2.2 等待目标响应-从错误循环中恢复

备注： 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.

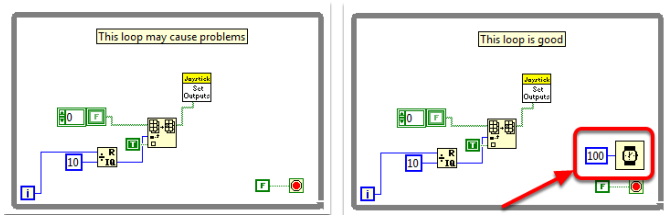
错误特征



此问题出现时的主要特征是在“等待目标响应”一步时尝试上传新的机器人代码，如上图所示。请注意，还有其他可能的原因导致这种现象（例如，从 C ++ 、Java 程序切换到 LabVIEW 程序），但是此处描述的步骤应该可以解决大多数或全部问题。

单击“取消”以关闭上传对话框。

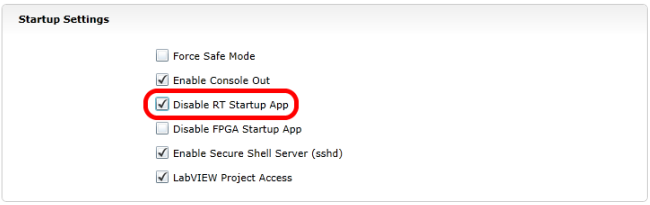
问题



这个问题的一个常见原因是 LabVIEW 代码中存在无限制循环。无限制循环是不包含任何延迟元素（例如左侧的延迟元素）的循环。如果不确定从哪里开始找，Disabled.VI，Periodic Tasks.VI 和 Vision

Processing.VI 是此类循环的常见位置。要解决代码问题，请在右侧循环中添加一个延迟元素，如“定时”选板中的“等待（ms）”VI。

设置禁用程序

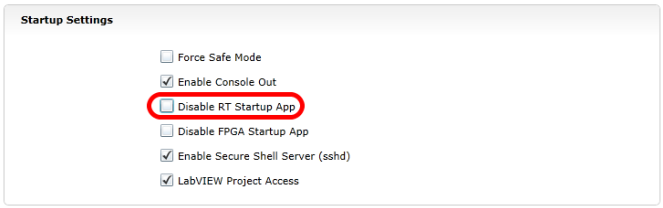


使用 roboRIO Web 服务器（有关更多详细信息，请参见“roboRIO Web Dashboard Startup Settings”上的文章）。选中“禁用 RT 启动应用程序”。

重启

使用设备上的“重置”按钮或单击网页右上角的“重新启动”以重新启动 roboRIO。

取消禁用程序



使用 roboRIO Web 服务器 (有关更多详细信息，请参见:ref:*roboRIO Web Dashboard Startup Settings <docs/software/roborio-info/roborio-web-dashboard:Startup Settings>* 上的文章)．取消选中“禁用 RT 启动应用程序”。

加载 LabVIEW 代码

加载 LabVIEW 代码（使用“运行”按钮或“以启动方式运行”）。在重启 roboRIO 之前，请确保将 LabVIEW 代码设置为“以启动方式运行”，否则您将需要再次按照上述说明进行操作。

13.2.3 如何切换两种相机模式

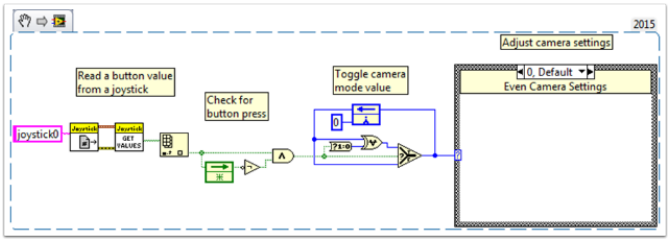
此代码展示了如何使用按钮在两种不同的相机模式之间切换。该代码包括四个部分。

在第一部分，读取手柄上按键的值。

接下来，使用一个 ** 反馈节点 ** 和一些布尔算法将当前读数与先前读数进行比较。这些共同确保了仅在最初按下按钮的一瞬间才切换相机模式，而不是在按住按钮时来回切换多次。

之后，第二阶段的结果将用来掩盖掉当前相机模式的值，以切换相机模式。这称为位屏蔽，通过使用 **XOR** 函数，代码将在第二阶段返回 true 时切换相机模式，否则不执行任何操作。

最后，您可以在条件结构的最后插入每种相机模式的代码。每次运行代码时，这一部分将运行当前相机模式的代码。



13.2.4 LabVIEW 实例和教程

热门教程

自动阶段定时运动教程

- 根据不同的时间间隔自动移动机器人
- **‘查看有关自动阶段运动的更多信息 <https://forums.ni.com/t5/FIRST-Robotics-Competition/Autonomous-Timed-Movement-Tutorial/ta-p/3732667?profile.language=en>‘__**

‘基本电动机控制教程 <https://forums.ni.com/t5/FIRST-Robotics-Competition/FRC-2014-Basic-Motor-Control-Tutorial/ta-p/3504064?profile.language=en>‘__

- 设置您的 roboRIO 电机硬件和软件
- 了解如何设置 FRC | reg |控制系统和 FRC 机器人项目
- **‘查看有关电动机控制的更多信息 <https://forums.ni.com/t5/FIRST-Robotics-Competition/Basic-Motor-Control-Tutorial/ta-p/3733426?profile.language=en>‘__**

‘图像处理教程 <https://forums.ni.com/t5/FIRST-Robotics-Competition/FRC-2015-Image-Processing-Tutorial/ta-p/3490518?profile.language=en>‘__

- 了解基本的图像处理技术以及如何使用 NI Vision Assistant
- **‘查看有关相机和图像处理的更多信息 <https://forums.ni.com/t5/FIRST-Robotics-Competition/Image-Processing-in-LabVIEW-for-FRC/ta-p/3732677?profile.language=en>‘__**

‘PID 控制指南 <https://forums.ni.com/t5/FIRST-Robotics-Competition/FRC-2015-PID-Tutorial/ta-p/3535805?profile.language=en>‘__

- 什么是 PID 控制，我该如何实施?

‘指令和控制指南 <https://forums.ni.com/t5/FIRST-Robotics-Competition/Command-and-Control-Tutorial/ta-p/3534946?profile.language=en>‘__

- 什么是指令与控制?
- 我该如何实施?

‘Driver Station 教程 <https://forums.ni.com/t5/FIRST-Robotics-Competition/Archived-FRC-2015-Driver-Station-Tutorial/ta-p/3535650?profile.language=en>‘__

- 了解 FRC Driver Station

‘测试模式教程 <https://forums.ni.com/t5/FIRST-Robotics-Competition/FRC-2015-Test-Mode-Tutorial/ta-p/3535797?profile.language=en>‘__

- 学习设置和使用测试模式

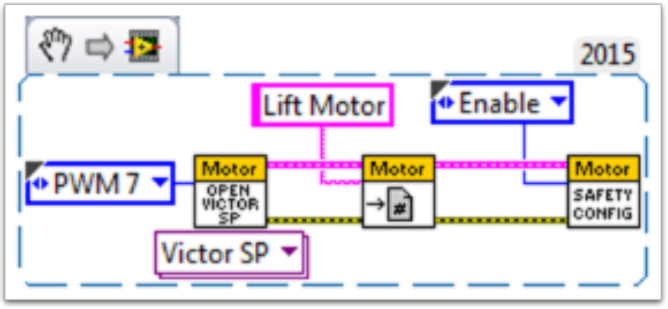
寻找更多示例和讨论？单击此处搜索更多文档或发布您自己的讨论，示例代码或教程！<https://forums.ni.com/t5/FIRST-Robotics-Competition/tkb-p/3019?profile.language=en>‘__’别忘了用标签标记您的帖子！

13.2.5 向项目中添加独立的电机

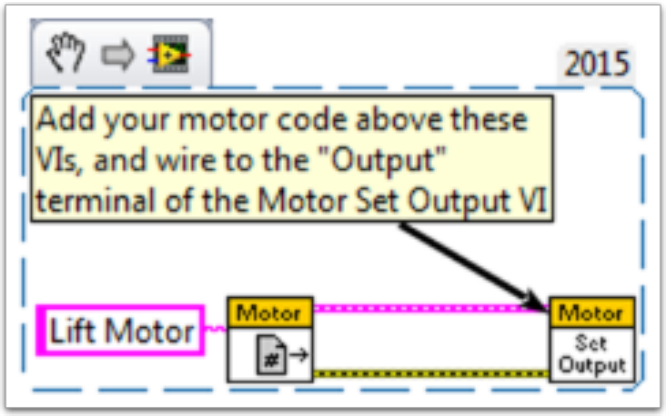
一旦你设置好了控制车轮的驱动器，您可能需要添加一个额外的电动机来控制独立于车轮的部件，例如机械臂。由于此马达不会成为坦克，拱形轮或麦克纳姆轮发动机的一部分，因此您绝对希望对其进行独立控制。

这些 VI 片段展示了如何在可能已经包含多个电机的项目中设置单个电机。如果看到 HAND> ARROW> LABVIEW 符号，只需将图像拖到框图中，瞧，代码！好的，操作方法如下。

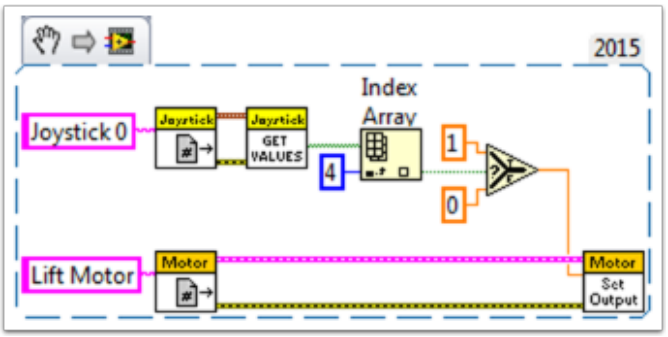
FIRST, create a motor reference in the **Begin.vi**, using the **Motor Control Open VI** and **Motor Control Refnum Registry Set VI**. These can be found by right-clicking in the block diagram and going to **WPI Robotics Library»RobotDrive»Motor Control**. Choose your *PWM* line and name your motor. 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.)



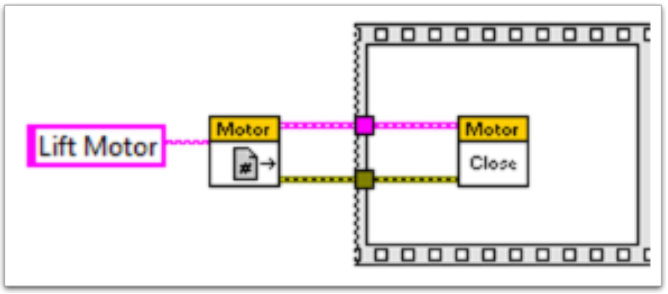
现在，使用 “Motor Control Refnum Registry Get” VI 在 Teleop.vi 中引用您的电机（名称必须准确），并告诉它用 Motor Control Set Output VI 做什么。它们与上述 VI 位于同一位置。



例如，如果按下操纵杆 0 上的按钮 4，则下一个代码片段指示提升马达向前移动，否则保持不动。对我而言，按钮 4 是 Xbox 样式控制器（“游戏杆 0”）上的左 bumper。有关更深入的操纵杆按钮选项的信息，请查看如何使用操纵杆按钮控制电机或电磁阀.



最后，我们需要使用“Motor Control Refnum Registry Get” VI 和“Motor Control Close” VI 关闭 Finish.vi 中的引用（就像对驱动器和操纵杆一样）。该图在一个平面序列结构内单独显示了 Close VI，但我们确实希望所有 Close VI 都在同一帧中。您可以将这两个 VI 放置在其他 Get VI 和 Close VI 下方（用于操纵杆和驱动器）。

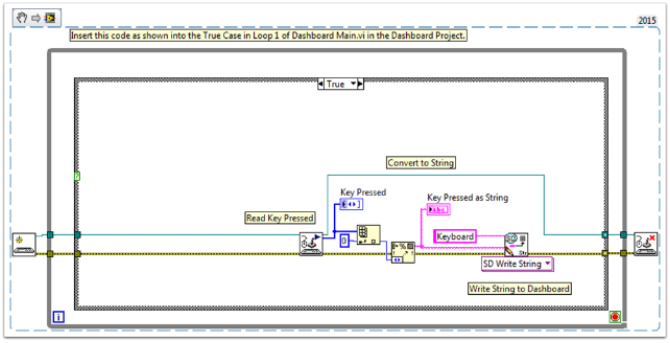


希望这可以帮助您编写最好的机器程序！祝你好运！

13.2.6 对 roboRIO 使用键盘导航

该示例为使用键盘导航代替操纵杆或其他控制器来控制机器人提供了一些建议。在这种情况下，我们使用 A，W，S 和 D 键来控制油箱驱动配置中的两个驱动马达。

第一个 VI 片段中的代码需要被包含在操作面板的主 VI。您可以将此代码插入到“循环 1”的 True 情况下。该代码在循环开始之前打开了与键盘的连接，并且在每次迭代中它都读取按下的键。该信息将转换为字符串，然后传递至机器人项目中的 Teleop VI。当循环 1 停止运行时，与键盘的连接将关闭。

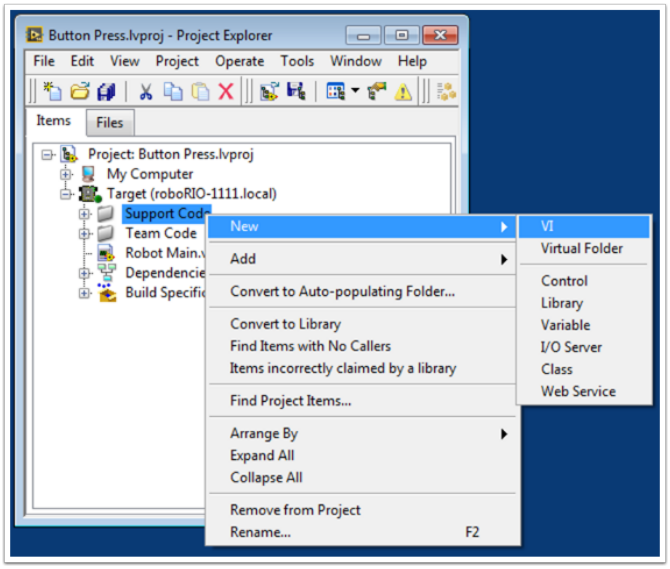


第二个 VI 片段代码应包含在遥控 VI 中。此片段将从仪表板读取被按下键位的字符串值。然后，一个“案例结构”根据按下的键确定应将哪些值写入左右电机。如图中样例所示，W 为正向，A 为左，D 为右，S 为反向；每种情况均以半速运行电机。您可以选择原封不动，更改特定值或添加其他代码以允许驱动程序调整速度，以根据需要快速或慢速行驶。电机值被输入后，它们将被写入驱动电机，并且被发布到仪表板上。

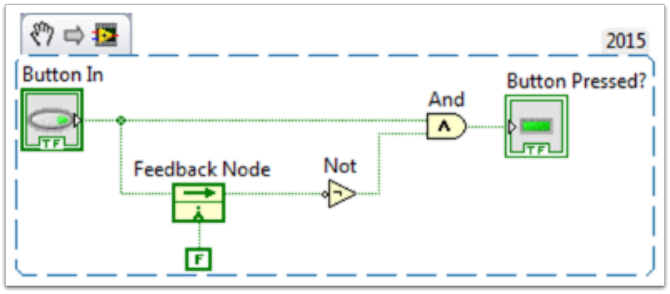
13.2.7 单响应按键

读取操纵杆数值时，按下操纵杆按钮将导致该按钮读数为 TRUE，直到松开该按钮为止。这意味着您极有可能在每次按下按键时读取多个 TRUE 值。如果在每次按下按钮时，你仅想读取一个 TRUE 值怎么办？通常将其称为“单响应按钮”。以下教程将向您展示如何创建一个子 VI，您可以将其放入 Teleop.vi。

首先，在项目的 Support Code 文件夹中创建一个新的 VI。



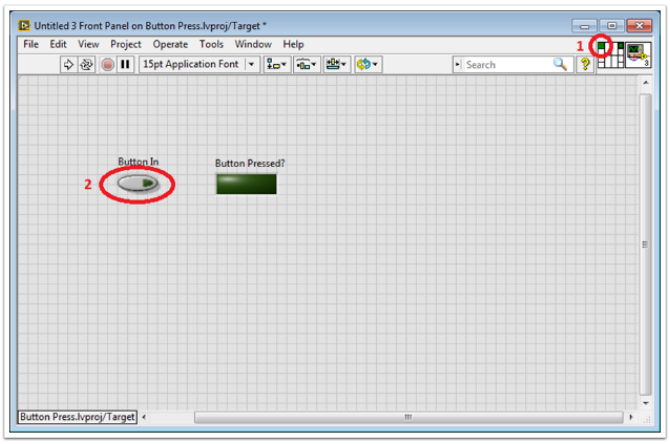
现在在新 VI 的程序框图上，添加以下代码段。



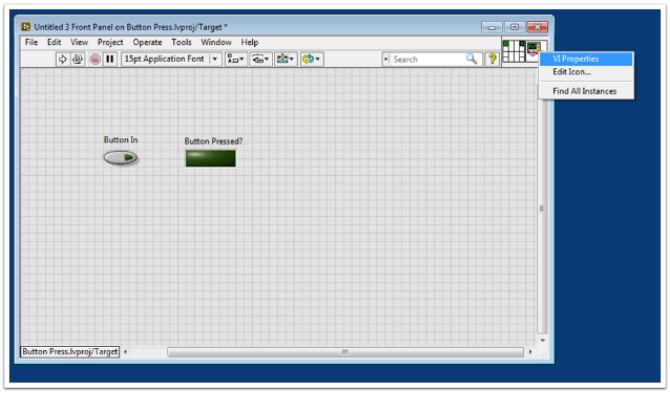
此代码使用了一个被称为反馈节点的功能。我们已经将按钮的当前值连接到了反馈节点的左侧。从反馈节点的箭头中出来的导线表示按钮的先前值。如果您的反馈节点上的箭头正朝着相反的方向（如此处所示），请单击右键以找到反向的选项。

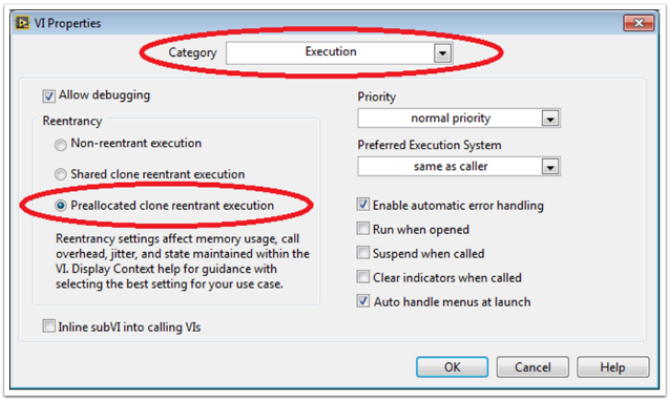
当按下按钮时，按钮的值从 FALSE 变为 TRUE。我们希望仅当按钮的当前值为 TRUE 且按钮的先前值为 FALSE 时，该 VI 的输出才为 TRUE。

接下来，我们需要将布尔控件和显示控件连接到 VI 的输入和输出。为此，请首先单击连接器窗格上的方格，然后单击按钮以将两者连接（请参见下图）。对显示控件重复此操作。

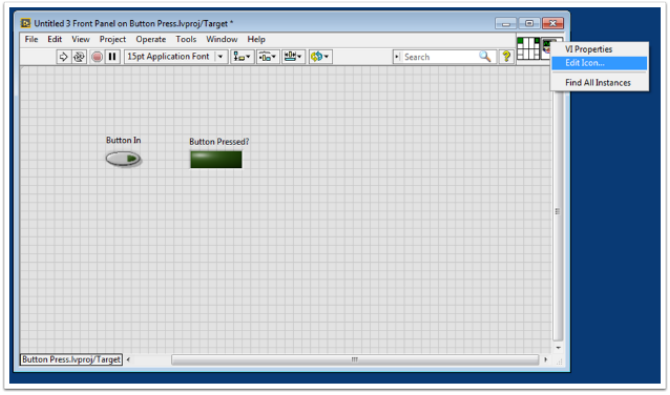


接下来，我们需要更改该 VI 的属性，以便我们可以在 TeleOp.vi 中使用这个 VI 的倍数。右键单击 VI 图标，然后转到 VI 属性。然后选择类别“执行”，然后选择“预分配的副本重入执行”。



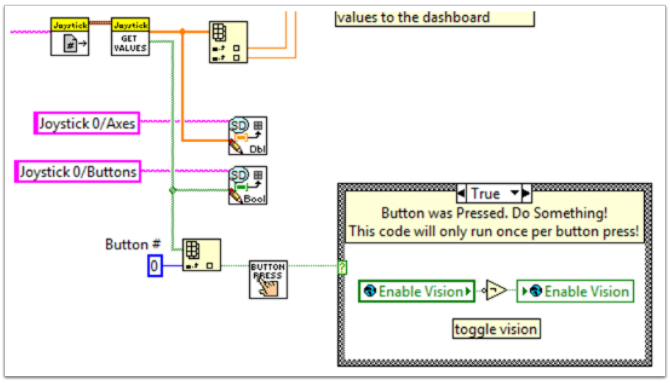


最后，我们应该更改 VI 图标，以更贴近 VI 的功能。右键单击图标，然后转到编辑图标。创建一个新的图标。



最后，命名并保存该 VI。现在，您可以将该 VI 从 Support Files 文件夹拖放到 TeleOp.vi 中。这是完成的 VI 的副本：`Button_Press.vi`

这是如何使用该 VI 的示例。



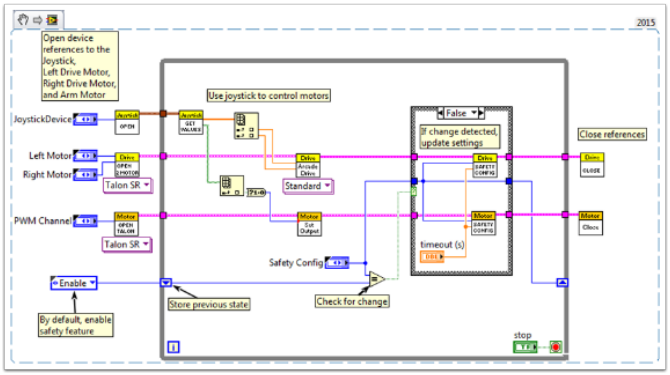
13.2.8 在机器人代码中添加安全功能

复杂项目的常见问题是确保所有代码都能按预期执行。当优先级高，执行时间长或频繁调用 roboRIO 的处理能力任务持续运行时，可能会出现问题。这导致由于处理器繁忙而无法执行其他任务。在大多数情况下，这只会减慢处理器对来自操纵杆和其他设备的输入的反应时间。但是，这也可能导致机器人的驱动电机在下达停止命令后仍保持运转很长时间。为了避免一切灾祸，您可以实施安全功能，以检查命令输入是否缺乏处理，并自动关闭可能有害的操作。

电机具有内置功能，可轻松执行安全检查。这些功能是：

- 机器驱动器安全配置
- 电机驱动器安全配置
- 继电器安全配置
- *PWM* Safety Configuration
- 电磁阀安全配置
- 机器驱动延迟和更新安全

在所有安全配置功能中，您可以在程序运行时启用和禁用这些安全检查，并配置您认为合适的最大时间限制。这些功能会管理一个所有启用了安全性的设备的缓存，并会检查其中是否有超过其时间限制的设备。如果有，则将禁用缓存中的所有设备，并且机器将立即停止或关闭其继电器/ PWM /电磁阀输出。下面的代码演示了如何使用“驱动器安全配置”功能来设置最大时间限制，即电动机在关闭之前所能接受的最长无输入时长。



要测试自关闭这一个安全机制，请尝试在循环中添加比您的最大时间限制更长的一个等待函数!

与实施安全检查有关的最终功能-机器人驱动延迟和更新安全性-使您可以将 roboRIO 置于自动阶段而不会超过前文设置的时间限制。它可以保持当前的电动机输出，而无需频繁调用驱动输出功能，而还可以确保定期进行安全检查，以防止电动机突然停止运行。

总的来说，强烈建议您在项目中进行某项安全检查，以确保您的机器人不会意外地处于危险状态!

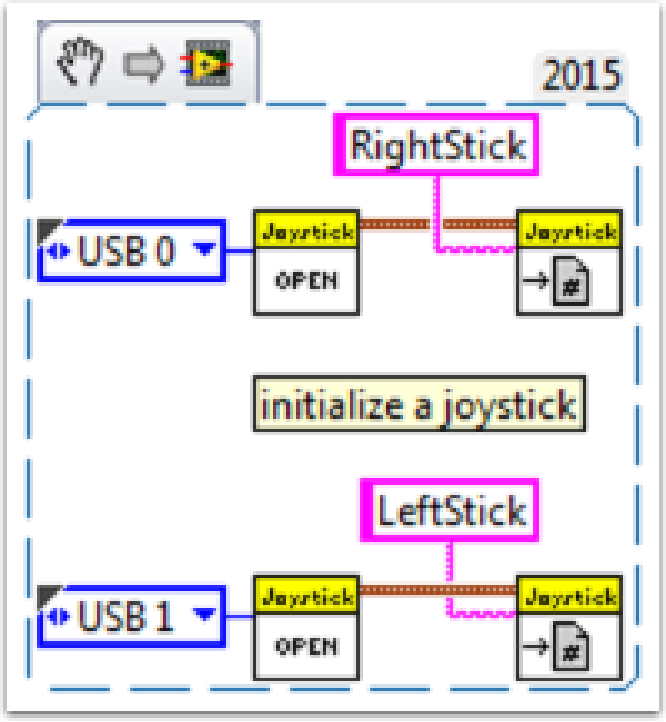
13.2.9 如何使用操纵杆按钮控制电机或电磁阀

随着驱动系统的正常工作，我们可以开始连接辅助设备，例如电动机和电磁阀。对此，我们通常将使用操纵杆按钮来控制这些设备。首先，我们将介绍几种通过操纵杆按钮控制设备的方法。

您是否知道可以从这样的文档中单击 VI 片段并将其拖放到您的 LabVIEW 代码中？用这个文档的 VI 片段先试试看吧

初始设置:

无论配置如何，您都需要在“Begin.vi”中添加一个，两个或更多（如果您真的很兴奋）手柄。第一个示例使 2 个手柄，而其他示例仅使用一个。给每个手柄起一个唯一的名称，这样我们就可以在其他地方使用它，例如下面的代码段。我将它们命名为“LeftStick”和“RightStick”，因为它们位于我桌子的左右两侧。如果你的手柄已经配置好了，那太好了，你可以跳过此步骤。

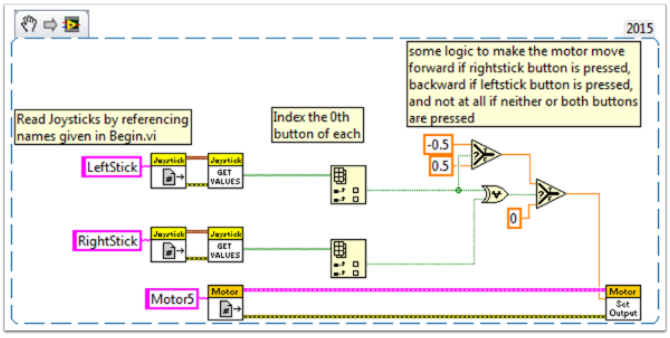


本文档中的其余代码将置于 Teleop.VI 中。在此处，我们将对操纵杆按钮进行编程，以控制电动机或电磁阀的不同功能。

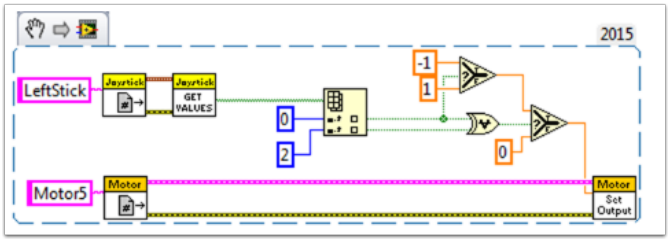
场景 1

“当我按下一个按钮时，我希望电机以一种方式移动，而当按下另一按钮时，我希望电机以另一种方式移动。”

该代码使用两个不同手柄上的按钮 0 来控制同一电机。如果按下 LeftStick 上的按钮 0，则电机向后移动，如果按下 RightStick 上的按钮 0，则电机向前移动。如果同时按下两个按钮或两个按钮均未按下，则电机不会移动。在这里，我将电机参考命名为“Motor5”，但是您可以在“Begin.vi”中随意命名电机。



您可能需要使用同一手柄上的多个按钮进行控制。有关此示例，请查看以下 VI 片段或场景 2 中的 VI 片段。



在这里，我使用了操纵杆按钮 0 和 2，但您可以随意使用所需的任何按钮。

场景 2

“我希望使用不同的手柄按钮令机器以不同的速度移动。”

如果您需要让一个电机根据所按的按钮执行不同的操作，则此示例可能会有所帮助。例如，假设我的手柄有一个扳机（按钮 0）和顶部的 4 个按钮（按钮 1 至 4）。在这种情况下，以下按钮应具有以下功能：

- 按钮 1-以一半速度向后移动
- 按钮 2-以半速前进
- 按钮 3-以 1/4 速度向后移动
- 按钮 4-以 1/4 速度向前移动
- 扳机-全速前进！（以满速向前移动）

然后，我们将从 JoystickGetValues.vi 中获取布尔数组，并将其连接到“布尔数组到数字”节点（数值选板转换选板）。这会将布尔数组转换为我们可以使用的数字。将此数字连接到条件结构。

每种情况都对应于数组中值的二进制表示形式。在此示例中，每种情况都对应一个按钮组合。我们添加了六种情况：0（关闭所有按钮），1（打开按钮 0），2（打开按钮 1），4（打开按钮 2），8（打开按钮 3）和 16（打开按钮 4）。请注意，我们跳过了值 3。3 将对应于同时按下的按钮 0 和 1。我们没有在需求中定义它，所以我们以默认情况处理。

在以下位置查看 LabVIEW 2014 条件结构帮助文档可能会有所帮助：

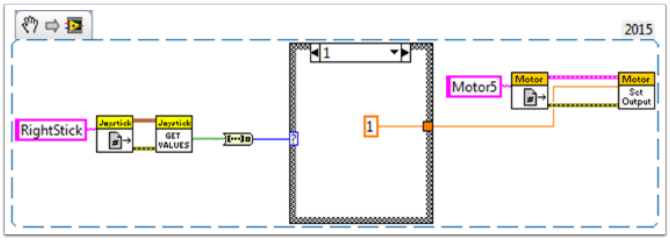
https://zone.ni.com/reference/en-XX/help/371361L-01/glang/case_structure/

这里还有关于条件结构的 3 个社区教程：

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



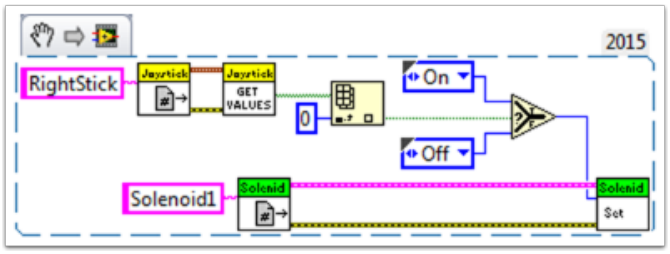
由于我们的要求很简单，因此在每种情况下我们只需要一个常数。我们可以使用 1 到-1 之间的任何常量值。对于情况 1（全力向前），我们使用 1，对于情况 2（半速向后），我们使用-0.5，依此类推。我将情况 0 保

留为默认值，因此如果按下多个按钮（达到任何未定义状态），电动机将停止。当然，您可以随意自定义这些状态。

场景 3

“我想用手柄按钮控制电磁阀。”

到目前为止，我们已经熟悉手柄如何以布尔数组的形式输出按钮。我们需要索引该数组以获得我们感兴趣的按钮数值，并将此布尔值连接到一个选择节点。由于 Solenoid Set.vi 需要一个枚举类作为输入，因此获取枚举的最简单方法是右键单击 Solenoid Set.vi 的 “Value” 输入，然后选择 “创建常量”。复制此常量，然后将一个副本连接到这个选择节点的 True 端口，将一个副本连接到选择节点的 False 端口。然后将该选择节点的输出连接到电磁阀 VI 的 “Value” 输入。



祝您编程愉快！

13.2.10 LabVIEW 中用于 FRC 的局部变量和全局变量

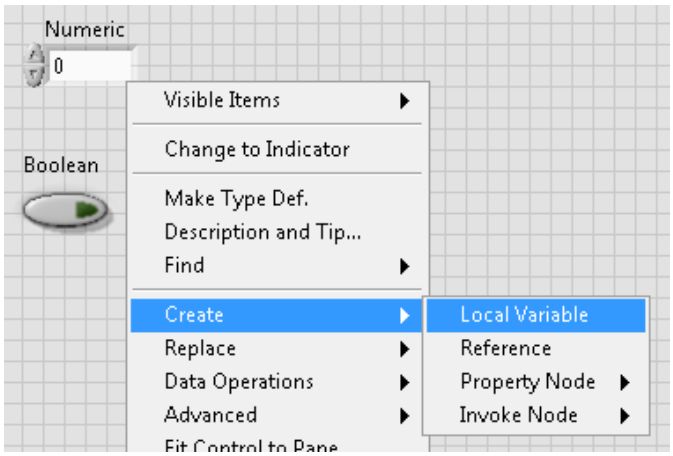
本示例介绍了局部变量和全局变量、如何在默认 LabVIEW for FRC® Robot Project 中使用它们以及如何在您的项目中使用它们。

局部变量和全局变量可用于在同一 VI（局部变量）或不同 VI（全局变量）内的位置之间传输数据，这打破了 LabVIEW 众所周知的传统的 “数据流范例 <https://www.ni.com/getting-started/labview-basics/dataflow>”。因此，当由于某种原因而无法将值直接连接到另一个节点时，它们可能很有用。

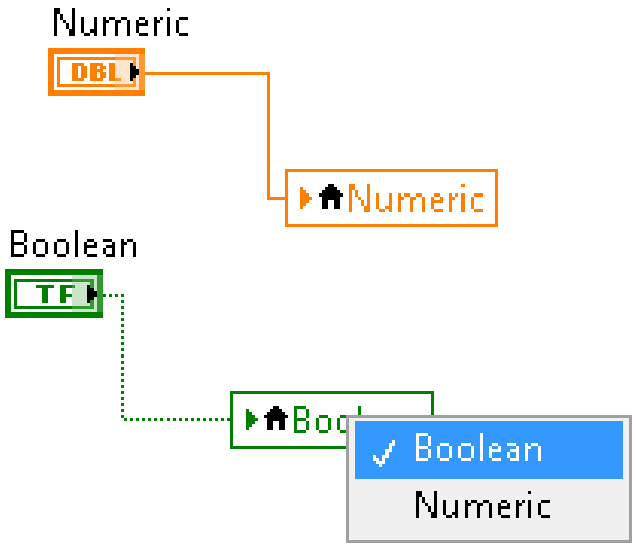
注意：一种可能的原因可能是您需要在连续的循环迭代之间传递数据。Miro_T 在这篇文章 <https://forums.ni.com/t5/FIRST-Robotics-Competition/Use-of-Shift-Registers-to-Pass-Data-Between-Loop-Iterations/ta-p/3498415?profile.language=en> 涵盖了它。还应该注意的是，LabVIEW 中的 “反馈节点 <https://zone.ni.com/reference/en-XX/help/371361L-01/lvconcepts/block_diagram_feedback/>” 可以等效于移位寄存器，尽管这可能是另一回事了！

局部和全局变量简介

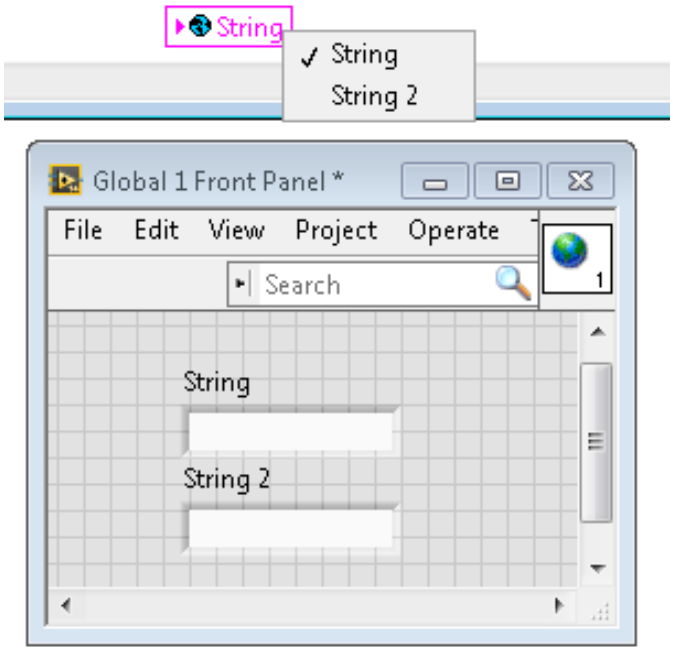
局部变量可在同一 VI 中使用。通过右键单击前面板上的输入控件或显示控件来创建局部变量：



您也可以从程序框图的“结构”选项板中创建局部变量。当一个 VI 中有多个局部变量时，可以单击鼠标左键选择它是哪个变量：



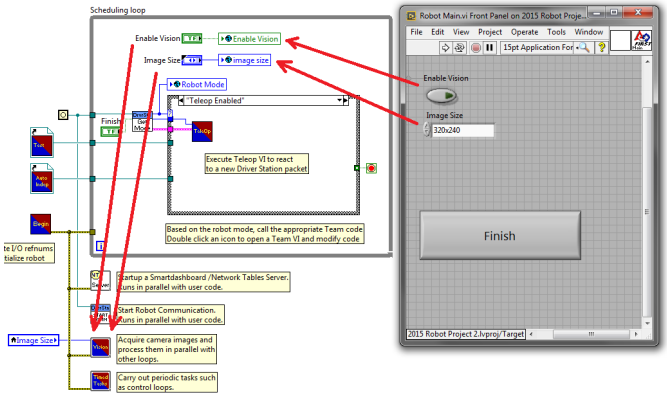
全局变量的创建略有不同。从“结构”选项板中向框图添加一个，然后注意，双击它会打开一个单独的前面板。该前面板没有程序框图，但是您可以根据需要向该前面板添加尽可能多的控件，并将其另存为 * .vi 文件：



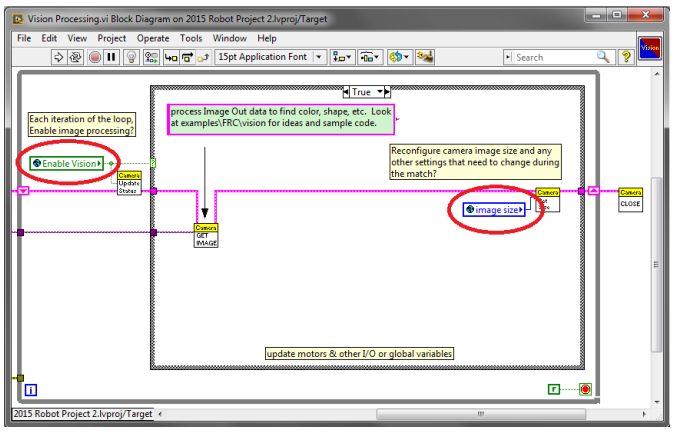
备注： 使用局部和全局变量时要非常小心，避免出现相互干扰的变量！本质上，请确保不要在不知道最后将变量写入哪个位置的情况下，在多个位置意外写入同一变量。有关更详尽的说明，请参阅‘此帮助文档 <https://zone.ni.com/reference/en-XX/help/371361L-01/lvconcepts/using_local_and_global/>‘

在默认的 LabVIEW for FRC 机器人项目中如何使用它们

在 Robot Main VI 的每次迭代期间，都会写入“Enable Vision”和“Image Size”两个全局变量。



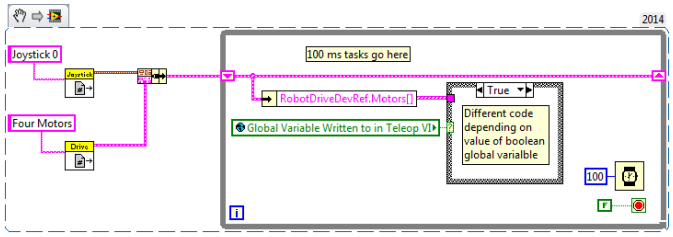
…然后在 Vision Processing VI 的每次迭代中进行读取：



这使用户可以在将程序从 LabVIEW 开发环境部署到 Robot Main VI 时，从 Robot Main 的前面板启用/禁用视觉并更改图像大小。

您如何在项目中使用它们？

请查看 Periodic Tasks VI 的框图。可能存在一些值，例如布尔值，可以将其写入 Teleop VI 中的全局变量，然后从 Periodical Tasks VI 中读取。然后，可以根据布尔型全局变量决定在 Periodic Tasks VI 中使用哪些代码或值：



13.2.11 在 LabVIEW 中使用压缩机

此代码段显示了如何设置 roboRIO 项目以使用气动控制模块（PCM）。当测量到罐中的压力超过特定阀值时，PCM 自动启动并停止压缩机。在 roboRIO 程序中，将需要添加以下 VI。

有关更多信息，请查看以下链接：

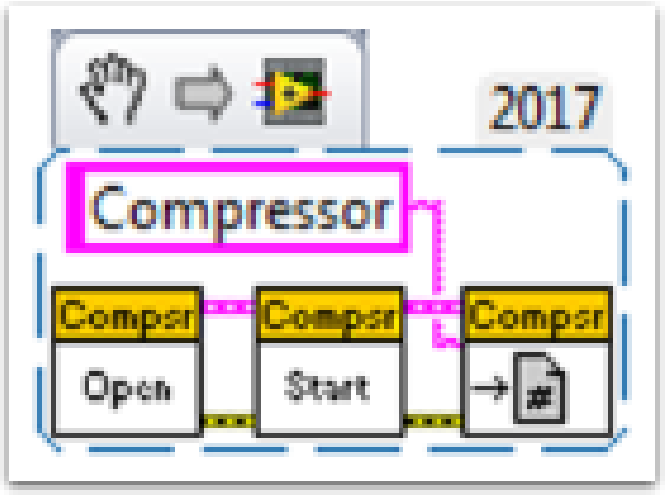
FRC Pneumatics Manual

PCM User’s Guide

‘roboRIO 的气动步骤 <http://team358.org/files/pneumatic/Pneumatics-StepByStep-roboRIO.pdf>‘__

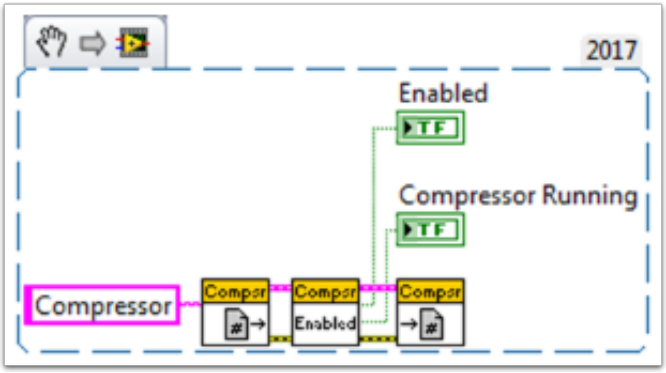
Begin VI

将此片段放在 Begin.vi 中。



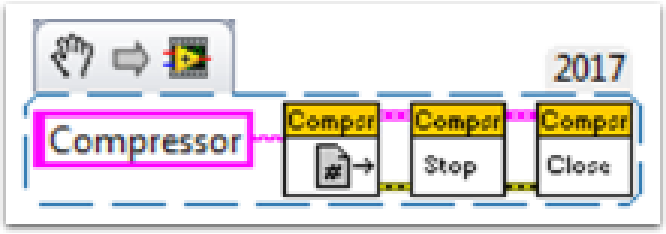
Teleop VI

将此片段放在 Teleop.vi 中。仅当您将输出用于其他过程时，才需要此部分。



Finish VI

将此片段放置在 Finish.vi 的结束引用、数据储存等框架中。



14 FRC Python Programming

14.1 pyproject.toml usage

备注： 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"
```

(续下页)

(接上页)

```
    # "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
```

备注: 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

备注: 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

警告： 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.

参见:

查看控制台输出

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
```

备注： 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

This section discusses the control of motors and pneumatics through motor controllers, solenoids and pneumatics, and their interface with Java and 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 Using Motor Controllers in Code

Motor controllers come in two main flavors: *CAN* and *PWM*. A CAN controller can send more detailed status information back to the roboRIO, whereas a PWM controller can only be set to a value. For information on using these motors with the WPILib drivetrain classes, see *Using the WPILib Classes to Drive your Robot*.

Using PWM Motor Controllers

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 *PWM Motor Controllers in Depth*. 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.

备注: 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
```

CAN Motor Controllers

A handful of CAN motor controllers are available through vendors such as CTR Electronics, REV Robotics, and Playing with Fusion. See *第三方 CAN 设备*, *第三方库*, and *Third Party Example Projects* for more information.

15.1.2 PWM Motor Controllers in Depth

提示: WPILib has extensive support for motor control. There are a number of classes that represent different types of motor controllers and servos. There are currently two classes of motor controllers, PWM based motor controllers and CAN based motor controllers. WPILib also contains composite classes (like DifferentialDrive) which allow you to control multiple motors with a single object. This article will cover the details of PWM motor controllers; CAN controllers and composite classes will be covered in separate articles.

PWM Controllers, brief theory of operation

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).

The PWM signal the controllers use for an input is a little bit different. Even at the bounds of the signal range (max forward or max reverse) the signal never approaches a duty cycle of 0% or 100%. Instead the controllers use a signal with a period of either 5ms or 10ms and a midpoint pulse width of 1.5ms. Many of the controllers use the typical hobby RC controller timing of 1ms to 2ms.

Raw vs Scaled output values

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.

Calibrating Motor Controllers

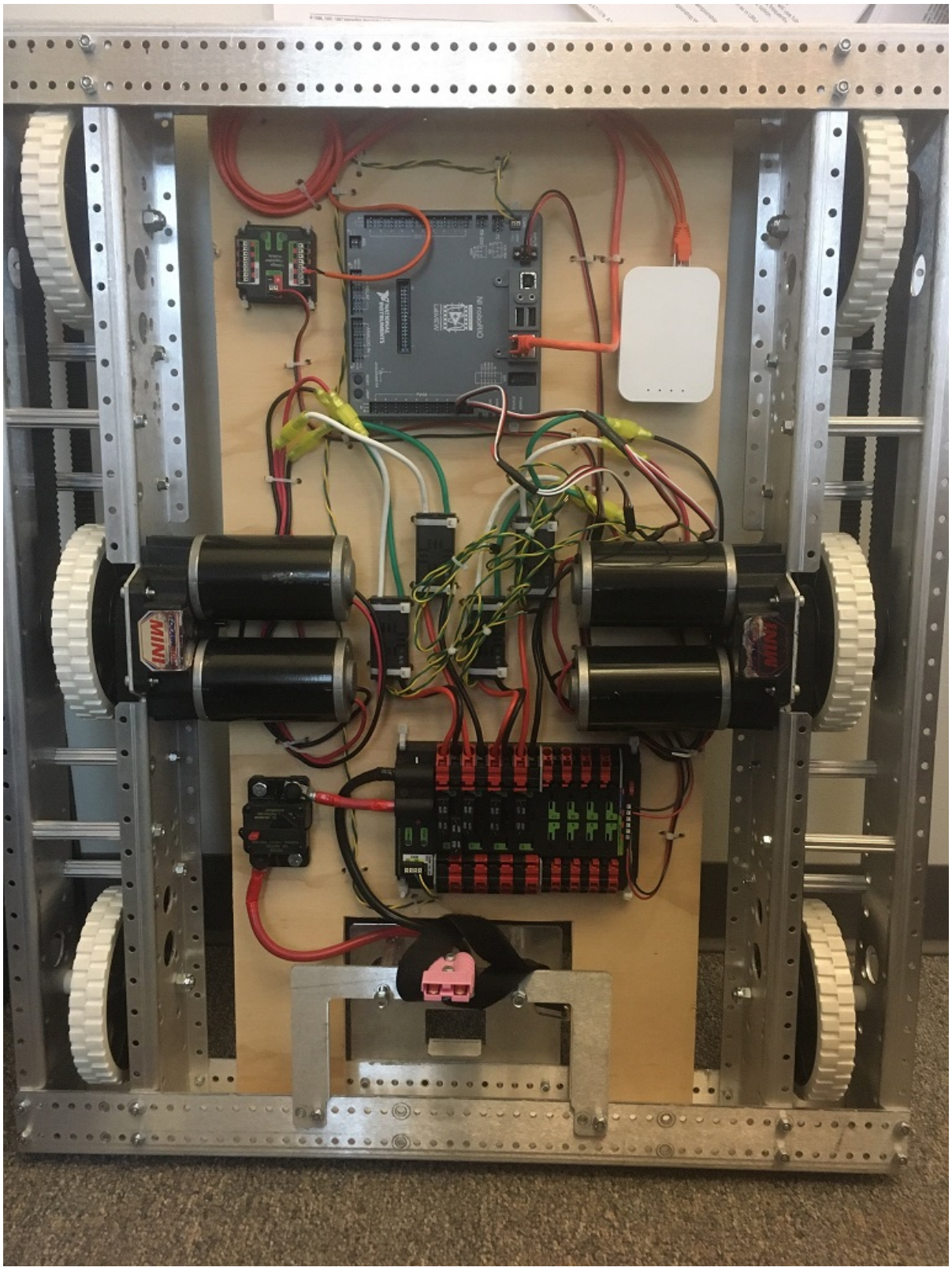
So if WPILib handles all this scaling, why would you ever need to calibrate your motor controller? The values WPILib uses for scaling are approximate based on measurement of a number of samples of each controller type. Due to a variety of factors, the timing of an individual motor controller may vary slightly. In order to definitively eliminate "humming" (midpoint signal interpreted as slight movement in one direction) and drive the controller all the way to each extreme, calibrating the controllers is still recommended. In general, the calibration procedure for each controller involves putting the controller into calibration mode then driving the input signal to each extreme, then back to the midpoint. For examples on how to use these motor controllers in your code, see *Using Motor Controllers in Code/Using PWM Motor Controllers*

15.1.3 Using the WPILib Classes to Drive your Robot

WPILib includes many classes to help make your robot get driving faster.

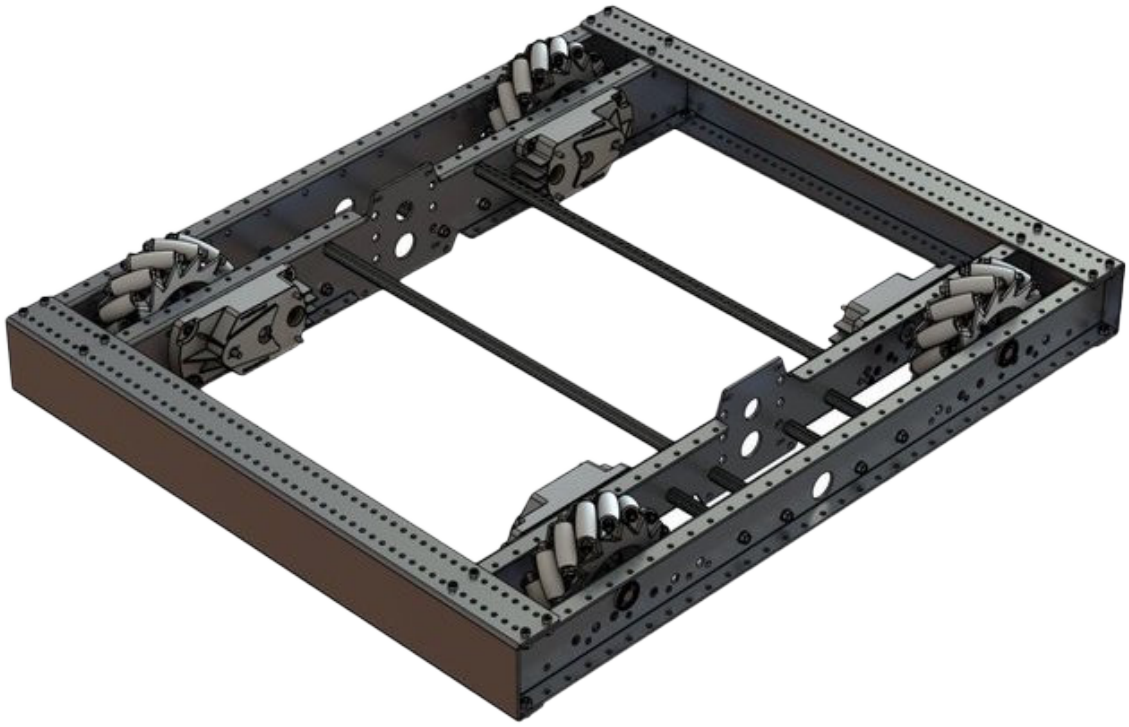
Standard drivetrains

Differential Drive Robots



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". The Kit of Parts drivetrain is an example of a differential drive. These drivetrains are capable of driving forward/backward and can turn by driving the two sides in opposite directions causing the wheels to skid sideways. These drivetrains are not capable of sideways translational movement.

Mecanum Drive



Mecanum drive is a method of driving using specially designed wheels that allow the robot to drive in any direction without changing the orientation of the robot. A robot with a conventional drivetrain (all wheels pointing in the same direction) must turn in the direction it needs to drive. A mecanum robot can move in any direction without first turning and is called a holonomic drive. The wheels (shown on this robot) have rollers that cause the forces from driving to be applied at a 45 degree angle rather than straight forward as in the case of a conventional drive.

When viewed from the top, the rollers on a mecanum drivetrain should form an 'X' pattern. This results in the force vectors (when driving the wheel forward) on the front two wheels pointing forward and inward and the rear two wheels pointing forward and outward. By spinning the wheels in different directions, various components of the force vectors cancel out, resulting in the desired robot movement. A quick chart of different movements has been provided below, drawing out the force vectors for each of these motions may help in understanding how these drivetrains work. By varying the speeds of the wheels in addition to the direction, movements can be combined resulting in translation in any direction and rotation, simultaneously.

Drive Class Conventions

Motor Inversion

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)
```

Squaring Inputs

When driving robots, it is often desirable to manipulate the joystick inputs such that the robot has finer control at low speeds while still using the full output range. One way to accomplish this is by squaring the joystick input, then reapplying the sign. By default the Differential Drive class will square the inputs. If this is not desired (e.g. if passing values in from a PIDController), use one of the drive methods with the squaredInputs parameter and set it to false.

Input Deadband

By default, the Differential Drive class applies an input deadband of 0.02. This means that input values with a magnitude below 0.02 (after any squaring as described above) will be set to 0. In most cases these small inputs result from imperfect joystick centering and are not sufficient to cause drivetrain movement, the deadband helps reduce unnecessary motor heating that may result from applying these small values to the drivetrain. To change the deadband, use the *setDeadband()* method.

Maximum Output

Sometimes drivers feel that their drivetrain is driving too fast and want to limit the output. This can be accomplished with the *setMaxOutput()* method. This maximum output is multiplied by result of the previous drive functions like deadband and squared inputs.

Motor Safety

Motor Safety is a mechanism in WPILib that takes the concept of a watchdog and breaks it out into one watchdog (Motor Safety timer) for each individual actuator. Note that this protection mechanism is in addition to the System Watchdog which is controlled by the Network Communications code and the FPGA and will disable all actuator outputs if it does not receive a valid data packet for 125ms.

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.

The Motor Safety feature operates by maintaining a timer that tracks how long it has been since the feed() method has been called for that actuator. Code in the Driver Station class initiates a comparison of these timers to the timeout values for any actuator with safety enabled every 5 received packets (100ms nominal). The set() methods of each motor controller class and the set() and setAngle() methods of the servo class call feed() to indicate that the output of the actuator has been updated.

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.

Axis Conventions

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.

备注: See the *Coordinate System* section for more detail about the axis conventions and coordinate systems.

Using the DifferentialDrive class to control Differential Drive robots

---

备注: 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 is a method provided for the control of "skid-steer" or "West Coast" drivetrains, such as the Kit of Parts chassis. Instantiating a DifferentialDrive is as simple as so:

Java

```
public class Robot extends TimedRobot {
  private DifferentialDrive m_robotDrive;
  private final PWMSparkMax m_leftMotor = new PWMSparkMax(0);
  private final PWMSparkMax m_rightMotor = new PWMSparkMax(1);

  @Override
  public void robotInit() {
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side␣
↪instead.
    m_rightMotor.setInverted(true);

    m_robotDrive = new DifferentialDrive(m_leftMotor::set, m_
↪rightMotor::set);
  }
```

C++ (Header)

```
  frc::PWMSparkMax m_leftMotor{0};
  frc::PWMSparkMax m_rightMotor{1};
  frc::DifferentialDrive m_robotDrive{
      [&](double output) { m_leftMotor.Set(output); },
      [&](double output) { m_rightMotor.Set(output); }};
```

C++ (Source)

```
  void RobotInit() override {
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side␣
↪instead.
    m_rightMotor.SetInverted(true);
  }
```

Python

```
    def robotInit(self):
        """Robot initialization function"""

        leftMotor = wpilib.PWMSparkMax(0)
        rightMotor = wpilib.PWMSparkMax(1)
        self.robotDrive = wpilib.drive.DifferentialDrive(leftMotor,␣
↪rightMotor)
        # We need to invert one side of the drivetrain so that positive␣
↪voltages
        # result in both sides moving forward. Depending on how your robot's
        # gearbox is constructed, you might have to invert the left side␣
↪instead.
        rightMotor.setInverted(True)
```

Multi-Motor DifferentialDrive

Many FRC® drivetrains have more than 1 motor on each side. Classes derived from `PWMMotorController` (Java / C++ / Python) have an `addFollower` method so that multiple follower motor controllers can be updated when the leader motor controller is commanded. CAN motor controllers have similar features, review the vendor's documention to see how to use them. The examples below show a 4 motor (2 per side) drivetrain. To extend to more motors, simply create the additional controllers and use additional `addFollower` calls.

Java

Class variables (e.g. in Robot.java or Subystem):

```
  // The motors on the left side of the drive.
  private final PWMSparkMax m_leftLeader = new PWMSparkMax(DriveConstants.
↪kLeftMotor1Port);
  private final PWMSparkMax m_leftFollower = new PWMSparkMax(DriveConstants.
↪kLeftMotor2Port);

  // The motors on the right side of the drive.
  private final PWMSparkMax m_rightLeader = new PWMSparkMax(DriveConstants.
↪kRightMotor1Port);
  private final PWMSparkMax m_rightFollower = new PWMSparkMax(DriveConstants.
↪kRightMotor2Port);
```

In robotInit or Subsystem constructor:

```
    m_leftLeader.addFollower(m_leftFollower);
    m_rightLeader.addFollower(m_rightFollower);

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

C++ (Header)

```
 private:
  // The motor controllers
  frc::PWMSparkMax m_left1;
  frc::PWMSparkMax m_left2;
  frc::PWMSparkMax m_right1;
  frc::PWMSparkMax m_right2;

  // The robot's drive
  frc::DifferentialDrive m_drive{[&](double output) { m_left1.Set(output); },
                                 [&](double output) { m_right1.Set(output); }
↪};
```

C++ (Source)

In robotInit or Subsystem constructor:

```
  m_left1.AddFollower(m_left2);
  m_right1.AddFollower(m_right2);

  // We need to invert one side of the drivetrain so that positive voltages
  // result in both sides moving forward. Depending on how your robot's
  // gearbox is constructed, you might have to invert the left side instead.
  m_right1.SetInverted(true);
```

Python

备注： 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)
```

(续下页)

(接上页)

```
    right = wpilib.MotorControllerGroup(frontLeft, rearLeft)

    self.drive = wpilib.drive.DifferentialDrive(left, right)
```

Drive Modes

备注： The DifferentialDrive class contains three different default modes of driving your robot's motors.

- Tank Drive, which controls the left and right side independently
- Arcade Drive, which controls a forward and turn speed
- Curvature Drive, a subset of Arcade Drive, which makes your robot handle like a car with constant-curvature turns.

The DifferentialDrive class contains three default methods for controlling skid-steer or WCD robots. Note that you can create your own methods of controlling the robot's driving and have them call tankDrive() with the derived inputs for left and right motors.

The Tank Drive mode is used to control each side of the drivetrain independently (usually with an individual joystick axis controlling each). This example shows how to use the Y-axis of two separate joysticks to run the drivetrain in Tank mode. Construction of the objects has been omitted, for above for drivetrain construction and here for Joystick construction.

The Arcade Drive mode is used to control the drivetrain using speed/throttle and rotation rate. This is typically used either with two axes from a single joystick, or split across joysticks (often on a single gamepad) with the throttle coming from one stick and the rotation from another. This example shows how to use a single joystick with the Arcade mode. Construction of the objects has been omitted, for above for drivetrain construction and here for Joystick construction.

Like Arcade Drive, the Curvature Drive mode is used to control the drivetrain using speed/throttle and rotation rate. The difference is that the rotation control input controls the radius of curvature instead of rate of heading change, much like the steering wheel of a car. This mode also supports turning in place, which is enabled when the third `boolean` parameter is true.

JAVA

```
public void teleopPeriodic() {
    // Tank drive with a given left and right rates
    myDrive.tankDrive(-leftStick.getY(), -rightStick.getY());

    // Arcade drive with a given forward and turn rate
    myDrive.arcadeDrive(-driveStick.getY(), -driveStick.getX());

    // Curvature drive with a given forward and turn rate, as well as a button for␣
↪turning in-place.
    myDrive.curvatureDrive(-driveStick.getY(), -driveStick.getX(), driveStick.
↪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))
```

Using the MecanumDrive class to control Mecanum Drive robots

MecanumDrive is a method provided for the control of holonomic drivetrains with Mecanum wheels, such as the Kit of Parts chassis with the mecanum drive upgrade kit, as shown above. Instantiating a MecanumDrive is as simple as so:

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);
```

(续下页)

(接上页)

```
    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
```

(续下页)

(接上页)

```
        )

        self.stick = wpilib.Joystick(self.kJoystickChannel)
```

Mecanum Drive Modes

备注: The drive axis conventions are different from common joystick axis conventions. See the *Axis Conventions* above for more information.

The MecanumDrive class contains two different default modes of driving your robot's motors.

- driveCartesian: Angles are measured clockwise from the positive X axis. The robot's speed is independent from its angle or rotation rate.
- drivePolar: Angles are measured counter-clockwise from straight ahead. The speed at which the robot drives (translation) is independent from its angle or rotation rate.

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)
```

Field-Oriented Driving

A 4th parameter can be supplied to the `driveCartesian(double ySpeed, double xSpeed, double zRotation, double gyroAngle)` method, the angle returned from a Gyro sensor. This will adjust the rotation value supplied. This is particularly useful with mecanum drive since, for the purposes of steering, the robot really has no front, back or sides. It can go in any direction. Adding the angle in degrees from a gyro object will cause the robot to move away from the drivers when the joystick is pushed forwards, and towards the drivers when it is pulled towards them, regardless of what direction the robot is facing.

The use of field-oriented driving makes often makes the robot much easier to drive, especially compared to a "robot-oriented" drive system where the controls are reversed when the robot is facing the drivers.

Just remember to get the gyro angle each time `driveCartesian()` is called.

备注: Many teams also like to ramp the joysticks inputs over time to promote a smooth acceleration and reduce jerk. This can be accomplished with a *Slew Rate Limiter*.

15.1.4 Repeatable Low Power Movement - Controlling Servos with 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)

Constructing a Servo object

JAVA

```
Servo exampleServo = new Servo(1);
```

C++

```
frc::Servo exampleServo {1};
```

PYTHON

```
exampleServo = wpilib.Servo(1)
```

A servo object is constructed by passing a channel.

Setting Servo Values

JAVA

```
exampleServo.set(.5);
exampleServo.setAngle(75);
```

C++

```
exampleServo.Set(.5);
exampleServo.SetAngle(75);
```

PYTHON

```
exampleServo.set(.5)
exampleServo.setAngle(75)
```

There are two methods of setting servo values in WPILib:

- Scaled Value - Sets the servo position using a scaled 0 to 1.0 value. 0 corresponds to one extreme of the servo and 1.0 corresponds to the other
- 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 Pneumatics 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

```
30  // Solenoid corresponds to a single solenoid.
31  // In this case, it's connected to channel 0 of a PH with the default CAN ID.
32  private final Solenoid m_solenoid = new Solenoid(PneumaticsModuleType.REVPH, 0);
```

```
88    /*
89     * The output of GetRawButton is true/false depending on whether
90     * the button is pressed; Set takes a boolean for whether
91     * to retract the solenoid (false) or extend it (true).
92     */
93    m_solenoid.set(m_stick.getRawButton(kSolenoidButton));
```

C++ (Header)

```
44  // Solenoid corresponds to a single solenoid.
45  // In this case, it's connected to channel 0 of a PH with the default CAN
46  // ID.
47  frc::Solenoid m_solenoid{frc::PneumaticsModuleType::REVPH, 0};
```

C++ (Source)

```
42  /*
43   * The output of GetRawButton is true/false depending on whether
44   * the button is pressed; Set takes a boolean for whether
45   * to retract the solenoid (false) or extend it (true).
46   */
47  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

```
37   // DoubleSolenoid corresponds to a double solenoid.
38   // In this case, it's connected to channels 1 and 2 of a PH with the default CAN ID.
39   private final DoubleSolenoid m_doubleSolenoid =
40       new DoubleSolenoid(PneumaticsModuleType.REVPH, 1, 2);
```

```
100      m_doubleSolenoid.set(DoubleSolenoid.Value.kForward);
101      m_doubleSolenoid.set(DoubleSolenoid.Value.kReverse);
```

C++ (Header)

```
49   // DoubleSolenoid corresponds to a double solenoid.
50   // In this case, it's connected to channels 1 and 2 of a PH with the default
51   // CAN ID.
52   frc::DoubleSolenoid m_doubleSolenoid{frc::PneumaticsModuleType::REVPH, 1, 2};
```

C++ (Source)

```
54     m_doubleSolenoid.Set(frc::DoubleSolenoid::kForward);
55     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.

备注: 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++):

备注： 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 硬件组件概述.

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 Sensor Overview - Software

备注: This section covers using sensors in software. For a guide to sensor hardware, see 传感器概述-硬件.

备注: While cameras may definitely be considered "sensors", vision processing is a sufficiently-complicated subject that it is covered in *its own section*, rather than here.

In order to be effective, it is often vital for robots to be able to gather information about their surroundings. Devices that provide feedback to the robot on the state of its environment are called "sensors." WPILib innately supports a large variety of sensors through classes included in the library. This section will provide a guide to both using common sensor types through WPILib, as well as writing code for sensors without official support.

What sensors does WPILIB support?

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 provides native support for:

- *Accelerometers*
- *Gyroscopes*
- *Ultrasonic rangefinders*
- *Potentiometers*

- *Counters*
- *Quadrature encoders*
- *Limit switches*

Additionally, WPILib includes lower-level classes for interfacing directly with the FPGA's digital and analog inputs and outputs.

15.3.2 Accelerometers - Software

备注: This section covers accelerometers in software. For a hardware guide to accelerometers, see 加速度计 - 硬件.

An accelerometer is a device that measures acceleration.

Accelerometers generally come in two types: single-axis and 3-axis. A single-axis accelerometer measures acceleration along one spatial dimension; a 3-axis accelerometer measures acceleration along all three spatial dimensions at once.

WPILib supports single-axis accelerometers through the *AnalogAccelerometer* class.

Three-axis accelerometers often require more complicated communications protocols (such as SPI or I2C) in order to send multi-dimensional data. WPILib has native support for the following 3-axis accelerometers:

- *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();
```

If users have a 3-axis analog accelerometer, they can use three instances of this class, one for each axis.

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;
```

Third-party accelerometers

While WPILib provides native support for a number of accelerometers that are available in the kit of parts or through FIRST Choice, there are a few popular AHRS (Attitude and Heading Reference System) devices commonly used in FRC that include accelerometers. These are generally controlled through vendor libraries, though if they have a simple analog output they can be used with the *AnalogAccelerometer* class.

Using accelerometers in code

备注: Accelerometers, as their name suggests, measure acceleration. Precise accelerometers can be used to determine position through double-integration (since acceleration is the second derivative of position), much in the way that gyroscopes are used to determine heading. However, the accelerometers available for use in FRC are not nearly high-enough quality to be used this way.

It is recommended to use accelerometers in FRC® for any application which needs a rough measurement of the current acceleration. This can include detecting collisions with other robots or field elements, so that vulnerable mechanisms can be automatically retracted. They may also be used to determine when the robot is passing over rough terrain for an autonomous routine (such as traversing the defenses in FIRST Stronghold).

For detecting collisions, it is often more robust to measure the jerk than the acceleration. The jerk is the derivative (or rate of change) of acceleration, and indicates how rapidly the forces on the robot are changing - the sudden impulse from a collision causes a sharp spike in the jerk. Jerk can be determined by simply taking the difference of subsequent acceleration measurements, and dividing by the time between them:

JAVA

```java
double prevXAccel = 0.0;
double prevYAccel = 0.0;

BuiltInAccelerometer accelerometer = new BuiltInAccelerometer();

@Override
public void robotPeriodic() {
    // Gets the current accelerations in the X and Y directions
    double xAccel = accelerometer.getX();
    double yAccel = accelerometer.getY();

    // Calculates the jerk in the X and Y directions
    // Divides by .02 because default loop timing is 20ms
    double xJerk = (xAccel - prevXAccel) / 0.02;
    double yJerk = (yAccel - prevYAccel) / 0.02;

    prevXAccel = xAccel;
    prevYAccel = yAccel;
}
```

C++

```cpp
double prevXAccel = 0.0;
double prevYAccel = 0.0;

frc::BuiltInAccelerometer accelerometer;

void Robot::RobotPeriodic() {
    // Gets the current accelerations in the X and Y directions
    double xAccel = accelerometer.GetX();
    double yAccel = accelerometer.GetY();

    // Calculates the jerk in the X and Y directions
    // Divides by .02 because default loop timing is 20ms
    double xJerk = (xAccel - prevXAccel) / 0.02;
    double yJerk = (yAccel - prevYAccel) / 0.02;

    prevXAccel = xAccel;
    prevYAccel = yAccel;
}
```

Most accelerometers legal for FRC use are quite noisy, and it is often a good idea to combine them with the `LinearFilter` class (Java, C++) to reduce the noise:

JAVA

```
BuiltInAccelerometer accelerometer = new BuiltInAccelerometer();

// Create a LinearFilter that will calculate a moving average of the measured X␣
↪acceleration over the past 10 iterations of the main loop

LinearFilter xAccelFilter = LinearFilter.movingAverage(10);

@Override
public void robotPeriodic() {
    // Get the filtered X acceleration
    double filteredXAccel = xAccelFilter.calculate(accelerometer.getX());
}
```

C++

```
frc::BuiltInAccelerometer accelerometer;

// Create a LinearFilter that will calculate a moving average of the measured X␣
↪acceleration over the past 10 iterations of the main loop
auto xAccelFilter = frc::LinearFilter::MovingAverage(10);

void Robot::RobotPeriodic() {
    // Get the filtered X acceleration
    double filteredXAccel = xAccelFilter.Calculate(accelerometer.GetX());
}
```

15.3.3 Gyroscopes - Software

备注： This section covers gyros in software. For a hardware guide to gyros, see 陀螺仪-硬件.

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.

备注： It is crucial that the robot remain stationary while calibrating a gyro.

ADIS16448

The ADIS16448 uses the `ADIS16448_IMU` class (Java, C++, Python). See the Analog Devices ADIS16448 documentation for additional information and examples.

> **警告:** 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.

> **警告:** 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.

备注: ADXRS450 Gyro accumulation is handled through special circuitry in the FPGA; accordingly only a single instance of `ADXRS450_Gyro` may be used.

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.

备注: Gyro accumulation is handled through special circuitry in the FPGA; accordingly, `AnalogGyro`'s may only be used on analog ports 0 and 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
```

Using gyros in code

备注： As gyros measure rate rather than position, position is inferred by integrating (adding up) the rate signal to get the total change in angle. Thus, gyro angle measurements are always relative to some arbitrary zero angle (determined by the angle of the gyro when either the robot was turned on or a zeroing method was called), and are also subject to accumulated errors (called "drift") that increase in magnitude the longer the gyro is used. The amount of drift varies with the type of gyro.

Gyros are extremely useful in FRC for both measuring and controlling robot heading. Since FRC matches are generally short, total gyro drift over the course of an FRC match tends to be manageably small (on the order of a couple of degrees for a good-quality gyro). Moreover, not all useful gyro applications require the absolute heading measurement to remain accurate over the course of the entire match.

Displaying the robot heading on the 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)
```

Stabilizing heading while driving

A very common use for a gyro is to stabilize robot heading while driving, so that the robot drives straight. This is especially important for holonomic drives such as mecanum and swerve, but is extremely useful for tank drives as well.

This is typically achieved by closing a PID controller on either the turn rate or the heading, and piping the output of the loop to one's turning control (for a tank drive, this would be a speed differential between the two sides of the drive).

> **警告:** Like with all control loops, users should be careful to ensure that the sensor direction and the turning direction are consistent. If they are not, the loop will be unstable and the robot will turn wildly.

Example: Tank drive stabilization using turn rate

The following example shows how to stabilize heading using a simple P loop closed on the turn rate. Since a robot that is not turning should have a turn rate of zero, the setpoint for the loop is implicitly zero, making this method very 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)
```

(续下页)

(接上页)

```
    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)
```

备注： MotorControllerGroup is *deprecated* in 2024. Can you help update the Python example?

More-advanced implementations can use a more-complicated control loop. When closing the loop on the turn rate for heading stabilization, PI loops are particularly effective.

Example: Tank drive stabilization using heading

The following example shows how to stabilize heading using a simple P loop closed on the heading. Unlike in the turn rate example, we will need to set the setpoint to the current heading before starting motion, making this method slightly more-complicated.

JAVA

```
// Use gyro declaration from above here

// The gain for a simple P loop
double kP = 1;

// The heading of the robot when starting the motion
double heading;

// Initialize motor controllers and drive
Spark left1 = new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

MotorControllerGroup leftMotors = new MotorControllerGroup(left1, left2);
MotorControllerGroup rightMotors = new MotorControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);

@Override
public void robotInit() {
    rightMotors.setInverted(true);
}
```

(续下页)

(接上页)

```
@Override
public void autonomousInit() {
    // Set setpoint to current heading at start of auto
    heading = gyro.getAngle();
}

@Override
public void autonomousPeriodic() {
    double error = heading - gyro.getAngle();

    // Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    drive.tankDrive(.5 + kP * error, .5 - kP * error);
}
```

C++

```
// Use gyro declaration from above here

// The gain for a simple P loop
double kP = 1;

// The heading of the robot when starting the motion
double heading;

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};

frc::MotorControllerGroup leftMotors{left1, left2};
frc::MotorControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::RobotInit() {
  rightMotors.SetInverted(true);
}

void Robot::AutonomousInit() {
    // Set setpoint to current heading at start of auto
    heading = gyro.GetAngle();
}

void Robot::AutonomousPeriodic() {
    double error = heading - gyro.GetAngle();

    // Drives forward continuously at half speed, using the gyro to stabilize the␣
↪heading
    drive.TankDrive(.5 + kP * error, .5 - kP * error);
}
```

PYTHON

```
from wpilib import Spark
from wpilib import MotorControllerGroup
from wpilib.drive import DifferentialDrive

def robotInit(self):
    # Use gyro declaration from above here

    # The gain for a simple P loop
    self.kP = 1

    # Initialize motor controllers and drive
    left1 = Spark(0)
    left2 = Spark(1)
    right1 = Spark(2)
    right2 = Spark(3)

    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)
```

More-advanced implementations can use a more-complicated control loop. When closing the loop on the heading for heading stabilization, PD loops are particularly effective.

Turning to a set heading

Another common and highly-useful application for a gyro is turning a robot to face a specified direction. This can be a component of an autonomous driving routine, or can be used during teleoperated control to help align a robot with field elements.

Much like with heading stabilization, this is often accomplished with a PID loop - unlike with stabilization, however, the loop can only be closed on the heading. The following example code will turn the robot to face 90 degrees with a simple P loop:

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};
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);
}
```

(续下页)

(接上页)

```
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)
```

As before, more-advanced implementations can use more-complicated control loops.

备注： Turn-to-angle loops can be tricky to tune correctly due to static friction in the drivetrain, especially if a simple P loop is used. There are a number of ways to account for this; one of the most common/effective is to add a "minimum output" to the output of the control loop. Another effective strategy is to cascade to well-tuned velocity controllers on each side of the drive.

15.3.4 Ultrasonics - Software

备注: This section covers ultrasonics in software. For a hardware guide to ultrasonics, see 超声波测距仪-硬件.

An ultrasonic sensor is commonly used to measure distance to an object using high-frequency sound. Generally, ultrasonics measure the distance to the closest object within their "field of view."

There are two primary types of ultrasonics supported natively by WPILib:

- *Ping-response ultrasonics*
- *Analog ultrasonics*

Ping-response ultrasonics

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;
```

Analog ultrasonics

Some ultrasonic sensors simply return an analog voltage corresponding to the measured distance. These sensors can may simply be used with the *AnalogPotentiometer* class.

Third-party ultrasonics

Other ultrasonic sensors offered by third-parties may use more complicated communications protocols (such as I2C or SPI). WPILib does not provide native support for any such ultrasonics; they will typically be controlled with vendor libraries.

Using ultrasonics in code

Ultrasonic sensors are very useful for determining spacing during autonomous routines. For example, the following code from the UltrasonicPID example project (Java, C++) will move the robot to 1 meter away from the nearest object the sensor detects:

Java

```
public class Robot extends TimedRobot {
  // distance the robot wants to stay from an object
  // (one meter)
  static final double kHoldDistanceMillimeters = 1.0e3;

  // proportional speed constant
  private static final double kP = 0.001;
  // integral speed constant
  private static final double kI = 0.0;
  // derivative speed constant
  private static final double kD = 0.0;

  static final int kLeftMotorPort = 0;
  static final int kRightMotorPort = 1;

  static final int kUltrasonicPingPort = 0;
  static final int kUltrasonicEchoPort = 1;

  // Ultrasonic sensors tend to be quite noisy and susceptible to sudden␣
↪outliers,
  // so measurements are filtered with a 5-sample median filter
  private final MedianFilter m_filter = new MedianFilter(5);
```

(续下页)

(接上页)

```
  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;
  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
```

(续下页)

(接上页)

```
  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:

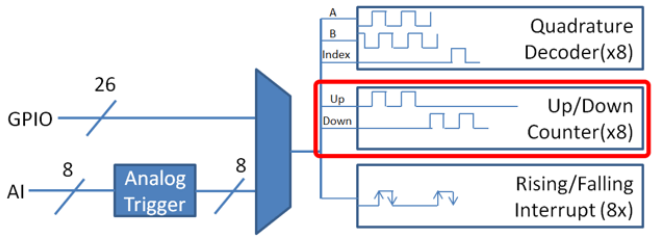
Java

```
    // 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.

备注: There are a total of 8 counter units in the roboRIO FPGA, meaning no more than 8 `Counter` objects may be instantiated at any one time, including those contained as resources in other WPILib objects. For detailed information on when a `Counter` may be used by another object, refer to the official API documentation.

Configuring a counter

The `Counter` class can be configured in a number of ways to provide differing functionalities.

Counter Modes

The `Counter` object may be configured to operate in one of four different modes:

1. *Two-pulse mode*: Counts up and down based on the edges of two different channels.
2. *Semi-period mode*: Measures the duration of a pulse on a single channel.
3. *Pulse-length mode*: Counts up and down based on the edges of one channel, with the direction determined by the duration of the pulse on that channel.
4. *External direction mode*: Counts up and down based on the edges of one channel, with a separate channel specifying the direction.

备注: 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.

Two-pulse mode

In two-pulse mode, the `Counter` will count up for every edge/pulse on the specified "up channel," and down for every edge/pulse on the specified "down channel." A counter can be initialized in two-pulse with the following code:

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);
```

Semi-period mode

In semi-period mode, the `Counter` will count the duration of the pulses on a channel, either from a rising edge to the next falling edge, or from a falling edge to the next rising edge. A counter can be initialized in semi-period mode with the following code:

JAVA

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.kSemiperiod);

@Override
public void robotInit() {
    // Set up the input channel for the counter
    counter.setUpSource(1);

    // Set the encoder to count pulse duration from rising edge to falling edge
    counter.setSemiPeriodMode(true);
}
```

C++

```
// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::kSemiperiod};

void Robot() {
    // Set up the input channel for the counter
    counter.SetUpSource(1);

    // Set the encoder to count pulse duration from rising edge to falling edge
    counter.SetSemiPeriodMode(true);
```

To get the pulse width, call the `getPeriod()` method:

JAVA

```
// Return the measured pulse width in seconds
counter.getPeriod();
```

C++

```
// Return the measured pulse width in seconds
counter.GetPeriod();
```

Pulse-length mode

In pulse-length mode, the counter will count either up or down depending on the length of the pulse. A pulse below the specified threshold time will be interpreted as a forward count and a pulse above the threshold is a reverse count. This is useful for some gear tooth sensors which encode direction in this manner. A counter can be initialized in this mode as follows:

JAVA

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.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)
```

External direction mode

In external direction mode, the counter counts either up or down depending on the level on the second channel. If the direction source is low, the counter will increase; if the direction source is high, the counter will decrease (to reverse this, see the next section). A counter can be initialized in this mode as follows:

JAVA

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.kExternalDirection);

@Override
public void robotInit() {
    // Set up the input channels for the counter
    counter.setUpSource(1);
    counter.setDownSource(2);

    // Set the decoding type to 2X
```

(续下页)

(接上页)

```
    counter.setUpSourceEdge(true, true);
}
```

C++

```
// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::kExternalDirection};

void RobotInit() {
    // Set up the input channels for the counter
    counter.SetUpSource(1);
    counter.SetDownSource(2);

    // Set the decoding type to 2X
    counter.SetUpSourceEdge(true, true);
```

Configuring counter parameters

备注： The `Counter` class does not make any assumptions about units of distance; it will return values in whatever units were used to calculate the distance-per-pulse value. Users thus have complete control over the distance units used. However, units of time are *always* in seconds.

备注： 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 (rising and falling).

Apart from the mode-specific configurations, the `Counter` class offers a number of additional configuration methods:

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);
```

Reading information from counters

Regardless of mode, there is some information that the `Counter` class always exposes to users:

Count

Users can obtain the current count with the `get()` method:

JAVA

```
// returns the current count
counter.get();
```

C++

```
// returns the current count
counter.Get();
```

Distance

备注: 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();
```

Rate

备注: Units of time for the `Counter` class are *always* in seconds.

Users can obtain the current rate of change of the counter with the `getRate()` method:

JAVA

```
// Gets the current rate of the counter
counter.getRate();
```

C++

```
// Gets the current rate of the counter
counter.GetRate();
```

Stopped

Users can obtain whether the counter is stationary with the `getStopped()` method:

JAVA

```
// Gets whether the counter is stopped
counter.getStopped();
```

C++

```
// Gets whether the counter is stopped
counter.GetStopped();
```

Direction

Users can obtain the direction in which the counter last moved with the `getDirection()` method:

JAVA

```
// Gets the last direction in which the counter moved
counter.getDirection();
```

C++

```
// Gets the last direction in which the counter moved
counter.GetDirection();
```

Period

备注: In *semi-period mode*, this method returns the duration of the pulse, not of the period.

Users can obtain the duration (in seconds) of the most-recent period with the `getPeriod()` method:

JAVA

```
// returns the current period in seconds
counter.getPeriod();
```

C++

```
// returns the current period in seconds
counter.GetPeriod();
```

Resetting a counter

To reset a counter 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.

JAVA

```
// Resets the encoder to read a distance of zero
counter.reset();
```

C++

```
// Resets the encoder to read a distance of zero
counter.Reset();
```

Using counters in code

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 *Encoders - Software*.

15.3.6 Encoders - Software

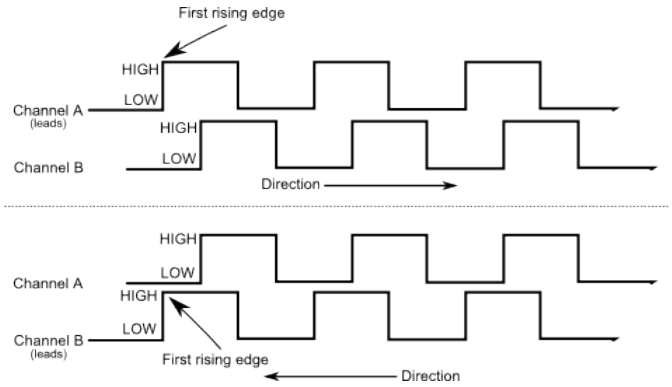
备注: This section covers encoders in software. For a hardware guide to encoders, see 编码器 - 硬件.

Encoders are devices used to measure motion (usually, the rotation of a shaft).

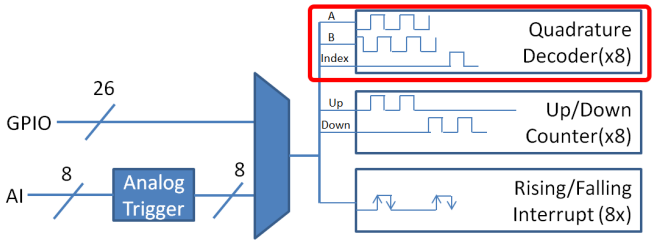
重要: The classes in this document are only used for encoders that are plugged directly into the roboRIO! Please reference the appropriate vendors' documentation for using encoders plugged into motor controllers.

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

The WPILib `Encoder` class can decode encoder signals in three different modes:

- **1X Decoding**: Increments the distance for every complete period of the encoder signal (once per four edges).
- **2X Decoding**: Increments the distance for every half-period of the encoder signal (once per two edges).
- **4X Decoding**: Increments the distance for every edge of the encoder signal (four times per period).

4X decoding offers the greatest precision, but at the potential cost of increased "jitter" in rate measurements. To use a different decoding type, use the following 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

备注： The `Encoder` class does not make any assumptions about units of distance; it will return values in whatever units were used to calculate the distance-per-pulse value. Users thus have complete control over the distance units used. However, units of time are *always* in seconds.

备注： 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).

The `Encoder` class offers a number of configuration methods:

JAVA

```
// Configures the encoder to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
encoder.setDistancePerPulse(4.0/256.0);

// Configures the encoder to consider itself stopped after .1 seconds
encoder.setMaxPeriod(0.1);

// Configures the encoder to consider itself stopped when its rate is below 10
encoder.setMinRate(10);

// Reverses the direction of the encoder
encoder.setReverseDirection(true);

// Configures an encoder to average its period measurement over 5 samples
// Can be between 1 and 127 samples
encoder.setSamplesToAverage(5);
```

C++

```
// Configures the encoder to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
encoder.SetDistancePerPulse(4.0/256.0);

// Configures the encoder to consider itself stopped after .1 seconds
encoder.SetMaxPeriod(0.1);

// Configures the encoder to consider itself stopped when its rate is below 10
encoder.SetMinRate(10);

// Reverses the direction of the encoder
encoder.SetReverseDirection(true);

// Configures an encoder to average its period measurement over 5 samples
// Can be between 1 and 127 samples
encoder.SetSamplesToAverage(5);
```

Reading information from Quadrature Encoders

The `Encoder` class provides a wealth of information to the user about the motion of the encoder.

Distance

备注： 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*.

Users can obtain the total distance traveled by the encoder with the `getDistance()` method:

JAVA

```
// Gets the distance traveled
encoder.getDistance();
```

C++

```
// Gets the distance traveled
encoder.GetDistance();
```

Rate

备注： Units of time for the `Encoder` class are *always* in seconds.

Users can obtain the current rate of change of the encoder with the `getRate()` method:

JAVA

```
// Gets the current rate of the encoder
encoder.getRate();
```

C++

```
// Gets the current rate of the encoder
encoder.GetRate();
```

Stopped

Users can obtain whether the encoder is stationary with the `getStopped()` method:

JAVA

```
// Gets whether the encoder is stopped
encoder.getStopped();
```

C++

```
// Gets whether the encoder is stopped
encoder.GetStopped();
```

Direction

Users can obtain the direction in which the encoder last moved with the `getDirection()` method:

JAVA

```
// Gets the last direction in which the encoder moved
encoder.getDirection();
```

C++

```
// Gets the last direction in which the encoder moved
encoder.GetDirection();
```

Period

Users can obtain the period of the encoder pulses (in seconds) with the `getPeriod()` method:

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

备注： 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

备注： 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

备注: 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

备注： 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*

备注: 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.);

    // 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); }};
```

(续下页)

(接上页)

```
void Robot::RobotInit() {
    // Configures the encoder's distance-per-pulse
    // The robot moves forward 1 foot per encoder rotation
    // There are 256 pulses per encoder rotation
    encoder.SetDistancePerPulse(1.0/256.0);

    // Invert the right side of the drivetrain. You might have to invert the other␣
↪side
    rightLeader.SetInverted(true);

    // Configure the followers to follow the leaders
    leftLeader.AddFollower(leftFollower);
    rightLeader.AddFollower(rightFollower);
}

void Robot::AutonomousPeriodic() {
    // Drives forward at half speed until the robot has moved 5 feet, then stops:
    if(encoder.GetDistance() < 5) {
        drive.TankDrive(0.5, 0.5);
    } else {
        drive.TankDrive(0, 0);
    }
}
```

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:

备注: 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);
```

(续下页)

(接上页)

```
        encoder.reset();
    }
}
```

C++

```
frc::Encoder encoder{0,1};

frc::Spark spark{0};

// Limit switch on DIO 2
frc::DigitalInput limit{2};

void AutonomousPeriodic() {
    // Runs the motor backwards at half speed until the limit switch is pressed
    // then turn off the motor and reset the encoder
    if(!limit.Get()) {
        spark.Set(-0.5);
    } else {
        spark.Set(0);
        encoder.Reset();
    }
}
```

15.3.7 Analog Inputs - Software

备注： This section covers analog inputs in software. For a hardware guide to analog inputs, see 模拟输入 - 硬件.

The roboRIO's FPGA supports up to 8 analog input channels that can be used to read the value of an analog voltage from a sensor. Analog inputs may be used for any sensor that outputs a simple voltage.

Analog inputs from the FPGA by default return a 12-bit integer proportional to the voltage, from 0 to 5 volts.

The AnalogInput class

备注： It is often more convenient to use the *Analog Potentiometers* wrapper class than to use `AnalogInput` directly, as it supports scaling to meaningful units.

Support for reading the voltages on the FPGA analog inputs is provided through the `AnalogInput` class (Java, C++).

Initializing an AnalogInput

An `AnalogInput` may be initialized as follows:

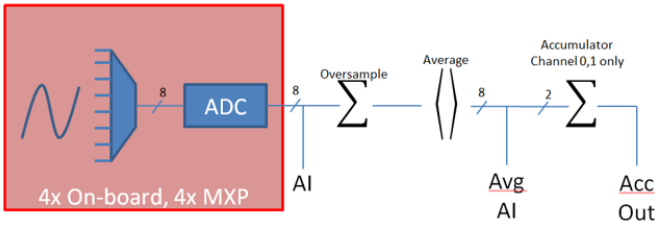
JAVA

```
// Initializes an AnalogInput on port 0
AnalogInput analog = new AnalogInput(0);
```

C++

```
// Initializes an AnalogInput on port 0
frc::AnalogInput analog{0};
```

Oversampling and Averaging



The FPGA's analog input modules supports both oversampling and averaging. These behaviors are highly similar, but differ in a few important ways. Both may be used at the same time.

Oversampling

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);
```

Averaging

Averaging behaves much like oversampling, except the accumulated values are divided by the number of samples so that the scaling of the returned values does not change. This is often more-convenient, but occasionally the additional roundoff error introduced by the rounding is undesirable.

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);
```

备注: When oversampling and averaging are used at the same time, the oversampling is applied *first,* and then the oversampled values are averaged. Thus, 2-bit oversampling and 2-bit averaging used at the same time will increase the scale of the returned values by approximately a factor of 2, and decrease the update rate by approximately a factor of 4.

Reading values from an AnalogInput

Values can be read from an AnalogInput with one of four different methods:

getValue

The `getValue` method returns the raw instantaneous measured value from the analog input, without applying any calibration and ignoring oversampling and averaging settings. The returned value is an integer.

JAVA

```
analog.getValue();
```

C++

```
analog.GetValue();
```

getVoltage

The `getVoltage` method returns the instantaneous measured voltage from the analog input. Oversampling and averaging settings are ignored, but the value is rescaled to represent a voltage. The returned value is a double.

JAVA

```
analog.getVoltage();
```

C++

```
analog.GetVoltage();
```

getAverageValue

The `getAverageValue` method returns the averaged value from the analog input. The value is not rescaled, but oversampling and averaging are both applied. The returned value is an integer.

JAVA

```
analog.getAverageValue();
```

C++

```
analog.GetAverageValue();
```

getAverageVoltage

The `getAverageVoltage` method returns the averaged voltage from the analog input. Rescaling, oversampling, and averaging are all applied. The returned value is a double.

JAVA

```
analog.getAverageVoltage();
```

C++

```
analog.GetAverageVoltage();
```

Accumulator

备注: The accumulator methods do not currently support returning a value in units of volts - the returned value will always be an integer (specifically, a `long`).

Analog input channels 0 and 1 additionally support an accumulator, which integrates (adds up) the signal indefinitely, so that the returned value is the sum of all past measured values. Oversampling and averaging are applied prior to accumulation.

JAVA

```
// Sets the initial value of the accumulator to 0
// This is the "starting point" from which the value will change over time
analog.setAccumulatorInitialValue(0);

// Sets the "center" of the accumulator to 0.  This value is subtracted from
// all measured values prior to accumulation.
analog.setAccumulatorCenter(0);

// Returns the number of accumulated samples since the accumulator was last started/
↪reset
analog.getAccumulatorCount();

// Returns the value of the accumulator.  Return type is long.
analog.getAccumulatorValue();

// Resets the accumulator to the initial value
analog.resetAccumulator();
```

C++

```
// Sets the initial value of the accumulator to 0
// This is the "starting point" from which the value will change over time
analog.SetAccumulatorInitialValue(0);

// Sets the "center" of the accumulator to 0.  This value is subtracted from
// all measured values prior to accumulation.
analog.SetAccumulatorCenter(0);

// Returns the number of accumulated samples since the accumulator was last started/
↪reset
analog.GetAccumulatorCount();

// Returns the value of the accumulator.  Return type is long.
analog.GetAccumulatorValue();

// Resets the accumulator to the initial value
analog.ResetAccumulator();
```

Obtaining synchronized count and value

Sometimes, it is necessarily to obtain matched measurements of the count and the value. This can be done using the `getAccumulatorOutput` method:

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);
```

Using analog inputs in code

The `AnalogInput` class can be used to write code for a wide variety of sensors (including potentiometers, accelerometers, gyroscopes, ultrasonics, and more) that return their data as an analog voltage. However, if possible it is almost always more convenient to use one of the other existing WPILib classes that handles the lower-level code (reading the analog voltages and converting them to meaningful units) for you. Users should only directly use `AnalogInput` as a "last resort."

Accordingly, for examples of how to effectively use analog sensors in code, users should refer to the other pages of this chapter that deal with more-specific classes.

15.3.8 Analog Potentiometers - Software

备注: This section covers analog potentiometers in software. For a hardware guide to analog potentiometers, see 模拟电位器 - 硬件.

Potentiometers are variable resistors that allow information about position to be converted into an analog voltage signal. This signal can be read by the roboRIO to control whatever device is attached to the potentiometer.

While it is possible to read information from a potentiometer directly with an *Analog Inputs - 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.

In fact, the `AnalogPotentiometer` name is something of a misnomer - this class should be used for the vast majority of sensors that return their signal as a simple, linearly-scaled analog voltage.

The AnalogPotentiometer class

备注: The "full range" or "scale" parameters in the `AnalogPotentiometer` constructor are scale factors from a range of 0-1 to the actual range, *not* from 0-5. That is, they represent a native fractional scale, rather than a voltage scale.

An AnalogPotentiometer can be initialized as follows:

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};
```

Customizing the underlying AnalogInput

备注: If the user changes the scaling of the `AnalogInput` with oversampling, this must be reflected in the scale setting passed to the `AnalogPotentiometer`.

If the user would like to apply custom settings to the underlying `AnalogInput` used by the `AnalogPotentiometer`, an alternative constructor may be used in which the `AnalogInput` is injected:

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};
```

Reading values from the AnalogPotentiometer

The scaled value can be read by simply calling the `get` method:

JAVA

```
pot.get();
```

C++

```
pot.Get();
```

Using AnalogPotentiometers in code

Analog sensors can be used in code much in the way other sensors that measure the same thing can be. If the analog sensor is a potentiometer measuring an arm angle, it can be used similarly to an *encoder*. If it is an ultrasonic sensor, it can be used similarly to other *ultrasonics*.

It is very important to keep in mind that actual, physical potentiometers generally have a limited range of motion. Safeguards should be present in both the physical mechanism and the code to ensure that the mechanism does not break the sensor by traveling past its maximum throw.

15.3.9 Digital Inputs - Software

备注: This section covers digital inputs in software. For a hardware guide to digital inputs, see 数字输入 - 硬件.

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 数字输入 - 硬件). Accordingly, digital inputs are most-commonly used with switches of some sort. Support for this usage is provided through the `DigitalInput` class (Java, C++).

The DigitalInput class

A `DigitalInput` can be initialized as follows:

JAVA

```
// Initializes a DigitalInput on DIO 0
DigitalInput input = new DigitalInput(0);
```

C++

```
// Initializes a DigitalInput on DIO 0
frc::DigitalInput input{0};
```

Reading the value of the DigitalInput

The state of the `DigitalInput` can be polled with the `get` method:

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();
```

Creating a DigitalInput from an AnalogInput

备注: An `AnalogTrigger` constructed with a port number argument can share that analog port with a separate `AnalogInput`, but two *AnalogInput* objects may not share the same port.

Sometimes, it is desirable to use an analog input as a digital input. This can be easily achieved using the `AnalogTrigger` class (Java, C++).

An `AnalogTrigger` may be initialized as follows. As with `AnalogPotentiometer`, an `AnalogInput` may be passed explicitly if the user wishes to customize the sampling settings:

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};
```

Setting the trigger points

备注: For details on the scaling of "raw" `AnalogInput` values, see *Analog Inputs - Software*.

To convert the analog signal to a digital one, it is necessary to specify at what values the trigger will enable and disable. These values may be different to avoid "dithering" around the transition point:

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);
```

Using DigitalInputs in code

As almost all switches on the robot will be used through a `DigitalInput`. This class is extremely important for effective robot control.

Limiting the motion of a mechanism

Nearly all motorized mechanisms (such as arms and elevators) in FRC® should be given some form of "limit switch" to prevent them from damaging themselves at the end of their range of motions. A short example is given below:

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 {
```

(续下页)

(接上页)

```
        spark.Set(0);
    }
}
```

Homing a mechanism

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 Programming Limit Switches

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.

Limit switches can have "normally open" or "normally closed" outputs. This will control if a high signal means the switch is opened or closed. To learn more about limit switch hardware see this *article*.

Controlling a Motor with Two Limit Switches

JAVA

```
DigitalInput toplimitSwitch = new DigitalInput(0);
DigitalInput bottomlimitSwitch = new DigitalInput(1);
PWMVictorSPX motor = new PWMVictorSPX(0);
Joystick joystick = new Joystick(0);

@Override
public void teleopPeriodic() {
    setMotorSpeed(joystick.getRawAxis(2));
}

public void setMotorSpeed(double speed) {
    if (speed > 0) {
        if (toplimitSwitch.get()) {
            // We are going up and top limit is tripped so stop
            motor.set(0);
        } else {
            // We are going up but top limit is not tripped so go at commanded speed
            motor.set(speed);
        }
    } else {
        if (bottomlimitSwitch.get()) {
            // We are going down and bottom limit is tripped so stop
            motor.set(0);
```

(续下页)

(接上页)

```
        } 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 Addressable LEDs

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*.

Instantiating the AddressableLED Object

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.

> **警告：** It is important to note that setting the length of the LED header is an expensive task and it's **not** recommended to run this periodically.

After the length of the strip has been set, you'll have to create an `AddressableLEDBuffer` object that takes the number of LEDs as an input. You'll then call `myAddressableLed.setData(myAddressableLEDBuffer)` to set the led output data. Finally, you can call `myAddressableLed.start()` to write the output continuously. Below is a full example of the initialization process.

> **备注：** C++ does not have an AddressableLEDBuffer, and instead uses an Array.

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();
}
```

备注： 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.

Setting the Entire Strip to One Color

Color can be set to an individual led on the strip using two methods. `setRGB()` which takes RGB values as an input and `setHSV()` which takes HSV values as an input.

Using RGB Values

RGB stands for Red, Green, and Blue. This is a fairly common color model as it's quite easy to understand. LEDs can be set with the `setRGB` method that takes 4 arguments: index of the LED, amount of red, amount of green, amount of blue. The amount of Red, Green, and Blue are integer values between 0-255.

Java

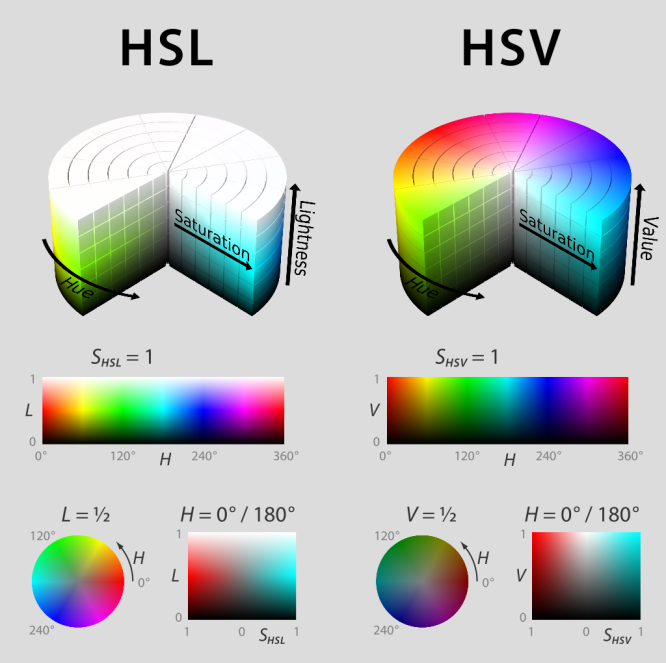
```
for (var i = 0; i < m_ledBuffer.getLength(); i++) {
   // Sets the specified LED to the RGB values for red
   m_ledBuffer.setRGB(i, 255, 0, 0);
}

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);
```

Using HSV Values

HSV stands for Hue, Saturation, and Value. Hue describes the color or tint, saturation being the amount of gray, and value being the brightness. In WPILib, Hue is an integer from 0 - 180. Saturation and Value are integers from 0 - 255. If you look at a color picker like Google's, Hue will be 0 - 360 and Saturation and Value are from 0% to 100%. This is the same way that OpenCV handles HSV colors. Make sure the HSV values entered to WPILib are correct, or the color produced might not be the same as was expected.



LEDs can be set with the `setHSV` method that takes 4 arguments: index of the LED, hue, saturation, and value. An example is shown below for setting the color of an LED strip to red (hue of 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);
```

Creating a Rainbow Effect

The below method does a couple of important things. Inside of the *for* loop, it equally distributes the hue over the entire length of the strand and stores the individual LED hue to a variable called `hue`. Then the for loop sets the HSV value of that specified pixel using the `hue` value.

Moving outside of the for loop, the `m_rainbowFirstPixelHue` then iterates the pixel that contains the "initial" hue creating the rainbow effect. `m_rainbowFirstPixelHue` then checks to make sure that the hue is inside the hue boundaries of 180. This is because HSV hue is a value from 0-180.

备注: It's good robot practice to keep the `robotPeriodic()` method as clean as possible, so we'll create a method for handling setting our LED data. We'll call this method `rainbow()` and call it from `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;
}
```

Now that we have our `rainbow` method created, we have to actually call the method and set the data of the 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 Motor Controllers

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.

备注： Another name for a motor controller is a speed controller.

提示： One can make a quick, non-competition-legal motor controller by removing the motor from a cordless BRUSHED drill and attaching PowerPoles or equivalents to the motor's leads. Make sure that the voltage supplied by the drill will not damage the motor, but note that the 775 is fine at up to 24 volts.

警告： Connecting a BRUSHLESS motor controller straight to power, such as to a conventional brushed motor controller, will destroy the motor!

15.5.1 FRC Legal Motor Controllers

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 Motor Controller (P/N: 410-334-1, 410-334-2)
- Jaguar Motor Controller (P/N: MDL-BDC, MDL-BDC24, and 217-3367) connected to *PWM* only
- Nidec Dynamo BLDC Motor with Controller to control integral actuator only (P/N 840205-000, am-3740)
- SD540 Motor Controller (P/N: SD540x1, SD540x2, SD540x4, SD540Bx1, SD540Bx2, SD540Bx4, SD540C)
- Spark Flex Motor Controller (P/N REV-11-2159, am-5276)
- Spark Motor Controller (P/N: REV-11-1200, am-4260)
- Spark MAX Motor Controller (P/N: REV-11-2158, am-4261)
- Talon FX Motor Controller (P/N 217-6515, 19-708850, am-6515, am-6515_Short, WCP-0940) for controlling integral Falcon 500 or Kraken X60 only,
- Talon Motor Controller (P/N: CTRE_Talon, CTRE_Talon_SR, and am-2195)
- Talon SRX Motor Controller (P/N: 217-8080, am-2854, 14-838288)
- Venom Motor with Controller (P/N BDC-10001) for controlling integral motor only
- Victor 884 Motor Controller (P/N: VICTOR-884-12/12)
- Victor 888 Motor Controller (P/N: 217-2769)
- Victor SP Motor Controller (P/N: 217-9090, am-2855, 14-868380)
- Victor SPX Motor Controller (P/N: 217-9191, 17-868388, am-3748)

15.6 Pneumatics

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 *Pneumatics APIs*.

15.6.1 FRC Legal Pneumatics controllers

- Pneumatics Control Module (P/N: am-2858, 217-4243)
- Pneumatic Hub (P/N REV-11-1852)

15.7 Relays

A relay controls power to a motor or custom electronics in an On/Off fashion.

15.7.1 FRC Legal Relay Modules

- Spike H-Bridge Relay (P/N: 217-0220 and SPIKE-RELAY-H)
- Automation Direct Relay (P/N: AD-SSR6M12-DC200D, AD-SSR6M25-DC200D, AD-SSR6M40-DC200D)
- Power Distribution Hub (PDH) switched channel (P/N REV-11-1850)

16 CAN 设备

16.1 使用 CAN 设备

CAN has many advantages over other methods of connection between the robot controller and peripheral devices.

- CAN 连接以菊链的形式连接设备，这种方式与将每个设备都连接到 RIO 上的接线时间短得多。
- 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 连接是双向的，所以 CAN 电机可以把数据传回 RIO，因此 CAN 电机比 PWM 电机可以拥有更多的功能。

For instructions on wiring CAN devices, see the relevant section of the *robot wiring guide*.

CAN 设备通常在 WPILib 中拥有自己的类。以下几节将介绍这些类的用法。

16.2 气动控制模块

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 Power Distribution Module

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.

> **警告:** 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 Reading the Bus Voltage

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++

```
    // Get the voltage going into the PDP, in Volts.
    // The PDP returns the voltage in increments of 0.05 Volts.
    double voltage = m_pdp.GetVoltage();
    frc::SmartDashboard::PutNumber("Voltage", voltage);
```

PYTHON

```
        # Get the voltage going into the PDP, in Volts.
        # The PDP returns the voltage in increments of 0.05 Volts.
        voltage = self.pdp.getVoltage()
        wpilib.SmartDashboard.putNumber("Voltage", voltage)
```

Monitoring the bus voltage can be useful for (among other things) detecting when the robot is near a brownout, so that action can be taken to avoid brownout in a controlled manner. See the *roboRIO Brownouts document* for more information.

16.4.3 Reading the Temperature

JAVA

```
    // Retrieves the temperature of the PDP, in degrees Celsius.
    double temperatureCelsius = m_pdp.getTemperature();
    SmartDashboard.putNumber("Temperature", temperatureCelsius);
```

C++

```
    // Retrieves the temperature of the PDP, in degrees Celsius.
    double temperatureCelsius = m_pdp.GetTemperature();
    frc::SmartDashboard::PutNumber("Temperature", temperatureCelsius);
```

PYTHON

```
        # Retrieves the temperature of the PDP, in degrees Celsius.
        temperatureCelsius = self.pdp.getTemperature()
        wpilib.SmartDashboard.putNumber("Temperature", temperatureCelsius)
```

Monitoring the temperature can be useful for detecting if the robot has been drawing too much power and needs to be shut down for a while, or if there is a short or other wiring problem.

16.4.4 Reading the Total Current, Power, and Energy

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++

```
    // 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

```
        # 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 Reading Individual Channel Currents

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

```
26      // Get the current going through channel 7, in Amperes.
27      // The PDP returns the current in increments of 0.125A.
28      // At low currents the current readings tend to be less accurate.
29      double current7 = m_pdp.getCurrent(7);
30      SmartDashboard.putNumber("Current Channel 7", current7);
```

C++

```
22      // Get the current going through channel 7, in Amperes.
23      // The PDP returns the current in increments of 0.125A.
24      // At low currents the current readings tend to be less accurate.
25      double current7 = m_pdp.GetCurrent(7);
26      frc::SmartDashboard::PutNumber("Current Channel 7", current7);
```

PYTHON

```
28          # Get the current going through channel 7, in Amperes.
29          # The PDP returns the current in increments of 0.125A.
30          # At low currents the current readings tend to be less accurate.
31          current7 = self.pdp.getCurrent(7)
32          wpilib.SmartDashboard.putNumber("Current Channel 7", current7)
```

Monitoring individual device current draws can be useful for detecting shorts or stalled motors.

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 第三方 CAN 设备

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*

以下提供了来自各个供应商的一些常见的第三方 CAN 设备，以及连接外部文档的链接。

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

CTRE 电机控制器

- ** Talon FX（搭配 Falcon 500 马达）**
  - 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

CTRE 传感器

- **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 机器人

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.

REV 电机控制器

- **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) 提供了 Venom 集成电机/控制器和飞行时间与距离传感器:

PWF 电机控制器

- **Venom**
  - API 文档 (Java, C++)
  - **‘技术手册 <https://www.playingwithfusion.com/include/getfile.php?fileid=7086>‘__**

PWF 传感器

- **Time of Flight Sensor**
  - API 文档 (Java, C++)
  - **‘技术手册 <https://www.playingwithfusion.com/include/getfile.php?fileid=7091>‘__**

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 FRC CAN 设备规格

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 编址

FRC CAN 节点根据预定义的方案分配仲裁 ID，该方案将 ID 分为 5 个部分：

设备类型

这是一个五位的数值，它代表了被编址的设备的类型。当前分配的设备类型可以在下表中找到。如果你想从“Reserved“库中选取新的设备类型，请向 FIRST 提交申请。

设备类型	
广播消息	0
机器人控制器	1
电机控制器	2
继电器控制器	3
陀螺仪传感器	4
加速度计	5
超声波传感器	6
齿轮传感器	7
配电模块	8
气动控制器	9
杂项	10
IO 接口	11
预留	12-30
固件升级	31

制造商

这是一个 8 位的数值，它代表了被编址的设备的制造商。当前分配的设备类型可以在下表中找到。如果你想从 “Reserved“库中选取新的制造商 ID，请向 FIRST 提交申请。

制造商	
广播	0
你	1
发光微	2
德卡	3
点阅电子	4
REV 机器人技术	5
抓钩	6
心灵感应器	7
团队使用	8
考艾岛实验室	9
Copperforge	10
Playing with Fusion	11
Studica	12
The Thrifty Bot	13
Redux Robotics	14
AndyMark	15
Vivid Hosting	16
预留	17-255

API /消息标识符

API/消息标识符是一个 10 位的数值，它代表了一个特定的指令或消息类型。这些标识符对于每一个制造商 + 设备类型都是唯一的（所以它对于 Luminary Micro 的电机可能是 “电压设置”，对于 CTR Electronic 的电机可能是 “状态获取”，对于 CTR 配电模块可能是 “通电 Current Get ”）

消息标识符进一步细分为 2 个子字段：6 位 API 类和 4 位 API 索引。

API 类

API 类别是 API 分组的 6 位数值。类似的消息被分组到单个 API 类中。下表显示了 Jaguar 电机的 API 类的示例。

API 类	
电压控制模式	0
调速模式	1
电压补偿模式	2
位置控制模式	3
电流控制模式	4
状态	5
周期状态	6
配置	7
确认	8

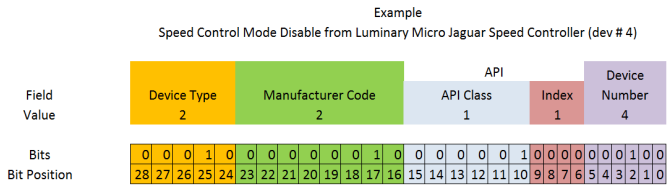
API 索引

API 索引是 API 类中特定消息的 4 位数值。下表显示了 Jaguar 电机速度控制 API 类的 API 索引值的示例。

API 索引	
启用控制	0
禁用控制	1
设定设定值	2
P 常数	3
I 常数	4
D 常数	5
设置参考	6
受信任的启用	7
受信任的设置无确认	8
可信设定设定值无确认	10
设定设定值无确认	11

设备编号

设备编号是一个 6 位数字，它代表特定类型的设备编号。设备应默认为设备 ID 0，以匹配 FRC 控制系统的其他组件。可以为设备特定的广播消息保留设备 0x3F。



16.6.2 受保护的框架

有执行、控制功能的 FRC CAN 节点（电机控制器，继电器，气动控制器等）必须采用一种方法来验证机器人是否已启用以及命令是否来自主机器人控制器（即 roboRIO）。

16.6.3 广播消息

广播消息是通过将设备类型和制造商字段设置为 0 来发送到所有节点的消息。广播消息的 API 类为 0。下表中显示了当前定义的广播消息：

描述	
禁用	0
系统停止	1
系统重置	2
设备分配	3
设备查询	4
中心	5
同步	6
更新	7
固件版本	8
枚举	9
系统恢复	10

Devices should disable immediately when receiving the Disable message (arbID 0). Implementation of other broadcast messages is optional.

16.6.4 FRC CAN 节点的要求

为了使 CAN 节点可以在 FRC 系统中使用，它们必须：

- 使用与规定的 FRC 格式匹配的 ID 进行通信：
  - 有效的已发布的 CAN 设备类型（根据表 1-CAN 设备类型）
  - 有效的，已发布的制造商 ID（根据表 2-CAN 制造商代码）
  - 由设备制造商分配和记录的 API 类与 API 索引
  - 如果设备类型的多个单元要在同一网络上共存，则用户可以选择设备编号。
- 符合“广播消息”部分中详细介绍的最低广播消息要求。
- 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.

描述	Byte	Width (bits)
Match time (seconds)	8	8
Match number	6-7	10
Replay number	6	6
Red alliance	5	1
Enabled	5	1
Autonomous mode	5	1
Test mode	5	1
System watchdog	5	1
Tournament type	5	3
Time of day (year)	4	6
Time of day (month)	3-4	4
Time of day (day)	3	5
Time of day (seconds)	2-3	6
Time of day (minutes)	1-2	6
Time of day (hours)	1	5

```
struct [[gnu::packed]] RobotState {
  uint64_t matchTimeSeconds : 8;
  uint64_t matchNumber : 10;
  uint64_t replayNumber : 6;
  uint64_t redAlliance : 1;
  uint64_t enabled : 1;
  uint64_t autonomous : 1;
  uint64_t testMode : 1;
  uint64_t systemWatchdog : 1;
  uint64_t tournamentType : 3;
  uint64_t timeOfDay_yr : 6;
  uint64_t timeOfDay_month : 4;
  uint64_t timeOfDay_day : 5;
  uint64_t timeOfDay_sec : 6;
  uint64_t timeOfDay_min : 6;
  uint64_t timeOfDay_hr : 5;
};
```

If the `System watchdog` flag is set, motor controllers are enabled. If 100 ms has passed since this packet was received, the robot program can be considered hung, and devices should act as if the robot has been disabled.

Note that all fields except `Enabled`, `Autonomous mode`, `Test mode`, and `System watchdog` will contain invalid values until an arbitrary time after the Driver Station connects.

17

基础编程

17.1 Git 版本控制介绍

重要： 有关 Git 的更深入的指南可以在 Git 网站 上找到。

git 是 Linus Torvalds 创建的分布式版本控制系统 (VCS)，也因创建和维护 Linux 内核而闻名。版本控制是一个为开发人员跟踪代码更改的系统。Git 版本控制的优点是：

- Separation of testing environments into *branches*
- 能够导航到特定的 提交而不删除历史记录
- 能够以多种方式管理 提交，包括将它们组合在一起
- 各种其他功能，请参见 这里

17.1.1 先决条件

重要： 本教程使用 Windows 操作系统

你必须从以下链接下载并安装 Git：

- Windows
- macOS
- Linux

备注： 你可能需要将 Git 添加到 路径 *<https://www.google.com/search?q=adding+git+to+path>*

17.1.2 Git 词汇

Git revolves around several core data structures and commands:

- **Repository** 代码的数据结构，包括根目录中的.git 文件夹
- **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:** 使用本地更改更新远程存储库
- **Pull:** 使用远程更改更新本地存储库
- **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 资料库

Git 存储库是一种数据结构，其中包含项目的结构，历史记录和文件。

Git 存储库通常包括：

- 一个.git 文件夹。此文件夹包含有关存储库的各种信息。
- 一个.gitignore 文件。该文件包含您你在提交时不希望包含的文件或目录。
- 文件和文件夹。这是存储库的主要内容。

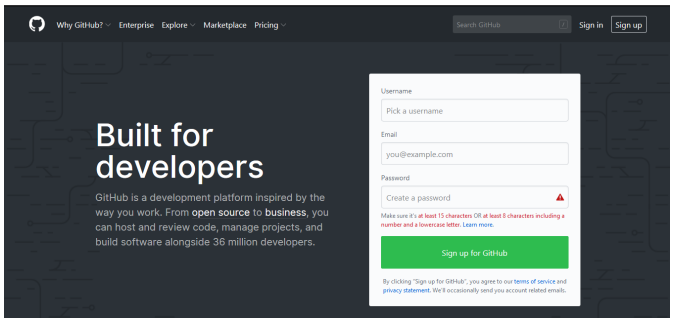
创建存储库

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.

备注: 有许多提供托管存储库服务的提供者。“Gitlab <https://about.gitlab.com>” _ 和 “Bitbucket <https://bitbucket.org/>” _ 是 Github 的一些替代方案。

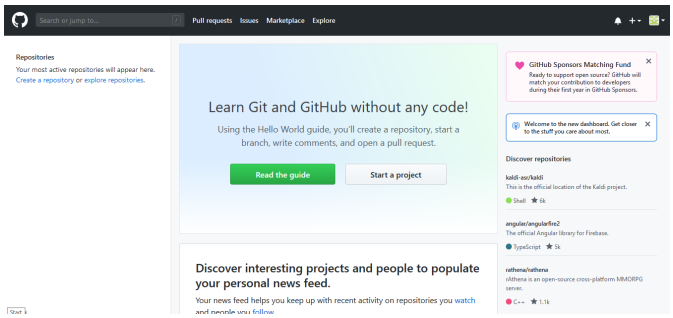
创建一个 GitHub 帐户

Go ahead and create a GitHub account by visiting the website and following the on-screen prompts.

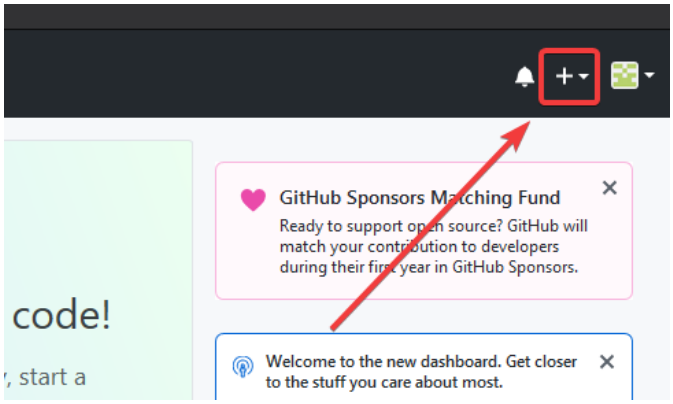


本地创作

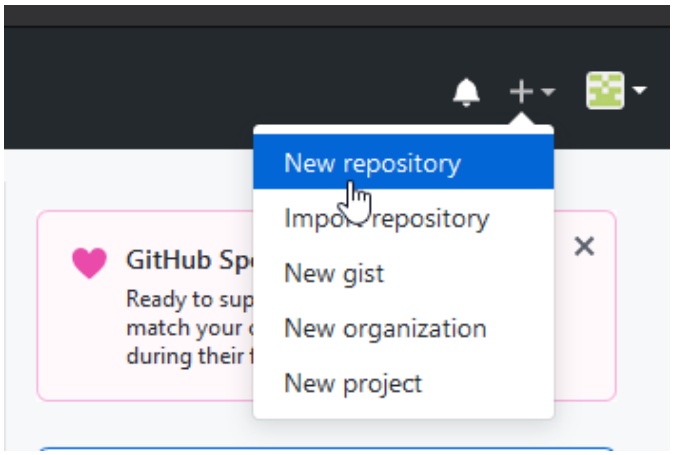
创建并验证您的帐户后，你会要访问主页。它看起来类似于展示的图像。



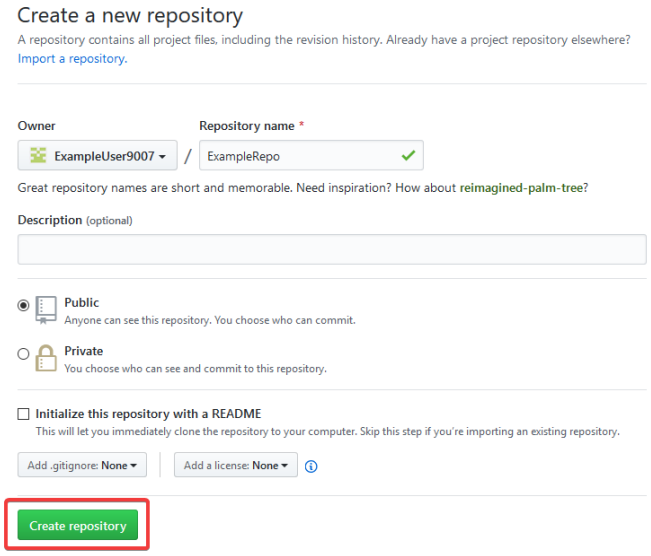
单击右上角的加号图标。



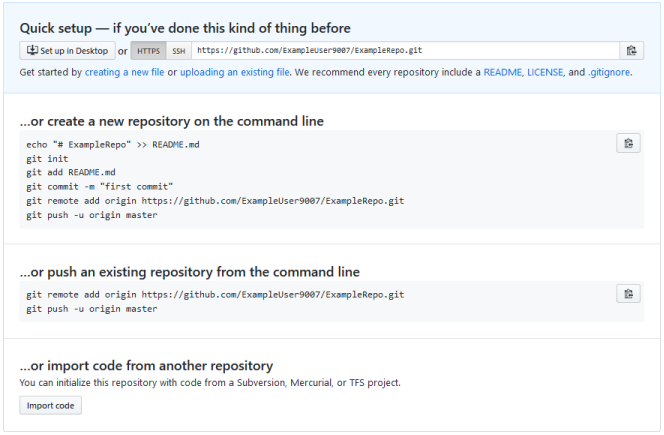
然后点击 * “New Repository” *



填写适当的信息，然后单击* “Create repository” *



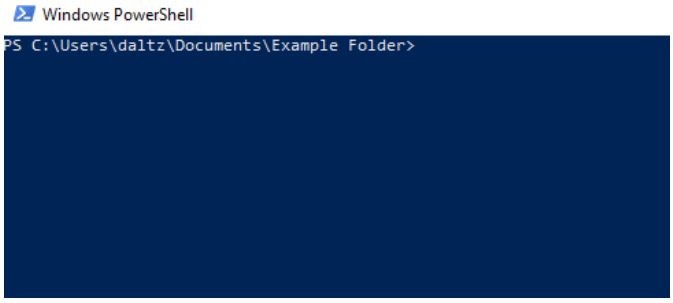
你应该会在屏幕上看到与此类似的内容



备注: The keyboard shortcut `Ctrl+~` can be used to open a terminal in Visual Studio Code

for Windows.

现在，你将要打开一个 PowerShell 窗口并导航到您的项目目录。可以在此处找到 <https://programminghistorian.org/en/lessons/intro-to-powershell> __ 上有关 PowerShell 的优秀教程。请使用你的搜索引擎，以了解如何在其他操作系统上打开终端。



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.

备注： 将文件路径 “C：Users ExampleUser9007 Documents Example Folder” 替换为你要在其中创建存储库的路径，然后替换远程 URL ‘https://github.com/ExampleUser9007/ExampleRepo.git’ . 以及你在先前步骤中创建的仓库的 URL。

空目录

```
> 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
```

现有的项目

```
> 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 提交

存储库主要由提交组成。提交是保存状态或代码的“版本”。

在前面的示例中，我们创建了一个名为 README.md 的文件。在你喜欢的文本编辑器中打开该文件，然后编辑几行。修补文件后，只需保存并关闭即可。导航到 PowerShell，然后键入以下命令。

```
> 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
```

备注： Writing good commit messages is a key part of a maintainable project. A guide on writing commit messages can be found here.

Git Pull

备注： 当用户不希望自动合并到当前工作分支时，可以使用 `git fetch` 。

此命令从远程存储库检索历史记录或提交。当遥控器包含你没有的工作时，它将尝试自动合并。请参阅: docs/software/basic-programming/git-getting-started:Merging。

运行: `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.

运行: `git commit -m " 在此处输入消息"`

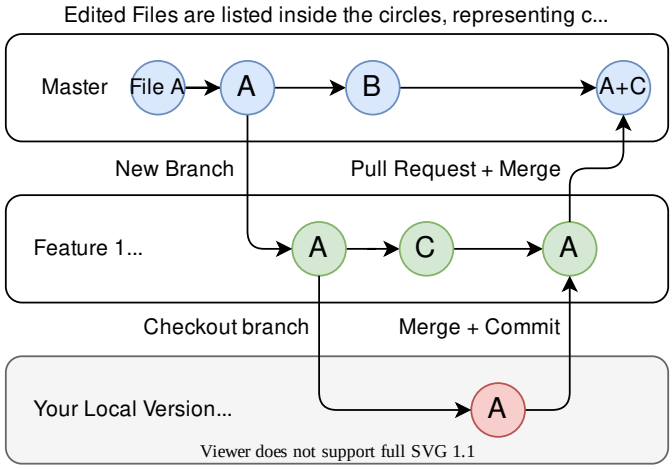
Git Push

将本地更改上传 (推送) 到远程 (云)

运行: `git push`

17.1.5 分支

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.



在上面的示例中，main 被分支 (或复制) 到分支 Feature 1 中，并且有人查看了分支，从而创建了本地副本。然后，某人提交 (或上传) 了他们的更改，将其合并到分支功能 1 中。你正在将更改从一个分支"合并"到另一个分支。

创建一个分支

运行: `git branch branch-name`，其中 branch-name 是要创建的分支的名称。新的分支历史将从当前的活动分支中创建。

输入一个分支

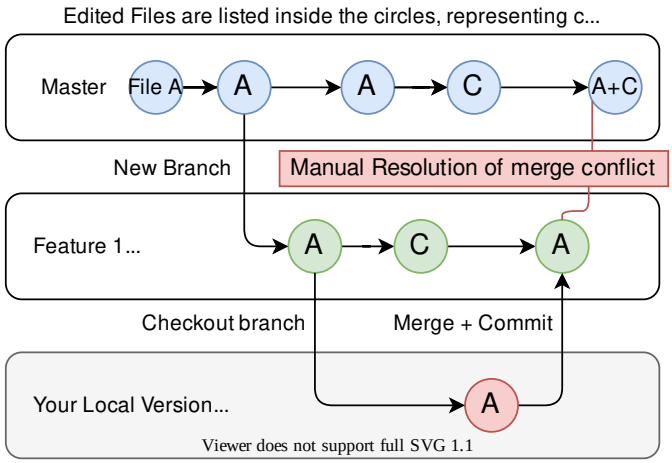
创建分支后，你必须进入分支。

运行: `git checkout branch-name` 其中 branch-name 是先前创建的分支。

17.1.6 合并

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 重置

有时历史记录需要被重置，或者提交需要被撤消。这可以通过多种方式完成。

还原提交

备注: 你无法还原合并，因为 git 不知道应选择哪个分支或源。

要还原导致提交的历史记录，运行 `git revert commit-id`。可以使用 `git log` 命令显示提交 ID。

重设 Head 指针

警告： 强制磁头复位是一个危险的指令。此指令将永久删除目标之外的所有历史记录。请三思后行！

运行: `git reset --hard commit-id`。

17.1.8 Forks

可以像处理分支一样处理 Fork。你可以将上游库 (原始存储库) 合并到本地库 (克隆存储库) 中。

克隆已存在的 "Repo"

在已经创建存储库并将其存储在远程服务器上的情况下，可以使用以下命令克隆该存储库:

```
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*.

更新 Fork

1. 添加上游: `git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git`
2. 确认通过以下方式添加了它: `git remote -v`
3. 从上游进行 Pull 更改: `git fetch upstream`
4. 将更改整合进 head: `git merge upstream/upstream-branch-name`

17.1.9 Gitignore

重要： 队伍 **请勿**修改其机器人项目中随附的 `.gitignore` 文件，这一点非常重要。这可能导致下线部署无法正常工作。

一个 `.gitignore` 文件通常用作不使用 `git add` 自动提交的文件列表。此文件中列出的任何文件或目录都将 **不提交**。它们也不会显示为 git status.

Additional Information can be found here.

隐藏一个文件夹

只需添加包含要隐藏的文件夹的新行，并在末尾加正斜杠

例如: `directory-to-exclude/`

隐藏一个文件

用要隐藏的文件名添加新行，包括相对于存储库根目录的任何前缀目录。

例如:“ directory / file-to-hide.txt “

例如:“ file-to-hide2.txt “

17.1.10 附加的信息

在 git <*https://git-scm.com/docs/gittutorial*> 官方网站上可以找到更深入的教程。

可以在 git‘flight 规则 <https://github.com/k88hudson/git-flight-rules/blob/master/README.md>‘_ 存储库中找到纠正常见错误的指南。

17.2 C++ 单位库

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 使用单位库

单位库是仅有标头的库。您必须在源文件中包含要使用的单位的相关标头。下面是可用单位的列表。

```
#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>
```

(续下页)

(接上页)

```
#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>
```

“units/math.h”标头提供了单位感知功能，例如“units::math::abs()”。

单位类型和容器类型

C ++ 单位库基于两种类型定义：单位类型和容器类型。

单位类型

单元类型对应于单元的抽象概念，没有任何实际的存储值。单元类型是单元库的基本“构建块”-所有单元类型都通过少量“基本”单元类型（例如“meters”，“‘seconds’等）在被构造时定义（通过“compound_unit”模版）。

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.

容器类型

容器类型对应于根据某个单位确定尺寸的实际数量。也就是说，它们是实际包含数值的内容。容器类型是使用具有“unit_t”模板的单元类型构造的。大多数单位类型都有一个对应的容器类型，其名称后缀有“_t”-例如，单元类型“units :: meter”对应于容器类型“units :: meter_t”。

每当使用特定数量的单位（作为变量或方法参数）时，它将是容器类型的实例。默认情况下，容器类型会将实际值存储为“double”-高级用户可以通过手动调用“unit_t”模板来更改此值。

单位和容器类型的完整列表可以在文档“<https://github.com/nholthaus/units#namespaces>”__中找到。

创建单位的实例

为了创建特定单元的实例，我们需要创建其容器类型的实例：

```
// The variable speed has a value of 5 meters per second.
units::meter_per_second_t speed{5.0};
```

另外，单位库还为一些较常见的容器类型定义了“类型文字 <https://en.cppreference.com/w/cpp/language/user_literal>”__。这些可以与通过“auto”进行的类型推断结合使用，以更简洁地定义一个单元：

```
// The variable speed has a value of 5 meters per second.
auto speed = 5_mps;
```

只要可以在另一个容器类型之间进行转换，单元也可以使用另一个容器类型的值来初始化。例如，可以从“foot_t”值创建“meter_t”值。

```
auto feet = 6_ft;
units::meter_t meters{feet};
```

实际上，代表可转换单位类型的所有容器类型都是“隐式可转换的”。因此，以下内容完全合法：

```
units::meter_t distance = 6_ft;
```

简而言之，我们可以在代码中的任何位置使用 *任意长度单位代替 任何其他长度单位*；单位库将自动为我们执行正确的转换。

执行含有单位的算术

容器类型支持其基础数据类型的所有普通算术运算，其附加条件是该运算必须是 *维的*。因此，必须始终对两种兼容的容器类型执行添加：

```
// 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;
```

乘法可以对任何一对容器类型执行，并得出复合单元的容器类型：

备注： 当计算产生复合单位类型时，如果明确指定结果类型，则仅在操作时检查此类型的有效性。如果使用“auto”，则不会进行此检查。例如，当我们将距离除以时间时，我们希望确保结果确实是速度即（“units :: meter_per_second_t”）。如果返回类型声明为“auto”，则不会进行此检查。

```
// 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>'' 函数

一些“std”函数 (例如 `clamp`) 被模板化以接受可以在其上执行算术运算的任何类型。以容器类型存储的数量可以使用这些功能而不会出现问题。

但是，其他“std“函数仅适用于普通数字类型 (例如 `double`)。单元库的 `units :: math` 命名空间包含一些接受单元的函数的包装。此类功能的示例包括 `sqrt`, `pow` 等。

```
auto area = 36_sq_m;
units::meter_t sideLength = units::math::sqrt(area);
```

去除 Unit Wrapper

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 WPILib 代码中的单位库示例

WPILib 新功能中的函数的几个参数 (例如: kinematics <docs/software/kinematics-and-odometry/intro-and-chassis-speeds:What is kinematics?>) 使用单位库。这是对轨迹 <docs/software/advanced-controls/trajectories/manipulating-trajectories:Sampling the trajectory> 进行采样的示例。

```
// 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);
```

某些 WPILib 类表示可以自然地与多种单位类型一起使用的对象 -例如，运动曲线可能会在线性距离 (例如米) 或角距离 (例如弧度) 上运行。对于这种类，必须将单位类型作为模板参数:

```
// 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};
```

有关更多详细文档，请访问官方 GitHub 页面 获取单位库。

17.3 The Java Units Library

The units library is a tool that helps programmers avoid mistakes related to units of measurement. It does this by keeping track of the units of measurement, and by ensuring that all operations are performed with the correct units. This can help to prevent errors that can lead to incorrect results, such as adding a distance in inches to a distance in meters.

An added benefit is readability and maintainability, which also reduces bugs. By making the units of measurement explicit in your code, it becomes easier to read and understand what your code is doing. This can also help to make your code more maintainable, as it is easier to identify and fix errors related to units of measurement.

The units library has a number of features:

- A set of predefined units, such as meters, degrees, and seconds.
- The ability to convert between different units.
- Support for performing arithmetic and comparisons on quantities with units.
- Support for displaying quantities with units in a human-readable format.

17.3.1 Terminology

Dimension
: Dimensions represent the nature of a physical quantity, such as length, time, or mass. They are independent of any specific unit system. For example, the dimension of meters is length, regardless of whether the length is expressed in meters, millimeters, or inches.

Unit
: Units are specific realizations of dimensions. They are the way of expressing physical quantities. Each dimension has a base unit, such as the meter for length, the second for time, the kilogram for mass. Derived units are formed by combining base units, such as meters per second for velocity.

Measure
: Measures are the specific magnitude of physical quantities, expressed in a particular unit. For example, 5 meters is a measure of distance.

These concepts are used within the Units Library. For example, the **measure** *10 seconds* has a magnitude of 10, the **dimension** is time, and the **unit** is seconds.

17.3.2 Using the Units Library

The Java units library is available in the `edu.wpi.first.units` package. The most relevant classes are:

- The various classes for predefined dimensions, such as Distance and Time
- Units, which contains a set of predefined units. Take a look a the Units javadoc to browse the available units and their types.
- Measure, which is used to tag a value with a unit.

备注: It is recommended to static import `edu.wpi.first.units.Units.*` to get full access to all the predefined units.

Java Generics

Units of measurement can be complex expressions involving various dimension, such as distance, time, and velocity. Nested generic type parameters allow for the definition of units that can represent such complex expressions. Generics are used to keep the library concise, reusable, and extensible, but it tends to be verbose due to the syntax for Java generics.

For instance, consider the type `Measure<Velocity<Distance>>`. This type represents a measurement for velocity, where the velocity itself is expressed as a unit of distance per unit of time. This nested structure allows for the representation of units like meters per second or feet per minute. Similarly, the type `Measure<Per<Voltage, Velocity<Distance>>>` represents a measurement for a ratio of voltage to velocity. This type is useful for representing quantities like volts per meter per second, the unit of measure for some *feedforward* gains.

It's important to note that not all measurements require such complex nested types. For example, the type `Measure<Distance>` is sufficient for representing simple units like meters or feet. However, for more complex units, the use of nested generic type parameters is essential.

For local variables, you may choose to use Java's *var* keyword instead of including the full type name. For example, these are equivalent:

```
Measure<Per<Voltage, Velocity<Distance>>> v = VoltsPerMeterPerSecond.of(8);
var v = VoltsPerMeterPerSecond.of(8);
```

Creating Measures

The `Measure` class is a generic type that represents a magnitude (physical quantity) with its corresponding unit. It provides a consistent and type-safe way to handle different dimensions of measurements, such as distance, angle, and velocity, but abstracts away the particular unit (e.g. meter vs. inch). To create a `Measure` object, you call the `Unit.of` method on the appropriate unit object. For example, to create a `Measure<Distance>` object representing a distance of 6 inches, you would write:

```
Measure<Distance> wheelDiameter = Inches.of(6);
```

Other measures can also be created using their `Unit.of` method:

```
Measure<Mass> kArmMass = Kilograms.of(1.423);
Measure<Distance> kArmLength = Inches.of(32.25);
Measure<Angle> kMinArmAngle = Degrees.of(5);
Measure<Angle> kArmMaxTravel = Rotations.of(0.45);
Measure<Velocity<Distance>> kMaxSpeed = MetersPerSecond.of(2.5);
```

Performing Calculations

The `Measure` class also supports arithmetic operations, such as addition, subtraction, multiplication, and division. These are done by calling methods on the objects. These operations always ensure that the units are compatible before performing the calculation, and they return a new `Measure` object. For example, you can add two `Measure<Distance>` objects together, even if they have different units:

```
Measure<Distance> distance1 = Inches.of(10);
Measure<Distance> distance2 = Meters.of(0.254);

Measure<Distance> totalDistance = distance1.plus(distance2);
```

In this code, the units library will automatically convert the measures to the same unit before adding the two distances. The resulting `totalDistance` object will be a new `Measure<Distance>` object that has a value of 0.508 meters, or 20 inches.

This example combines the wheel diameter and gear ratio to calcualate the distance per rotation of the wheel:

```
Measure<Distance> wheelDiameter = Inches.of(3);
double gearRatio = 10.48;
Measure<Distance> distancePerRotation = wheelDiameter.times(Math.PI).
↪divide(gearRatio);
```

> **警告：** 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(
```

(续下页)

(接上页)

```
      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;
  }
}
```

警告： `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)));
}
```

备注: 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.

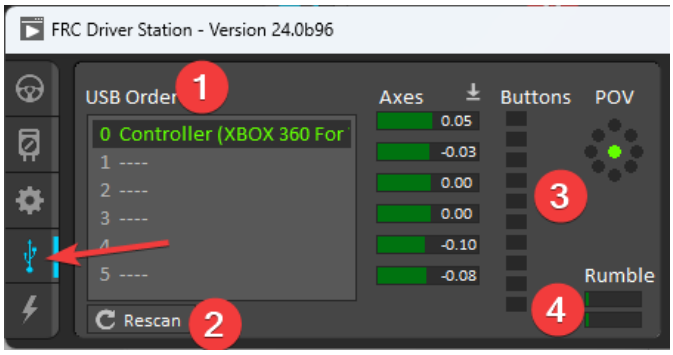
```
public class ElectricCharge extends Unit<ElectricCharge> {
  public ElectricCharge(double baseUnitEquivalent, String name, String symbol) {
    super(ElectricCharge.class, baseUnitEquivalent, name, symbol);
  }

  // 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: 将命令绑定到触发器.

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.

备注: 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.

重要: 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.

重要： 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
```

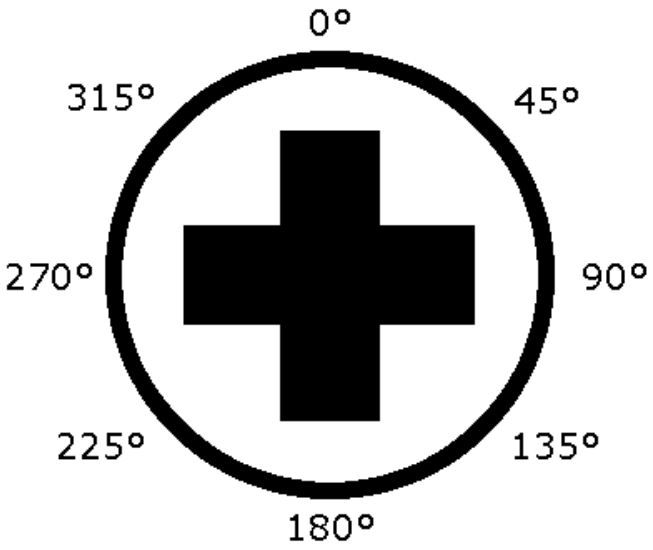
PYTHON

```
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)`.

重要： 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

备注： Usage such as the following is for code not using the command-based framework. For button usage in the command-based framework, see 将命令绑定到触发器.

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.

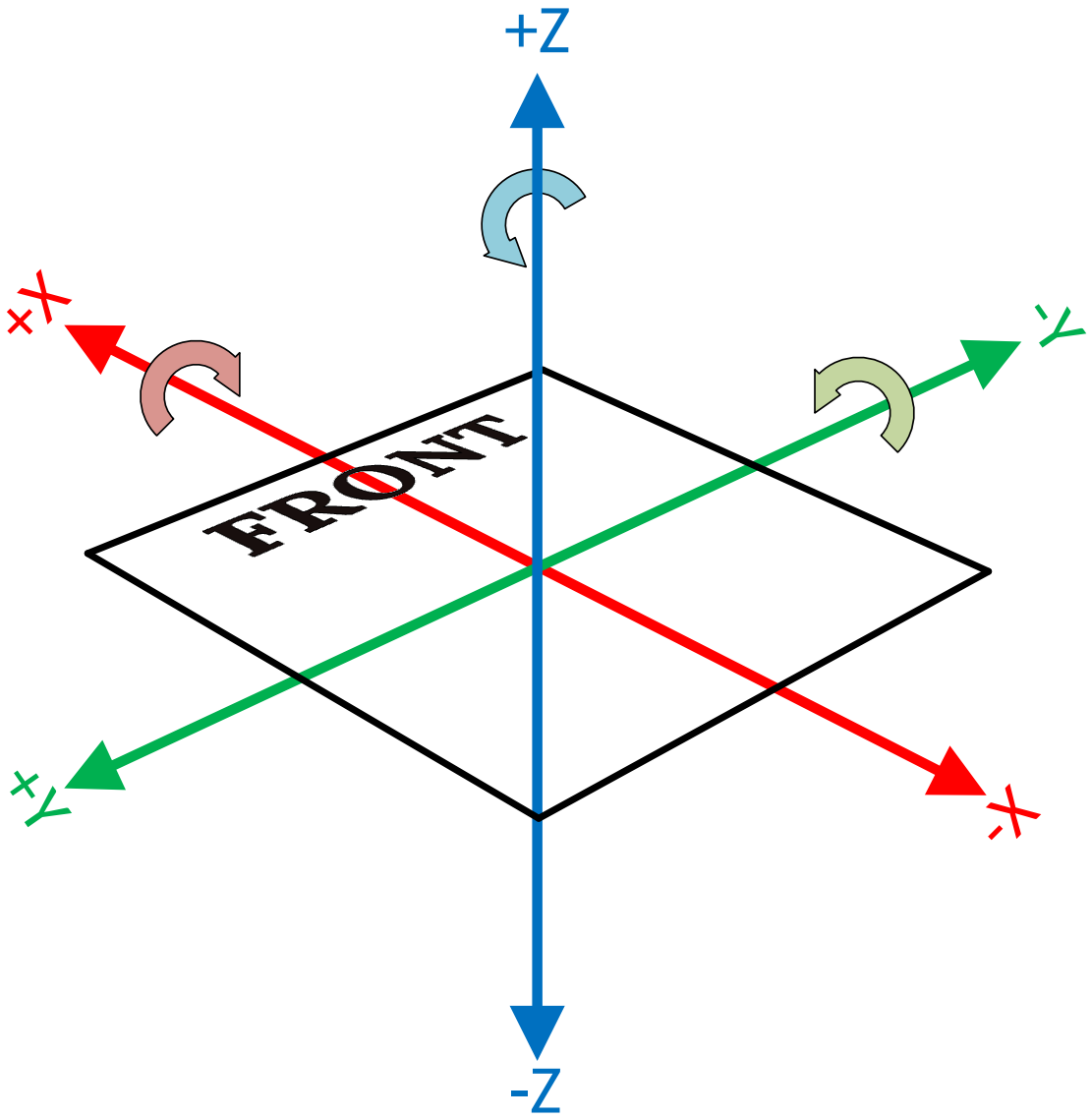


图 1: Robot coordinate system in three dimensions

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.

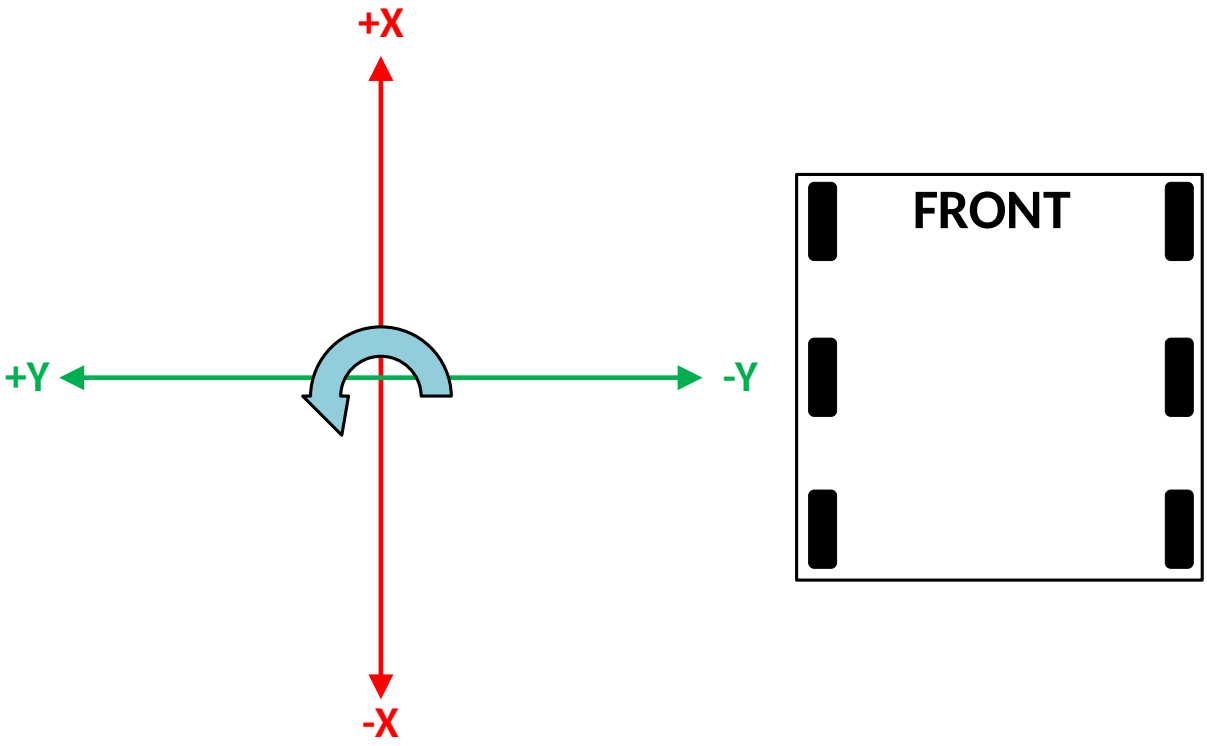


图 2: Robot coordinate system in two dimensions

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).

备注： 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.

警告： 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.

警告： Many sensors that read rotation around an axis, such as encoders and IMU's, read continuously. This means they read more than one rotation, so when rotating past 180°

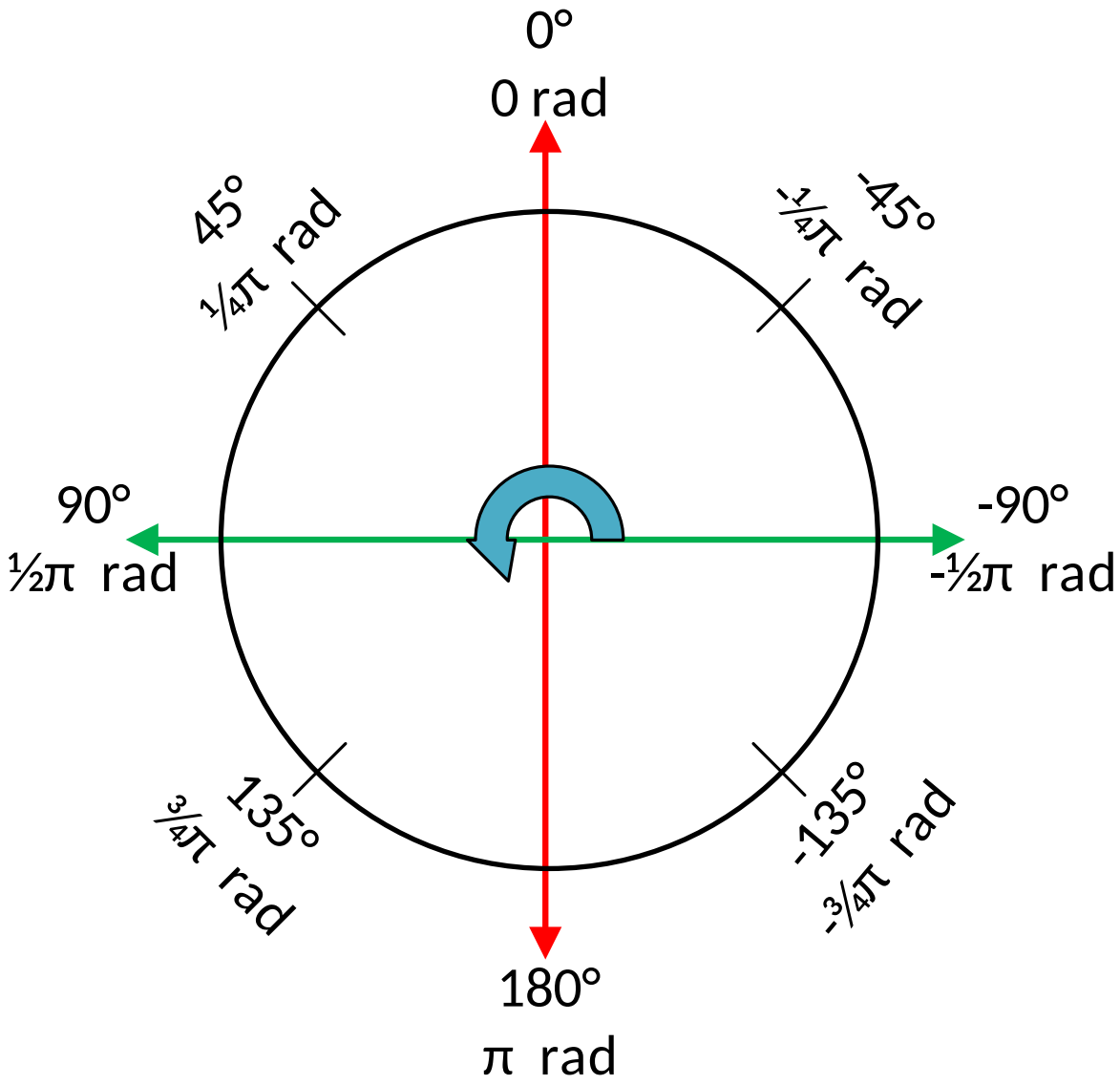


图 3: Unit circle with common angles

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.

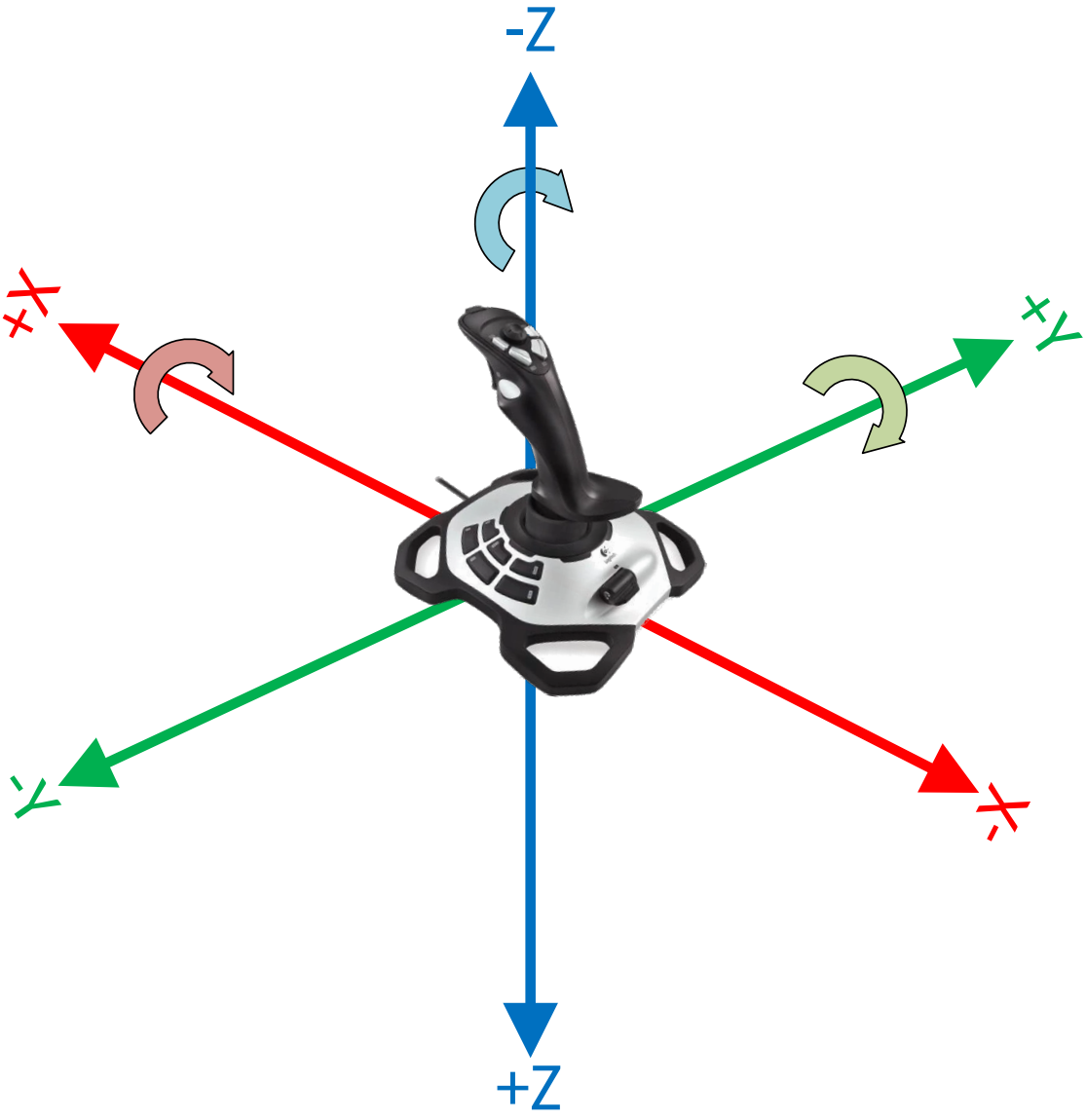


图 4: Joystick coordinate system

JAVA

```
public void teleopPeriodic() {
    // Arcade drive with a given forward and turn rate
    myDrive.arcadeDrive(-driveStick.getY(), -driveStick.getX());
}
```

C++

```
void TeleopPeriodic() override {
    // Arcade drive with a given forward and turn rate
    myDrive.ArcadeDrive(-driveStick.GetY(), -driveStick.GetX());
}
```

PYTHON

```
def teleopPeriodic(self):
    # Arcade drive with a given forward and turn rate
    self.myDrive.arcadeDrive(-self.driveStick.getY(), -self.driveStick.getX())
```

The code calls the `DifferentialDrive.arcadeDrive(xSpeed, zRotation)` method, with values it gets from the `Joystick` class:

- The first argument is `xSpeed`
  - Robot: `xSpeed` is the speed along the robot's X axis, which is forward/backward.
  - Joystick: The driver sets forward/backward speed by rotating the joystick along its Y axis, which is pushing the joystick forward/backward.
  - Code: Moving the joystick forward is negative Y rotation, whereas moving the robot forward is along the positive X axis. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.
- The second argument is `zRotation`
  - Robot: `zRotation` is the speed of rotation along the robot's Z axis, which is rotating left/right.
  - Joystick: The driver sets rotation speed by rotating the joystick along its X axis, which is pushing the joystick left/right.
  - Code: Moving the joystick to the right is positive X rotation, whereas robot rotation is CCW positive. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.

Mecanum drivetrain example

Mecanum drivetrains are holonomic, meaning they have the ability to move side-to-side. This type of drivetrain can move forward/backward and rotate around the Z axis like differential drivetrains, but it can also move side-to-side along the robot's Y axis. Consider a common arcade drive scheme using a single joystick where the driver pushes the joystick forward/backward for forward/backward robot movement, pushes the joystick left/right to move side-to-side, and twists the joystick to rotate the robot.

JAVA

```
public void teleopPeriodic() {
    // Drive using the X, Y, and Z axes of the joystick.
    m_robotDrive.driveCartesian(-m_stick.getY(), -m_stick.getX(), -m_stick.getZ());
}
```

C++

```
void TeleopPeriodic() override {
    // Drive using the X, Y, and Z axes of the joystick.
    m_robotDrive.driveCartesian(-m_stick.GetY(), -m_stick.GetX(), -m_stick.GetZ());
}
```

PYTHON

```
def teleopPeriodic(self):
    // Drive using the X, Y, and Z axes of the joystick.
    self.robotDrive.driveCartesian(-self.stick.getY(), -self.stick.getX(), -self.
↪stick.getZ())
```

The code calls the `MecanumDrive.driveCartesian(xSpeed, ySpeed, zRotation)` method, with values it gets from the `Joystick` class:

- The first argument is `xSpeed`
  - Robot: `xSpeed` is the speed along the robot's X axis, which is forward/backward.
  - Joystick: The driver sets forward/backward speed by rotating the joystick along its Y axis, which is pushing the joystick forward/backward.
  - Code: Moving the joystick forward is negative Y rotation, whereas robot forward is along the positive X axis. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.
- The second argument is `ySpeed`
  - Robot: `ySpeed` is the speed along the robot's Y axis, which is left/right.
  - Joystick: The driver sets left/right speed by rotating the joystick along its X axis, which is pushing the joystick left/right.
  - Code: Moving the joystick to the right is positive X rotation, whereas robot right is along the negative Y axis. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.

- The third argument is `zRotation`
  - Robot: `zRotation` is the speed of rotation along the robot's Z axis, which is rotating left/right.
  - Joystick: The driver sets rotation speed by twisting the joystick along its Z axis, which is twisting the joystick left/right.
  - Code: Twisting the joystick to the right is positive Z rotation, whereas robot rotation is CCW positive. This means the joystick value needs to be inverted by placing a - (minus sign) in front of the value.

Swerve drivetrain example

Like mecanum drivetrains, swerve drivetrains are holonomic and have the ability to move side-to-side. Joystick control can be handled the same way for all holonomic drivetrains, but WPILib doesn't have a built-in robot drive class for swerve. Swerve coding is described in other sections of this documentation, but an example of using joystick input to set `ChassisSpeeds` values is included below. Consider the same common arcade drive scheme described in the mecanum section above. The scheme uses a single joystick where the driver pushes the joystick forward/backward for forward/backward robot movement, pushes the joystick left/right to move side-to-side, and twists the joystick to rotate the robot.

JAVA

```
// Drive using the X, Y, and Z axes of the joystick.
var speeds = new ChassisSpeeds(-m_stick.getY(), -m_stick.getX(), -m_stick.getZ());
```

C++

```
// Drive using the X, Y, and Z axes of the joystick.
frc::ChassisSpeeds speeds{-m_stick.GetY(), -m_stick.GetX(), -m_stick.GetZ()};
```

PYTHON

```
# Drive using the X, Y, and Z axes of the joystick.
speeds = ChassisSpeeds(-self.stick.getY(), -self.stick.getX(), -self.stick.getZ())
```

The three arguments to the `ChassisSpeeds` constructor are the same as `driveCartesian` in the mecanum section above; `xSpeed`, `ySpeed`, and `zRotation`. See the description of the arguments, and their joystick input in the section above.

17.5.5 Robot drive kinematics

Kinematics is a topic that is covered in a different section, but it's worth discussing here in relation to the coordinate system. It is critically important that kinematics is configured using the coordinate system described above. Kinematics is a common starting point for coordinate system errors that then cascade to basic drivetrain control, field oriented driving, pose estimation, and path planning.

When you construct a `SwerveDriveKinematics` or `MecanumDriveKinematics` object, you specify a translation from the center of your robot to each wheel. These translations use the coordinate system above, with the origin in the center of your robot.

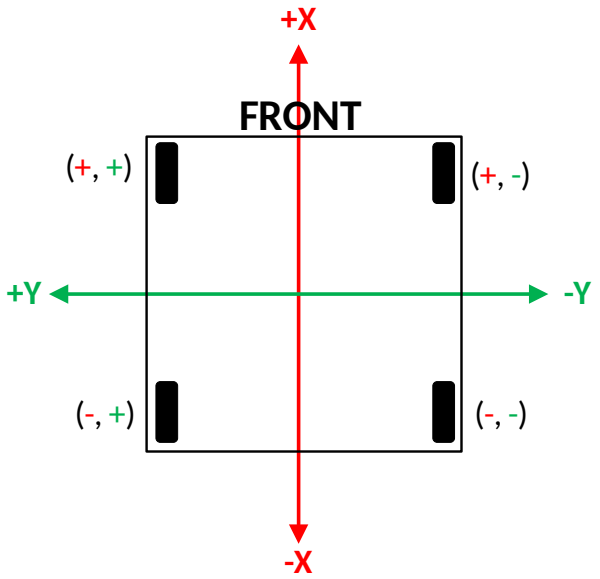


图 5: Kinematics with translation signs

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')

警告: 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.

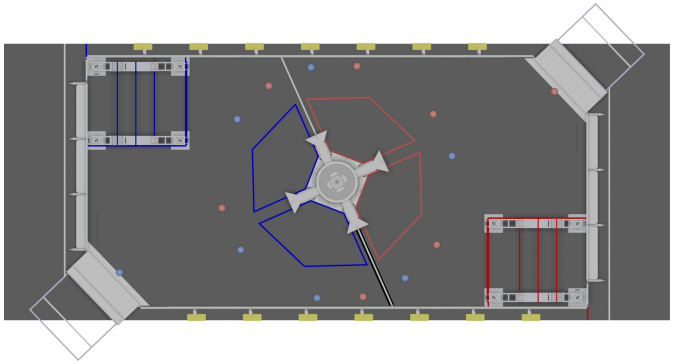


图 6: Rotated field from RAPID REACT in 2022[1]

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

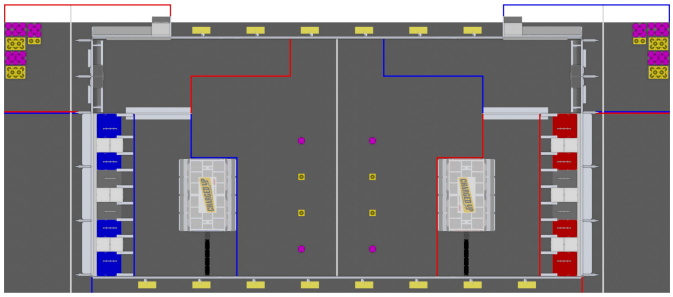


图 7: Mirrored field from CHARGED UP in 2023[2]

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.

警告: 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.

备注: 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 (操作标签)*.

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.

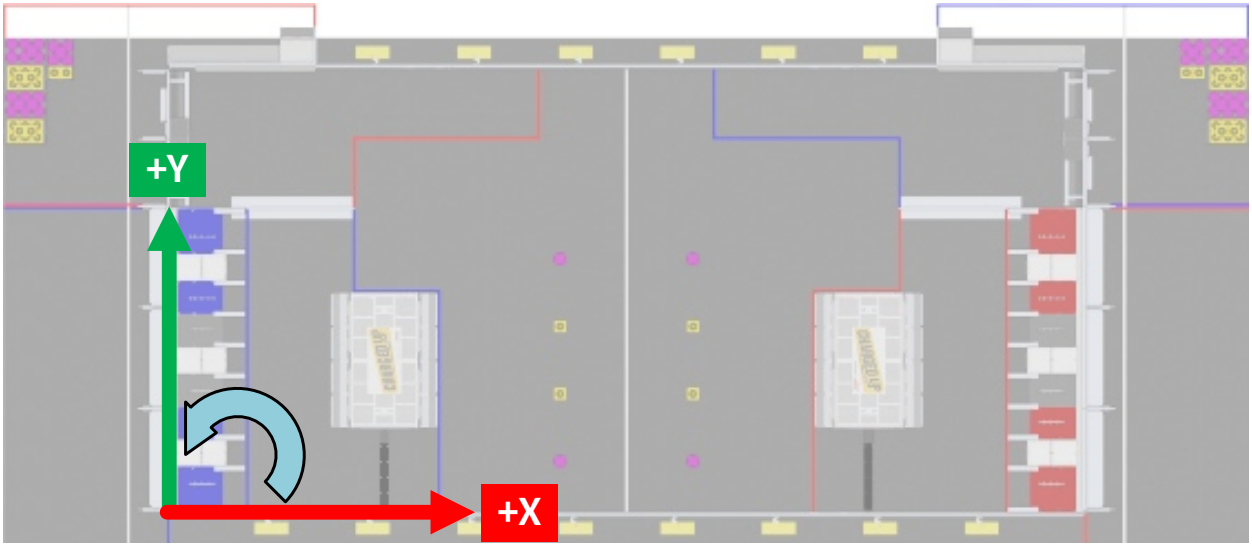


图 8: CHARGED UP with blue origin

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
```

(续下页)

(接上页)

```
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:

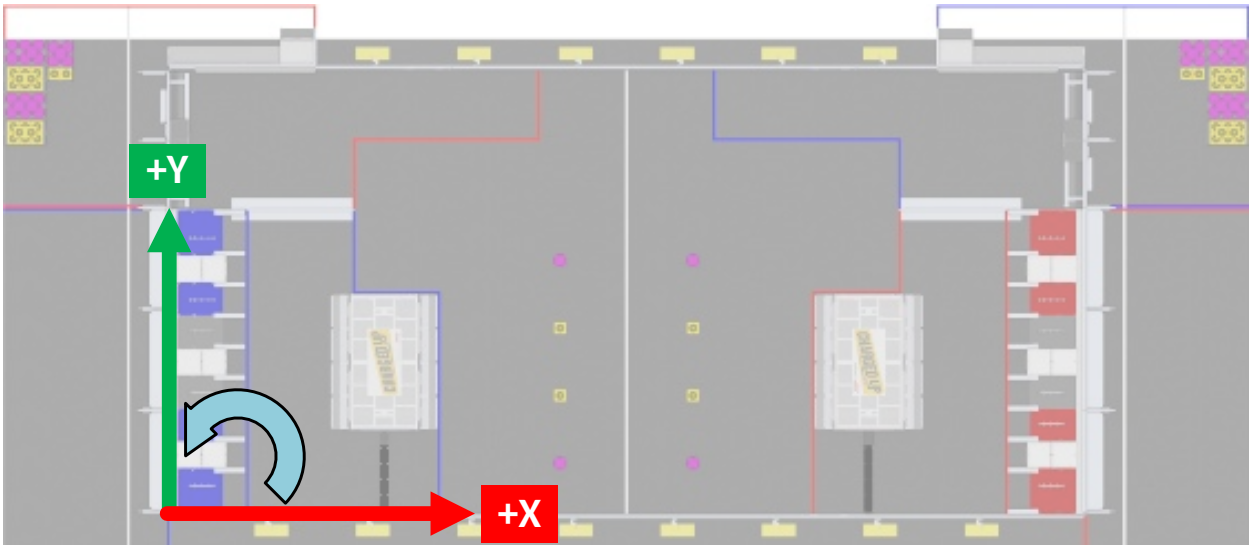


图 9: CHARGED UP field with blue alliance as origin

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.

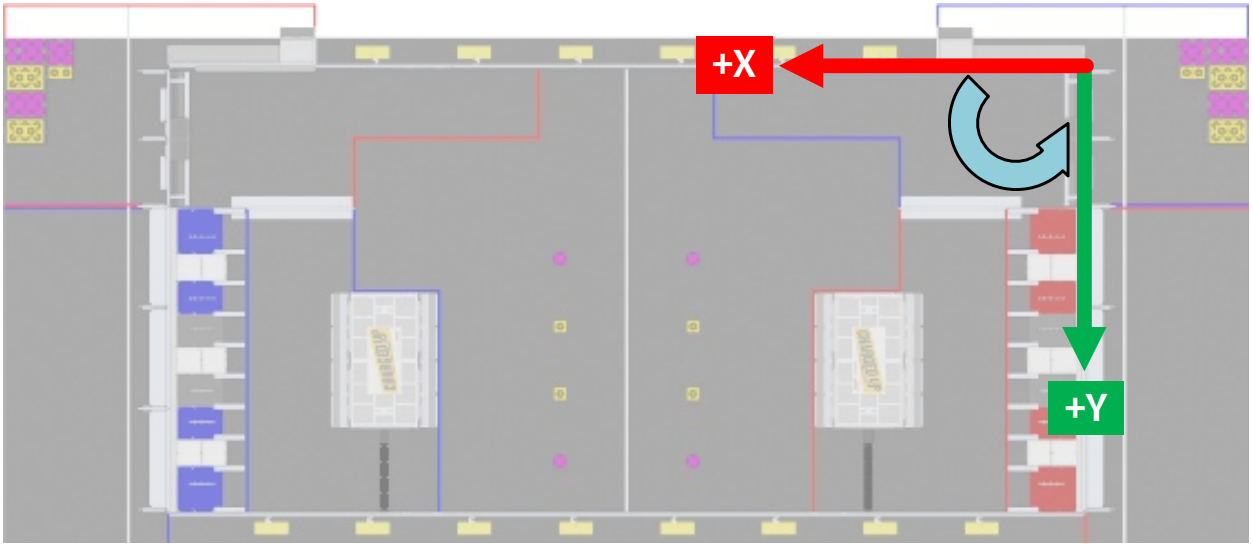


图 10: CHARGED UP field with red alliance as origin

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
```

(续下页)

(接上页)

```
↪'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

```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

```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();
    }
  }
```

```java
  /** 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++

```cpp
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});
}
```

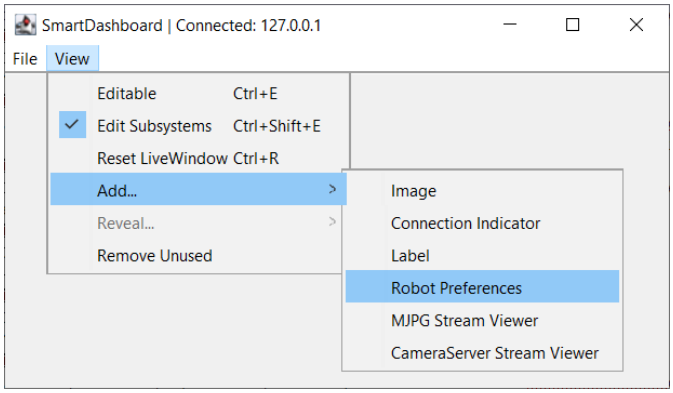
Python

```
    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)
```

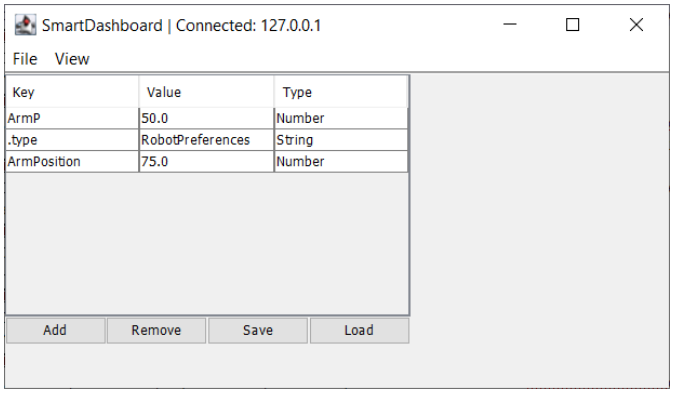
17.6.3 Using Preferences in SmartDashboard

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.

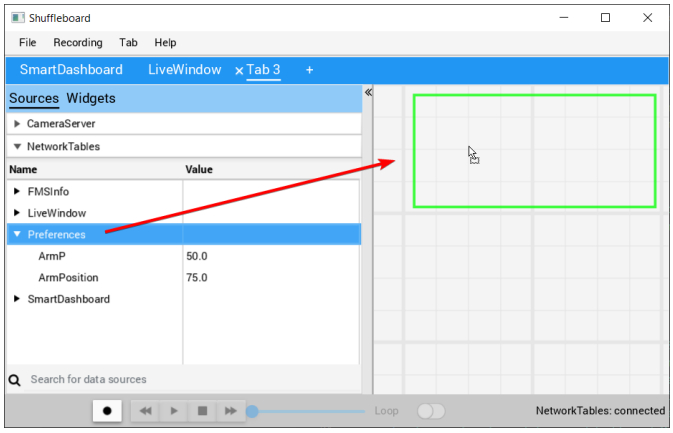
Editing Preferences in SmartDashboard



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.

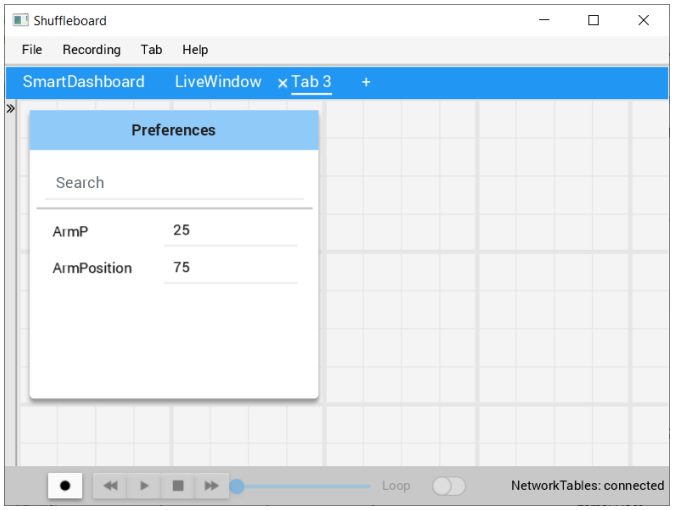
17.6.4 Using Preferences in Shuffleboard

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

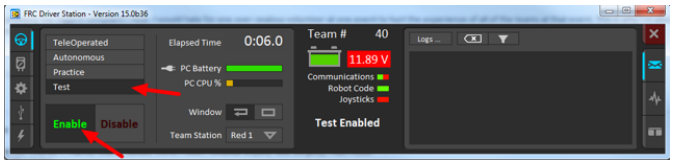


The values are shown here with the default values from the code. If the values need to be adjusted they can be edited here.

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

重要： Since 2024, LiveWindow in Test Mode is disabled by default! See *Enabling LiveWindow in Test Mode* to enable it.

With LiveWindow, all actuator outputs can be controlled on the Dashboard and all sensor values can be seen. PID Controllers can also be tuned. The sensors and actuators are added automatically, no code is necessary. See *SmartDashboard: Test Mode and Live Window* for more details.

17.8 Reading Stacktraces

`An unexpected error has occurred.`

When your robot code hits an unexpected error, you will see this message show up in some console output (Driver Station or RioLog). You'll probably also notice your robot abruptly stop, or possibly never move. These unexpected errors are called *unhandled exceptions*.

When an unhandled exception occurs, it means that your code has one or more bugs which need to be fixed.

This article will explore some of the tools and techniques involved in finding and fixing those bugs.

17.8.1 What's a "Stack Trace"?

The `unexpected error has occurred` message is a signal that a *stack trace* has been printed out.

In Java and C++, the *call stack* data structure is used to store information about which function or method is currently being executed.

A *stack trace* prints information about what was on this stack when the unhandled exception occurred. This points you to the lines of code which were running just before the problem happened. While it doesn't always point you to the exact *root cause* of your issue, it's usually the best place to start looking.

17.8.2 What's an "Unhandled Exception"?

An unrecoverable error is any condition which arises where the processor cannot continue executing code. It almost always implies that, even though the code compiled and started running, it no longer makes sense for execution to continue.

In almost all cases, the root cause of an unhandled exception is code that isn't correctly implemented. It almost never implies that any hardware has malfunctioned.

17.8.3 So How Do I Fix My Issue?

Read the Stack Trace

To start, search above the `unexpected error has occurred` for the stack trace.

Java

In Java, it should look something like this:

```
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)
```

There's a few important things to pick out of here:

- There was an `Error`
- The error was due to an `Unhandled exception`
- The exception was a `java.lang.NullPointerException`
- The error happened while running line 24 inside of `Robot.java`
  - `robotInit` was the name of the method executing when the error happened.
- `robotInit` is a function in the `frc.robot.Robot` package (AKA, your team's code)
- `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:

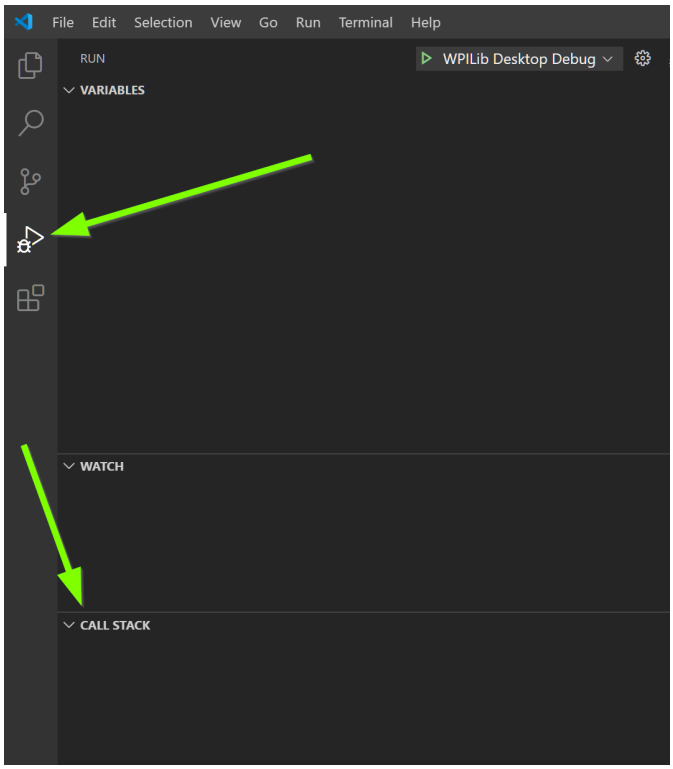
```
Error at frc.robot.Robot.buggyMethod(TooManyBugs.java:1138): Unhandled exception:␣
↪java.lang.NullPointerException
        at frc.robot.Robot.buggyMethod(TooManyBugs.java:1138)
        at frc.robot.Robot.barInit(Bar.java:21)
        at frc.robot.Robot.fooInit(Foo.java:34)
        at frc.robot.Robot.robotInit(Robot.java:24)
        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.

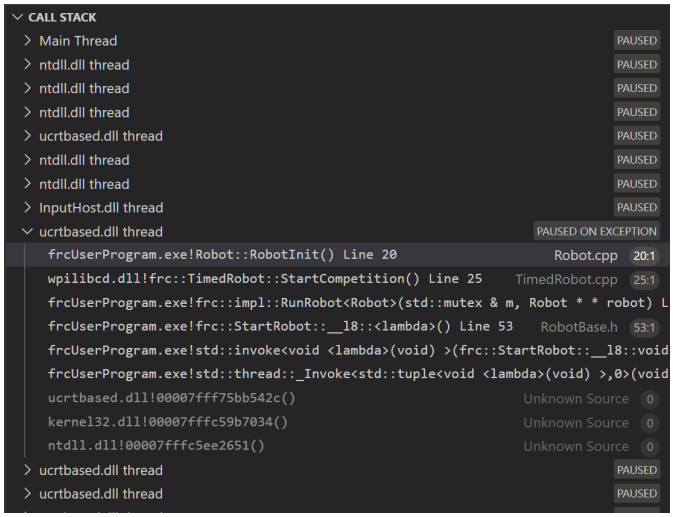
C++

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:



There's a few important things to pick out of here:

- 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.

备注: 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

```
19 PWMSparkMax armMotorCtrl;
20
21 @Override
22 public void robotInit() {
23       armMotorCtrl.setInverted(true);
24 }
```

C++

```
17 class Robot : public frc::TimedRobot {
18    public:
19       void RobotInit() override {
20          motorRef->SetInverted(false);
21       }
22
23    private:
24       frc::PWMVictorSPX m_armMotor{0};
25       frc::PWMVictorSPX* motorRef;
26 };
```

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.

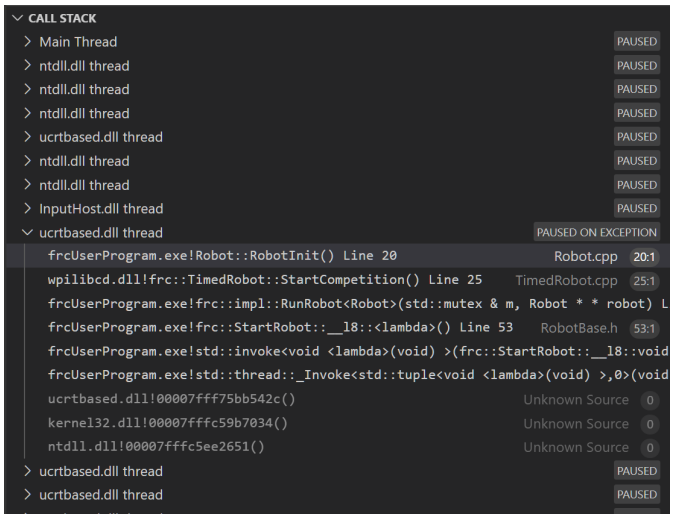
Alternatively, you can step through lines of code with the single step debugger, and stop when you hit line 23. Inspecting the `armMotorCtrl` object at that point would show that it is null.

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.

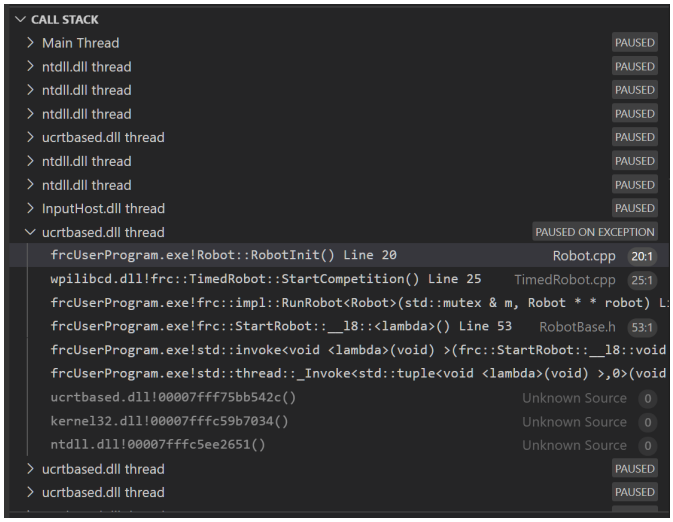
Alternatively, by running the single-step debugger and stopping on line 24, you could inspect the value of all variables to discover `shoulderToElbow_in` has a value of `0`.

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

```java
int armLengthRatio;
int elbowToWrist_in = 39;
int shoulderToElbow_in = 3;

@Override
public void robotInit() {

   armLengthRatio = elbowToWrist_in / shoulderToElbow_in;

}
```

C++

```cpp
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>
```

(续下页)

(接上页)

```
↪(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]
```

(续下页)

(接上页)

```
        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);
```

(续下页)

(接上页)

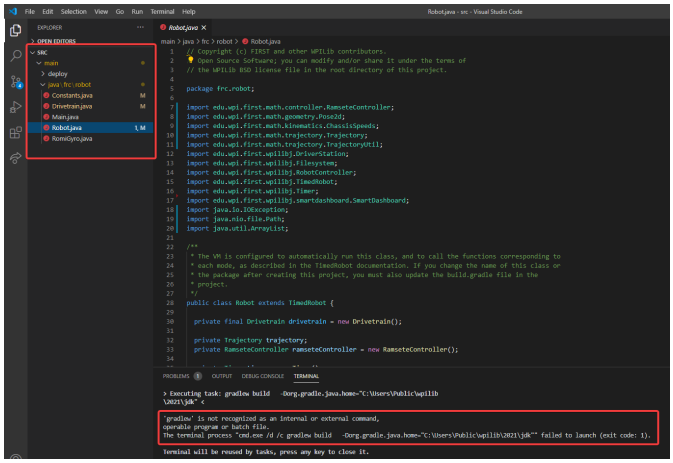
```
27
28 }
```

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):

备注: 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):

备注: 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 Get Alliance Color

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 Getting your Alliance Color and Doing an Action

JAVA

```
Optional<Alliance> ally = DriverStation.getAlliance();
if (ally.isPresent()) {
    if (ally.get() == Alliance.Red) {
        <RED ACTION>
    }
    if (ally.get() == Alliance.Blue) {
        <BLUE ACTION>
    }
}
else {
    <NO COLOR YET ACTION>
}
```

C++

```
using frc::DriverStation::Alliance;
if (auto ally = frc::DriverStation::GetAlliance()) {
    if (ally.value() == Alliance::kRed) {
        <RED ACTION>
    }
    if (ally.value() == Alliance::kBlue) {
        <BLUE ACTION>
    }
}
else {
    <NO COLOR YET ACTION>
}
```

PYTHON

```
from wpilib import DriverStation

ally = DriverStation.getAlliance()
if ally is not None:
    if ally == DriverStation.Alliance.kRed:
        <RED ACTION>
    elif ally == DriverStation.Alliance.kBlue:
        <BLUE ACTION>
else:
    <NO COLOR YET ACTION>
```

17.11 Java Garbage Collection

Java garbage collection is the process of automatically managing memory for Java objects. The Java Virtual Machine (JVM) is responsible for creating and destroying objects, and the garbage collector is responsible for identifying and reclaiming unused objects.

Java garbage collection is an automatic process, which means that the programmer does not need to explicitly deallocate memory. The garbage collector keeps track of which objects are in use and which are not, and it periodically reclaims unused objects.

17.11.1 Object Creation

Creating a large number of objects in Java can lead to memory and performance issues. While the Java Garbage Collector (GC) is designed to handle memory management efficiently, creating too many objects can overwhelm the GC and cause performance degradation.

Memory Concerns

When a large number of objects are created, it increases the overall memory footprint of the application. While the overhead for a single object may be insignificant, it can become substantial when multiplied by a large number of objects.

备注： *VisualVM* can be used to see where memory is allocated.

Performance Concerns

The GC's job is to periodically identify and reclaim unused objects in memory. While garbage collection is running on an FRC robot coded in Java, execution of the robot program is paused. When the GC has to collect a large number of objects, it has to pause the application to run more frequently or for longer periods of time. This is because the GC has to perform more work to collect and process each object.

GC-related performance degradation in robot programs can manifest as occasional pauses, freezes, or loop overruns as the GC works to reclaim memory.

Design Considerations

If you anticipate your application creating a large number of short-lived objects, it is important to consider design strategies to mitigate the potential memory and performance issues. Here are some strategies to consider:

- Minimize object creation: Carefully evaluate the need for each object creation. If possible, reuse existing objects or use alternative data structures, such as arrays or primitives, to avoid creating new objects.
- Efficient data structures: Use data structures that are well-suited for the type of data you are working with. For example, if you are dealing with a large number of primitive values, consider using arrays or collections specifically designed for primitives.

17.11.2 Diagnosing Out of Memory Errors with Heap Dumps

All objects in Java are retained in a section of memory called the *heap*. As objects typically consume the greatest amount of memory in a Java program, it is often useful to take a snapshot of the state of the heap—a heap dump—to analyze memory issues. Heap dumps only capture the state of a program's heap at a single point in time, so they are unlikely to be useful if not captured exactly at the time the program is experiencing memory issues.

Since `OutOfMemoryErrors` both crash the program and are a common reason to want a heap dump, the JVM can be configured to automatically take a heap dump the moment an `OutOfMemoryError` is caught by the JVM. To configure these options, locate the `frcJava` code block in your project's `build.gradle`:

```
15 deploy {
16     targets {
17         roborio(getTargetTypeClass('RoboRIO')) {
18             // Team number is loaded either from the .wpilib/wpilib_preferences.json
19             // or from command line. If not found an exception will be thrown.
```

(续下页)

(接上页)

```
            // 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*.

警告: 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.

> **小心:** 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() {
```

(续下页)

(接上页)

```
  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.

> **小心：** 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 支持资源

除了此处的文档之外，”FRC reg “还提供了多种其他资源。帮助团队了解控制系统和软件。

18.1 其他文档

除了这个网站，还有其他几个地方可以供团队查阅文档：

- “<NI FRC Community Documents Section <https://forums.ni.com/t5/FIRST-Robotics-Competition/bd-p/1014?profile.language=en&view=documents>”
- “<FIRST Inspires Technical Resources Page <https://www.firstinspires.org/resource-library?flagged=All&combine=&field_content_type_value%5B0%5D=first_robotics_competition&field_resource_library_tags_tid=171&sort_by=created_1>”
- CTRE Software & Resources Page
- REV Robotics Documentation

18.2 论坛

还是有疑问？有问题无法被文档内的内容解答？一下论坛将为你提供专业的官方解读：

- NI FRC Support Forum (roboRIO, LabVIEW and Driver Station software questions, as well as roboRIO repairs)
- “<FIRST Inspires Control System Forum <https://forums.firstinspires.org/forum/general-discussions/first-programs/first-robotics-competition/competition-discussion/control-system?f=1338>”(走线，硬件和 Driver Station 相关的问题)
- “<FIRST Inspires Programming Forum <https://forums.firstinspires.org/forum/general-discussions/first-programs/first-robotics-competition/competition-discussion/programming-aa>”(与 C++，Java，或 LabVIEW 相关的程序问题)

18.3 CTRE 资源

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 REV 机器人技术支持

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 错误报告

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 FRC 词汇表

加速度计
一种用于测量一个或多个轴上加速度的通用传感器。

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.

自动阶段
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.

不推荐使用
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.

场地管理系统
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
一种将机器人代码部署到 roboRIO 的机制。

陀螺仪
一种测量转速的设备。它可以将旋转测量值相加以确定机器人的航向。（简称“陀螺仪”）

指向
机器人指向的方向，通常以度数表示。

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.

国际货币联盟
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.

OP
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).

KOP 机箱
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.

模拟
: 团队在没有可用实际机器人的情况下测试其代码的一种方法。

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.

手动阶段
每次比赛的第二阶段称为"远程操作时段"("遥控"),由控制机器人的驱动程序组成。

topic
In *NetworkTables*, a named data channel.

torque
A force applied at a distance from some axis of rotation

弹道
轨迹是一条平滑的曲线,在曲线的每个点上都有速度和加速度,连接了场上的两个端点。

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

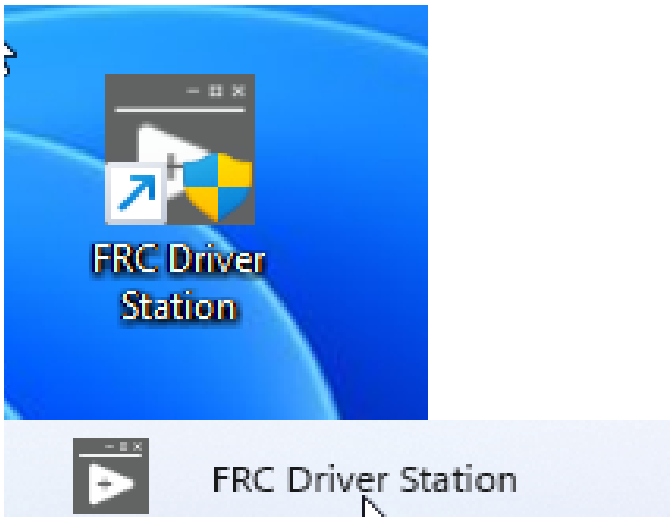
机器操控台

20.1 FRC Driver Station (由 NI LabVIEW 提供支持)

这篇文章讲的是 FRC ® Driver Station (由 NI LabVIEW 提供支持) 的使用和特性。

关于安装 Driver Station 软件的信息，请参见本文档：5.4 Installing the FRC Game Tools

20.1.1 启动 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.

备注: By default the FRC Driver Station launches the *LabVIEW Dashboard*. It can also be configured on 设置栏 to launch the other Dashboards: *SmartDashboard* and *Shuffleboard*. WPILib must be *installed* to use SmartDashboard and Shuffleboard.

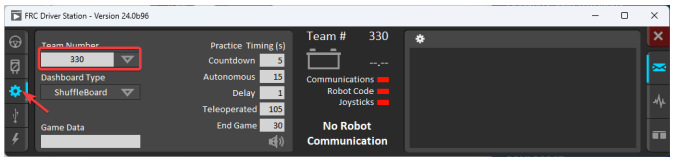
20.1.2 Driver Station 快捷键

- `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.

备注： 无论 Driver Station 是否有键盘焦点，空格键都会紧急停止机器人

警告： 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.

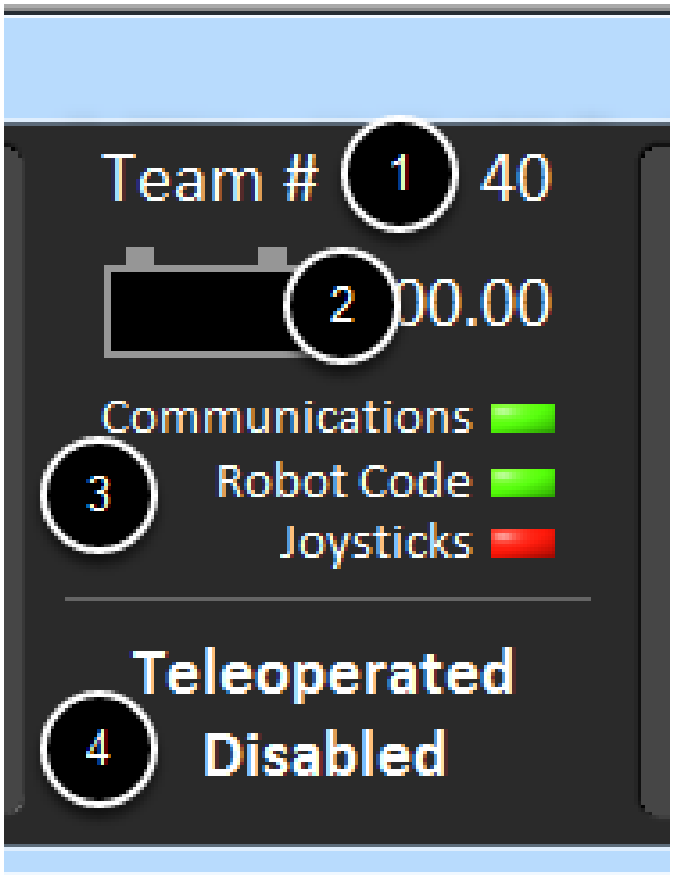
20.1.3 设置 Driver Station



Driver Station 应该被设置成你的队伍号，才能连上你的机器人。单击 Setup 按钮，然后在“队伍号”一栏里面输入你的队伍号。按回车或者单击窗口外面来完成设置。

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*.

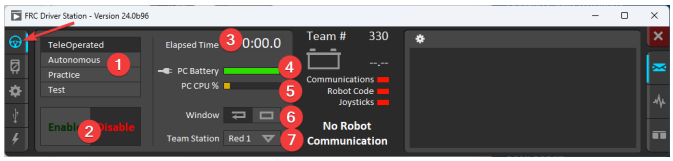
20.1.4 状态栏



Driver Station 的状态栏位于显示屏的中央，并且无论选择哪个选项，它将始终可见。它将显示有关 Driver Station 和机器人状态的重要信息：

1. Team # - 是 Driver Station 当前配置的队号。这应该匹配您的 FRC 队号。要更改 Driver Station 中的队号，请参阅“设置选项”。
2. Battery Voltage （电池电压）- 如果 Driver Station 已连接至 roboRio 并与其通信，则会在电池图标中以数字形式显示当前电池电压，并显示一小段电压-时间图表。当 roboRIO 掉电情况发生时，数字指示器的背景将变为红色。有关更多信息，请参见本文档：29.4 roboRIO Brownout and Understanding Current Draw
3. Maor Status Indicator (主要状态指示器) - 这三个指示器显示 Driver Station 的主要状态项目。“Communication” 指示 Driver Station 当前是否正在与 roboRIO 上的 FRC 网络通信任务进行通信（通信被分为 TCP 和 UDP 通信）。“The Robot Code” 指示器显示机器人代码当前是否正在运行（是否运行由机器人代码中的 Driver Station Task 是否更新电池电压来确定）。“Joysticks” 指示器显示是否至少插入了一个操纵杆，并被 Driver Station 识别。
4. Status String - The Status String provides an overall status message indicating the state of the robot. Some examples are “No Robot Communication”, “No Robot Code”, “Emergency Stopped”, and “Teleoperated Enabled”. When the roboRIO brownout is triggered this will display “Voltage Brownout”.

20.1.5 Operation Tab (操作标签)

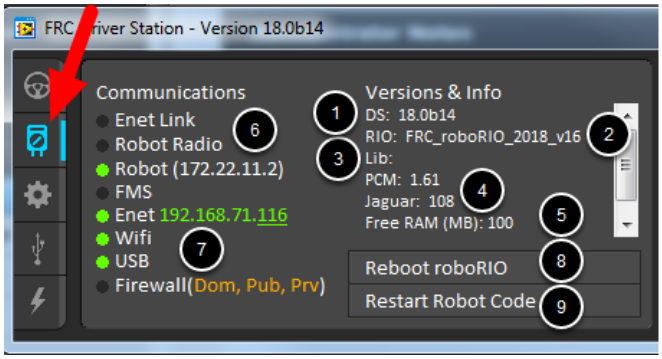


“Operation Tab” 用于控制机器人的操作模式，并在机器人运行时提供其他的关键状态指示。

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-未连接到 FMS 时，向机器人传输 Team Station 设置

备注： 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.

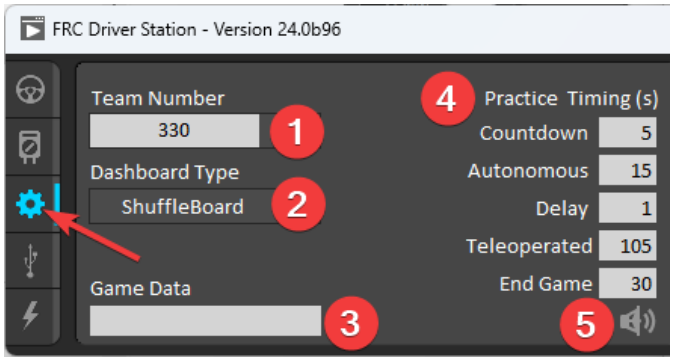
20.1.6 Diagnostics Tab (诊断标签)



Dianostics Tab 包含额外的状态指示器，团队可以使用这些指示器来诊断其机器人的问题：

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 (连接指示器) - 指示器的上半部分会显示多个组件的连接状态
   - “Enet Link” (局域网连接) 会指示电脑是否连接上了 ethernet 网线接口
   - “Robot Radio” (机器人信号) 会指示在 10.XX.YY.1 机器人无线桥接的延迟 ping 值
   - “Robot” 会指示在使用 mDNS 时 (后面带有静态 10.TE.AM.2 地址) roboRIO 的延迟 ping 值
   - “FMS” (场控) 会指示 Driver Station 是否从 FMS 收到数据包 (这不是个 ping 指示器)
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” (以太网) 会指示所检测到的 Ethenet adapter 的 IP 地址
   - “WiFi” 会指示是否已经启用无线网卡
   - “USB” 会指示是否检测到 roboRIO 已经使用 USB 连接
   - “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 设置栏

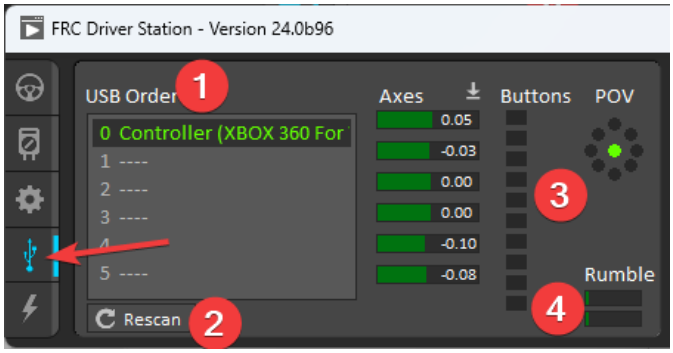


设置栏包含一系列按钮，可以用来控置 Driver Station 的操作

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 (练习模式时序) - 这些窗口制练习模式序列各部分的时序。在练习模式下启用机器人后，Driver Station 会自动执行从上到下的指示。
5. Audio Control (音频控制) - 此按钮控制使用练习模式时是否发出音频。

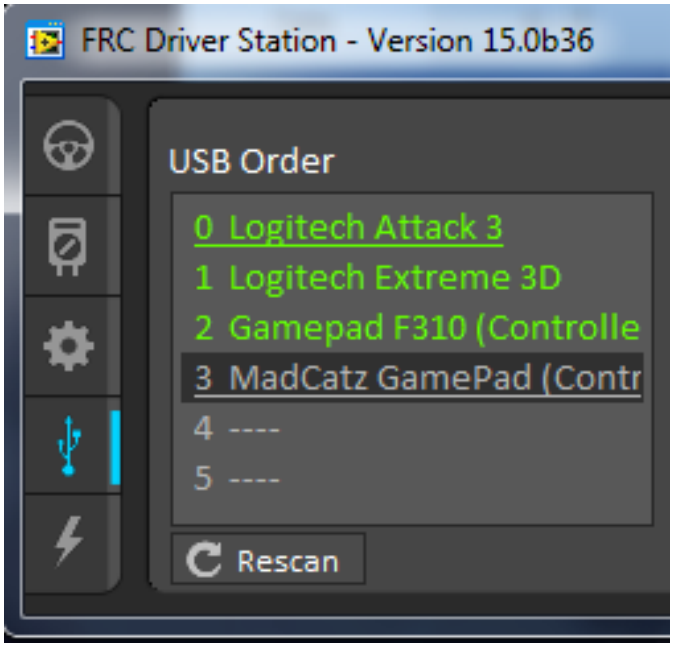
20.1.8 USB 设备栏



USB 设备栏包括有关连接到 Driver Station 的 USB 设备的信息

1. USB Setup List (USB 设置列表) - 这包含连接到 Driver Station 的所有兼容 USB 设备的列表。按下设备上的按钮将以绿色突出显示名称，并在设备名称前加 2 * s
2. *Rescan* - This button will force a Rescan of the USB devices. While the robot is disabled, the DS will automatically scan for new devices and add them to the list. To force a complete re-scan or to re-scan while the robot is Enabled (such as when connected to FMS during a match) press F1 or use this button.
3. Device indicators (设备指示器) - 这些指示器显示 Joystick 手柄的轴、按钮和 POV 的状态
4. Rumble (蜂鸣器) - 对于 X 输入设备 (例如 Xbox 控制器) 会出现蜂鸣器控制。这可以用于测试设备的蜂鸣功能。最上面一栏是 "Right Rumble" (右蜂鸣器)，最下面一栏是 "Left Rumpble" (左蜂鸣器)。单击并按住任何蜂鸣器栏将启动部分蜂鸣器 (左侧是 rumble = 0，右侧是 rumbelr = 1)。这只能用来控制，不会指示机器人代码里的蜂鸣器的值。

Re-Arranging and Locking Devices (重新排序和锁定设备)



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.

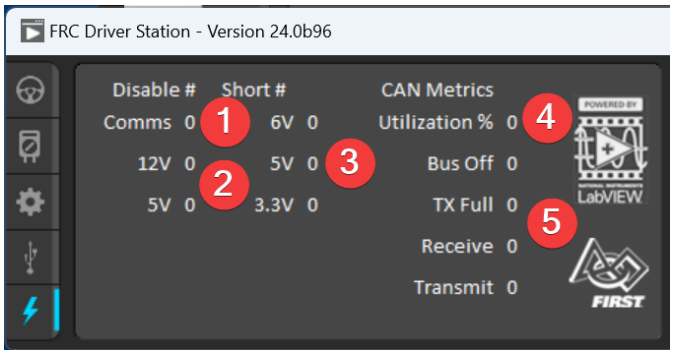
备注: If you have two or more of the same device, they should maintain their position as long as all devices remain plugged into the computer in the same ports they were locked in. If you switch the ports of two identical devices the lock should follow the port, not the device. If you re-arrange the ports (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.

例如：上面的图片有 4 个设备

- 一个锁定的" Logitech Attack 3" 手柄。这个设备会待在这各位，除非被拽到了别的插槽或者解锁
- 一个解锁的 "Logitech Extreme 3D" 手柄
- 一个解锁的 "Gamepad F310 (Controller)" 是一个 Logitech F310 gamepad
- 一个锁定但断开的" MadCatz GamePad (Controller)"，这是一个 MadCatz Xbox 360 Controller。

在此示例中，拔下 Logitech Extreme 3D 游戏杆将导致 F310 游戏手柄向上移动到插槽 1。插入 MadCatz 游戏手柄（即使已卸下插槽 1 和 2 中的设备并且这些插槽为空）也会导致其占用插槽 3。

20.1.9 CAN/Power 选项

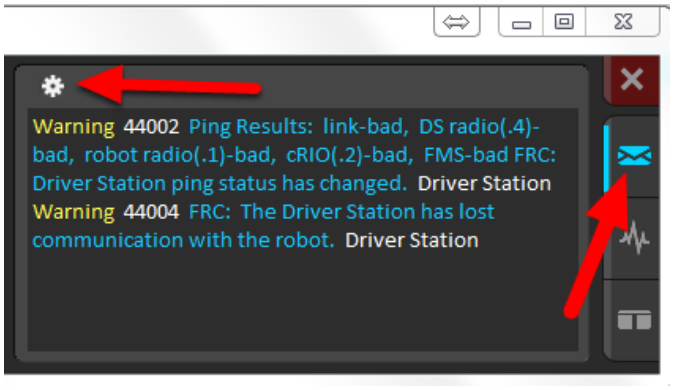


Driver Station 左边一栏最后一个选项是 CAN/Robot Powe 选项。这个选项包含 roboRIO 供电信息和 CAN 线的状态。

1. Comms Faults (Comms 故障) - 显示 Driver Station 连接后出现了多少个 Comms 故障
2. 12V Faults (12V 故障) - 显示 Driver Station 连接后出现多少个输入电流故障 (电压过低)。
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 (CAN Bus 使用率) - 指示 CAN 线的占用率
5. CAN faults (CAN 线故障) - 指示 Driver Station 连接之后出现的 4 种 CAN 线错误的数量

如果检测到故障，这一栏的指示器 (上图显示的是蓝色) 会变红。

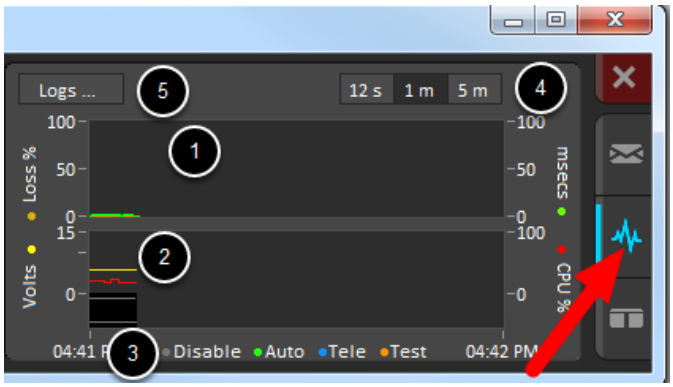
20.1.10 信息栏



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 图标栏



图标栏会绘制并显示更高级的机器人状态指示器，来帮助队伍诊断机器人的问题：

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. 底部图形以黄色（相对于左侧的轴）绘制电池电压，roboRIO CPU 以红色（相对于右侧的轴）绘制，DS Requested 模式为图表底部的实线，而 Robot 模式为它上面不连续线。
3. 此按键在底图中显示用于“DS 请求”和“机器人报告”模式的颜色。
4. Chart scale - These controls change the time scale of the DS Charts.
5. This button launches the *DS Log File Viewer*.

DS Requested 模式是 Driver Station 命令机器人进入的模式。Robot Reported 模式是基于每种语言的编码框架中包含的报告方法实际运行的代码。

20.1.12 双窗口栏

The last tab on the right side is the Both tab which displays Messages and Charts side by side.

20.2 Driver Station 葵花宝典

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.

是否想确保 Driver Station 不是您在 FIRST 机器人大赛（FRC）比赛中的拦路虎？在停止搭建日与竞争之间的这段时间内，搭建和配置可靠的 Driver Station 笔记本电脑是一个简单的项目。继续阅读以查找许多团队在数千场比赛中吸取的教训。

20.2.1 出发参加比赛之前

1. 专门将笔记本电脑仅用作 Driver Station。许多团队都这样做。一台专用机器可让您管理目标的配置–准备好在现场比赛。“专用”表示除安装或正在运行的 FRC 提供的 Driver Station 软件和关联的 Dashboard 外，没有其他软件。
2. 在您的 Driver Station 使用商务级笔记本电脑。为什么？它们比 Best Buy 买的 $ 300 黑色星期五特别款更加耐用。他们将在比赛碰撞中存活下来。商业级笔记本电脑具有更高质量的设备驱动程序，并且与消费类笔记本电脑相比，驱动程序的维护时间更长。这使您的投资持续更长时间。联想 ThinkPad T 系列和 Dell Latitude 是您在竞赛中经常看到的两个受欢迎的商务级品牌。eBay 上每天都有成千上万的待售商品。rookie kit 中提供的笔记本电脑是入门级机器。团队通常会升级到更大的显示器，因为他们通过视觉和 Dashboard 来做更多的事情。
3. 考虑使用二手笔记本电脑而不是新笔记本电脑。FRC ® Driver Station 和 Dashboard 软件使用的系统资源很少，因此您不用购买新的笔记本电脑 - 而是购买便宜的 4-5 岁的二手笔记本电脑。您甚至可能会获得通过您所在地区的二手电脑商店的捐赠。

备注： 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. 笔记本推荐性能
   a. RAM –8GB of RAM or greater
   b. 显示尺寸为 13 英寸或更大，最低分辨率为 1440x1050。
   c. 接口
      i. 内置以太网端口是首选。确保它是标准端口。铰接式以太网端口不能承受重复使用。
      ii. 使用以太网端口保护程序进行以太网连接。这样可以延长笔记本电脑上端口的寿命。如果您有一台带有铰接式以太网端口的消费级笔记本电脑，这尤其重要。
      iii. 如果您的笔记本电脑上的以太网端口不可靠，请更换笔记本电脑（推荐过的）或从知名品牌处购买 USB 以太网加密狗。许多团队发现 USB 以太网不如内置以太网可靠，这主要是由于廉价的硬件和不良的驱动程序。在 KOP 中，给新手使用的加密狗因运作良好而享有盛誉。
      iv. 最少 2 个 USB 端口
   d. 键盘。很难在现场对仅触屏笔记本进行故障排除。
   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. 比赛前一周安装所有 Windows 更新。这使您有时间确保更新不会干扰驱动程序站功能。为此，请打开 Windows Update 设置页面，然后查看您是否是最新的。如果没有，请安装挂起的更新。重新启动并再次检查以确保您是最新的。
6. 更改 Windows 更新的“更新时间”，以防止在比赛时间安装更新。导航到“开始” -> “设置” -> “更新和安全性” -> “Windows Update”，然后选择“更改更新时间”。如果您要参加比赛，请考虑时区差异。这将有助于确保您的驱动程序站不会由于在现场安装更新而重启或失败。
7. 删除任何 3rd 方防病毒或反恶意软件。相反，请在 Windows 10 或 11 上使用 Windows Defender。由于您只是为了 Windows 和 FRC 软件更新而连接到 Internet，因此风险很低。仅在您的机器操控台上安装驾驶所需的软件。您的目标是消除可能干扰正常操作的变量。删除机器附带的所有不需要的

预装软件（“膨胀软件”）。在活动前一天晚上回到酒店时，不要将笔记本电脑用作 Steam 机器来玩游戏。许多团队甚至拥有单独的编程笔记本电脑。

8. 避免学校 IT 部门管理的 Windows 10 或 11 安装。这些部署是为学校环境而建立的，并且经常伴随着干扰机器人操作的不必要的软件。
9. 笔记本电池/电源
   a. 在你的电源计划中，在“用电池”和“接通电源”中关闭“使计算机进入睡眠”
   b. 关闭 USB 选择性暂停：
      i. 右键单击工具栏的电池图标，然后选择“电源选项”。
      ii. 电源计划中“更改计划设置”
      iii. 点击“更改高级电源设置”
      iv. 在“高级电源设置”中向下滚动，并分别禁用在“使用电池”和“插入电源”情况下“USB 选择性暂停”
   c. 进行上面的更改后，请确保笔记本电脑电池至少可以续航一个小时。这样，机器人和 drive team 就可以在没有电源的情况下有足够的时间穿过队列到达联盟站。
10. 带上可靠的 USB 转以太网线以用于连接到 roboRIO。
11. 增加保持力/应力释放，以防止游戏杆/游戏手柄控制器掉落在地板上和/扯到 USB 端口。这有助于防止控制器连接出现间歇性问题。
12. 您的 Windows 用户账户必须是 Administrator 账户。

20.2.2 赛场时

1. Turn off Windows firewall using *these instructions*.
2. 使用专用的硬件 Wi-Fi 开关或在控制面板中将无线网卡禁用，以关闭 Wi-Fi。
3. 在 pit 区给 driver station 充电
4. 删除登录密码或确保 drive team 团队中的每个人都知道密码。您会惊奇地发现操作手不知道笔记本电脑的密码就经常到达现场。
5. 使用 LabView 教程中的说明，确保将 LabView 代码永久部署并设置为“启动时运行”。如果每次打开机器人时都必须部署代码，那说明您出大问题。
6. 将网站浏览限制为 FRC 相关的网站。这样可以最大程度地减少比赛中感染恶意软件的机会。
7. 不要计划使用互联网进行软件更新。场地中可能没有 Wi-Fi，并且酒店的 Wi-Fi 的质量差异很大。如果确实需要更新，请与 pit 区中的 Control System Advisor 联系。

20.2.3 每场比赛之前

1. 请确保笔记本电脑在你上一场之前保持打开和登录
2. 比赛时，关闭比赛中不需要的程序，例如 Visual Studio Code 或 LabView。
3. 将笔记本电脑充电器带到现场。每个操作区都为您提供电源。
4. 用魔术贴胶带将笔记本电脑固定在操作站桌子上。您永远不会知道您的联盟合作伙伴何时会遇到自主编程问题并毁于一旦。
5. 确保将操纵杆和控制器插进正确的 USB 端口。

a. 在 FRC Driver Station 软件的 USB 选项中，拖放以根据需要分配操纵杆。

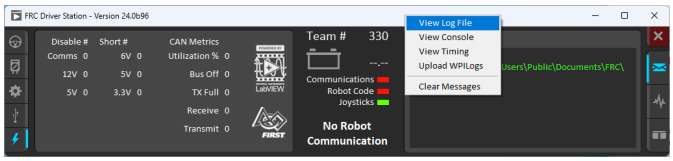
b. 如果操纵杆/控制器未显示为绿色，请使用重新扫描按钮（F1）

c. 如果在比赛期间拔下操纵杆或控制器并重新插回电源或者变成灰色，那么使用重新扫描按钮（F1）。

6. Ensure your *Dashboard is connected to the robot* after your driver station connects to the robot.

20.3 Driver Station 日志文件查看器

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.



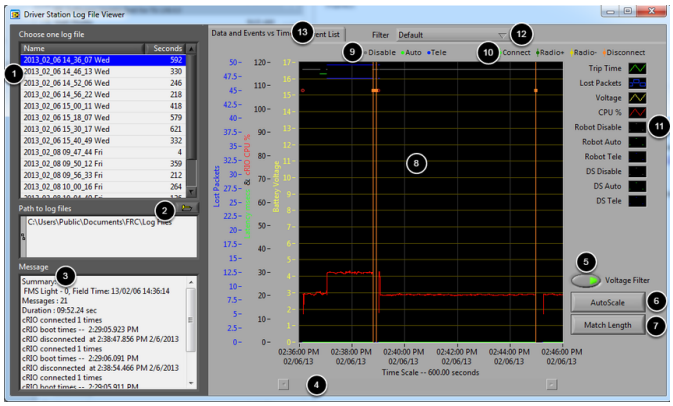
备注： Several alternative tools exist that provide similar functionality to the FRC Driver Station Log Viewer. *AdvantageScope* is an option included in WPILib, and DSLOG Reader is a third-party option. Note that WPILib offers no support for third-party projects.

20.3.1 活动日志

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`.

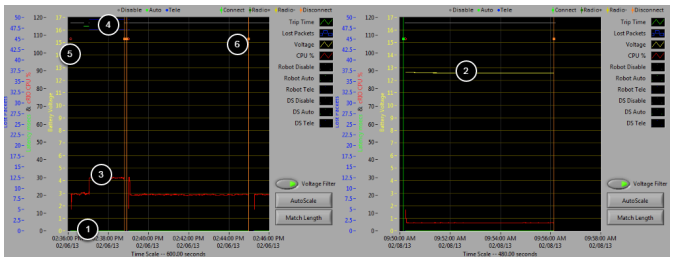
20.3.2 日志浏览器用户界面



日志查看器包含许多控件和显示栏，以帮助分析 Driver Station 日志文件：

1. 文件选择框-此窗口显示当前所选文件夹中的所有可用日志文件。单击列表中的日志文件以将其选中。
2. 日志文件的路径-此框显示查看器正在寻找日志文件的当前文件夹。这默认为 Driver Station 在其中存储日志文件的文件夹。单击文件夹图标浏览到其他位置。
3. 消息框-此框显示事件日志中所有消息的摘要。将鼠标悬停在图表上的事件上时，此框将更改为显示该事件的信息。
4. 滚动条-放大图表时，该滚动条允许水平滚动图形。
5. 电压筛选器-此控件可打开和关闭电压筛选（默认为打开）。当未收到电池电压时（表示 DS 未与 roboRIO 通信），电压筛选会过滤掉 CPU 占用率，机械模式和往返时延等数据。
6. 自动缩放-此按钮将图表缩小以显示日志中的所有数据。
7. 比赛时长-此按钮将图表缩放到大约 FRC 比赛时长（显示 2 分钟 30 秒）。它不会自动定位比赛开始的地方，您将不得不使用滚动条滚动以定位自动模式的开始。
8. 图表-此显示显示来自 DS 日志文件的图表数据（电压，往返时延，roboRIO 的 CPU 占用率，丢失的数据包和机械模式），以及叠加的事件数据（在图表上显示为点，在选定事件中显示为跨越整个事件的竖线）。将鼠标悬停在图表上的事件标记上，屏幕左下方的“消息”窗口中会显示有关事件的信息。
9. 机械模式键-屏幕顶部显示的机械模式键
10. 主要事件键-主要事件的键，在图表上显示为竖线
11. 图形键 - 图表数据的按键
12. 筛选控制-下拉菜单选择筛选的模式（下面会说明筛选模式）
13. 选项控制 - 在显示图表（数据与事件关于时间）和事件列表之间切换的控件

20.3.3 使用图形显示

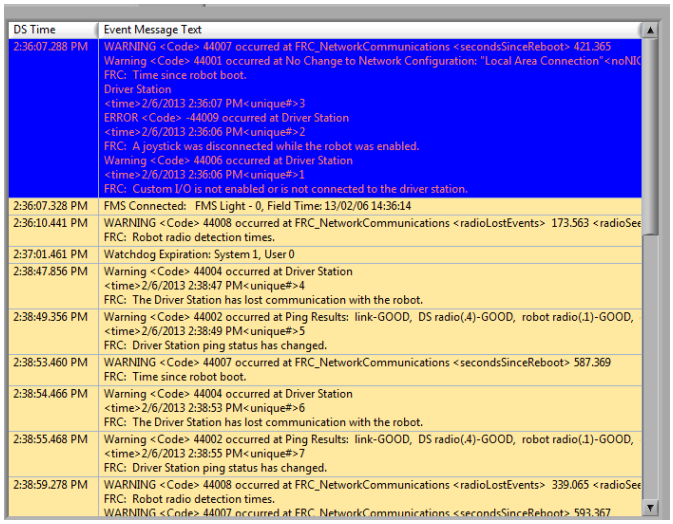


图表显示包含下面的信息：

1. 以毫秒为单位的跳闸时间图（绿线）和每秒丢失的数据包数（显示为蓝色竖线）。在这些示例图像中，“往返时延”在图的底部是一条平坦的绿线，并且没有丢失的数据包
2. 电池电压显示为一条黄线。
3. roboRIO CPU 占用率是一条红线
4. 机械模式和 DS 模式的图形。显示屏的顶部显示了由机器操控台控制的模式。底部显示的是机器人代码报告的模式。在此示例中，机器人在禁用和自动阶段下不会报告其模式，而会在手动阶段期间报告。
5. 事件标记将显示在图表上，指示事件发生的时间。错误将以红色显示；警告将显示为黄色。将鼠标悬停在事件标记上，将在屏幕左下方的“消息”框中显示有关事件的信息。
6. 主要事件显示为穿过显示界面的竖线。

单击并拖动目标区域，来放大图表的一部分。你可以只放大时间线，但是不能垂直放大。

20.3.4 事件列表



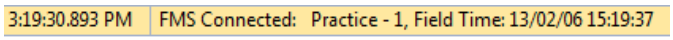
“事件列表”选项显示由 Driver Station 记录的事件（警告和错误）列表。显示的事件和详细信息由当前活动的晒选确定（图像显示“所有事件，所有信息”过滤器处于活动状态）。

20.3.5 筛选

日志查看器里三个筛选器现在可用：

1. 默认：此筛选器过滤掉由 Driver Station 产生的许多错误和警告。该筛选器对于识别由机器人上的代码引发的错误很有用。
2. “所有事件和时间”：此筛选器显示所有事件及其发生的时间
3. “所有事件，所有信息”：此筛选器显示所有事件和所有记录的信息。此时，此过滤器与“所有事件和时间”之间的主要区别在于，此选项显示首次出现特定消息的“唯一”指示符。

20.3.6 根据比赛识别日志



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.

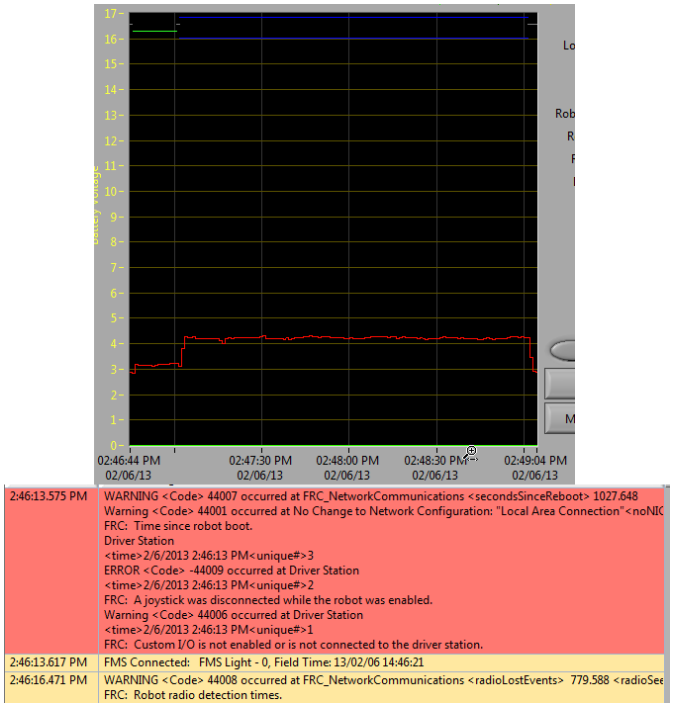
20.3.7 使用日志查看器识别常见的连接失败

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.

备注： 请注意，已使用“匹配长度”按钮将本节中显示的所有日志文件缩放为匹配长度，然后滚动到自主模式的开头。另外，许多日志都不包含电池电压信息，用于日志捕获的平台未正确接线以报告电池电压。

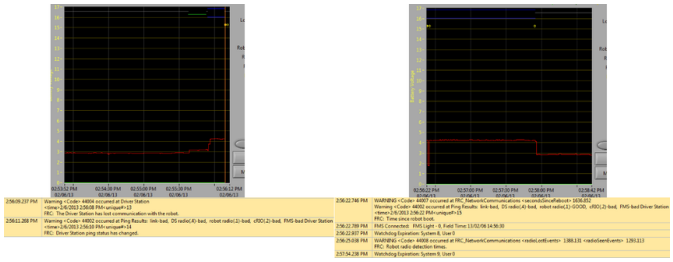
小技巧： 下面显示了在日志查看器中找到的一些错误消息，并且在 driver-station-errors-warnings 文章中进行了详细说明。

“Normal“日志



这是正常比赛日志的示例。第一个框中包含的错误和警告来自 Driver Station 首次启动时的时间，可以忽略。看到这些事件在 “FMS Connected：” 事件之前发生就能确认。所显示的最后一个事件也可以忽略，它也是从机器人第一次连接到 Driver Station 以来（它发生在连接 FMS 后 3 秒），发生在比赛开始前约 30 秒。

从场控系统断连

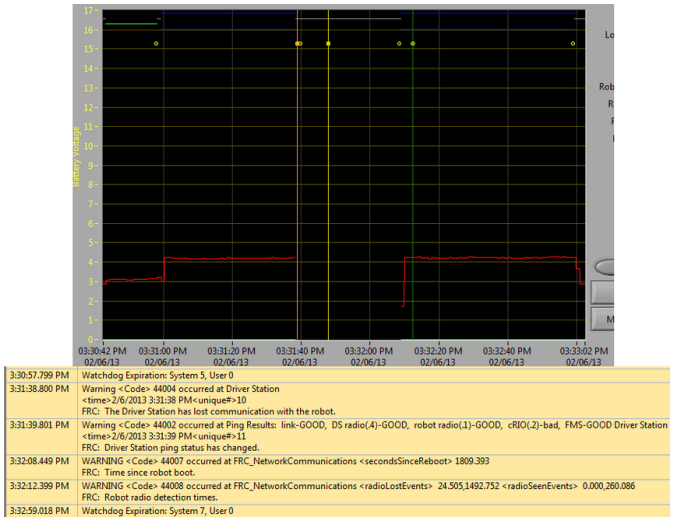


当 DS 与 FMS 断开连接，并与机器人断开连接时，在比赛期间，它可能会将日志分段。连接失败的关键指标是第一个日志的最后一个事件，表明与 FMS 的连接现在处于 “不良” 状态，第二个日志中的第二个事件是一条新的 FMS 连接消息，随后 DS 立即转换为 Teleop Enabled。此类故障的最常见原因是以太网线没有插好或者 DS 电脑上的以太网端口损坏。

roboRIO 重启

“robot 启动后的时间”消息是 roboRIO 重新启动导致连接失败的主要指示。在此日志中，DS 如第一个事件所示，在 3:01:36 失去了与 roboRIO 的连接。第二个事件表明，连接失败后启动的 ping 操作对 roboRIO 以外的所有设备均成功。在 3:01:47，roboRIO 再次开始对 ping 做出响应，另外一个 ping 在 3:01:52 失败。在 3:02:02，Driver Station 连接到 roboRIO，并且 roboRIO 报告它已启动 3.682 秒。这清楚表明 roboRIO 已重新启动。代码继续加载，并在 3:02:24 时报告与相机通信的错误。还有一条警告，指示在代码完成启动之前，没有机器人代码正在运行。

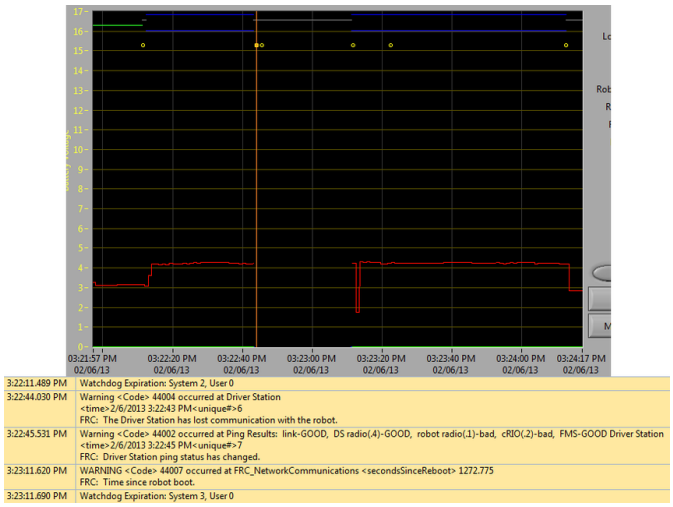
机器人上以太网线有误



机械人上的以太网电缆出现问题，主要是通过 roboRIO 的 ping 延迟变高以及 roboRIO 重新连接时出现的 Radio Lost 和 Radio Seen 事件表示。roboRIO 重新连接时的“自机械手启动后的时间”消息还将指示 roboRIO 尚未重新启动。在此示例中，机器人以太网电缆在 3:31:38 断开连接。ping 状态指示无线电仍处于连接状态。当机器人在 3:32:08 重新连接时，“自机器人启动以来的提示”为 1809 秒，表明

roboRIO 显然没有重新启动。在 3:32:12，机器人指示它在 24.505 秒前丢失了无线电并在 0.000 秒前返回。这些点在图表上绘制为竖线，黄色表示无线电丢失，绿色表示可见无线电。请注意，时间与实际事件略有偏差，如通过断开连接和连接所示，但有助于提供有关正在发生的事件的其他信息。

路由器重启



机器人路由器的重新启动通常特点为 40-45 秒与无线电的连接断开。在此示例中，无线电在 3:22:44 短暂断电，导致无线电开始重新启动。3:22:45 的事件表明对 ping 无线电失败。DS 在 3:23:11 重新恢复与 roboRIO 的通信，并且 roboRIO 指示其已启动 1272.775 秒，从而排除了 roboRIO 重新启动的情况。请注意，路由器上的网络开关会很快恢复，因此瞬时断电可能不会导致“无线电丢失”或“看到无线电”两个事件。较长的干扰可能导致 DS 记录路由器事件。在这种情况下，指向路由器重新启动的区别因素是路由器与 DS 的 ping 状态。如果路由器重置，则路由器将无法连接。如果问题是机器人上的网线或连接问题，则路由器 ping 应该保持“良好”状态。

20.4 Driver Station 报错/警告

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 没插上的操作柄

```
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.
```

拔下操纵杆时会触发此错误。与消息文本相反，即使未启用机械手，甚至未将其连接到 DS，也会打印此错误。即使操纵杆已正确连接并正常工作，每次启动 Driver Station 时，您都会看到此消息的单个实例。

备注: Joystick Unplugged warnings can be silenced by calling `DriverStation.silenceJoystickConnectionWarning(true)` (Java, C++)

20.4.2 丢失通信

```
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.
```

每当 Driver Station 失去与机器人的通信（通信指示灯从绿色变为红色）时，就会打印此警告消息。在建立通信之前，DS 启动时将打印此消息的单个实例。

20.4.3 ping 状态

```
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 自机器人启动后时间

```
WARNING<Code>44007 occurred at FRC_NetworkCommunications
**<secondsSinceReboot> 3.585**
FRC: Time since robot boot.
```

每次 DS 开始与 roboRIO 通信时，都会打印此消息。该消息指示 roboRIO 的正常运行时间（以秒为单位），可用于确定是否由于 roboRIO 重新启动而导致通信丢失。

20.4.5 信号检测时间

```
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.
```

当 DS 开始与 roboRIO 通信时，可能会打印此消息，并指示自上次丢失和看到无线电以来的时间（以秒为单位）。在消息上方的第一个示例图像中，该消息指示 roboRIO 与无线电的连接在消息被打印之前 19 秒钟

已丢失，并且在消息被打印时就再次看到了无线电。如果自启动 roboRIO 以来发生了多个 radioLost 或 radioSeen 事件，则将包括每种类型的最多 2 个事件，并以逗号分隔。

20.4.6 没有代码

```
Warning<Code>44003 occurred at Driver Station
<time>2/8/2013 9:50:13 AM<unique#>8
FRC: No robot code is currently running.
```

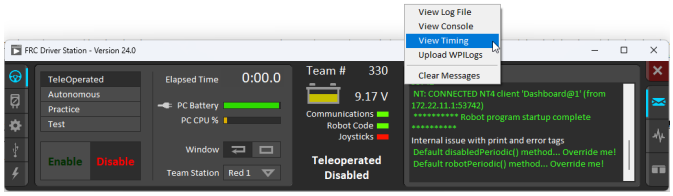
DS 开始与 roboRIO 通信但未检测到运行中的机械人代码时，将显示此消息。如果在启动 roboRIO 时 Driver Station 已打开并正在运行，则将打印此消息的 1 个实例，因为 DS 将在机器人代码完成加载之前开始与 roboRIO 通信。

20.5 Driver Station Timing Viewer

The 2024 Driver Station has a a new window to help diagnose robot control issues.

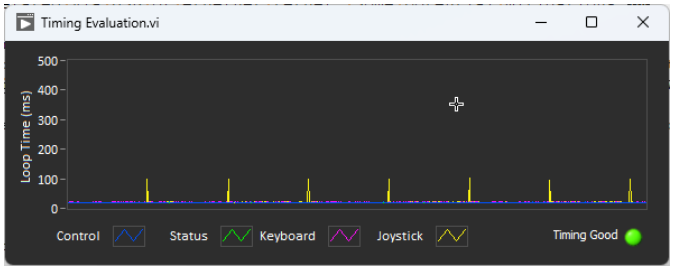
20.5.1 Opening the Timing Viewer

To start the Driver Station Timing Viewer, select the gear icon and then select *View Timing*.

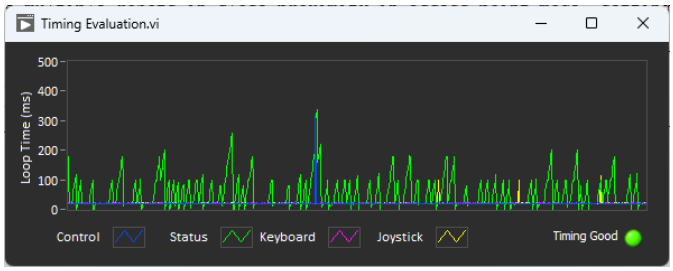


20.5.2 Viewing Timing

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 给季后赛场控系统的路由器编程

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 前提条件

Install the FRC® Radio Configuration Utility software per the instructions in *Programming your radio*

在开始使用该软件之前：

1. 禁用计算机上的 WiFi 连接，因为这可能会阻止配置使用与网桥正确通信
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.

程序配置

运行时，“路由器配置实用程序”会将许多配置设置编程到无线电中。这些设置适用于所有模式下的路由器（包括在事件时）。这些包括：

- Set a static IP of `10.TE.AM.1`
- Set an alternate IP on the wired side of `192.168.1.1` for future programming
- 桥接有线端口，以便可以互换使用
- The LED configuration noted in the status light referenced below.
- 4Mb/s bandwidth limit on the outbound side of the wireless interface (may be disabled for home use)
- QoS rules for internal packet prioritization (affects internal buffer and which packets to discard if bandwidth limit is reached). These rules are:
  - Robot Control and Status (UDP `1110`, `1115`, `1150`)
  - Robot TCP & *NetworkTables* (TCP `1735`, `1740`)

  - Bulk (All other traffic). (disabled if BW limit is disabled)
- *DHCP* server enabled. Serves out:
  - `10.TE.AM.11` - `10.TE.AM.111` on the wired side
  - `10.TE.AM.138` - `10.TE.AM.237` on the wireless side
  - Subnet mask of `255.255.255.0`
  - Broadcast address `10.TE.AM.255`
- DNS server enabled. DNS server IP and domain suffix (`.lan`) are served as part of the DHCP.

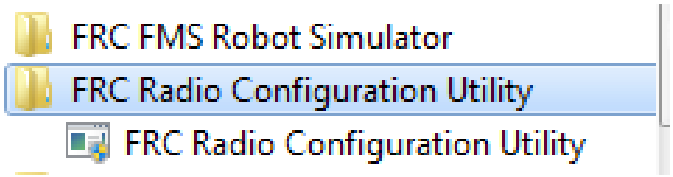
小技巧: 有关配置后路由器状态灯的详细信息，请参见“状态灯参考”：34.3.3 OpenMesh Radio

使用“路由器配置-实用程序”的团队版本进行编程时，用户帐户将保留（或设置为）固件-默认“仅适用于DAP”：

- 用户名：root
- 密码：root

备注: 不建议手动修改配置

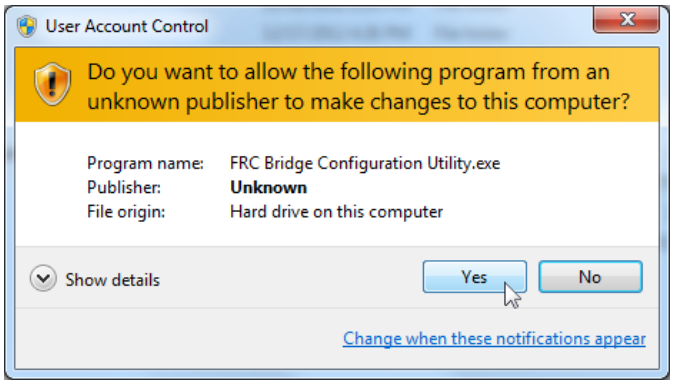
20.6.2 启动软件



使用“开始”菜单或桌面快捷方式启动程序。

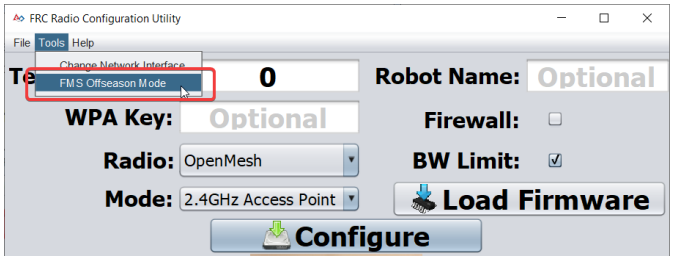
备注: If you need to locate the program, it is installed to `C:\Program Files (x86)\FRC Radio Configuration Utility`. For 32-bit machines the path is `C:\Program Files\FRC Radio Configuration Utility`

20.6.3 如果出现提示，请允许程序进行更改



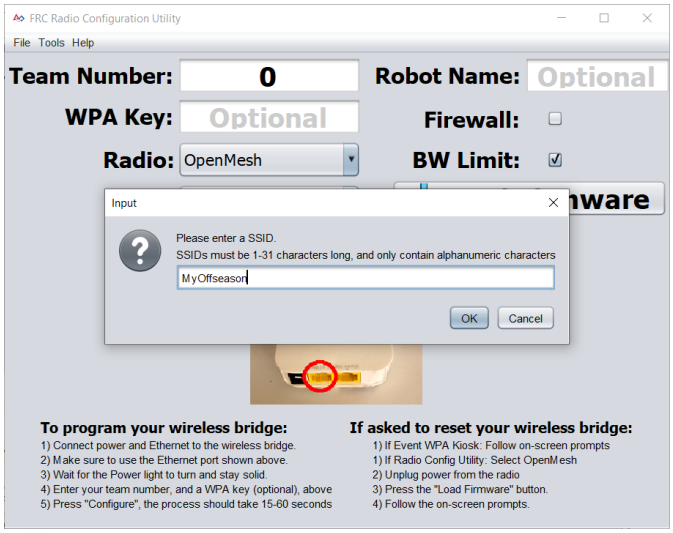
A prompt may appear about allowing the configuration utility to make changes to the computer. Click *Yes* if the prompt appears.

20.6.4 Enter FMS Offseason Mode



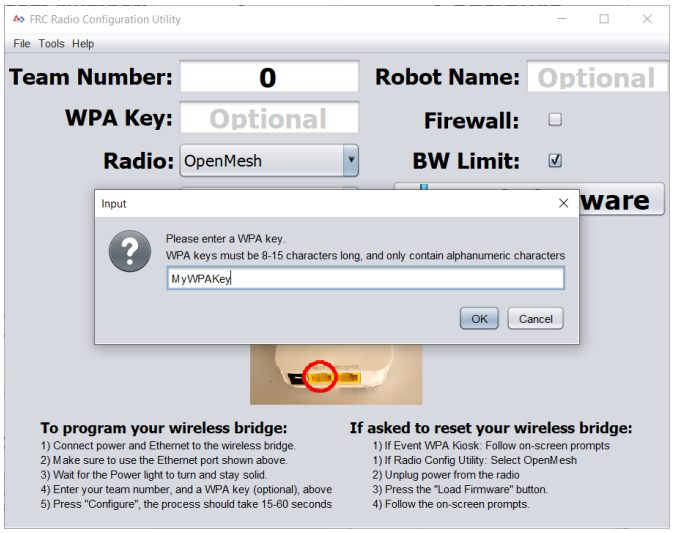
单击“工具’’ -> “FMS-Lite 模式’’以进入 FMS-Lite 模式。

20.6.5 输入 SSID



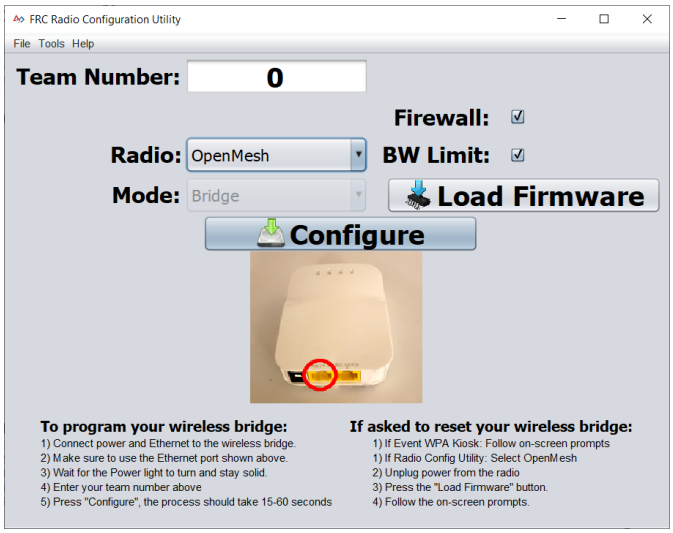
在框中输入您的无线网络的 SSID（名称），然后单击“确定”。

20.6.6 输入 WPA 密钥



在框中输入网络的 WPA 密钥，然后单击“确定”。如果使用的是不安全的网络，请将该框保留为空白。

20.6.7 路由器程序



现在，Kiosk 已准备好对任意数量的路由器进行编程，以连接到所输入的网络。要为每个路由器编程，请将路由器连接到 Kiosk，在框中设置队伍编号，然后单击配置。

通过从“路由 qi1”下拉列表中选择适当的选项，Kiosk 将对 OpenMesh，D-Link Rev A 或 D-Link Rev B 无线电进行编程，使其能够在季后赛场控网络上工作。

备注： 在此模式下，不会在 D-Link 路由器上配置带宽限制和 QoS。

20.6.8 更改 SSID 或密钥

如果您输入的内容有误或需要更改 SSID 或 WPA 密钥，请转到 “工具” 菜单，然后单击 “FMS-Lite 模式” 以使 Kiosk 退出 FMS-Lite 模式。当您再次单击以使 Kiosk 处于 FMS-Lite 模式时，将再次提示您输入 SSID 和密钥。

20.6.9 Troubleshooting

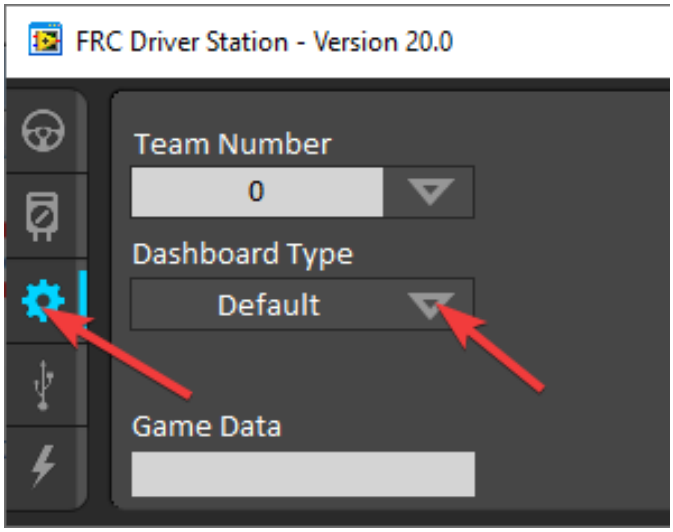
See the troubleshooting steps in *Programming your radio*

20.7 手动设置操控站以启动自定义仪表盘

备注: 如果 WPILib 没有安装到默认位置 (例如当文件手动复制到 PC 时)，选择的仪表盘可能不能正确启动。要让 DS 在启动时启动自定义指示板，必须手动修改默认指示板的设置。

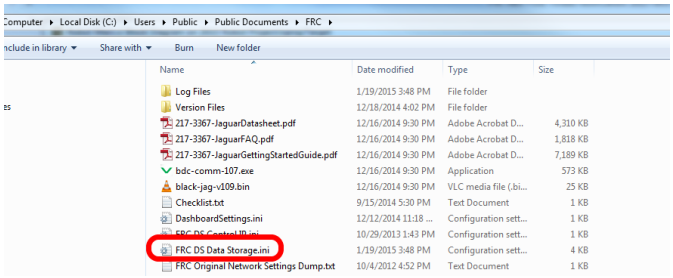
警告: 大多数安装都不需要这样，尝试首先使用适当的:ref:*Dashboard Type setting <docs/software/driverstation/driver-station:Setup Tab>* 用于你的语言。

20.7.1 将操控站设置为默认



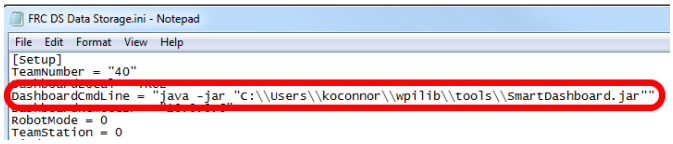
打开操控站软件，点击设置选项卡，将仪表盘设置为默认值。**然后关闭操控站!**

20.7.2 打开 DS 数据存储文件



浏览文件夹 `C:\Users\Public\Documents\FRC`，双击 `FRC DS Data Storage` 来打开它.

20.7.3 DashboardCmdLine



定位以“DashboardCmdLine“开头的行。修改它，使其指向仪表板，以便在操控站启动时启动

虚拟仪器自定义仪表板

将 = 后面的字符串替换为 `"C:\\PATH\\TO\\DASHBOARD.exe"` 其中指定的路径是到仪表板 `exe` 文件的路径。保存 `FRC DS Data Storage` 文件。

java 仪表盘

使用 `java -jar "C:\\PATH\\TO\\DASHBOARD.jar"` 替换 = 后的字符串，其中指定的路径是仪表板 `jar` 文件的路径。保存 `FRC DS Data Storage` 文件。

小技巧: Shuffleboard 和 Smartdashboard 要求 Java 11

WPILib 安装程序中的仪表板

用 `wscript "C:\\Users\\Public\\wpilib\\YYYY\\tools\\DASHBOARD.vbs"` 替换 = 后的字符串。`YYYY` 是年份，`DASHBOARD.vbs` 是 `Shuffleboard.vbs` 或 `Smartdashboard.vbs` 。保存 `FRC DS Data Storage` 文件

20.7.4 启动 Driver Station

现在，Driver Station 每次打开时都应启动仪表板。

21

机器人制造者

21.1 机器人制造者-介绍

21.1.1 RobotBuilder 概述

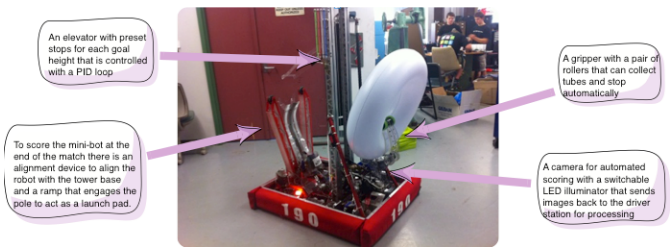
RobotBuilder 是旨在帮助机器人开发过程的应用程序。RobotBuilder 可以帮助您:

- 生成样板代码。
- 组织你的机器人并找出它的关键子系统是什么。
- 检查您的所有传感器和执行器是否有足够的通道。
- 生成接线图。
- 轻松修改您的操作员界面。
- 更多

通过遵循对于任何机器人都相同的几个步骤，使用 RobotBuilder 创建程序是非常简单的过程。本课描述了您可以执行的步骤。您可以在本文档的后续部分中找到有关每个步骤的更多详细信息。

备注: RobotBuilder 使用新的指令框架生成代码。有关新框架的更多详细信息，请参阅:ref: 基于指令的编程 *<docs/software/commandbased/index:Command-Based Programming>*。

将机器人划分为不同的子系统

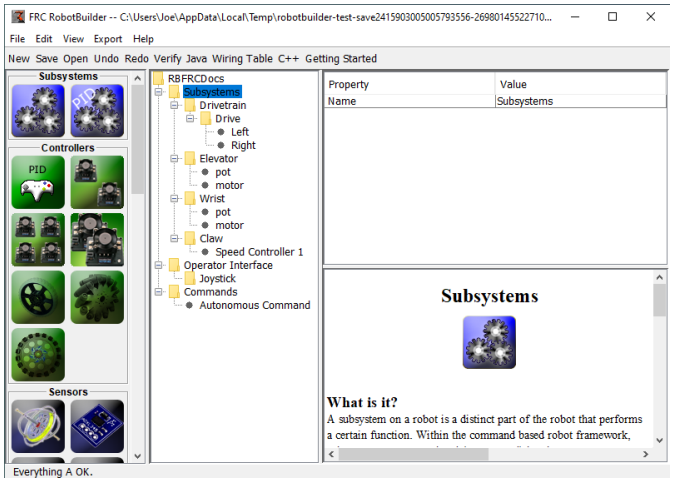


您的机器人是由许多较小的系统组成的，例如动力传动系统，手臂，射手，收集器，机械手，腕关节等。您应该研究一下机器人的设计并将其分解成较小的，可单独操作的子系统。在这个特定的例子中，有一个升降

舵，一个小型机器人对准装置，一个抓具和一个摄像系统。另外，可能包括驱动器基座。机器人的这些部分中的每一个都受到单独控制。

有关更多信息，请参阅创建子系统 <robotbuilder-creating-subsystem>。

将每个子系统添加到项目



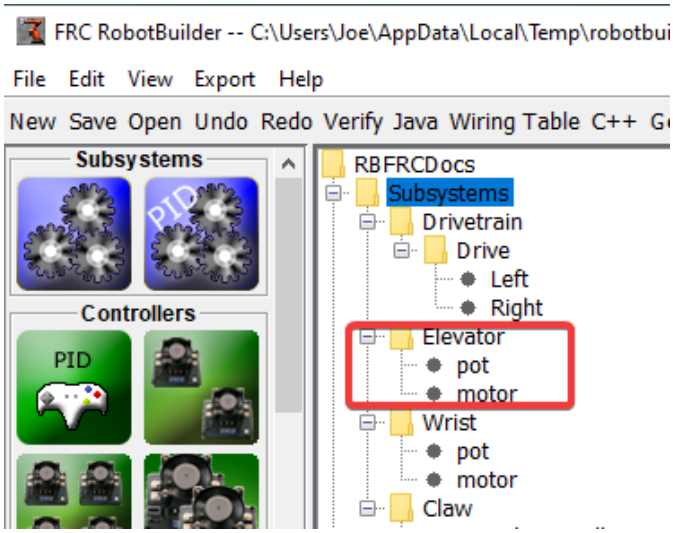
每个子系统将被添加到 RobotBuilder 的 “Subsystems” 文件夹中，并赋予一个有意义的名称。对于每个子系统，都有几个属性被填充以指定有关子系统的更多信息。另外，您可能要创建两种类型的子系统：

1. PIDSubsystems-通常希望使用 PID 控制器控制子系统的运行。这是程序中的代码，可使子系统，例如手臂角度，更快地到达所需位置，然后在到达所需位置时停止。PIDSubsystems 内置了 PID 控制器代码，通常比自己添加起来更方便。PIDSubsystems 具有确定设备何时到达目标位置的传感器和驱动至设定点的执行器（电动机控制器）。
2. 常规子系统-这些子系统没有集成的 PID 控制器，用于没有 PID 控制的子系统来进行反馈，或者用于需要比默认嵌入式 PID 控制器更复杂的控制的子系统。

当您仔细阅读本文档的更多内容时，不同类型的子系统之间的差异将变得更加明显。

有关更多信息，请参见创建子系统 <robotbuilder-creating-subsystem> 和创建子系统 <docs/software/wpilib-tools/robotbuilder/writing-code/robotbuilder-writing-subsystem-code:Writing the Code for a Subsystem> 的代码。

向每个子系统添加组件



每个子系统都包含许多执行器，传感器和控制器，用于执行其操作。这些传感器和执行器将添加到与其关联的子系统。每个传感器和执行器均来自 RobotBuilder 面板，并拖到相应的子系统中。对于每一个传感器和执行器，通常都必须设置一些特定的属性，例如端口号和特定于组件的其他参数。

在此示例中，有一个升降舵系统，该子系统使用已拖动到电梯子系统的电动机和电位计（电动机和电位计）。

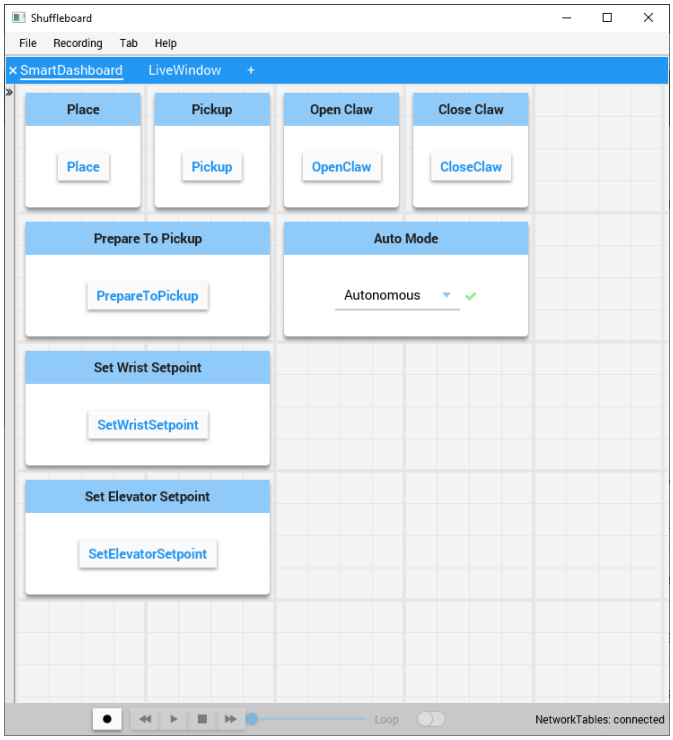
添加描述子系统目标的命令

指令是机器人将执行的不同目标。这些指令是通过拖动“命令”文件夹下的指令来添加的。创建指令时，有 7 种选择（如图左侧的调色板所示）：

- 普通指令-这是最灵活的指令，您必须编写所有代码才能执行实现目标所需的所需动作。
- 定时指令 - 这些指令是在超时后结束的指令的简化版本
- Instant commands - these commands are a simplified version of a command that runs for one iteration and then ends
- 指令组-这些指令是按顺序和并行运行的其他命令的组合。在您实现了许多基本指令之后，可以使用这些指令建立更复杂的操作。
- 设定设定值值指令-指令将 PID 子系统移动到固定的设定值或所需的位置。
- PID 指令- 这些指令有一个内置的 PID 控制器，可与常规子系统一起使用。
- 条件指令 - 这些指令选择在初始化时运行的两个指令之一。

有关更多信息，请参见创建指令 <robotbuilder-creating-command> 和编写指令代码 <docs/software/wpilib-tools/robotbuilder/writing-code/robotbuilder-writing-command-code:Writing the Code for a Command>。

测试每个命令

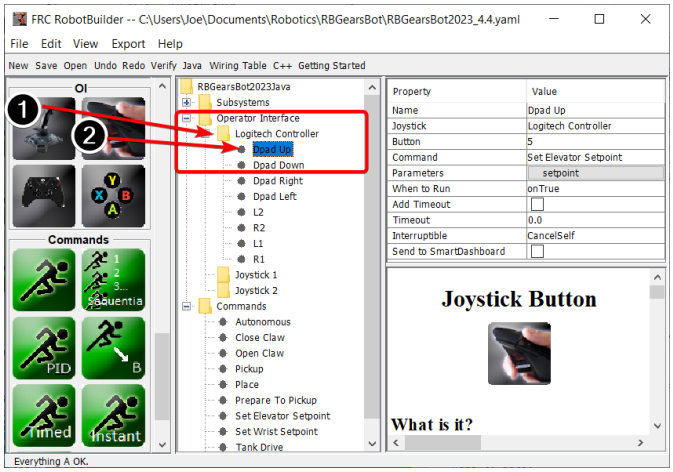


每个指令都可以从 Shuffleboard 或 SmartDashboard 运行。这对于在将指令添加到操作员界面或指令组之前测试命令非常有用。只要您选中“SmartDashboard 上的按钮”属性，就会在 SmartDashboard 上创建一个按钮。当您按下按钮时，该指令将运行，您可以检查它是否执行了所需的操作。

通过创建按钮，可以分别测试每个指令。如果所有指令可以单独运行，则可以确定机器人将可以整体运行。

有关更多信息，请参阅:doc: 使用 *Smartdashboard* 进行测试。

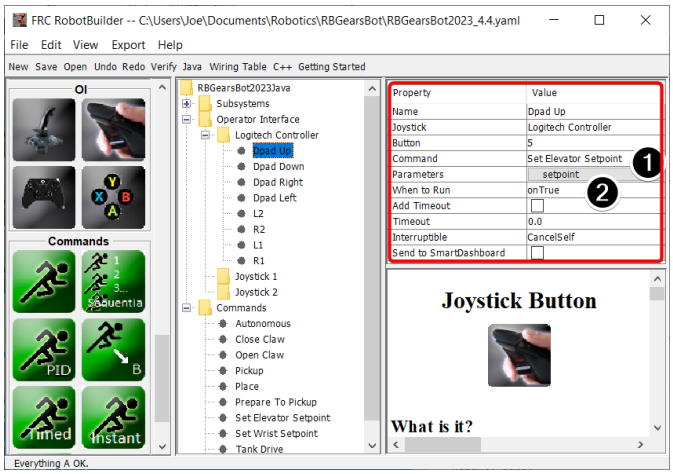
添加操作界面组件



操作界面由操纵杆，手柄和其他 HID 输入设备组成。您可以在 RobotBuilder 中的程序中添加操作界面组件（操纵杆，操纵杆按钮）。它将自动生成将初始化所有组件并将其连接到特定的指令。

将操作界面组件从面板拖到 RobotBuilder 程序中的 “Operator Interface” 文件夹中。首先（1）将操纵杆添加到程序中，然后将按钮放在关联的操纵杆下方（2），并给它们起有意义的名称，例如 ShootButton。

将指令连接到操作界面

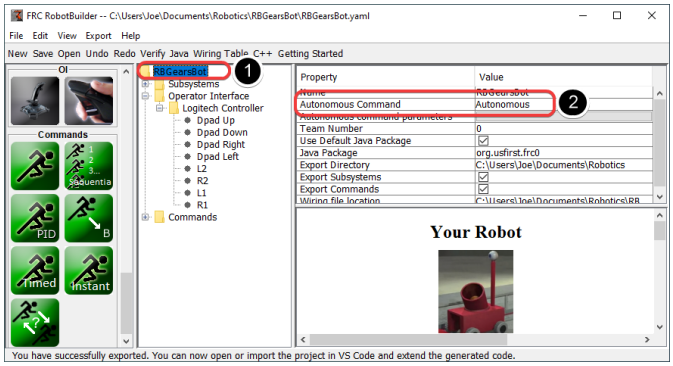


可以将指令与按钮关联，以便在按下按钮时执行指令。在大多数情况下，此方法可以处理机器人程序的大部分远程操作部分。

只需通过（1）将指令添加到 RobotBuilder 程序中的 JoystickButton 对象，然后（2）设置运行指令的条件，即可完成此操作。

有关更多信息，请参见将操作界面连接到指令 <robotbuilder-operator-interface-to-command>‘。

开发自动命令

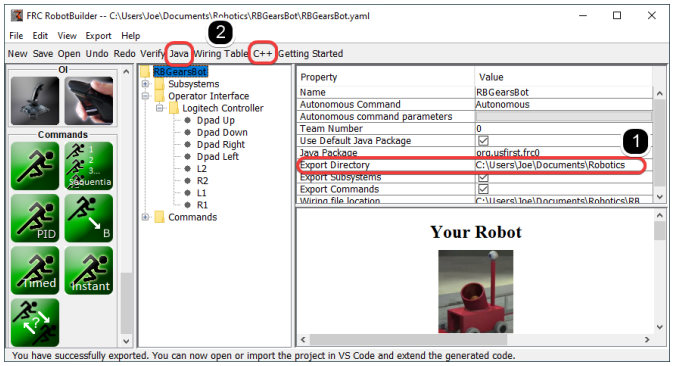


不同的指令使开发自动程序变得简单。您只需指定当机器人进入自动阶段应运行的指令，它将自动进行调度。如果您如上所述测试了指令，则只需选择要运行的指令即可。

选择 RobotBuilder 项目 (1) 根目录下的机器人，然后编辑自动阶段指令属性 (2) 以选择要运行的指令。就是这么简单！

有关更多信息，请参阅:doc: 设置自动阶段指令。

生成代码



在上述过程中的任何时候，您都可以让 RobotBuilder 生成一个 C++ 或 Java 程序来代表您创建的项目。这是通过在项目属性 (1) 中指定项目的位置，然后单击相应的工具栏按钮生成代码 (2) 来完成的。

有关更多信息，请参阅: Generating RobotBuilder Code "<docs/software/wpilib-tools/robotbuilder/writing-code/robotbuilder-generating-code:Generating Code for a Project>"

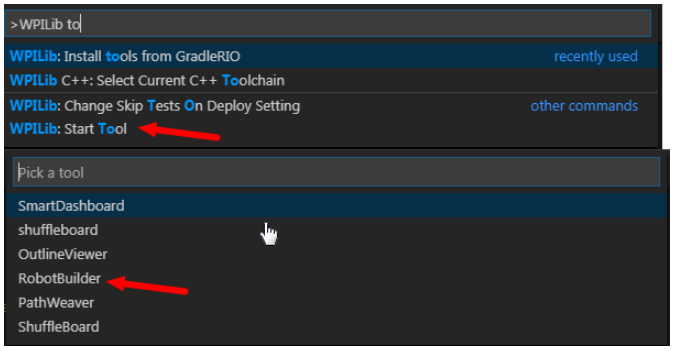
21.1.2 启动 RobotBuilder

备注: RobotBuilder 是一个 Java 程序所以应该可以在任何支持 Java 的平台上运行。我们已经在 macOS , Windows 和各种版本的 Linux 上成功运行了 RobotBuilder 。

获取 RobotBuilder

RobotBuilder 作为 WPILib 离线安装包的一部分被下载。若想得到更多信息，请看:ref:*Windows/macOS/Linux installation guides <docs/zero-to-robot/step-2/wpilib-setup:WPILib Installation Guide>*

选项 1—从 Visual Studio Code 开始

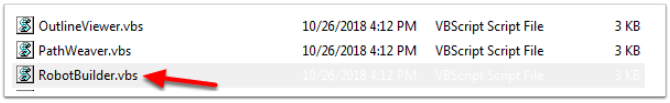


按下 "Ctrl+Shift+P" 然后输入 "WPILib" 或者单击右上角的 WPILib 的标志以气动 WPILib 的指令面板。选择 "Start Tool"，然后选择 "Robot Builder"。

选项 2-捷径

Shortcuts are installed to the Windows Start Menu and the `2024 WPILib Tools` folder on the desktop.

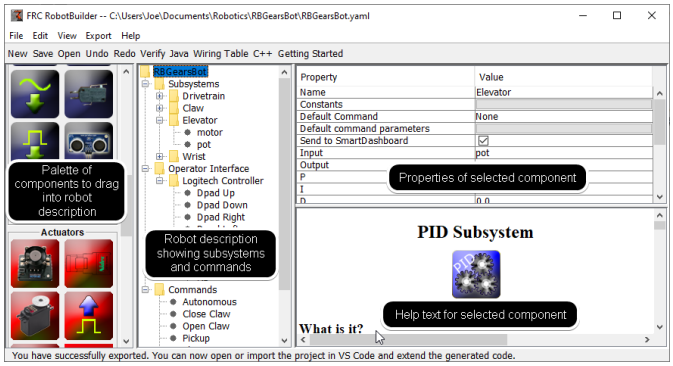
选项 3-从脚本运行



安装过程将工具安装在 `~/wpilib/YYYY/tools` 中（YYYY 是年份，~ 在 Windows 系统中是“C:UsersPublic“）。

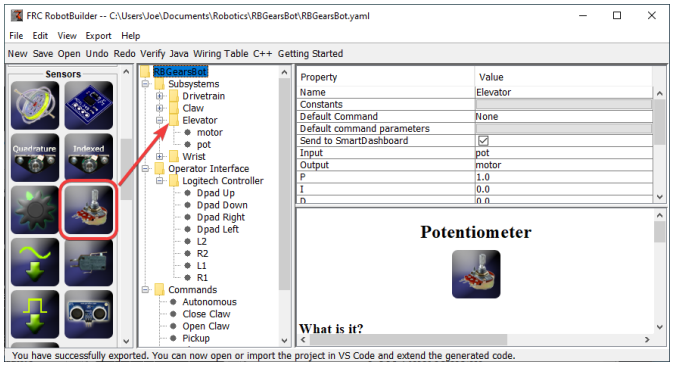
在文件夹中你可以找到“.vbs“（Windows）和“.py“（macOS/Linux）文件，可以用来发动各个工具。这些脚本使用正确的 JDK 来发动各个工具，并且是你需要用来发动工具的脚本。

21.1.3 RobotBuilder 用户界面



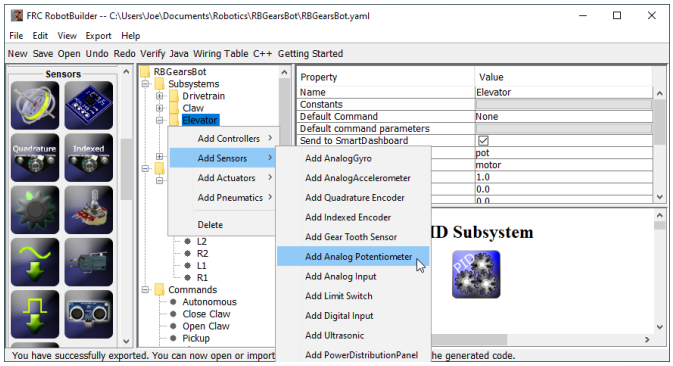
RobotBuilder 有一个专为机器人程序的快速开发设计的用户界面。几乎所有操作都是通过拖放或从下拉列表中选择选项来执行的。

将项目从调色板拖到机器人描述中



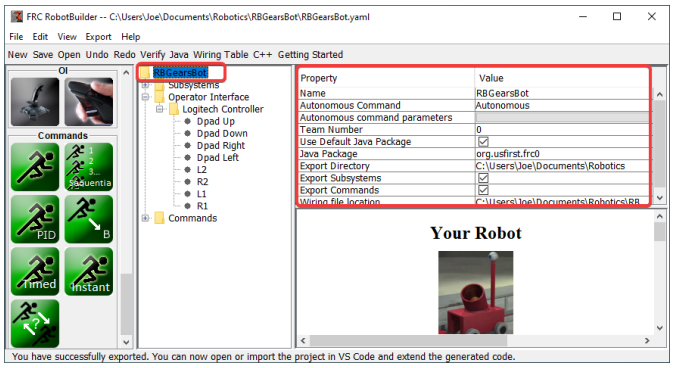
您可以通过在配置板项上开始拖动并在您希望项目所在的容器上结束拖动，将项目从配置板拖动到 robot 描述。在这个例子中，向电梯子系统投放一个电位计。

用右键菜单增加零件



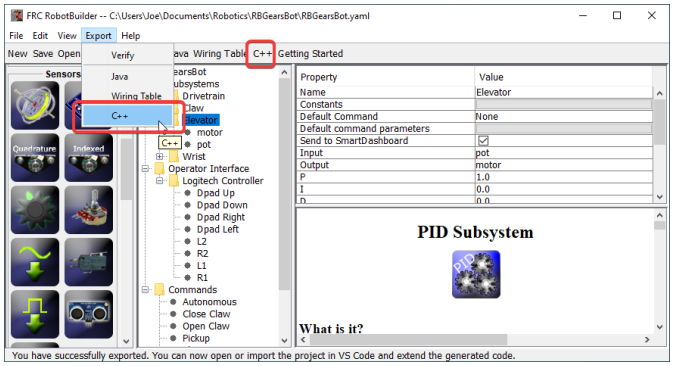
向机器人描述中添加物品的一个快捷方法是右键单击容器对象（电梯）并选择要添加的物品（电位器）。这与使用拖放是相同的，但是对某些人来说可能更简单。

编辑机器人描述项目的属性



选定项的属性将显示在属性查看器中。可以通过选择右边列中的值来编辑属性。

使用菜单系统



RobotBuilder 的操作既可以通过菜单系统选择，也可以从工具栏中选择等效项（如果可用的话）。

21.1.4 设置机器人项目

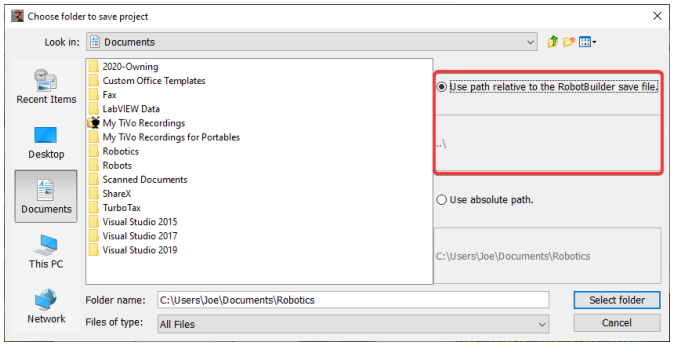
RobotBuilder 程序有一些默认属性需要设置，以便生成的程序和其他生成的文件能够正常工作。这个设置信息存储在机器人描述的属性中（第一行）。

机器人项目属性

描述机器人的属性是：

- **Name** - 被创建的机器人项目的名字
- **自动阶段指令** - 当程序处于自动阶段模式时默认运行的指令
- **自动阶段指令参数** - 自动阶段指令的参数
- **团队编号** - 项目的团队编号，用于在部署代码时定位机器人。
- **使用默认 Java 包** - 如果选中 RobotBuilder 将使用默认包 (frc.robot)。否则，您可以指定要使用的自定义包名称。
- **Java Package** - 生成项目代码时使用的生成 Java 包的名称
- **导出目录** - 选择导出到 Java 或 C++ 时项目生成的文件夹
- **Export Subsystems** - 检查 RobotBuilder 是否应该从项目中导出子系统类
- **Export Commands** - 检查 RobotBuilder 是否应该从项目中导出命令类
- **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.

在 RobotBuilder 项目中使用源代码控制

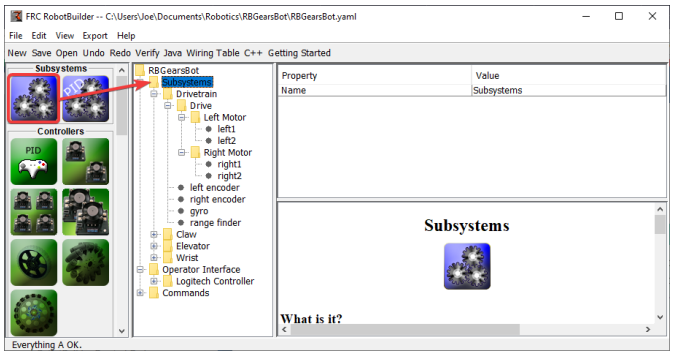


在使用源代码控制时，项目通常会在许多计算机上使用，并且到项目目录的路径可能因用户计算机的不同而不同。如果 RobotBuilder 项目文件是使用绝对路径存储的，那么它通常会包含用户名，并且不能跨多台计算机使用。要做到这一点，选择“相对路径”并将路径指定为项目文件的目录偏移量。在上面的示例中，项目文件存储在文件层次结构中项目文件上方的文件夹中。在这种情况下，用户名不是路径的一部分，它可以在您的所有计算机上移植。

21.1.5 创建子系统

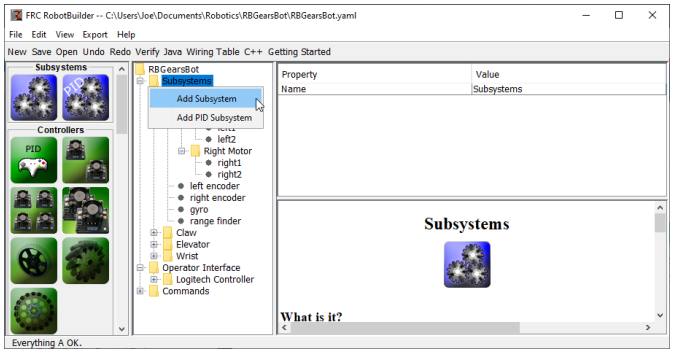
Subsystems are classes that encapsulate (or contain) all the data and code that make a subsystem on your robot operate. The first step in creating a robot program with the RobotBuilder is to identify and create all the subsystems on the robot. Examples of subsystems are grippers, ball collectors, the 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.

使用面板创建子系统



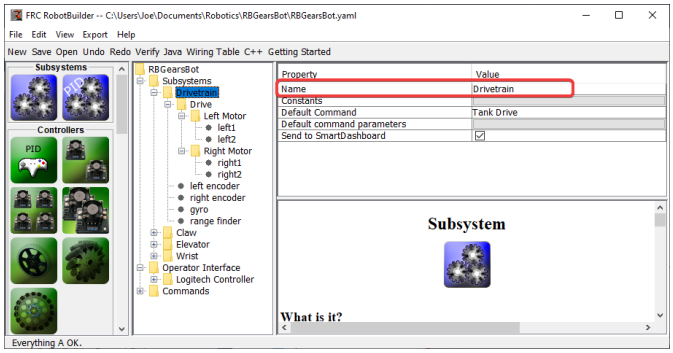
将子系统图标从面板拖到“robot description”中的“Subsystems”文件夹中，以创建子系统类。

使用上下文菜单创建子系统



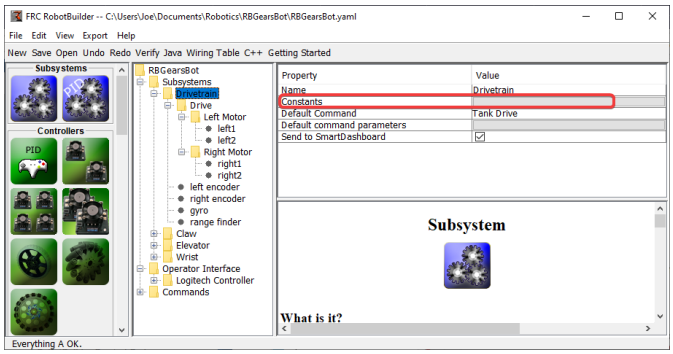
右键单击“robot description”中的“subsystem”文件夹然后将子系统添加到该文件夹。

命名子系统



如上所述通过拖动或使用上下文菜单创建子系统后，只需键入您要为子系统指定的名称。名称可以是多个由空格分隔的单词，RobotBuilder 会连接这些单词为您生成合适的 Java 或 C++ 类名称。

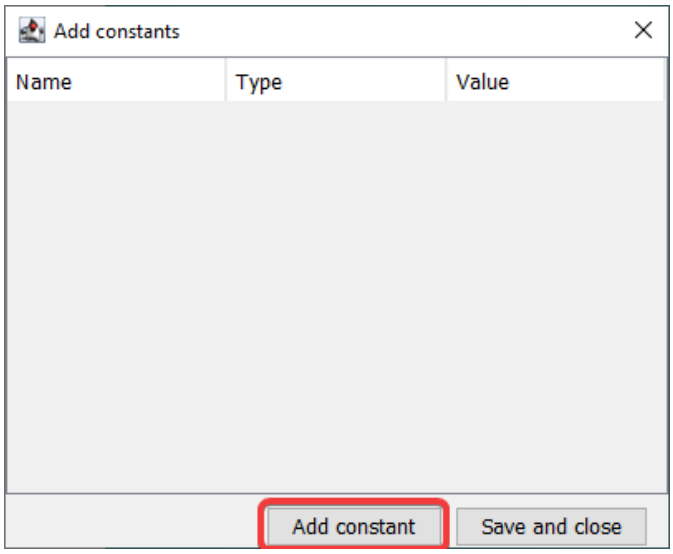
添加常量



常量对于减少代码中的幻数数量非常有用。在子系统中，它们可用于跟踪某些值，例如升降机特定高度的传感器值，或驱动机器人的速度。

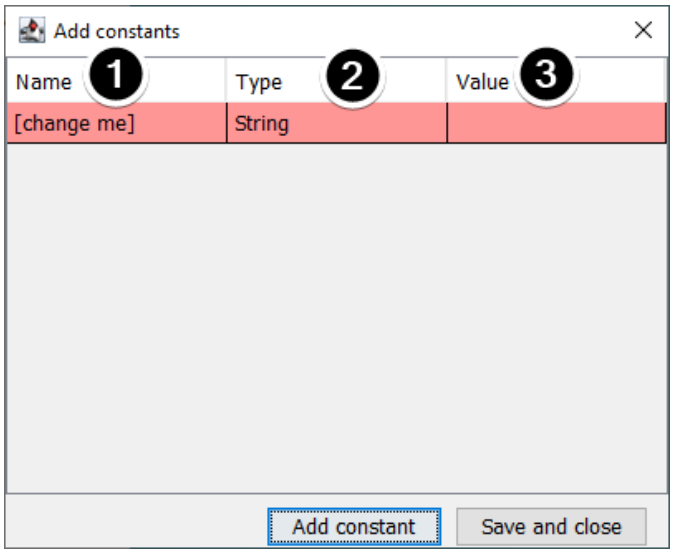
默认情况下，子系统中没有常量。按“常量”旁边的按钮打开一个对话框来创建一些。

创建常量



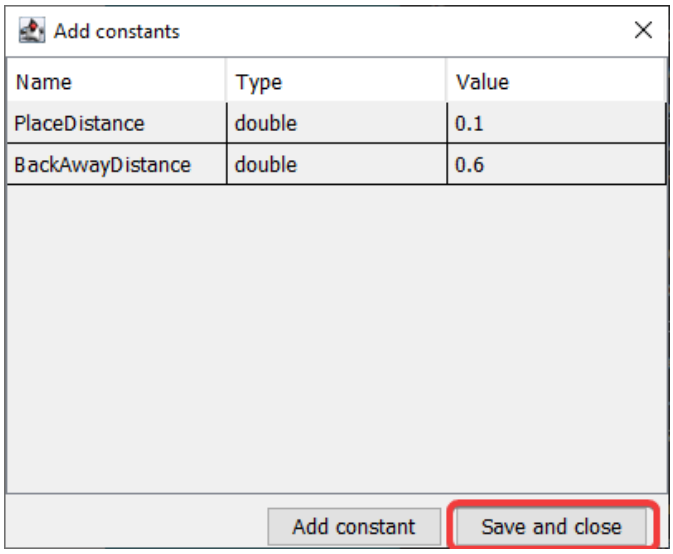
常量表一开始是空的。按“添加常量”添加一个。

添加常量



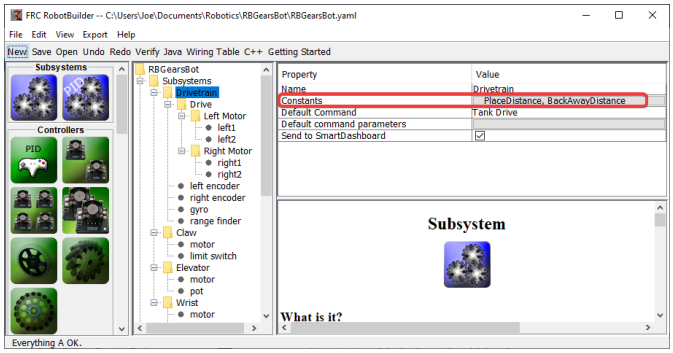
1. 常量的名称。将其更改为描述性内容。在传动系统的这个例子中，一些好的常数可能是“PlaceDistance”和“BackAwayDistance”。
2. 常量的类型。这很可能是一个双精度值，但您可以选择以下之一：字符串、双精度、整数、长整数、布尔值或字节。
3. 常量的值。

保存常量



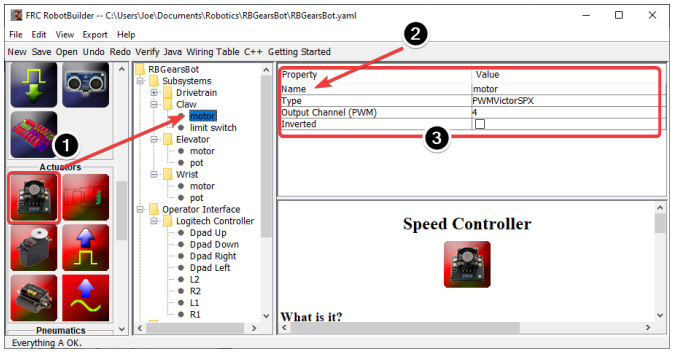
添加常量并设置其值后，只需按“保存并关闭”即可保存常量并关闭对话框。如果您不想保存，请按窗口顶部的退出按钮。

在保存之后



保存常量后，名称将出现在子系统属性的“常量”按钮中。

将执行器/传感器拖入子系统



将组件添加到子系统需要三个步骤：

1. 根据需要将执行器或传感器从调色板拖到子系统中。
2. 给执行器或传感器起一个有意义的名字
3. 编辑子系统中每个项目的属性，例如模块编号和通道编号。

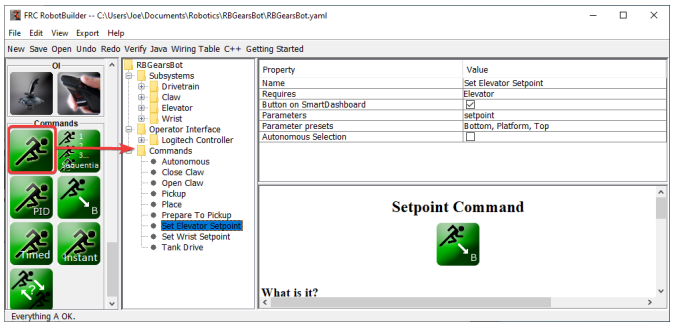
RobotBuilder 将自动为机器人上的每个模块使用递增的通道编号。如果您尚未为机器人接线，您可以让 RobotBuilder 为每个传感器或执行器分配唯一的通道编号，并根据生成的接线表为机器人接线。

这只是在 RobotBuilder 中创建子系统，随后将为子系统生成骨架代码。要使其实际操作您的机器人，请参阅：ref: *为子系统编写代码 <docs/software/wpilib-tools/robotbuilder/writing-code/robotbuilder-writing-subsystem-code:Writing the Code for a Subsystem>* .

21.1.6 创建一个指令

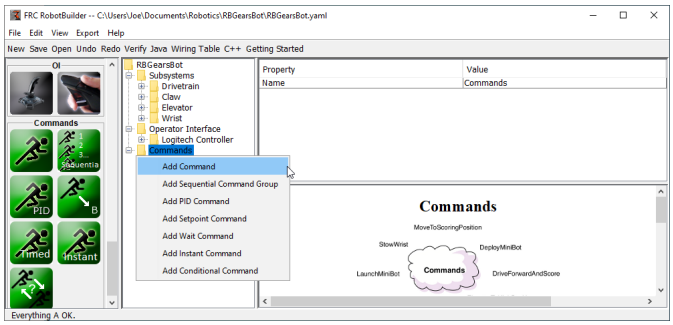
指令是您创建的类，用于为子系统提供行为或动作。子系统类应设置子系统的操作，例如 MoveElevator 启动升降舵移动，或 ElevatorToSetPoint 设置升降舵的 PID 设定点。这些命令将启动子系统执行并跟踪其完成时间。

将指令拖到指令文件夹



可以将简单指令从面板拖到机器人描述中。该命令将在“指令”文件夹下创建。

使用目录菜单创建指令



您还可以使用机器说明中“指令”文件夹中的右键单击上下文菜单来创建指令。

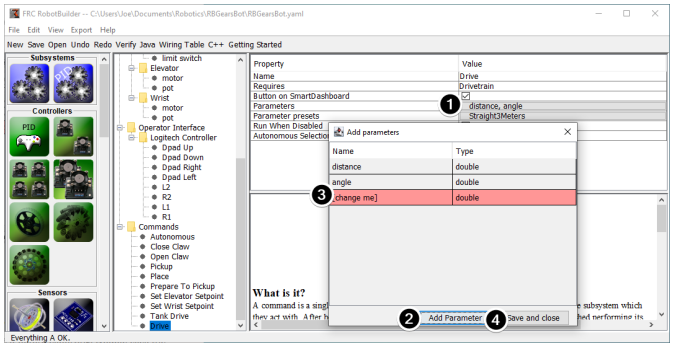
配置指令

Property	Value
Name	Set Elevator Setpoint
Requires	Elevator
Button on SmartDashboard	☑
Parameters	setpoint
Parameter presets	Bottom, Platform, Top
Autonomous Selection	☐

1. 用有意义的名称命名指令，以描述该指令将执行的操作。指令应该像在代码中统一命名，尽管单词之间可以有空格。
2. 设置此指令使用的子系统。此指令被加入列表后，它将自动停止当前正在运行的也需要此指令的任何指令。如果当前正在运行打开夹持器的指令（需要夹持器子系统），并且已将关闭夹持器指令加入列表，它将立即停止打开并开始关闭。
3. 告诉 RobotBuilder 是否应在 SmartDashboard 上为该指令创建按钮。将为每个参数预设一个按钮。
4. 设置此指令需要的参数。具有参数的单个命令可以与不带参数的两个或多个指令具有相同的作用。例如，“前进驱动”，“后退驱动”和“行驶距离”指令可以合并为一个单独的指令，该指令采用方向和距离值。
5. 设置参数预设。使用指令时，可以在 RobotBuilder 中的其他位置同时使用这些指令，例如将其绑定到操纵杆按钮或为子系统设置默认指令。
6. 禁用时运行。禁用机器人时允许指令运行。但是，禁用时指令的任何执行器都不会启动。
7. 自动阶段选择。是否应将指令添加到“可发送选择器”，以便可以将其选择为自动阶段指令。

设定值指令带有单个参数（“setpoint”，类型为 double）；不能为设置点指令添加，编辑或删除参数。

添加和编辑参数

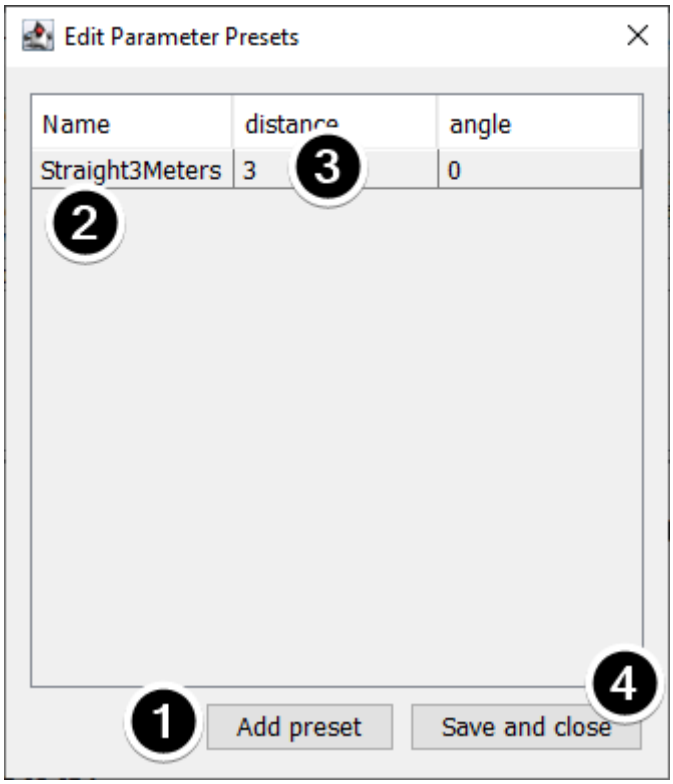


要添加或编辑参数：

1. 单击属性表的“值”列中的按钮
2. 按下“添加参数”按钮添加参数
3. 刚刚添加的参数。名称默认为“[change me]”，类型默认为 String。默认名称无效，因此您必须在导出之前进行更改。双击“名称”单元格，开始更改名称；双击“类型”单元格以选择类型。
4. 保存并关闭按钮将保存所有更改并关闭窗口。

只需拖动即可重新排列行，可以通过选择行并按 Delete 或 Backspace 将其删除。

添加和编辑参数预设

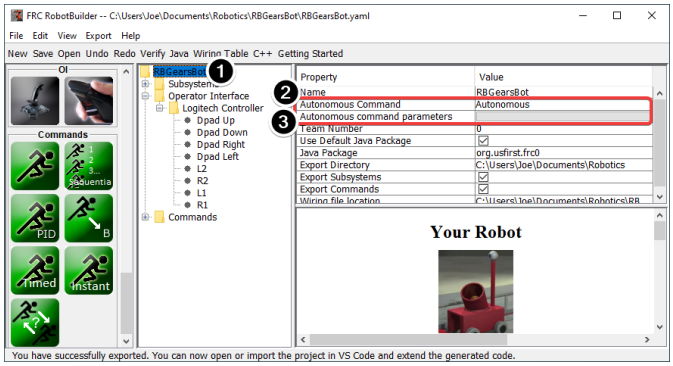


1. 单击“添加参数集”以添加新的预设。
2. 将预设名称更改为描述性名称。本示例中的预设用于打开和关闭夹持器子系统。
3. 更改一个（或多个）预设的参数值。您可以输入一个值（例如“3.14”），也可以从指令所需的子系统中定义的常量中进行选择。请注意，常量的类型必须与参数的类型相同-例如，您不能将 int 型常量传递给 double 型参数。
4. 单击“保存并关闭”以保存更改并退出对话框；要退出而不保存，请按窗口顶部栏中的退出按钮。

21.1.7 设置自动阶段命令

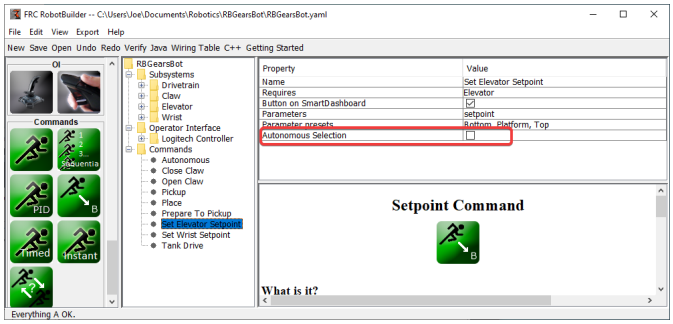
由于一个指令仅仅是机器人执行的一个或多个行动（行为），将机器人的自动运行描述成一个指令很有道理。虽然它可以是一个指令，它更可能是一个指令组（一组一起执行的指令）。

RobotBuilder 为可发送选择器 生成代码，它允许自动阶段指令运行以从仪表板被选择。



要指定在仪表板上未选择其他指令时运行的默认自动阶段指令，请执行以下操作：

- 在机器人程序说明中选择机器人
- 在将机器人置于自动阶段下时应运行的指令，填写“自动阶段”字段。这是一个下拉字段，将为您提供选择已定义的任何指令的选项。
- 设置指令采用的参数（如果有）。



要选择指令作为选项添加到“可发送选择器”中，请选中“自动阶段”复选框。

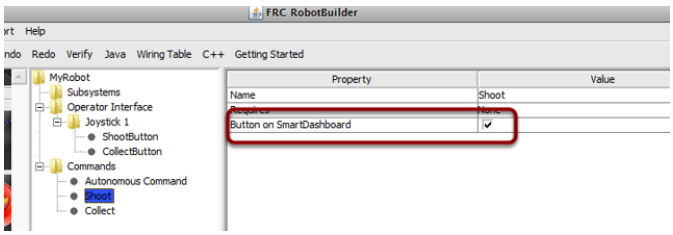
当机器人进入自动阶段时，将调度所选的自动阶段指令。

21.1.8 使用 Shuffleboard 测试指令

通过向 Shuffleboard/SmartDashboard 添加按钮来触发指令，可以轻松测试指令。通过这种方式，不需要与机器人程序的其余部分集成，并且可以轻松地独立测试指令。这是验证指令的最简单方法，因为在程序中使用一行代码，就可以在 Shuffleboard 上创建一个按钮来运行指令。然后可以将这些按钮留在原位以在将来验证子系统和命令操作。

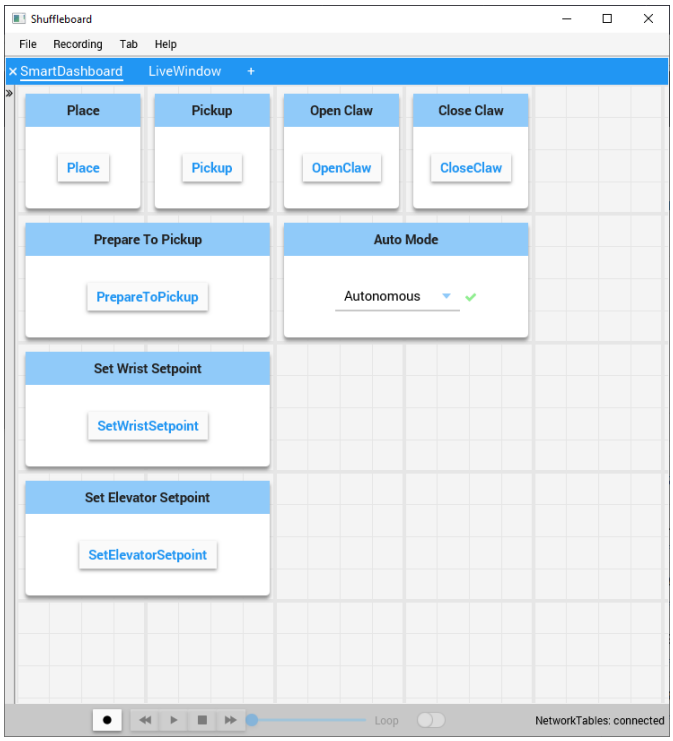
这个带来的额外好处是它能够更好地协调数个同时在给出指令的程序部分。当代码被写入机器的主程序项目后，每个指令都可以被拉出来单独调试。

在 Shuffleboard 上创建按钮



通过将机器人命令的实例放置到仪表板上，可以在 SmartDashboard 上创建按钮。这是一种非常常见的操作，因此已将其作为复选框添加到 RobotBuilder。编写命令时，请确保已选中该复选框，然后按钮将自动生成。

操作按钮



按钮将自动生成并显示在仪表板屏幕上。您可以重新排列 Shuffleboard 上的按钮。在这个例子中，有许多命令，每个命令都有一个相关的测试按钮。按命令按钮将运行指令。一旦按下，再次按下它将中断指令，导致调用 `Interrupted()` 方法。

手动添加命令

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));
```

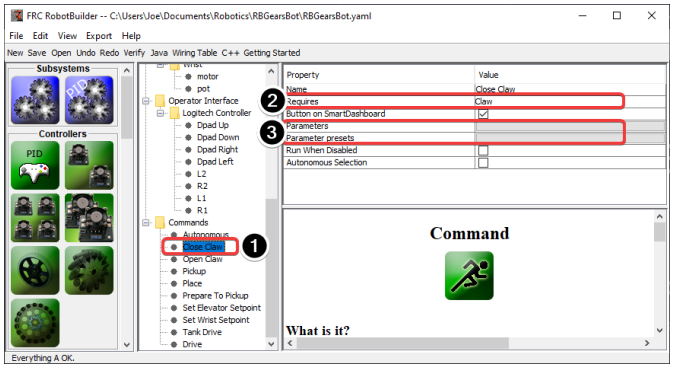
可以通过自己编写代码将命令手动添加到 Shuffleboard。这是通过将指令的实例以及应该与 Shuffleboard 上的按钮相关联的名称传递给 PutData 方法来完成的。只要按下按钮，就会安排这些实例。结果与 RobotBuilder 生成的代码完全相同，尽管单击 RobotBuilder 中的复选框比手动编写所有代码要容易得多。

21.1.9 将操作界面连接到指令

指令控制机器人的行为。该指令将子系统启动到某种操作模式，例如提升和升降舵，并继续运行，直到达到某个设定点或超时。然后，该命令将处理等待子系统完成的过程。这样，指令可以依次运行以开发更复杂的行为。

每当按下操作界面上的按钮时，RobotBuilder 还将生成代码以调度指令运行。您还可以编写代码以在发生特定触发条件时运行指令。

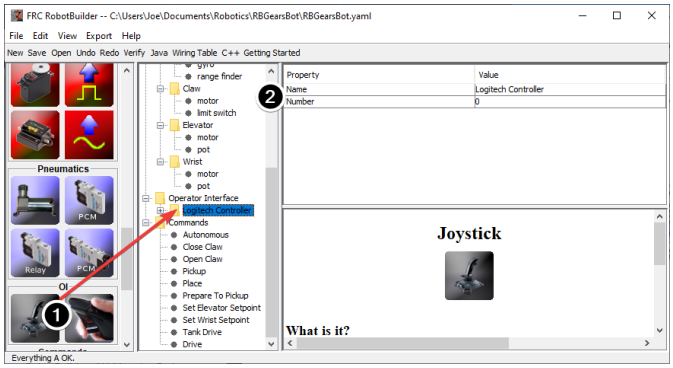
按下按钮运行指令



在此示例中，我们希望安排“Close Claw”指令在按下 Logitech 游戏手柄（按钮 6）上的 dpad 右方向按钮时运行。

1. 运行的指令叫做“Close Claw”，它的作用是关闭机器人的爪子
2. 请注意，该指令需要 Claw 子系统。这将确保即使在使用爪的同时发生另一个操作时，该指令也开始运行。在这种情况下，先前的指令将被中断。
3. 参数使一个指令可以执行多项操作。预设让您定义传递给指令的值并重复使用它们

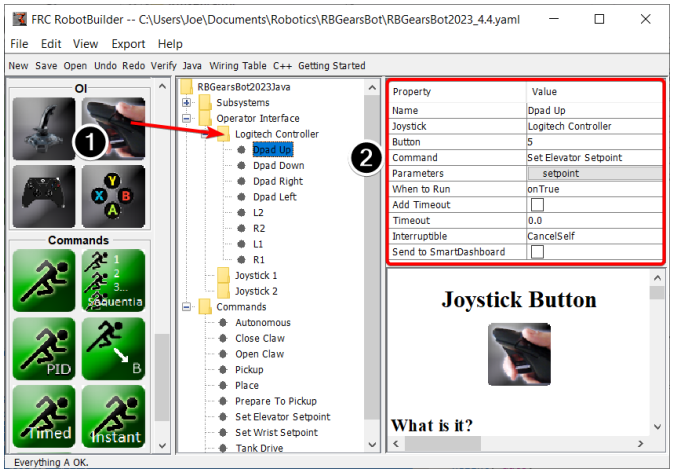
将操纵杆添加到机器人程序



将操纵杆添加到机器人程序

1. 将操纵杆拖到机器人程序中的“操作员界面”文件夹中
2. 命名操纵杆，以使其能反映操纵杆的用途，并设置 USB 端口号

将按钮链接到“移动电梯”指令



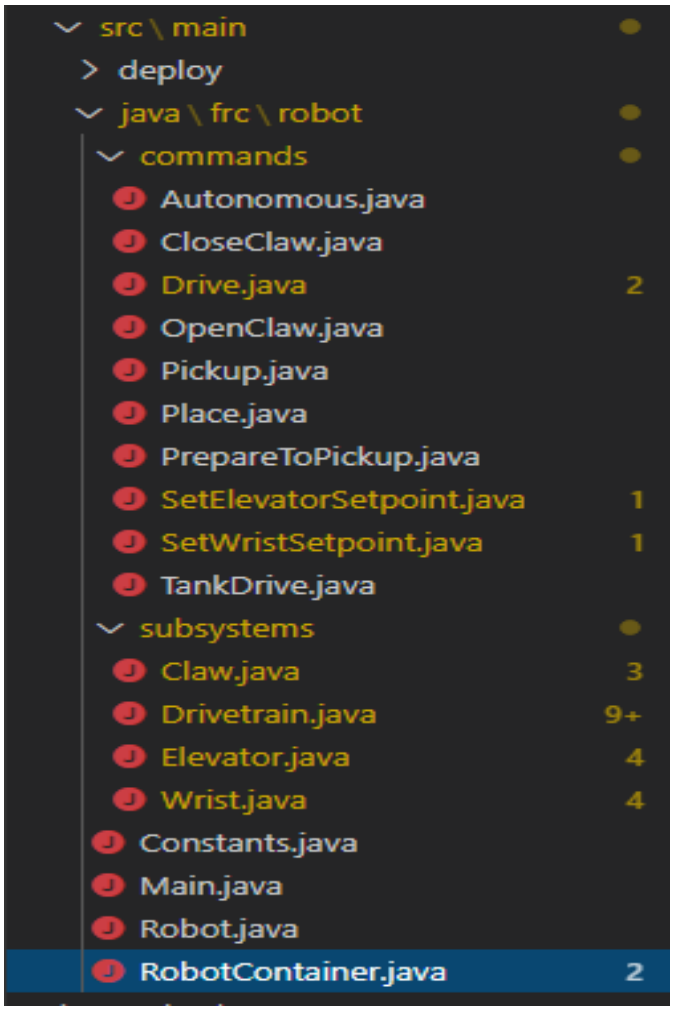
将应当被按下的按钮添加到程序中

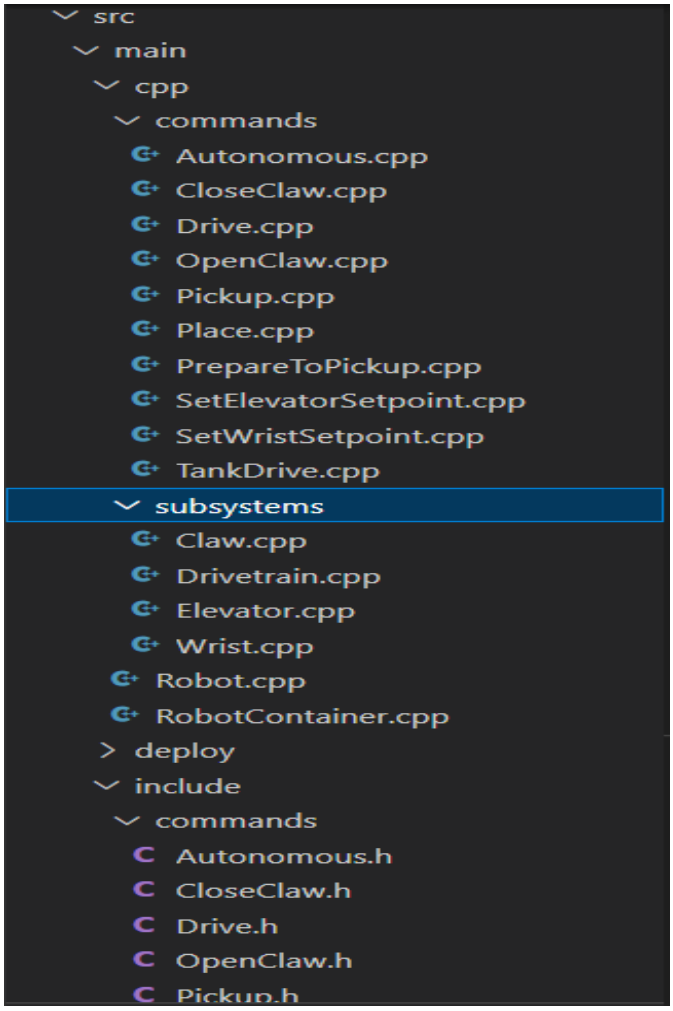
1. 将操纵杆按钮拖到操纵杆（罗技控制器）上，使其位于操纵杆下方
2. Set the properties for the button: the button number, the command to run when the button is pressed, parameters the command takes, and the *When to run* property to *onTrue* to indicate that the command should run whenever the joystick button is pressed.

备注： 必须将操纵杆按钮拖到操纵杆上（下方）。添加按钮之前，操作员界面文件夹中必须存在有操纵杆。

21.1.10 RobotBuilder 创建的代码

RobotBuilder 生成的项目的布局





RobotBuilder 生成的项目由用于命令的包（Java）或文件夹（C++）和子系统的另一个包组成。每个指令或子系统对象都存储在这些容器下。在项目的顶层，您将找到机器人主程序 (RobotContainer.java/C++)。

有关基于指令的机器人组织的更多信息，请参阅：doc:*/docs/software/commandbased/structuring-command-based-project*

自动生成代码

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);
```

当修改机器人描述并重新导出代码时，RobotBuilder 被设计成不能修改您对文件所做的任何更改的形式，从而保留您的代码。这使得 RobotBuilder 成为一个全生命周期的工具。为了知道哪些代码可以被 RobotBuilder 修改，它生成的部分可能要重写，并使用一些特殊注释分隔。上面的例子显示了这些注释。不要在这些注释块中添加任何代码，下次从 RobotBuilder 导出项目时，它将被重写。

如果必须修改其中一个版块中的代码，则可以删除注释，但这将阻止以后进一步的更新。在上面的示例中，如果删除了//BEGIN 和//END 注释，那么稍后在 RobotBuilder 中会添加另一个所需的子系统，它将不会在下一个导出中生成。

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.

主要的机器人程序

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
24  * functions corresponding to each mode, as described in the TimedRobot
25  * documentation. If you change the name of this class or the package after
26  * creating this project, you must also update the build.properties file in
27  * the project.
28  */
```

(续下页)

(接上页)

```
29 public class Robot extends TimedRobot { // (1)
30
31     private Command m_autonomousCommand;
32
33     private RobotContainer m_robotContainer;
34
35     /**
36      * This function is run when the robot is first started up and should be
37      * used for any initialization code.
38      */
39     @Override
40     public void robotInit() {
41         // Instantiate our RobotContainer.  This will perform all our button bindings,
    ↪ and put our
42         // autonomous chooser on the dashboard.
43         m_robotContainer = RobotContainer.getInstance();
44         HAL.report(tResourceType.kResourceType_Framework, tInstances.kFramework_
    ↪RobotBuilder);
45     }
46
47     /**
48     * This function is called every robot packet, no matter the mode. Use this for␣
    ↪items like
49     * diagnostics that you want ran during disabled, autonomous, teleoperated and␣
    ↪test.
50     *
51     * <p>This runs after the mode specific periodic functions, but before
52     * LiveWindow and SmartDashboard integrated updating.
53     */
54     @Override
55     public void robotPeriodic() {
56         // Runs the Scheduler.  This is responsible for polling buttons, adding newly-
    ↪scheduled
57         // commands, running already-scheduled commands, removing finished or␣
    ↪interrupted commands,
58         // and running subsystem periodic() methods.  This must be called from the␣
    ↪robot's periodic
59         // block in order for anything in the Command-based framework to work.
60         CommandScheduler.getInstance().run();  // (2)
61     }
62
63
64     /**
65     * This function is called once each time the robot enters Disabled mode.
66     */
67     @Override
68     public void disabledInit() {
69     }
70
71     @Override
72     public void disabledPeriodic() {
73     }
74
75     /**
76     * This autonomous runs the autonomous command selected by your {@link␣
    ↪RobotContainer} class.
77     */
```

(续下页)

(接上页)

```
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++ (Header)

```
// ROBOTBUILDER TYPE: Robot.
#pragma once

#include <frc/TimedRobot.h>
#include <frc2/command/Command.h>

#include "RobotContainer.h"

class Robot : public frc::TimedRobot { // {1}
 public:
  void RobotInit() override;
  void RobotPeriodic() override;
  void DisabledInit() override;
  void DisabledPeriodic() override;
  void AutonomousInit() override;
  void AutonomousPeriodic() override;
  void TeleopInit() override;
  void TeleopPeriodic() override;
  void TestPeriodic() override;

 private:
  // Have it null by default so that if testing teleop it
  // doesn't have undefined behavior and potentially crash.
  frc2::Command* m_autonomousCommand = nullptr;

  RobotContainer* m_container = RobotContainer::GetInstance();
};
```

C++ (Source)

```
// ROBOTBUILDER TYPE: Robot.

#include "Robot.h"

#include <frc/smartdashboard/SmartDashboard.h>
#include <frc2/command/CommandScheduler.h>

void Robot::RobotInit() {}

/**
 * This function is called every robot packet, no matter the mode. Use
 * this for items like diagnostics that you want to run during disabled,
 * autonomous, teleoperated and test.
 *
 * <p> This runs after the mode specific periodic functions, but before
 * LiveWindow and SmartDashboard integrated updating.
 */
void Robot::RobotPeriodic() { frc2::CommandScheduler::GetInstance().Run(); } // (2)

/**
 * This function is called once each time the robot enters Disabled mode. You
 * can use it to reset any subsystem information you want to clear when the
 * robot is disabled.
```

(续下页)

(接上页)

```
 */
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
```

这是 RobotBuilder 生成的主程序。这个计划有几个部分 (突出显示的部分):

1. 这个类扩展了 timerobot。TimeRobot 会每 20 毫秒调用你的’‘autonomouPeriodic() ‘‘和’ teleopPeriodic() ‘‘的方法。
2. 在每 20ms 调用一次的 robotsPeriodic 方法中，进行一次调度传递。
3. 提供的自动阶段指令在 `autonomousInit()` 方法中被安排在自动阶段开始时，并在自动阶段结束时在 “teleopInit()“ 中取消。

RobotContainer

Java

```
// ROBOTBUILDER TYPE: RobotContainer.

package frc.robot;

import frc.robot.commands.*;
import frc.robot.subsystems.*;
import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj2.command.Command.InterruptionBehavior;

// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.InstantCommand;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj2.command.button.JoystickButton;
import frc.robot.subsystems.*;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS


/**
 * This class is where the bulk of the robot should be declared.  Since Command-based is a
 * "declarative" paradigm, very little robot logic should actually be handled in the {@link Robot}
 * periodic methods (other than the scheduler calls).  Instead, the structure of the robot
 * (including subsystems, commands, and button mappings) should be declared here.
 */
public class RobotContainer {

  private static RobotContainer m_robotContainer = new RobotContainer();

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
// The robot's subsystems
    public final Wrist m_wrist = new Wrist(); // (1)
    public final Elevator m_elevator = new Elevator();
    public final Claw m_claw = new Claw();
    public final Drivetrain m_drivetrain = new Drivetrain();

// Joysticks
private final Joystick joystick2 = new Joystick(2); // (3)
private final Joystick joystick1 = new Joystick(1);
private final Joystick logitechController = new Joystick(0);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS


  // A chooser for autonomous commands
  SendableChooser<Command> m_chooser = new SendableChooser<>();

  /**
  * The container for the robot.  Contains subsystems, OI devices, and commands.
```

(续下页)

(接上页)

```
61    */
62    private RobotContainer() {
63          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
64      // Smartdashboard Subsystems
65      SmartDashboard.putData(m_wrist); // (6)
66      SmartDashboard.putData(m_elevator);
67      SmartDashboard.putData(m_claw);
68      SmartDashboard.putData(m_drivetrain);
69
70
71      // SmartDashboard Buttons
72      SmartDashboard.putData("Close Claw", new CloseClaw( m_claw )); // (6)
73      SmartDashboard.putData("Open Claw: OpenTime", new OpenClaw(1.0, m_claw));
74      SmartDashboard.putData("Pickup", new Pickup());
75      SmartDashboard.putData("Place", new Place());
76      SmartDashboard.putData("Prepare To Pickup", new PrepareToPickup());
77      SmartDashboard.putData("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0,
    ↪ m_elevator));
78      SmartDashboard.putData("Set Elevator Setpoint: Platform", new␣
    ↪SetElevatorSetpoint(0.2, m_elevator));
79      SmartDashboard.putData("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3,␣
    ↪m_elevator));
80      SmartDashboard.putData("Set Wrist Setpoint: Horizontal", new SetWristSetpoint(0,␣
    ↪m_wrist));
81      SmartDashboard.putData("Set Wrist Setpoint: Raise Wrist", new SetWristSetpoint(-
    ↪45, m_wrist));
82      SmartDashboard.putData("Drive: Straight3Meters", new Drive(3, 0, m_drivetrain));
83      SmartDashboard.putData("Drive: Place", new Drive(Drivetrain.PlaceDistance,␣
    ↪Drivetrain.BackAwayDistance, m_drivetrain));
84
85      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
86      // Configure the button bindings
87      configureButtonBindings();
88
89      // Configure default commands
90          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SUBSYSTEM_DEFAULT_COMMAND
91      m_drivetrain.setDefaultCommand(new TankDrive(() -> getJoystick1().getY(), () ->␣
    ↪getJoystick2().getY(), m_drivetrain)); // (5)
92
93
94      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SUBSYSTEM_DEFAULT_COMMAND
95
96      // Configure autonomous sendable chooser
97          // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
98
99      m_chooser.addOption("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0, m_
    ↪elevator));
100     m_chooser.addOption("Set Elevator Setpoint: Platform", new SetElevatorSetpoint(0.
    ↪2, m_elevator));
101     m_chooser.addOption("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3, m_
    ↪elevator));
102     m_chooser.setDefaultOption("Autonomous", new Autonomous()); // (2)
103
104     // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
105
106     SmartDashboard.putData("Auto Mode", m_chooser);
107   }
```

(续下页)

(接上页)

```

  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;
    }

public Joystick getJoystick1() {
        return joystick1;
```

(续下页)

(接上页)

```
    }

public Joystick getJoystick2() {
        return joystick2;
    }


    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=FUNCTIONS

  /**
   * Use this to pass the autonomous command to the main {@link Robot} class.
   *
   * @return the command to run in autonomous
   */
  public Command getAutonomousCommand() {
    // The selected command will be run in autonomous
    return m_chooser.getSelected();
  }


}
```

C++ (Header)

```
// 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

class RobotContainer {

```

(续下页)

(接上页)

```
public:

    frc2::Command* GetAutonomousCommand();
    static RobotContainer* GetInstance();

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=PROTOTYPES
// The robot's subsystems
Drivetrain m_drivetrain; // (1)
Claw m_claw;
Elevator m_elevator;
Wrist m_wrist;


frc::Joystick* getJoystick2();
frc::Joystick* getJoystick1();
frc::Joystick* getLogitechController();

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=PROTOTYPES

private:

    RobotContainer();

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
// Joysticks
frc::Joystick m_logitechController{0}; // (3)
frc::Joystick m_joystick1{1};
frc::Joystick m_joystick2{2};

frc::SendableChooser<frc2::Command*> m_chooser;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS

Autonomous m_autonomousCommand;
    static RobotContainer* m_robotContainer;

    void ConfigureButtonBindings();
};
```

C++ (Source)

```
// ROBOTBUILDER TYPE: RobotContainer.

#include "RobotContainer.h"
#include <frc2/command/ParallelRaceGroup.h>
#include <frc/smartdashboard/SmartDashboard.h>



RobotContainer* RobotContainer::m_robotContainer = NULL;

RobotContainer::RobotContainer() : m_autonomousCommand(
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR
){
```

(续下页)

(接上页)

```
24
25
26
27      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTOR
28
29      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
30      // Smartdashboard Subsystems
31      frc::SmartDashboard::PutData(&m_drivetrain);
32      frc::SmartDashboard::PutData(&m_claw);
33      frc::SmartDashboard::PutData(&m_elevator);
34      frc::SmartDashboard::PutData(&m_wrist);
35
36
37      // SmartDashboard Buttons
38      frc::SmartDashboard::PutData("Drive: Straight3Meters", new Drive(3, 0, &m_
    ↪drivetrain)); // (6)
39      frc::SmartDashboard::PutData("Drive: Place", new Drive(Drivetrain::PlaceDistance,␣
    ↪Drivetrain::BackAwayDistance, &m_drivetrain));
40      frc::SmartDashboard::PutData("Set Wrist Setpoint: Horizontal", new␣
    ↪SetWristSetpoint(0, &m_wrist));
41      frc::SmartDashboard::PutData("Set Wrist Setpoint: Raise Wrist", new␣
    ↪SetWristSetpoint(-45, &m_wrist));
42      frc::SmartDashboard::PutData("Set Elevator Setpoint: Bottom", new␣
    ↪SetElevatorSetpoint(0, &m_elevator));
43      frc::SmartDashboard::PutData("Set Elevator Setpoint: Platform", new␣
    ↪SetElevatorSetpoint(0.2, &m_elevator));
44      frc::SmartDashboard::PutData("Set Elevator Setpoint: Top", new␣
    ↪SetElevatorSetpoint(0.3, &m_elevator));
45      frc::SmartDashboard::PutData("Prepare To Pickup", new PrepareToPickup());
46      frc::SmartDashboard::PutData("Place", new Place());
47      frc::SmartDashboard::PutData("Pickup", new Pickup());
48      frc::SmartDashboard::PutData("Open Claw: OpenTime", new OpenClaw(1.0_s, &m_claw));
49      frc::SmartDashboard::PutData("Close Claw", new CloseClaw( &m_claw ));
50
51      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SMARTDASHBOARD
52
53      ConfigureButtonBindings();
54
55      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT-COMMANDS
56  m_drivetrain.SetDefaultCommand(TankDrive([this] {return getJoystick1()->GetY();},␣
    ↪[this] {return getJoystick2()->GetY();}, &m_drivetrain)); // (5)
57
58      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT-COMMANDS
59
60      // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
61
62      m_chooser.AddOption("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0, &
    ↪m_elevator));
63      m_chooser.AddOption("Set Elevator Setpoint: Platform", new SetElevatorSetpoint(0.
    ↪2, &m_elevator));
64      m_chooser.AddOption("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3, &m_
    ↪elevator));
65
66      m_chooser.SetDefaultOption("Autonomous", new Autonomous()); // (2)
67
68      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
69
```

(续下页)

(接上页)

```
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
115 frc::Joystick* RobotContainer::getLogitechController() {
116    return &m_logitechController;
117 }
118 frc::Joystick* RobotContainer::getJoystick1() {
```

(续下页)

(接上页)

```
119     return &m_joystick1;
120 }
121 frc::Joystick* RobotContainer::getJoystick2() {
122     return &m_joystick2;
123 }
124
125     // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=FUNCTIONS
126
127
128 frc2::Command* RobotContainer::GetAutonomousCommand() {
129   // The selected command will be run in autonomous
130   return m_chooser.GetSelected();
131 }
```

这是由 RobotBuilder 生成的 RobotContainer，它是定义子系统和操作员界面的地方。该计划有许多部分（高亮的部分）：

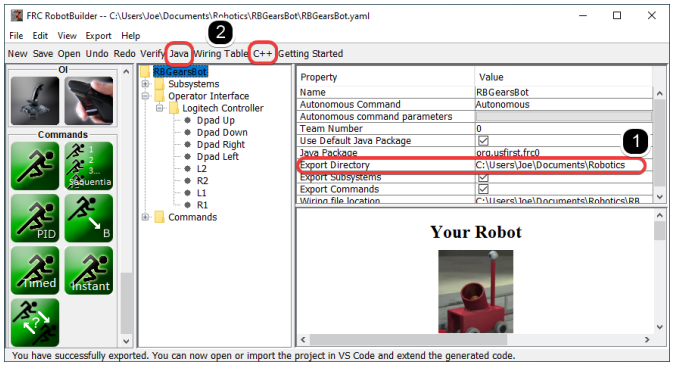
1. 每个子系统都在此处声明。它们可以作为参数传递给任何需要它们的命令。
2. 如果 RobotBuilder 机器人属性中提供了自动阶段指令，则会将其添加到可在仪表板上选择的可发送选择器。
3. 所有操作员界面组件的代码都在此处生成。
4. 此外，将 OI 按钮链接到应该运行的指令的代码也在此处生成。
5. Commands to be run on a subsystem when no other commands are running are defined here.
6. 通过仪表板运行的命令在此处指令。

21.2 机器人制造者-编写代码

21.2.1 为项目生成代码

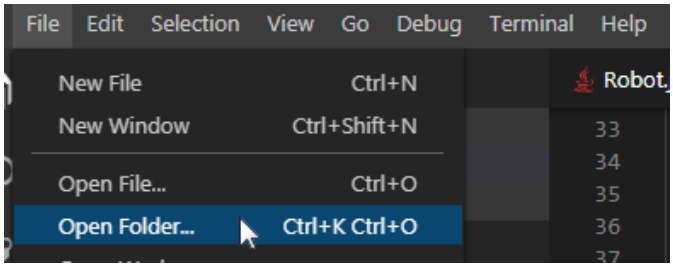
在 RobotBuilder 中建立了你的机器人框架以后，你需要将代码导出并装入到 Visual Studio Code 。这个文章描述了这样做的过程。

为项目生成代码



核实导出目录指向你想要的地方（1）然后点击 Java 或 C++（2）来生成一个 VS Code 项目或升级代码。

在 VisualStudio Code 中打开项目。

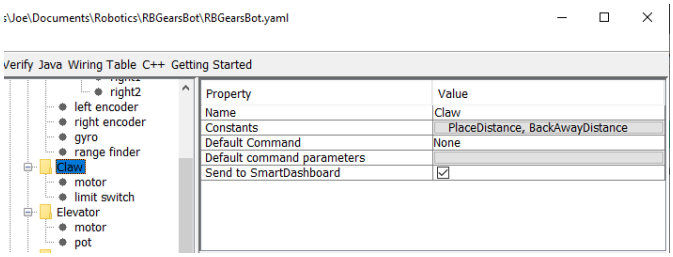


打开 VS Code 并选择 **File -> Open Folder** 。导航到你的输出位置并且点击 **Select Folder** 。

21.2.2 为子系统编写代码

添加代码以创建实际工作子系统非常简单。对于不使用反馈的简单子系统，结果证明非常简单。在本节中，我们将看一个 *Claw* 子系统的示例。*Claw* 子系统还有一个限位开关来确定物体是否在抓地力中。

爪子系统的 RobotBuilder 表示



机械臂末端的爪子是一个由单个 VictorSPX 电机控制器控制的子系统。我们希望电机做三件事，打开，关闭和停止移动。这是子系统的责任。打开和关闭的时间将在本教程后面的指令处讨论到。我们还将定义一个方法来判断爪子是否正在抓取物体。

添加子系统功能

Java

```
11 // ROBOTBUILDER TYPE: Subsystem.
12
13 package frc.robot.subsystems;
14
15
16 import frc.robot.commands.*;
17 import edu.wpi.first.wpilibj.livewindow.LiveWindow;
18 import edu.wpi.first.wpilibj2.command.SubsystemBase;
19
20 // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
21 import edu.wpi.first.wpilibj.DigitalInput;
22 import edu.wpi.first.wpilibj.motorcontrol.MotorController;
23 import edu.wpi.first.wpilibj.motorcontrol.PWMVictorSPX;
24
```

(续下页)

(接上页)

```
    // 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
    }

    @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() {
```

(续下页)

(接上页)

```
        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
}

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);
```

(续下页)

(接上页)

```
}

void Claw::Stop() {
    m_motor.Set(0.0);
}

bool Claw::IsGripping() {
    return m_limitswitch.Get();
}
```

将方法添加到 `claw.java` 或 `claw.cpp` 将打开，关闭和停止爪子移动并获取爪子限位开关。这些将由实际操作爪子的命令使用。

备注： 注释已从此文件中删除，以便更轻松地查看此文档的更改。

请注意，名为“motor”和“limitswitch”的成员变量是由 RobotBuilder 创建的，因此可以在整个子系统中使用。每个拖入的调色板项目都有一个成员变量，其名称在 RobotBuilder 中给出。

将方法声明添加到头文件（仅 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();
```

(续下页)

(接上页)

```
    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


};
```

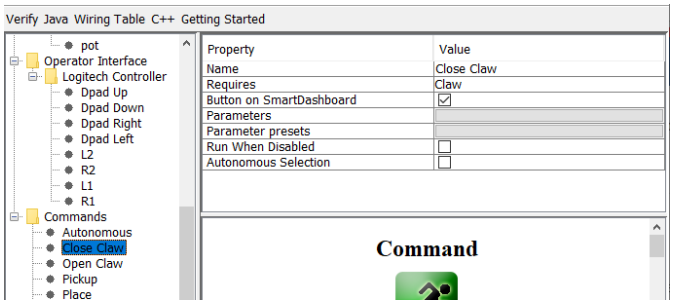
除了将方法添加到类实现文件“Claw.cpp”之外，还需要将方法的声明添加到头文件“Claw.h”中。必须添加的那些声明显示在此处。

要将行为添加到爪子子系统以处理打开和关闭，您需要*define commands*。

21.2.3 为指令写代码。

子系统类让机器人的机械装置移动，但为了让它在正确的时间停止并通过更复杂的操作顺序，你编写命令。之前在:doc: *编写子系统代码时，我们为机器人上的‘Claw* 子系统开发了代码，用于启动爪子张开、闭合或停止移动。现在我们将为一个命令编写代码，该指令将在正确的时间实际运行爪形电机，以使爪形打开和关闭。我们的爪形示例是一个非常简单的机制，我们运行电机 1 秒钟以将其打开或直到限位开关跳闸将其关闭。

在 RobotBuilder 中关闭 Claw 指令



这是 RobotBuilder 中 *CloseClaw* 指令的定义。请注意，它需要 *Claw* 子系统。这将在下一步中解释。

生成的 CloseClaw 类

JAVA

```
// ROBOTBUILDER TYPE: Command.

package frc.robot.commands;
import edu.wpi.first.wpilibj2.command.CommandBase;

```

(续下页)

(接上页)

```
// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS
import frc.robot.subsystems.Claw;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=IMPORTS

/**
*
*/
public class CloseClaw extends CommandBase {

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_DECLARATIONS
        private final Claw m_claw;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_DECLARATIONS

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS


    public CloseClaw(Claw subsystem) {


    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES

        m_claw = subsystem;
        addRequirements(m_claw);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
    }

    // Called when the command is initially scheduled.
    @Override
    public void initialize() {
        m_claw.close(); // (1)
    }

    // Called every time the scheduler runs while the command is scheduled.
    @Override
    public void execute() {
    }

    // Called once the command ends or is interrupted.
    @Override
    public void end(boolean interrupted) {
        m_claw.stop(); // (3)
    }

    // Returns true when the command should end.
    @Override
    public boolean isFinished() {
        return m_claw.isGripping(); // (2)
    }

    @Override
```

(续下页)

(接上页)

```
    public boolean runsWhenDisabled() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
        return false;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
    }
}
```

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 将为 'CloseClaw' 指令生成类文件。指令代表了爪子的行为，即随着时间的推移进行的操作。要操作这个非常简单的爪形机构，电机需要在关闭方向上运行。*Claw* 子系统具有启动和停止电机向正确方向运行的方法。命令的职责是在正确的时间运行电机。添加框中显示的代码行以添加此行为。

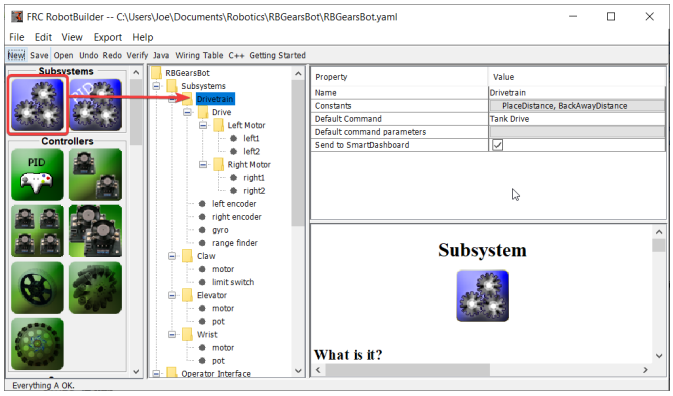
1. 通过调用在 *CloseClaw* Initialize 方法中添加到 *Claw* 子系统的 *Close()* 方法，启动爪形电机朝关闭方向移动。
2. This command is finished when the limit switch in the *Claw* subsystem is tripped.
3. `End()` 方法在命令完成时被调用，它是一个清理的地方。在这种情况下，电机停止，因为时间已用完。

21.2.4 用 Tank Drive 和 Joystick 操控机器人

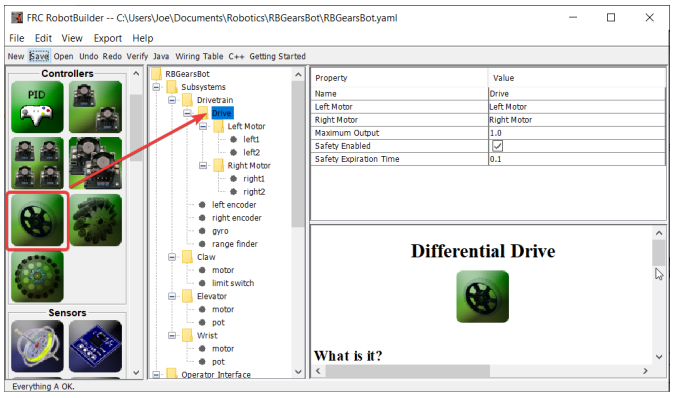
一个常见的用例是有一个操纵杆来驱动作为子系统一部分的一些执行器。问题是操纵杆是在 RobotContainer 类中创建的，要控制的电机在子系统中。这个想法是创建一个指令，在调度时从操纵杆读取输入并调用在驱动电机的子系统上创建的方法。

在这个例子中，一个底盘子系统在一个用一对操纵杆的 tank drive 被运行。

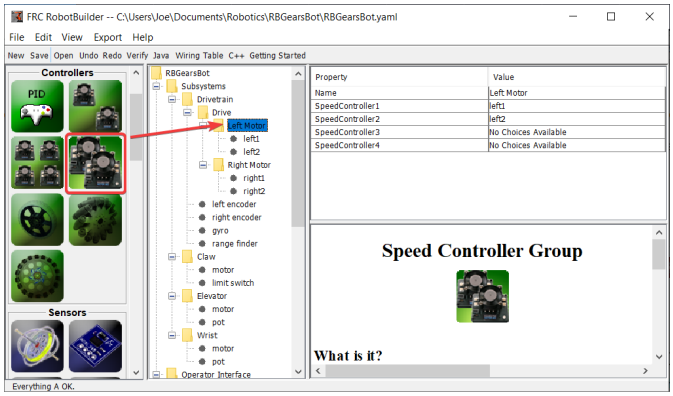
建造一个 Drive Train Subsystem 。



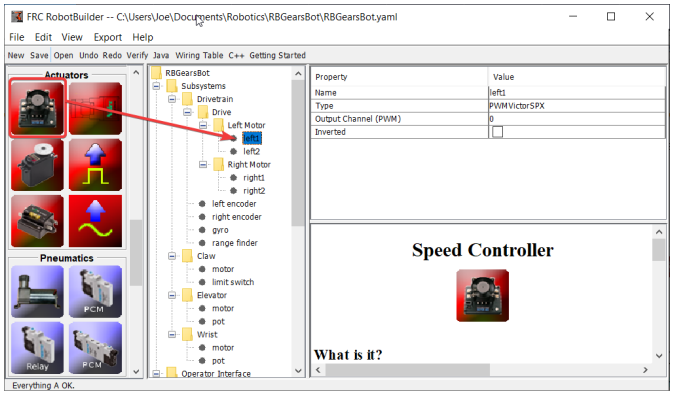
创建一个名为 Drive Train 的子系统。它的职责是处理机器人基地的驱动。



在 Drive Train 内部，为两个电机驱动器创建一个差分驱动器对象。作为差动驱动类的一部分，有一个左电机和右电机。

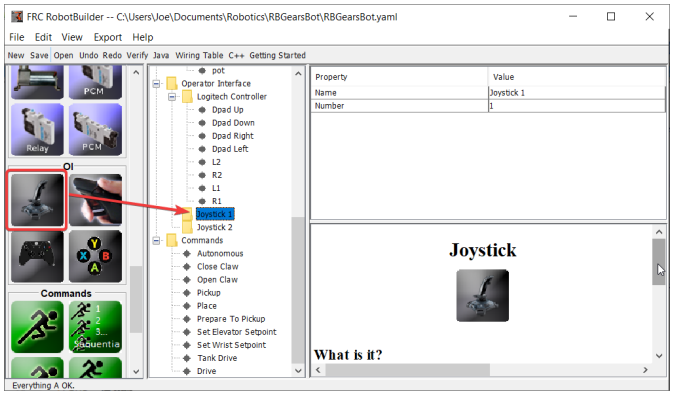


Since we want to use more then two motors to drive the robot, inside the Differential Drive, create two Motor Controller Groups. These will group multiple motor controllers so they can be used with Differential Drive.



Finally, create two Motor Controllers in each Motor Controller Group.

将 Joystick 添加到 Operator Interface。



将两个操纵杆添加到 Operator Interface。其中一个为左杆，另一个为右杆。两个操纵杆的 y 轴是用来让机器人左右移动的。

备注： 注意在继续下一步前用 C++ 或 Java 输出程序。

建造一个 Method 来在 Subsystem 中 Write the Motor。

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

    /**
    *
    */
    public Drivetrain() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
left1 = new PWMVictorSPX(0);
```

(续下页)

(接上页)

```
 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() {
        // This method will be called once per scheduler run

    }

    @Override
```

(续下页)

(接上页)

```
    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};
frc::PWMVictorSPX m_left1{0};

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
public:
Drivetrain();
```

(续下页)

(接上页)

```

    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);


 AddChild("Right Motor", &m_rightMotor);


 AddChild("right2", &m_right2);
 m_right2.SetInverted(false);
```

(续下页)

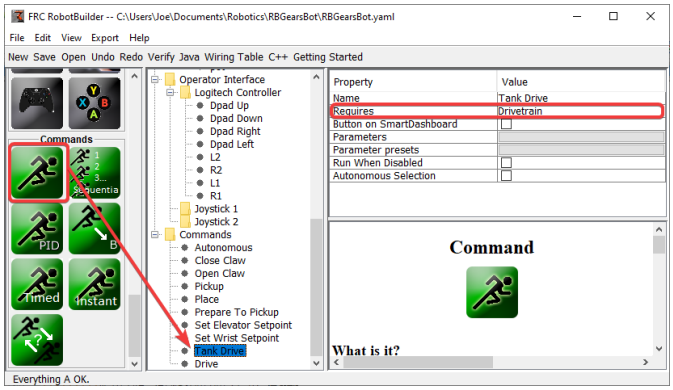
(接上页)

```
47
48  AddChild("right1", &m_right1);
49  m_right1.SetInverted(false);
50
51  AddChild("Left Motor", &m_leftMotor);
52
53
54  AddChild("left2", &m_left2);
55  m_left2.SetInverted(false);
56
57  AddChild("left1", &m_left1);
58  m_left1.SetInverted(false);
59
60      // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
61  }
62
63  void Drivetrain::Periodic() {
64      // Put code here to be run every loop
65
66  }
67
68  void Drivetrain::SimulationPeriodic() {
69      // This method will be called once per scheduler run when in simulation
70
71  }
72
73  // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS
74
75  // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CMDPIDGETTERS
76
77
78  // Put methods for controlling this subsystem
79  // here. Call these from Commands.
80
81      void Drivetrain::Drive(double left, double right) {
82          m_drive.TankDrive(left, right);
83      }
```

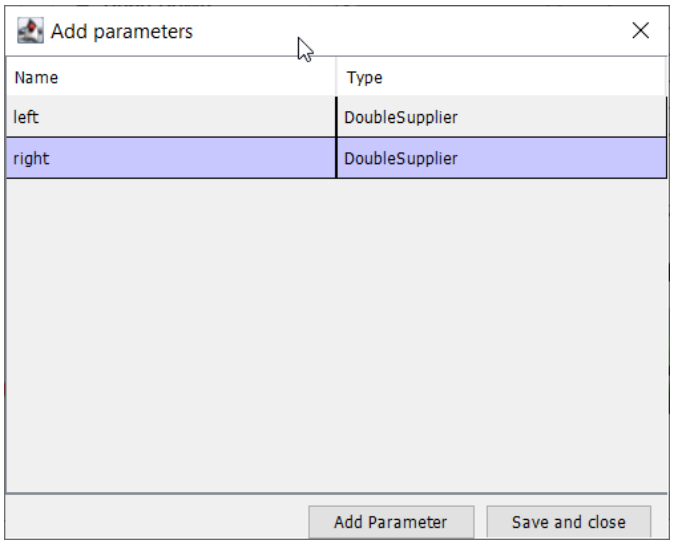
Create a method that takes the joystick inputs, in this case the left and right driver joystick. The values are passed to the DifferentialDrive object that in turn does tank steering using the joystick values. Also create a method called stop() that stops the robot from driving, this might come in handy later.

备注： 为清楚起见，此示例中删除了一些 RobotBuilder 输出

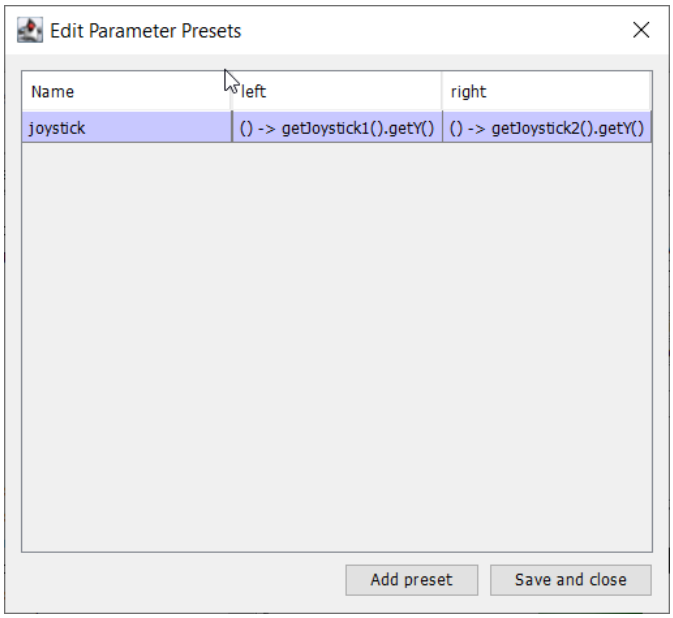
读取 Joystick Values 和 Call the Subsystem Methods 。



创建一个指令，在本例中称为 Tank Drive。它的目的是读取操纵杆值并将它们发送到 Drive Base 子系统。请注意，此指令需要 Drive Train 子系统。这将导致它在任何其他尝试使用 Drive Train 时停止运行。



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();}`

备注： 注意在继续下一步前用 C++ 或 Java 输出程序。

把这个代码加入到 Driving 中。

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

```

(续下页)

(接上页)

```
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS


    public TankDrive(Drivetrain subsystem) {


    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=VARIABLE_SETTING
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES

        m_drivetrain = subsystem;
        addRequirements(m_drivetrain);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
    }

    // Called when the command is initially scheduled.
    @Override
    public void initialize() {
    }

    // Called every time the scheduler runs while the command is scheduled.
    @Override
    public void execute() {
        m_drivetrain.drive(m_left.getAsDouble(), m_right.getAsDouble());
    }

    // Called once the command ends or is interrupted.
    @Override
    public void end(boolean interrupted) {
        m_drivetrain.drive(0.0, 0.0);
    }

    // Returns true when the command should end.
    @Override
    public boolean isFinished() {
        return false;
    }

    @Override
    public boolean runsWhenDisabled() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
        return false;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DISABLED
    }
}
```

C++ (Header)

```
// 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
};
```

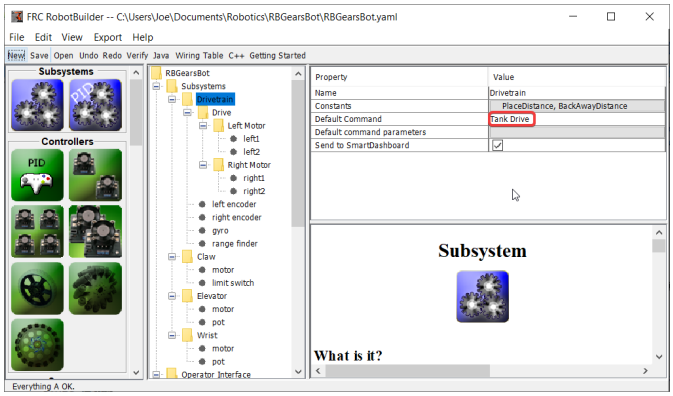
C++ (Source)

```
// 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
}
```

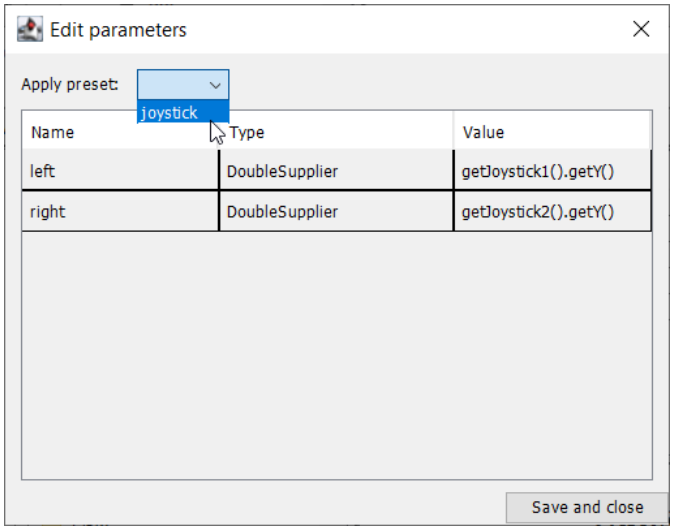
Add code to the execute method to do the actual driving. All that is needed is pass the for the left and right parameters to the Drive Train subsystem. The subsystem just uses them for the tank steering method on its DifferentialDrive object. And we get tank steering.

我们还填写了 `end()` 方法，以便当此指令被中断或停止时，电机将停止作为安全预防措施。

创建 **Default Command** 。



最后一步是使 Tank Drive 指令成为 Drive Train 子系统的“默认指令”。这意味着只要没有其他指令使用 Drive Train，操纵杆将处于控制之中。这可能是理想的行为。当自动阶段代码运行时，它还需要传动系统并中断 Tank Drive 指令。自动阶段代码完成后，DriveWithJoysticks 指令将自动重新启动（因为它是默认命令），操作员将重新获得控制权。如果您编写任何执行遥控自动驾驶的代码，这些指令也应该“需要” DriveTrain，以便它们也会中断 Tank Drive 指令并拥有完全控制权。



The final step is to choose the joystick parameter preset previously set up.

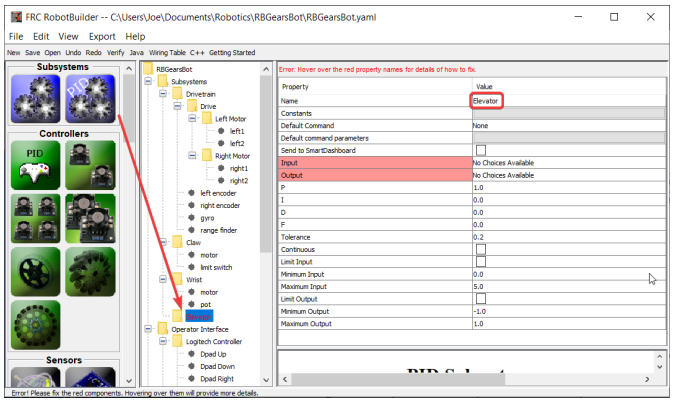
备注： 确保在继续前把你的程序导出到 C++ 或 Java。

21.3 机器人制造者-高级

21.3.1 使用 PID 子系统控制执行器

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.

创建一个 PID 子系统

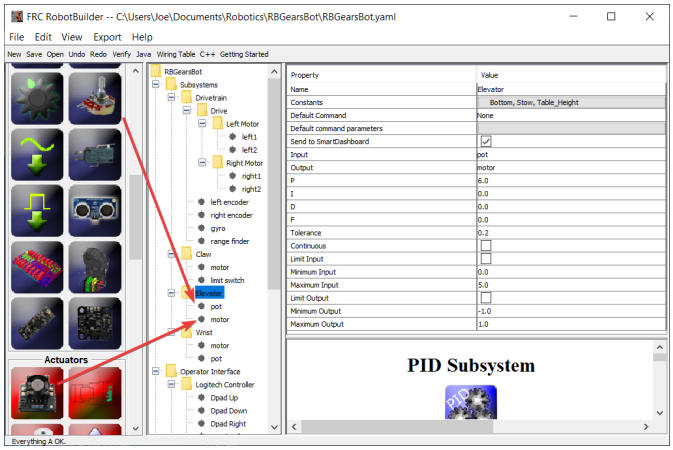


创建一个使用反馈来控制机制的位置或速度的子系统非常容易。

1. 将 PID 子系统从面板拖到机械手描述中的子系统文件夹中
2. 将 PID 子系统重命名为对子系统更有意义的名称，在本例中为升降机

请注意，机械手说明的某些部分已变为红色。这表明这些组件（PIDSubsystem）尚未完成，需要填写。缺失或不正确的属性以红色显示。

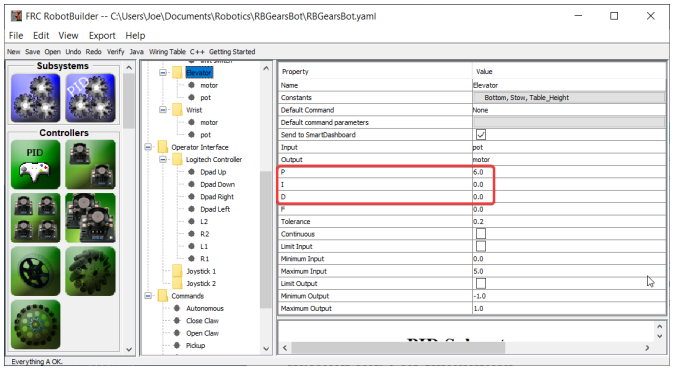
将传感器和执行器添加到 PID 子系统



为 PID 子系统添加缺少的组件

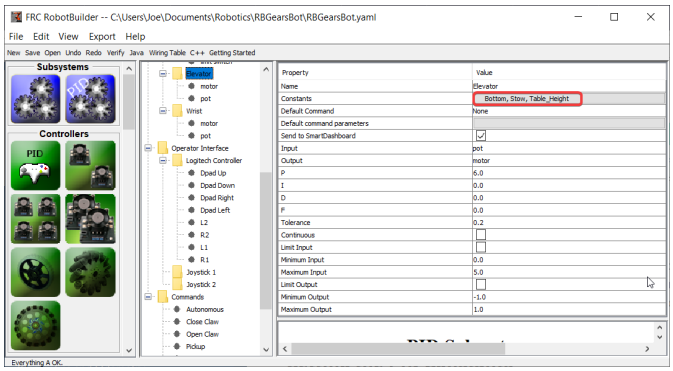
1. 将执行器（电机控制器）拖到特定的子系统中-在本例中为电梯
2. 将将用于反馈的传感器拖动到子系统，在这种情况下，该传感器是一个电位计，可提供电梯高度反馈。

填写 PID 参数

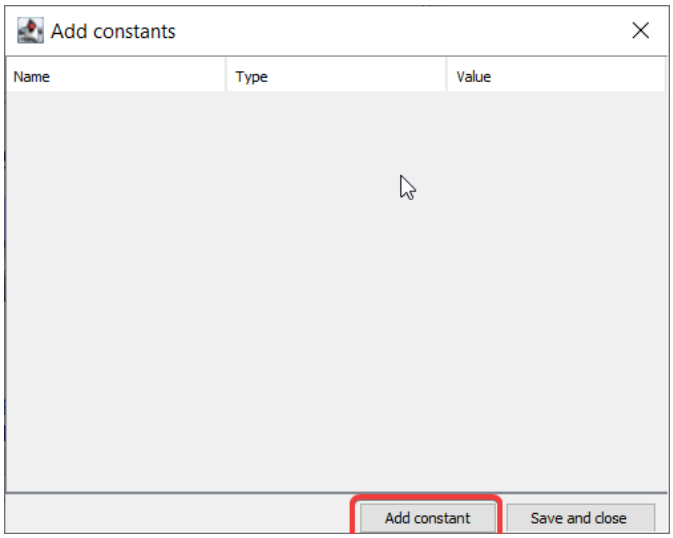


需要填写 P、I 和 D 值以获得所需的组件灵敏度和稳定性。对于我们的升降机，我们使用 6.0 和 0 的比例常数来表示 I 和 D 项。

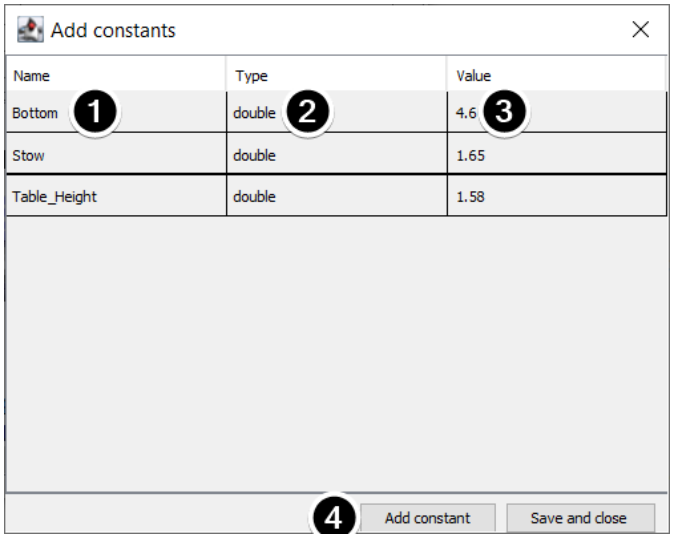
创建设定点常数



为了更轻松地管理升降机设定值，我们将创建常量来管理设定值。单击常量框以显示常量对话框。



单击:guilabel: 添加常量按钮



1. 填写常量的名称，在本例中：底部
2. 从下拉菜单中选择常量的类型，在本例中：双精度小数
3. 为常量选择一个值，在本例中为：4.65
4. 点击:guilabel:*add constant* 继续添加常量
5. After entering all constants, Click *Save and close*

21.3.2 编写 PID 子系统的代码

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.

确保已在 RobotBuilder 中创建了升降舵 PID 子系统。对于我们的升降舵，我们对 I 和 D 项使用 6.0 和 0 的比例常数。设置完毕后，请使用“导出”菜单或 Java / C ++ 工具栏菜单为项目生成 Java / C ++ 代码。

RobotBuilder generates the PIDSubsystem methods such that no additional code is needed for basic operation.

设置 PID 常数

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
}
```

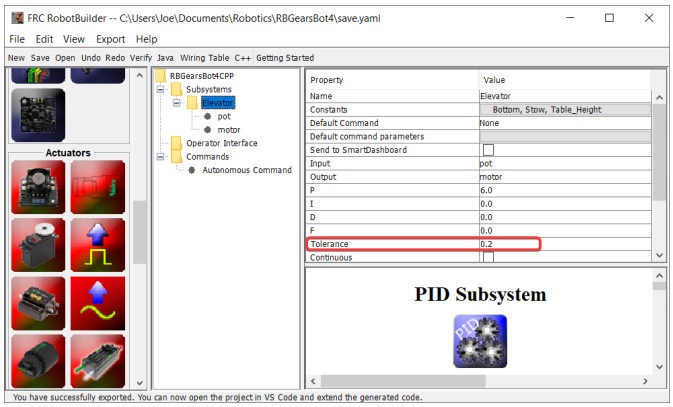
The `useOutput` method writes the calculated PID output directly to the motor.

这就是创建电梯 PIDSubsystem 所需要的所有内容。

21.3.3 设定点指令

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.

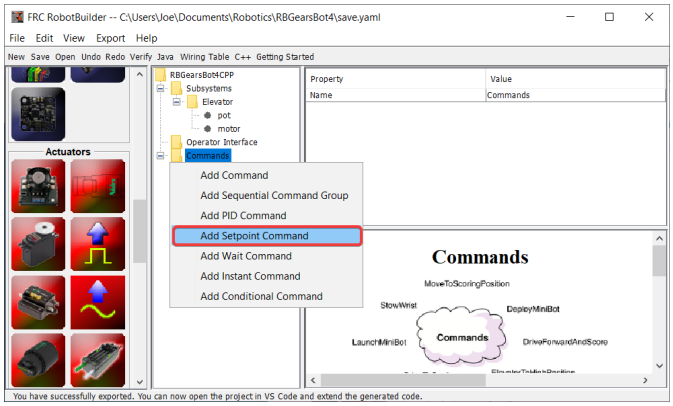
从 PIDSubsystem 开始



假设在机器人中有一个带有电位器的腕关节，该电位器可测量角度。首先，创建一个 PIDSubsystem <robotbuilder-writing-pidsubsystem-code>，其中包括用于移动腕关节的电动机和用于测量角度的电位计。PIDSubsystem 应该具有所有 PID 常数，并且可以正常工作。

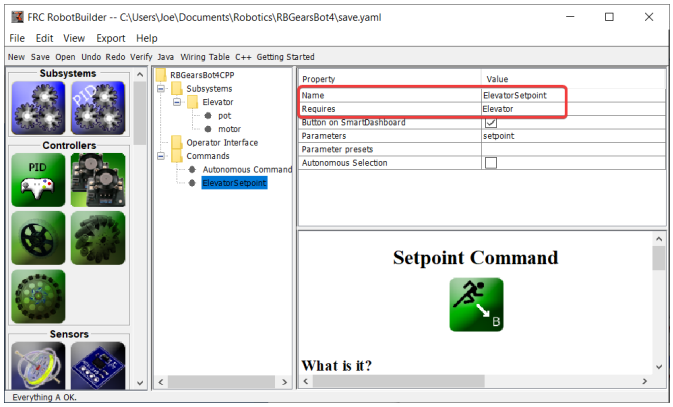
设置“公差”参数是很重要的。这可以控制当前值与设定值之间的距离，并且可以将其视为目标。这是 SetpointCommand 用于移至下一个指令的条件。

创建设定点指令

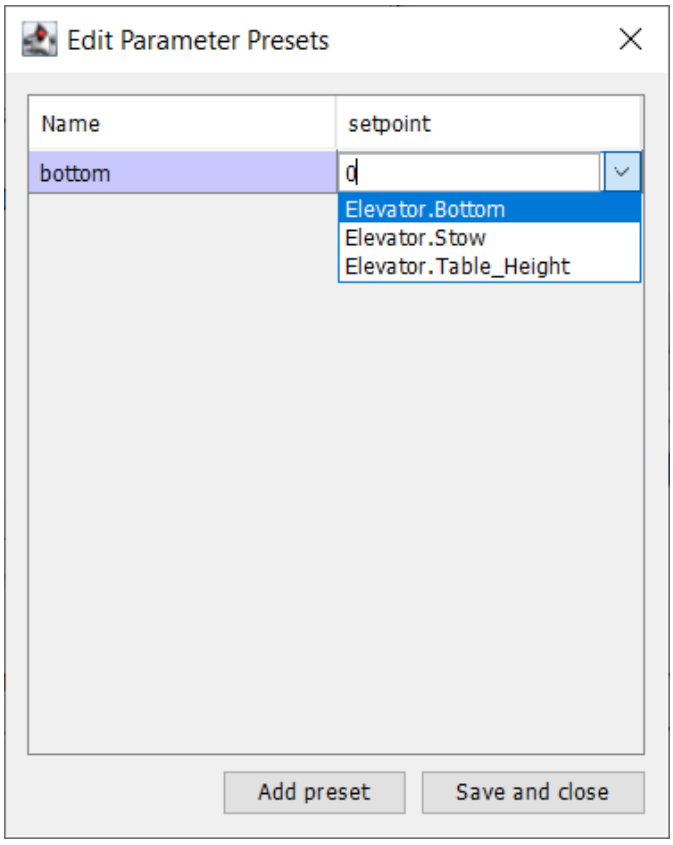


右键单击面板中的 Commands 文件夹，然后选择“添加设定点指令”。

设定点指令参数



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.

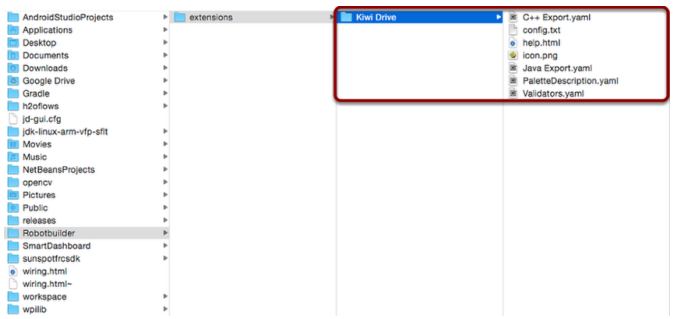
每当调度此指令时，它会自动将子系统驱动到指定的设定值。当达到设定值在 PIDSubsystem 中指定的公差范围内时，该指令结束，下一条指令开始。在 PIDSubsystem 中，指定公差很重要，否则该指令可能永远不会结束，因为无法达到公差。

备注: 有关 PID 控制的更多信息，请参阅高级控件简介 <docs/software/advanced-controls/introduction/index:Advanced Controls Introduction>。

21.3.4 添加自定义组件

RobotBuilder 在创建仅将 WPILib 用于电机，控制器和传感器的机器人程序时效果很好。但是对于使用自定义类的团队，RobotBuilder 对这些类没有任何支持，因此需要采取一些步骤才能在 RobotBuilder 中使用它们

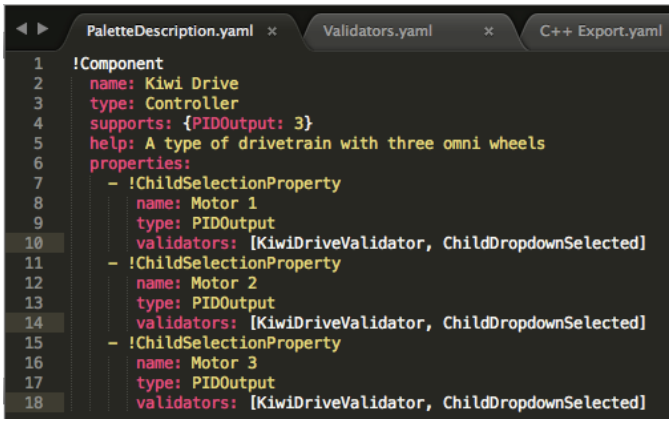
自定义组件结构



所有自定义组件都放在 ~/ wpilib / YYYY / Robotbuilder / extensions 中，其中🄴是 Windows 上的 C：Users Public，而 YYYY 是 FRC | reg |年。

自定义组件需要七个文件和一个文件夹。该文件夹包含描述组件及其导出方式的文件。该名称应与组件名称相同（例如，奇异果驱动器控制器 Kiwi drive controller 的名称应为 “Kiwi Drive”，六电动机驱动器控制器 six-motor drive controller 的名称应为 “Robot Drive 6”，等等) 这些文件应具有与此处显示的名称和扩展名相同的名称和扩展名。其他文件可以与这七个文件一起放在文件夹中，但是七个文件必须存在，以便 RobotBuilder 识别自定义组件。

PaletteDescription.yaml



逐行：

- ！Component：表明新组件的开始
- name：组件的名称。这就是调色板/树中将显示的内容-也应与包含文件夹的名称相同
- type：组件的类型（稍后将对此进行深入说明）
- supports：可以支持的每种类型的组件数量的图像。RobotBuilder 中的电动机控制器都是 PID 输出，因此 kiwi 驱动器可以支持三个 PID 输出。如果某个组件不支持任何东西（例如传感器或电机控制器），则请忽略此行
- help：短字符串，当这些组件之一被鼠标停留在之上时会给出有用的信息
- properties：此组件的属性列表。在此 kiwi 驱动器示例中，有三个非常相似的属性，每个电动机一个。ChildSelectionProperty 允许用户从正在编辑的子组件的子组件中选择给定类型的组件（因此，在这里，它们将显示一个下拉列表，要求输入已添加到 Kiwi 驱动器的 PID 输出-即电动机控制器）

RobotBuilder 支持的组件类型（区分大小写）：

- 指令
- 子系统
- PID 输出（电机控制器）
- PIDSource（实现 PIDSource 的传感器，例如模拟电位器，编码器）
- 传感器（无法执行 PIDSource 的传感器，例如限位开关）
- 控制器（机器人驱动器，PID 控制器等）
- 执行器（不是电动机的输出，例如电磁阀，伺服）
- 操作杆
- 操纵杆按钮

属性

与自定义组件相关的属性：

- StringProperty：在组件需要字符串时使用，例如组件名称
- BooleanProperty：当组件需要布尔值时使用在 SmartDashboard 上放置一个按钮
- DoubleProperty：在组件需要数字值使用，例如，PID 常数选择属性
- ChildSelectionProperty：当您需要选择子组件时使用，例如 RobotDrive 中的电机控制器
- TypeSelectionProperty：当您需要从程序中的任何位置选择给定类型的任何组件时使用，例如 PID 指令的输入和输出

每个属性的字段所述：

```
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.
```

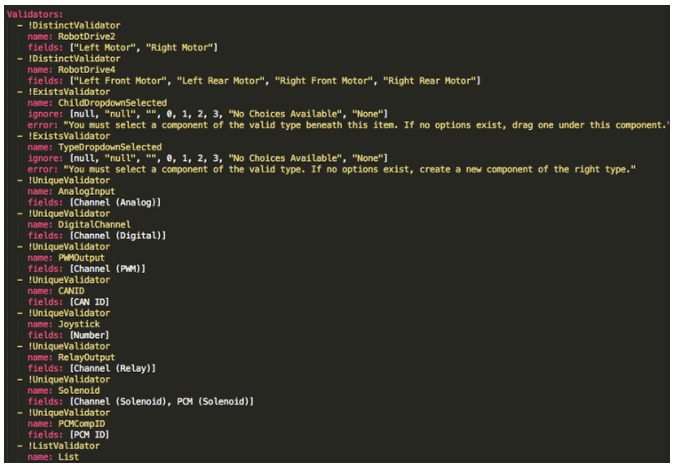
Validators.yaml

Validators.yaml

```
1  !DistinctValidator
2    name: KiwiDriveValidator
3    fields: ["Motor 1", "Motor 2", "Motor 3"]
```

您可能已经在 PaletteDescription.yaml 中每个电动机属性的验证器条目中注意到 “KiwiDriveValidator”。它不是内置的验证器，因此必须在 Validators.yaml 中定义。这个示例验证器非常简单-只需确保每个命名字段的值都与其他字段的不同。

内置验证器和验证器类型



内置验证器非常有用（特别是用于端口/通道的 UniqueValidators），但是有时需要自定义验证器，如上一步

- DistinctValidator：确保每个给定字段的值都是唯一的
- ExistsValidator：确保已使用此验证器为属性设置一个值
- UniqueValidator：确保该属性的值在给定字段中是全局唯一
- ListValidator：确保列表属性中的所有值均有效

C ++ 输出.yaml

文件的逐行细分：

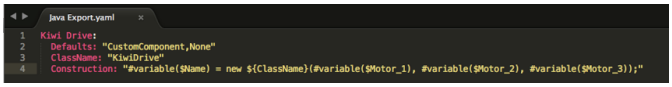
- Kiwi 驱动器：要导出的组件的名称。这与在 PaletteDescription.yaml 中设置的名称相同，也和包含此文件的文件夹的名称相同。
- 默认值：提供一些默认值，包括该组件所需的包含值，类的名称，构造模板等。默认情况下，CustomComponent 向使用该组件的每个生成的文件中添加“Custom / $ {ClassName} .h”的包含项（例如“RobotDrive.h”将具有“#include“Custom / KiwiDrive.h“文件的顶部）
- ClassName：您要添加的自定义类的名称。
- 构造：有关如何构造组件的说明。变量将被其值替换（“$ {ClassName}”将被替换为“Kiwi 驱动器”），然后将对宏进行求值（例如，“＃变量（$ Name）”可能会替换为“drivebaseKiwiDrive”）。

这个例子要求一个带有构造函数的 Kiwi 驱动器类型

```
KiwiDrive(SpeedController, SpeedController, SpeedController)
```

如果您的团队使用 Java，则此文件可以为空。

Java 导出.yaml



与 C ++ 导出文件非常相似；唯一的区别应该是施工线。施工线要求一个带有构造函数的 KiwiDrive 类型

```
KiwiDrive(SpeedController, SpeedController, SpeedController)
```

如果您的团队使用 C ++，则此文件可以为空。

使用宏和变量

宏是 RobotBuilder 用来将变量转换为文本的简单函数，这些文本将插入生成的代码中。它们始终以“#”符号开头，并且语法类似于函数：`<macro_name> (arg0, arg1, arg2, ...)`”。您可能需要使用的唯一宏是``#variable (component_name)`

“#variable”接受一个字符串，通常是在某个地方定义的变量（即“名称”是给 RobotBuilder 中的组件命名的名称，例如“手臂电机”），然后将其转换为在中定义的变量的名称。生成的代码。例如，`#variable (“Arm Motor”) ``产生字符串 “ArmMotor`

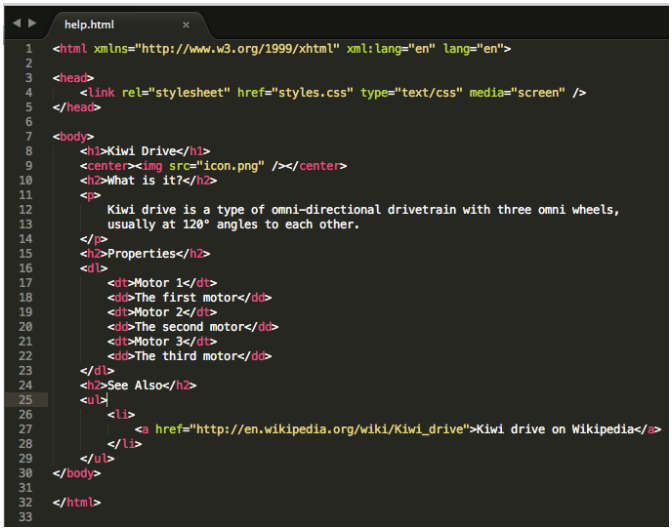
通过在变量名称前放置一个美元符号（“$”）来引用变量，可以将其可选地放在花括号内，以方便地将变量与文件中的其他文本区分开。解析文件后，将美元符号，变量名和花括号替换为变量的值（例如，将“${ClassName}”替换为“KiwiDrive”）。

变量可以是组件属性（例如，在 kiwi 驱动器示例中为“Motor 1”，“Motor 2”，“Motor 3”）或以下之一：

1. Short_Name：在 RobotBuilder 的编辑器面板中为组件指定的名称
2. Name：组件的全名。如果组件在子系统中，则这是在子系统名称后附加的简称
3. Export：应在其中创建该组件的文件名。对于执行器，控制器和传感器等组件，应为“RobotMap”；或对于如操作手柄及其他自定义 OI 组件时，应为“OI”。请注意，“CustomComponent”默认值将导出到 RobotMap。
4. Import：要使用此组件需要包含或导入的文件。
5. Declaration：类似于构造的指令，用于声明此组件类型的变量。默认情况下，此设置为“无”
6. Construction：有关如何创建此组件的新实例的说明
7. LiveWindow：有关如何将此组件添加到 LiveWindow 的说明
8. Extra：有关任何其他功能或方法的指令，要求此组件正常运行，例如需要设置编码类型的编码器。
9. Prototype（仅针对 C ++）：在声明组件的文件中创建的函数的原型，通常是 OI 类中的吸气剂
10. Function：要在声明组件的文件中创建的函数，通常是 OI 类中的吸气剂
11. PID：如果有组件，则是一个获取组件的 PID 输出的指令（例如“#variable（$ Short_Name）-> PIDGet ()”）
12. ClassName：组件代表的类的名称（例如“Kiwi 驱动器”或“操作杆”）

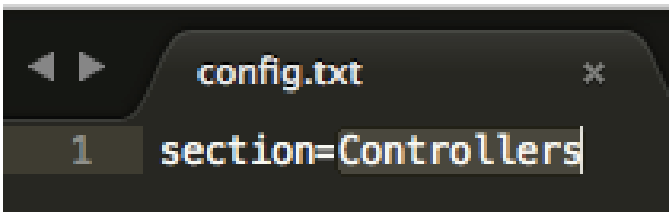
如果您的变量名称中带有空格（例如“Motor 1”，“Right Front Motor”等），则在导出文件中使用空格时，需要用下划线替换空格。

help.html



一个 HTML 文件，提供有关组件的信息。最好使它尽可能详细，尽管如果这一个（或多个）程序员对组件足够熟悉，或者它很简单以至于在详细描述中没什么用处时，这个是肯定没有必要的。

config.txt



一个配置文件，用于保存有关组件的其他信息。当前，只有调色板的该部分可以放入组件。

面板的各部分（区分大小写）：

- Subsystems 子系统
- Controllers 控制器
- Sensors 感测器
- Actuators 执行器
- Pneumatics 气动式
- OI
- Commands 指令

icon.png

显示在选用板和帮助页面中的图标。这应该是 64x64 的.png 文件。

它应该使用该部分的配色方案和常规样式，以避免视觉混乱，但这是完全可选的。图标和背景的 Photoshop“.psd’’ 文件位于 “src / main / icons / icons <https://github.com/wpilibsuite/RobotBuilder/tree/master/src/main/icons/icons>‘_ 中，图标和背景的 png 文件位于 “src / main / resources / icons <https://github.com/wpilibsuite/RobotBuilder/tree/master/src/main/resources/icons>‘_ 中。

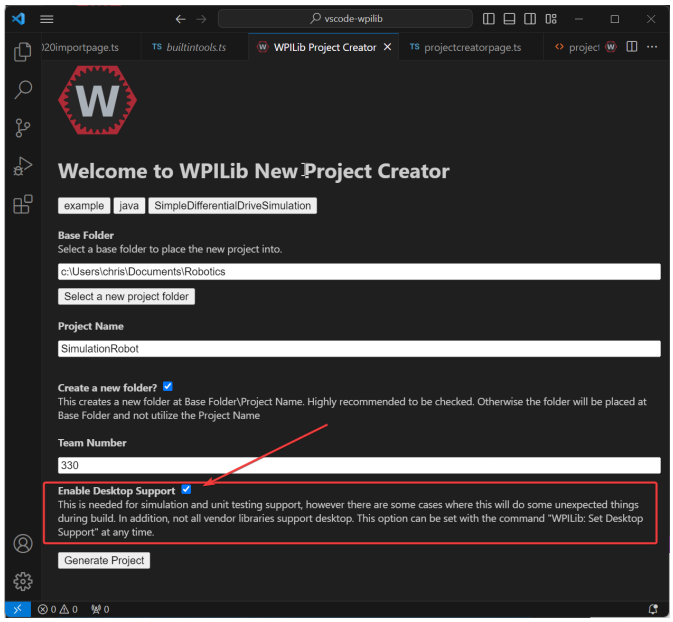
22 机器人模拟

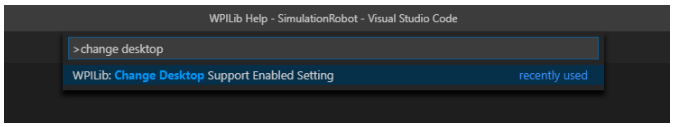
22.1 机器人模拟简介

通常，团队可能想在没有可用的实际机器人的情况下测试其代码。WPILib 使团队能够使用简单的 gradle 命令来模拟各种机器人功能。

Java/C++

指令使用桌面模拟器需要启用桌面支持。这可以通过在创建机器人项目时选中“启用桌面支持复选框”或通过运行 Visual Studio Code 命令面板中的“WPILib：更改启用桌面支持设置”来完成。





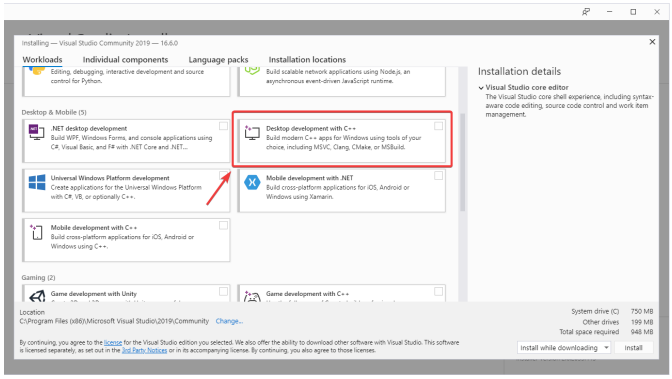
也可以通过手动编辑位于机器人项目根目录下的 build.gradle 文件来启用桌面支持。只需将 “includeDesktopSupport = false’’ 更改为 “includeDesktopSupport = true’’

重要： 重要的是要注意启用桌面/模拟支持会产生意想不到的后果。并非所有供应商都支持此选项，并且使用它们的库的代码甚至可能在尝试运行模拟时崩溃！

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.

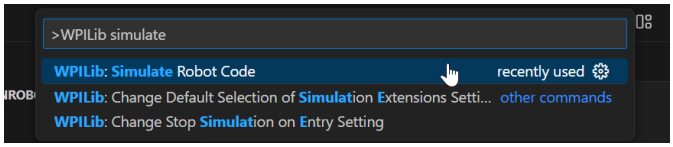
备注： C++ robot simulation requires that a native compiler to be installed. For Windows, this would be Visual Studio 2022 version 17.9 or later (**not** VS Code), macOS requires Xcode 14 or later, and Linux (Ubuntu) requires the `build-essential` package.

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



Running Robot Simulation

可以使用 VS Code 运行基本的机器人仿真。通过使用 VS Code 的命令选项板，无需使用任何命令即可完成此操作。



Visual Studio Code 中的控制台输出应如下所示。但是，团队可能想要实际 * 测试 * 他们的代码，而不是仅仅运行模拟。可以使用 WPILib 的 Simulation GUI <simulation-gui>‘来完成。

```
********** Robot program starting **********
Default disabledInit() method... Override me!
Default disabledPeriodic() method... Override me!
Default robotPeriodic() method... Override me!
```

重要： Simulation can also be run outside of VS Code using `./gradlew simulateJava` for Java or `./gradlew simulateNative` for C++.

备注： 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

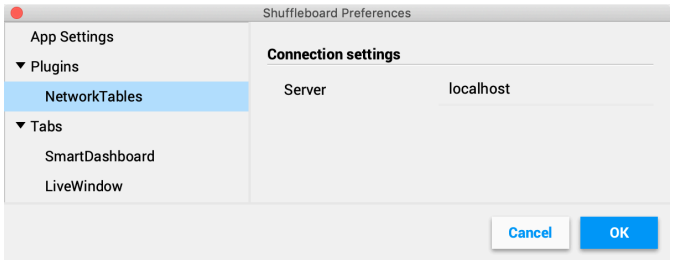
```
python3 -m robotpy sim
```

22.1.1 运行机器人面板

Shuffleboard, SmartDashboard, Glass, and AdvantageScope can be used with WPILib simulation when they are configured to connect to the local computer (i.e. `localhost`).

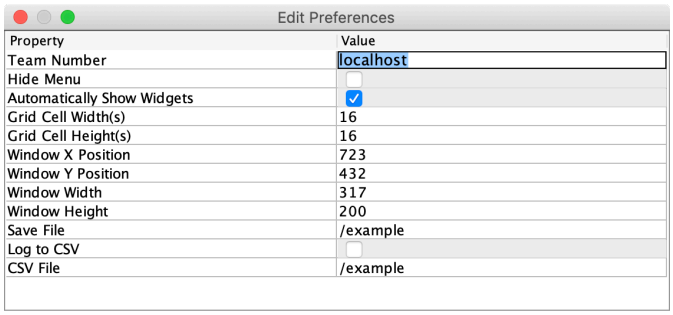
Shuffleboard

Shuffleboard is automatically configured to look for a NetworkTables instance from the robotRIO 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`.



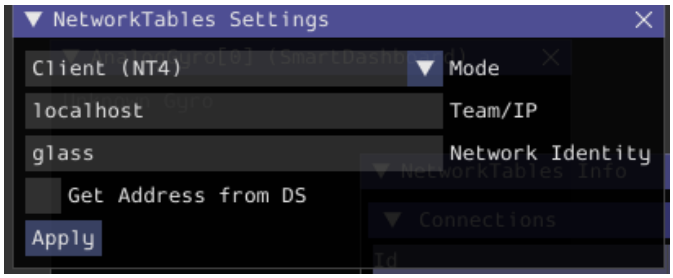
智能仪表板

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`.



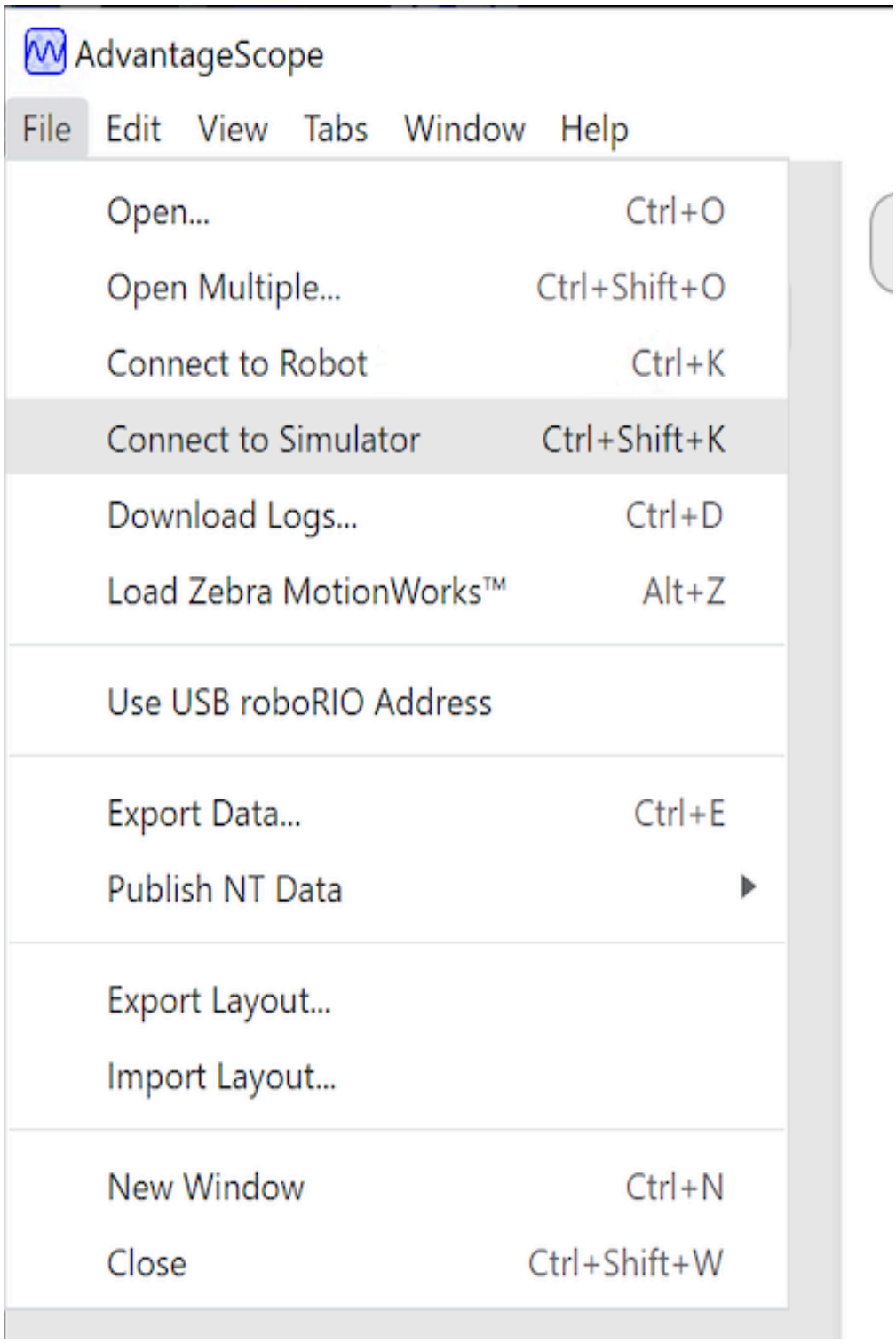
Glass

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`.

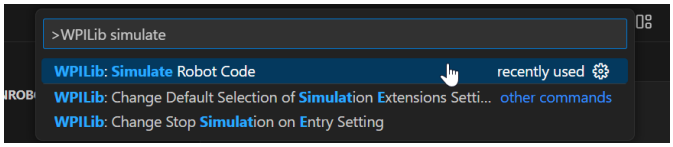

AdvantageScope
File
Edit
View
Tabs
Window
Help
Open... Ctrl+O
Open Multiple... Ctrl+Shift+O
Connect to Robot Ctrl+K
Connect to Simulator Ctrl+Shift+K
Download Logs... Ctrl+D
Load Zebra MotionWorks™ Alt+Z
Use USB roboRIO Address
Export Data... Ctrl+E
Publish NT Data
Export Layout...
Import Layout...
New Window Ctrl+N
Close Ctrl+Shift+W

22.2 模拟特定的用户界面元素

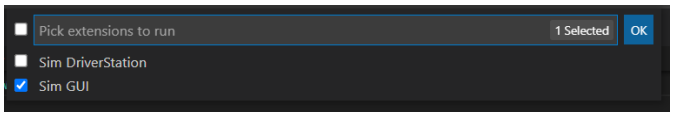
WPILib 扩展了机器人仿真功能，以引入图形用户界面（GUI）组件。这使团队可以轻松地可视化其机器人的输入和输出。

备注： 模拟 GUI 在许多方面与:ref:*Glass <docs/software/dashboards/glass/introduction:Introduction to Glass>* 非常相似。下面的一些页面将链接到 Glass 部分，这些部分描述了两种 GUI 的共同元素。

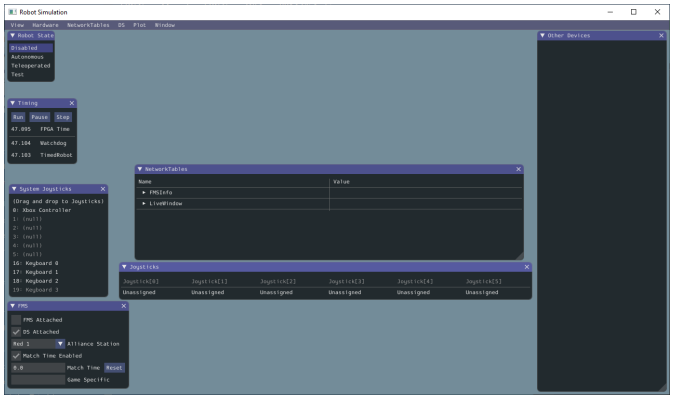
22.2.1 运行 GUI



您只需通过 ** 运行模拟 ** 命令面板选项启动 GUI。

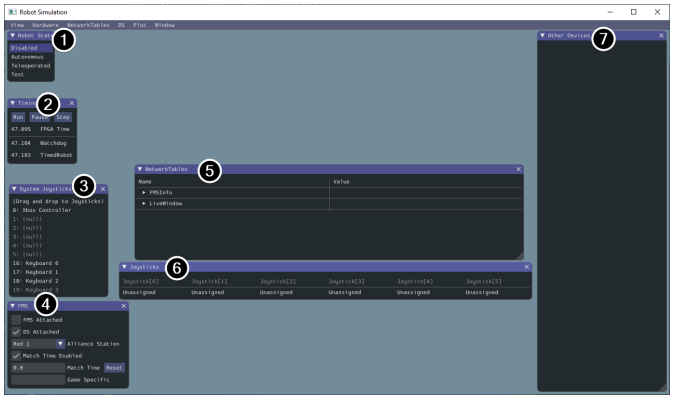


And the `Sim GUI` option should popup in a new dialog and will be selected by default. Press *Ok*. This will now launch the Simulation GUI!



22.2.2 使用 GUI

学习布局



默认情况下，模拟 GUI 上会显示以下项目：

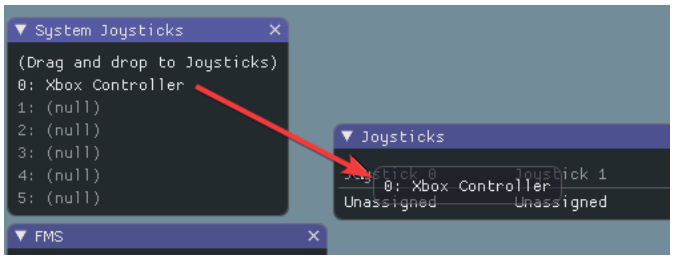
1. **机器人状态**-这是机器人的当前状态或“模式”。你可以单击标签来更改模式，就像在普通 Driver Station 上一样。
2. **计时** - 显示机器人计时器的值并允许操纵计时。
3. **系统操纵杆**-这是当前连接到系统的操纵杆列表。
4. **FMS** - This is used for simulating many of the common *FMS* systems.
5. **NetworkTables** - 显示已发布到 NetworkTables 的数据。
6. **操纵杆**-这是可以直接从机器人代码中拉出的操纵杆。
7. **其他设备** - 这包括不属于任何其他类别的设备，例如部件套件中包含的 ADXRS450 陀螺仪或支持模拟的第三方设备。

可以从硬件菜单添加以下项目，但默认情况下不显示。

1. **Addressable LEDs** - 这显示了由“AddressableLED”类控制的 LED。
2. **模拟输入**-包括通常使用 roboRIO 上的 ** ANALOG IN **连接器的任何设备，例如任何基于模拟的陀螺仪。
3. ** DIO **-（**Digital Input Output 数字输入输出）这包括使用 roboRIO 上的** DIO **连接器的所有设备。
4. ** Encoders **-这将显示扩展或使用 Encoder 类的所有实例化设备。
5. **PDPs** - 显示配电面板对象。
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. **继电器**-这包括任何继电器设备。这包括 VEX Spike 继电器。
8. **螺线管**-这是“已连接”螺线管的列表。当你创建电磁对象并推动输出时，此处显示这些。

将系统操纵杆添加到操纵杆

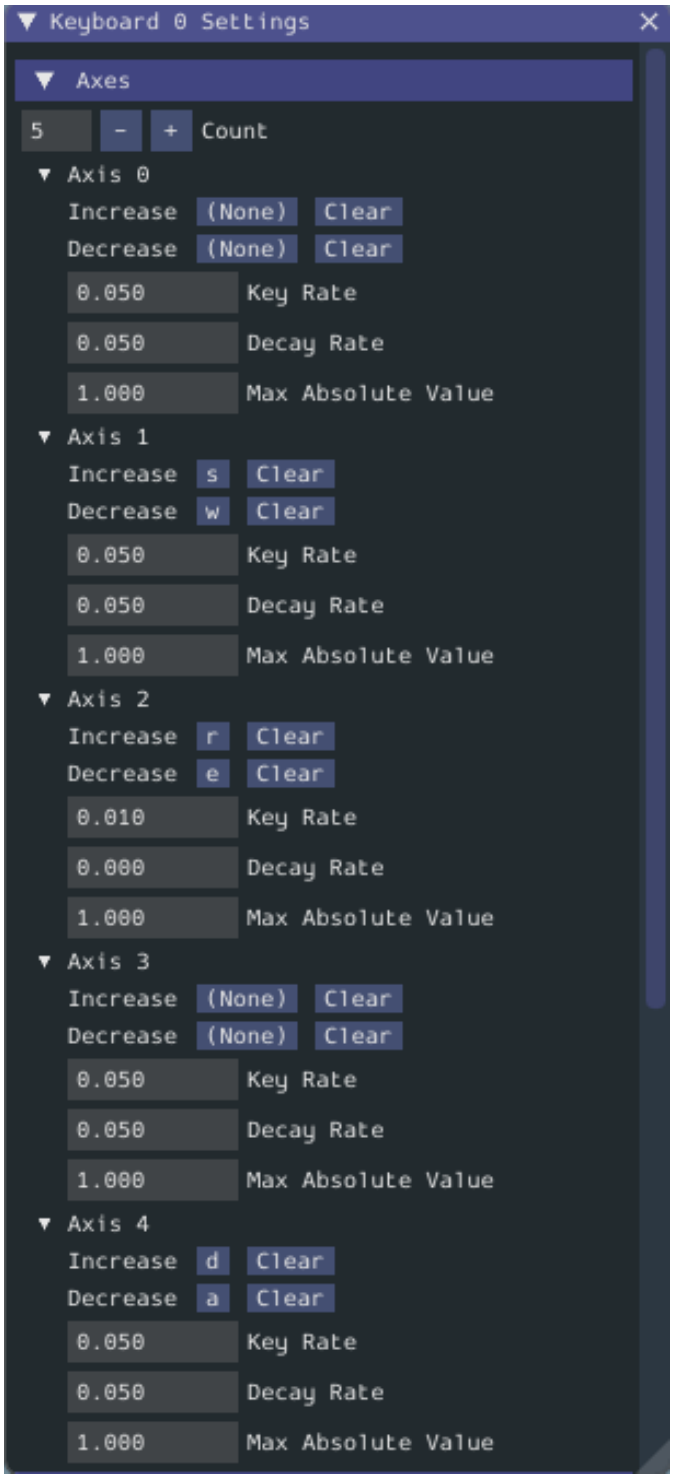
要从系统操纵杆列表中添加操纵杆，只需在“系统操纵杆”菜单下单击并拖动显示的操纵杆到“操纵杆”菜单。



备注： FRC | reg | Driver Station 对连接的游戏手柄进行特殊映射，并且 WPILib 仿真器默认不会“映射”这些手柄。您可以通过按“操纵杆”菜单下的“映射手柄”开关来打开此行为。

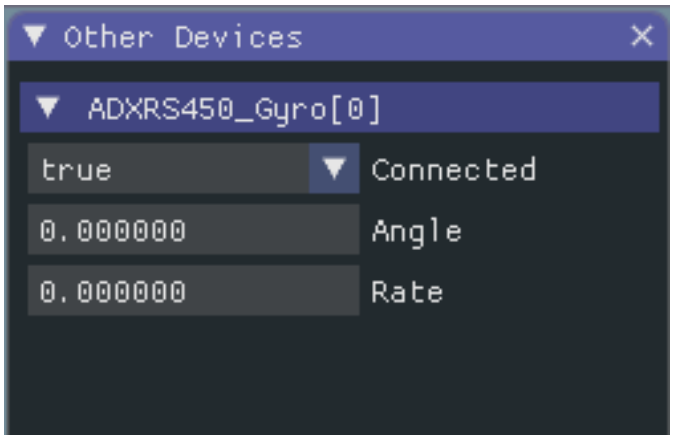
将键盘用作操纵杆

您可以通过单击并拖动键盘项目之一（例如键盘 0）来将键盘添加到系统操纵杆列表中，就像上面的操纵杆一样。要编辑键盘设置，请转到菜单栏中的 DS 项，然后选择键盘 0 设置。这允许您控制哪个键盘按钮控制哪个轴。这是如何使键盘类似于 Xbox 控制器（使用轴 1 和 4）上的分体式街机驱动器的常见示例：


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

修改 ADXRS450 输入

使用 ADXRS450 对象是测试基于陀螺仪的输出的绝佳方法。这将显示在“其他设备”菜单中。然后会显示一个下拉菜单，其中显示了各种选项，例如“Connected”，“Angle”和“Rate”。所有这些值都是可以更改的值，并且是你的机器人代码并可以即时使用。



22.2.3 根据机器人代码确定模拟

如果在运行机器人仿真时供应商库无法编译，则可以使用“RobotBase.isReal ()”包装它们的内容，这将返回“布尔值”。

JAVA

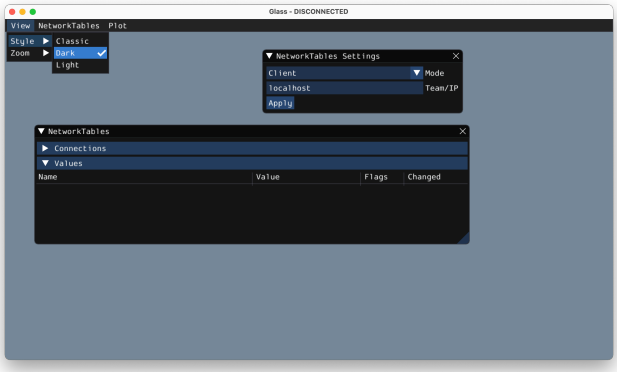
```
TalonSRX motorLeft;
TalonSRX motorRight;

public Robot() {
 if (RobotBase.isReal()) {
   motorLeft = new TalonSRX(0);
   motorRight = new TalonSRX(1);
 }
}
```

备注： 在 C ++ 中，重新分配值类型需要移动或复制分配。除非使用指针而不是值类型，否则既不支持 SIM 又缺少移动或副本分配运算符的供应商类无法通过条件分配解决。

22.2.4 更改视图设置

“视图”菜单项包含可以自定义的“缩放”和“样式”设置。“缩放”选项决定了应用程序中文本的大小，而“样式”选项则允许您在“标准”，“亮”和“暗”模式之间进行选择.

深色样式设置的例子如下：



22.2.5 清除申请数据

模拟 GUI 的应用程序数据，包括小部件大小和位置以及小部件的其他自定义信息，都存储在“imgui.ini”文件中。该文件存储在运行模拟的项目目录的根目录中。

`imgui.ini` 配置文件可以简单地删除以将模拟 GUI 恢复到“干净的状态”。

22.3 用 WPILib 进行物理模拟

因为：ref：状态空间表示法‘允许我们紧凑地表示：术语：‘系统的动力学，我们可以利用它为模拟机器人上的物理系统提供后端。这些模拟器的目的是在不修改现有的非仿真用户代码的情况下模拟机器人机构的运动。这种模拟器的基本流程如下：

- 在正常用户代码中：
  - PID 或类似的控制算法从编码器（或其他传感器）读数中生成电压命令
  - 电机输出被设置
- 在模拟周期代码中：
  - 模拟的：术语：状态使用中：术语：‘输入‘更新的，通常是来自 PID 回路设置的电机的电压
  - 模拟编码器（或其他传感器）读数被设置为用户代码在下一个时间步骤中使用

22.3.1 WPILib 的模拟类

WPILib 提供以下物理模拟类：

- LinearSystemSim，用于具有线性动力学的系统建模
- FlywheelSim
- DifferentialDrivetrainSim
- ElevatorSim, which models gravity in the direction of elevator motion
- SingleJointedArmSim, which models gravity proportional to the arm angle
- BatterySim，仅根据汲取的电流估算电池电压骤降

所有模拟类（差速驱动器模拟器除外）都继承自:code:*LinearSystemSim*‘类。默认情况下，动力学是线性系统动力学:*math:*mathbf{x}_{k+1} = mathbf{A}mathbf{x}_k + mathbf{B}mathbf{u}_k‘。子类重写:code:*UpdateX(x, u, dt)* 方法，以提供自定义的非线性动力学，例如对重力建模。

备注： 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 用户代码中的用法

WPILib 中的示例项目 <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/elevatorsimulation>‘__ 提供了以下内容。

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.

备注： 电梯和手臂模拟器将防止模拟位置超过给定的最小或最大高度或角度。如果要模拟具有无限旋转或运动的机构，: code:‘LinearSystemSim‘可能是更好的选择。

Java

```
47  // Simulation classes help us simulate what's going on, including gravity.
48  private final ElevatorSim m_elevatorSim =
49      new ElevatorSim(
50          m_elevatorGearbox,
51          Constants.kElevatorGearing,
52          Constants.kCarriageMass,
53          Constants.kElevatorDrumRadius,
54          Constants.kMinElevatorHeightMeters,
55          Constants.kMaxElevatorHeightMeters,
56          true,
```

(续下页)

(接上页)

```
        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};
```

下一步，代码："遥循环" /代码："遥循环"（java / C++）使用简单的 PID 控制回路来驱动我们的电梯到离地面 30 英寸的设置点。

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);
}
```

其次，：code：*simulationPeriodic*/:code：‘SimulationPeriodic‘(JAVA/C++) 利用施加在电机上的电压来更新电梯的模拟位置。我们使用的是：代码：“模拟周期”，因为它只为模拟机器人周期性地运行。这意味着我们的模拟代码不会在真正的机器人上运行。

备注： 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`.

最后，利用模拟电梯的位置设置模拟编码器的距离读数，利用电梯绘制的估计电流设置机器人的电池电压。

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 设备仿真

WPILib 提供了一种以 SimDevice API 形式管理仿真设备数据的方法。

22.4.1 模拟核心 WPILib 设备类

核心 WPILib 设备类（即 “Encoder’’, “Ultrasonic’’ 等）具有名为 “EncoderSim’’, “UltrasonicSim’’ 等的仿真类。这些类允许与设备数据进行交互，而这些交互在模拟之外是不可能的或无效的。在模拟之外构造它们可能不会干扰您的代码，但是调用它们的函数等是未定义的行为-最好的情况下，它们什么都不做，更糟的情况可能会使您的代码崩溃！将功能仿真代码放在仅仿真功能中（例如 simulationPeriodic（））或用 RobotBase.isReal（）/ RobotBase :: IsReal（）检查将它们包装起来（即 constexpr“在 C ++ 中）。

备注： 本示例将使用 EncoderSim 类作为示例。其他模拟类的使用几乎是相同的。

创建模拟设备对象

可以通过两种方式构造仿真设备对象：

- 接受常规硬件对象的构造函数。
- 接受设备连接到的端口/索引/通道号的构造函数或工厂方法。这些数字将与用于构造常规硬件对象的数字相同。这对于单元测试 <unit-testing> 尤其有用。

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};
```

读写设备数据

每个模拟类对于每个字段 “Xxx’’ 都有 getter（getXxx () /''GetXxx ()）和 setter（setXxx（value）/SetXxx（value））函数 “。getter 函数将返回与常规设备类的 getter 相同的函数。

JAVA

```
simEncoder.setCount(100);
encoder.getCount(); // 100
simEncoder.getCount(); // 100
```

C++

```
simEncoder.SetCount(100);
encoder.GetCount(); // 100
simEncoder.GetCount(); // 100
```

注册回调

除了 getter 和 setter 之外，每个字段还具有一个 registerXxxCallback () 函数，该函数注册一个回调函数，只要该字段值发生更改并返回 CallbackStore 对象，该回调函数便会运行。回调接受字段名称的字符串参数和包含新值的 “HALValue” 对象。从 “HALValue’’ 中获取值之前，请检查包含的值的类型。可能的类型是 “HALValue.kBoolean’’ /HAL_BOOL''，``HALValue.kDouble''/``HAL_DOUBLE''，``HALValue.kEnum''/``HAL_ENUM''，``HALValue.kInt ``/``HAL_INT，HALValue.kLong /HAL_LONG。

在 Java 中，对 CallbackStore 对象调用 close () 取消回调。保留对对象的引用，以免垃圾被回收-否则回调将被 GC 取消。要向回调提供任意数据，请在 lambda 中捕获它或使用方法引用。

在 C ++ 中，将 “CallbackStore” 对象保存在正确的范围内-当对象超出范围并被销毁时，回调将被取消。可以通过 param 参数将任意数据传递给回调。

> **警告：** 尝试从包含其他类型的 HALValue 中检索类型的值是未定义的行为。

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 模拟其他设备-SimDeviceSim 类

备注: 供应商可能会实现与 SimDevice API 的连接，与此处所述稍有不同。他们可能还会提供特定于其设备类的仿真类。有关他们支持什么以及如何获得更多信息，请参阅供应商的文档。

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*.

使用与厂商用来在其设备类中构造基础 "SimDevice'' 的键相同的字符串键创建 "SimDeviceSim'' 对象。该密钥是设备在 "其他设备'' 选项卡中显示的密钥，通常为 "前缀：设备名称 [索引]'' 的形式。如果键包含端口/索引/通道号，则可以将它们作为单独的参数传递给 "SimDeviceSim" 构造函数。该密钥包含一个默认情况下在 SimGUI 中隐藏的前缀，可以通过选择：guilabel：'Show prefix'选项来显示。在传递给 "SimDeviceSim"的密钥中不包含此前缀将与设备不匹配!

JAVA

```
SimDeviceSim device = new SimDeviceSim(deviceKey, index);
```

C++

```
frc::sim::SimDeviceSim device{deviceKey, index};
```

一旦有了 “SimDeviceSim”, 就可以获取代表设备字段的 “SimValue” 对象。特定类型的 “SimDouble’’, “SimInt’’, “SimLong’’, “SimBoolean’’ 和 “SimEnum“子类也存在, 并且应该使用它们来代替类型不安全的 “SimValue “课。这些是使用与供应商用来定义该字段的字符串相同的字符串键从 “SimDeviceSim” 构造的。该键是该字段在 SimGUI 中显示的键。尝试在模拟之外或设备或字段键不匹配时检索 “SimValue” 对象将返回 “null” -这可能会导致 Java 中出现 “NullPointerException” 或 C ++ 中出现未定义的行为。

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 Drivetrain Simulation Tutorial

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++) uses a more traditional approach to data flow.

Both of these examples are also available in the VS Code *New Project* window.

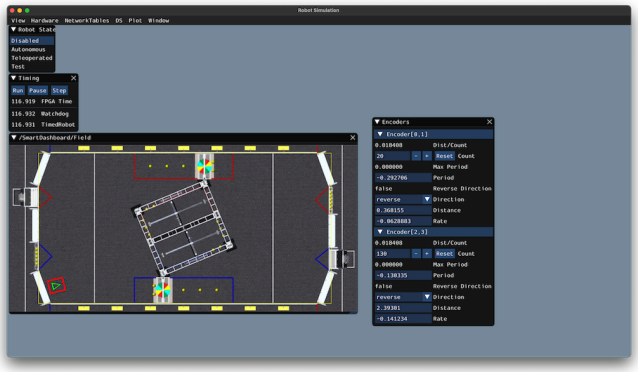
22.5.1 Drivetrain Simulation Overview

备注: The code in this tutorial does not use any specific framework (i.e. command-based vs. simple data flow); however, guidance will be provided in certain areas for how to best implement certain pieces of code in specific framework types.

The goal of this tutorial is to provide guidance on implementing simulation capabilities for a differential-drivetrain robot. By the end of this tutorial, you should be able to:

1. Understand the basic underlying concepts behind the WPILib simulation framework.
2. Create a drivetrain simulation model using your robot’s physical parameters.

3. Use the simulation model to predict how your real robot will move given specific voltage inputs.
4. Tune feedback constants and squash common bugs (e.g. motor inversion) before having access to physical hardware.
5. Use the Simulation GUI to visualize robot movement on a virtual field.



Why Simulate a Drivetrain?

The drivetrain of a robot is one of the most important mechanisms on the robot –therefore, it is important to ensure that the software powering your drivetrain is as robust as possible. By being able to simulate how a physical drivetrain responds, you can get a head start on writing quality software before you have access to the physical hardware. With the simulation framework, you can verify not only basic functionality, like making sure that the inversions on motors and encoders are correct, but also advanced capabilities such as verifying accuracy of path following.

22.5.2 Step 1: Creating Simulated Instances of 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.

备注： Simulation objects associated with a particular subsystem should live in that subsystem. An example of this is in the `StateSpaceDriveSimulation` (Java, C++) example.

Simulating Encoders

The `EncoderSim` class allows users to set encoder positions and velocities on a given `Encoder` object. When running on real hardware, the `Encoder` class interacts with real sensors to count revolutions (and convert them to distance units automatically if configured to do so); however, in simulation there are no such measurements to make. The `EncoderSim` class can accept these simulated readings from a model of your drivetrain.

备注: It is not possible to simulate encoders that are directly connected to CAN motor controllers using WPILib classes. For more information about your specific motor controller, please read your vendor's documentation.

JAVA

```
// These represent our regular encoder objects, which we would
// create to use on a real robot.
private Encoder m_leftEncoder = new Encoder(0, 1);
private Encoder m_rightEncoder = new Encoder(2, 3);

// These are our EncoderSim objects, which we will only use in
// simulation. However, you do not need to comment out these
// declarations when you are deploying code to the roboRIO.
private EncoderSim m_leftEncoderSim = new EncoderSim(m_leftEncoder);
private EncoderSim m_rightEncoderSim = new EncoderSim(m_rightEncoder);
```

C++

```
#include <frc/Encoder.h>
#include <frc/simulation/EncoderSim.h>

...

// These represent our regular encoder objects, which we would
// create to use on a real robot.
frc::Encoder m_leftEncoder{0, 1};
frc::Encoder m_rightEncoder{2, 3};

// These are our EncoderSim objects, which we will only use in
// simulation. However, you do not need to comment out these
// declarations when you are deploying code to the roboRIO.
frc::sim::EncoderSim m_leftEncoderSim{m_leftEncoder};
frc::sim::EncoderSim m_rightEncoderSim{m_rightEncoder};
```

Simulating Gyroscopes

Similar to the `EncoderSim` class, simulated gyroscope classes also exist for commonly used WPILib gyros –`AnalogGyroSim` and `ADXRS450_GyroSim`. These are also constructed in the same manner.

备注: 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 Step 2: Creating a Drivetrain Model

In order to accurately determine how your physical drivetrain will respond to given motor voltage inputs, an accurate model of your drivetrain must be created. This model is usually created by measuring various physical parameters of your real robot. In WPILib, this drivetrain simulation model is represented by the `DifferentialDrivetrainSim` class.

Creating a `DifferentialDrivetrainSim` from Physical Measurements

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:

- The type and number of motors on one side of the drivetrain.
- 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).
- The mass of the drivetrain (it is recommended to use the mass of the entire robot itself, as it will more accurately model the acceleration characteristics of your robot for trajectory tracking).
- The radius of the drive wheels.
- The track width (distance between left and right wheels).
- Standard deviations of measurement noise: this represents how much measurement noise you expect from your real sensors. The measurement noise is an array with 7 elements, with each element representing the standard deviation of measurement noise in x, y, heading, left velocity, right velocity, left position, and right position respectively. This option can be omitted in C++ or set to `null` in Java if measurement noise is not desirable.

You can calculate the measurement noise of your sensors by taking multiple data points of the state you are trying to measure and calculating the standard deviation using a tool like Python. For example, to calculate the standard deviation in your encoders' velocity estimate, you can move your robot at a constant velocity, take multiple measurements, and calculate their standard deviation from the known mean. If this process is too tedious, the values used in the example below should be a good representation of average noise from encoders.

备注: The standard deviation of the noise for a measurement has the same units as that measurement. For example, the standard deviation of the velocity noise has units of m/s.

备注: It is very important to use SI units (i.e. meters and radians) when passing parameters in Java. In C++, the *units library* can be used to specify any unit type.

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.
```

(续下页)

(接上页)

```
  // 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.

重要: 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.

This constructor takes the following parameters:

- A linear system representing the drivetrain –this can be created using the identification gains.
- The track width (distance between the left and right wheels).
- The type and number of motors on one side of the drivetrain.
- 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 radius of the drive wheels.
- Standard deviations of measurement noise: this represents how much measurement noise you expect from your real sensors. The measurement noise is an array with 7 elements, with each element representing the standard deviation of measurement noise in x, y, heading, left velocity, right velocity, left position, and right position respectively. This option can be omitted in C++ or set to `null` in Java if measurement noise is not desirable.

You can calculate the measurement noise of your sensors by taking multiple data points of the state you are trying to measure and calculating the standard deviation using a tool like Python. For example, to calculate the standard deviation in your encoders' velocity estimate, you can move your robot at a constant velocity, take multiple measurements, and calculate their standard deviation from the known mean. If this process is too tedious, the values used in the example below should be a good representation of average noise from encoders.

备注: The standard deviation of the noise for a measurement has the same units as that measurement. For example, the standard deviation of the velocity noise has units of m/s.

备注: It is very important to use SI units (i.e. meters and radians) when passing parameters in Java. In C++, the *units library* can be used to specify any unit type.

JAVA

```
// Create our feedforward gain constants (from the identification
// tool)
static final double KvLinear = 1.98;
static final double KaLinear = 0.2;
static final double KvAngular = 1.5;
static final double KaAngular = 0.3;

// Create the simulation model of our drivetrain.
private DifferentialDrivetrainSim m_driveSim = new DifferentialDrivetrainSim(
  // Create a linear system from our identification gains.
  LinearSystemId.identifyDrivetrainSystem(KvLinear, KaLinear, KvAngular, KaAngular),
  DCMotor.getNEO(2),       // 2 NEO motors on each side of the drivetrain.
  7.29,                    // 7.29:1 gearing reduction.
  0.7112,                  // The track width is 0.7112 meters.
  Units.inchesToMeters(3), // The robot uses 3" radius wheels.

  // The standard deviations for measurement noise:
  // x and y:          0.001 m
  // heading:          0.001 rad
  // l and r velocity: 0.1   m/s
  // l and r position: 0.005 m
  VecBuilder.fill(0.001, 0.001, 0.001, 0.1, 0.1, 0.005, 0.005));
```

C++

```
#include <frc/simulation/DifferentialDrivetrainSim.h>
#include <frc/system/plant/LinearSystemId.h>
#include <units/acceleration.h>
#include <units/angular_acceleration.h>
#include <units/angular_velocity.h>
#include <units/voltage.h>
#include <units/velocity.h>

...

// Create our feedforward gain constants (from the identification
// tool). Note that these need to have correct units.
static constexpr auto KvLinear = 1.98_V / 1_mps;
static constexpr auto KaLinear = 0.2_V / 1_mps_sq;
static constexpr auto KvAngular = 1.5_V / 1_rad_per_s;
static constexpr auto KaAngular = 0.3_V / 1_rad_per_s_sq;
// The track width is 0.7112 meters.
static constexpr auto kTrackwidth = 0.7112_m;

// Create the simulation model of our drivetrain.
frc::sim::DifferentialDrivetrainSim m_driveSim{
  // Create a linear system from our identification gains.
  frc::LinearSystemId::IdentifyDrivetrainSystem(
    KvLinear, KaLinear, KvAngular, KaAngular, kTrackWidth),
  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}};
```

Creating a `DifferentialDrivetrainSim` of the KoP Chassis

The `DifferentialDrivetrainSim` class also has a static `createKitbotSim()` (Java) / `CreateKitbotSim()` (C++) method that can create an instance of the `DifferentialDrivetrainSim` using the standard Kit of Parts Chassis parameters. This method takes 5 arguments, two of which are optional:

- The type and number of motors on one side of the drivetrain.
- 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 diameter of the wheels installed on the drivetrain.
- The moment of inertia of the drive base (optional).
- Standard deviations of measurement noise: this represents how much measurement noise you expect from your real sensors. The measurement noise is an array with 7 elements, with each element representing the standard deviation of measurement noise

in x, y, heading, left velocity, right velocity, left position, and right position respectively. This option can be omitted in C++ or set to `null` in Java if measurement noise is not desirable.

You can calculate the measurement noise of your sensors by taking multiple data points of the state you are trying to measure and calculating the standard deviation using a tool like Python. For example, to calculate the standard deviation in your encoders' velocity estimate, you can move your robot at a constant velocity, take multiple measurements, and calculate their standard deviation from the known mean. If this process is too tedious, the values used in the example below should be a good representation of average noise from encoders.

备注: The standard deviation of the noise for a measurement has the same units as that measurement. For example, the standard deviation of the velocity noise has units of m/s.

备注: It is very important to use SI units (i.e. meters and radians) when passing parameters in Java. In C++, the *units library* can be used to specify any unit type.

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.
);
```

备注: You can use the `KitbotMotor`, `KitbotGearing`, and `KitbotWheelSize` enum (Java) / struct (C++) to get commonly used configurations of the Kit of Parts Chassis.

重要: Constructing your `DifferentialDrivetrainSim` instance in this way is just an approximation and is intended to get teams quickly up and running with simulation. Using empirical

values measured from your physical robot will always yield more accurate results.

22.5.4 Step 3: Updating the Drivetrain Model

Now that the drivetrain model has been made, it needs to be updated periodically with the latest motor voltage commands. It is recommended to do this step in a separate `simulationPeriodic()` / `SimulationPeriodic()` method inside your subsystem and only call this method in simulation.

备注: 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.

There are three main steps to updating the model:

1. Set the *input* of the drivetrain model. These are the motor voltages from the two sides of the drivetrain.
2. Advance the model forward in time by the nominal periodic timestep (Usually 20 ms). This updates all of the drivetrain's states (i.e. pose, encoder positions and velocities) as if 20 ms had passed.
3. Update simulated sensors with new positions, velocities, and angles to use in other places.

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.
```

(续下页)

(接上页)

```
  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.
  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());
}
```

重要: If the right side of your drivetrain is inverted, you MUST negate the right voltage in the `SetInputs()` call to ensure that positive voltages correspond to forward movement.

重要: Because the drivetrain simulator model returns positions and velocities in meters and m/s respectively, these must be converted to encoder ticks and ticks/s when calling `SetDistance()` and `SetRate()`. Alternatively, you can configure `SetDistancePerPulse` on the encoders to have the `Encoder` object take care of this automatically –this is the approach that is taken in the example above.

Now that the simulated encoder positions, velocities, and gyroscope angles have been set,

you can call `m_leftEncoder.GetDistance()`, etc. in your robot code as normal and it will behave exactly like it would on a real robot. This involves performing odometry calculations, running velocity PID feedback loops for trajectory tracking, etc.

22.5.5 Step 4: Updating Odometry and Visualizing Robot Position

Now that the simulated encoder positions, velocities, and gyro angles are being updated with accurate information periodically, this data can be used to update the pose of the robot in a periodic loop (such as the `periodic()` method in a `Subsystem`). In simulation, the periodic loop will use simulated encoder and gyro readings to update odometry whereas on the real robot, the same code will use real readings from physical hardware.

备注: For more information on using odometry, see *this document*.

Robot Pose Visualization

The robot pose can be visualized on the Simulator GUI (during simulation) or on a dashboard such as Glass (on a real robot) by sending the odometry pose over a `Field2d` object. A `Field2d` can be trivially constructed without any constructor arguments:

JAVA

```
private Field2d m_field = new Field2d();
```

C++

```
#include <frc/smartdashboard/Field2d.h>

..

frc::Field2d m_field;
```

This `Field2d` instance must then be sent over NetworkTables. The best place to do this is in the constructor of your subsystem.

JAVA

```
public Drivetrain() {
  ...
  SmartDashboard.putData("Field", m_field);
}
```

C++

```
#include <frc/smartdashboard/SmartDashboard.h>

Drivetrain() {
  ...
  frc::SmartDashboard::PutData("Field", &m_field);
}
```

备注： The `Field2d` instance can also be sent using a lower-level NetworkTables API or using the *Shuffleboard API*.

Finally, the pose from your odometry must be updated periodically into the `Field2d` object. Remember that this should be in a general `periodic()` method i.e. one that runs both during simulation and during real robot operation.

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());
}
```

重要： It is important that this code is placed in a regular `periodic()` method –one that is called periodically regardless of mode of operation. If you are using the command-based library, this method already exists. If not, you are responsible for calling this method periodically from the main `Robot` class.

备注： 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 单元测试

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 编写可测试的代码

备注： This example can be easily adapted to the command-based paradigm by having `Intake` inherit from `SubsystemBase`.

我们的子系统将是一个 Infinite Recharge 进气机构，包含一个活塞和一个电机：活塞展开/收回进气口，电机将把电池拉到里面。如果未部署进气机构，我们不希望电机运行，因为它不会做任何事情。

为了为每个测试提供一个"干净的石板"，我们需要有一个函数来销毁对象并释放所有硬件分配。在 Java 中，这是通过实现 AutoCloseable 接口及其.close() 方法来完成的，通过调用成员的.close() 方法来销毁每个成员对象 - 一个没有 ' 的对象 '.close()" 方法可能不需要关闭。在 C++ 中，当对象超出作用域时会自动调用默认的析构函数，并调用成员对象的析构函数。

备注： 供应商可能不支持与此处显示的方式相同的资源关闭。有关他们支持的内容和方式的更多信息，请参阅供应商的文档。

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);
  }
```

(续下页)

(接上页)

```
  public void deploy() {
    m_piston.set(DoubleSolenoid.Value.kForward);
  }

  public void retract() {
    m_piston.set(DoubleSolenoid.Value.kReverse);
    m_motor.set(0); // turn off the motor
  }

  public void activate(double speed) {
    if (isDeployed()) {
      m_motor.set(speed);
    } else { // if piston isn't open, do nothing
      m_motor.set(0);
    }
  }

  public boolean isDeployed() {
    return m_piston.get() == DoubleSolenoid.Value.kForward;
  }

  @Override
  public void close() {
    m_piston.close();
    m_motor.close();
  }
}
```

C++ (Header)

```
#include <frc/DoubleSolenoid.h>
#include <frc/motorcontrol/PWMSparkMax.h>

#include "Constants.h"

class Intake {
 public:
  void Deploy();
  void Retract();
  void Activate(double speed);
  bool IsDeployed() const;

 private:
  frc::PWMSparkMax m_motor{IntakeConstants::kMotorPort};
  frc::DoubleSolenoid m_piston{frc::PneumaticsModuleType::CTREPCM,
                               IntakeConstants::kPistonFwdChannel,
                               IntakeConstants::kPistonRevChannel};
};
```

C++ (Source)

```cpp
#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
}

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 编写测试

重要： 测试放置在 `test` 源集中：`/src/test/java/` 和 `/src/test/cpp/` 分别用于 Java 和 C++ 测试。该源根目录之外的文件无法访问测试框架 - 由于未解析的引用，这将导致编译失败。

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.

每个测试方法应至少包含一个*assertion*（Java 中的“assert*()”或 C++ 中的“EXPECT_*()”）。这些断言在运行时验证条件，如果不满足条件，则测试失败。如果测试方法中有多个断言，第一个失败的断言将使测试崩溃 - 执行不会到达后面的断言。

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.

备注： 浮点值的比较不准确，因此应该使用可接受的错误参数（DELTA）进行比较。

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
```

(续下页)

(接上页)

```
  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());
}
```

有关 JUnit 和 Google Test 的更高级用法，请参阅框架文档。

22.6.3 运行测试

备注： 测试将始终在您的桌面上以模拟方式运行。有关先决条件和更多信息，请参阅:doc: 模拟介绍。

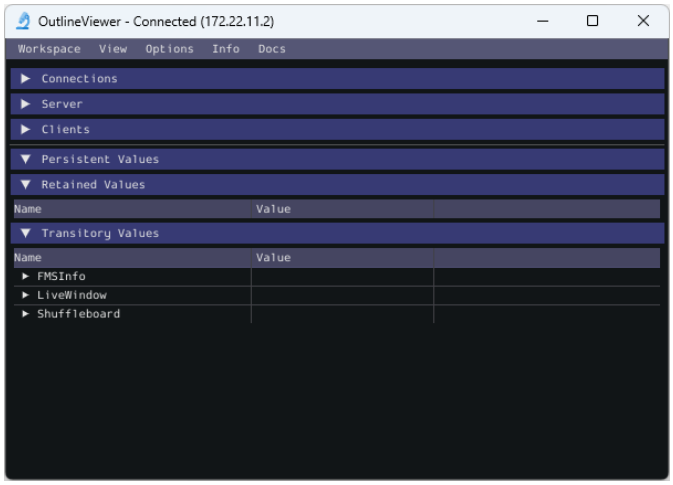
要运行 Java 测试，请确保您的“build.gradle”文件包含以下块：

```
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.

默认情况下，Gradle 会在构建机器人代码（包括部署）时运行测试。这将增加部署时间，失败的测试将导致构建和部署失败。为了防止这种情况发生，您可以使用指令面板中的:guilabel:*Change Skip Tests On Deploy Setting* 来配置是否在部署时运行测试。

23 OutlineViewer

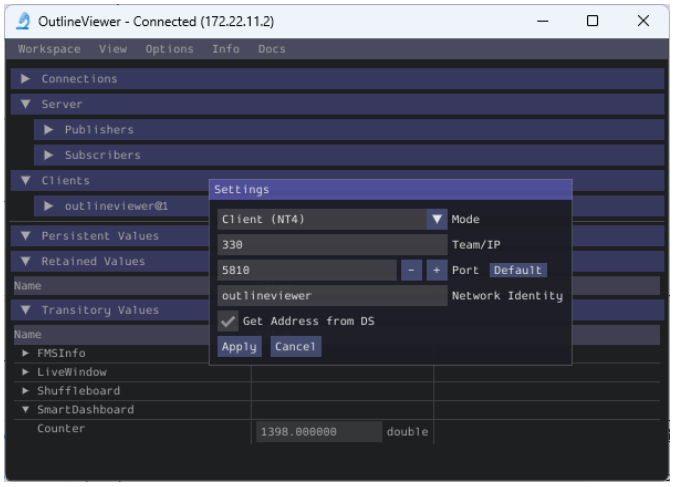


OutlineViewer is a utility used to view, modify and add to the contents of the NetworkTables for debugging purposes. It displays all key value pairs currently in the NetworkTables and can be used to modify the value of existing keys or add new keys to the table. OutlineViewer is included in the Java and C++ language installations.

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 *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*.

备注： You can use `localhost` instead of a team number to point OutlineViewer at a simulated robot, Romi or XRP.

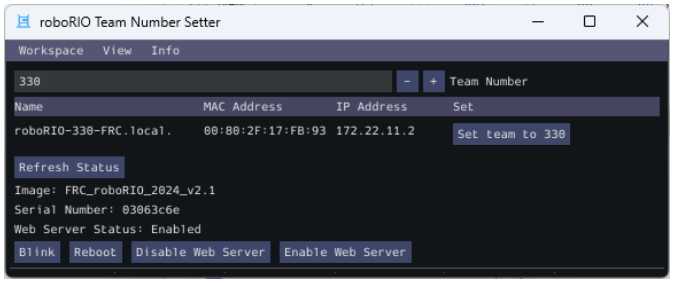


如果想在 Networktable 上添加新的数值对，要在页面上选择一个位置，并选中对应的数据类型。

备注： 使用 LabView 的队伍可以使用 LabView Dashboard 中“变量 Variables”的标签。这样就可以达成与 OutlineViewer 同样的效果。

---

24 roboRIO Team Number Setter



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

In Visual Studio Code, press `Ctrl+Shift+P` and type `WPILib` or click the WPILib logo in the top right to launch the WPILib Command Palette. Select *Start Tool*, then select *roboRIOTeamNumberSetter*.

Connect to the roboRIO over USB to use the tool, as this is the simplest method when the team number hasn't been set.

24.1 Setting Team Number

Enter your team number in the *Team Number* field and select *Set team to xxxx*. This will take about a second, then press the *Reboot* button to reboot the roboRIO so the new team number takes effect.

24.2 Enabling/Disabling Webserver

The *roboRIO's webserver* provides some debuging and enables some configuration. However, it also takes memory away from the robot program. You can disable it by clicking on the *Disable Web Server* button. If you'd like to enable it again, you can click *Enable Web Server*.

24.3 roboRIO Identification

Clicking the *Blink* button will cause the roboRIO's Radio LED to blink a few times to help identify the roboRIO.

25 视觉处理

25.1 视觉介绍

25.1.1 什么是视觉

FRC| reg |中的视觉是一种通过使用连接到机器人的摄像头，以帮助团队在自动阶段和远程操作期间得分和驾驶的技术。

视觉方式

在 FRC 比赛中，大多数团队采用的视觉方法主要有两种。

流媒体

此方法将相机的数据流传输到机器操控台，以便驾驶员和操作手可以从机器人的角度获取视觉信息。这种方法很简单，不需要花费很多时间来实现。如果不需要视觉处理功能，这是一个不错的选择。

- *Streaming using the roboRIO*

处理

Instead of only streaming the camera to the Driver Station, this method involves using the frames captured by the camera to compute information, such as a game piece’s or target’s angle and distance from the camera. This method requires more technical knowledge and time in order to implement, as well as being more computationally expensive. However, this method can help improve autonomous performance and assist in “auto-scoring” operations during the teleoperated period. This method can be done using the roboRIO or a coprocessor such as the Raspberry Pi using OpenCV.

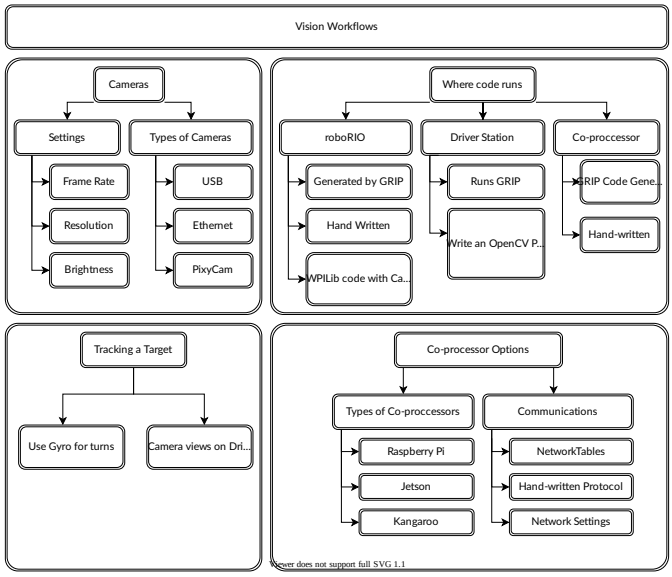
- *Vision Processing with Raspberry Pi*
- *Vision Processing with the roboRIO*

有关使用协处理器进行视觉处理的优缺点的其他信息，请参阅下一页:ref:*docs/software/vision-processing/introduction/strategies-for-vision-programming:Strategies for Vision Programming*.

25.1.2 视觉编程策略

使用计算机视觉是使您的机器人对现场的元素做出反应并使其更加自动化的一种好方法。通常在 FRC | reg |比赛中，可以通过自动将球或其他比赛物件射入目标，或导航到场地上获得奖励分数。计算机视觉是解决其中许多问题的好方法。而且，如果您拥有可以实现这一操作的自动代码，那么它也可以在遥控操作期间使用，以帮助驾驶员。

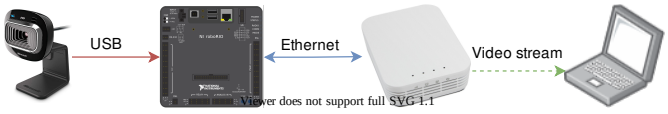
视觉处理系统的的组件以及视觉程序应该运行的位置有多种选项。WPILib 和相关的工具也支持多种选项，并为团队的决定提供了很大的灵活性。本文将让您了解许多可用的选择和权衡。



OpenCV 计算机视觉库

OpenCV 是一个计算机视觉软件库，在学术界和工业界广泛使用。它有来自硬件制造商的支持，提供 GPU 加速处理，它绑定多种语言，包括 c++，Java，和 Python。它还在许多 web 站点、书籍、视频和培训课程有详细的文档，因此有许多可用资源来帮助学习如何使用它。c++ 和 WPILib Java 版本都包含有 OpenCV 库，可提供捕获、处理和查看的视频，以及帮助创建视觉算法的工具。有关 OpenCV 的更多信息，请参见 https://opencv.org。

在 roboRIO 上运行视觉代码



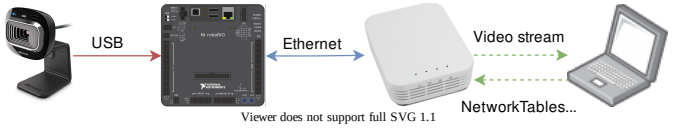
Vision code can be embedded into the main robot program on the roboRIO. Building and running the vision code is straightforward because it is built and deployed along with the robot program. The vision code can be written in C++, Java, or Python. The disadvantage of this approach is that having vision 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.

使用 CameraServer 类提供的函数，视频流可以发送到仪表盘，比如 Shuffleboard，这样操作手就可以看到摄像头看到的内容。此外，可以使用 OpenCV 命令将注释添加到图像中，以便在仪表盘视图中识别目标或其他感兴趣的对象。

DS 计算机上的视觉代码

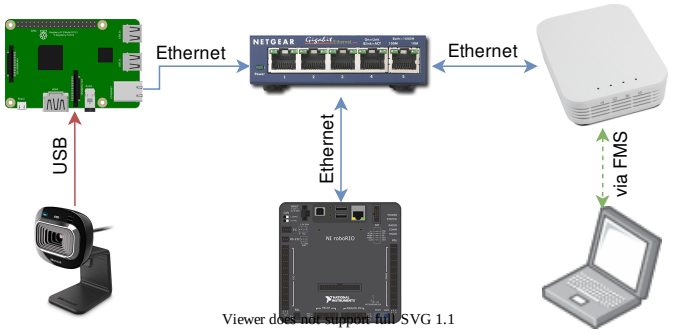


When vision code is running on the DS computer, the video is streamed back to the Driver Station laptop for processing. Even the older Classmate laptops are substantially faster at vision processing than the roboRIO. You can write your own vision program using a language of your choosing. Python makes a good choice since there is a native NetworkTables implementation and the OpenCV bindings are very good.

图像处理后，关键值，如目标位置、距离或任何您需要的东西，可以通过网络表格发送回机器人。这种方法通常具有较高的延迟，因为需要将图像发送到笔记本电脑会增加延迟。带宽的限制也限制了用于处理的图像的最大分辨率和 FPS。

The video stream can be displayed on Shuffleboard or SmartDashboard.

协处理器上的视觉代码



像 Raspberry Pi 这样的协处理器是支持视觉代码的理想选择 (参见:ref: ‘docs/software/vision-processing/frcvision/ using-raspberry - pie -for- FRC:Using the Raspberry Pi for FRC ‘)。优点是它们可以全速运行，不干扰机器人程序。在这种情况下，摄像机可能连接到协处理器，或者 (对于以太网摄像机来说) 机器人上的一个以太网开关。该程序可以用任何语言编写;Python 是一个很好的选择，因为它可以简单地绑定到 OpenCV 和 NetworkTables。一些团队使用高性能的视觉协同处理器，如 Nvidia Jetson，以获得最快的速度和最高的分辨率，尽管这种方法通常需要高级的 Linux 和编程知识。

与其他两种方法相比，这种方法需要更多的编程专业知识和少量的额外权重，但除此之外，它带来了两个方面的最佳效果，因为协处理器比 roboRIO 快得多，并且图像处理可以以最小的延迟或带宽使用进行。

据可以通过网络或串行连接从协处理器上的视觉程序发送到机器人。

相机选择

WPILib 支持多种摄像机选项。摄像机有许多影响操作的参数；例如，帧频和图像分辨率影响接收图像的质量，但当设置过高的影响处理时间时，如果发送到操控站，可能会超过现场的可用带宽。

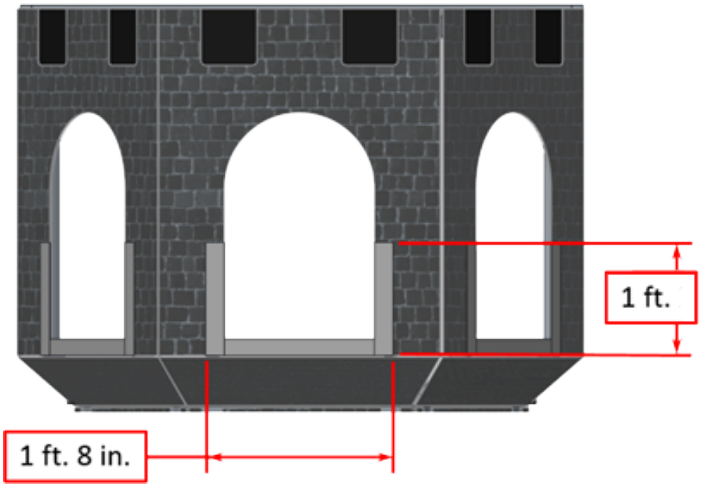
c++ 和 Java 中的 CameraServer 用于与连接到机器人的摄像机进行对接。它通过源对象检索用于本地处理的帧，并将流发送到您的操控站，以便在那里查看或处理。

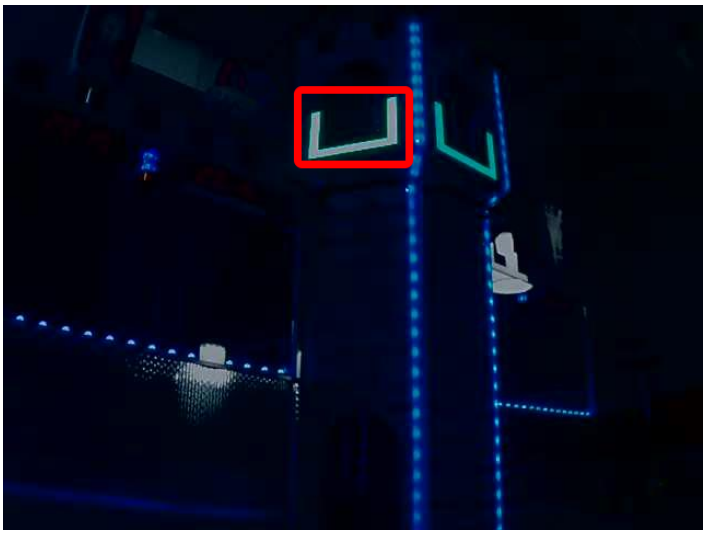
25.1.3 目标信息和反射

许多 FRC | reg |游戏在场元素上贴有反光带，以帮助视觉处理。本文档介绍了 2016 FRC 游戏中的视觉目标以及构成目标的材料的视觉属性。

备注： 有关所有现场组件的正式尺寸和图纸，请参阅官方现场图纸。

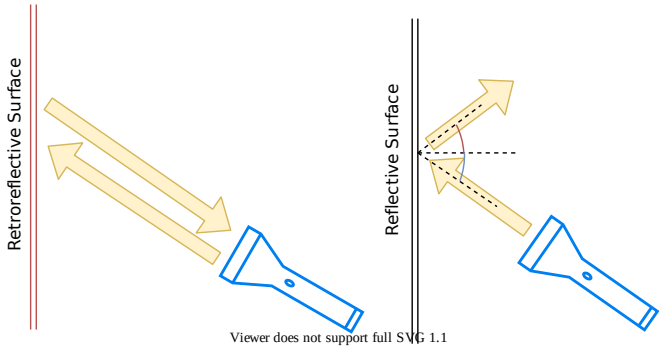
目标





每个 2016 年的视觉目标均由 1 英寸 8 英寸宽，1 英寸高的 U 形组成，该 U 形由 2 英寸宽的逆向反射材料（3M 8830 银标膜）制成。目标位于紧邻每个高目标底部的位置。当正确照明时，逆向反射带会产生明亮和/或颜色饱和的标记。

逆反射率与反射率



高反射率的材料通常会成为镜像，以使光线以补充角度“反射”出去。如左上方所示，蓝色和红色角度总计为 180 度。一个等效的解释是，光线围绕表面法线反射垂直于该表面绘制的绿线。请注意，仅当蓝角约为 90 度时，指向表面的光才会返回光源。

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.

反射示例

该材料应该相对熟悉，因为它通常用于增强道路标志，自行车和行人的夜间可见性。

最初，反射似乎对于夜间安全而言似乎不是有用的属性，但是，如上所示，当光线和眼睛彼此靠近时，反射光会返回到眼睛，并且即使在很远的距离处，该材料也明亮地发光。由于驾驶员的眼睛与车辆前灯之间的夹角较小，因此夜间使用时，逆反射材料可以大大提高远处物体的可见度。

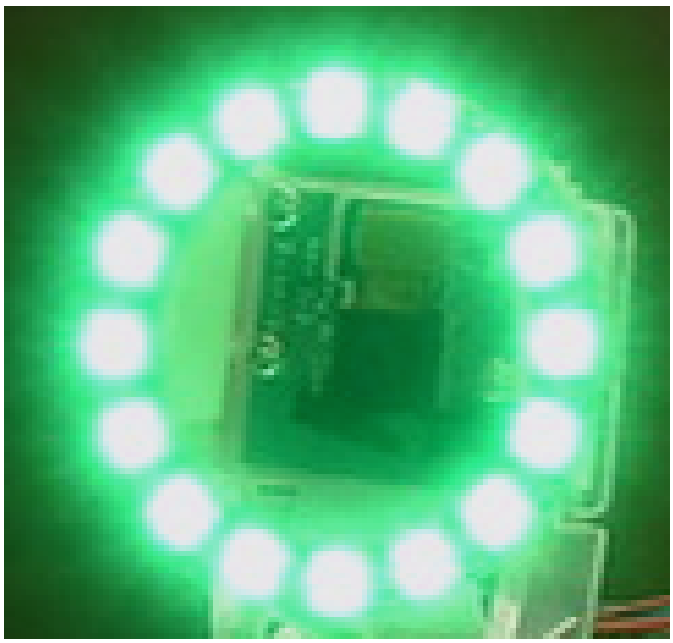
示范

为了进一步探索反射材料的特性：

1. 将一块材料放在墙壁或垂直表面上
2. 站在 10-20 英尺远的地方，并用小手电筒照亮材料。
3. 从腹部按钮处的灯光开始，然后慢慢抬起直到两眼之间。随着光线靠近您的眼睛，返回的光线强度将迅速增加。
4. 通过移至房间中的其他位置并重复来更改角度。明亮的反射应该在很大的视角范围内发生，但是从光源到眼睛的角度是关键，并且必须很小。

尝试使用不同的光源。这种材料的反射性是白色涂料的数百倍。因此昏暗的光源可以正常工作。例如，红色的自行车安全灯将表明光源的颜色决定了反射光的颜色。如果可能，将几个团队成员放置在不同的位置，每个位置都有自己的光源。这将表明效果在很大程度上是独立的，并且材质可以同时为各个团队成员显示不同的颜色。这也表明该材料在很大程度上不受环境光的影响。返回观察者的光几乎完全由他们控制的光源或直接在他们后面的光源决定。使用手电筒，确定环境中已经存在的其他反光物品：衣服，背包，鞋子等上的物品。

灯光



我们已经看到，除非有光源指向，否则反光带将不会发光，并且光源必须非常靠近相机镜头或观察者的眼睛。尽管有多种方法可以实现此目的，但是要研究的一种非常有用的光源类型是上图所示的环形闪光灯。它将光源直接放在相机镜头上或镜头周围，并提供非常均匀的照明。由于其明亮的输出和较小的尺寸，LED 对于构造这种类型的设备特别有用。

如上所示，价格低廉的圆形 LED 可以提供各种颜色和尺寸，并且易于连接到相机，有些甚至可以通过 Raspberry Pi 断电。尽管不是为漫射均匀照明而设计，但它们可以很好地使逆反射胶带发光。可通过 FIRST Choice 获得一个小的绿色 LED 环。其他类似的 LED 环也可以从供应商处获得，例如 SuperBrightLED。

样本图片

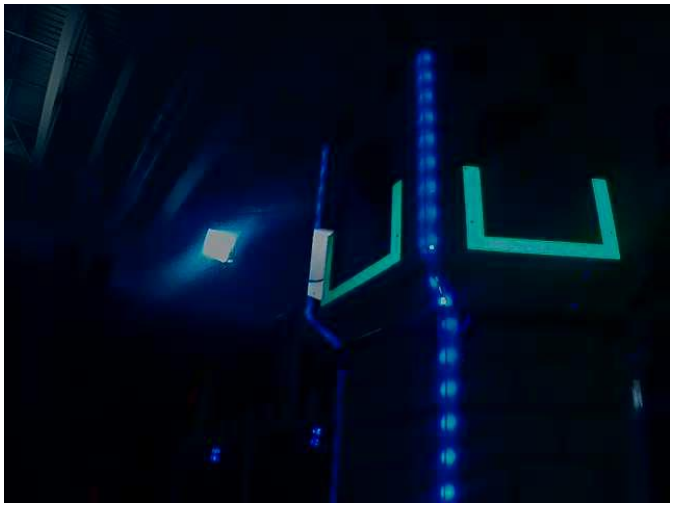
示例图像与每种语言的代码示例一起放置（与 LabVIEW 打包在一起，并在单独的 ZIP 中，与 C ++ / Java 示例位于相同的位置）。

25.1.4 识别和处理目标

捕获图像后，下一步就是识别图像中的视觉目标。本文档将通过一种方法来确定 2016 年赛季的视觉目标。请注意，本节中使用的图像是在相机刻意将其设置为曝光不足的情况下拍摄的，经过如此配置，除了被照亮的目标外，还会产生非常暗的图像，有关详细信息，请参见“相机设置”部分。

原始图片

下图是本示例将要使用的起始图像。其使用了 * FIRST * | reg |中可找到的的绿色环形灯拍摄。其可选配不同尺寸的附加环形灯。视觉代码示例提供了其他示例图像。



什么是 HSL / HSV？

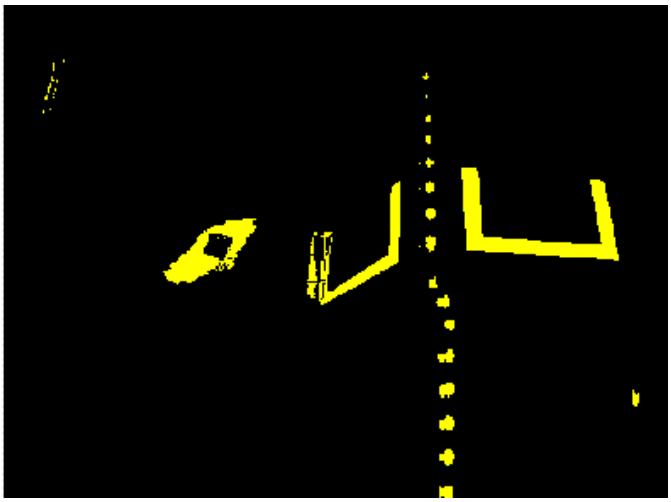
颜色的色相或色调通常出现在艺术家的色轮上，并包含红色，橙色，黄色，绿色，蓝色，靛蓝和紫罗兰色的彩虹色。色相是使用色轮上的径向角指定的，但是在实际成像中，圆的径向角通常仅包含 256 个单位，其从零（红色）开始，循环穿过彩虹渐变，然后在上端回绕为红色。颜色的饱和度指定颜色的浓量，或色调颜色与灰色阴影的比率。较高的比率表示更多彩色，较少灰色。零饱和度没有色相，并且完全是灰色。“亮度”或“值”指示与色调混合的灰色阴影。黑色为 0，白色为 255。

该示例代码使用 HSV 颜色空间指定了目标的颜色。这一操作的主要原因是因为通过使用“值”（HSV）或“亮度”（HSL）分量，可以轻松地将目标相对于图像其余部分的亮度用作过滤标准。使用 HSV 色彩系统的另一个原因是，当在 HSV 色彩空间中进行阈值过滤操作时，能 roboRIO 上更有效地运行。

遮罩

一开始，像素值会与恒定的颜色或亮度值进行比较，以创建下面以黄色显示的二元遮罩。这一步消除了除了反光带以外的大部分像素。如果颜色相对饱和，明亮且一致，则基于颜色的遮罩效果会很好。使用 HSL（色相，饱和度和亮度）或 HSV（色相，饱和度和值）颜色空间时，捕捉颜色的区别通常比 RGB（红色，绿色和蓝色）空间更准确。当颜色范围在一维或多个维上很大时，尤其如此。

请注意，除了目标之外，图像的其他明亮部分（顶灯和塔顶照明）也会被遮罩步骤捕获。



像素块分析

遮罩操作之后，像素块报告将检查像素块的面积，边界矩形和等效矩形。这些用于计算几个计分项，以帮助选择最矩形的形状。下文所述的每个测试都会生成一个分数（0-100），然后将其与预定义的分数限制进行比较，以确定像素块是否为目标。

覆盖区域

面积分数是通过将像素块的面积与像素块周围绘制的边界框的面积进行比较来计算的。全反射条的面积为 80 平方英寸（🄵516：数学：*cm ^ 2*）。包含目标的矩形区域为 240 平方英寸（🄵0.15：math：*m ^ 2*）。这意味着面积与边界框面积之间的理想比率为 1/3。接近 1/3 的面积比将产生接近 100 的得分，比例越偏离 1/3 时，得分越接近 0。

长宽比

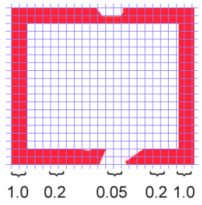
长宽比得分基于（像素块宽度/像素块高度）。像素块的宽度和高度使用称为“等效矩形”的值来确定。等效矩形是边长为：math：‘x‘和：math：‘y‘的矩形，其中：math：‘2x+2y‘等于像素块周长，而：math：‘x cdot y‘等于像素块面积。等效矩形用于纵横比计算，因为与使用边框相比，等效矩形受矩形倾斜的影响较小。当使用边框矩形作为宽高比时，随着矩形的倾斜，高度会增加，宽度会减小。

视觉目标宽为 20 英寸（508 毫米），高为 12 英寸（304.8 毫米），比率为 1.6。将检测到像素块的纵横比与该理想比例进行比较。当该比例与目标比例匹配时，纵横比得分会归一化为 100，并随着该比例的变化而线性下降。

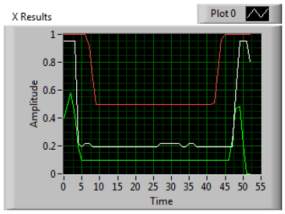
力矩

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.

X / Y 配置文件



Column averages for a particle rectangle.



White line is the average, red is upper limit, and green is lower limit..

边缘分数描述了像素块在 X 和 Y 方向上是否都匹配适当的轮廓。如图所示，它是使用从原始图像中提取的边界框上的行和列平均值，并将其与轮廓蒙版进行比较来计算的。根据行平均值或列平均值中上限值和下限值之间的值的数量，分数的范围为 0 到 100。

测量

如果像素块得分足够好，其将视为目标，此时则可以计算一些实际的测量值，例如位置和距离。该示例代码包含这些基本度量，因此让我们看一下所涉及的数学以更好地理解它。

位置

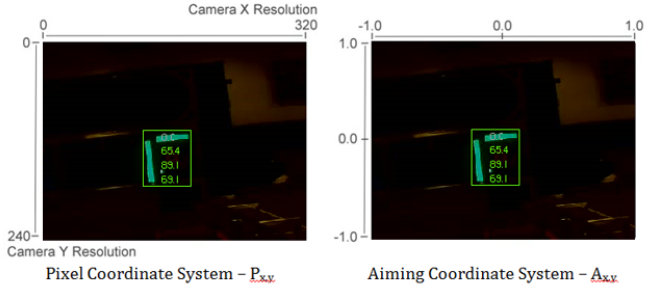
像素块和边界框都很好地描述了目标位置，但是所有坐标均以像素为单位，其中 0,0 位于屏幕的左上角，其右边缘和下边缘由相机分辨率确定。这对于像素数学来说是一个有用的系统，但是对于驱动机器人却没有那么有用。因此，让我们将其更改为可能更有用的内容。

要将点从像素系统转换为瞄准系统，可以使用以下公式。

所得到的坐标系是接近我们想要的，但是 Y 轴是反着的。这可以通过将该点乘以 [1，-1] 来纠正。(注意: 在示例代码中没有这样做)。这个坐标系统是有效的，因为它有一个中心原点和规模类似于操纵杆输出和驱动输入。

$$A_{x□y} = \ left□P_{x□y} - \ frac\ textitresolution_{x□y}2\ right□/\ frac\ textitresolution_{x□y}2$$

$$A_{x,y} = (P_{x,y} - \frac{resolution_{x,y}}{2}) / \frac{resolution_{x,y}}{2}$$



视场

您可以使用已知常数和目标在坐标平面上的位置来确定距目标的距离，偏航和俯仰。但是，为了计算这些，您必须确定您的 FOV（视场）。为了凭经验确定垂直视场，请将相机设置为与平面保持一定距离，然后测量最顶部和最底部像素行之间的距离。

$$\frac{1}{2}FOV_{vertical} = tan\left(\frac{\frac{1}{2}distance_y}{distance_z}\right)$$

您可以使用相同的方法，但使用像素的第一列和最后一列之间的距离来找到水平 FOV。

俯仰和偏航

一旦知道了视野和目标在瞄准坐标系中的位置，就可以轻松找到目标相对于机器人的俯仰和偏航。

$$pitch = \ fracA_y 2FOV_{vertical}$$

$$yaw = \frac{A_x}{2}FOV_{horizontal}$$

距离

如果目标的高度与机器人的高度明显不同，则可以使用已知的常数，例如目标和摄像机的物理高度以及摄像机的安装角度，来计算摄像机与目标之间的距离。

$$distance = \frac{height_{target} - height_{camera}}{tan(angle_{camera} + pitch)}$$

另一种选择是创建一个面积到距离的查找表，或估算面积和距离的反变常数（inverse variation constant）。但是，此方法不太准确。

备注： 为了获得上述估计角度和距离的方法的最佳结果，您可以使用 OpenCV 校准相机，并使用校准矩阵重新投影目标像素。这样能够消除可能影响精度的任何失真。

25.1.5 读取和处理视频：CameraServer 类

概念

FRC | reg |中通常使用的摄像机（商品 USB 和以太网摄像机，例如 Axis 摄像机）提供相对有限的操作模式。通常，它们仅以单个分辨率和帧速率提供单个图像输出（通常以 RGB 压缩格式，例如 JPG）输出。USB 相机受到特别限制，因为一次只能有一个应用程序可以访问相机。

CameraServer 支持多个摄像机。它处理诸如断开摄像机连接时自动重新连接之类的细节，并使来自摄像机的图像可供多个“客户端”使用（例如，您的机器人代码和仪表板都可以同时连接到摄像机）。

相机名称

CameraServer 中的每个摄像机都必须唯一命名。这也是在仪表板中为相机显示的名称。CameraServer 的 “startAutomaticCapture ()” 和 “addAxisCamera ()” 函数的某些变体会自动命名相机（例如 “USB Camera 0” 或 “Axis Camera”），或者您可以为相机指定一个更具描述性的名称（例如 “进气凸轮”）。唯一的要求是每个摄像机都有一个唯一的名称。

USB 相机注意事项

CPU 使用率

CameraServer 旨在通过仅在需要时执行压缩和解压缩操作，并在未连接任何客户端时自动禁用流传输来最大程度地减少 CPU 使用率。

为了最大程度地减少 CPU 使用率，仪表板分辨率应设置为与摄像头相同的分辨率。这使 CameraServer 无需解压缩和重新压缩图像，而是可以简单地将从照相机接收的 JPEG 图像直接转发到仪表板。请务必注意，更改仪表板上的分辨率不会更改相机分辨率；可以通过在相机对象上调用 setResolution () 来更改相机分辨率。

USB 带宽

roboRIO 一次只能通过其 USB 接口发送和接收大量数据。相机图像可能需要大量数据，因此相对容易遇到此限制。USB 带宽错误的最常见原因是选择非 JPEG 视频模式或分辨率过高，尤其是在连接多台摄像机时。

体系结构

CameraServer 由两层组成，高层 WPILib ** CameraServer 类 ** 和底层 ** cscore 库 **。

CameraServer 类

CameraServer 类（WPILib 的一部分）提供了一个高级接口，用于将摄像机添加到您的机器人代码中。它还负责将有关摄像机和摄像机服务器的信息发布到 NetworkTables，以便 LabVIEW Dashboard 和 Shuffleboard 等 Driver Station 仪表板可以列出摄像机并确定其流的位置。它使用单例模式来维护所有创建的摄像机和服务器的数据库。

CameraServer 中的一些关键功能包括：

- `startAutomaticCapture ()`：添加一个 USB 摄像头（例如 Microsoft LifeCam）并为其启动服务器，以便可以从仪表板上对其进行查看。
- `addAxisCamera ()`：添加一个 Axis 相机。即使您没有在机器人代码中处理 Axis 相机的图像，也可能要使用此功能，以便 Axis 相机出现在 “仪表板” 的相机下拉列表中。它还会启动服务器，以便当驱动程序站通过 USB 连接到 roboRIO 时仍可以查看 Axis 流（如果将 Axis 摄像机和 roboRIO 都连接到两个机器人无线电以太网端口，则在竞争中很有用）。
- `getVideo ()`：获取对摄像机的 OpenCV 访问。这使您可以从相机获取图像以在 roboRIO 上进行图像处理（在您的机器人代码中）。
- `putVideo ()`：启动一个服务器，您可以向其中提供 OpenCV 图像。这使您可以将自定义处理和/或带注释的图像传递到仪表板。

cscore 库

cscore 库提供了较低级别的实现：

- 从 USB 和 HTTP（例如 Axis）相机获取图像
- 更改相机设置（例如对比度和亮度）
- 更改相机视频模式（像素格式，分辨率和帧频）
- 充当 Web 服务器并将图像作为标准 MJPEG 流提供
- 将图像与 OpenCV“Mat’’ 对象之间来回转换以进行图像处理

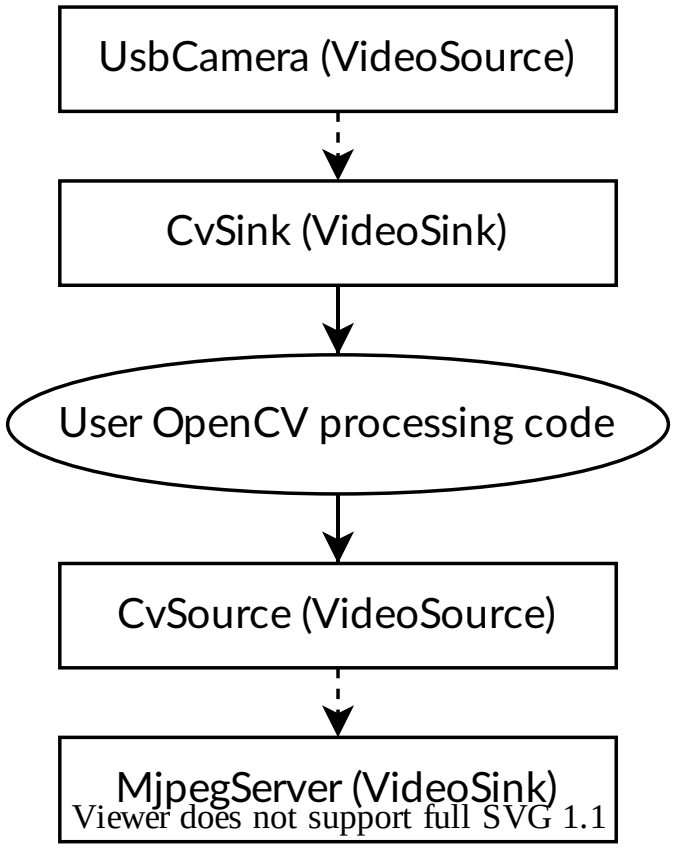
来源和接收

cscore 库的基本体系结构与 MJPGStreamer 相似，功能在源和接收器之间分配。可以创建多个源并同时创建多个接收器。

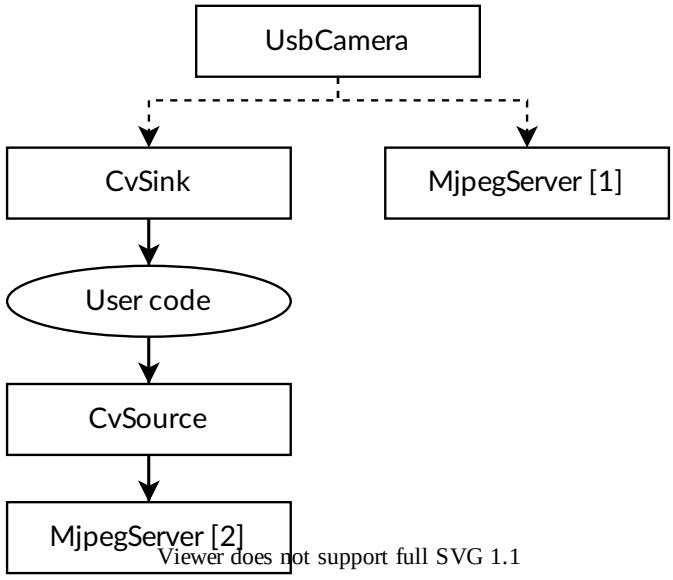
生成图像的对象是源，接受/使用图像的对象是接收器。生成/使用是从库的角度进行的。因此，相机是来源（它们生成图像）。MJPEG Web 服务器是一个接收器，因为它接受程序中的图像（即使它可能会将这些图像转发到 Web 浏览器或仪表板上）。源可以连接到多个接收器，但是接收器可以连接到一个且仅一个源。当接收器连接到源时，cscore 库会负责将每个映像从源传递到接收器。

- **来源 ** 获取单独的帧（例如 USB 摄像头提供的帧），并在有新帧可用时触发事件。如果没有接收器正在监听特定的源，则该库可能会暂停或与源断开连接，以节省处理器和 I / O 资源。该库可通过简单地暂停和恢复事件的触发来自动处理摄像机的断开/重新连接（例如，断开连接不会导致出现新的帧，不会出现错误）。
- 接收者监听特定来源的事件，获取最新图像，并以适当的格式将其转发到目的地。与源相似，如果特定接收器处于非活动状态（例如，没有客户端通过 HTTP 服务器连接到已配置的 MJPEG），则该库可能会禁用其部分处理以节省处理器资源。

用户代码（例如 FRC 机器人程序中使用的代码）可以通过 OpenCV 源对象和接收器对象充当源（提供处理过的帧，就好像它是照相机一样），也可以充当接收器（接收要处理的帧）。因此，从相机获取图像并将处理后的图像输出的图像处理线程如下图所示：



因为源可以连接多个接收器，所以线程可以分支。例如，还可以通过将 UsbCamera 源连接到 CvSink 之外的第二个 MjpegServer 接收器来提供原始摄像机图像，如下图所示：



当摄像机捕获到新图像时，CvSink 和 MjpegServer [1] 都会接收到它。

上图是以下 CameraServer 片段创建的：

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);
```

CameraServer 实现在 cscore 级别上有效地执行了以下操作（出于解释目的）。CameraServer 处理许多细节，例如为所有 cscore 对象创建唯一名称以及自动选择端口号。CameraServer 还会保留已创建对象的单例注册表，因此如果它们超出范围，它们不会被破坏。

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);
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);
```

参考计数

所有 cscore 对象都在内部进行引用计数。将接收器连接到源会增加源的引用计数，因此仅严格要将接收器保持在范围内。CameraServer 类保留使用 CameraServer 函数创建的所有对象的注册表，因此，以这种方式创建的源和接收器永远不会超出范围（除非显式删除）。

25.1.6 2017 视觉程序示例

LabVIEW

其他 LabVIEW 范例随附 2017 LabVIEW Vision 范例。在启动屏幕上，单击支持-> 查找 FRC | reg |。示例或在任何其他 LabVIEW 窗口中，单击帮助-> 查找示例，然后找到 Vision 文件夹以查找 2017 Vision 示例。示例图像与示例捆绑在一起。

25.2 带有 WPILibPi 的视觉

25.2.1 在 Raspberry Pi 上使用 WPILibPi 的视频演练

备注： 视频中提到了 FRCVision，这是 WPILibPi 的旧名称。

在 2020 年的 “WPI 主办的 RSN 春季会议” 上，WPILib 团队的 Peter Johnson 就 FRC | reg |发表了演讲。Raspberry Pi 的愿景。

演示文稿的链接如下：here.

25.2.2 使用协同处理器的视觉处理

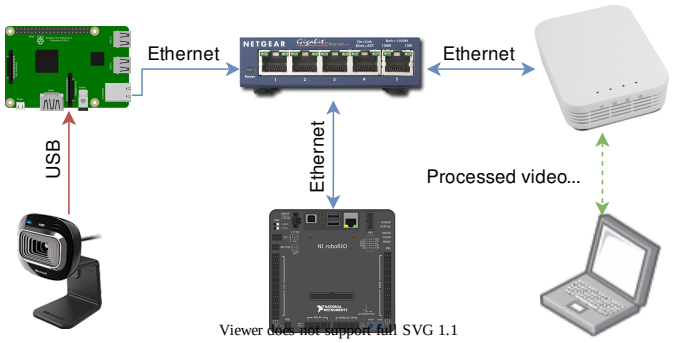
使用诸如 OpenCV 之类的库进行视觉处理来识别目标或游戏作品通常会占用大量 CPU 资源。通常，负载并不是太重要，而且 roboRIO 可以轻松处理该处理。在摄像机流更多或图像处理复杂的情况下，最好通过将代码和摄像机连接放在不同的处理器上来卸载 roboRIO。在 FRC | reg |中有很多种处理器可供选择。例如 Raspberry PI，基于 Intel 的袋鼠，LimeLight，以实现极致的简洁性，或者为更复杂的视觉代码提供图形加速器，例如 nVidia Jetson 型号之一。

策略

我们需要一款对应的软件来设置我们的协同处理器，一般包括有：

- OpenCV - 一个开源的计算机视觉库
- 术语: Network tables（网络表） - 用于将图像处理的结果传输到 RoboRIO 的程序里
- Camera server library（摄像头服务器库） - 用于摄像头的连接与传输视频流以确保在控制面板上能看到视频
- 语言库（对应视觉程序的编程语言）
- 你的程序（能探测到物体的那种）

将协同处理器连接到 RoboRIO 的办法有两种。一，将协同处理器通过额外的以太网接口与路由器相连；二，若是要连接更多设备的话，可以在机器上添加一个小型的网络交换机。摄像头插入协同处理器后会获取图像、处理图像，最终将结果（通常是目标的地点信息）通过 Network tables（网络表）传输到机器以便瞄准与控制方向。



将视频流传输至控制面板

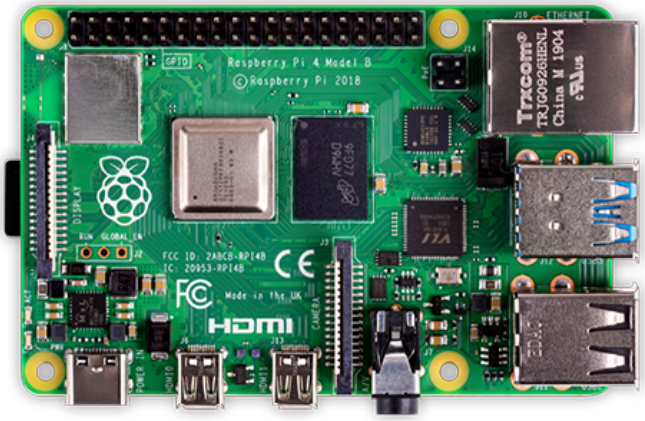
我们通常会简单地将摄像头的数据直接通过机器人的网络传到控制面板。使用以上方法，我们可以将一个或多个视频流传输到网络并且通过例如 Shuffleboard 或网页浏览器在控制面板上显示。使用 Shuffleboard 的好处有几点：1、便于调整摄像头的分辨率；2、便于调整摄像头的比特率；3、可以将视频流与其他数据一起传输给机器人。

同时，我们也可以在处理图像时添加注释并传输到控制面板，例如目标的线或是盒子，如此以便于操控手们看清机器人周围的情况。

25.2.3 在 FRC 中运用树莓派

树🅴派是最受欢迎的协同处理器之一，因为：

- Low cost - around $35
- 非常高的可用性：我们可以从很多不同的供应商中看到树🅴派，包括亚马逊
- Very good performance - the current Raspberry Pi 3b+ has the following specifications:
- Technical Specifications: - Broadcom BCM2837BO 64 bit ARMv8 QUAD Core A53 64bit Processor powered Single Board Computer run at 1.4GHz - 1GB RAM - BCM43143 WiFi on board - Bluetooth Low Energy (BLE) on board - 40 pin extended GPIO - 4 x USB2 ports - 4 pole Stereo output and Composite video port - Full size HDMI - CSI camera port for connecting the Raspberry - Pi camera - DSI display port for connecting the Raspberry - Pi touch screen display - MicroSD port for loading your operating system and storing data - Upgraded switched Micro USB power source (now supports up to 2.5 Amps.



预先建立立树莓派图形库

为了确保树🅴派对于队伍而言是非常简单的，在这里我们提供了树🅴派的图形库。其可以被拷贝到一张 MicroSD 卡上，插入树🅴派，然后启动。在默认情况下，它支持：

- A web interface for configuring it for the most common functions
- 支持在网络接口上设置任意个数摄像头的视屏流 (默认情况下是一个)
- OpenCV, Network Tables(网络表), Camera Server(摄像头服务器), 以及 c++, Java 和 Python 自定义程序的语言库

如果只是想将一个或者多个摄像头的视屏流上传到网络（或者仪表盘）的话，是不需要任何编程的，只需要在网络界面进行配置就可以了。

下一节讨论如何将映像安装到闪存卡上并启动 Pi。

25.2.4 你需要哪些东西来让树莓派处理理图像

如果你想要让树🄵派作为一个协同处理器来处理视屏或者图像，你需要:

- A Raspberry Pi 3 B, Raspberry Pi 3 B+, or a Raspberry Pi 4 B
- 至少 8 GB 的 Micro SD 卡可容纳所有提供的软件，建议的速度等级为 10（10MB / s）
- 一根以太网线用来连接树🄵派和 roboRIO
- A USB micro power cable to connect to the Voltage Regulator Module (VRM) on your robot. It is recommended to use the VRM connection for power rather than powering it from one of the roboRIO USB ports for higher reliability
- A laptop that can write the MicroSD card, either using a USB dongle (preferred) or a SD to MicroSD adapter that ships with most MicroSD cards

所示为便宜的 USB 加密狗，它将编写 FRC | reg |。映像到 MicroSD 卡。

25.2.5 将图形库安装到 MicroSD 卡上

获得 FRC 树莓派的图形库

该图像存储在 WPILibPi‘repository <https://github.com/wpilibsuite/WPILibPi/releases> __ 的 GitHub 发布页面上。

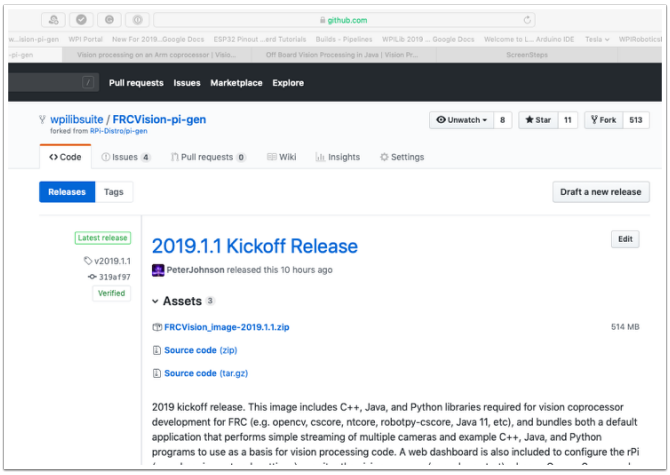
除了这里的指示与介绍，还可以前往 GitHub 的网页上看 Documentation。

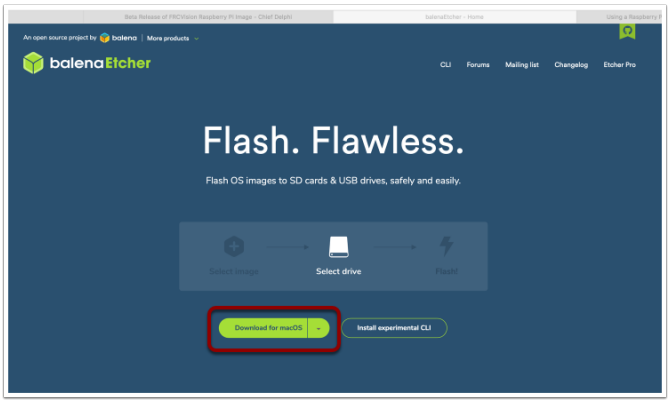
由于图形库比较大，所以推荐在较快的网络下下载。永远要使用最新的发布版本 (将会显示在列表的最顶端)。

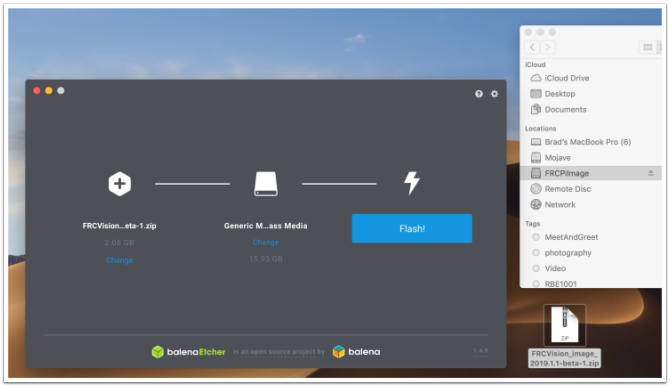
将图形库拷🄵贝到你的 MicroSD 卡上

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.

使用 Etcher，选择好你的 MicroSD 卡所在的磁盘作为目标，并点击 “Flash” 来进行格式化。预计整个过程将持续 3 分钟。


Pull requests
Issues
Marketplace
Explore
wpilibsuite / FRCVision-pi-gen
forked from RPi-Distro/pi-gen
Unwatch 8
Star 11
Fork 513
Code
Issues 4
Pull requests 0
Wiki
Insights
Settings
Releases
Tags
Draft a new release
Latest release
v2019.1.1
319af97
Verified
2019.1.1 Kickoff Release
Edit
PeterJohnson released this 10 hours ago
Assets 3
FRCVision_image-2019.1.1.zip
514 MB
Source code (zip)
Source code (tar.gz)
2019 kickoff release. This image includes C++, Java, and Python libraries required for vision coprocessor development for FRC (e.g. opencv, cscore, ntcore, robotpy-cscore, Java 11, etc), and bundles both a default application that performs simple streaming of multiple cameras and example C++, Java, and Python programs to use as a basis for vision processing code. A web dashboard is also included to configure the rPi


balenaEtcher
CLI
Forums
Mailing list
Changelog
Etcher Pro
Flash. Flawless.
Flash OS images to SD cards & USB drives, safely and easily.
Select drive
Download for macOS
Install experimental CLI


Flash!
iCloud
iCloud Drive
Desktop
Documents
Locations
Brad's MacBook Pro (6)
Mojave
FRCPiImage
Remote Disc
Network
Tags
MeetAndGreet
photography
Video
RBE1001

使用用树莓派进行行行测试

1. 将 MicroSD 卡插入树🄴派 3(简记为 rPi 3)，并插电。
2. 将 rPi3 通过网线连接到 LAN 或者 PC。打开网络浏览器并连接到 “http://wpilibpi.local/“来打开 web dashboard。在最开始，系统的启动文件系统都是可读写的，但是在将来其将默认变成只读。所以如果要进行任何形式的更改，需要点击 “可写” 按钮来完成操作。

登录到树莓派

大多数 rPi 操作可以在网络终端界面完成，但是一些更高级的，比如程序开发则需要进行登录，若要登录，请使用树🄴派的默认密码：

```
Username: pi
 Password: raspberry
```

25.2.6 树莓派

FRC 终端

FRC | reg | Raspberry PI 的映像包含一个控制台，可以在任何 Web 浏览器中查看该控制台，从而可以轻松地：

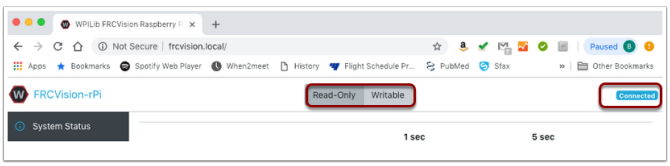
- Look at the Raspberry PI status
- 查看摄像头的后台进程状态
- 查看或者更改网络设置
- 查看接入树莓派的摄像头以及额外的摄像头
- 加载一个视觉程序到树莓派上

设置树莓派为只读还是读写

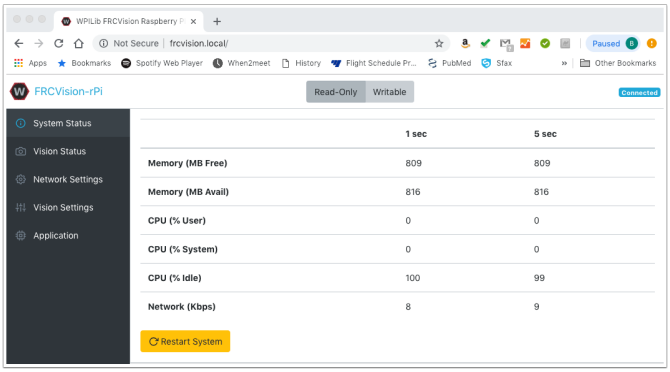
通常来说树莓派是被设置为只读的，这意味着其文件系统是无法被更改的，这保证了如果电源在树莓派的文件系统关闭前先关闭，文件系统不会被破坏。但是当一些设置被改变了 (接下来的内容)，那么新的设置将无法被保存在树莓派的文件系统当中。提供了按钮来允许文件系统设置为只读或者读写。如果按钮无法被点击，请查看系统的只读状态。

树莓派的网络连接状态

在终端的右上角有一个标签显示树🄴派是否处于连接状态。如果树🄴派断开了连接，那么标签就会从 Connected 变为 Disconnected。



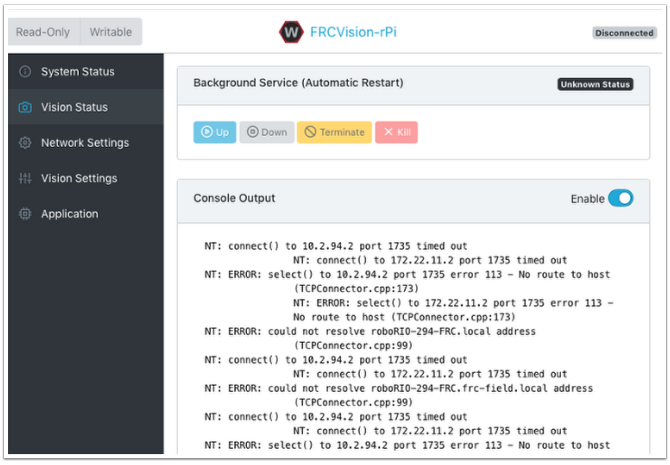
系统状态



系统状态显示了在任何时候树莓派的 CPU 状态。这里有两列状态值：第一列是在过去 1s 当中的平均值，第二列是在过去 5s 当中的平均状态。如下：

- 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

视觉状态

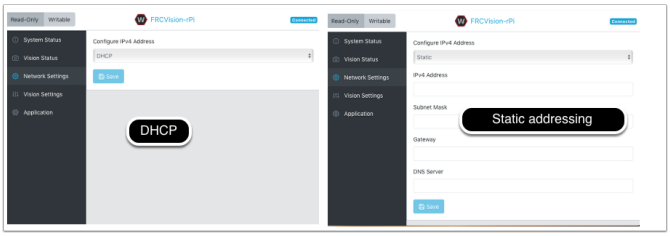


用来监控树莓派中运行摄像头程序的进程，可能是默认程序，也可能是使用 Java，C++，Python 编写的自定义程序。同样，你可以允许控制台输出并查看后台摄像头服务的输出信息。在这个示例中，树莓派只是连接到一台没有运行 NetworkTables 服务器（通常是 roboRIO）的笔记本电脑，所以将会出现很多无法连接到网络表（Network Tables）的提示信息（NT：connect( )）。

网络设置

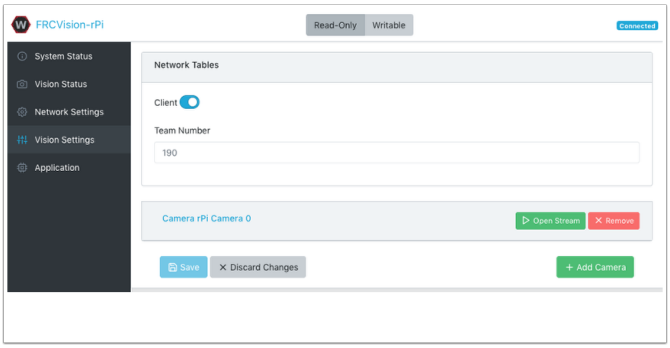
树莓派有以下几个网络连接选项：

- *DHCP* - the default name resolution usually used by the roboRIO. The default name is wpilibpi.local.
- Static - 需要明确填写静态 IP 地址，网络掩码，以及路由器信息
- DHCP with Static FallBack - 树莓派将通过 DHCP 获得 IP 地址。但如果找不到 DHCP 服务器，那么它会使用提供的静态 IP 地址和其他参数



上图描述的是 DHCP 和静态 IP 地址的设置。无论使用那种选项，树莓派的 mDNS 域名应始终有效。

视觉设置

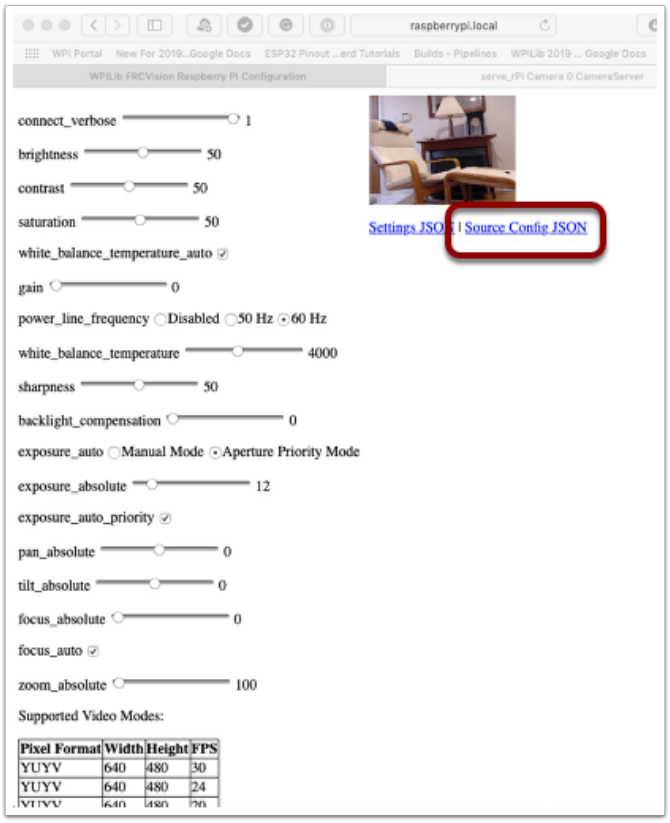


The Vision Settings are to set the parameters for each camera and whether the rPI should be a NetworkTables client or server. There can only be one server on the network and the roboRIO is always a server. Therefore when connected to a roboRIO, the rPI should always be in client mode with the team number 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).

如果要查看并更改某一摄像头的所有设置，请点击此摄像头的名称。如上图所示，摄像头的名称是 Camera rPi Camera 0。点击后，会有如下图所示的配置界面。

在这个状态下可以对摄像头进行设置。在图示页面的底部还有此摄像头所支持的所有模式 (包含长度，宽度，帧率)。

备注： 如果摄像机图像在“打开流”屏幕上不可见，请在页面底部检查支持的视频模式。然后返回“视觉设置”并单击有问题的摄像机，并验证所支持的视频模式中是否列出了像素格式，宽度，高度和 FPS。



保存现在的设置以在重启后使用

树🄵派会在启动的时候加载所有的摄像头设置。如上图所示的修改是临时的。想要让其储存下来，需要点击按钮 “Load Source Config From Camera”，之后当前设置会填入如下所示的界面。最后点击页面下方的保存按钮。提醒：必须要让文件系统处于 Writeable（读写）的状态，设置其的按钮位于页面的顶部。

上图展示了一些常用的摄像头设定值，包括亮度、白平衡、曝光。这些都会在用户的 JSON 文件被应用之前载入摄像头。所以如果用户 JSON 文件包含了这些设置，那么文本框内的这些内容将会被用户 JSON 中的相关内容覆盖。

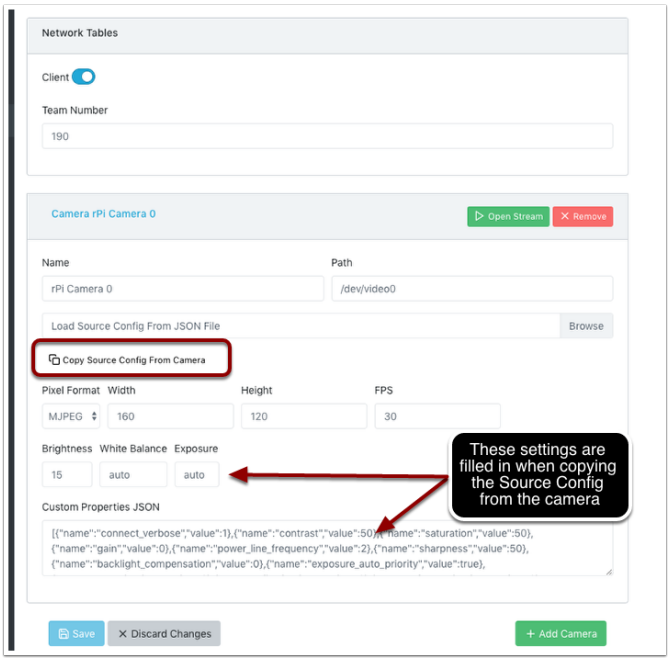
应用

应用程序一栏展示了当前正在树🄵派上运行的程序。

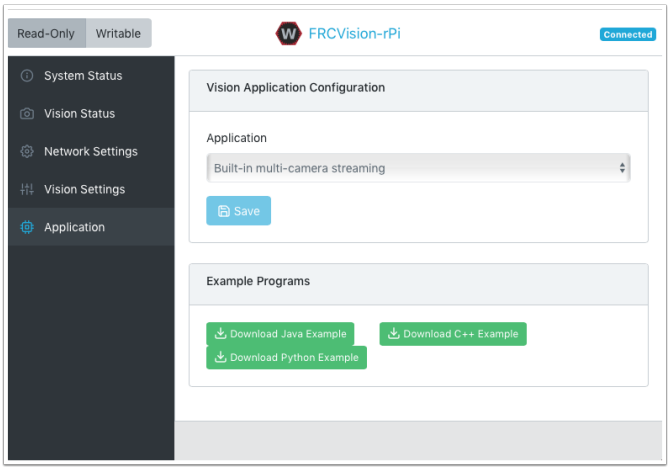
视觉工作流

对于每一种支持的编程语言：C++、Java、Python，都有 OpenCV 的样例程序。这些样例程序都可以捕获并且传输视频流到树🄵派上。另外，这些示例程序少量地使用了 OpenCV，并都可作为模板根据具体需求扩展内容。树🄵派的应用程序一栏还支持以下的程序工作流：

- 传输一个或者多个摄像头的视频到 Driver Station，并通过 ShuffleBoard 显示图像
- 在树🄵派上通过内置的工具链编辑并编译一个示例程序（每种程序语言都有对应的一个示例：C++、Java、Python）

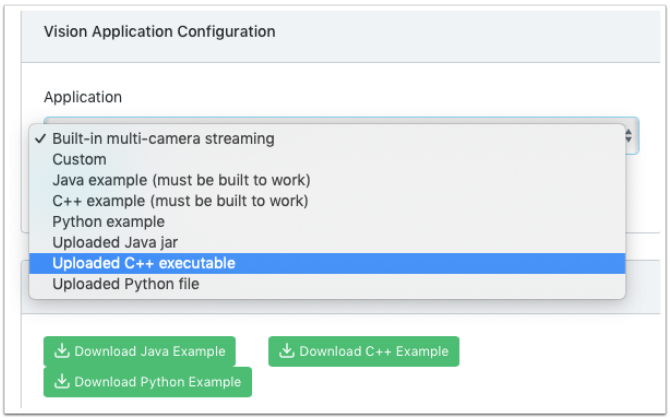


- 下载一个你所需要的编程语言的示例程序并且在开发用电脑上进行编辑和编译。之后再将其上传至树莓派
- 通过自定义的应用程序和脚本（DIY）完成所有的任务（通常是基于一个示例的）



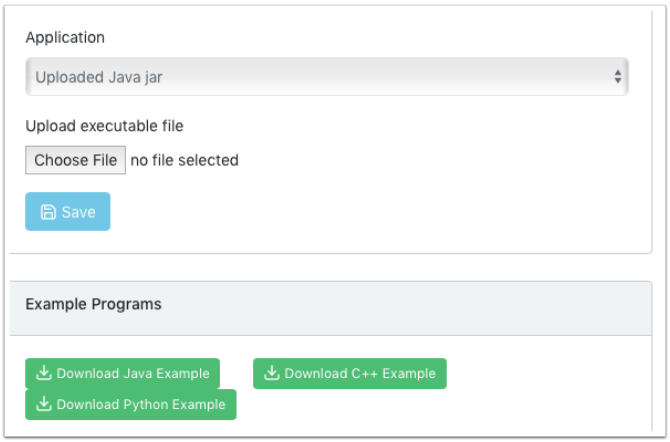
通过选择下拉菜单中的选项，可以改变正在运行的应用程序。选择有：

- 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.
- 自定义的应用程序，不会上传任何东西到树莓派，假定开发者想要运行其自己的程序和脚本
- Java, C++ or Python pre-installed sample programs that can be edited into your own application.
- Java，C ++ 或 Python 上传的程序。Java 程序需要带有已编译程序的.jar 文件，而 C ++ 程序需要将 rPi 可执行文件上载到 rPI。



当选择上传选项时，只可在文件选择窗口选择 jar 文件，可执行文件或是 Python 程序上传到树莓派。下列的图片展示了选择 Upload Java jar 选项之后，可单击“选择文件”按钮选择 jar 文件，最后点击“Save”按钮上传已选择的文件。

提醒：为了向树莓派保存新文件，必须要让文件系统处于 Writeable（读写）的状态，而设置其的按钮位于页面的上方。但是在保存之后，建议将其改回只读状态，这样可以保护文件系统不被误操作。



25.2.7 CameraServer 的应用

从 CameraServer 抓取帧

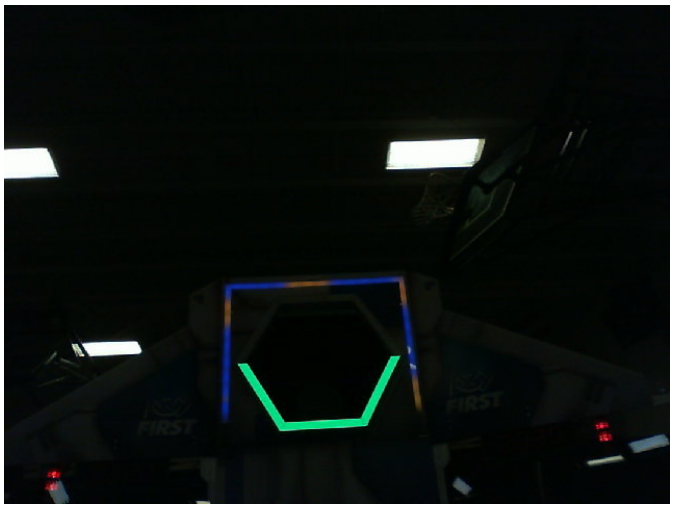
WPILibPi 图像带有所有必要的库，以构成您自己的视觉处理系统。为了从摄像机获取当前帧，可以使用 CameraServer 库。有关 CameraServer 的信息，请参阅：docs / software / vision-processing / introduction / cameraserver-class：读取和处理视频：CameraServer 类。

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
```

备注： 由于历史原因，OpenCV 在图像中被读为 **BGR**，而不是 **RGB**。如果你想把它转化为 RGB，需要使用 “cv2.cvtColor”。

下面是一个图片的例子，可以从 CameraServer 抓取。



向 CameraServer 发送帧

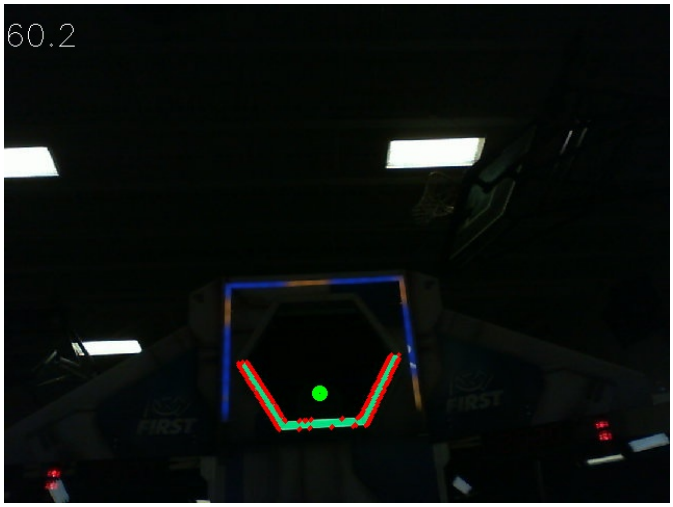
有时，你可能希望将处理过的视频帧发送回 CameraServer 实例以进行调试，或者在类似 Shuffleboard 这样的仪表板应用程序中进行查看。

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)
```

例如，处理代码可以用红色勾勒出目标，并以黄色显示角，以便调试。

下面是一个完整处理图像的示例，该图像将被发送回 CameraServer 并显示在驱动站计算机上。



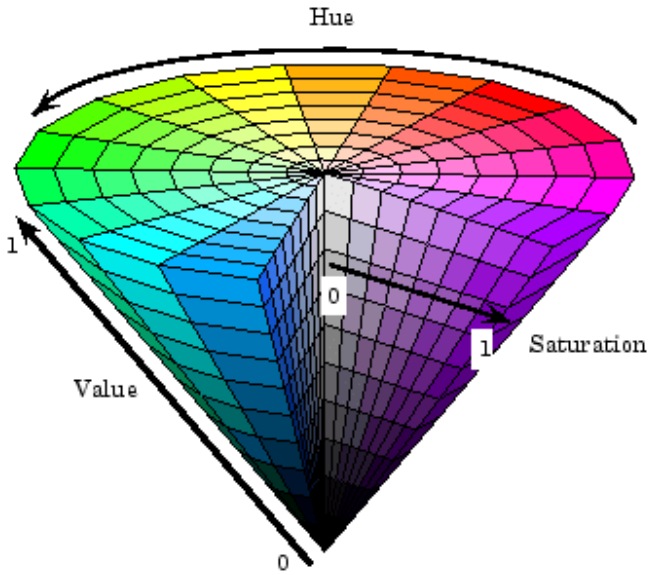
25.2.8 阈值图像

为了将彩色图像，例如由您的相机捕获的图像，转换为以目标为“前景”的二进制图像，我们需要使用每个像素的色相，饱和度和值对图像进行阈值处理。

HSV 模型

与 RGB 不同，HSV 不仅可以根据像素的颜色进行过滤，还可以根据颜色的强度和亮度进行过滤。

- 色相：测量像素的颜色。
- 饱和度：测量像素的颜色强度。
- Value: Measures the brightness of the pixel.



您可以使用 OpenCV 将 BGR 图像矩阵转换为 HSV。

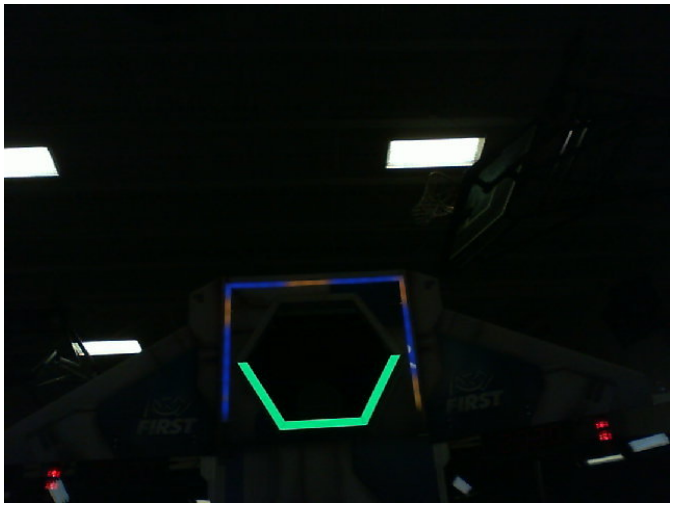
PY

```
hsv_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2HSV)
```

备注： OpenCV 的色相范围从 1° 到 180°, 而不是普通的 1° 到 360°。为了将通用色相值转换为 OpenCV, 请除以 2。

阈值

我们将以现场图像为例来说明整个图像处理过程。



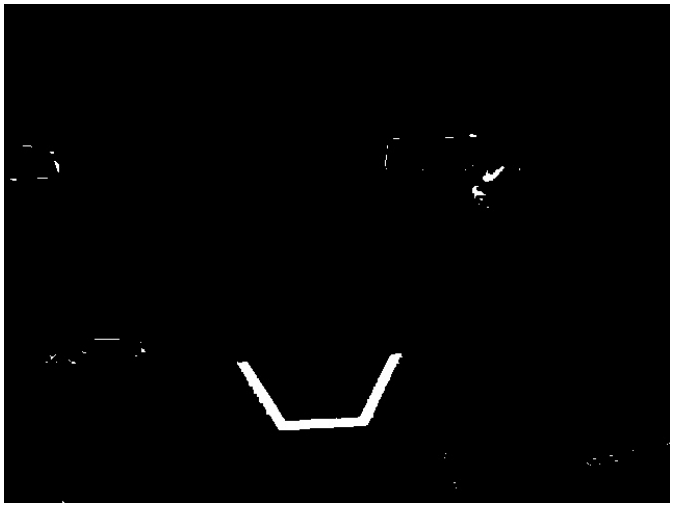
通过使用 HSV 对图像进行阈值处理，可以将图像分为视觉目标（前景）和相机看到的其他物体（背景）。以下代码示例通过使用 HSV 值进行阈值处理将 HSV 图像转换为二进制图像。

PY

```
binary_img = cv2.inRange(hsv_img, (min_hue, min_sat, min_val), (max_hue, max_sat, max_
↪val))
```

备注： 这些值可能必须在每个场所进行调整，因为不同场所的环境照明可能会有所不同。建议通过 NetworkTables 编辑这些值，以便于即时编辑。

阈值化后，您的图像应如下所示。



如您所见，阈值处理可能不是 100%干净。您可以使用形态学运算来处理噪声。

25.2.9 形态学运算

有时，在对图像进行阈值处理后，二进制图像中会出现不必要的噪点。形态学操作可以帮助去除图像中的噪点。

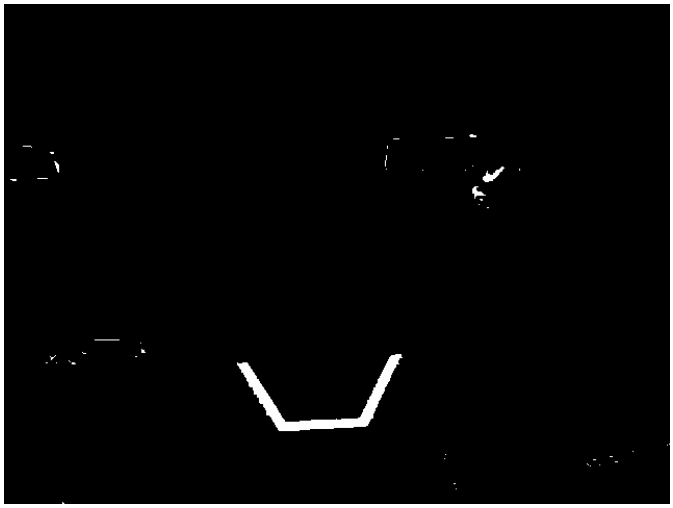
核心

核心是简单的形状，其中原点叠加在二进制图像值 1 的每个像素上。OpenCV 将内核限制为 NxN 矩阵，其中 N 为奇数。内核的起源是中心。一个常见的内核是

$$kernel = \ beginpmatrix1 1 1$$
$$1 1 1$$
$$1 1 1 \ endpmatrix$$

不同的核心可以对图像产生不同的影响，例如仅侵蚀或垂直扩展。

作为参考，这是我们创建的二进制映像：

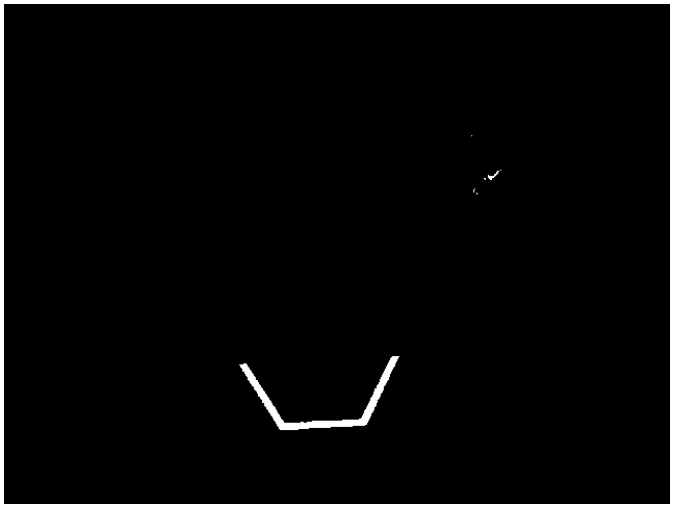


侵蚀

计算机视觉中的侵蚀类似于土壤上的侵蚀。它消除了前景对象的边界。此过程可以消除背景噪音。

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.erode(binary_img, kernel, iterations = 1)
```



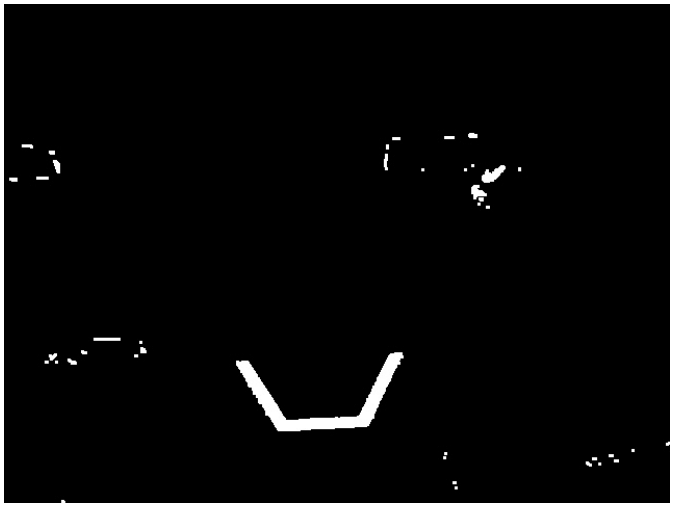
在腐蚀过程中，如果二进制图像的像素没有完全包含叠加的内核像素，则将其叠加的像素删除。

扩张

扩张与侵蚀相反。它没有远离边界，反而增加了边界。此过程可以去除较大区域内的小孔。

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.dilate(binary_img, kernel, iterations = 1)
```



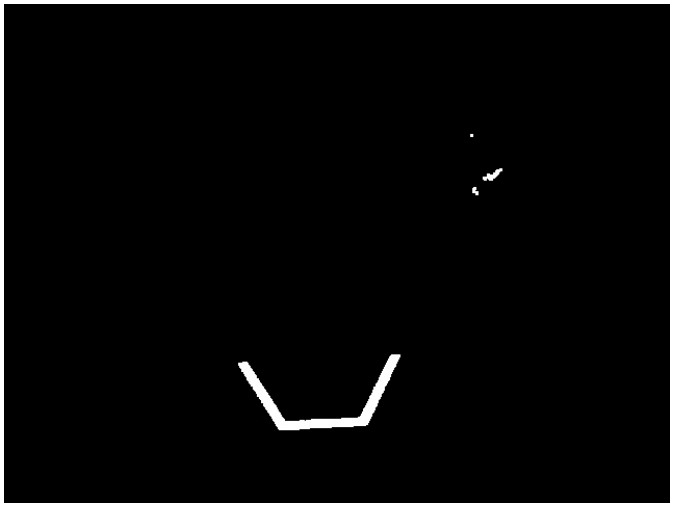
在扩张过程中，每个叠加核的每个像素都包含在扩张中。

开放

开放是侵蚀，然后是扩张。此过程可消除噪声，而不会影响较大特征的形状。

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel)
```



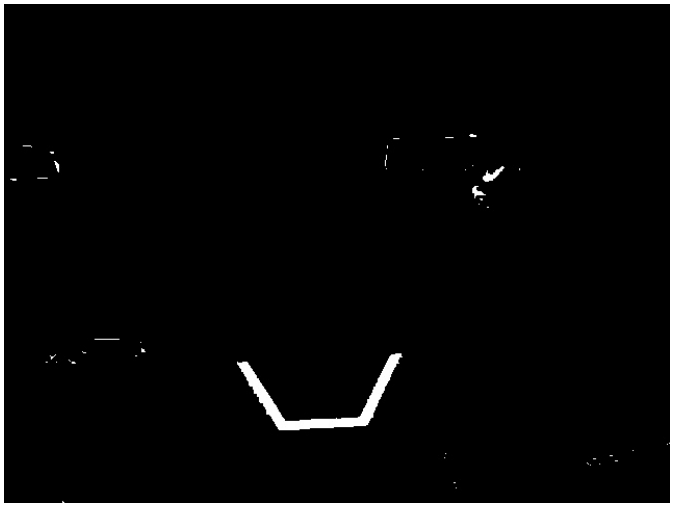
备注： 在这种特定情况下，为了消除右上角的像素，应该进行更多次开放迭代。

关闭

关闭是扩张，然后是侵蚀。此过程可去除小孔或裂口，而不会影响较大特征的形状。

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, kernel)
```



25.2.10 使用轮廓

在使用形态学操作对阈值进行处理并消除了噪声之后，您现在就可以使用 OpenCV 的 “findContours” 方法了。此方法使您可以根据二进制图像生成轮廓。

查找和过滤轮廓

PY

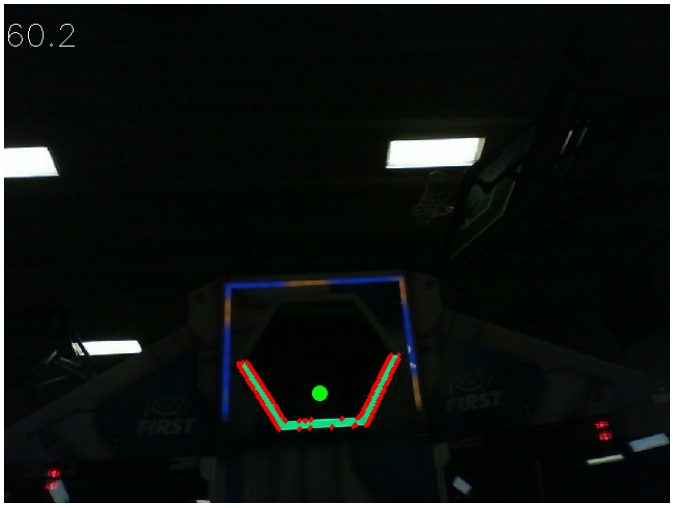
```
_, contours, _ = cv2.findContours(binary_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
↪SIMPLE)
```

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
   #
```

如果绘制刚发现的轮廓，则外观应如下所示：



从轮廓提取信息

现在，您已经找到了所需的轮廓，现在想要获取有关轮廓的信息，例如中心，角和旋转。

中心

PY

```
rect = cv2.minAreaRect(contour)
center, _, _ = rect
center_x, center_y = center
```

角

PY

```
corners = cv2.convexHull(contour)
corners = cv2.approxPolyDP(corners, 0.1 * cv2.arcLength(contour), True)
```

旋转

PY

```
_, _, rotation = cv2.fitEllipse(contour)
```

有关如何使用这些值的更多信息，请参阅：docs / software / vision-processing / introduction / identification-and-processing-the-targets：Measurements

发布到 NetworkTables

您可以使用 NetworkTables 将这些属性发送到 Driver Station 和 RoboRIO。可以在 Raspberry Pi 或 RoboRIO 本身上进行其他处理。

PY

```
import ntcore

nt = ntcore.NetworkTableInstance.getDefault().getTable('vision')

#
# Initialization code here
#

while True:

   #
   # Image processing code here
   #

   nt.putNumber('center_x', center_x)
   nt.putNumber('center_y', center_y)
```

25.2.11 基本视觉示例

在本例子中，通过设置基础视觉来展示在位置坐标系统中将目标的位置传输到网络表中：ref:*here <docs/software/vision-processing/introduction/identifying-and-processing-the-targets:Measurements>* ，并使用 CameraServer 生成一个矩形来展示检测到的目标的边界。此示例将会显示发送到 CameraServer 上的图像处理代码的帧率。

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()
```

(续下页)

(接上页)

```
    CameraServer.startAutomaticCapture()

    input_stream = CameraServer.getVideo()
    output_stream = CameraServer.putVideo('Processed', width, height)

    # Table for vision output information
    vision_nt = nt.getTable('Vision')

    # Allocating new images is very expensive, always try to preallocate
    img = np.zeros(shape=(240, 320, 3), dtype=np.uint8)

    # Wait for NetworkTables to start
    time.sleep(0.5)

    while True:
        start_time = time.time()

        frame_time, input_img = input_stream.grabFrame(img)
        output_img = np.copy(input_img)

        # Notify output of error and skip iteration
        if frame_time == 0:
            output_stream.notifyError(input_stream.getError())
            continue

        # Convert to HSV and threshold image
        hsv_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2HSV)
        binary_img = cv2.inRange(hsv_img, (0, 0, 100), (85, 255, 255))

        _, contour_list = cv2.findContours(binary_img, mode=cv2.RETR_EXTERNAL,␣
↪method=cv2.CHAIN_APPROX_SIMPLE)

        x_list = []
        y_list = []

        for contour in contour_list:

            # Ignore small contours that could be because of noise/bad thresholding
            if cv2.contourArea(contour) < 15:
                continue

            cv2.drawContours(output_img, contour, -1, color=(255, 255, 255),␣
↪thickness=-1)

            rect = cv2.minAreaRect(contour)
            center, size, angle = rect
            center = tuple([int(dim) for dim in center])  # Convert to int so we can␣
↪draw

            # Draw rectangle and circle
            cv2.drawContours(output_img, [cv2.boxPoints(rect).astype(int)], -1,␣
↪color=(0, 0, 255), thickness=2)
            cv2.circle(output_img, center=center, radius=3, color=(0, 0, 255),␣
↪thickness=-1)

            x_list.append((center[0] - width / 2) / (width / 2))
```

(续下页)

(接上页)

```
            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 What Are AprilTags?

AprilTags are a system of visual tags developed by researchers at the University of Michigan to provide low overhead, high accuracy localization for many different applications.

Additional information about the tag system and its creators can be found on their website. This document attempts to summarize the content for FIRST robotics related purposes.

Application to FRC

In the context of FRC, AprilTags are useful for helping your robot know where it is at on the field, so it can align itself to some goal position.

AprilTags have been in development since 2011, and have been refined over the years to increase the robustness and speed of detection.

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.

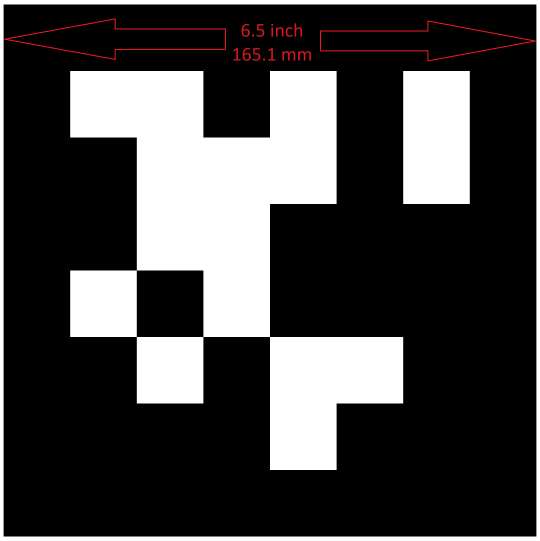
The AprilTag library implementation defines standards on how sets of tags should be designed. Some of the possible tag families are described here.

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.

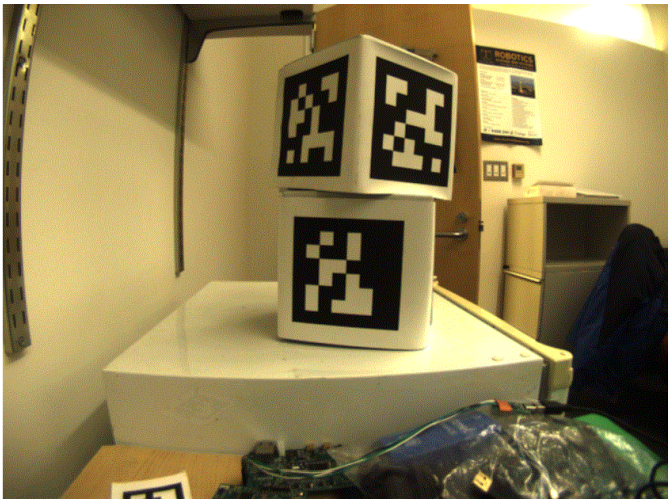
WPILib has forked the repository to add new features for FRC. These include:

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.

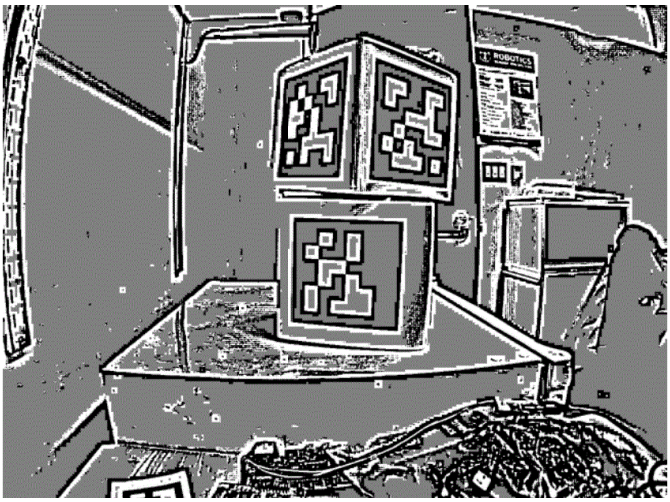
Remove Colors

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.

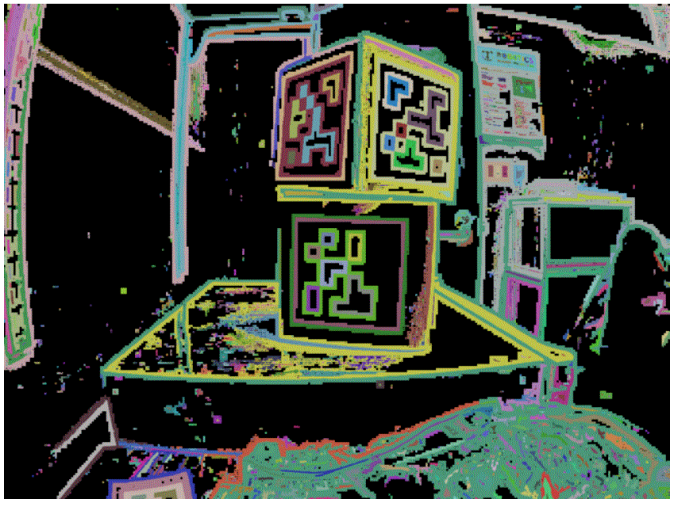
Adaptive Threshold



An adaptive threshold algorithm is run to classify each pixel as "definitely light", "definitely dark", or "not sure".

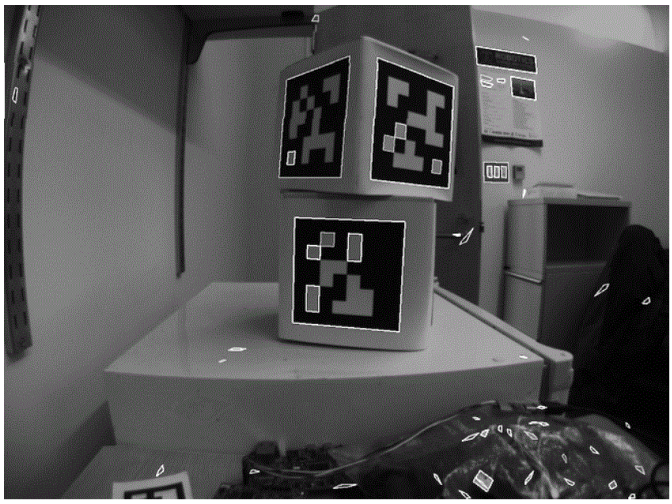
The threshold is calculated by looking at the pixel's brightness, compared to a small neighborhood of pixels around it.

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.

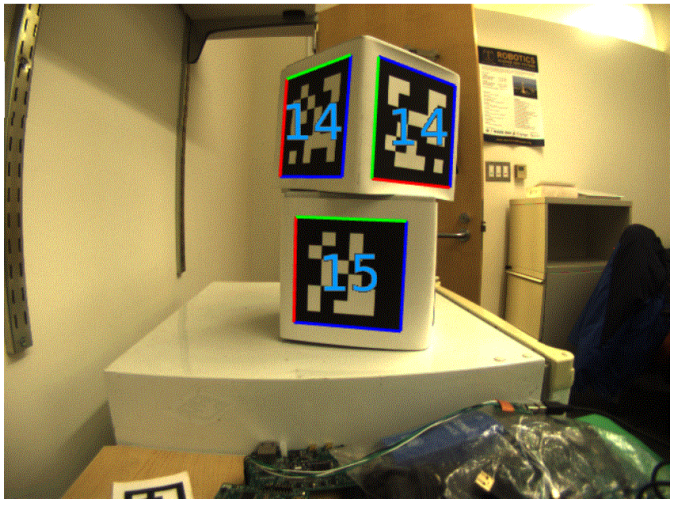
Decode ID

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

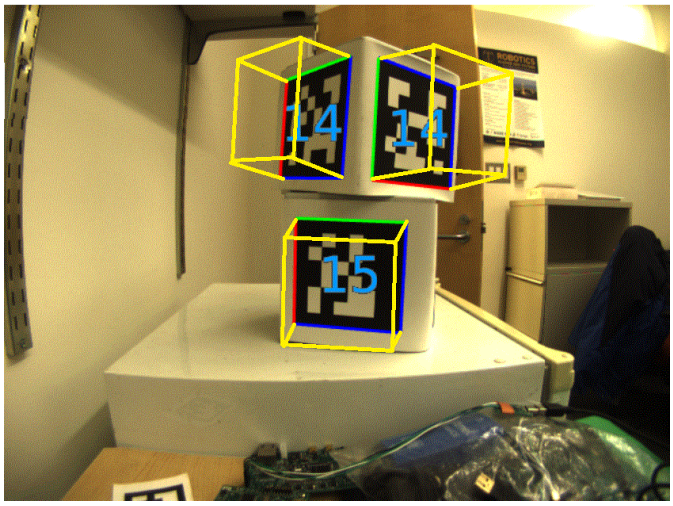
2D Alignment

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.

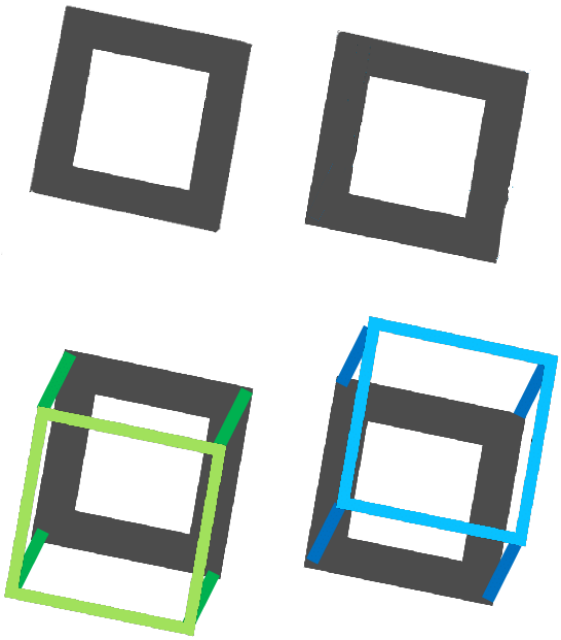
If the camera's position on the robot is known, the robot's position on the field can also be estimated.

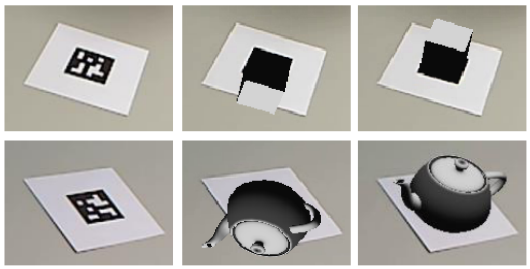
These estimates can be incorporated into the WPILib pose estimation classes.

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:





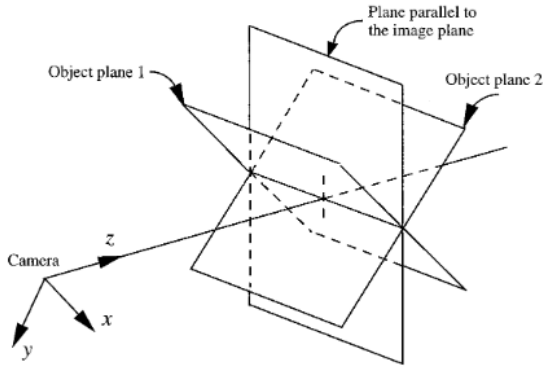


FIG. 4. Two object poses giving the same image under the SOP approximation.

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:

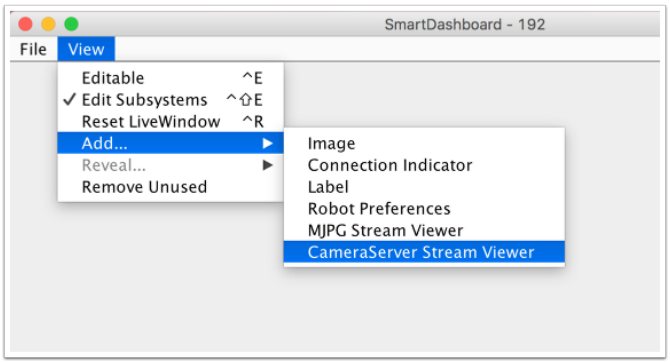
- AprilTags v1
- AprilTags v2
- AprilTags v3
- Pose Ambiguity

25.4 关于 RoboRIO 的视觉

25.4.1 使用 roboRIO 上的 CameraServer

简易 CameraServer 程序

下面的程序会启动 USB 相机的自动捕获，就像连接到 roboRIO 的 Microsoft LifeCam。在这种模式下，相机将捕捉帧并将它们发送到仪表盘。要查看图片，使用 "view" 创建一个 CameraServer tream Viewer 工具窗，然后在仪表板上创建 "Add" 菜单。图像未经过处理，只是从相机转发到仪表板。



JAVA

```
7  import edu.wpi.first.cameraserver.CameraServer;
8  import edu.wpi.first.wpilibj.TimedRobot;
9
10 /**
11  * Uses the CameraServer class to automatically capture video from a USB webcam and␣
   ↪send it to the
12  * FRC dashboard without doing any vision processing. This is the easiest way to get␣
   ↪camera images
13  * to the dashboard. Just add this to the robotInit() method in your program.
14  */
15 public class Robot extends TimedRobot {
16   @Override
17   public void robotInit() {
18     CameraServer.startAutomaticCapture();
19   }
20 }
```

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.
    """
```

高级相机服务器程序

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;
```

(续下页)

(接上页)

```
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
                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
      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);
```

(续下页)

(接上页)

```
    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.

> **警告：** Image processing is a CPU intensive task, and because of the Python Global Interpreter Lock (GIL) **we do NOT recommend using cscore directly in your robot process**. Don't do it. Really.
>
> For more information on the GIL and its effects, you may wish to read the following resources:
>
> - Python Wiki: Global Interpreter Lock
> - Efficiently Exploiting Multiple Cores with Python

This introduces a number of rules that your image processing code must follow to efficiently and safely run on the RoboRIO:

- Your image processing code must be in its own file
- Never import the `cscore` package from your robot code, it will just waste memory
- Never import the `wpilib` or `hal` packages from your image processing file
- The camera code will be killed when the `robot.py` program exits. If you wish to perform cleanup, you should register an atexit handler.

> **警告：** `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:

```
1  import wpilib
2
3
4  class MyRobot(wpilib.TimedRobot):
5      """
6      This is a demo program showing the use of OpenCV to do vision processing. The␣
   ↪image is acquired
7      from the USB camera, then a rectangle is put on the image and sent to the␣
   ↪dashboard. OpenCV has
```

(续下页)

(接上页)

```
    many methods for different types of processing.
    """

```

The `launch("vision.py")` function says to launch `vision.py` and call the `run` function in that file. Here's what is in `vision.py`:

```
# NOTE: This code runs in its own process, so we cannot access the robot here,
#       nor can we create/use/see wpilib objects
#
# To try this code out locally (if you have robotpy-cscore installed), you
# can execute `python3 -m cscore vision.py:main`
#

import cv2
import numpy as np

from cscore import CameraServer as CS


def main():
    CS.enableLogging()

    # Get the UsbCamera from CameraServer
    camera = CS.startAutomaticCapture()
    # Set the resolution
    camera.setResolution(640, 480)

    # Get a CvSink. This will capture images from the camera
    cvSink = CS.getVideo()
    # Setup a CvSource. This will send images back to the Dashboard
    outputStream = CS.putVideo("Rectangle", 640, 480)

    # Allocating new images is very expensive, always try to preallocate
    mat = np.zeros(shape=(480, 640, 3), dtype=np.uint8)

    while True:
        # Tell the CvSink to grab a frame from the camera and put it
        # in the source image.  If there is an error notify the output.
        time, mat = cvSink.grabFrame(mat)
        if time == 0:
            # Send the output the error.
            outputStream.notifyError(cvSink.getError())
            # skip the rest of the current iteration
            continue

        # Put a rectangle on the image
        cv2.rectangle(mat, (100, 100), (400, 400), (255, 255, 255), 5)

        # Give the output stream a new image to display
        outputStream.putFrame(mat)
```

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 使用多台相机

切换驱动程序视图

如果您只想更改驱动程序的外观并正在使用 SmartDashboard，则 SmartDashboard CameraServer Stream Viewer 具有一个选项（“选定的相机路径”），该选项读取给定的：term：‘NetworkTables‘键并更改 “相机选择”” 设置为该值（显示该摄像机）。然后，机器人代码只需要将 NetworkTables 键设置为正确的摄像机名称即可。假设 “选定的摄像机路径” 设置为 “CameraSelection”，下面的代码使用操纵杆 1 触发按钮状态来显示 camera1 和 camera2。

Java

```
UsbCamera camera1;
UsbCamera camera2;
Joystick joy1 = new Joystick(0);
NetworkTableEntry cameraSelection;

@Override
public void robotInit() {
    camera1 = CameraServer.startAutomaticCapture(0);
    camera2 = CameraServer.startAutomaticCapture(1);

    cameraSelection = NetworkTableInstance.getDefault().getTable("").getEntry(
↪"CameraSelection");
}

@Override
public void teleopPeriodic() {
    if (joy1.getTriggerPressed()) {
        System.out.println("Setting camera 2");
        cameraSelection.setString(camera2.getName());
    } else if (joy1.getTriggerReleased()) {
        System.out.println("Setting camera 1");
        cameraSelection.setString(camera1.getName());
    }
}
```

C++

```
cs::UsbCamera camera1;
cs::UsbCamera camera2;
frc::Joystick joy1{0};

nt::NetworkTableEntry cameraSelection;

void RobotInit() override {
  camera1 = frc::CameraServer::StartAutomaticCapture(0);
  camera2 = frc::CameraServer::StartAutomaticCapture(1);

  cameraSelection = nt::NetworkTableInstance::GetDefault().GetTable("")->GetEntry(
↪"CameraSelection");
}
```

(续下页)

(接上页)

```
void TeleopPeriodic() override {
  if (joy1.GetTriggerPressed()) {
    std::cout << "Setting Camera 2" << std::endl;
    cameraSelection.SetString(camera2.GetName());
  } else if (joy1.GetTriggerReleased()) {
    std::cout << "Setting Camera 1" << std::endl;
    cameraSelection.SetString(camera1.GetName());
  }
}
```

PYTHON

`robot.py` contents:

```
import wpilib
from ntcore import NetworkTableInstance

class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        self.joy1 = wpilib.Joystick(0)
        self.cameraSelection = NetworkTableInstance.getDefault().getTable("").
↪getEntry("CameraSelection")
        wpilib.CameraServer.launch("vision.py:main")

    def teleopPeriodic(self):
        if self.joy1.getTriggerPressed():
            print("Setting camera 2")
            self.cameraSelection.setString("USB Camera 1")
        elif self.joy1.getTriggerReleased():
            print("Setting camera 1")
            self.cameraSelection.setString("USB Camera 0")
```

`vision.py` contents:

```
from cscore import CameraServer

def main():
    CameraServer.enableLogging()

    camera1 = CameraServer.startAutomaticCapture(0)
    camera2 = CameraServer.startAutomaticCapture(1)

    CameraServer.waitForever()
```

如果使用其他仪表板，则可以通过动态更改相机服务器来改变所使用的相机。如果你在名称上选择打开 camera1 的数据流查看器，则机器人代码将根据操纵杆触发器将流内容更改为 camera1 或 camera2

JAVA

```
UsbCamera camera1;
UsbCamera camera2;
VideoSink server;
Joystick joy1 = new Joystick(0);

@Override
public void robotInit() {
    camera1 = CameraServer.startAutomaticCapture(0);
    camera2 = CameraServer.startAutomaticCapture(1);
    server = CameraServer.getServer();
}

@Override
public void teleopPeriodic() {
    if (joy1.getTriggerPressed()) {
        System.out.println("Setting camera 2");
        server.setSource(camera2);
    } else if (joy1.getTriggerReleased()) {
        System.out.println("Setting camera 1");
        server.setSource(camera1);
    }
}
```

C++

```
cs::UsbCamera camera1;
cs::UsbCamera camera2;
cs::VideoSink server;
frc::Joystick joy1{0};
bool prevTrigger = false;

void RobotInit() override {
  camera1 = frc::CameraServer::StartAutomaticCapture(0);
  camera2 = frc::CameraServer::StartAutomaticCapture(1);
  server = frc::CameraServer::GetServer();
}

void TeleopPeriodic() override {
  if (joy1.GetTrigger() && !prevTrigger) {
    std::cout << "Setting Camera 2" << std::endl;
    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
```

保持数据流开放

默认情况下，cscore 库通常主动地关闭未使用的相机。这意味着当您切换相机时，它可能会与未使用的相机断开连接，因此，在重新连接到相机时，切换回会有一些延迟。即使在没有使用的情况下，也要保持两个相机的连接都打开，请使用“SetConnectionStrategy ()”方法让库保持数据流开放

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.
```

(续下页)

(接上页)

```
↪SetConnectionStrategy(cs::VideoSource::ConnectionStrategy::kConnectionKeepOpen);
  camera2.
↪SetConnectionStrategy(cs::VideoSource::ConnectionStrategy::kConnectionKeepOpen);
}

void TeleopPeriodic() override {
  if (joy1.GetTrigger() && !prevTrigger) {
    std::cout << "Setting Camera 2" << std::endl;
    server.SetSource(camera2);
  } else if (!joy1.GetTrigger() && prevTrigger) {
    std::cout << "Setting Camera 1" << std::endl;
    server.SetSource(camera1);
  }
  prevTrigger = joy1.GetTrigger();
}
```

PYTHON

robot.py contents:

```
import wpilib
from ntcore import NetworkTableInstance

class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        self.joy1 = wpilib.Joystick(0)
        self.cameraSelection = NetworkTableInstance.getDefault().getTable("").
↪getEntry("CameraSelection")
        wpilib.CameraServer.launch("vision.py:main")

    def teleopPeriodic(self):
        if self.joy1.getTriggerPressed():
            print("Setting camera 2")
            self.cameraSelection.setString("USB Camera 1")
        elif self.joy1.getTriggerReleased():
            print("Setting camera 1")
            self.cameraSelection.setString("USB Camera 0")
```

vision.py contents:

```
from cscore import CameraServer, VideoSource

def main():
    CameraServer.enableLogging()

    camera1 = CameraServer.startAutomaticCapture(0)
    camera2 = CameraServer.startAutomaticCapture(1)

    camera1.setConnectionStrategy(VideoSource.ConnectionStrategy.kConnectionKeepOpen)
    camera2.setConnectionStrategy(VideoSource.ConnectionStrategy.kConnectionKeepOpen)

    CameraServer.waitForever()
```

备注： 如果两个相机都是 USB 端口，则可能会遇到分辨率更高的 USB 带宽限制，因为在所有这些情况下，

roboRIO 都将同时将数据从两个相机传输到 roboRIO（在选项 1 和 2 的短时间内，以及在选项 3 中连续输入）。从理论上讲，库有可能（仅）在选项 2 的情况下避免这种同时发生，但目前尚未实现。

不同的相机会报告的带宽使用情况有所不同，如果已达到极限，库会通过提示错误消息来提醒你：

```
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)
```

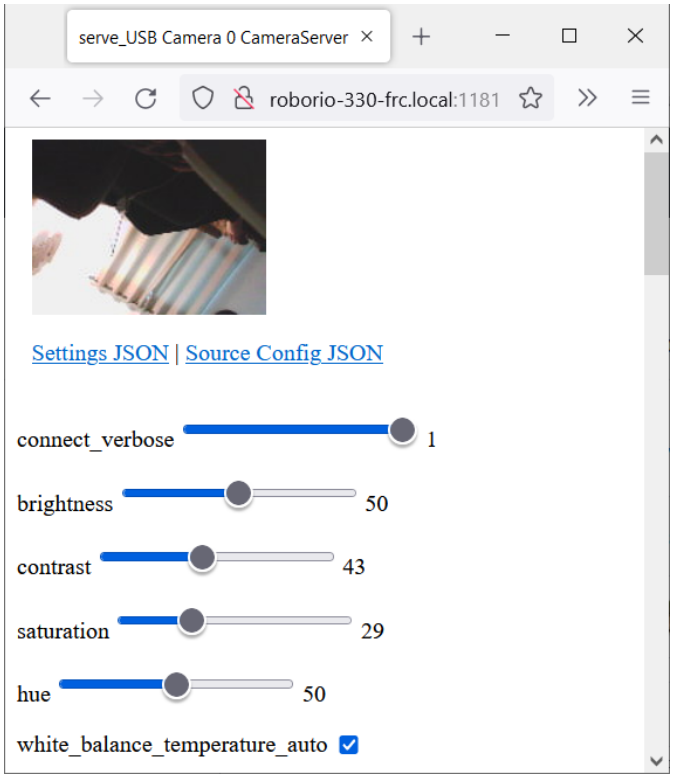
如果您使用的是选项 3，则会在“RobotInit ()”期间出现此错误。因此，你应该只尝试所需的分辨率并根据需要进行调整，直到两边都没有收到该错误提醒并且未超过带宽限制。

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 简易 *CameraServer* 程序.

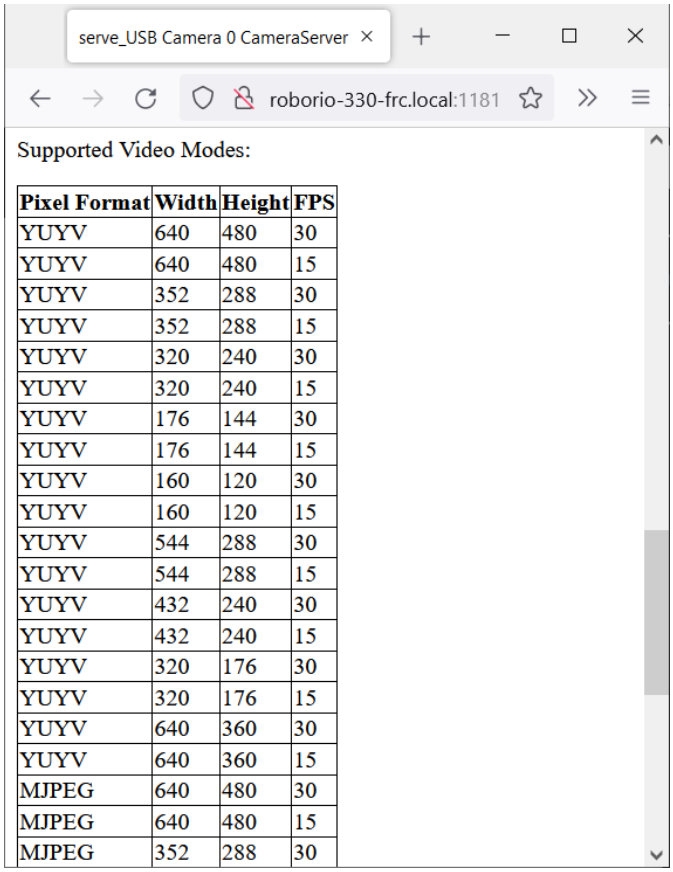
备注： The port 1181 is used for the first camera. The port increments for additional camera, so if you have two cameras, the replace 1181 above with 1182.

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

基于指令的编程

此系列文章可作为 WPILib 基于指令的框架的介绍和参考。

有关使用基于指令的框架的示例项目的集合，请参见:ref:*docs/software/examples-tutorials/wpilib-examples:Command-Based Examples*。

26.1 什么是“Command-Based”编程？

WPILib 支持一种称为“基于命令”编程的机器人编程方法。通常，“基于命令的”既可以引用一般的编程范例，也可以引用为方便起见而包括的 WPILib 库资源集。

“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.

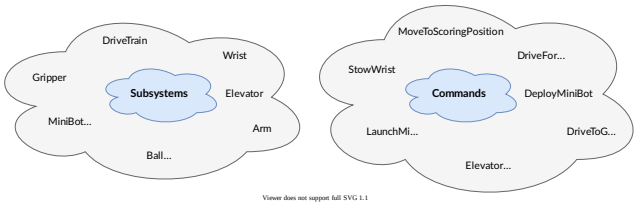
JAVA

```
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 子系统和指令



基于命令的编程模式基于两个核心抽象类：**指令 ** 和 ** 子系统**。

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 指令 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 指令如何运行

备注: 有关更详细的说明，请参考: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 指令

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 指令的结构

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).

初始化

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.

执行

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.

结束

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()`.

指定结束条件

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.

重要： 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`.

备注： 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

```
25  /** Grabs the hatch. */
26  public Command grabHatchCommand() {
27    // implicitly require `this`
28    return this.runOnce(() -> m_hatchSolenoid.set(kForward));
29  }
30
31  /** Releases the hatch. */
32  public Command releaseHatchCommand() {
33    // implicitly require `this`
34    return this.runOnce(() -> m_hatchSolenoid.set(kReverse));
35  }
```

C++ (Header)

```
20  /**
21   * Grabs the hatch.
22   */
23  frc2::CommandPtr GrabHatchCommand();
24
25  /**
26   * Releases the hatch.
27   */
28  frc2::CommandPtr ReleaseHatchCommand();
```

C++ (Source)

```
frc2::CommandPtr HatchSubsystem::GrabHatchCommand() {
  // implicitly require `this`
  return this->RunOnce(
      [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kForward); });
}

frc2::CommandPtr HatchSubsystem::ReleaseHatchCommand() {
  // implicitly require `this`
  return this->RunOnce(
      [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kReverse); });
}
```

The `run` factory, backed by the `RunCommand` (Java, C++) class, creates a command that calls a lambda repeatedly, until interrupted.

JAVA

```
// A split-stick arcade command, with forward/backward controlled by the left
// hand, and turning controlled by the right.
new RunCommand(() -> m_robotDrive.arcadeDrive(
    -driverController.getLeftY(),
    driverController.getRightX()),
    m_robotDrive)
```

C++

```
// A split-stick arcade command, with forward/backward controlled by the left
// hand, and turning controlled by the right.
frc2::RunCommand(
  [this] {
    m_drive.ArcadeDrive(
        -m_driverController.GetLeftY(),
        m_driverController.GetRightX());
  },
  {&m_drive}))
```

The `startEnd` factory, backed by the `StartEndCommand` (Java, C++) class, calls one lambda when scheduled, and then a second lambda when interrupted.

JAVA

```
Commands.startEnd(
    // Start a flywheel spinning at 50% power
    () -> m_shooter.shooterSpeed(0.5),
    // Stop the flywheel at the end of the command
    () -> m_shooter.shooterSpeed(0.0),
    // Requires the shooter subsystem
    m_shooter
)
```

C++

```
frc2::cmd::StartEnd(
  // Start a flywheel spinning at 50% power
  [this] { m_shooter.shooterSpeed(0.5); },
  // Stop the flywheel at the end of the command
  [this] { m_shooter.shooterSpeed(0.0); },
  // Requires the shooter subsystem
  {&m_shooter}
)
```

`FunctionalCommand` (Java, C++) accepts four lambdas that constitute the four command lifecycle methods: a `Runnable`/`std::function<void()>` for each of `initialize()` and `execute()`, a `BooleanConsumer`/`std::function<void(bool)>` for `end()`, and a `BooleanSupplier`/`std::function<bool()>` for `isFinished()`.

JAVA

```
new FunctionalCommand(
    // Reset encoders on command start
    m_robotDrive::resetEncoders,
    // Start driving forward at the start of the command
    () -> m_robotDrive.arcadeDrive(kAutoDriveSpeed, 0),
    // Stop driving at the end of the command
    interrupted -> m_robotDrive.arcadeDrive(0, 0),
    // End the command when the robot's driven distance exceeds the desired value
    () -> m_robotDrive.getAverageEncoderDistance() >= kAutoDriveDistanceInches,
    // Require the drive subsystem
    m_robotDrive
)
```

C++

```
frc2::FunctionalCommand(
  // Reset encoders on command start
  [this] { m_drive.ResetEncoders(); },
  // Start driving forward at the start of the command
  [this] { m_drive.ArcadeDrive(ac::kAutoDriveSpeed, 0); },
  // Stop driving at the end of the command
  [this] (bool interrupted) { m_drive.ArcadeDrive(0, 0); },
  // End the command when the robot's driven distance exceeds the desired value
  [this] { return m_drive.GetAverageEncoderDistance() >= kAutoDriveDistanceInches; },
  // Requires the drive subsystem
  {&m_drive}
)
```

To print a string and ending immediately, the library offers the `Commands.print(String)`/`frc2::cmd::Print(std::string_view)` factory, backed by the `PrintCommand` (Java, C++) subclass of `InstantCommand`.

Waiting

Waiting for a certain condition to happen or adding a delay can be useful to synchronize between different commands in a command composition or between other robot actions.

To wait and end after a specified period of time elapses, the library offers the `Commands.waitSeconds(double)`/`frc2::cmd::Wait(units::second_t)` factory, backed by the `WaitCommand` (Java, C++) class.

JAVA

```
// Ends 5 seconds after being scheduled
new WaitCommand(5.0)
```

C++

```
// Ends 5 seconds after being scheduled
frc2::WaitCommand(5.0_s)
```

To wait until a certain condition becomes `true`, the library offers the `Commands.waitUntil(BooleanSupplier)`/`frc2::cmd::WaitUntil(std::function<bool()>)` factory, backed by the `WaitUntilCommand` class (Java, C++).

JAVA

```
// Ends after m_limitSwitch.get() returns true
new WaitUntilCommand(m_limitSwitch::get)
```

C++

```
// Ends after m_limitSwitch.Get() returns true
frc2::WaitUntilCommand([&m_limitSwitch] { return m_limitSwitch.Get(); })
```

Control Algorithm Commands

There are commands for various control setups:

- `PIDCommand` uses a PID controller. For more info, see *PID 指令*.
- `TrapezoidProfileCommand` tracks a trapezoid motion profile. For more info, see *TrapezoidProfileCommand*.
- `ProfiledPIDCommand` combines PID control with trapezoid motion profiles. For more info, see *ProfiledPIDCommand*.
- `MecanumControllerCommand` (Java, C++) is useful for controlling mecanum drivetrains. See API docs and the **MecanumControllerCommand** (Java, C++) example project for more info.

- `SwerveControllerCommand` (Java, C++) is useful for controlling swerve drivetrains. See API docs and the **SwerveControllerCommand** (Java, C++) example project for more info.
- `RamseteCommand` (Java, C++) is useful for path following with differential drivetrains ("tank drive"). See API docs and the *Trajectory Tutorial* for more info.

26.2.4 Custom Command Classes

Users may also write custom command classes. As this is significantly more verbose, it's recommended to use the more concise factories mentioned above.

备注: 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

```
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++

```
#pragma once

#include <frc2/command/Command.h>
#include <frc2/command/CommandHelper.h>

#include "subsystems/ExampleSubsystem.h"

/**
```

(续下页)

(接上页)

```
 * 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:
  /**
   * Creates a new ExampleCommand.
   *
   * @param subsystem The subsystem used by this command.
   */
  explicit ExampleCommand(ExampleSubsystem* subsystem);

 private:
  ExampleSubsystem* m_subsystem;
};
```

26.2.5 简单的指令示例

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();
  }
```

(续下页)

(接上页)

```
28
29    @Override
30    public boolean isFinished() {
31      return true;
32    }
33  }
```

C++ (Header)

```
5   #pragma once
6
7   #include <frc2/command/Command.h>
8   #include <frc2/command/CommandHelper.h>
9
10  #include "subsystems/HatchSubsystem.h"
11
12  /**
13   * A simple command that grabs a hatch with the HatchSubsystem.  Written
14   * explicitly for pedagogical purposes.  Actual code should inline a command
15   * this simple with InstantCommand.
16   *
17   * @see InstantCommand
18   */
19  class GrabHatch : public frc2::CommandHelper<frc2::Command, GrabHatch> {
20   public:
21    explicit GrabHatch(HatchSubsystem* subsystem);
22
23    void Initialize() override;
24
25    bool IsFinished() override;
26
27   private:
28    HatchSubsystem* m_hatch;
29  };
```

C++ (Source)

```
5   #include "commands/GrabHatch.h"
6
7   GrabHatch::GrabHatch(HatchSubsystem* subsystem) : m_hatch(subsystem) {
8     AddRequirements(subsystem);
9   }
10
11  void GrabHatch::Initialize() {
12    m_hatch->GrabHatch();
13  }
14
15  bool GrabHatch::IsFinished() {
16    return true;
17  }
```

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.

那更复杂的情况呢？以下是来自同一示例项目的驱动指令：

Java

```
5  package edu.wpi.first.wpilibj.examples.hatchbottraditional.commands;
6
7  import edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems.
   ↪DriveSubsystem;
8  import edu.wpi.first.wpilibj2.command.Command;
9  import java.util.function.DoubleSupplier;
10
11 /**
12  * A command to drive the robot with joystick input (passed in as {@link␣
   ↪DoubleSupplier}s). Written
13  * explicitly for pedagogical purposes - actual code should inline a command␣
   ↪this simple with {@link
14  * edu.wpi.first.wpilibj2.command.RunCommand}.
15  */
16 public class DefaultDrive extends Command {
17   private final DriveSubsystem m_drive;
18   private final DoubleSupplier m_forward;
19   private final DoubleSupplier m_rotation;
20
21   /**
22    * Creates a new DefaultDrive.
23    *
24    * @param subsystem The drive subsystem this command wil run on.
25    * @param forward The control input for driving forwards/backwards
26    * @param rotation The control input for turning
27    */
28   public DefaultDrive(DriveSubsystem subsystem, DoubleSupplier forward,␣
   ↪DoubleSupplier rotation) {
29     m_drive = subsystem;
30     m_forward = forward;
31     m_rotation = rotation;
32     addRequirements(m_drive);
33   }
34
35   @Override
36   public void execute() {
37     m_drive.arcadeDrive(m_forward.getAsDouble(), m_rotation.getAsDouble());
38   }
39 }
```

C++ (Header)

```cpp
#pragma once

#include <functional>

#include <frc2/command/Command.h>
#include <frc2/command/CommandHelper.h>

#include "subsystems/DriveSubsystem.h"

/**
 * A command to drive the robot with joystick input passed in through lambdas.
 * Written explicitly for pedagogical purposes - actual code should inline a
 * command this simple with RunCommand.
 *
 * @see RunCommand
 */
class DefaultDrive : public frc2::CommandHelper<frc2::Command, DefaultDrive> {
 public:
  /**
   * Creates a new DefaultDrive.
   *
   * @param subsystem The drive subsystem this command wil run on.
   * @param forward The control input for driving forwards/backwards
   * @param rotation The control input for turning
   */
  DefaultDrive(DriveSubsystem* subsystem, std::function<double()> forward,
               std::function<double()> rotation);

  void Execute() override;

 private:
  DriveSubsystem* m_drive;
  std::function<double()> m_forward;
  std::function<double()> m_rotation;
};
```

C++ (Source)

```cpp
#include "commands/DefaultDrive.h"

#include <utility>

DefaultDrive::DefaultDrive(DriveSubsystem* subsystem,
                           std::function<double()> forward,
                           std::function<double()> rotation)
    : m_drive{subsystem},
      m_forward{std::move(forward)},
      m_rotation{std::move(rotation)} {
  AddRequirements(subsystem);
}

```

(续下页)

(接上页)

```
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.

重要： 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(
```

(续下页)

(接上页)

```
                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.

> **警告:** 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.

备注: 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

```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)

```cpp
#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.
```

(续下页)

(接上页)

```
   *
   * @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 子系统

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 创建子系统

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 简单子系统示例

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

```
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) {
```

(续下页)

(接上页)

```
  SubsystemBase::InitSendable(builder);

  // Publish the solenoid state to telemetry.
  builder.AddBooleanProperty(
      "extended",
      [this] { return m_hatchSolenoid.Get() == frc::DoubleSolenoid::kForward; },
      nullptr);
}
```

请注意，子系统从外部代码隐藏了 DoubleSolenoid 的存在（被声明为 “private’’），而是公开公开了两个更高级别的描述性机器人动作：“grabHatch ()’’ 和 “releaseHatch ()’’ “。以这种方式 “隐藏” 双螺线管之类的 “实现细节” 是极为重要的；这样可以确保子系统外部的代码永远不会导致螺线管处于意外状态。它还允许用户更改实现方式（例如，可以使用电动机代替气动方式），而子系统外部的任何代码都不必随之更改。

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`
    return this.runOnce(() -> m_hatchSolenoid.set(kReverse));
  }

  @Override
```

(续下页)

(接上页)

```
  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)

```
#include "subsystems/HatchSubsystem.h"

#include <wpi/sendable/SendableBuilder.h>

using namespace HatchConstants;

HatchSubsystem::HatchSubsystem()
```

(续下页)

(接上页)

```
    : 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)

```
30   void Periodic() override;
```

C++ (Source)

```
30 void DriveSubsystem::Periodic() {
31   // Implementation of subsystem periodic method goes here.
32   m_odometry.Update(m_gyro.GetRotation2d(),
33                     units::meter_t{m_leftEncoder.GetDistance()},
34                     units::meter_t{m_rightEncoder.GetDistance()});
35   m_fieldSim.SetRobotPose(m_odometry.GetPose());
36 }
```

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

备注： 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));
```

备注： A command that is assigned as the default command for a subsystem must require that subsystem.

26.5 将命令绑定到触发器

除了在自动阶段开始时调度的自动阶段命令以及在当前不使用其子系统的情况下自动调度的缺省命令之外，运行命令的最常见方法是将其绑定到触发事件，例如人工操作员按下的按钮。基于命令的范例使此操作非常容易。

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

备注： 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.

备注： 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

```
    // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
    m_driverController.a().onTrue(m_robotArm.setArmGoalCommand(2));
```

C++

```
  // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
  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

```
    // While holding the shoulder button, drive at half speed
    new JoystickButton(m_driverController, Button.kRightBumper.value)
        .whileTrue(new HalveDriveSpeed(m_robotDrive));
```

C++

```
  // While holding the shoulder button, drive at half speed
  frc2::JoystickButton(&m_driverController,
                       frc::XboxController::Button::kRightBumper)
      .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 构成触发器

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 构建基于命令的机器人项目

用户可以随意使用自己喜欢的基于命令的库（鼓励高级用户使用），但是新用户可能需要一些有关如何构建基本的基于命令的机器人项目的指南。

WPILib 示例存储库中包含基于命令的机器人项目的标准模板（Java <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/templates/commandbased> __，C ++ <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/templates/commandbased> __）。本节将向用户介绍此模板的结构。

根包/目录通常将包含四个类：

“Main”，这是主要的机器人应用程序（仅 Java）。新用户 * 不应 * 接触此类。“Robot”，负责机器人代码的主控制流程。“RobotContainer”包含机器人子系统和命令，是执行大多数声明性机器人设置（例如按钮绑定）的地方。“Constants”，其中包含可在整个机器人中使用的全局可访问常量。

根目录还将包含两个子包/子目录：“Subsystems”包含所有用户定义的子系统类。“Commands”包含所有用户定义的命令类。

26.6.1 Robot

由于 “Robot’’（Java <https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/templates/commandbased/Robot.java>‘__，**‘C ++（Header）<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/templates/co**
‘C ++（Source）<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp
负责程序的控制流，基于命令的声明式范例旨在最大程度地减少由于用户必须对显式程序控制流程给予足够的关注，基于命令的项目的“机器人”类应该基本上为空。但是，必须包含一些重要的内容

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   }
```

在 Java 中，应在“robotInit()”方法期间构造“RobotContainer”的实例——这很重要，因为大多数声明式机器人设置将从“RobotContainer”构造函数调用。

在 C ++ 中，这是不需要的，因为 RobotContainer 是值成员，并且将在构建“Robot”时进行构造。

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();
}
```

“robotPeriodic()”方法中必须包含“CommandScheduler.getInstance().run()”调用；没有此调用，调度程序将不会执行任何调度的命令。由于“TimedRobot”以默认的主循环频率 50Hz 运行，因此这是将调用定期命令和子系统方法的频率。不建议新用户从其代码中的其他任何地方调用此方法。

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();
```

(续下页)

(接上页)

```
    }
  }
```

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();
  }
}
```

“autonomousInit()”方法调度由“RobotContainer”实例返回的自治命令。选择要运行的自治命令的逻辑可以在“RobotContainer”内部处理。

Java

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

C++ (Source)

```
void Robot::TeleopInit() {
  // This makes sure that the autonomous stops running when
  // teleop starts running. If you want the autonomous to
  // continue until interrupted by another command, remove
  // this line or comment it out.
  if (m_autonomousCommand) {
    m_autonomousCommand->Cancel();
  }
}
```

“teleopInit()”方法取消所有仍在运行的自治命令。这通常是好的做法。

高级用户可以随意在自己认为合适的各种初始化和定期方法中添加其他代码；但是，应该注意的是，在“Robot.java”中包含大量命令式机器人代码与基于命令的范例的声明式设计哲学相反，并且可能导致结构混乱/结构混乱的代码。

26.6.2 RobotContainer

此类(Java,**'C ++(Header)<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/**
'C ++(Source)<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp
是大多数基于命令的机器人的设置地方。在此类中，您将定义机器人的子系统和命令，将这些命令绑定到触发事件（例如按钮），并指定将在自治例程中运行的命令。新手可能需要对此类进行解释：

Java

```
23   private final ExampleSubsystem m_exampleSubsystem = new ExampleSubsystem();
```

C++ (Header)

```
32   ExampleSubsystem m_subsystem;
```

请注意，子系统在“RobotContainer”中被声明为私有字段。这与基于命令的框架的前身形成了鲜明的对比，但与商定的面向对象的最佳实践更加一致。如果子系统被声明为全局变量，则它允许用户从代码中的任何位置访问它们。虽然这可以使某些事情变得容易（例如，无需将子系统传递给命令以使这些命令可以访问它们），但由于不立即执行，因此使得程序的控制流更难以跟踪很明显，代码的哪些部分可以更改或可以被代码的其他哪些部分更改。这也妨碍了资源管理系统执行其工作的能力，因为易于访问使用户更容易意外地在资源管理命令之外意外调用子系统方法。

Java

```
61     return Autos.exampleAuto(m_exampleSubsystem);
```

C++ (Source)

```
34   return autos::ExampleAuto(&m_subsystem);
```

由于子系统被声明为私有成员，因此必须将它们显式传递给命令（一种称为“依赖注入”的模式），以便这些命令在其上调用方法。这是通过“ExampleCommand”完成的，该示例传递了指向“ExampleSubsystem”的指针。

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␣
```

(续下页)

(接上页)

```
↪Flight
   * joysticks}.
   */
  private void configureBindings() {
    // Schedule `ExampleCommand` when `exampleCondition` changes to `true`
    new Trigger(m_exampleSubsystem::exampleCondition)
        .onTrue(new ExampleCommand(m_exampleSubsystem));

    // Schedule `exampleMethodCommand` when the Xbox controller's B button is pressed,
    // cancelling on release.
    m_driverController.b().whileTrue(m_exampleSubsystem.exampleMethodCommand());
  }
```

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)

```
32 frc2::CommandPtr RobotContainer::GetAutonomousCommand() {
33   // An example command will be run in autonomous
34   return autos::ExampleAuto(&m_subsystem);
35 }
```

最后，“getAutonomousCommand()”方法为用户提供了一种方便的方法，可以将所选的自主命令发送到主要的 Robot 类（需要在启动自主类时对其进行调度）。

26.6.3 Constants

“Constants”类(Java,**‘C ++(Header)<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExa**
C ++ 中这不是一个类，而是一个定义了多个名称空间的头文件）是可全局访问的机器人的位置可以存储常数（例如速度，单位转换系数，PID 增益和传感器/电动机端口）。建议用户将这些常量分成与子系统或机械手模式相对应的各个内部类，以使变量名更短。

在 Java 中，所有常量都应声明为“public static final”，以便它们可以全局访问且不能更改。在 C ++ 中，所有常量都应为“constexpr”。

有关实践中“constants”类的外观的更多说明性示例，请参见各种基于命令的示例项目的示例：

- FrisbeeBot（Java，C ++）
- GyroDriveCommands（Java，C ++）
- Hatchbot（Java，C ++）
- RapidReactCommandBot (Java, C++)

在 Java 中，建议通过静态导入必要的内部类从其他类中使用常量。“import static”语句将类的静态名称空间导入到您正在工作的类中，以便可以直接引用任何“static”常量，就好像它们是在该类中定义的一样。在 C ++ 中，使用“使用命名空间’’可以达到相同的效果：

JAVA

```
import static edu.wpi.first.wpilibj.templates.commandbased.Constants.OIConstants.*;
```

C++

```
using namespace OIConstants;
```

26.6.4 Subsystems

用户定义的子系统应放在此软件包/目录中。

26.6.5 Commands

用户定义的命令应放在此软件包/目录中。

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:

备注： 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:

备注: 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:

备注： 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 命令调度程序

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.

“CommandScheduler”还运行每个已注册“Subsystem”的“periodic()”方法。

26.8.1 使用命令调度程序

“CommandScheduler”是一个 *单例类*，这意味着它是一个只有一个实例的全局类。因此，为了访问调度程序，用户必须调用“CommandScheduler.getInstance()”命令。

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).

但是，有一个例外：用户必须从其“Robot”类的“robotPeriodic()”方法中调用“CommandScheduler.getInstance().run()”。如果不这样做，则调度程序将永远不会运行，并且命令框架将无法运行。提供的基于命令的项目模板已包含此调用。

26.8.2 “schedule()”方法

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;
```

(续下页)

(接上页)

```
      }
    }
    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);
    }
  }

```

(续下页)

(接上页)

```
  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 调度程序运行顺序

备注： 每个“Command”的“initialize()”方法都是在调度命令时调用的，而不必在调度程序运行时调用（除非该命令已绑定到按钮）。

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++).

步骤 1：运行子系统定期方法

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)

```
  // Run the periodic method of all registered subsystems.
  for (auto&& subsystem : m_impl->subsystems) {
    subsystem.getFirst()->Periodic();
    if constexpr (frc::RobotBase::IsSimulation()) {
      subsystem.getFirst()->SimulationPeriodic();
    }
    m_watchdog.AddEpoch("Subsystem Periodic()");
  }
```

步骤 2：轮询命令调度触发器

备注： 有关触发器绑定如何工作的更多信息，请参阅:doc:*binding-commands-to-triggers*

其次，调度程序轮询所有已注册触发器的状态，以查看是否应调度已绑定到这些触发器的任何新命令。如果满足调度绑定命令的条件，则对命令进行调度并运行其“Initialize()”方法。

Java

```
    // Poll buttons for new commands to add.
    loopCache.poll();
    m_watchdog.addEpoch("buttons.run()");
```

C++ (Source)

```
  // Poll buttons for new commands to add.
  loopCache->Poll();
  m_watchdog.AddEpoch("buttons.Run()");
```

步骤 3：运行/完成计划的命令

第三，调度程序调用每个当前调度的命令的“execute()”方法，然后通过调用“isFinished()”方法检查命令是否完成。如果命令已完成，则还将调用“end()”方法，并对该命令进行调度，并释放其所需的子系统。

请注意，此调用顺序是针对每个命令按顺序进行的——因此，一个命令可能会先调用其“end()”方法，而另一命令可能会调用其“execute()”方法。命令按计划的顺序处理。

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);
      }

```

(续下页)

(接上页)

```
      for (auto&& requirement : command->GetRequirements()) {
        m_impl->requirements.erase(requirement);
      }

      m_impl->scheduledCommands.erase(command);
      m_watchdog.AddEpoch(command->GetName() + ".End(false)");
    }
  }
```

步骤 4：调度默认命令

最后，任何已注册的“子系统”都有其默认命令（如果有的话）。注意，此时将调用默认命令的“initialize()”方法。

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 禁用调度程序

可以通过调用“CommandScheduler.getInstance().disable()”来禁用调度程序。禁用时，调度程序的“schedule()”和“run()”命令将不执行任何操作。

可以通过调用“CommandScheduler.getInstance.()enable()”来重新启用调度程序。

26.8.5 命令事件方法

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) {
```

(续下页)

(接上页)

```
      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 有关 C ++ 指令的技术讨论

备注： 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.

备注： The model was further changed in 2023, as described *below*.

26.9.1 所有权模型

基于指令的旧框架使用原始指针，这意味着用户必须在其机器人代码中使用“new’”（导致手动堆分配）。由于没有明确指示谁拥有命令（调度程序，指令组或用户自己），因此不清楚谁应该负责释放内存。

旧的基于命指令的框架中的几个示例涉及如下代码：

```
#include "PlaceSoda.h"
#include "Elevator.h"
#include "Wrist.h"

PlaceSoda::PlaceSoda() {
  AddSequential(new SetElevatorSetpoint(Elevator::TABLE_HEIGHT));
  AddSequential(new SetWristSetpoint(Wrist::PICKUP));
  AddSequential(new OpenClaw());
}
```

In the command-group above, the component commands of the command group were being heap allocated and passed into `AddSequential` all in the same line. This meant that user had no reference to that object in memory and therefore had no means of freeing the allocated memory once the command group ended. The command group itself never freed the memory and neither did the command scheduler. This led to memory leaks in robot programs (i.e. memory was allocated on the heap but never freed).

这个明显的问题是重写框架的原因之一。这次重写引入了一个全面的所有权模型，以及智能指针的使用，这些智能指针将在超出范围时自动释放内存。

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"

使用"std::unique_ptr"以让我们清楚地确定谁拥有对象。由于无法复制"std::unique_ptr", 因此不会有超过一个"std::unique_ptr"实例指向堆上的同一内存块。例如，"SequentialCommandGroup"的构造函数采用"std :: vector <std::unique_ptr<Command» &&"。这意味着它需要对向量"std::unique_ptr <Command>"的右值引用。让我们逐步看一些示例代码，以更好地理解这一点：

```
// Let's create a vector to store our commands that we want to run sequentially.
std::vector<std::unique_ptr<Command>> commands;

// Add an instant command that prints to the console.
commands.emplace_back(std::make_unique<InstantCommand>([]{ std::cout << "Hello"; },␣
↪requirements));

// Add some other command: this can be something that a user has created.
commands.emplace_back(std::make_unique<MyCommand>(args, needed, for, this, command));

// Now the vector "owns" all of these commands. In its current state, when the vector␣
↪is destroyed (i.e.
// it goes out of scope), it will destroy all of the commands we just added.

// Let's create a SequentialCommandGroup that will run these two commands␣
↪sequentially.
auto group = SequentialCommandGroup(std::move(commands));

// Note that we MOVED the vector of commands into the sequential command group,␣
↪meaning that the
// command group now has ownership of our commands. When we call std::move on the␣
↪vector, all of its
// contents (i.e. the unique_ptr instances) are moved into the command group.

// Even if the vector were to be destroyed while the command group was running,␣
↪everything would be OK
// since the vector does not own our commands anymore.
```

使用"std::shared_ptr"时，没有明确的所有权模型，因为"std::shared_ptr"可能有多个实例指向同一块内存。如果指令在"std::shared_ptr"实例中，则指令组或指令调度程序将无法获得所有权并在指令完成执行后释放内存，因为用户可能仍然不知不觉中仍然拥有"std::shared_ptr"。指向范围内某处内存块的实例。

26.9.2 CRTP 的使用

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:

Command Decorators

新的基于指令的框架包括称为“command decorators”的功能，该功能允许用户执行以下操作：

```
auto task = MyCommand().AndThen([] { std::cout << "This printed after my command␣
↪ended."; },
  requirements);
```

计划好“task”之后，它将首先执行“MyCommand()”，一旦该命令执行完毕，它将把消息打印到控制台。在内部实现此目标的方法是使用顺序指令组。

回顾上一节，为了构建顺序指令组，我们需要一个指向每个指令的唯一指针的向量。为打印功能创建唯一的指针非常简单：

```
temp.emplace_back(
   std::make_unique<InstantCommand>(std::move(toRun), requirements));
```

这里的“temp”存储我们需要传递给“SequentialCommandGroup”构造函数的指令向量。但是在添加“InstantCommand”之前，我们需要将“MyCommand()”添加到“SequentialCommandGroup”中。我们该怎么做？

```
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.

解决问题的方法

我们对此的最初解决方案是在“Command”中创建一个虚拟方法，称为“TransferOwnership()”，“Command”的每个子类都必须重写。这样的覆盖看起来像这样：

```
std::unique_ptr<Command> TransferOwnership() && override {
  return std::make_unique<MyCommand>(std::move(*this));
}
```

因为代码将在派生的子类中，所以“*this”实际上将指向所需的子类实例，并且用户具有派生类的类型信息以构成唯一的指针。

经过几天的审议，提出了一种 CRTP 方法。在这里，将存在一个称为“CommandCommander”的中间派生类“Command”。“CommandHelper”有两个模板参数，原始基类和所需的派生子类。让我们看一下“CommandHelper”的基本实现以了解这一点：

```
// 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>
```

(续下页)

(接上页)

```
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)));
  }
};
```

因此，使您的自定义命令扩展为“CommandHelper”而不是“Command”将自动为您实现此样板，这就是要求团队使用似乎是一种相当晦涩的处理方式的原因。

回到我们的“AndThen()”示例，我们现在可以执行以下操作：

```
// 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 缺乏高级装饰

大多数 C ++ 装饰器都采用“std::function <void()>”代替实际的命令本身。考虑了在装饰器中接收实际命令的想法，例如“AndThen()”，“BeforeStarting()”等，但由于多种原因而被放弃。

模板装饰器

因为我们需要在编译时知道要添加到指令组的指令的类型，所以我们将需要使用模板（对于多个指令来说是可变的）。但是，这似乎没什么大不了的。无论如何，指令组的构造函数都会这样做：

```
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));
}
```

备注： 除了我们上面描述的向量构造器之外，这是“SequentialCommandGroup”的辅助构造器。

但是，当我们制作模板函数时，必须将其定义内联声明。这意味着我们将需要在“Command.h”标头中实例化“SequentialCommandGroup”，这会带来问题。“SequentialCommandGroup.h”包括“Command.h”。如果我们在“Command.h”内包含“SequentialCommandGroup.h”，则具有循环依赖关系。那我们现在怎么办呢?

我们在“Command.h”的顶部使用前向声明:

```
class SequentialCommandGroup;

class Command { ... };
```

然后我们在“Command.cpp”中包含“SequentialCommandGroup.h”。但是，如果这些装饰器函数是模板化的，则无法在“.cpp”文件中编写定义，从而导致循环依赖。

Java 与 C ++ 语法

这些装饰器通常在 Java 中保存比在 C ++ 中更多的详细信息（因为 Java 需要原始的“new”调用），因此通常，如果您在用户代码中手动创建指令组，则在 C ++ 中并不会产生太大的区别。

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] {
```

(续下页)

(接上页)

```
            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 通过 PID 子系统和 PID 指令进行 PID 控制

备注： 有关这些基于指令的程序包使用的 WPILib PID 控制功能的描述，请参阅 WPILib 中的:ref:*docs/software/advanced-controls/controllers/pidcontroller:PID Control in 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 PID 子系统

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.

创建一个 PID 子系统

备注： If `periodic` is overridden when inheriting from `PIDSubsystem`, make sure to call `super.periodic()`! Otherwise, PID functionality will not work properly.

创建“PIDSubsystem”的子类时，用户必须覆盖其父类的以下两个抽象方法以提供该类将在其常规运算中使用的功能：

getMeasurement()

Java

```
protected abstract double getMeasurement();
```

C++

```
virtual double GetMeasurement() = 0;
```

“getMeasurement”方法返回过程变量的当前测量值。PIDSubsystem 将自动从其“periodic()”块中调用此方法，并将它的值传递给控制循环。

用户应覆盖此方法，以返回他们希望的用作过程变量测量的任何传感器读数。

useOutput()

Java

```
protected abstract void useOutput(double output, double setpoint);
```

C++

```
virtual void UseOutput(double output, double setpoint) = 0;
```

useOutput() 方法使用 PID 控制器的输出和当前设定值（这通常对于计算前馈很有用）。“PIDSubsystem”将自动从其“periodic()”块中调用此方法，并将控制环的计算输出传递给该方法。

用户应覆盖此方法，以将最终计算出的控制输出传递给其子系统的电机。

控制器中的传递

用户还必须通过“PIDController”和“PIDSubsystem”的子类的超类构造函数调用将“PIDController”传递给“PIDSubsystem”基类。这一步骤用于指定 PID 增益以及周期（如果用户使用的是非标准主机器人循环周期）。

用户还可以通过调用“getController()”在构造函数主体中对控制器进行其他修改（例如启用连续输入）。

使用 PID 子系统

一旦创建了“PIDSubsystem”子类的实例，就可以在指令中通过以下方法使用它：

setSetpoint()

“setSetpoint()”方法可用于设置“PIDSubsystem”的设定值。子系统将使用已经定义的输出自动跟踪到设定点：

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() 和 disable()

“enable()”和“disable()”方法能够启用和禁用“PIDSubsystem”的 PID 控制。启用子系统后，它将自动运行控制回路并跟踪设定值。禁用时，子系统不执行任何控制。

此外，“enable()”方法能够重置内部的“PIDController”，并且“disable()”方法能够调用用户定义的‘useOutput()’_方法，并将输出值和设定值都设置为“0”。

完整的 PID 子系统示例

在实践中使用时，“PIDSubsystem”是什么样的？以下示例取自 FrisbeeBot 示例项目（Java <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/frisbeebot>‘__，C ++）：

Java

```
5  package edu.wpi.first.wpilibj.examples.frisbeebot.subsystems;
6
7  import edu.wpi.first.math.controller.PIDController;
8  import edu.wpi.first.math.controller.SimpleMotorFeedforward;
9  import edu.wpi.first.wpilibj.Encoder;
10 import edu.wpi.first.wpilibj.examples.frisbeebot.Constants.ShooterConstants;
11 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
12 import edu.wpi.first.wpilibj2.command.PIDSubsystem;
13
14 public class ShooterSubsystem extends PIDSubsystem {
15   private final PWMSparkMax m_shooterMotor = new PWMSparkMax(ShooterConstants.
     ↪kShooterMotorPort);
16   private final PWMSparkMax m_feederMotor = new PWMSparkMax(ShooterConstants.
     ↪kFeederMotorPort);
17   private final Encoder m_shooterEncoder =
18       new Encoder(
19           ShooterConstants.kEncoderPorts[0],
```

(续下页)

(接上页)

```
          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);
  }

  @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;

```

(续下页)

(接上页)

```
  double GetMeasurement() override;

  bool AtSetpoint();

  void RunFeeder();

  void StopFeeder();

 private:
  frc::PWMSparkMax m_shooterMotor;
  frc::PWMSparkMax m_feederMotor;
  frc::Encoder m_shooterEncoder;
  frc::SimpleMotorFeedforward<units::turns> m_shooterFeedforward;
};
```

C++ (Source)

```
#include "subsystems/ShooterSubsystem.h"

#include <frc/controller/PIDController.h>

#include "Constants.h"

using namespace ShooterConstants;

ShooterSubsystem::ShooterSubsystem()
    : PIDSubsystem{frc::PIDController{kP, kI, kD}},
      m_shooterMotor(kShooterMotorPort),
      m_feederMotor(kFeederMotorPort),
      m_shooterEncoder(kEncoderPorts[0], kEncoderPorts[1]),
      m_shooterFeedforward(kS, kV) {
  m_controller.SetTolerance(kShooterToleranceRPS.value());
  m_shooterEncoder.SetDistancePerPulse(kEncoderDistancePerPulse);
  SetSetpoint(kShooterTargetRPS.value());
}

void ShooterSubsystem::UseOutput(double output, double setpoint) {
  m_shooterMotor.SetVoltage(units::volt_t{output} +
                            m_shooterFeedforward.Calculate(kShooterTargetRPS));
}

bool ShooterSubsystem::AtSetpoint() {
  return m_controller.AtSetpoint();
}

double ShooterSubsystem::GetMeasurement() {
  return m_shooterEncoder.GetRate();
}

void ShooterSubsystem::RunFeeder() {
  m_feederMotor.Set(kFeederSpeed);
}

void ShooterSubsystem::StopFeeder() {
```

(续下页)

(接上页)

```
  m_feederMotor.Set(0);
}
```

通过命令使用“PIDSubsystem”非常简单：

Java

```
  private final Command m_spinUpShooter = Commands.runOnce(m_shooter::enable, m_
↪shooter);
  private final Command m_stopShooter = Commands.runOnce(m_shooter::disable, m_
↪shooter);

    // We can bind commands while retaining references to them in RobotContainer

    // Spin up the shooter when the 'A' button is pressed
    m_driverController.a().onTrue(m_spinUpShooter);

    // Turn off the shooter when the 'B' button is pressed
    m_driverController.b().onTrue(m_stopShooter);
```

C++

```
  frc2::CommandPtr m_spinUpShooter =
      frc2::cmd::RunOnce([this] { m_shooter.Enable(); }, {&m_shooter});

  frc2::CommandPtr m_stopShooter =
      frc2::cmd::RunOnce([this] { m_shooter.Disable(); }, {&m_shooter});
```

C++ (Source)

```
  // We can bind commands while keeping their ownership in RobotContainer

  // Spin up the shooter when the 'A' button is pressed
  m_driverController.A().OnTrue(m_spinUpShooter.get());

  // Turn off the shooter when the 'B' button is pressed
  m_driverController.B().OnTrue(m_stopShooter.get());
```

26.10.2 PID 指令

The `PIDCommand` class allows users to easily create commands with a built-in PIDController.

创建一个 PID 指令

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.

备注: If subclassing `PIDCommand` and overriding any methods, make sure to call the `super` version of those methods! Otherwise, PID functionality will not work properly.

无论哪种情况，都会通过将必要的参数传递给“PIDCommand”的构造函数来创建“PIDCommand”（如果定义了子类，则可以通过“super()”调用来完成）：

Java

```
27  /**
28   * Creates a new PIDCommand, which controls the given output with a PIDController.
29   *
30   * @param controller the controller that controls the output.
31   * @param measurementSource the measurement of the process variable
32   * @param setpointSource the controller's setpoint
33   * @param useOutput the controller's output
34   * @param requirements the subsystems required by this command
35   */
36  public PIDCommand(
37      PIDController controller,
38      DoubleSupplier measurementSource,
39      DoubleSupplier setpointSource,
40      DoubleConsumer useOutput,
41      Subsystem... requirements) {
```

C++

```
28  /**
29   * Creates a new PIDCommand, which controls the given output with a
30   * PIDController.
31   *
32   * @param controller        the controller that controls the output.
33   * @param measurementSource the measurement of the process variable
34   * @param setpointSource   the controller's reference (aka setpoint)
35   * @param useOutput         the controller's output
36   * @param requirements      the subsystems required by this command
37   */
38  PIDCommand(frc::PIDController controller,
39             std::function<double()> measurementSource,
40             std::function<double()> setpointSource,
41             std::function<void(double)> useOutput,
42             Requirements requirements = {});
```

controller

“controller”参数是在指令中将使用的“PIDController”对象。通过传递此参数，用户可以详细指定 PID 增益和控制器的周期（如果用户使用的是非标准主机器人循环周期）。

当子类化“PIDCommand”时，可以通过调用 getController() 对构造函数主体中的控制器进行额外的修改（例如启用连续输入）。

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`.

当子类化“PIDCommand”时，高级用户可以通过修改类的“m_measurement”字段来进一步修改度量值提供器。

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.

当子类化“PIDCommand”时，高级用户可以通过修改类的“m_setpoint”字段来进一步修改设定值提供器。

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`.

当子类化“PIDCommand”时，高级用户可以通过修改类的“m_useOutput”字段来进一步修改输出值使用器。

要求

像所有内联命令一样，“PIDCommand”允许用户将其子系统要求指定为构造函数参数。

完整的 PID 指令示例

在实际使用中，“PIDCommand”是什么样的？以下示例来自 GyroDriveCommands 示例项目（Java，C ++）：

Java

```java
package edu.wpi.first.wpilibj.examples.gyrodrivecommands.commands;

import edu.wpi.first.math.controller.PIDController;
import edu.wpi.first.wpilibj.examples.gyrodrivecommands.Constants.DriveConstants;
import edu.wpi.first.wpilibj.examples.gyrodrivecommands.subsystems.DriveSubsystem;
import edu.wpi.first.wpilibj2.command.PIDCommand;

/** A command that will turn the robot to the specified angle. */
public class TurnToAngle extends PIDCommand {
  /**
   * 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
   */
  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();
```

(续下页)

(接上页)

```
}
```

And, for an *inlined* example:

Java

```
    // Stabilize robot to drive straight with gyro when left bumper is held
    new JoystickButton(m_driverController, Button.kL1.value)
        .whileTrue(
            new PIDCommand(
                new PIDController(
                    DriveConstants.kStabilizationP,
                    DriveConstants.kStabilizationI,
                    DriveConstants.kStabilizationD),
                // Close the loop on the turn rate
                m_robotDrive::getTurnRate,
                // Setpoint is 0
                0,
                // Pipe the output to the turning controls
                output -> m_robotDrive.arcadeDrive(-m_driverController.getLeftY(),␣
↪output),
                // Require the robot drive
                m_robotDrive));
```

C++

```
  // Stabilize robot to drive straight with gyro when L1 is held
  frc2::JoystickButton(&m_driverController, frc::PS4Controller::Button::kL1)
      .WhileTrue(
          frc2::PIDCommand(
              frc::PIDController{dc::kStabilizationP, dc::kStabilizationI,
                                 dc::kStabilizationD},
              // Close the loop on the turn rate
              [this] { return m_drive.GetTurnRate(); },
              // Setpoint is 0
              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 通过 TrapezoidProfileSubsystems 和 TrapezoidProfileCommands 进行运动分析

备注: 有关这些基于指令的包装程序使用的 WPILib 运动分析功能的描述，请参阅:ref:*docs/software/advanced-controls/controllers/trapezoidal-profiles:Trapezoidal Motion Profiles in WPILib*。

备注: “TrapezoidProfile” 指令包装通常用于自定义或外部控制器的组合。要将梯形运动轮廓与 WPILib 的 “PIDController” 结合使用，请参阅:doc:*profilepid-subsystems-commands*。

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++).

为了进一步帮助团队将运动分析集成到基于指令的机器人项目中，WPILib 为 “TrapezoidProfile” 类提供了两个便捷包装器：“TrapezoidProfileSubsystem”，该包装器自动在其 “periodic()” 中生成并执行运动配置文件。方法和 “TrapezoidProfileCommand”，后者执行一个用户提供的 “TrapezoidProfile”。

26.11.1 TrapezoidProfileSubsystem

备注: 在 C ++ 中，“TrapezoidProfileSubsystem” 类在用于距离测量的单位类型（可能是角度的或线性的）上模板化。传入的值 * 必须 * 具有与距离单位一致的单位，否则将引发编译时错误。有关 C ++ 单元的更多信息，请参阅:ref:*docs/software/basic-programming/cpp-units:The C++ Units Library*。

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.

创建一个 TrapezoidProfileSubsystem

备注: If `periodic` is overridden when inheriting from `TrapezoidProfileSubsystem`, make sure to call `super.periodic()`! Otherwise, motion profiling functionality will not work properly.

当子类化 “TrapezoidProfileSubsystem” 时，用户必须重写单个抽象方法，以提供该类将在其常规操作中使用的功能：

useState()

Java

```
protected abstract void useState(TrapezoidProfile.State state);
```

C++

```
virtual void UseState(State state) = 0;
```

“useState()”方法消耗运动配置文件的当前状态。“TrapezoidProfileSubsystem”将自动从其“periodic()”块中调用此方法，并将与子系统当前进度相对应的运动配置文件状态通过运动配置文件传递给该方法。

用户可以在这种状态下做任何想做的事情。典型的用例（如‘Full TrapezoidProfileSubsystem Example‘_ 中所示）是使用状态获取外部“智能”电动机控制器的设定值和前馈。

构造器参数

用户必须通过子类的超类构造函数调用将一组“TrapezoidProfile.Constraints”传递给“TrapezoidProfileSubsystem”基类。这用于将自动生成的轮廓约束到给定的最大速度和加速度。

用户还必须通过机制的初始位置。

如果正在使用非标准的主循环周期，则高级用户可以在循环周期中输入备用值。

使用 TrapezoidProfileSubsystem

一旦创建了“TrapezoidProfileSubsystem”子类的实例，命令就可以通过以下方法使用它：

setGoal()

备注： 如果希望将目标设置为隐式目标速度为零的简单距离，则存在“setGoal()”重载，该重载采用单个距离值而不是完整的运动轮廓状态。

“setGoal()”方法可用于设置“TrapezoidProfileSubsystem”的目标状态。子系统将自动执行针对目标的配置文件，将每次迭代时的当前状态传递给提供的‘useState()‘_ 方法。

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.

完整的 TrapezoidProfileSubsystem 示例

在实践中使用时，“TrapezoidProfileSubsystem” 是什么样的？以下示例摘自 ArmbotOffobard 示例项目（Java，C ++）：

Java

```
5  package edu.wpi.first.wpilibj.examples.armbotoffboard.subsystems;
6
7  import edu.wpi.first.math.controller.ArmFeedforward;
8  import edu.wpi.first.math.trajectory.TrapezoidProfile;
9  import edu.wpi.first.wpilibj.examples.armbotoffboard.Constants.ArmConstants;
10 import edu.wpi.first.wpilibj.examples.armbotoffboard.ExampleSmartMotorController;
11 import edu.wpi.first.wpilibj2.command.Command;
12 import edu.wpi.first.wpilibj2.command.Commands;
13 import edu.wpi.first.wpilibj2.command.TrapezoidProfileSubsystem;
14
15 /** A robot arm subsystem that moves with a motion profile. */
16 public class ArmSubsystem extends TrapezoidProfileSubsystem {
17   private final ExampleSmartMotorController m_motor =
18       new ExampleSmartMotorController(ArmConstants.kMotorPort);
19   private final ArmFeedforward m_feedforward =
20       new ArmFeedforward(
21           ArmConstants.kSVolts, ArmConstants.kGVolts,
22           ArmConstants.kVVoltSecondPerRad, ArmConstants.kAVoltSecondSquaredPerRad);
23
24   /** Create a new ArmSubsystem. */
25   public ArmSubsystem() {
26     super(
27         new TrapezoidProfile.Constraints(
28             ArmConstants.kMaxVelocityRadPerSecond, ArmConstants.
↪kMaxAccelerationRadPerSecSquared),
```

(续下页)

(接上页)

```
        ArmConstants.kArmOffsetRads);
    m_motor.setPID(ArmConstants.kP, 0, 0);
  }

  @Override
  public void useState(TrapezoidProfile.State setpoint) {
    // Calculate the feedforward from the sepoint
    double feedforward = m_feedforward.calculate(setpoint.position, setpoint.
↪velocity);
    // Add the feedforward to the PID output to get the motor output
    m_motor.setSetpoint(
        ExampleSmartMotorController.PIDMode.kPosition, setpoint.position, feedforward␣
↪/ 12.0);
  }

  public Command setArmGoalCommand(double kArmOffsetRads) {
    return Commands.runOnce(() -> setGoal(kArmOffsetRads), this);
  }
}
```

C++ (Header)

```
#pragma once

#include <frc/controller/ArmFeedforward.h>
#include <frc2/command/Commands.h>
#include <frc2/command/TrapezoidProfileSubsystem.h>
#include <units/angle.h>

#include "ExampleSmartMotorController.h"

/**
 * A robot arm subsystem that moves with a motion profile.
 */
class ArmSubsystem : public frc2::TrapezoidProfileSubsystem<units::radians> {
  using State = frc::TrapezoidProfile<units::radians>::State;

 public:
  ArmSubsystem();

  void UseState(State setpoint) override;

  frc2::CommandPtr SetArmGoalCommand(units::radian_t goal);

 private:
  ExampleSmartMotorController m_motor;
  frc::ArmFeedforward m_feedforward;
};
```

C++ (Source)

```cpp
#include "subsystems/ArmSubsystem.h"

#include "Constants.h"

using namespace ArmConstants;
using State = frc::TrapezoidProfile<units::radians>::State;

ArmSubsystem::ArmSubsystem()
    : frc2::TrapezoidProfileSubsystem<units::radians>(
          {kMaxVelocity, kMaxAcceleration}, kArmOffset),
      m_motor(kMotorPort),
      m_feedforward(kS, kG, kV, kA) {
  m_motor.SetPID(kP, 0, 0);
}

void ArmSubsystem::UseState(State setpoint) {
  // Calculate the feedforward from the sepoint
  units::volt_t feedforward =
      m_feedforward.Calculate(setpoint.position, setpoint.velocity);
  // Add the feedforward to the PID output to get the motor output
  m_motor.SetSetpoint(ExampleSmartMotorController::PIDMode::kPosition,
                      setpoint.position.value(), feedforward / 12_V);
}

frc2::CommandPtr ArmSubsystem::SetArmGoalCommand(units::radian_t goal) {
  return frc2::cmd::RunOnce([this, goal] { this->SetGoal(goal); }, {this});
}
```

通过命令使用“TrapezoidProfileSubsystem”可能非常简单：

Java

```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++

```cpp
  // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
  m_driverController.A().OnTrue(m_arm.SetArmGoalCommand(2_rad));

  // Move the arm to neutral position when the 'B' button is pressed.
  m_driverController.B().OnTrue(
      m_arm.SetArmGoalCommand(ArmConstants::kArmOffset));
```

26.11.2 TrapezoidProfileCommand

备注： 在 C ++ 中，“TrapezoidProfileCommand”类在用于距离测量的单位类型（可能是角度的或线性的）上模板化。传入的值 * 必须 * 具有与距离单位一致的单位，否则将引发编译时错误。有关 C ++ 单元的更多信息，请参阅:ref:*docs/software/basic-programming/cpp-units:The C++ Units Library*。

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.

创建一个 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.

备注： 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.

无论哪种情况，都通过将必要的参数传递给其构造函数来创建“TrapezoidProfileCommand”（如果定义了子类，则可以通过“super()”调用来完成）：

Java

```
28   /**
29    * Creates a new TrapezoidProfileCommand that will execute the given {@link␣
↪TrapezoidProfile}.
30    * Output will be piped to the provided consumer function.
31    *
32    * @param profile The motion profile to execute.
33    * @param output The consumer for the profile output.
34    * @param goal The supplier for the desired state
35    * @param currentState The current state
36    * @param requirements The subsystems required by this command.
37    */
38   @SuppressWarnings("this-escape")
39   public TrapezoidProfileCommand(
40       TrapezoidProfile profile,
41       Consumer<State> output,
42       Supplier<State> goal,
43       Supplier<State> currentState,
```

C++

```
/**
 * Creates a new TrapezoidProfileCommand that will execute the given
 * TrapezoidalProfile. Output will be piped to the provided consumer function.
 *
 * @param profile      The motion profile to execute.
 * @param output       The consumer for the profile output.
 * @param goal The supplier for the desired state
 * @param currentState The current state
 * @param requirements The list of requirements.
 */
TrapezoidProfileCommand(frc::TrapezoidProfile<Distance> profile,
                        std::function<void(State)> output,
                        std::function<State()> goal,
                        std::function<State()> currentState,
                        Requirements requirements = {})
```

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.

requirements

像所有内联命令一样，“TrapezoidProfileCommand”允许用户将其子系统要求指定为构造函数参数。

完整的 TrapezoidProfileCommand 示例

在实践中使用时，“TrapezoidProfileSubsystem”是什么样的？以下示例取自 DriveDistanceOffboard 示例项目（Java <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/drivedistanceoffboard>‘__，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 {
  /**
   * 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{
          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 在基于指令中将运动分析和 PID 结合

备注: 有关这些基于指令的包装程序使用的 WPILib PID 控制功能的描述，请参阅:ref:*docs/software/advanced-controls/controllers/pidcontroller:PID Control in WPILib*。

常见的 FRC | reg |控制解决方案是将用于设定值生成的梯形运动曲线与用于设定值跟踪的 PID 控制器配对。为此，WPILib 包含了自己的:ref:*ProfiledPIDController <docs/software/advanced-controls/controllers/profiled-pidcontroller:Combining Motion Profiling and PID Control with ProfiledPIDController>* 类。为了进一步帮助团队将该功能集成到他们的机器人中，基于指令的框架为“ProfiledPIDController”类提供了两个便捷包装：“ProfiledPIDSubsystem”将控制器集成到子系统中以及“ProfiledPIDCommand”，它将控制器集成到指令中。

26.12.1 ProfiledPIDSubsystem

备注: 在 C ++ 中，“ProfiledPIDSubsystem”类在用于距离测量的单位类型（可能是角度的或线性的）上模板化。传入的值 * 必须 * 具有与距离单位一致的单位，否则将引发编译时错误。有关 C ++ 单元的更多信息，请参阅:ref:*docs/software/basic-programming/cpp-units:The C++ Units Library*。

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.

创建一个 ProfiledPIDSubsystem

备注: If `periodic` is overridden when inheriting from `ProfiledPIDSubsystem`, make sure to call `super.periodic()`! Otherwise, control functionality will not work properly.

子类化”ProfiledPIDSubsystem”时，用户必须重写两个抽象方法以提供该类将在其常规操作中使用的功能：

getMeasurement()

Java

```
protected abstract double getMeasurement();
```

C++

```
virtual Distance_t GetMeasurement() = 0;
```

“getMeasurement” 方法返回过程变量的当前测量值。“PIDSubsystem” 将自动从其 “periodic()” 块中调用此方法，并将其值传递给控制循环。

用户应重写此方法，以返回他们希望用作过程变量测量的任何传感器读数。

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.

用户应重写此方法，以将最终计算出的控制输出传递给其子系统的电机。

传递至控制器

用户还必须通过子类的超类构造函数调用将 “ProfiledPIDController” 传递给 “ProfiledPIDSubsystem” 基类。这用于指定 PID 增益，运动曲线约束和周期（如果用户使用的是非标准主机器人循环周期）。

可以通过调用 “getController()” 在构造函数主体中对控制器进行其他修改（例如启用连续输入）。

使用 ProfiledPIDSubsystem

一旦创建了 “PIDSubsystem” 子类的实例，命令就可以通过以下方法使用它：

setGoal()

备注： 如果希望将目标设置为隐式目标速度为零的简单距离，则存在 “setGoal()” 重载，该重载采用单个距离值而不是完整的运动轮廓状态。

“setGoal()” 方法可用于设置 “PIDSubsystem” 的设定值。子系统将使用定义的输出自动跟踪到设定点：

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() 和 disable()

“enable()”和“disable()”方法启用和禁用“ProfiledPIDSubsystem’”的自动控制。启用子系统后，它将自动运行运动曲线和控制循环并跟踪目标。禁用时，不执行任何控制。

另外，“enable()”方法重置内部 ProfileDPIDController，“disable()”方法调用用户定义的‘useOutput()‘_ 方法，将 output 和 setpoint 都设置为“0”。

完整 ProfiledPIDSubsystem 示例

在实际使用中，PIDSubsystem 是什么样的？以下示例摘自 ArmBot 示例项目（Java <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/armbot>‘__，C ++）：

Java

```
package edu.wpi.first.wpilibj.examples.armbot.subsystems;

import edu.wpi.first.math.controller.ArmFeedforward;
import edu.wpi.first.math.controller.ProfiledPIDController;
import edu.wpi.first.math.trajectory.TrapezoidProfile;
import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.examples.armbot.Constants.ArmConstants;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
import edu.wpi.first.wpilibj2.command.ProfiledPIDSubsystem;

/** A robot arm subsystem that moves with a motion profile. */
public class ArmSubsystem extends ProfiledPIDSubsystem {
  private final PWMSparkMax m_motor = new PWMSparkMax(ArmConstants.kMotorPort);
  private final Encoder m_encoder =
      new Encoder(ArmConstants.kEncoderPorts[0], ArmConstants.kEncoderPorts[1]);
  private final ArmFeedforward m_feedforward =
      new ArmFeedforward(
          ArmConstants.kSVolts, ArmConstants.kGVolts,
          ArmConstants.kVVoltSecondPerRad, ArmConstants.kAVoltSecondSquaredPerRad);

  /** Create a new ArmSubsystem. */
  public ArmSubsystem() {
    super(
        new ProfiledPIDController(
```

(续下页)

(接上页)

```
            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:
  frc::PWMSparkMax m_motor;
  frc::Encoder m_encoder;
  frc::ArmFeedforward m_feedforward;
};
```

C++ (Source)

```cpp
#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;
}
```

通过命令使用“ProfiledPIDSubsystem”可以非常简单：

Java

```java
    // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
    m_driverController
        .a()
        .onTrue(
            Commands.runOnce(
                () -> {
                  m_robotArm.setGoal(2);
                  m_robotArm.enable();
                },
                m_robotArm));
```

C++

```
  // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
  m_driverController.A().OnTrue(frc2::cmd::RunOnce(
      [this] {
        m_arm.SetGoal(2_rad);
        m_arm.Enable();
      },
      {&m_arm}));
```

26.12.2 ProfiledPIDCommand

备注： 在 C ++ 中，“ProfiledPIDCommand”类以用于距离测量的单位类型为模板，该单位类型可以是角度或线性的。传入的值 * 必须 * 具有与距离单位一致的单位，否则将引发编译时错误。有关 C ++ 单元的更多信息，请参阅:ref:*docs/software/basic-programming/cpp-units:The C++ Units Library*。

The `ProfiledPIDCommand` class (Java, C++) allows users to easily create commands with a built-in ProfiledPIDController.

创建一个 PIDCommand

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.

备注： If subclassing `ProfiledPIDCommand` and overriding any methods, make sure to call the `super` version of those methods! Otherwise, control functionality will not work properly.

无论哪种情况，都通过向构造函数传递必要的参数来创建“ProfileDPIDCommand”（如果定义了子类，则可以通过调用“super()”来完成）：

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(
      ProfiledPIDController controller,
      DoubleSupplier measurementSource,
```

(续下页)

(接上页)

```
    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 = {})
```

controller

“controller”参数是在指令中将使用的“ProfiledPIDController”对象。通过传递此参数，用户可以指定 PID 增益，运动曲线约束和控制器的周期（如果用户使用的是非标准的主机器人循环周期）。

当将“ProfiledPIDCommand”子类化时，可以通过调用“getController()”对构造函数主体中的控制器进行其他修改（例如，启用连续输入）。

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`.

在子类化“ProfiledPIDCommand”时，高级用户可以通过修改类的“m_measurement”字段来进一步修改度量提供者。

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.

当子类化“ProfiledPIDCommand”时，高级用户可以通过修改类的“m_goal”字段来进一步修改设定值供应商。

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`.

在子类化“ProfiledPIDCommand”时，高级用户可以通过修改类的“m_useOutput”字段来进一步修改输出使用者。

requirements

像所有内联命令一样，“ProfiledPIDCommand”允许用户将其子系统要求指定为构造函数参数。

完整 ProfiledPIDCommand 示例

在实际使用中，“ProfiledPIDCommand”是什么样的？以下示例来自 GyroDriveCommands 示例项目 (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) {
22     super(
23         new ProfiledPIDController(
```

(续下页)

(接上页)

```
            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.
   *
   * @param targetAngleDegrees The angle to turn to
   * @param drive              The drive subsystem to use
```

(续下页)

(接上页)

```
   */
  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++)

> **警告:** 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

运动学和测距法

27.1 Introduction to Kinematics and The ChassisSpeeds Class

备注： Kinematics and odometry uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.1.1 什么是运动学？

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 什么是测距法？

测距法涉及使用机器人上的传感器来估计机器人在场上的位置。在 FRC 中，这些传感器通常是几个编码器（具体数量取决于驱动类型）和一个陀螺仪来测量机器人的角度。运动学类利用运动学类和用户定期输入的速度（和转弯时的角度）来估计机器人在场上的位置。

27.1.3 The ChassisSpeeds Class

“ChassisSpeeds“对象对新的 WPILib 运动学和里程测量套件至关重要。“ChassisSpeeds“对象表示机器人底盘的速度。该结构有三个组成部分：

- `vx`：机器人在 x(前进) 方向的速度。
- `vy`：机器人在 y（横向）方向的速度。(正值表示机器人向左移动)。
- `omega`：机器人的角速度，单位为每秒弧度。

备注： 非整体性传动系统 (即不能横向移动的传动系统，例如：差速驱动) 由于不能横向移动，其 “vy “分量为零。

27.1.4 构造一个 ChassisSpeeds 对象

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 从相对场地速度创建 ChassisSpeeds 对象

当机器人角度给定时，还可以用一组场内相对速度创建一个“底盘速度“对象。这将一组相对于场域的期望速度（例如，朝向对面联盟站和朝向右场边界）转换为“ChassisSpeeds“对象，它表示相对于机器人框架的速度。这对于实现面向场地的控制，对转向或麦轮驱动机器人是有用的。

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))
```

备注： 角速度没有明确说是“相对于场”，因为从场角度或机器人角度测量的角速度是一样的。

27.2 差动驱动运动学

“DifferentialDriveKinematics“类是一个有用的工具，它可以在 “ChassisSpeeds“对象和 “DifferentialDriveWheelSpeeds“对象之间进行转换，其中包含差动驱动机器人左右两侧的速度。

27.2.1 构造运动学对象

“DifferentialDriveKinematics” 对象接受一个构造函数参数，即机器人的轨道宽度。这代表了差速器上两组轮子之间的距离。

备注： 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 将底盘速度转换为车轮速度

The `toWheelSpeeds(ChassisSpeeds speeds)` (Java / Python) / `ToWheelSpeeds(ChassisSpeeds speeds)` (C++) method should be used to convert a `ChassisSpeeds` object to a `DifferentialDriveWheelSpeeds` object. This is useful in situations where you have to convert a linear velocity (`vx`) and an angular velocity (`omega`) to left and right wheel velocities.

JAVA

```
// Creating my kinematics object: track width of 27 inches
DifferentialDriveKinematics kinematics =
  new DifferentialDriveKinematics(Units.inchesToMeters(27.0));

// Example chassis speeds: 2 meters per second linear velocity,
// 1 radian per second angular velocity.
var chassisSpeeds = new ChassisSpeeds(2.0, 0, 1.0);

// Convert to wheel speeds
DifferentialDriveWheelSpeeds wheelSpeeds = kinematics.toWheelSpeeds(chassisSpeeds);

// Left velocity
double leftVelocity = wheelSpeeds.leftMetersPerSecond;

// Right velocity
double rightVelocity = wheelSpeeds.rightMetersPerSecond;
```

C++

```
// 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 将车轮速度转换为底盘速度

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

```
// 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++

```
// 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

```
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
```

(续下页)

(接上页)

```
# 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

# Angular velocity
angularVelocity = chassisSpeeds.omega
```

27.3 差速驱动测距法

用户可以使用差速驱动运动学类来执行:ref:*odometry <docs/software/kinematics-and-odometry/intro-and-chassis-speeds:What is odmetry?>*。WPILib 中包含了一个“差动驱动里程测量“类，可以用来跟踪差动驱动机器人在现场的位置。

备注： 由于这种方法只使用编码器和陀螺仪，机器人在场上的位置估计会随着时间的推移而漂移，特别是当你的机器人在游戏过程中与其他机器人接触时。不过，在自主期，测距法通常非常准确。

27.3.1 创建 Odometry 对象

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`.

备注： 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 更新机器人姿势

“update“方法可用于更新机器人在场地上的位置。该方法必须定期调用，最好是在:ref:‘Subsystem <docs/software/commandbased/subsystems:Subsystems>‘的 “periodic()“方法中调用。“update“方法返回机器人新的更新姿势。该方法获取机器人的陀螺仪角度，以及左编码器距离和右编码器距离。

备注: 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 重置机器人姿势

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.

重要： 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.

备注： 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 转向驱动运动学

“SwerveDriveKinematics’’ 类是一个有用的工具，可在 “ChassisSpeeds’’ 对象和几个 “SwerveModuleState’’ 对象之间进行转换，该对象包转向驱动机器人的每个速度模块的速度和角度。

备注： Swerve drive kinematics uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.4.1 转向模块状态类

“SwerveModuleState’’ 类包含有关转向驱动单个模块的速度和角度的信息。“SwerveModuleState’’ 的构造函数接受两个参数，即车轮在模块上的速度和模块的角度。

备注： 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.

备注： 角度 0 对应于面向前方的模块。

27.4.2 构造运动学对象

“SwerveDriveKinematics’’ 类接受可变数量的构造函数参数，每个参数都是相对于机器人中心的转向模块的位置（作为 “Translation2d’’。构造函数参数的数量与转向的数量相对应模块。

备注： 转向驱动必须具有 2 个或更多模块。

备注： 在 C ++ 中，该类以模块数量为模板。因此，在将 “SwerveDriveKinematics” 对象构造为类的成员变量时，必须将多个模块作为模板参数传入。例如，对于具有四个模块的典型转向驱动，运动学对象必须按以下方式构造：“frc :: SwerveDriveKinematics <4> m_kinematics {…}’’。

模块的位置必须相对于机器人的中心。正的 x 值表示朝着机器人的前端移动，而正的 y 值表示朝着机器人的左侧移动。

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
);
```

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 将底盘速度转换为模块状态

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.

该方法返回的数组中的元素与运动学对象的构造顺序相同。例如，如果运动学对象的构造顺序是前左模块位置、前右模块位置、后左模块位置和后右模块位置，那么数组中的元素将依次是前左模块状态、前右模块状态、后左模块状态和后右模块状态。

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)
```

模块角度优化

“SwerveModuleState’’ 类包含一个静态 “optimize () “（Java）/“Optimize () “（C ++）方法，用于 “优化’’ 给定 “SwerveModuleState’’ 的速度和角度设定值 “以最大程度减少航向的变化。例如，如果逆运动学中某个模块的角度设定点为 90 度，但是您当前的角度为-89 度，则此方法将自动抵消模块设定点的速度并使角度设定点为-90 度以减小距离模块必须行进。

该方法有两个参数：所需状态（通常从 toSwerveModuleStates 方法）和当前角度。它将返回新的优化状态，您可以将其用作反馈控制回路中的设定值。

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()))
```

定向驱动

:ref:‘Recall<docs/software/kinematics-and-odometry/intro-and-chassis-speeds: 从现场相关速度中创建 ChassisSpeeds 对象 >‘可以从一组所需的现场导向速度中创建 “ChassisSpeeds“对象。这个功能可以用来从一组所需的定向驱动速度获得模块状态。

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
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)
```

使用自定义旋转中心

有时，对于某些规避操作，可能需要绕一个特定的角旋转。WPILib 也支持这种类型的行为。相同的 ToSwerveSpeeds () 方法接受旋转中心的第二个参数（作为 Translation2d）。就像轮子的位置一样，代表旋转中心的 “Translation2d’’ 应该相对于机器人中心。

备注： 由于所有机器人都是一个刚性框架，因此，从 “底盘速度 “对象中提供的 “vx “和 “vy “速度仍然适用于整个机器人。然而，来自 “底盘速度 “对象的 “omega“将从旋转中心测量。

例如，可以将旋转中心设置在某个轮子上，如果提供的 “ChassisSpeeds“对象的 “vx“和 “vy“为零，而 “omega“非零，机器人就会出现围绕该特定的转向模块旋转。

27.4.4 将模块状态转换为底盘速度

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
    });
  }
}
```

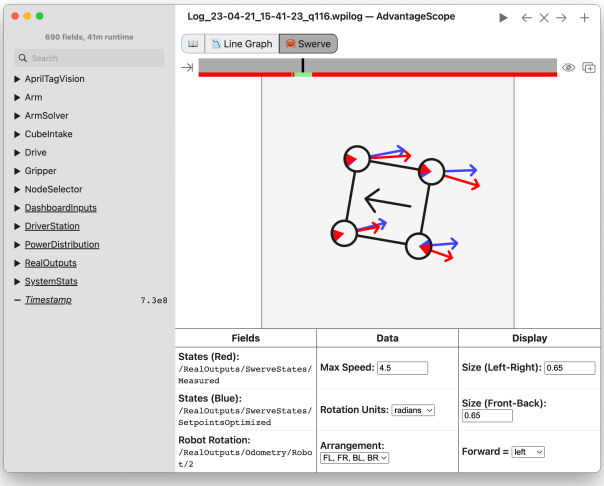
C++

```
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 转向驱动测距法

用户可以使用转向驱动器运动学类来执行:ref: odometry <docs/software/kinematics-and-odometry/intro-and-chassis-speeds:What is odometry?>‘。WPILib 包含一个“SwerveDriveOdometry’’类，该类可用于追踪场上的转向驱动机器人的位置。

备注： 由于这种方法只使用编码器和陀螺仪，机器人在场上的位置估计会随着时间的推移而漂移，特别是当你的机器人在游戏过程中与其他机器人接触时。不过，在自主期，测距法通常非常准确。

27.5.1 创建测距法对象

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`.

备注： 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 更新机器人姿势

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.

调用此"update"方法必需为周期性,最好是在 Subsystem<docs/software/commandbased/subsystems:Subsystems 的" periodic()" 方法中调用。"update" 方法将返回机器人的新的姿势。

JAVA

```
@Override
public void periodic() {
  // Get the rotation of the robot from the gyro.
  var gyroAngle = m_gyro.getRotation2d();

  // Update the pose
  m_pose = m_odometry.update(gyroAngle,
    new SwerveModulePosition[] {
      m_frontLeftModule.getPosition(), m_frontRightModule.getPosition(),
      m_backLeftModule.getPosition(), m_backRightModule.getPosition()
    });
}
```

C++

```
void Periodic() override {
  // Get the rotation of the robot from the gyro.
  frc::Rotation2d gyroAngle = m_gyro.GetRotation2d();

  // Update the pose
  m_pose = m_odometry.Update(gyroAngle,
    {
      m_frontLeftModule.GetPosition(), m_frontRightModule.GetPosition(),
      m_backLeftModule.GetPosition(), m_backRightModule.GetPosition()
    };
  )
}
```

PYTHON

```
def periodic(self):
  # Get the rotation of the robot from the gyro.
  self.gyroAngle = self.gyro.getRotation2d()

  # Update the pose
  self.pose = self.odometry.update(self.gyroAngle,
      self.frontLeftModule.getPosition(), self.frontRightModule.getPosition(),
      self.backLeftModule.getPosition(), self.backRightModule.getPosition()
  )
```

27.5.3 重置机器人姿势

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.

重要： 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.

备注： 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 麦克纳姆驱动运动学

“MecanumDriveKinematics“类是一个有用的工具，它可以在 “ChassisSpeeds“对象和 “MecanumDriveWheelSpeeds“对象之间进行转换，后者包含了麦克纳姆轮驱动器上四个车轮的速度。

备注： Mecanum kinematics uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.6.1 构造运动学对象

`MecanumDriveKinematics``类接受四个构造函数参数，每个参数是相对于机器人中心的轮子的位置（作为``Translation2d`）。参数的顺序是前左，前右，后左，后右。轮子的位置必须相对于机器人的中心。正的 x 值代表向机器人前面移动，正的 y 值代表向机器人左边移动。

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 将底盘速度转换为车轮速度

The `toWheelSpeeds(ChassisSpeeds speeds)` (Java / Python) / `ToWheelSpeeds(ChassisSpeeds speeds)` (C++) method should be used to convert a `ChassisSpeeds` object to a `MecanumDriveWheelSpeeds` object. This is useful in situations where you have to convert a forward velocity, sideways velocity, and an angular velocity into individual wheel speeds.

JAVA

```
// Example chassis speeds: 1 meter per second forward, 3 meters
// per second to the left, and rotation at 1.5 radians per second
// counterclockwise.
ChassisSpeeds speeds = new ChassisSpeeds(1.0, 3.0, 1.5);

// Convert to wheel speeds
MecanumDriveWheelSpeeds wheelSpeeds = kinematics.toWheelSpeeds(speeds);

// Get the individual wheel speeds
double frontLeft = wheelSpeeds.frontLeftMetersPerSecond
double frontRight = wheelSpeeds.frontRightMetersPerSecond
```

(续下页)

(接上页)

```
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)
```

定向驱动

:ref:‘Recall <docs/software/kinematics-and-odometry/intro-and-chassis-speeds:Creating a ChassisSpeeds object from field-relative speeds>“‘ChassisSpeeds“对象可以从一组所需的场向速度中创建。这个功能可以用来从一组期望的场向速度中获得轮速。

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)
```

使用自定义旋转中心

有时，对于某些规避操作，可能需要绕一个特定的角旋转。WPILib 也支持这种类型的行为。相同的 ToWheelSpeeds () 方法接受旋转中心的第二个参数（作为 Translation2d）。就像轮子的位置一样，代表旋转中心的 “Translation2d’’ 应该相对于机器人中心。

备注： 由于所有机器人都是一个刚性框架，因此，从 “底盘速度 “对象中提供的 “vx “和 “vy “速度仍然适用于整个机器人。然而，来自 “底盘速度 “对象的 “omega“将从旋转中心测量。

例如，可以将旋转中心设置在某个轮子上，如果提供的 “ChassisSpeeds“对象的 “vx“和 “vy“为零，而 “omega“非零，机器人就会出现围绕该轮子旋转。

27.6.3 将车轮速度转换为底盘速度

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 麦克纳姆驱动测距法

用户可以使用麦克纳姆驱动器运动学类来执行:ref odometry <docs/software/kinematics-and-odometry/intro-and-chassis-speeds:What is odometry?>。WPILib 包含一个 “MecanumDriveOdometry’’ 类，可用于跟踪麦克纳姆驱动机器人在现场的位置。

备注： 由于这种方法只使用编码器和陀螺仪，机器人在场上的位置估计会随着时间的推移而漂移，特别是当你的机器人在游戏过程中与其他机器人接触时。不过，在自主期，测距法通常非常准确。

27.7.1 创建测距法对象

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`.

备注： 0 degrees / radians represents the robot angle when the robot is facing directly toward your opponent’s alliance station. As your robot turns to the left, your gyroscope angle should increase. The `Gyro` interface supplies `getRotation2d`/`GetRotation2d` that you can use for this purpose. See *Coordinate System* for more information about the coordinate system.

JAVA

```
// Locations of the wheels relative to the robot center.
Translation2d m_frontLeftLocation = new Translation2d(0.381, 0.381);
Translation2d m_frontRightLocation = new Translation2d(0.381, -0.381);
Translation2d m_backLeftLocation = new Translation2d(-0.381, 0.381);
Translation2d m_backRightLocation = new Translation2d(-0.381, -0.381);

// Creating my kinematics object using the wheel locations.
MecanumDriveKinematics m_kinematics = new MecanumDriveKinematics(
  m_frontLeftLocation, m_frontRightLocation, m_backLeftLocation, m_backRightLocation
);

// Creating my odometry object from the kinematics object and the initial wheel␣
↪positions.
// Here, our starting pose is 5 meters along the long end of the field and in the
// center of the field along the short end, facing the opposing alliance wall.
```

(续下页)

(接上页)

```
MecanumDriveOdometry m_odometry = new MecanumDriveOdometry(
  m_kinematics,
  m_gyro.getRotation2d(),
  new MecanumDriveWheelPositions(
    m_frontLeftEncoder.getDistance(), m_frontRightEncoder.getDistance(),
    m_backLeftEncoder.getDistance(), m_backRightEncoder.getDistance()
  ),
  new Pose2d(5.0, 13.5, new Rotation2d())
);
```

C++

```
// Locations of the wheels relative to the robot center.
frc::Translation2d m_frontLeftLocation{0.381_m, 0.381_m};
frc::Translation2d m_frontRightLocation{0.381_m, -0.381_m};
frc::Translation2d m_backLeftLocation{-0.381_m, 0.381_m};
frc::Translation2d m_backRightLocation{-0.381_m, -0.381_m};

// Creating my kinematics object using the wheel locations.
frc::MecanumDriveKinematics m_kinematics{
  m_frontLeftLocation, m_frontRightLocation,
  m_backLeftLocation, m_backRightLocation
};

// Creating my odometry object from the kinematics object. Here,
// our starting pose is 5 meters along the long end of the field and in the
// center of the field along the short end, facing forward.
frc::MecanumDriveOdometry m_odometry{
  m_kinematics,
  m_gyro.GetRotation2d(),
  frc::MecanumDriveWheelPositions{
    units::meter_t{m_frontLeftEncoder.GetDistance()},
    units::meter_t{m_frontRightEncoder.GetDistance()},
    units::meter_t{m_backLeftEncoder.GetDistance()},
    units::meter_t{m_backRightEncoder.GetDistance()}
  },
  frc::Pose2d{5_m, 13.5_m, 0_rad}};
```

PYTHON

```
from wpimath.geometry import Translation2d
from wpimath.kinematics import MecanumDriveKinematics
from wpimath.kinematics import MecanumDriveOdometry
from wpimath.kinematics import MecanumDriveWheelPositions
from wpimath.geometry import Pose2d
from wpimath.geometry import Rotation2d

# Locations of the wheels relative to the robot center.
frontLeftLocation = Translation2d(0.381, 0.381)
frontRightLocation = Translation2d(0.381, -0.381)
backLeftLocation = Translation2d(-0.381, 0.381)
backRightLocation = Translation2d(-0.381, -0.381)
```

(续下页)

(接上页)

```
# 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 更新机器人姿势

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 重置机器人姿势

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.

重要： 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.

备注： 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 网络表

This section outlines the details of using the NetworkTables (v4) API to communicate information across the robot network.

重要： 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 什么是网络表

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.

备注： NetworkTables has changed substantially in 2023. For more information on migrating pre-2023 code to use the new features, see *Migrating from NetworkTables 3.0 to NetworkTables 4.0*.

28.1.1 NetworkTables Concepts

First, let's define some terms:

- **Topic**: a named data channel. Topics have a fixed data type (for the lifetime of the topic) and *mutable* properties.
- **Publisher**: defines the topic and creates and sends timestamped data values.
- **Subscriber**: receives timestamped data value updates to one or more topics.
- **Entry**: a combined publisher and subscriber. The subscriber is always active, but the publisher is not created until a publish operation is performed (e.g. a value is "set", aka published, on the entry). This may be more convenient than maintaining a separate publisher and subscriber.
- **Property**: named information (metadata) about a topic stored and updated separately from the topic's data. A topic may have any number of properties. A property's value can be any data type that can be represented in JSON.

NetworkTables supports a range of data types, including *boolean*, numeric, string, and arrays of those types. Supported numeric data types are single or double precision *floating point*, or 64-bit integer. There is also the option of storing raw data (an array of bytes), which can be used for representing binary encoded structured data. Types are represented as strings for efficiency reasons. There is also an *enumeration* for the most common types in the NetworkTables API.

Topics are created when the first publisher announces the topic and are removed when the last publisher stops publishing. It's possible to subscribe to a topic that has not yet been created/published.

Topics have properties. Properties are initially set by the first publisher, but may be changed at any time. Similarly to values, property changes to a topic are propagated to all subscribers to that topic. Properties are structured data (JSON), but at the top level are simply a key/value store (a JSON map). Some properties have defined behavior, but arbitrary ones can be set by the application.

Publishers specify the topic's data type; while there can be multiple publishers to a single topic, they must all be publishing the same data type. This is enforced by the NetworkTables server (the first publisher "wins"). Typically single-topic subscribers also specify what data type they're expecting to receive on a topic and thus won't receive value updates of other types.

28.1.2 Retained and Persistent Topics

While by default topics are *transitory* and disappear after the last publisher stops publishing, topics can be marked as *retained* (via setting the "retained" property to true) to prevent them from disappearing. For retained topics, the server acts as an implicit publisher of the last value, and will keep doing so as long as the server is running. This is primarily useful for configuration values; e.g. an autonomous mode selection published by a dashboard should set the topic as retained so its value is preserved in case the dashboard disconnects.

Additionally, topics can be marked as *persistent* via setting the "persistent" property to true. These operate similarly to retained topics, but in addition, persistent topic values are automatically saved to a file on the server and when the server starts up again, the topic is created and its last value is published by the server.

28.1.3 Value Propagation

The server keeps a copy of the last published value for every topic. When a subscriber initially subscribes to a topic, the server sends the last published value. After that initial value, new value updates are communicated to subscribers each time the publisher sends a new value.

NetworkTables is a client/server system; clients do not talk directly to each other, but rather communicate via the server. Typically, the robot program is the server, and other pieces of software on other computers (e.g. the driver station or a coprocessor) are clients that connect to it. Thus, when a coprocessor (client) publishes a value, the value is sent first from the coprocessor (client) to the robot program (server), and then the robot program distributes that value to any subscribers (e.g. the robot program local program, or other clients such as dashboards).

The server does not send topic changes or value updates to clients that have not subscribed to the topic.

By default, NetworkTables sends value updates periodically, batching the data to help limit the number of small packets being sent over the network. Also, by default, only the most recent value is transmitted; any intermediate value changes made between network transmissions are discarded. This behavior can be changed via publish/subscribe options–publishers and subscribers can indicate that all value updates should be preserved and communicated via the "send all" option. In addition, it is possible to force NetworkTables to "flush" all current updates to the network; this is useful for minimizing latency.

28.1.4 Timestamps

All NetworkTable value updates are timestamped at the time they are published. Timestamps in NetworkTables are measured in integer microseconds.

NetworkTables automatically synchronizes time between the server and clients. Each client maintains an offset between the client local time and the server time, so when a client publishes a value, it stores a timestamp in local time and calculates the equivalent server timestamp. The server timestamp is what is communicated over the network to any subscribers. This makes it possible e.g. for a robot program to get a reasonable estimation of the time when a value was published on a coprocessor relative to the current time.

Because of this, two timestamps are visible through the API: a server timestamp indicating the time (estimated) on the server, and a local timestamp indicating the time on the client. When the RoboRIO is the NetworkTables server, the server timestamp is the same as the FPGA timestamp returned by `Timer.getFPGATimestamp()` (except the units are different: NetworkTables uses microseconds, while `getFPGATimestamp()` returns seconds).

28.1.5 NetworkTables Organization

Data is organized in NetworkTables in a hierarchy much like a filesystem's folders and files. There can be multiple subtables (folders) and topics (files) that may be nested in whatever way fits the data organization desired. At the top level (`NetworkTableInstance`: Java, C++, Python), topic names are handled similar to absolute paths in a filesystem: subtables are represented as a long topic name with slashes ("/") separating the nested subtable and value names. A `NetworkTable` (Java, C++, Python) object represents a single subtable (folder), so topic names are relative to the NetworkTable's base path: e.g. for a root table called "SmartDashboard" with a topic named "xValue", the same topic can be accessed via `NetworkTableInstance` as a topic named "/SmartDashboard/xValue". However, unlike

a filesystem, subtables don't really exist in the same way folders do, as there is no way to represent an empty subtable on the network–a subtable "appears" only as long as there are topics published within it.

OutlineViewer is a utility for exploring the values stored in NetworkTables, and can show either a flat view (topics with absolute paths) or a nested view (subtables and topics).

There are some default tables that are created automatically when a robot program starts up:

表名	用途
/SmartDashboard	用于存储使用 SmartDashboard.put() 方法集写入 SmartDashboard 或 Shuffleboard 的值。
/LiveWindow	用于存储测试模式（在驱动程序站上测试）值。这些通常是子系统以及关联的传感器和执行器。
/FMSInfo	来自 Driver Station 和现场管理系统的有关当前进行的比赛的信息

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
    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␣
```

(续下页)

(接上页)

```
↪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
    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

```
// 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
```

(续下页)

(接上页)

```
NT_Publisher dblPub = NT_Publish(dblTopic, NT_DOUBLE, "double", &options);

// PublishEx allows setting initial properties. The properties string must
// be a JSON map.
NT_Publisher dblPub =
    NT_PublishEx(dblTopic, NT_DOUBLE, "double", "{\"myprop\", 5}", NULL, 0);

// publish a default value
NT_SetDefaultDouble(dblPub, 0.0);

// publish a value with current timestamp
NT_SetDouble(dblPub, 1.0);
NT_SetDouble(dblPub, 2.0, 0);  // 0 = use current time

// publish a value with a specific timestamp; NT_Now() can
// be used to get the current time.
int64_t time = NT_Now();
NT_SetDouble(dblPub, 3.0, time);

// stop publishing
NT_Unpublish(dblPub);
```

Python

```
class Example:
    def __init__(self, dblTopic: ntcore.DoubleTopic):

        # start publishing; the return value must be retained (in this case,␣
↪via
        # an instance variable)
        self.dblPub = dblTopic.publish()

        # publish options may be specified using PubSubOption
        self.dblPub = dblTopic.publish(ntcore.
↪PubSubOptions(keepDuplicates=True))

        # publishEx provides additional options such as setting initial
        # properties and using a custom type string. Using a custom type␣
↪string for
        # types other than raw and string is not recommended. The properties␣
↪string
        # must be a JSON map.
        self.dblPub = dblTopic.publishEx("double", '{"myprop": 5}')

    def periodic(self):
        # publish a default value
        self.dblPub.setDefault(0.0)

        # publish a value with current timestamp
        self.dblPub.set(1.0)
        self.dblPub.set(2.0, 0)  # 0 = use current time

        # publish a value with a specific timestamp with microsecond␣
↪resolution.
```

(续下页)

(接上页)

```
        # On the roboRIO, this is the same as the FPGA timestamp (e.g.
        # RobotController.getFPGATime())
        self.dblPub.set(3.0, ntcore._now())

    # often not required in robot code, unless this class doesn't exist for
    # the lifetime of the entire robot program, in which case close() needs␣
↪to be
    # called to stop publishing
    def close(self):
        # stop publishing
        self.dblPub.close()
```

28.3.2 Subscribing to a Topic

A *subscriber* receives value updates made to a topic. Similar to publishers, NetworkTable subscribers are represented as type-specific Subscriber classes (e.g. `BooleanSubscriber`: Java, C++, Python) that must be stored somewhere to continue subscribing.

Subscribers have a range of different ways to read received values. It's possible to just read the most recent value using `get()`, read the most recent value, along with its timestamp, using `getAtomic()`, or get an array of all value changes since the last call using `readQueue()` or `readQueueValues()`.

Java

```
public class Example {
  // the subscriber is an instance variable so its lifetime matches that of␣
↪the class
  final DoubleSubscriber dblSub;

  public Example(DoubleTopic dblTopic) {
    // start subscribing; the return value must be retained.
    // the parameter is the default value if no value is available when␣
↪get() is called
    dblSub = dblTopic.subscribe(0.0);

    // subscribe options may be specified using PubSubOption
    dblSub =
        dblTopic.subscribe(0.0, PubSubOption.keepDuplicates(true),␣
↪PubSubOption.pollStorage(10));

    // subscribeEx provides the options of using a custom type string.
    // Using a custom type string for types other than raw and string is not␣
↪recommended.
    dblSub = dblTopic.subscribeEx("double", 0.0);
  }

  public void periodic() {
    // simple get of most recent value; if no value has been published,
    // returns the default value passed to the subscribe() function
    double val = dblSub.get();

    // get the most recent value; if no value has been published, returns
```

(续下页)

(接上页)

```
    // the passed-in default value
    double val = dblSub.get(-1.0);

    // subscribers also implement the appropriate Supplier interface, e.g.␣
↪DoubleSupplier
    double val = dblSub.getAsDouble();

    // get the most recent value, along with its timestamp
    TimestampedDouble tsVal = dblSub.getAtomic();

    // read all value changes since the last call to readQueue/
↪readQueueValues
    // readQueue() returns timestamps; readQueueValues() does not.
    TimestampedDouble[] tsUpdates = dblSub.readQueue();
    double[] valUpdates = dblSub.readQueueValues();
  }

  // often not required in robot code, unless this class doesn't exist for
  // the lifetime of the entire robot program, in which case close() needs␣
↪to be
  // called to stop subscribing
  public void close() {
    // stop subscribing
    dblSub.close();
  }
}
```

C++

```
class Example {
  // the subscriber is an instance variable so its lifetime matches that of␣
↪the class
  // subscribing is automatically stopped when dblSub is destroyed by the␣
↪class destructor
  nt::DoubleSubscriber dblSub;

 public:
  explicit Example(nt::DoubleTopic dblTopic) {
    // start subscribing; the return value must be retained.
    // the parameter is the default value if no value is available when␣
↪get() is called
    dblSub = dblTopic.Subscribe(0.0);

    // subscribe options may be specified using PubSubOptions
    dblSub =
        dblTopic.subscribe(0.0,
        {.pollStorage = 10, .keepDuplicates = true});

    // SubscribeEx provides the options of using a custom type string.
    // Using a custom type string for types other than raw and string is not␣
↪recommended.
    dblSub = dblTopic.SubscribeEx("double", 0.0);
  }
```

(续下页)

(接上页)

```
  void Periodic() {
    // simple get of most recent value; if no value has been published,
    // returns the default value passed to the Subscribe() function
    double val = dblSub.Get();

    // get the most recent value; if no value has been published, returns
    // the passed-in default value
    double val = dblSub.Get(-1.0);

    // get the most recent value, along with its timestamp
    nt::TimestampedDouble tsVal = dblSub.GetAtomic();

    // read all value changes since the last call to ReadQueue/
↪ReadQueueValues
    // ReadQueue() returns timestamps; ReadQueueValues() does not.
    std::vector<nt::TimestampedDouble> tsUpdates = dblSub.ReadQueue();
    std::vector<double> valUpdates = dblSub.ReadQueueValues();
  }
};
```

C++ (Handle-based)

```
class Example {
  // the subscriber is an instance variable, but since it's a handle, it's
  // not automatically released, so we need a destructor
  NT_Subscriber dblSub;

 public:
  explicit Example(NT_Topic dblTopic) {
    // start subscribing
    // Using a custom type string for types other than raw and string is not␣
↪recommended.
    dblSub = nt::Subscribe(dblTopic, NT_DOUBLE, "double");

    // subscribe options may be specified using PubSubOptions
    dblSub =
        nt::Subscribe(dblTopic, NT_DOUBLE, "double",
        {.pollStorage = 10, .keepDuplicates = true});
  }

  void Periodic() {
    // get the most recent value; if no value has been published, returns
    // the passed-in default value
    double val = nt::GetDouble(dblSub, 0.0);

    // get the most recent value, along with its timestamp
    nt::TimestampedDouble tsVal = nt::GetAtomic(dblSub, 0.0);

    // read all value changes since the last call to ReadQueue/
↪ReadQueueValues
    // ReadQueue() returns timestamps; ReadQueueValues() does not.
    std::vector<nt::TimestampedDouble> tsUpdates =␣
↪nt::ReadQueueDouble(dblSub);
    std::vector<double> valUpdates = nt::ReadQueueValuesDouble(dblSub);
```

(续下页)

(接上页)

```
  }

  ~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;
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

```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
    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);
    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);
  }

  ~Example() {
    // stop publishing and subscribing
    nt::ReleaseEntry(dblEntry);
  }
```

C

```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

```
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);
```

(续下页)

(接上页)

```
    // 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");
```

(续下页)

(接上页)

```
    // 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

```
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
```

(续下页)

(接上页)

```
        # 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.
```

(续下页)

(接上页)

```
↪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();
```

(续下页)

(接上页)

```
    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(
```

(续下页)

(接上页)

```
            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
```

(续下页)

(接上页)

```
      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);
```

(续下页)

(接上页)

```
    }

    // 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.

> **警告:** 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");
```

(续下页)

(接上页)

```
    // 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"
```

(续下页)

(接上页)

```
            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)
```

(续下页)

(接上页)

```
        # 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

```
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++

```
#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() {
```

(续下页)

(接上页)

```
    // 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()
```

(续下页)

(接上页)

```
        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.

基本的客户端程序如下例所示。

Java

```
import edu.wpi.first.networktables.DoubleSubscriber;
import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableInstance;
import edu.wpi.first.networktables.NetworkTablesJNI;
import edu.wpi.first.util.CombinedRuntimeLoader;

import java.io.IOException;

import edu.wpi.first.cscore.CameraServerJNI;
import edu.wpi.first.math.WPIMathJNI;
import edu.wpi.first.util.WPIUtilJNI;


public class Program {
    public static void main(String[] args) throws IOException {
        NetworkTablesJNI.Helper.setExtractOnStaticLoad(false);
        WPIUtilJNI.Helper.setExtractOnStaticLoad(false);
        WPIMathJNI.Helper.setExtractOnStaticLoad(false);
        CameraServerJNI.Helper.setExtractOnStaticLoad(false);

        CombinedRuntimeLoader.loadLibraries(Program.class, "wpiutiljni",
↪"wpimathjni", "ntcorejni",
                "cscorejnicvstatic");
        new Program().run();
    }

    public void run() {
        NetworkTableInstance inst = NetworkTableInstance.getDefault();
        NetworkTable table = inst.getTable("datatable");
        DoubleSubscriber xSub = table.getDoubleTopic("x").subscribe(0.0);
```

(续下页)

(接上页)

```
        DoubleSubscriber ySub = table.getDoubleTopic("y").subscribe(0.0);
        inst.startClient4("example client");
        inst.setServer("localhost"); // where TEAM=190, 294, etc, or use␣
↪inst.setServer("hostname") or similar
        inst.startDSClient(); // recommended if running on DS computer; this␣
↪gets the robot IP from the DS
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                System.out.println("interrupted");
                return;
            }
            double x = xSub.get();
            double y = ySub.get();
            System.out.println("X: " + x + " Y: " + y);
        }
    }
}
```

C++

```
#include <chrono>
#include <thread>
#include <fmt/format.h>
#include <networktables/NetworkTableInstance.h>
#include <networktables/NetworkTable.h>
#include <networktables/DoubleTopic.h>

int main() {
  auto inst = nt::NetworkTableInstance::GetDefault();
  auto table = inst.GetTable("datatable");
  auto xSub = table->GetDoubleTopic("x").Subscribe(0.0);
  auto ySub = table->GetDoubleTopic("y").Subscribe(0.0);
  inst.StartClient4("example client");
  inst.SetServerTeam(TEAM);  // where TEAM=190, 294, etc, or use inst.
↪setServer("hostname") or similar
  inst.StartDSClient();  // recommended if running on DS computer; this gets␣
↪the robot IP from the DS
  while (true) {
    using namespace std::chrono_literals;
    std::this_thread::sleep_for(1s);
    double x = xSub.Get();
    double y = ySub.Get();
    fmt::print("X: {} Y: {}\n", x, y);
  }
}
```

C++ (Handle-based)

```
#include <chrono>
#include <thread>
#include <fmt/format.h>
#include <ntcore_cpp.h>

int main() {
  NT_Inst inst = nt::GetDefaultInstance();
  NT_Subscriber xSub =
      nt::Subscribe(nt::GetTopic(inst, "/datatable/x"), NT_DOUBLE, "double");
  NT_Subscriber ySub =
      nt::Subscribe(nt::GetTopic(inst, "/datatable/y"), NT_DOUBLE, "double");
  nt::StartClient4(inst, "example client");
  nt::SetServerTeam(inst, TEAM, 0);  // where TEAM=190, 294, etc, or use␣
↪inst.setServer("hostname") or similar
  nt::StartDSClient(inst, 0);  // recommended if running on DS computer;␣
↪this gets the robot IP from the DS
  while (true) {
    using namespace std::chrono_literals;
    std::this_thread::sleep_for(1s);
    double x = nt::GetDouble(xSub, 0.0);
    double y = nt::GetDouble(ySub, 0.0);
    fmt::print("X: {} Y: {}\n", x, y);
  }
}
```

C

```
#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".

然后，该实例作为具有组号的 NetworkTables 客户端启动（roboRIO 始终是服务器）。此外，如果程序在机器操控台计算机上运行，则使用 startDSClient () 方法，NetworkTables 将从机器操控台获取机械手 IP 地址。

然后，此示例程序简单地每秒循环一次，获取 x 和 y 的值并将其打印在控制台上。在现实的程序中，客户端可能正在处理或生成供机器人使用的值。

28.8.1 使用 Gradle 构建

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'
11 }
12
```

(续下页)

(接上页)

```
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();
  }
```

(续下页)

(接上页)

```
  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

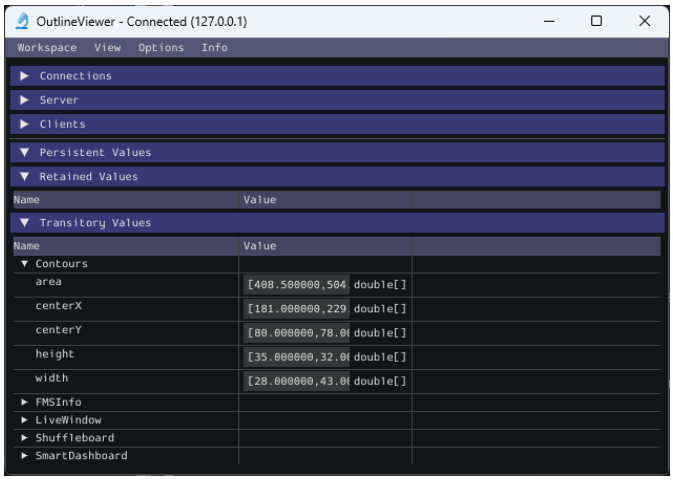
C++ values are now returned/used as value objects (plain `nt::Value`) instead of shared pointers to them (`std::shared_ptr<nt::Value>`).

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.

通常，这些值是针对一个或多个感兴趣区域（例如目标或游戏片段）的，并且会返回多个实例。在下面的示例中，图像处理器返回了多个 `x`，`y`，宽度，高度和区域，并且机器人程序可以通过进一步的处理来找出返回的值中哪些是有趣的。

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.

下面的两个示例都是高度简化的程序，仅说明了网络表的用法。所有代码都在 robotInit() 方法中，因此仅在程序启动时运行。在您的程序中，这些值的获取大多会在比赛的自动阶段或手动阶段，使机器人瞄向一个方向的指令或控制循环中进行。

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) {
      System.out.print(area + " ");
    }
```

(续下页)

(接上页)

```
    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;
}
```

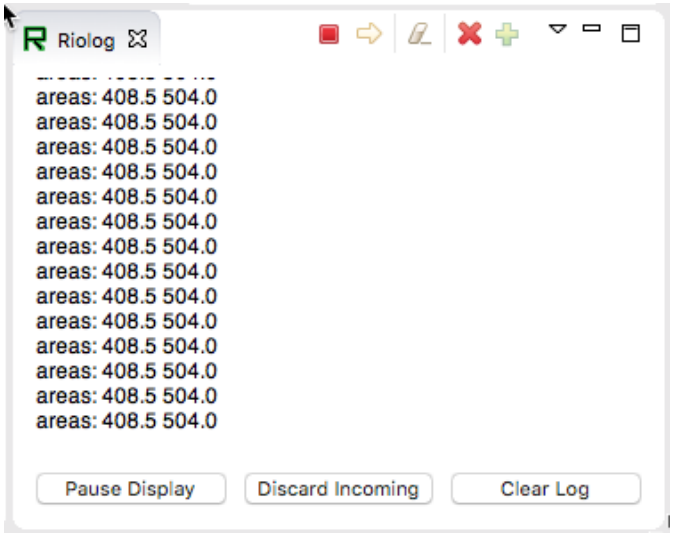
PYTHON

```
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)
```

获取值并在此程序中打印它们的步骤是：

1. 声明表变量，该变量将保存具有值的子表的实例。
2. 初始化子表实例，以便以后可用于检索值。
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



在这种情况下，程序仅查看关于 areas 数组，但在实际示例中，更可能所有值都会被使用。使用 VS Code 中的 Riolog 或 Driver Station 日志，您可以看到被检索的值。该程序使用的是示例静态图像，因此区域数值不会改变。但是您可以想象在机器人上安装摄像头后，值会不断变化。

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++).

Trajectory Tutorial Overview

备注: Before following this tutorial, it is helpful (but not strictly necessary) to have a baseline familiarity with WPILib's *PID control*, *feedforward*, and *trajectory* features.

备注: The robot code in this tutorial uses the *command-based* framework. The command-based framework is strongly recommended for beginning and intermediate teams.

The goal of this tutorial is to provide "end-to-end" instruction on implementing a trajectory-following autonomous routine for a differential-drive robot. By following this tutorial, readers will learn how to:

1. Accurately characterize their robot's drivetrain to obtain accurate feedforward calculations and approximate feedback gains.
2. Configure a drive subsystem to track the robot's pose using WPILib's odometry library.
3. Generate a simple trajectory through a set of waypoints using WPILib's `TrajectoryGenerator` class.
4. Follow the generated trajectory in an autonomous routine using WPILib's `RamseteCommand` class with the calculated feedforward/feedback gains and pose.

This tutorial is intended to be approachable for teams without a great deal of programming expertise. While the WPILib library offers significant flexibility in the manner in which its trajectory-following features are implemented, closely following the implementation outlined in this tutorial should provide teams with a relatively-simple, clean, and repeatable solution for autonomous movement.

The full robot code for this tutorial can be found in the RamseteCommand Example Project (Java, C++).

Why Trajectory Following?

FRC® games often feature autonomous tasks that require a robot to effectively and accurately move from a known starting location to a known scoring location. Historically, the most common solution for this sort of task in FRC has been a "drive-turn-drive" approach - that is, drive forward by a known distance, turn by a known angle, and drive forward by another known distance.

While the "drive-turn-drive" approach is certainly functional, in recent years teams have begun tracking smooth trajectories which require the robot to drive and turn at the same time. While this is a fundamentally more-complicated technical task, it offers significant benefits: in particular, since the robot no longer has to stop to change directions, the paths can be driven much faster, allowing a robot to score more game pieces during the autonomous period.

Beginning in 2020, WPILib now supplies teams with working, advanced code solutions for trajectory generation and tracking, significantly lowering the "barrier-to-entry" for this kind of advanced and effective autonomous motion.

Required Equipment

To follow this tutorial, you will need ready access to the following materials:

1. A differential-drive robot (such as the AndyMark AM14U5), equipped with:
   - Quadrature encoders for measuring the wheel rotation of each side of the drive.
   - A gyroscope for measuring robot heading.
2. A driver-station computer configured with:
   - *FRC Driver Station*.
   - *WPILib*.
   - *The System Identification Toolsuite*.

Step 1: Characterizing Your Robot Drive

备注: For detailed instructions on using the System Identification tool, see its *dedicated documentation*.

备注: 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.

备注: 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.

Gathering the Data

We begin by gathering our drive identification data.

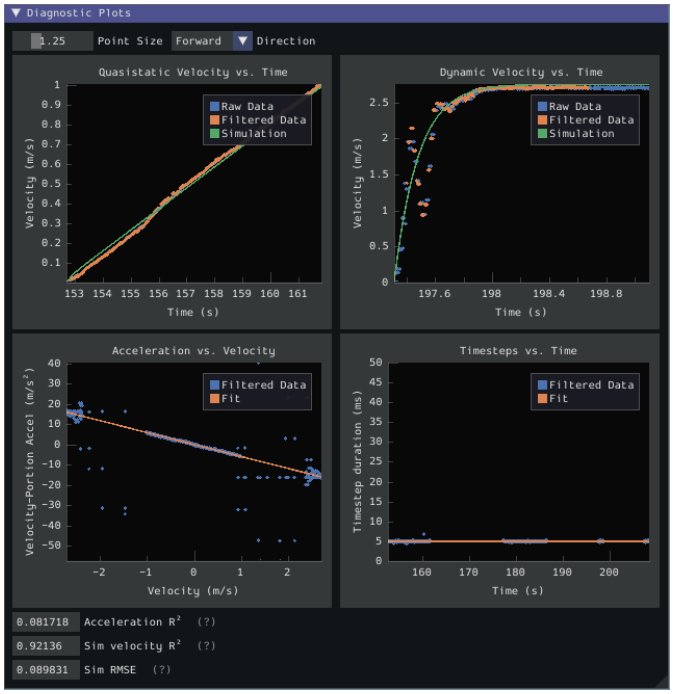
1. *Configure and Deploy your robot project*.
2. *Run the identification Routine*.

Analyzing the Data

Once the identification routine has been run and the data file has been saved, it is time to *open it in the analysis pane*.

Checking Diagnostics

Per the *system identification guide*, we first view the diagnostics to ensure that our data look reasonable:

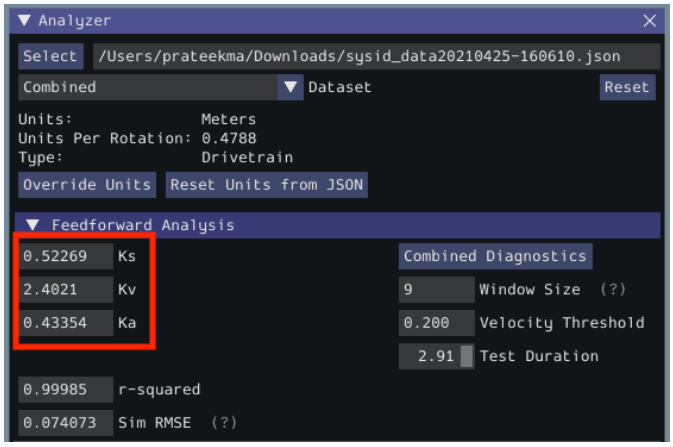


As our data look reasonably linear, and the fit metrics are within acceptable parameters, we proceed to the next step.

Record Feedforward Gains

备注： Feedforward gains do *not*, in general, transfer across robots. Do *not* use the gains from this tutorial for your own robot.

We now record the feedforward gains calculated by the tool:



Since our wheel diameter was specified in meters, our feedforward gains are in the following units:

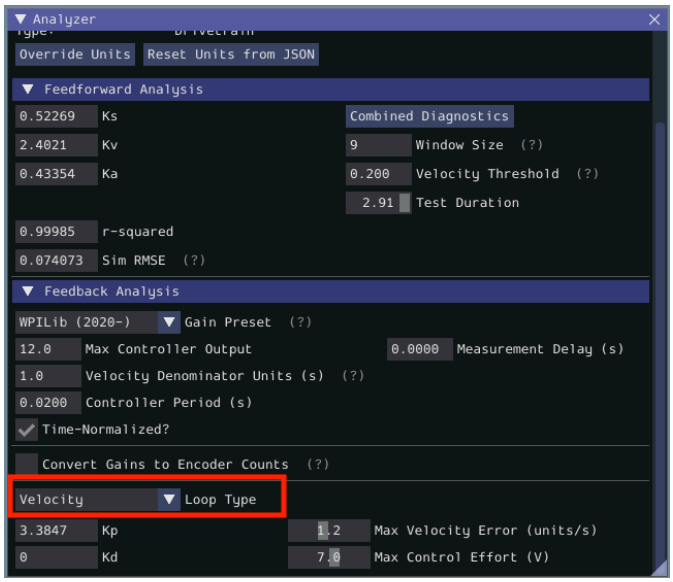
- kS: Volts
- kV: Volts * Seconds / Meters
- kA: Volts * Seconds^2 / Meters

If you have specified your units correctly, your feedforward gains will likely be within an order of magnitude of the ones reported here (a possible exception exists for kA, which may be vanishingly small if your robot is light). If they are not, it is possible you specified one of your drive parameters incorrectly when generating your robot project. A good test for this is to calculate the "theoretical" value of kV, which is 12 volts divided by the theoretical free speed of your drivetrain (which is, in turn, the free speed of the motor times the wheel circumference divided by the gear reduction). This value should agree very closely with the kV measured by the tool - if it does not, you have likely made an error somewhere.

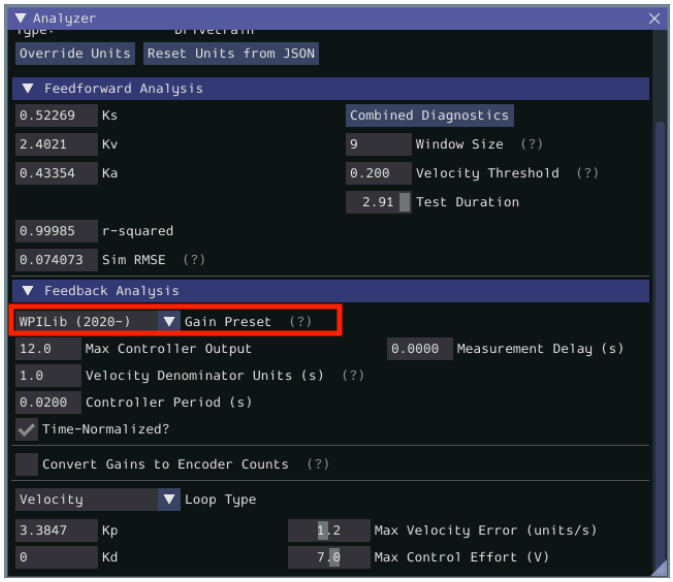
Calculate Feedback Gains

备注: Feedback gains do *not*, in general, transfer across robots. Do *not* use the gains from this tutorial for your own robot.

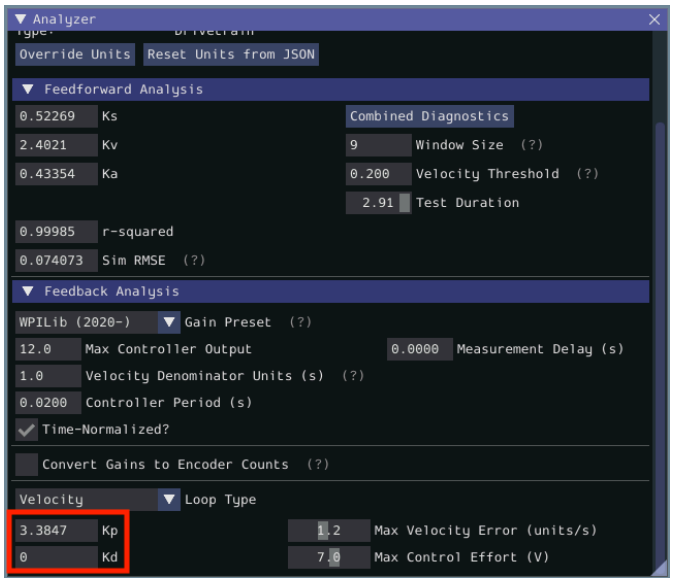
We now *calculate the feedback gains* for the PID control that we will use to follow the path. Trajectory following with WPILib's RAMSETE controller uses velocity closed-loop control, so we first select `Velocity` mode in the identification tool:



Since we will be using the WPILib PIDController for our velocity loop, we furthermore select the `WPILib (2020-)` option from the drop-down "presets" menu. This is *very* important, as the feedback gains will not be in the correct units if we do not select the correct preset:



Finally, we calculate and record the feedback gains for our control loop. Since it is a velocity controller, only a P gain is required:



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.

Step 2: Entering the Calculated Constants

备注： In C++, it is important that the feedforward constants be entered as the correct unit type. For more information on C++ units, see *C++ 单位库*.

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.

Feedforward/Feedback Gains

Firstly, we must enter the feedforward and feedback gains which we obtained from the identification tool.

备注： Feedforward and feedback gains do *not*, in general, transfer across robots. Do *not* use the gains from this tutorial for your own 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

Additionally, we must create an instance of the `DifferentialDriveKinematics` class, which allows us to use the trackwidth (i.e. horizontal distance between the wheels) of the robot to convert from chassis speeds to wheel speeds. As elsewhere, we keep our units in meters.

JAVA

```
    public static final double kTrackwidthMeters = 0.69;
    public static final DifferentialDriveKinematics kDriveKinematics =
        new DifferentialDriveKinematics(kTrackwidthMeters);
```

C++

```
38 inline constexpr auto kTrackwidth = 0.69_m;
39 extern const frc::DifferentialDriveKinematics kDriveKinematics;
```

Max Trajectory Velocity/Acceleration

We must also decide on a nominal max acceleration and max velocity for the robot during path-following. The maximum velocity value should be set somewhat below the nominal free-speed of the robot. Due to the later use of the `DifferentialDriveVoltageConstraint`, the maximum acceleration value is not extremely crucial.

> **警告:** 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

```
57     public static final double kMaxSpeedMetersPerSecond = 3;
58     public static final double kMaxAccelerationMetersPerSecondSquared = 1;
```

C++

```
61 inline constexpr auto kMaxSpeed = 3_mps;
62 inline constexpr auto kMaxAcceleration = 1_mps_sq;
```

Ramsete Parameters

Finally, we must include a pair of parameters for the RAMSETE controller. The values shown below should work well for most robots, provided distances have been correctly measured in meters - for more information on tuning these values (if it is required), see *构造 Ramsete 控制器对象*.

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++

```
// Reasonable baseline values for a RAMSETE follower in units of meters and
// seconds
inline constexpr auto kRamseteB = 2.0 * 1_rad * 1_rad / (1_m * 1_m);
inline constexpr auto kRamseteZeta = 0.7 / 1_rad;
```

Step 3: Creating a Drive Subsystem

Now that our drive is characterized, it is time to start writing our robot code *proper*. As mentioned before, we will use the *command-based* framework for our robot code. Accordingly, our first step is to write a suitable drive *subsystem* class.

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

```
package edu.wpi.first.wpilibj.examples.ramsetecommand.subsystems;

import edu.wpi.first.math.geometry.Pose2d;
import edu.wpi.first.math.kinematics.DifferentialDriveOdometry;
import edu.wpi.first.math.kinematics.DifferentialDriveWheelSpeeds;
import edu.wpi.first.util.sendable.SendableRegistry;
import edu.wpi.first.wpilibj.ADXRS450_Gyro;
import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import edu.wpi.first.wpilibj.examples.ramsetecommand.Constants.DriveConstants;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
import edu.wpi.first.wpilibj2.command.SubsystemBase;

public class DriveSubsystem extends SubsystemBase {
  // The motors on the left side of the drive.
  private final PWMSparkMax m_leftLeader = new PWMSparkMax(DriveConstants.
↪kLeftMotor1Port);
  private final PWMSparkMax m_leftFollower = new PWMSparkMax(DriveConstants.
↪kLeftMotor2Port);

  // The motors on the right side of the drive.
  private final PWMSparkMax m_rightLeader = new PWMSparkMax(DriveConstants.
↪kRightMotor1Port);
  private final PWMSparkMax m_rightFollower = new PWMSparkMax(DriveConstants.
↪kRightMotor2Port);

  // The robot's drive
  private final DifferentialDrive m_drive =
      new DifferentialDrive(m_leftLeader::set, m_rightLeader::set);

  // The left-side drive encoder
  private final Encoder m_leftEncoder =
      new Encoder(
          DriveConstants.kLeftEncoderPorts[0],
          DriveConstants.kLeftEncoderPorts[1],
          DriveConstants.kLeftEncoderReversed);
```

(续下页)

(接上页)

```
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    }
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.
```

(续下页)

(接上页)

```
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();
134      m_rightEncoder.reset();
135    }
136
137    /**
138     * Gets the average distance of the two encoders.
139     *
140     * @return the average of the two encoder readings
141     */
142    public double getAverageEncoderDistance() {
143      return (m_leftEncoder.getDistance() + m_rightEncoder.getDistance()) / 2.0;
144    }
145
146    /**
```

(续下页)

(接上页)

```
147     * Gets the left drive encoder.
148     *
149     * @return the left drive encoder
150     */
151    public Encoder getLeftEncoder() {
152      return m_leftEncoder;
153    }
154
155    /**
156     * Gets the right drive encoder.
157     *
158     * @return the right drive encoder
159     */
160    public Encoder getRightEncoder() {
161      return m_rightEncoder;
162    }
163
164    /**
165     * Sets the max output of the drive. Useful for scaling the drive to drive more␣
    ↪slowly.
166     *
167     * @param maxOutput the maximum output to which the drive will be constrained
168     */
169    public void setMaxOutput(double maxOutput) {
170      m_drive.setMaxOutput(maxOutput);
171    }
172
173    /** Zeroes the heading of the robot. */
174    public void zeroHeading() {
175      m_gyro.reset();
176    }
177
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    }
186
187    /**
188     * Returns the turn rate of the robot.
189     *
190     * @return The turn rate of the robot, in degrees per second
191     */
192    public double getTurnRate() {
193      return -m_gyro.getRate();
194    }
195  }
```

C++ (Header)

```
#pragma once

#include <frc/ADXRS450_Gyro.h>
#include <frc/Encoder.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/geometry/Pose2d.h>
#include <frc/kinematics/DifferentialDriveOdometry.h>
#include <frc/motorcontrol/PWMSparkMax.h>
#include <frc2/command/SubsystemBase.h>
#include <units/voltage.h>

#include "Constants.h"

class DriveSubsystem : public frc2::SubsystemBase {
 public:
  DriveSubsystem();

  /**
   * Will be called periodically whenever the CommandScheduler runs.
   */
  void Periodic() override;

  // Subsystem methods go here.

  /**
   * Drives the robot using arcade controls.
   *
   * @param fwd the commanded forward movement
   * @param rot the commanded rotation
   */
  void ArcadeDrive(double fwd, double rot);

  /**
   * Controls each side of the drive directly with a voltage.
   *
   * @param left the commanded left output
   * @param right the commanded right output
   */
  void TankDriveVolts(units::volt_t left, units::volt_t right);

  /**
   * Resets the drive encoders to currently read a position of 0.
   */
  void ResetEncoders();

  /**
   * Gets the average distance of the TWO encoders.
   *
   * @return the average of the TWO encoder readings
   */
  double GetAverageEncoderDistance();

  /**
   * Gets the left drive encoder.
   *
```

(续下页)

(接上页)

```
   * @return the left drive encoder
   */
  frc::Encoder& GetLeftEncoder();

  /**
   * Gets the right drive encoder.
   *
   * @return the right drive encoder
   */
  frc::Encoder& GetRightEncoder();

  /**
   * Sets the max output of the drive.  Useful for scaling the drive to drive
   * more slowly.
   *
   * @param maxOutput the maximum output to which the drive will be constrained
   */
  void SetMaxOutput(double maxOutput);

  /**
   * Returns the heading of the robot.
   *
   * @return the robot's heading in degrees, from -180 to 180
   */
  units::degree_t GetHeading() const;

  /**
   * Returns the turn rate of the robot.
   *
   * @return The turn rate of the robot, in degrees per second
   */
  double GetTurnRate();

  /**
   * Returns the currently-estimated pose of the robot.
   *
   * @return The pose.
   */
  frc::Pose2d GetPose();

  /**
   * Returns the current wheel speeds of the robot.
   *
   * @return The current wheel speeds.
   */
  frc::DifferentialDriveWheelSpeeds GetWheelSpeeds();

  /**
   * Resets the odometry to the specified pose.
   *
   * @param pose The pose to which to set the odometry.
   */
  void ResetOdometry(frc::Pose2d pose);

private:
  // Components (e.g. motor controllers and sensors) should generally be
  // declared private and exposed only through public methods.
```

(续下页)

(接上页)

```

  // The motor controllers
  frc::PWMSparkMax m_left1;
  frc::PWMSparkMax m_left2;
  frc::PWMSparkMax m_right1;
  frc::PWMSparkMax m_right2;

  // The robot's drive
  frc::DifferentialDrive m_drive{[&](double output) { m_left1.Set(output); },
                                 [&](double output) { m_right1.Set(output); }};

  // The left-side drive encoder
  frc::Encoder m_leftEncoder;

  // The right-side drive encoder
  frc::Encoder m_rightEncoder;

  // The gyro sensor
  frc::ADXRS450_Gyro m_gyro;

  // Odometry class for tracking robot pose
  frc::DifferentialDriveOdometry m_odometry;
};
```

C++ (Source)

```
#include "subsystems/DriveSubsystem.h"

#include <frc/geometry/Rotation2d.h>
#include <frc/kinematics/DifferentialDriveWheelSpeeds.h>

using namespace DriveConstants;

DriveSubsystem::DriveSubsystem()
    : m_left1{kLeftMotor1Port},
      m_left2{kLeftMotor2Port},
      m_right1{kRightMotor1Port},
      m_right2{kRightMotor2Port},
      m_leftEncoder{kLeftEncoderPorts[0], kLeftEncoderPorts[1]},
      m_rightEncoder{kRightEncoderPorts[0], kRightEncoderPorts[1]},
      m_odometry{m_gyro.GetRotation2d(), units::meter_t{0}, units::meter_t{0}} {
  wpi::SendableRegistry::AddChild(&m_drive, &m_left1);
  wpi::SendableRegistry::AddChild(&m_drive, &m_right1);

  m_left1.AddFollower(m_left2);
  m_right1.AddFollower(m_right2);

  // We need to invert one side of the drivetrain so that positive voltages
  // result in both sides moving forward. Depending on how your robot's
  // gearbox is constructed, you might have to invert the left side instead.
  m_right1.SetInverted(true);

  // Set the distance per pulse for the encoders
  m_leftEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
```

(续下页)

(接上页)

```
  m_rightEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());

  ResetEncoders();
}

void DriveSubsystem::Periodic() {
  // Implementation of subsystem periodic method goes here.
  m_odometry.Update(m_gyro.GetRotation2d(),
                    units::meter_t{m_leftEncoder.GetDistance()},
                    units::meter_t{m_rightEncoder.GetDistance()});
}

void DriveSubsystem::ArcadeDrive(double fwd, double rot) {
  m_drive.ArcadeDrive(fwd, rot);
}

void DriveSubsystem::TankDriveVolts(units::volt_t left, units::volt_t right) {
  m_left1.SetVoltage(left);
  m_right1.SetVoltage(right);
  m_drive.Feed();
}

void DriveSubsystem::ResetEncoders() {
  m_leftEncoder.Reset();
  m_rightEncoder.Reset();
}

double DriveSubsystem::GetAverageEncoderDistance() {
  return (m_leftEncoder.GetDistance() + m_rightEncoder.GetDistance()) / 2.0;
}

frc::Encoder& DriveSubsystem::GetLeftEncoder() {
  return m_leftEncoder;
}

frc::Encoder& DriveSubsystem::GetRightEncoder() {
  return m_rightEncoder;
}

void DriveSubsystem::SetMaxOutput(double maxOutput) {
  m_drive.SetMaxOutput(maxOutput);
}

units::degree_t DriveSubsystem::GetHeading() const {
  return m_gyro.GetRotation2d().Degrees();
}

double DriveSubsystem::GetTurnRate() {
  return -m_gyro.GetRate();
}

frc::Pose2d DriveSubsystem::GetPose() {
  return m_odometry.GetPose();
}

frc::DifferentialDriveWheelSpeeds DriveSubsystem::GetWheelSpeeds() {
  return {units::meters_per_second_t{m_leftEncoder.GetRate()},
```

(续下页)

(接上页)

```
          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);
}
```

Configuring the Drive Encoders

The drive encoders measure the rotation of the wheels on each side of the drive. To properly configure the encoders, we need to specify two things: the ports the encoders are plugged into, and the distance per encoder pulse. Then, we need to write methods allowing access to the encoder values from code that uses the subsystem.

Encoder Ports

The encoder ports are specified in the encoder's constructor, like so:

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]},
```

Encoder Distance per Pulse

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

```
65    m_leftEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
66    m_rightEncoder.setDistancePerPulse(DriveConstants.kEncoderDistancePerPulse);
```

C++ (Source)

```
32  m_leftEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
33  m_rightEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
```

Encoder Accessor Method

To access the values measured by the encoders, we include the following method:

重要: The returned velocities **must** be in meters! Because we configured the distance per pulse on the encoders above, calling `getRate()` will automatically apply the conversion factor from encoder units to meters. If you are not using WPILib's `Encoder` class, you must perform this conversion either through the respective vendor's API or by manually multiplying by a conversion factor.

Java

```
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  }
```

C++ (Source)

```
frc::DifferentialDriveWheelSpeeds DriveSubsystem::GetWheelSpeeds() {
  return {units::meters_per_second_t{m_leftEncoder.GetRate()},
          units::meters_per_second_t{m_rightEncoder.GetRate()}};
}
```

We wrap the measured encoder values in a `DifferentialDriveWheelSpeeds` object for easier integration with the `RamseteCommand` class later on.

Configuring the Gyroscope

The gyroscope measures the rate of change of the robot's heading (which can then be integrated to provide a measurement of the robot's heading relative to when it first turned on). In our example, we use the Analog Devices ADXRS450 FRC Gyro Board, which was included in the kit of parts for several years:

Java

```
  // The gyro sensor
  private final ADXRS450_Gyro m_gyro = new ADXRS450_Gyro();
```

C++ (Header)

```
  // The gyro sensor
  frc::ADXRS450_Gyro m_gyro;
```

Gyroscope Accessor Method

To access the current heading measured by the gyroscope, we include the following method:

Java

```
  /**
   * Returns the heading of the robot.
   *
   * @return the robot's heading in degrees, from -180 to 180
   */
  public double getHeading() {
    return m_gyro.getRotation2d().getDegrees();
  }
```

C++ (Source)

```
76 units::degree_t DriveSubsystem::GetHeading() const {
77   return m_gyro.GetRotation2d().Degrees();
78 }
```

Configuring the Odometry

Now that we have our encoders and gyroscope configured, it is time to set up our drive subsystem to automatically compute its position from the encoder and gyroscope readings.

First, we create a member instance of the `DifferentialDriveOdometry` class:

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}} {
```

Updating the Odometry

The odometry class must be regularly updated to incorporate new readings from the encoder and gyroscope. We accomplish this inside the subsystem's `periodic` method, which is automatically called once per main loop iteration:

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()});
}
```

Odometry Accessor Method

To access the robot's current computed pose, we include the following method:

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();
}
```

重要: 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.

Voltage-Based Drive Method

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

```
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   }
```

C++ (Source)

```
49  void DriveSubsystem::TankDriveVolts(units::volt_t left, units::volt_t right) {
50    m_left1.SetVoltage(left);
51    m_right1.SetVoltage(right);
52    m_drive.Feed();
53  }
```

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.

警告: `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.

Step 4: Creating and Following a Trajectory

With our drive subsystem written, it is now time to generate a trajectory and write an autonomous command to follow it.

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)),
107             // End 3 meters straight ahead of where we started, facing forward
108             new Pose2d(3, 0, new Rotation2d(0)),
109             // Pass config
110             config);
111
112     RamseteCommand ramseteCommand =
113         new RamseteCommand(
114             exampleTrajectory,
115             m_robotDrive::getPose,
116             new RamseteController(AutoConstants.kRamseteB, AutoConstants.
↪kRamseteZeta),
```

(续下页)

(接上页)

```
            new SimpleMotorFeedforward(
                DriveConstants.ksVolts,
                DriveConstants.kvVoltSecondsPerMeter,
                DriveConstants.kaVoltSecondsSquaredPerMeter),
            DriveConstants.kDriveKinematics,
            m_robotDrive::getWheelSpeeds,
            new PIDController(DriveConstants.kPDriveVel, 0, 0),
            new PIDController(DriveConstants.kPDriveVel, 0, 0),
            // RamseteCommand passes volts to the callback
            m_robotDrive::tankDriveVolts,
            m_robotDrive);

    // Reset odometry to the initial pose of the trajectory, run path following
    // command, then stop at the end.
    return Commands.runOnce(() -> m_robotDrive.resetOdometry(exampleTrajectory.
↪getInitialPose()))
        .andThen(ramseteCommand)
        .andThen(Commands.runOnce(() -> m_robotDrive.tankDriveVolts(0, 0)));
  }
}
```

C++ (Source)

```
frc2::CommandPtr RobotContainer::GetAutonomousCommand() {
  // Create a voltage constraint to ensure we don't accelerate too fast
  frc::DifferentialDriveVoltageConstraint autoVoltageConstraint{
      frc::SimpleMotorFeedforward<units::meters>{
          DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
      DriveConstants::kDriveKinematics, 10_V};

  // Set up config for trajectory
  frc::TrajectoryConfig config{AutoConstants::kMaxSpeed,
                               AutoConstants::kMaxAcceleration};
  // Add kinematics to ensure max speed is actually obeyed
  config.SetKinematics(DriveConstants::kDriveKinematics);
  // Apply the voltage constraint
  config.AddConstraint(autoVoltageConstraint);

  // An example trajectory to follow.  All units in meters.
  auto exampleTrajectory = frc::TrajectoryGenerator::GenerateTrajectory(
      // Start at the origin facing the +X direction
      frc::Pose2d{0_m, 0_m, 0_deg},
      // Pass through these two interior waypoints, making an 's' curve path
      {frc::Translation2d{1_m, 1_m}, frc::Translation2d{2_m, -1_m}},
      // 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>{
```

(续下页)

(接上页)

```
          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); }, {}));
}
```

Configuring the Trajectory Constraints

First, we must set some configuration parameters for the trajectory which will ensure that the generated trajectory is followable.

Creating a Voltage Constraint

The first piece of configuration we will need is a voltage constraint. This will ensure that the generated trajectory never commands the robot to go faster than it is capable of achieving with the given voltage supply:

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++ (Source)

```
// Create a voltage constraint to ensure we don't accelerate too fast
frc::DifferentialDriveVoltageConstraint autoVoltageConstraint{
    frc::SimpleMotorFeedforward<units::meters>{
        DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
    DriveConstants::kDriveKinematics, 10_V};
```

Notice that we set the maximum voltage to 10V, rather than the nominal battery voltage of 12V. This gives us some "headroom" to deal with "voltage sag" during operation.

Creating the Configuration

Now that we have our voltage constraint, we can create our `TrajectoryConfig` instance, which wraps together all of our path constraints:

Java

```
  // Create config for trajectory
  TrajectoryConfig config =
      new TrajectoryConfig(
              AutoConstants.kMaxSpeedMetersPerSecond,
              AutoConstants.kMaxAccelerationMetersPerSecondSquared)
          // Add kinematics to ensure max speed is actually obeyed
          .setKinematics(DriveConstants.kDriveKinematics)
          // Apply the voltage constraint
          .addConstraint(autoVoltageConstraint);
```

C++ (Source)

```
// Set up config for trajectory
frc::TrajectoryConfig config{AutoConstants::kMaxSpeed,
                             AutoConstants::kMaxAcceleration};
// Add kinematics to ensure max speed is actually obeyed
config.SetKinematics(DriveConstants::kDriveKinematics);
// Apply the voltage constraint
config.AddConstraint(autoVoltageConstraint);
```

Generating the Trajectory

With our trajectory configuration in hand, we are now ready to generate our trajectory. For this example, we will be generating a "clamped cubic" trajectory - this means we will specify full robot poses at the endpoints, and positions only for interior waypoints (also known as "knot points"). As elsewhere, all distances are in meters.

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++ (Source)

```
  // 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);
```

备注: Instead of generating the trajectory on the roboRIO as outlined above, one can also *import a PathWeaver JSON*.

Creating the RamseteCommand

We will first reset our robot's pose to the starting pose of the trajectory. This ensures that the robot's location on the coordinate system and the trajectory's starting position are the same.

Java

```
    // Reset odometry to the initial pose of the trajectory, run path following
    // command, then stop at the end.
    return Commands.runOnce(() -> m_robotDrive.resetOdometry(exampleTrajectory.
↪getInitialPose()))
```

C++ (Source)

```
  // Reset odometry to the initial pose of the trajectory, run path following
  // command, then stop at the end.
  return frc2::cmd::RunOnce(
```

It is very important that the initial robot pose match the first pose in the trajectory. For the purposes of our example, the robot will be reliably starting at a position of `(0,0)` with a heading of `0`. In actual use, however, it is probably not desirable to base your coordinate system on the robot position, and so the starting position for both the robot and the trajectory should be set to some other value. If you wish to use a trajectory that has been defined in robot-centric coordinates in such a situation, you can transform it to be relative to the robot's current pose using the `transformBy` method (Java, C++). For more information about transforming trajectories, see 转变轨迹.

Now that we have a trajectory, we can create a command that, when executed, will follow that trajectory. To do this, we use the `RamseteCommand` class (Java, C++)

Java

```
    RamseteCommand ramseteCommand =
        new RamseteCommand(
            exampleTrajectory,
            m_robotDrive::getPose,
            new RamseteController(AutoConstants.kRamseteB, AutoConstants.
↪kRamseteZeta),
            new SimpleMotorFeedforward(
                DriveConstants.ksVolts,
                DriveConstants.kvVoltSecondsPerMeter,
                DriveConstants.kaVoltSecondsSquaredPerMeter),
            DriveConstants.kDriveKinematics,
            m_robotDrive::getWheelSpeeds,
            new PIDController(DriveConstants.kPDriveVel, 0, 0),
            new PIDController(DriveConstants.kPDriveVel, 0, 0),
            // RamseteCommand passes volts to the callback
            m_robotDrive::tankDriveVolts,
            m_robotDrive);
```

C++ (Source)

```
  frc2::CommandPtr ramseteCommand{frc2::RamseteCommand(
      exampleTrajectory, [this] { return m_drive.GetPose(); },
      frc::RamseteController{AutoConstants::kRamseteB,
                             AutoConstants::kRamseteZeta},
      frc::SimpleMotorFeedforward<units::meters>{
          DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
      DriveConstants::kDriveKinematics,
      [this] { return m_drive.GetWheelSpeeds(); },
      frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
      frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
      [this](auto left, auto right) { m_drive.TankDriveVolts(left, right); },
      {&m_drive})};
```

This declaration is fairly substantial, so we'll go through it argument-by-argument:

1. The trajectory: This is the trajectory to be followed; accordingly, we pass the command the trajectory we just constructed in our earlier steps.
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. The robot drive: This is the drive subsystem itself, included to ensure the command does not operate on the drive at the same time as any other command that uses the drive.

Finally, note that we append a final "stop" command in sequence after the path-following command, to ensure that the robot stops moving at the end of the trajectory.

Video

If all has gone well, your robot's autonomous routine should look something like this:

29.1.2 PathWeaver

备注: 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.

路径编译器的介绍

备注： 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.

自动阶段是比赛的重要部分。当机器人自主完成令人印象深刻的事情时，大家都会十分兴奋。为了得分，机器人通常需要去某个地方。机器人到达该位置的速度越快，得到分数的速度也就越快。传统的自动驾驶方法是沿直线行驶，转至一定角度，然后再次沿直线行驶。这种方法效果很好，但是机器人在每次直行和转弯之后都要花费大量的时间来停止和重新启动。

自治的一种更高级的，在自动阶段期间执行的方法称为“路径规划”。机器人无需直线行驶并在直线结束后转弯，而是连续运动，以曲线运动的方式行驶。这样可以减少转弯停止时间。

WPILib 包含一个轨迹生成套件，团队可以使用它来生成和跟踪轨迹。本系列文章将介绍如何使用 PathWeaver 生成和可视化轨迹。有关以下轨迹的综合教程，请访问端到端轨迹教程。

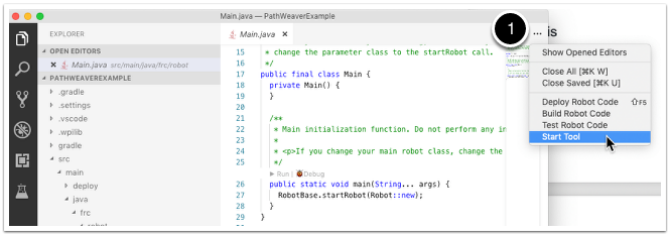
备注： *Trajectory following* 代码是使用 PathWeaver 所必需的。我们建议您从跟踪轨迹开始，然后使用简单的路径进行操作。从那里您可以继续测试由 PathWeaver 生成的更复杂的路径。

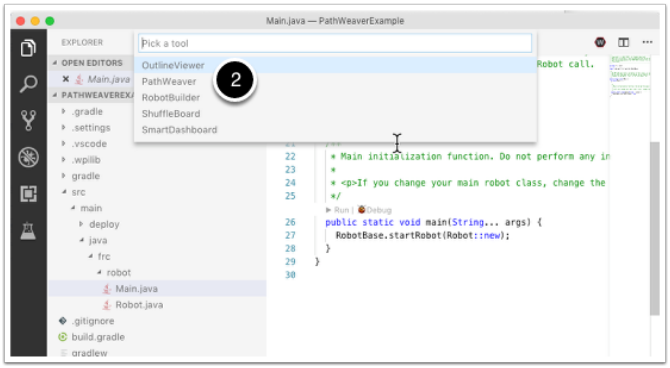
创建一个路径编织项目

路径编织器是用于绘制机器人要遵循的路径的工具。单个程序的路径存储在路径编织器项目中。

开始编织路径

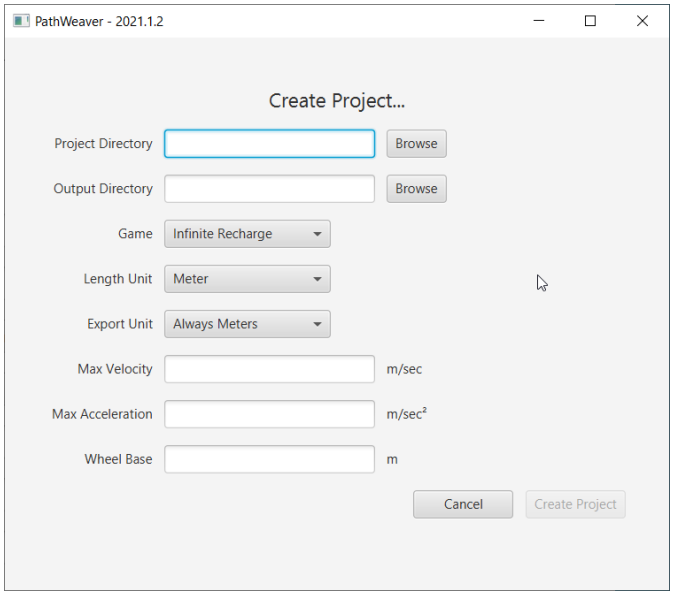
单击 Visual Studio 代码界面右上角的省略号图标可启动路径编织器。您必须从 WPILib 项目中选择一个源文件才能看到该图标。然后单击“启动工具”，然后单击“Path Weaver”，如下所示。





创建项目

要创建路径编织项目，请单击“Create Project”，然后填写项目创建表单。请注意，将鼠标悬停在表单中的任何场地上将显示有关所需内容的更多信息。



关于 **Project Directory:** ：这通常是顶层项目目录，其中包含机器人程序的 build.gradle 和 src 文件。选择此目录是使用路径编织器的预期方式，并且会使它在正确的目录中定位所有输出文件，以将路径自动部署到您的机器人。

关于 **Output directory: **：存储路径以部署到您的机器人的目录。如果您在上一步中（根据建议）指定了我们的机器人项目的顶级项目文件夹，则可以选择填写项目目录。

关于 **Game:**：选择正确的比赛相关信息（正在使用哪个 FRC 比赛）将引用正确的场地图像叠加层。您也可以创建自己的场地图像，该过程将在本系列的后面部分中描述。

关于 **Length Unit:**：使用路径编织器使轨迹可视化时，用于描述机器人和进行现场测量的单位。最好使用米作为单位，因为这是遵循 WPILib 轨迹的单位。

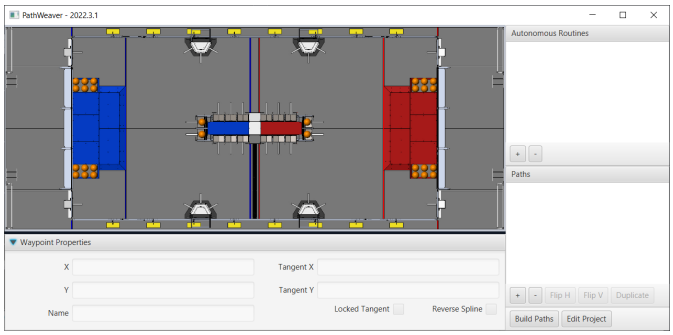
导出单位：导出路径和航点时要使用的单位。如果您打算使用 WPILib 轨迹，则应选择“始终仪表”。选择与项目相同的单位将以与上面的长度单位相同的单位导出。

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.

关于 **Wheel Base:**：这表达的是机器人左右轮之间的距离。这用于确保差速器驱动时任何一个车轮均不会在转弯时超过指定的最大速度。

路径编织器的用户界面

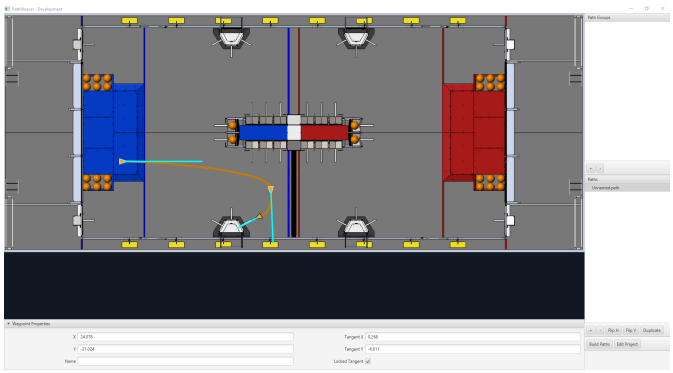


路径编织器的用户界面包含以下内容：

1. 左上角的场地区域，占据了路径编织器窗口的大部分。轨迹将被绘制在此区域上。
2. 当前选定导航点的属性将显示在底部窗格中。这些属性包括 X 坐标和 Y 坐标，以及每个点上所对应的切线。
3. 一组路径（或“自动操作”模式）将显示在窗口的右上方。这是在单个自动模式下查看所有轨迹的便捷方式。
4. 机器人可能遵循的各个路径将显示在窗口的右下方。
5. “Build Path”按钮将以 JSON 格式导出轨迹。机器人代码也可以使用这些 JSON 文件来跟踪轨迹。
6. “Edit Project”按钮允许您编辑项目的属性。

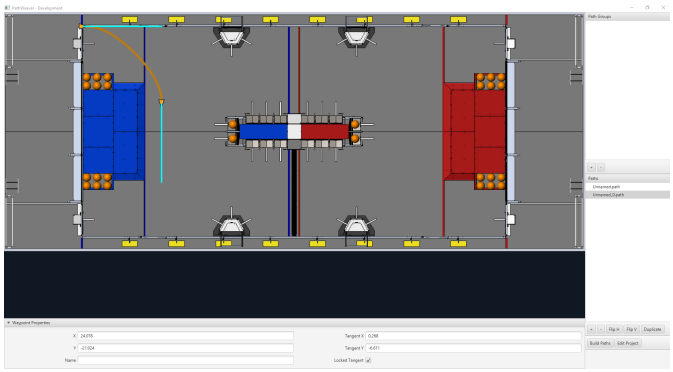
Visualizing PathWeaver Trajectories

PathWeaver’s primary feature is to visualize trajectories. The following images depict a smooth trajectory that represents a trajectory that a robot might take during the autonomous period. Paths can have any number of waypoints that can allow more complex driving to be described. In this case there are 3 waypoints (including the start and stop) depicted with the triangle icons. Each waypoint consists of a X, Y position on the field as well as a robot heading described as the X and Y tangent lines.



Creating the Initial Trajectory

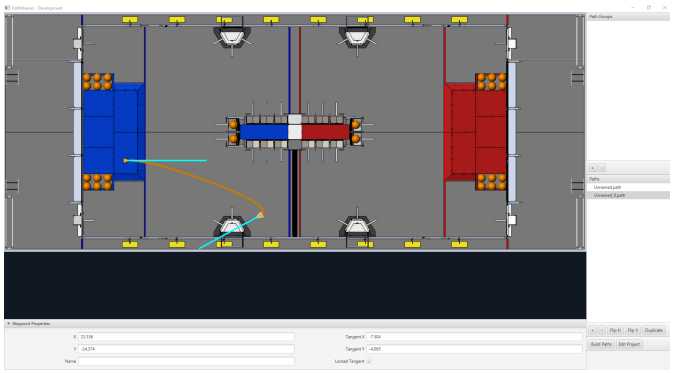
To start creating a trajectory, click the + (plus) button in the path window. A default trajectory will be created that probably does not have the proper start and end points that you desire. The path also shows the tangent vectors (teal lines) for the start and end points. Changing the angle of the tangent vectors changes the shape of the trajectory.



Drag the start and end points of the trajectory to the desired locations. Notice that in this case, the default trajectory does not start in a legal spot for the 2019 game. We can drag the initial waypoint to make the robot start on the HAB.

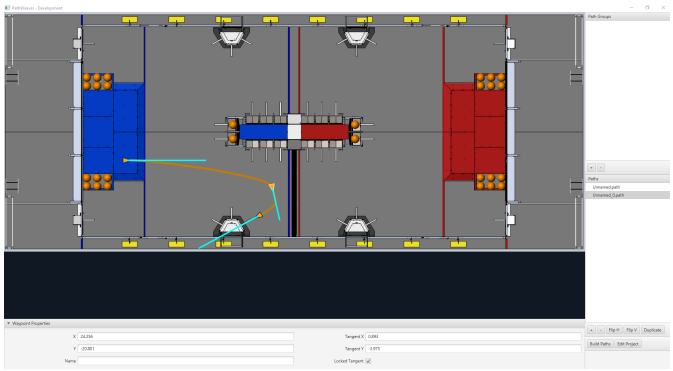
Changing a Waypoint Heading

The robot heading can be changed by dragging the tangent vector (teal) line. Here, the final waypoint was dragged to the desired final pose and was rotated to face the rocket.



Adding Additional Waypoints to Control the Robot Path

Adding additional waypoints and changing their tangent vectors can affect the path that is followed. Additional waypoints can be added by dragging in the middle of the path. In this case, we added another waypoint in the middle of the path.



Locking the Tangent Lines

Locking tangent lines prevents them from changing when the trajectory is being manipulated. The tangent lines will also be locked when the point is moved.

More Precise control of Waypoints

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.

导入路径编织器 JSON

“trajectory yutil” 类可用于将 PathWeaver JSON 导入机器人代码中。本文将介绍导入轨迹。请访问:ref:*end-to-end trajectory tutorial <docs/software/pathplanning/trajectory-tutorial/index:Trajectory Tutorial>* for more information on following the trajectory.

TrajectoryUtil 中的 fromPathweaverJson（Java）/ FromPathweaverJson（C ++）静态方式可用于根据 roboRIO 文件系统上存储的 JSON 文件创建轨迹。

重要: 为了与模拟器 GUI 中的’ ‘Field2d ‘ ‘视图兼容，导出 JSON 的坐标已经更改。之前 (2021 年之前)，y 坐标的范围是从-27 英尺到 0 英尺，而现在，y 坐标的范围是从 0 英尺到 27 英尺 (0 在屏幕底部，27 英尺在屏幕顶部)。这应该不会影响正确的队伍: 裁判: “重新设置他们的里程计到轨迹的起始姿势。<docs/software/pathplanning/trajectory-tutorial/creating-following-trajectory:Creating the RamseteCommand>‘ before path following.

备注: 路径编织者将 JSON 文件放置在 `src/main/deploy/paths` 中，该文件将自动放置在 roboRIO 文件系统上的 `/home/lvuser/deploy/paths` 中，并可以使用 getDeployDirectory 进行访问，如下所示。

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) {
      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());
}
```

在上面的示例中，”YourPath‘” 应该替换为您的路径名。

警告： 从 Java 文件中加载 PathWeaver JSON 可能需要多次循环迭代，所以强烈建议机器人在启动时加载这些路径。

将场地图像加入到路径编织器。

此处是对如何将您个人的场地图像加入路径编织器的指示，此处将使用 2019 年的比赛作为示例。

比赛内容是从 Linux 和 macOS 上的 “~/PathWeaver/Games“或 Windows 上的 “%USERPROFILE%/PathWeaver/Games“目录中加载的。这些文件可以位于关于比赛的特定的子目录中，也可以位于 Games 目录中的 zip 文件中。ZIP 文件必须遵循与比赛内容目录相同的布局; JSON 文件必须位于 ZIP 文件的根目录中 (不能位于子目录中)。

Download the example *FIRST* Destination Deep Space field definition here. Other field definitions are available in the allwpilib GitHub repository.

文件格式

```
~/PathWeaver
  /Games
   /Custom Game
      custom-game.json
      field-image.png
   OtherGame.zip
```

JSON 格式

```
{
   "game": "game name",
   "field-image": "relative/path/to/img.png",
   "field-corners": {
      "top-left": [x, y],
      "bottom-right": [x, y]
   },
   "field-size": [width, length],
```

(续下页)

(接上页)

```
   "field-unit": "unit name"
}
```

场地图像的路径是相对于 JSON 文件的。为简单起见，图像文件应与 JSON 文件位于同一目录中。

场角是 X 坐标与 Y 坐标在场地左上角与右下角的交点，用于定义场地图像中可操作的矩形区域。非矩形的操作区域将不被支持。

场地的尺寸是可操作区域的长和宽，场地的尺寸将以既定的单位给出。

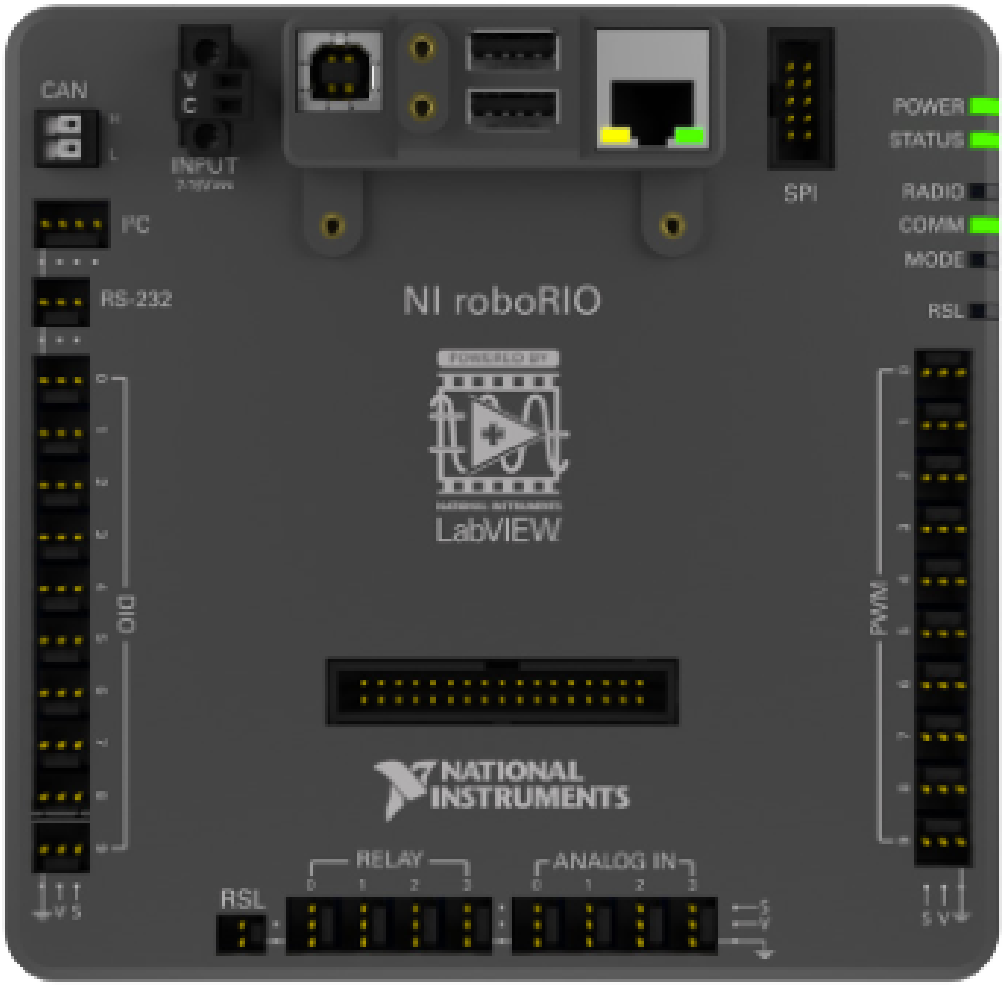
场地单位不区分大小写，可以以米，厘米，毫米，英寸，英尺，码或英里为单位。支持单数，复数和缩写 (例如，“meter”，“meters” 和 “m” 对于指定米均有效)。

备注： 当创建一个新的域图像时，应该在外部留下一个边界 (建议至少 20 像素)，这样域边缘上的路径点就可以访问了。

30

roboRIO 主控板

30.1 roboRIO 简介



roboRIO 在设计时特别考虑了 FIRST。roboRIO 具有实时处理器 + FPGA（现场可编程门阵列）的基本体系结构，但比工业上使用的某些类似系统更强大，更轻巧，更小。

roboRIO 是可重新配置的机器人控制器，包括用于内部集成电路（I2C），串行外围设备接口（SPI），RS232，USB，以太网，脉冲宽度调制（PWM）的内置端口以及用于快速连接公共传感器的继电器以及机器人技术中使用的执行器。该控制器具有 LED，按钮，板载加速度计和自定义电子端口。它具有板载双核 ARM 实时 Cortex-A9 处理器和可定制的 Xilinx FPGA。

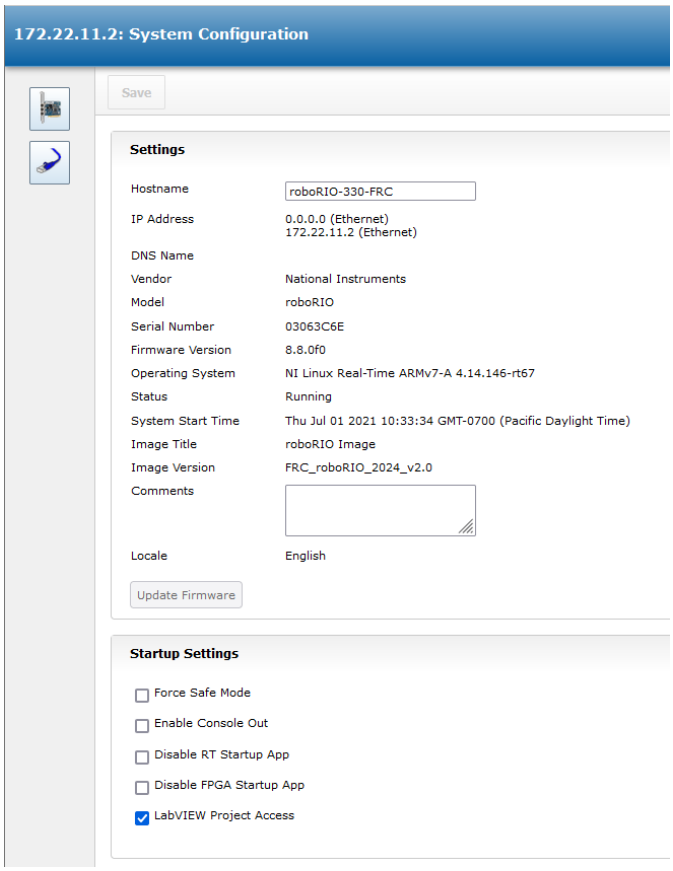
Detailed information on the roboRIO can be found in the roboRIO User Manual and in the roboRIO technical specifications.

Before deploying programs to your roboRIO, you must first image the roboRIO: *roboRIO 1 roboRIO 2*.

30.2 roboRIO 网络仪表板

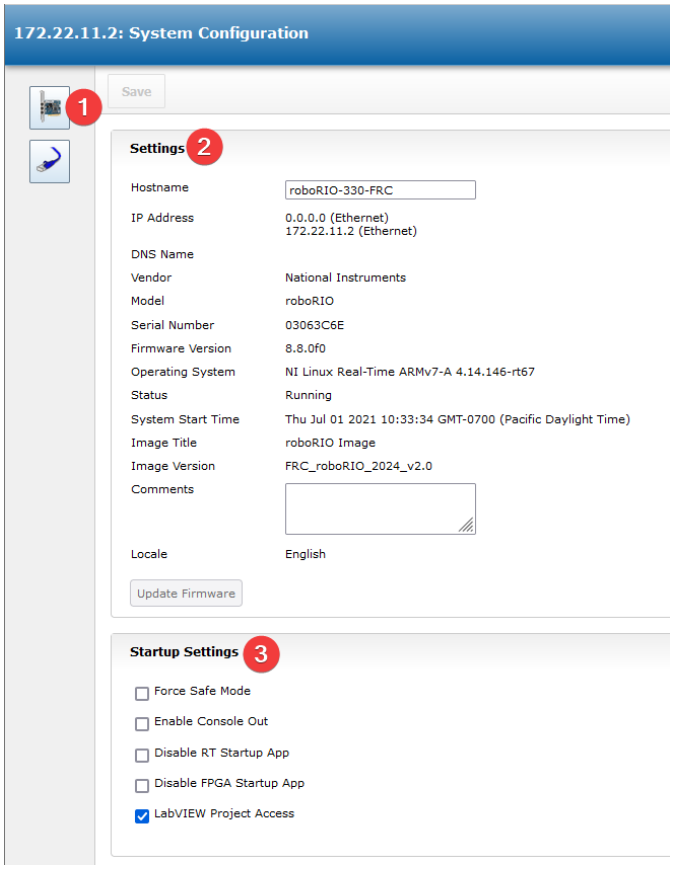
roboRIO 网络仪表板是 roboRIO 内置的网页，可用于检查状态和更新 roboRIO 的设置。

30.2.1 打开网络仪表板



要打开网络仪表板，请打开网络浏览器，然后在地址栏中输入 roboRIO 的地址（使用 USB 连接时，地址为 172.22.11.2，或 “roboRIO-####-FRC.local，其中 #### 是您的团队编号。两个接口都没有前导零）。有关 mDNS 和 roboRIO 网络的详细信息，请参阅此文档：文档/网络/网络引入/ ip 配置：IP 配置

30.2.2 系统配置选项卡



Web 仪表板的主屏幕是“系统配置”选项卡，其中包含 5 个主要部分：

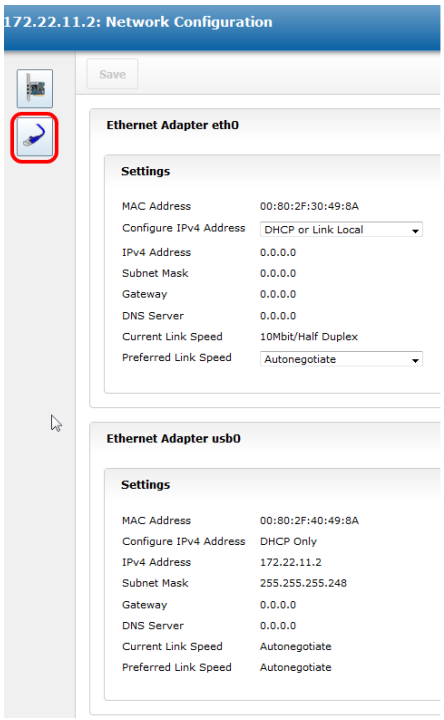
1. 导航栏 - 此部分允许您导航到网络仪表板的不同部分。下面将介绍通过此导航栏可访问的不同页面。
2. 系统设置 - 此部分包含有关系统设置的信息。应使用 roboRIO Imaging 工具根据您的团队编号设置“主机名”字段，不要手动修改。本节包含设备 IP，固件版本和镜像版本等信息。
3. 启动设置 - 此部分包含 roboRIO 的启动设置。这些将在下面的子步骤中描述。
4. 系统资源（未在图片中显示）- 本节提供系统资源的快照，例如内存和 CPU 负载。

启动设置

- 强制安全模式 - 强制控制器进入安全模式。此功能可用于解决镜像问题。相对于这种方法，建议使用 roboRIO 上的“重置”按钮将设备置于安全模式（在已通电的情况下，按住休息按钮 5 秒钟）。默认未选中。

- 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.**
- 禁用 RT 启动应用程序 - 选中此框将禁用启动时运行的代码。如果您发现 roboRIO 对新程序的下载没有响应，则可将其用于故障排除。默认未选中。
- 禁用 FPGA 启动应用程序 - 此框不应被选中。
- LabVIEW Project Access - **It is recommended to leave this box checked.** This setting allows LabVIEW projects to access the roboRIO.

30.2.3 网络配置



此页面显示 roboRIO 的网络适配器的配置。不建议更改该页面的任何数据。有关 roboRIO 网络的更多信息，请参见本文：: ref: *docs / networking / networking-introduction / ip-configurations*: *IP 配置*

30.3 roboRIO FTP

备注: roboRIO 启用了 SFTP 和匿名 FTP。本文介绍如何分别使用这两种方式来访问 roboRIO 的文件系统。

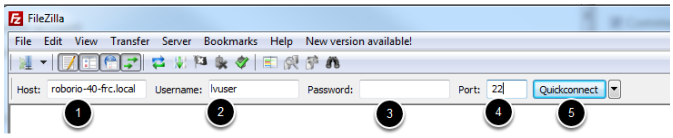
30.3.1 SFTP

推荐使用 SFTP 访问 roboRIO 文件系统。因为您将使用的账户与运行您程序的相同，复制的文件应始终具有与代码兼容的权限。

软件

有许多免费的 SFTP 程序。本文将介绍 FileZilla 的使用。在继续进行操作之前，您可以下载并安装 'FileZilla <https://filezilla-project.org/download.php?type=client>'__，也可以将以下说明推广到您选择的 SFTP 客户端上。

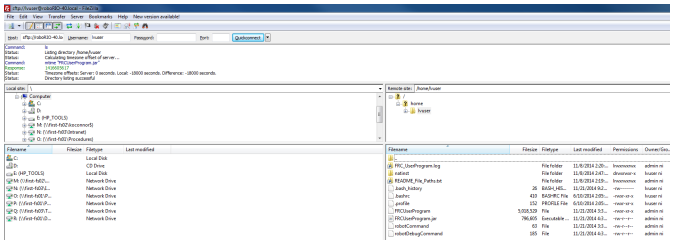
连接到 roboRIO



要连接到 roboRIO：

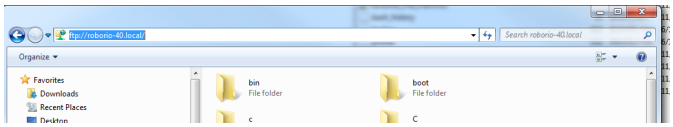
1. 在“主机”框中输入 mDNS 名称（roboRIO-TEAM-frc.local）
2. 在“用户名”框中输入“lvuser”（您的程序在此账户下运行）
3. 将密码框留空
4. 在端口框中输入“22”（SFTP 默认端口）
5. 单击“快速连接”

浏览 roboRIO 文件系统



连接到 roboRIO 后，Filezilla 将打开 homelvuser 目录。右窗格是远程系统（roboRIO），左窗格是本地系统（您的计算机）。每个窗格的顶部显示您正在浏览的当前目录的层次结构，底部窗格显示目录的内容。要传输文件，只需单击该文件并从一侧拖动到另一侧。要在 roboRIO 上创建目录，请右键单击并选择“创建目录”。

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 roboRIO 用户帐户和 SSH

备注： 本文档包含通常的 FRC® 编程中不需要的高级主题。

roboRIO 映像包含许多帐户。本文将重点介绍用于 FRC 的两个帐户，并提供有关其用途的一些详细信息。本文还将介绍如何通过 SSH 连接到 roboRIO。

30.4.1 roboRIO 用户帐户

roboRIO 映像包含许多用户帐户，但其中有两个主要用于 FRC。

admin

“管理员”帐户具有对系统的根权限，可用于操纵系统文件或设置。团队在使用此帐户时应格外小心，因为它可以修改可能损坏 roboRIO 操作系统的设置和文件。该帐户的凭据为：

“用户名：admin’’

密码：

备注： 密码是故意留白的。

lvuser

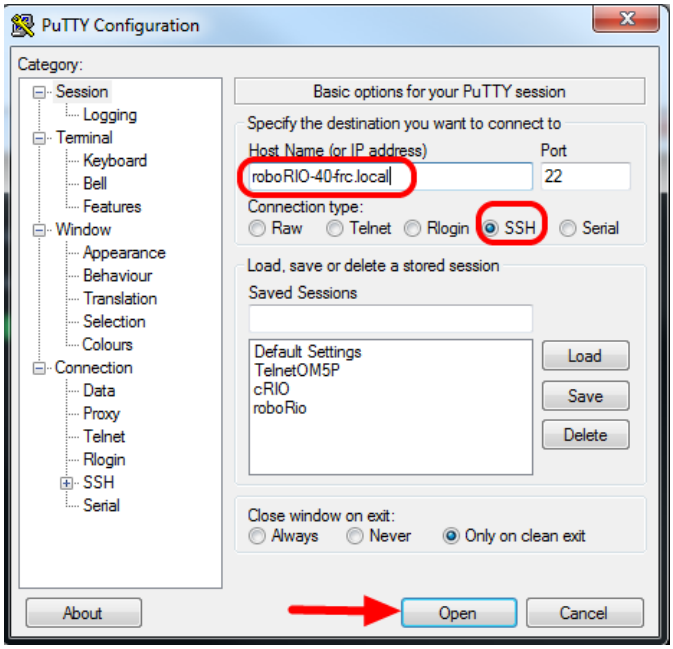
“lvuser”帐户是用于运行所有三种语言写成的用户代码的帐户。此帐户的凭据不应被更改。在开发 roboRIO，团队可以使用该帐户（通过 ssh 或 sftp），以确保对文件或设置所做的任何更改和代码运行都在同一账号下完成。

危险： Changing the default ssh passwords for either “lvuser” or “admin” will prevent C++, Java, and Python teams from uploading code.

30.4.2 SSH 协议

SSH（安全外壳）是用于安全数据通信的协议。当广泛涉及 Linux 系统（例如在 roboRIO 上运行的系统）时，它通常指的是使用 SSH 协议访问命令行控制台。这可用于在远程系统上执行命令。可用于 SSH 的免费客户端是 PuTTY：https：//www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

打开 Putty



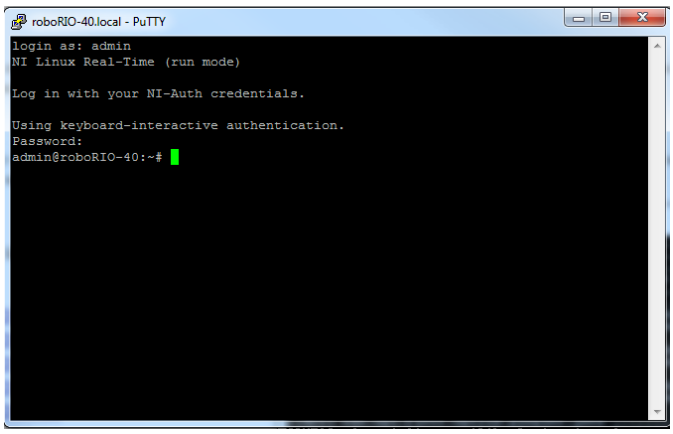
打开 Putty（在任何安全提示下单击 OK）。然后设定以下设置：

1. 主机名：roboRIO-TEAM-frc.local（其中 TEAM 是您的团队编号，示例中的团队编号为 40）
2. 连接类型：SSH

其他设置可以保留为默认值。单击“打开”以打开连接。如果看到有关 SSH 密钥的提示，请单击“确定”。

如果通过 USB 连接，您可以使用 172.22.11.2 作为主机名。如果您的 roboRIO 被设置为静态 IP，您可以通过以太网/无线连接，将该 IP 用作主机名。

登录



当看到提示时，输入用户名（有关说明，请参见上文），然后按 Enter。提示输入密码时，按 Enter（两个帐户的密码均为空白）。

30.5 roboRIO 掉电处理以及了解电能消耗

为了在高电流消耗期间帮助维持电池电压以保护其自身和其他控制系统组件（例如在高电流事件中保护无线电设备），roboRIO 包含分阶段的掉电保护方案。本文介绍了此方案，提供了有关主动计划的系统电流消耗的信息，并介绍了如何使用 PDP 和 DS Log File Viewer 的新功能去了解掉电事件，如果它们真的发生在您的机器人上的话。

30.5.1 roboRIO 掉电保护

roboRIO 使用分阶段的掉电保护方案来尝试稳定自身和其他控制系统组件的输入电压，以防止在大电流消耗的情况下，电池电压拉的过低而导致设备复位。

阶段 1 - 6v 输出下降

触发电压 - 6.8V

When the voltage drops below 6.8V, the 6V output on the *PWM* pins will start to drop.

阶段 2 - 输出禁用

触发电压 - 6.3V

当电压降至 6.3V 以下时，控制器将进入掉电保护状态。当进入该状态时，会有下列指示：

- roboRIO 上的电源 LED 指示灯将变为琥珀色
- Driver Station 上的电压显示背景将变为红色
- Driver Station 上的显示模式将转变为“电压不足”
- DS 的 CAN / Power 选项卡会将 12V 故障计数器加 1。
- DS 将在 DS 日志中记录一次掉电事件。

控制器将采取以下步骤来尝试保持电池电压：

- PWM 输出将被禁用。对于已设置中性值的 PWM 输出（WPILib 中的所有电机控制器），在禁用输出之前，将发送一个中性脉冲。
- 禁用 6V，5V，3.3V 用户轨（包括 PWM 引脚上的 6V 输出，DIO 连接器组中的 5V 引脚，模拟组中的 5V 引脚，SPI 和 I2C 组中的 3.3V 引脚以及 MXP bank 中的 5V 和 3.3V 引脚）
- 配置为输出的 GPIO 进入 High-Z 模式
- 继电器输出被禁用（设为低电平）
- 向基于 CAN 的电机控制器发送明确的禁用指令
- Pneumatic Devices such as the CTRE Pneumatics Control Module and REV Pneumatic Hub are disabled

控制器将保持此状态，直到电压上升至大于 7.5V 或降至触发下一阶段的掉电的电压

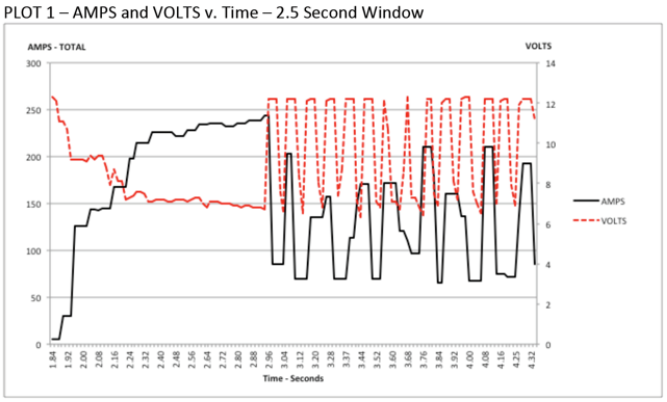
阶段 3-装置停电

**触发电压 - 4.5V **

低于 4.5V 时，设备可能会停电。确切的停电电压可能低于此电压，这取决于设备的负载。

控制器将保持断电状态，直到电压升高到 4.65V 以上。这时设备将按正常顺序启动。

30.5.2 避免掉电 - 主动规划电流使用



避免掉电的关键是主动规划机器人的电流消耗。最好的方法是创建某种形式的功率预算。这种预算可能是相当复杂的时间和对应的预计电流消耗的量化，以便在比赛结束时全面了解电流使用情况和电池状态；也可以是简单地关于电流使用的清单。为创建功率预算：

1. 制定最大的“持续”电流消耗（此处将“持续”宽松地定义为非瞬时的）。这可能是创建功耗预算中最困难的部分。维持 7+ 伏特电压时，电池可以维持的确切电流取决于多种因素，例如电池的健康状况（请参阅用于测量电池的健康状况的此 <docs/hardware/hardware-basics/robot-battery:Robot Battery Basics> 文章）和充电状态。如“NP18-12 数据表 <https://www.farnell.com/datasheets/575631.pdf>”__ 中所示，随着充电状态的降低，尤其是电流消耗的增加，端子电压图变得非常陡峭。该数据表显示，在 3CA 连续负载（54A）下，全新电池可以连续运行 6 分钟以上，同时保持端子电压超过 7V。如上图所示（在“Team 234s 驱动器系统测试”文件 <https://www.chiefdelphi.com/t/paper-new-control-functions-drive-system-testing/139165>‘__ 的允许下使用），即使使用了一块新电池，绘制 240A 的时间也可能超过一两秒。这为我们设定持续电流消耗提供了一些界限。出于本练习的目的，我们将限制设置为 180A。

2. 列出机器人的不同功能，例如动力传动系统，操纵器，主要游戏机制等。
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. 如果您在分析中确定了互斥功能，请考虑在在软件中加强体现。您还可以在机器人程序中使用 PDP 电流监控（下面将详细介绍）来动态设置输出限制或排除（例如，当传动系统电流超过 X 时不要运行机械马达，或者当传动系统电流超过 Y 时仅让马达半功率运行）。

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 识别掉电

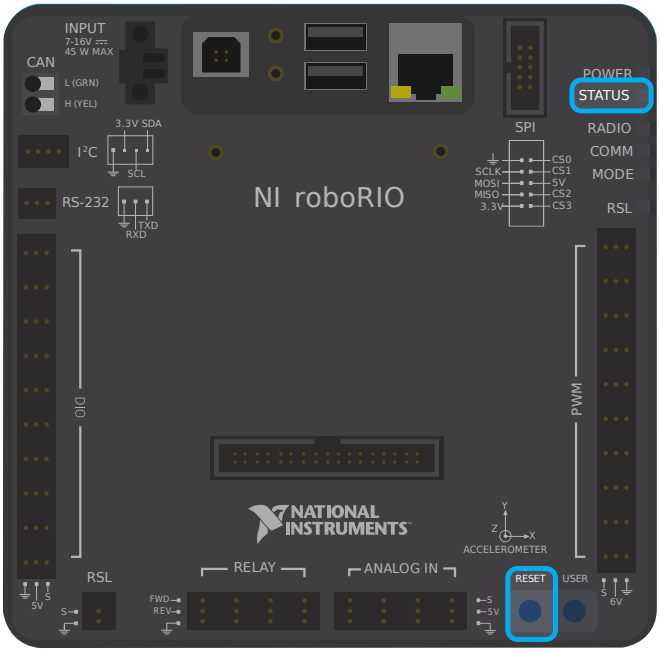
识别掉电的最简单方法是单击 DS 的 CANPower 选项卡，然后检查 12V 故障计数。或者，您可以在发现掉电后使用 Driver Station Log Viewer 查看 Driver Station Log。该日志将用亮橙色线标识掉电情况，如上图所示（请注意，这些掉电是由台式电源引起的，可能无法反映通常发生的 FRC 机器人掉电的持续时间和行为）。

30.6 使用安全模式恢复 roboRIO

有时 roboRIO 可能会损坏，以至于无法使用正常的启动和镜像过程将其恢复。将 roboRIO 引导到安全模式可以使设备成功恢复镜像。

重要： 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 进入安全模式



要将 roboRIO 引导到安全模式：

1. 给 roboRIO 通电
2. 按住重置按钮，直到状态 LED 点亮（约 5 秒钟），然后松开重置按钮
3. roboRIO 将以安全模式启动（状态 LED 以 3 次为一组的形式闪烁指示）

30.6.2 恢复 roboRIO

The roboRIO can now be imaged by using the roboRIO Imaging Tool as described in *Imaging your roboRIO*.

30.6.3 关于安全模式

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

高级 GradleRIO

GradleRIO 是为 roboRIO 部署机器人代码提供动力的机制。GradleRIO 建立在流行的 Gradle 依赖关系和构建管理系统上。本节重点介绍了可用于团队改进其工作流程的“高级”配置。

31.1 通过机器人代码使用外部库

警告： 使用外部库可能会对您的机器人代码产生意想不到的行为！除非您知道自己在做什么，否则不建议这样做！

通常，团队可能希望添加外部 Java 或 C ++ 库，以用于其机器人代码。本文重点介绍将 Java 库添加到您的 Gradle 依赖项或 C ++ 团队具有的选项。

31.1.1 Java

备注： 任何依赖于本机库（JNI）的外部依赖关系都可能无法正常工作。

Java 添加外部依赖项非常简单。您只需添加所需的“存储库’’和“依赖项’’即可。

机器人项目默认情况下在 build.gradle 文件中没有 repository {} 块。您将必须自己添加。在“dependencies {}”块上方，请添加以下内容：

```
repositories {
   mavenCentral()
   ...
}
```

可以将 mavenCentral () 替换为要导入的库使用的任何存储库。现在，您必须添加对库本身的依赖关系。这是通过在“dependencies {}”块中添加必要的行来完成的。以下示例展示了如何将 Apache Commons 添加到您的 Gradle 项目中。

```
dependencies {
   implementation 'org.apache.commons:commons-lang3:3.6'
   ...
}
```

现在，您运行一个构建并确保已下载依赖项。在运行构建之前，Intellisense 可能无法正常工作！

31.1.2 C++

由于需要为 roboRIO 进行编译，因此向您的机器人项目中添加 C ++ 依赖项并非易事。您有两种选择。

1. 将所需库的源代码复制到您的机器人项目中。
2. 以 ‘vendordep 模板 <https://github.com/wpilibsuite/vendor-template>‘__ 为例，并创建一个 vendordep。

复制源代码

只需将必要的源代码和/或标题复制到您的机器人项目中。然后，您可以配置任何必要的平台参数，如下所示：

```
nativeUtils.platformConfigs.named("linuxx86-64").configure {
   it.linker.args.add('-lstdc++fs') // links in C++ filesystem library
}
```

创建一个 Vendordep

请按照 “vendordep 存储库 <https://github.com/wpilibsuite/vendor-template>” __ 中的说明进行操作。

31.2 使用 GitHub Actions 为机器人代码设置 CI

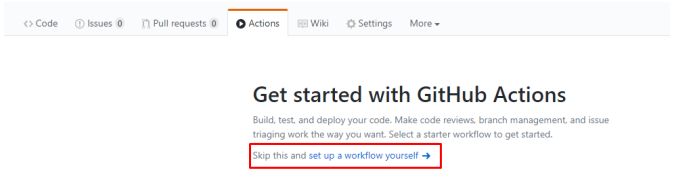
在团队环境中工作的一个重要方面是能够测试推送到像 GitHub 一样的中央存储库的代码。例如，项目经理或首席开发人员可能希望在合并拉取请求之前运行一组单元测试，或者可能想要确保存储库主分支上的所有代码都处于正常工作状态。

GitHub Actions <https://github.com/features/actions> _ 是一项服务，它允许团队和个人在各个分支上的代码以及请求请求上构建和运行单元测试。这些类型的服务通常称为 “连续集成” 服务。本教程将向你展示如何在机器人代码项目上设置 GitHub Actions 。

备注： 本教程假定您团队的机器人代码托管在 GitHub 上。有关 Git 和 GitHub 的简介，请参见：doc：入门指南 *</docs/software/basic-programming/git-getting-started>*。

31.2.1 创建动作

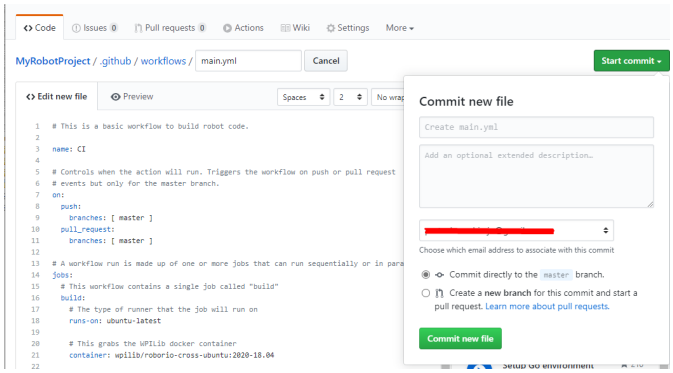
进行 CI 处理的指令存储在 YAML 文件中。为了创建动作，请单击存储库顶部的“操作”选项卡。然后单击“自行设置工作流程”超链接。



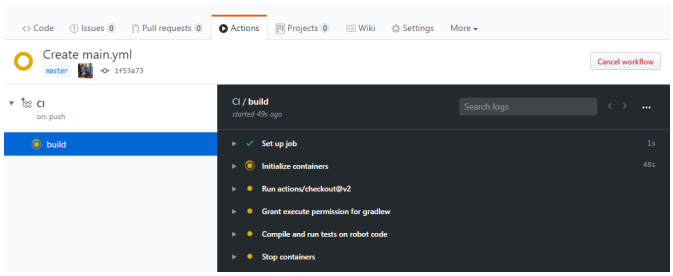
你现在会看到一个文本编辑器。将所有默认文本替换为以下内容：

```yaml
# This is a basic workflow to build robot code.

name: CI

# Controls when the action will run. Triggers the workflow on push or pull request
# events but only for the main branch.
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

# A workflow run is made up of one or more jobs that can run sequentially or in␣
↪parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # This grabs the WPILib docker container
    container: wpilib/roborio-cross-ubuntu:2024-22.04

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
    # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
    - uses: actions/checkout@v4

    # Declares the repository safe and not under dubious ownership.
    - 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
```

然后，通过单击屏幕右上角的“开始提交”按钮来保存更改。如果愿意，您可以修改默认提交消息。然后，单击绿色的“提交新文件”按钮。



现在，无论何时将提交推送到 main 或打开 pull 请求，GitHub 都会自动运行构建。要监视任何构建的状态，你可以单击屏幕顶部的“操作”选项卡。



31.2.2 Actions YAML 文件的细分

以下是 YAML 文件的细分。尽管不需要对每一行都有严格的了解，但是一定程度的了解将帮助您添加更多功能并调试可能出现的潜在问题。

```
# 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 ]
```

此代码块指示操作将在何时运行。当前，该操作将在将提交推送到 main 或针对 main 打开拉取请求时运行。

```
# 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
```

每个 Action 工作流程都由一个或多个作业组成，这些作业 (依次一个接一个) 运行或并行 (同时) 运行。在我们的工作流程中，只有一项“构建”工作。

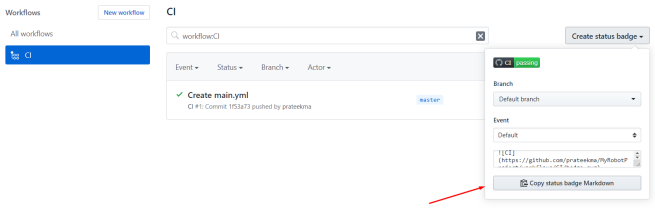
我们指定希望工作在 Ubuntu 虚拟机上和虚拟化的 Docker 容器 <https://www.docker.com/resources/what-container> _ 中运行，该容器包含 JDK，C ++ 编译器和 roboRIO 工具链。

```
# 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
```

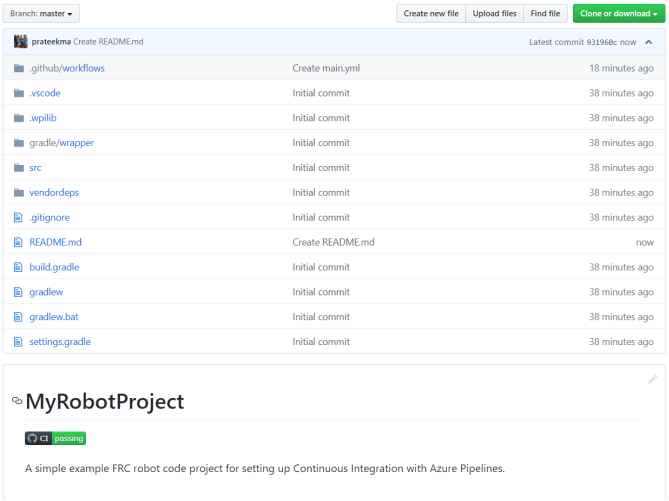
Each job has certain steps that will be executed. This job has four steps. The first step involves checking out the repository to access the robot code. The second step is a workaround for a GitHub Actions issue. The third step involves giving the virtual machine permission to execute gradle tasks using `./gradlew`. The final step runs `./gradlew build` to compile robot code and run any unit tests.

31.2.3 将构建状态标志添加到 README.md 文件

将 CI 状态标记添加到存储库的 README 文件的顶部非常有帮助，以快速检查 main 上最新版本的状态。为此，请单击屏幕顶部的“操作”选项卡，然后选择屏幕左侧的“CI”选项卡。然后，点击右上角的“创建状态标记”按钮，然后复制状态徽章 Markdown 代码。



最后，将复制的 Markdown 代码粘贴到 README 文件的顶部，提交并推送更改。现在，你应该在主存储库页面上看到 GitHub Actions 状态标记。



31.3 使用代码格式化程序

存在代码格式化程序，以确保编写的代码样式在整个代码库中保持一致。这在许多大型项目中都得到过应用。从 Android 到 OpenCV。团队可能希望在整个机器人代码中添加格式化程序，以确保代码库始终保持可读性和一致性。

在本文中，我们将重点介绍 Java 团队使用 Spotpot <https://github.com/diffplug/spotless> __ 和 C ++ 团队使用 wpiformat <https://github.com/wpilibsuite/styleguide/blob/main/wpiformat/README.rst>__。

31.3.1 一尘不染

配置

为了实现 Spotless 功能，需要进行必要的 build.gradle 更改。在 build.gradle 的 plugins {} 块中，添加 Spotless 插件，使其看起来类似于以下内容。

```
plugins {
   id "java"
   id "edu.wpi.first.GradleRIO" version "2024.1.1"
   id 'com.diffplug.spotless' version '6.20.0'
}
```

然后确保您添加了一个必需的 “spotless {}’’ 块以正确配置一尘不染。这可以放在 build.gradle 的末尾。

```
spotless {
   java {
      target fileTree('.') {
            include '**/*.java'
            exclude '**/build/**', '**/build-*/**'
      }
      toggleOffOn()
      googleJavaFormat()
      removeUnusedImports()
      trimTrailingWhitespace()
```

(续下页)

(接上页)

```
      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()
   }
}
```

Running Spotless

可以使用../gradlew spotlessApply 来运行 Spotless，这将应用所有格式选项。您还可以仅通过添加格式化程序的名称来指定特定任务。一个示例是 "./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
```

(续下页)

(接上页)

```
      - uses: actions/setup-java@v4
        with:
          distribution: 'zulu'
          java-version: 17
      - run: ./gradlew spotlessCheck
```

选项说明

每个“格式’’部分突出显示项目中自定义文件的格式。Java 和 GroovyGradle 本身受 Spotless 的支持，因此它们的定义有所不同。

分解这一点，我们可以将其分为多个部分。

- 格式化 Java
- 格式化 Gradle 文件
- 格式化 XML 文件
- 格式化其他文件

它们都是相似的，除了一些小的差异将被解释。下面的示例将突出显示“java {}”块。

```
java {
   target fileTree('.') {
      include '**/*.java'
      exclude '**/build/**', '**/build-*/**'
   }
   toggleOffOn()
   googleJavaFormat()
   removeUnusedImports()
   trimTrailingWhitespace()
   endWithNewline()
}
```

让我们解释每个选项的含义。

```
target fileTree('.') {
   include '**/*.java'
   exclude '**/build/**', '**/build-*/**'
}
```

上面的例子告诉我们 Java 类在哪里，并排除了 build 目录。其余选项都不言自明。

- toggleOffOn () 增加了使项目 Spotless 的能力。用法如下所示

```
// format:off

public void myWeirdFunction() {

}

// format:on
```

- “googleJavaFormat () “告诉 Spotless 地根据 ‘Google 样式指南 <https://google.github.io/styleguide/javaguide.html>‘__
- `removeUnusedImports()` will remove any unused imports from any of your Java classes

- “trimTrailingWhitespace（）“将删除行尾的所有多余空格
- “endWithNewline（）“将在课程末尾添加换行符

在 groovyGradle 块中，有一个 Greclipse 选项。这是一尘不染的格式程序，用于格式化 gradle 文件。

另外，在 xml 块中有一个“eclipseWtp”选项。这代表“Gradle Web 工具平台”，并且是格式化“xml”文件的格式化程序。不使用任何 XML 文件的团队可能希望不包括此配置。

备注： 完整的配置列表位于‘Spotless README <https://github.com/diffplug/spotless>‘__

线尾问题

Spotless 将尝试在每个 OS 上应用行尾，这意味着如果两个用户使用不同的 OS（Unix 与 Windows），则 Git 差异将不断变化。建议从多个操作系统向同一存储库做出贡献的团队使用.gitattributes 文件。以下内容足以处理行尾。

```
*.gradle text eol=lf
*.java text eol=lf
*.md text eol=lf
*.xml text eol=lf
```

31.3.2 WPI 格式

要求

- **‘Python 3.6 或更高版本 <https://www.python.org/>‘__**

您可以通过在终端或命令提示符中键入“pip3 install wpiformat”来安装‘wpiformat <https://github.com/wpilibsuite/styleguide/blob/main/wpiformat/README.rst>‘__。

用法

可以通过在控制台中键入“wpiformat’’来运行 wpiformat。这将使用“clang-format’’格式化。需要三个配置文件（“.clang-format”，“。styleguide”，“。styleguide-license”）。这些必须存在于项目根目录中。

- “.clang-format’’::download：下载 *<https://raw.githubusercontent.com/wpilibsuite/allwpilib/main/.clang-format>*
- “.styleguide-license’’::download：下载 *<https://raw.githubusercontent.com/wpilibsuite/allwpilib/main/.styleguide-license>*

样式指南示例如下所示：

```
cppHeaderFileInclude {
   \.h$
   \.hpp$
   \.inc$
   \.inl$
}
```

(续下页)

(接上页)

```
cppSrcFileInclude {
   \.cpp$
}

modifiableFileExclude {
   gradle/
}
```

备注： 团队可以根据需要调整.styleguide 和.styleguide-license。重要的是不要删除它们，因为它们是运行 wpiformat 所必需的！

You can turn this into a *CI check* by running `git --no-pager diff --exit-code HEAD`, as shown in the example GitHub Actions workflow below:

```yaml
name: Lint and Format

on:
  pull_request:
  push:
jobs:
  wpiformat:
    name: "wpiformat"
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: Fetch all history and metadata
        run: |
          git checkout -b pr
          git branch -f main origin/main
      - name: Set up Python 3.8
        uses: actions/setup-python@v5
        with:
          python-version: 3.8
      - name: Install wpiformat
        run: pip3 install wpiformat==2024.32
      - name: Run
        run: wpiformat
      - name: Check output
        run: git --no-pager diff --exit-code HEAD
```

31.4 Gradlew Tasks

This article aims to highlight the gradle commands supported by the WPILib team for user use. These commands can be viewed by typing `./gradlew tasks` at the root of your robot project. Not all commands shown in `./gradlew tasks` and unsupported commands will not be documented here.

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 第三方库

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.

备注： For Python teams, Git metadata is always copied to your robot during the deploy process. You can use `wpilib.deployinfo.getDeployData()` to retrieve the stored information.

31.5.1 Installing gversion

To install gversion add the following line to the plugins block of `build.gradle`. This tells Gradle to use gversion in the project.

```
plugins {
    // ...
    id "com.peterabeles.gversion" version "1.10"
}
```

In order to generate the file when building the project, add the following block to the bottom of `build.gradle`.

```
project.compileJava.dependsOn(createVersionFile)
gversion {
  srcDir       = "src/main/java/"
  classPackage = "frc.robot"
  className    = "BuildConstants"
  dateFormat   = "yyyy-MM-dd HH:mm:ss z"
  timeZone     = "America/New_York" // Use preferred time zone
  indent       = "  "
}
```

The `srcDir`, `classPackage`, and `className` parameters tell the plugin where to put the manifest file, and what to name it. The `timeZone` field can be set to your team's time zone based on this list of timezone IDs. The `dateFormat` parameter is based on this Java class.

To test this, run a build of your project through the WPILib menu in the top right. Once the code has finished building, there should be a file called `BuildConstants.java` in your `src/main/java` folder. The following is an example of what this file should look like:

```
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";
```

(续下页)

(接上页)

```
  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.

> **警告:** 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
```

(续下页)

(接上页)

```
            // want to store a team number in this file.
            team = project.frc.getTeamNumber()
            debug = project.frc.getDebugOrDefault(false)

            artifacts {
                // First part is artifact name, 2nd is artifact type
                // getTargetTypeClass is a shortcut to get the class type using a string

                frcJava(getArtifactTypeClass('FRCJavaArtifact')) {
                }

                // Static files artifact
                frcStaticFileDeploy(getArtifactTypeClass('FileTreeArtifact')) {
                    files = project.fileTree('src/main/deploy')
                    directory = '/home/lvuser/deploy'
                }
            }
        }
    }
}
```

We will be replacing the highlighted lines with:

```
frcJava(getArtifactTypeClass('FRCJavaArtifact')) {
   // Enable VisualVM connection
   jvmArgs.add("-Dcom.sun.management.jmxremote=true")
   jvmArgs.add("-Dcom.sun.management.jmxremote.port=1198")
   jvmArgs.add("-Dcom.sun.management.jmxremote.local.only=false")
   jvmArgs.add("-Dcom.sun.management.jmxremote.ssl=false")
   jvmArgs.add("-Dcom.sun.management.jmxremote.authenticate=false")
   jvmArgs.add("-Djava.rmi.server.hostname=10.XX.XX.2") // Replace XX.XX with team number
}
```

We are adding a few arguments here. In order:

- Enable remote debugging
- Set the remote debugging port to 1198
- Allow listening from remote targets
- Disable SSL authentication being required
- Set the hostname to the roboRIOs team number. Be sure to replace this.

重要: The hostname when connected via USB-B should be `172.22.11.2`.

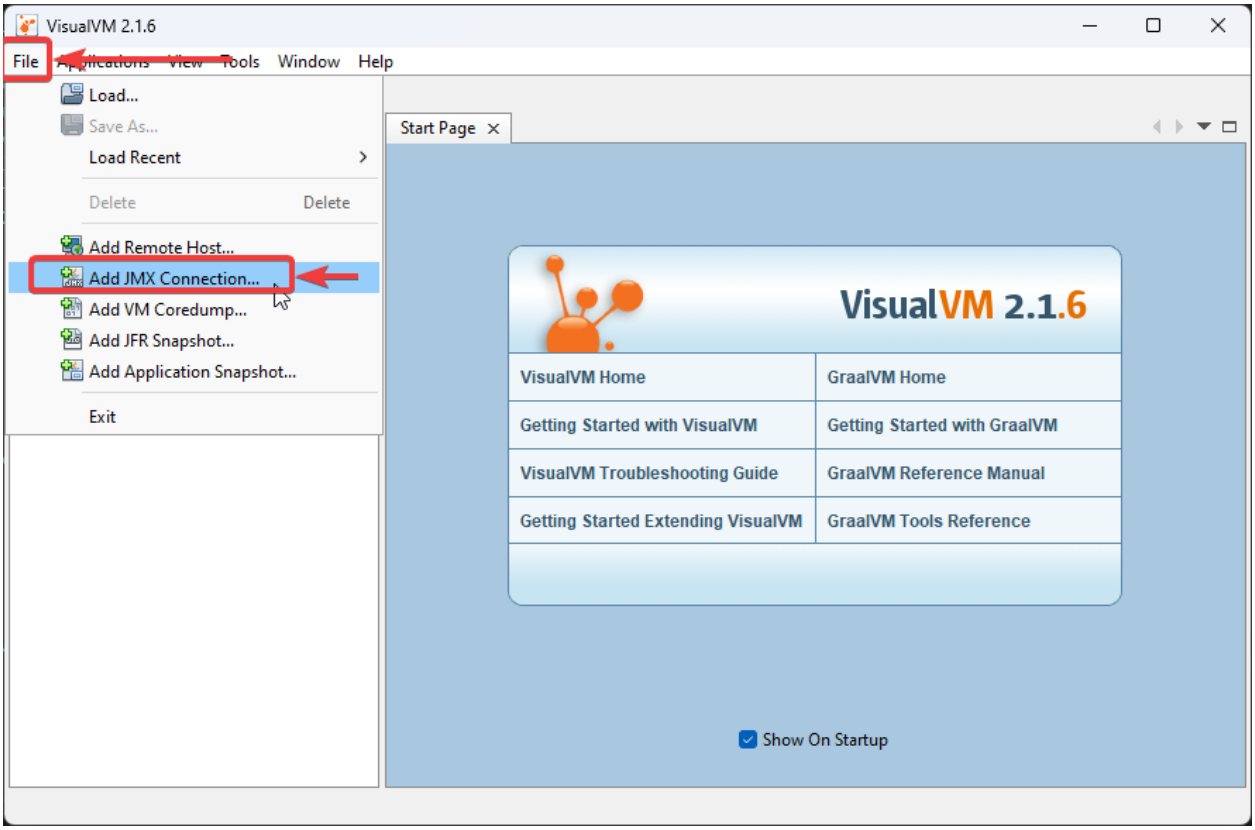
31.7.3 Running VisualVM

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:

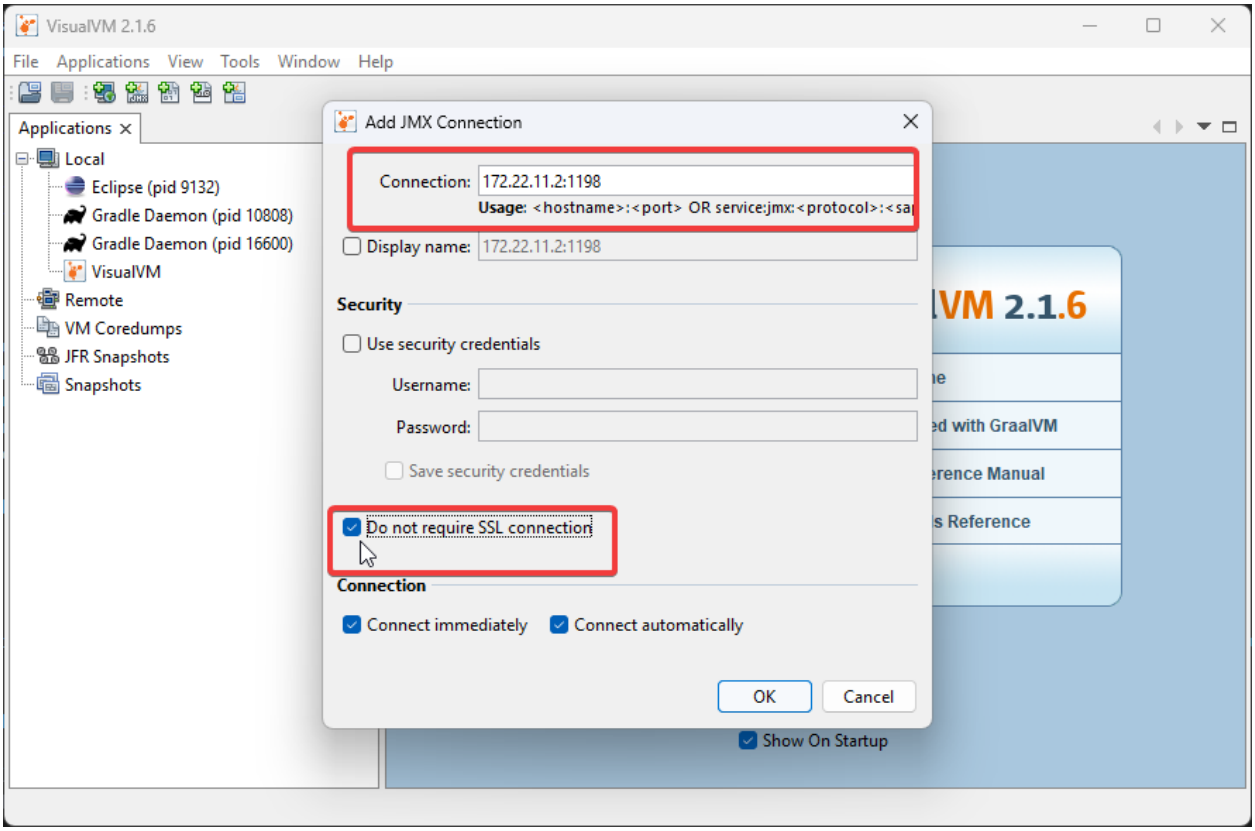
```
cd "C:\Users\Public\wpilib\visualvm_217\bin"
./visualvm --jdkhome "C:\Users\Public\wpilib\2024\jdk"
```

重要： The exact path `visualvm_217` may vary and depends on the version of VisualVM downloaded.

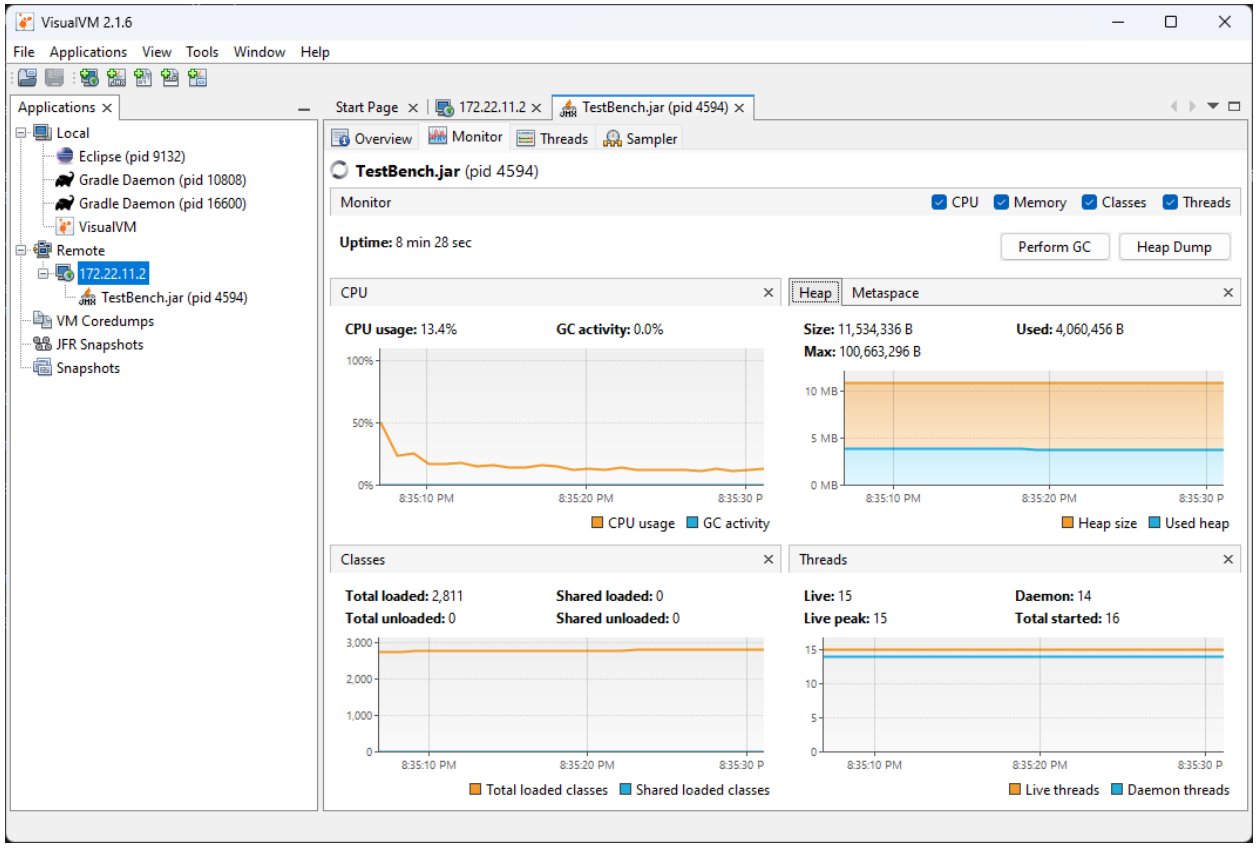
This should launch VisualVM. Once launched, open the *Add JMX Connection* dialog.



Once opened, configure the connection details and hostname. Ensure that *Do not require SSL connection* is ticked.

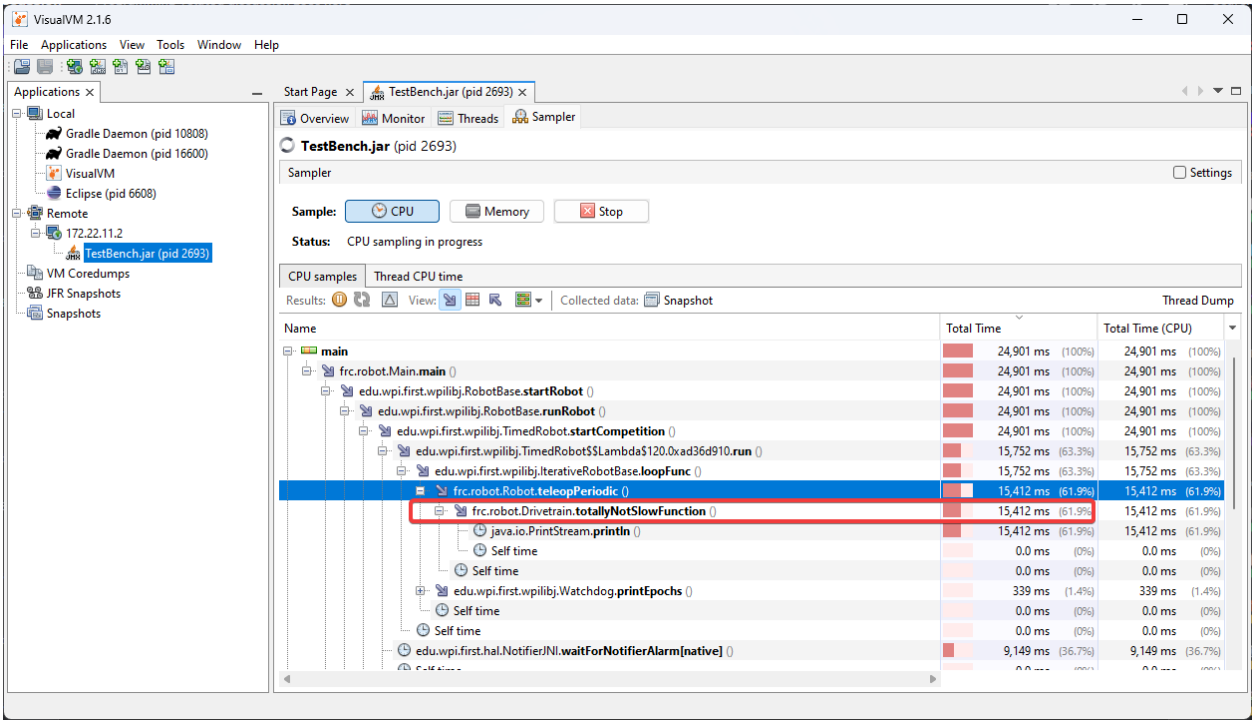


If correctly done, a new menu option in the left-hand sidebar will appear. Clicking on it will show you a detailed dashboard of the running JVM application.



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.

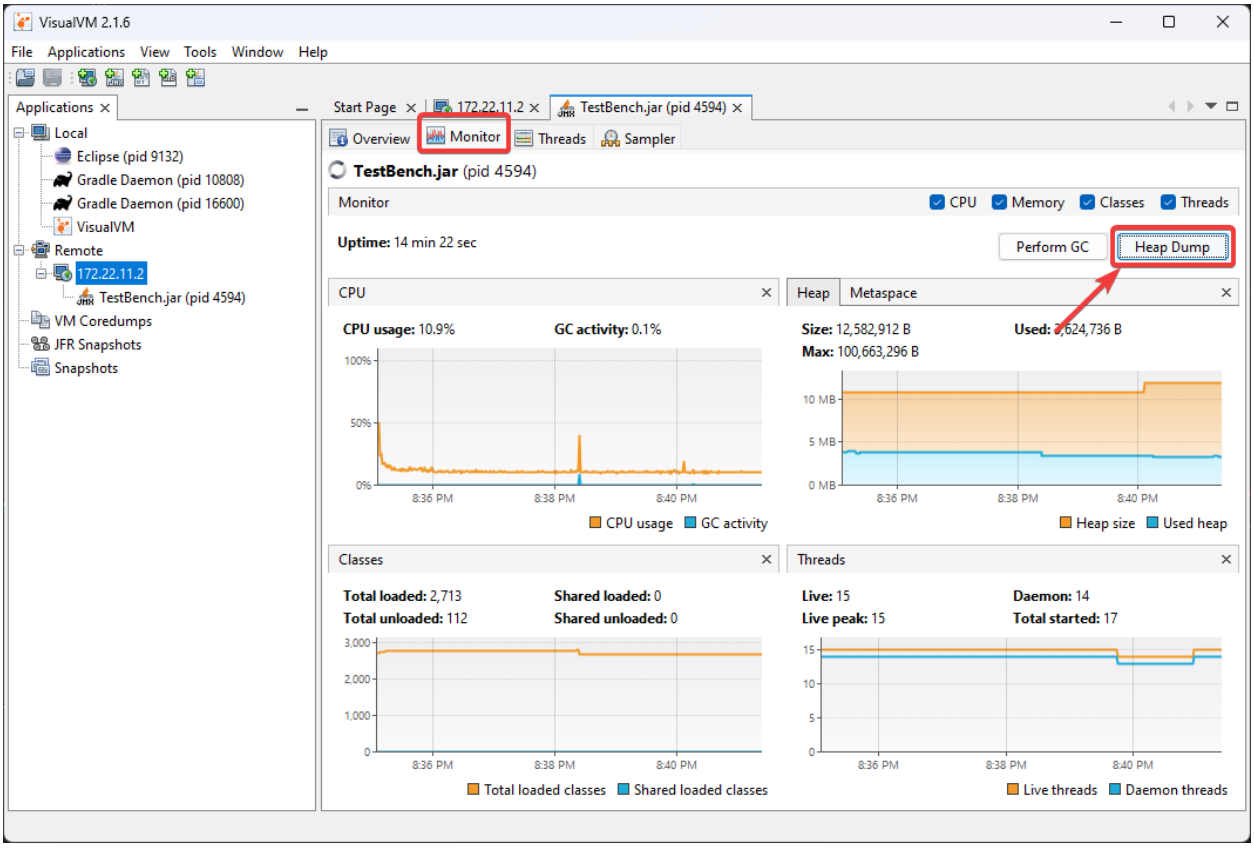
```
public static void totallyNotSlowFunction() {
   for (int i = 0; i < 2000; i++) {
      System.out.println("HAHAHAHA");
   }
}
```

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*.



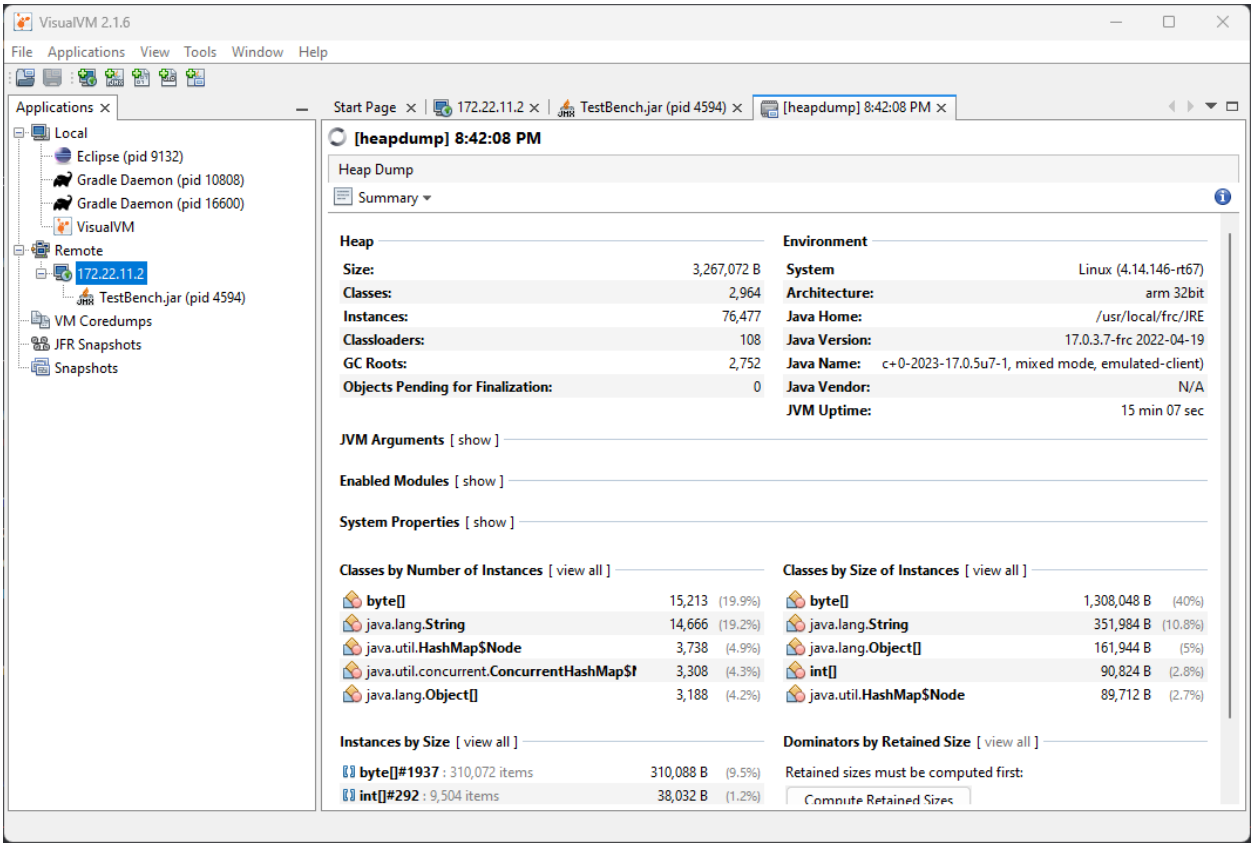
This heap dump will be stored on the target system (roboRIO) and must be retrieved using SFTP. See *this article* for information on retrieving the dump from the roboRIO.

Once downloaded, the dump can be analyzed with VisualVM.

小技巧： You can also *configure the JVM to take a heap dump automatically when your robot code runs out of memory*.

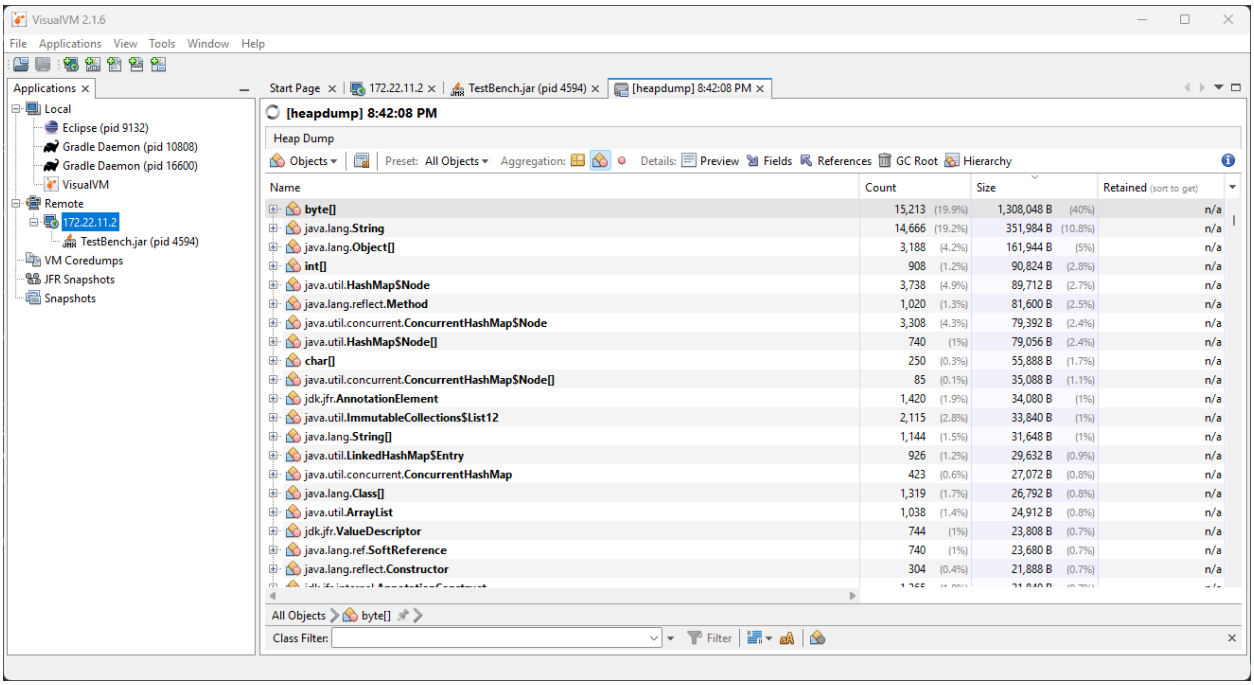
31.7.6 Analyzing a Heap Dump

Reopen VisualVM if closed using the previous instructions. Then click on *File* and *Load*. Navigate to the retrieved dump file and load it.

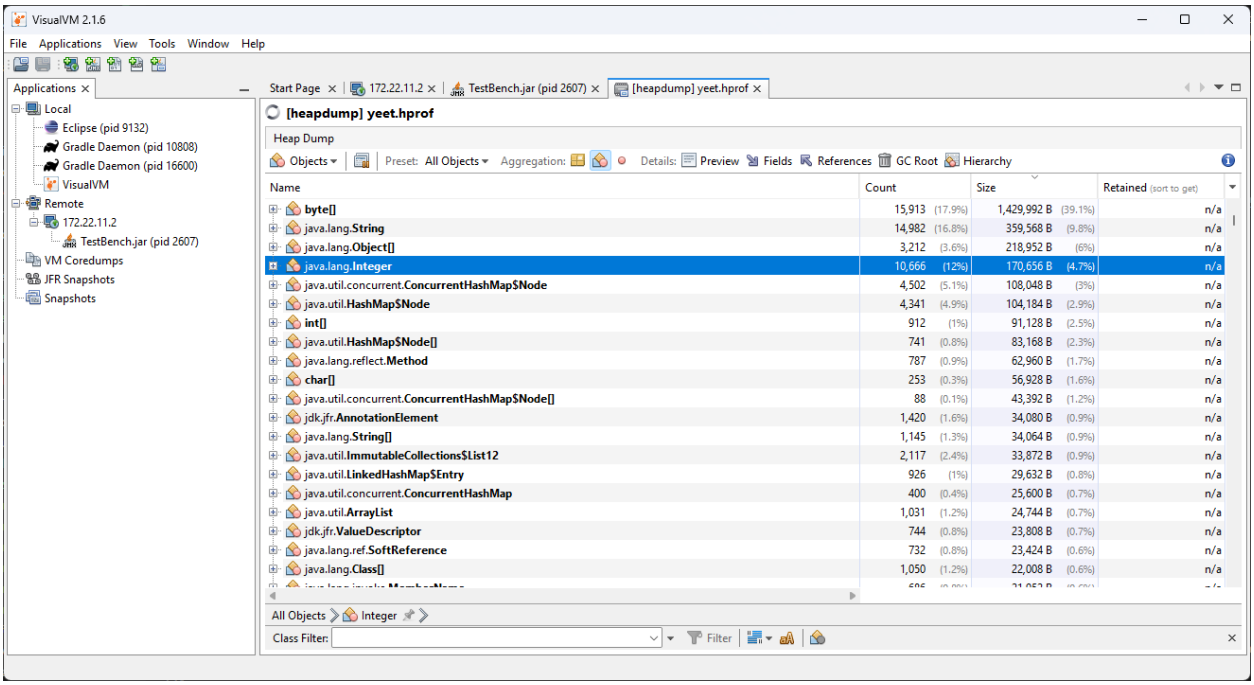


Clicking on *Summary* and selecting *Objects* instead will show a breakdown of objects by quantity. The below screenshot showcases a completely empty robot program, and then one that creates an million large `ArrayList` of integers.

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 进阶控制

本节介绍 WPILib 中的高级控制功能，例如各种反馈/前馈控制算法和轨迹跟踪。

32.1 基于视频的 FRC 自主验证视频演练

在 2020 年的 “WPI 主办的 RSN 春季会议” 上，WPILib 团队的 Tyler Veness 作了关于 FRC | reg |中基于模型的自治验证的演讲。

The link to the presentation is available here.

32.2 高级控制介绍

32.2.1 控制系统基础

备注： This article includes sections of Controls Engineering in FRC by Tyler Veness with permission.

The Need for Control Systems

控制系统无处不在，我们每天都与之互动。您可能会看到一小部分，包括带有恒温器的加热器和空调，巡航控制和汽车上的防抱死制动系统（ABS），以及现代笔记本电脑上的风扇速度调节。控制系统监视或控制此类系统的行为，并且可以由直接控制它们的人（手动控制）或仅由机器（自动控制）组成。

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.

例如，在存在不确定性的情况下，我们如何证明自动驾驶汽车上的闭环控制器能够安全运行并满足期望的性能规格？控制理论是一种代数和几何的应用，用来分析和预测系统的行为，使它们按照我们的要求做出反应，并使它们对干扰和不确定性具有鲁棒性。

简单来说，控制工程是应用于控制理论的工程过程。因此，它不仅仅是应用数学。虽然控制理论背后有一些漂亮的数学运算，但是控制工程学是一门工程学，就像其他任何取舍一样。控制理论给出的解决方案应始终通过我们的性能规格进行检查和了解。我们不需要完美；我们只需要能满足我们的规格即可。

命名法

大多数高级工程主题的资源都需要具备一定的知识水平。部分问题在于术语的使用。虽然它能有效地与业内人士交流想法，但不熟悉它的新人却会迷失方向。

由控制系统控制的执行器系统或集合称为“工厂”。控制器用于将设备从其当前状态驱动到某些所需状态（参考）。不包含开环控制器，即从工厂输出中测得的信息的控制器

从工厂的输出反馈信息的控制器称为闭环控制器或反馈控制器。

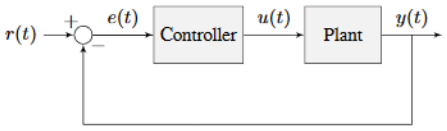


Figure 1.1: Control system nomenclature for a closed-loop system

$r(t)$	reference	$u(t)$	control input
$e(t)$	error	$y(t)$	output

备注： 从工厂的角度定义系统的输入和输出。所示的负反馈控制器将基准与输出之间的差（也称为误差）驱动为零。

什么是增益？

*增益 * 是一个比例值，表示稳态时输入信号的大小与输出信号的大小之间的关系。许多系统包含一种方法，通过该方法可以更改增益，从而为系统提供或多或少的“功率”。

下图显示了一个具有假设输入和输出的系统。由于输出是输入幅度的两倍，因此系统的增益为 2。

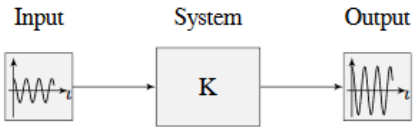


Figure 1.2: Demonstration of system with a gain of $K = 2$

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.

方框图

在设计或分析控制系统时，以图形方式对建模很有用。框图用于此目的。它们可以被系统地操纵和简化。

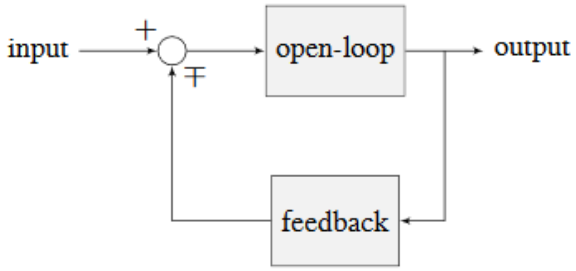


Figure 1.3: Block diagram with nomenclature

开环增益是从输入（圆）处的求和节点到输出分支的总增益。如果断开反馈环路，这将是系统的增益。反馈增益是从输出返回到输入求和节点的总增益。总和节点的输出是其输入的总和。

下图是在反馈配置中具有更正式表示法的框图。

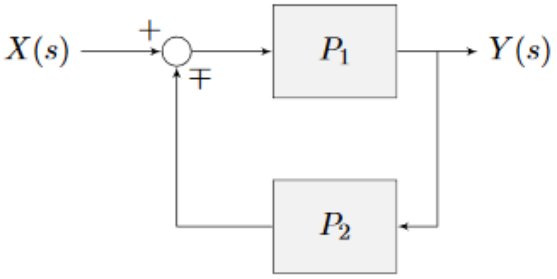


Figure 1.4: Feedback block diagram

：math：'mp'表示“减号或加号”，其中减号表示负反馈。

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

备注: 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:

备注: 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

备注: For a guide on implementing PID control in code with WPILib, see *WPILib 中的前馈控制*.

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 PID 简介

备注: For a guide on implementing PID control with WPILib, see *WPILib 中的 PID 控制*.

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.

备注: 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.

比例项

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*.

$$u[t] = K_p e[t]$$

其中 K_p 是成比例增益，$e(t)$ 是当前时间:math:*t*. 的误差。

下图显示了由 P 控制器控制的：term：‘system 的框图。

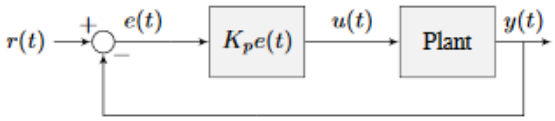


Figure 2.1: P controller block diagram

比例增益就像“软件定义的弹簧”一样，将：system‘拉向所需位置。从物理学中回想起，我们将弹簧建模为:math:‘F = - kx，其中:math:*F‘是施加的力，:math:‘k‘是比例常数，:math:‘x* 是位移。从平衡点开始这可以用另一种方式写成：$F = k(0 - x)$ 其中:math:*0* 是平衡点。如果我们将平衡点设为反馈控制器的：term：*setpoint*，则方程具有一对一的对应关系。

$$F = k(r - x)$$
$$u(t) = K_p e(t) = K_p(r(t) - y(t))$$

因此，比例控制器将 system’ s 1 output‘拉向:term:‘setpoint 的“力”与:term:‘error‘成正比，就像弹簧一样。

导数项

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*.

$$u(t) = K_p e(t) + K_d \frac{de}{dt}$$

其中 K_p‘□□□□□□ : $math$: ‘K_d‘□□□□□□ : $math$: ‘$e(t)$ 是当前时间:math:‘t‘的误差。

下图显示了由 PD 控制器控制的：term：‘system 的框图。

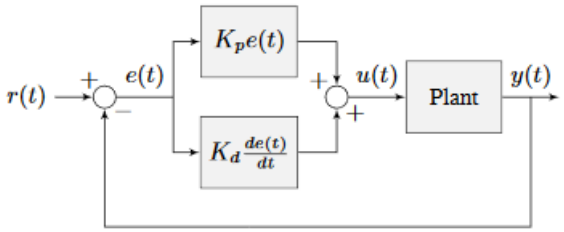


Figure 2.2: PD controller block diagram

PD 控制器具有用于位置的比例控制器 (K_p) 和用于速度的比例控制器 (K_d)。速度:term:*setpoint*‘是由位置:*term:‘setpoint* 随时间变化的方式隐式提供的。为了证明这一点，我们将重新排列 PD 控制器的公式。

$$u_k = K_p e_k + K_d \frac{e_k - e_{k-1}}{dt}$$

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.

$$\begin{aligned}
u_k &= K_p(r_k - x_k) + K_d \frac{(r_k - x_k) - (r_{k-1} - x_{k-1})}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{r_k - x_k - r_{k-1} + x_{k-1}}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{r_k - r_{k-1} - x_k + x_{k-1}}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{(r_k - r_{k-1}) - (x_k - x_{k-1})}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \left(\frac{r_k - r_{k-1}}{dt} - \frac{x_k - x_{k-1}}{dt} \right)
\end{aligned}$$

注意:math:*frac{r_k - r_{k-1}}{dt}*‘是：*term*：*‘setpoint*‘的速度的方式。出于相同的原因，*:math:*frac{x_k - x_{k-1}}{dt}‘ 是 system’ s 在给定时间步长下的速度。这意味着 PD 控制器的:math:‘K_d‘项将估计的速度驱动到:term:‘setpoint 速度。

如果：term：‘setpoint‘是常数，则隐式速度：term：‘setpoint‘为零，因此：math：‘K_d‘项在移动时会降低：term：‘system 的速度。这就像一个“软件定义的阻尼器”。这些通常在闭门器上看到，它们的阻尼力随速度线性增加。

积分项

重要： 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.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau$$

其中：math：*K_p‘是比例增益，：math：‘K_i‘是积分增益，：math：‘e（t）‘是当前时间:math:‘t‘的误差‘和:math:*tau‘ 是集成变量。

积分从时间:math:*0*‘到当前时间:*math:‘t*‘进行积分。我们使用:*math:*tau‘来进行积分，因为我们需要一个变量来在整个积分中采用多个值，但是不能使用:math:‘t‘来实现，因为我们已经将其定义为当前时间。

下图显示了由 PI 控制器控制的*system* 的框图。

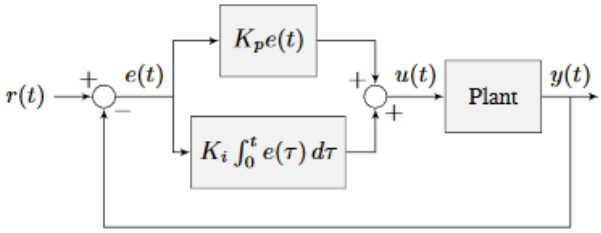


Figure 2.3: PI controller block diagram

当*system* 处于稳定状态时关闭系统 setpoint‘时，比例项可能太小而无法将输出一直拉至:term:‘setpoint，导数项为零。这可能导致*steady-state error* ，如图 2.4 所示。

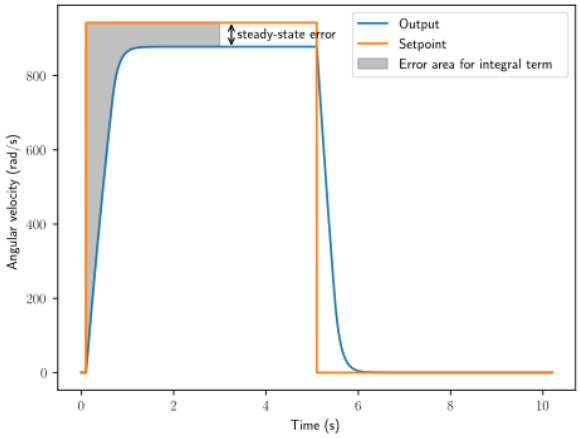


Figure 2.4: P controller with steady-state error

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.

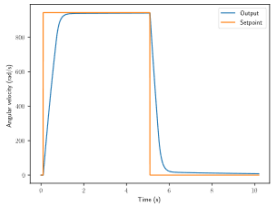


Figure 2.5: PI controller without steady-state error

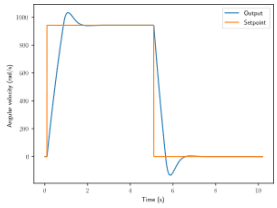


Figure 2.6: PI controller with overshoot from large K_i gain

Putting It All Together

备注： 有关使用 WPILib 提供的 PIDController 的信息，请参见*relevant article*。

When these terms are combined by summing them all together, one gets the typical definition for a PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de}{dt}$$

其中 K_p 是比例增益，K_i 是积分增益，K_d‘□□□□□□ : $math$: ‘$e(t)$ 是当前时间的误差。t‘□ : $math$:tau‘是积分变量。

下图显示了 PID 控制器的框图。

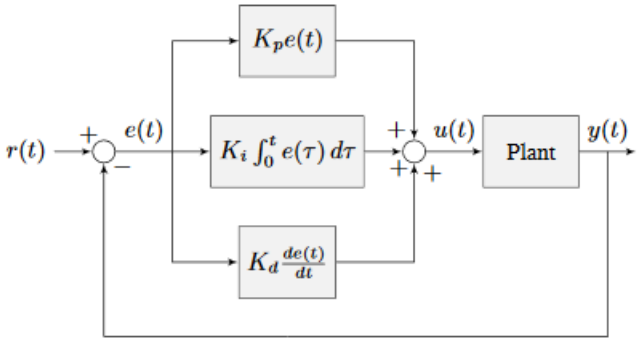


Figure 2.7: PID controller block diagram

回应类型

由 PID 控制器驱动的*system* 通常具有三种类型的响应：欠阻尼，过阻尼和临界阻尼。这些如图 2.8 所示。

对于图 2.7 中的 step responses <step response>，:term:‘rise time 是指:term:*system*‘在应用*:term:‘step input*‘后初始到达参考所花费的时间。*:term:‘Settling time<settling time>* 是指在应用*:term:‘step input* 之后，*system* 在*reference* 考建立时所花费的时间。

在建立之前，*衰减 * 的响应会在*reference* 附近振荡。*过度 * 响应

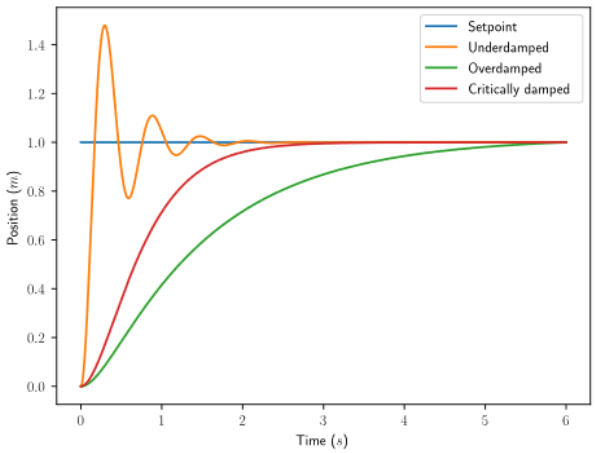


Figure 2.8: PID controller response types

上升缓慢，并且不会超出*reference*。“临界阻尼”响应具有最快的*rise time*，而不会超出*reference*。

32.2.5 WPI 的 PID 简介视频

你是否曾在设计快速移动的机器人系统时遇到过困难？当移动固定的距离或速度，操作手臂或电梯，或任何其他需要特定动作的电机控制系统时，就会出现这样的挑战。在这段视频中，WPI 的教授 Dmitry Berenson 阐述了机器人控制和 PID 控制是如何工作的。

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

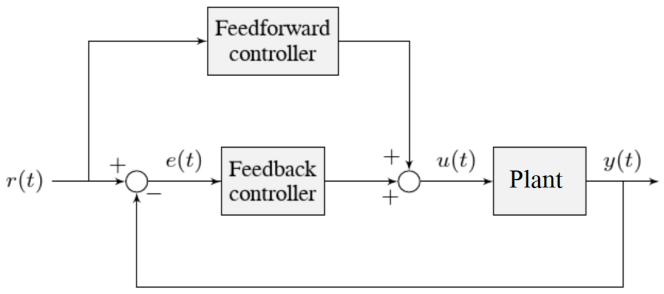
In this section, we will tune a simple velocity controller for a flywheel. The tuning principles explained here will also work for almost any velocity control scenario.

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 *SimpleMotor* 前馈, 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

---

备注： 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.

备注： 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.

[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.

Common Issues with Bang-Bang Controllers

Note that the system behavior is not perfect, because of delays in the control loop. These can result from the nature of the sensors, measurement filters, loop iteration timers, or even delays in the control hardware itself. Collectively, these cause a cycle of "overshoot" and "undershoot", as the output repeatedly goes above and below the setpoint. This oscillation is unavoidable with a bang-bang controller.

Typically, the steady-state oscillation of a bang-bang controller is small enough that it performs quite well in practice. However, rapid on/off cycling of the control effort can cause mechanical issues - the cycles of rapidly applying and removing forces can loosen bolts and joints, and put a lot of stress on gearboxes.

The abrupt changes in control effort can cause abrupt changes in current draw if the system's inductance is too low. This may stress motor control hardware, and cause eventual damage or failure.

Finally, this technique only works for mechanisms that accelerate relatively slowly. A more in-depth discussion of the details *can be found here*.

Bang-bang control sacrifices a lot for simplicity and high performance (in the sense of fast convergence to the setpoint). To achieve "smoother" control, we need to consider a different control strategy.

Pure Feedforward Control

Interact with the simulation below to see how the flywheel system responds when controlled only by a feedforward controller.

To tune the feedforward controller, increase the velocity feedforward gain K_v until the flywheel approaches the correct setpoint over time. If the flywheel overshoots, reduce K_v.

The exact gain used by the simulation is $K_v = 0.0075$.

We can see that a pure feedforward control strategy works reasonably well for flywheel velocity control. As we mentioned earlier, this is why it's possible to control most motors "directly" with joysticks, without any explicit "control loop" at all. However, we can still do better - the pure feedforward strategy cannot reject disturbances, and so takes a while to recover after the ball is introduced. Additionally, the motor may not perfectly obey the feedforward equation (even after accounting for vibration/noise). To account for these, we need a feedback controller.

Pure Feedback Control

Interact with the simulation below to see how the flywheel system responds when controlled by only a feedback (PID) controller.

Perform the following:

1. Set K_p, K_i, K_d, and K_v to zero.
2. Increase K_p until the *output* starts to oscillate around the *setpoint*, then decrease it until the oscillations stop.
3. *In some cases*, increase K_i if *output* gets "stuck" before converging to the *setpoint*.

备注： 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.

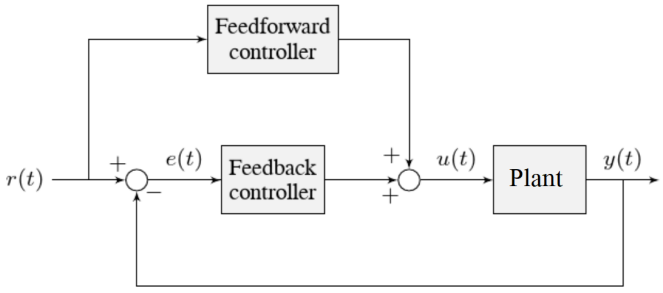
Turret Model Description

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 *SimpleMotor* 前馈, 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.

备注： 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.

备注： 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 *SimpleMotor* 前馈 equation (as long as the "system noise" option is disabled). We might then expect, like in the case of the *flywheel velocity controller*, that we should be able to achieve perfect convergence-to-setpoint with a feedforward loop alone.

However, our feedforward equation relates *velocity* and *acceleration* to voltage - it allows us to control the *instantaneous motion* of our mechanism with high accuracy, but it does not allow us direct control over the *position*. This is a problem even in our simulation (in which the feedforward equation is the *actual* equation of motion), because unless we employ a *motion profile* to generate a sequence of velocity setpoints we can ask the turret to jump immediately from one position to another. This is impossible, even for our simulated turret.

The resulting behavior from the feedforward controller is to output a single "voltage spike" when the position setpoint changes (corresponding to a single loop iteration of very high velocity), and then zero voltage (because it is assumed that the system has already reached the setpoint). In practice, we can see in the simulation that this results in an initial "impulse" movement towards the target position, that stops at some indeterminate position in-between. This kind of response is called a "kick," and is generally seen as undesirable.

You may notice that *smooth* motion below the turret's maximum achievable speed can be followed accurately in the simulation with feedforward alone. This is misleading, however, because no real mechanism perfectly obeys its feedforward equation. With the "system noise"

option enabled, we can see that even smooth, slow motion eventually results in compounding position errors when only feedforward control is used. To accurately converge to the setpoint, we need to use a feedback (PID) controller.

Pure Feedback Control

Interact with the simulation below to examine how the turret system responds when controlled only by a feedback (PID) controller.

Perform the following:

1. Set K_p, K_i, K_d, and K_v to zero.
2. Increase K_p until the mechanism responds to a sudden change in setpoint by moving sharply to the new position. If the controller oscillates too much around the setpoint, reduce *K_p* until it stops.
3. Increase K_d to reduce the amount of "lag" when the controller tries to track a smoothly moving setpoint (reminder: click and drag the turret's directional indicator to move it smoothly). If the controller starts to oscillate, reduce *K_d* until it stops.

Gains of $K_p = 0.3$ and $K_d = 0.05$ yield rapid and stable convergence to the setpoint. Other, similar gains will work nearly as well.

Issues with Feedback Control Alone

Note that even with system noise enabled, the feedback controller is able to drive the turret to the setpoint in a stable manner over time. However, it may not be possible to smoothly track a moving setpoint without lag using feedback alone, as the feedback controller can only respond to errors once they have built up. To get the best of both worlds, we need to combine our feedback controller with a feedforward controller.

Combined Feedforward and Feedback Control

Interact with the simulation below to examine how the turret system responds under simultaneous feedforward and feedback control.

Tuning the combined turret controller is simple - we first tune the feedforward controller following the same procedure as in the feedforward-only section, and then we tune the PID controller following the same procedure as in the feedback-only section. Notice that PID portion of the controller is *much* easier to tune "on top of" an accurate feedforward.

The optimal gains for the combined controller are just the optimal gains for the individual controllers: gains of $K_v = 0.15$, $K_p = 0.3$, and $K_d = 0.05$ yield rapid and stable convergence to the setpoint and relatively accurate tracking of smooth motion. Other, similar gains will work nearly as well.

Once tuned properly, the combined controller should accurately track a smoothly moving setpoint, and also accurately converge to the setpoint over time after a "jump" command.

Tuning Conclusions

Choice of Control Strategies

Like in the case of the *vertical arm*, and unlike the case of the *flywheel*, we are trying to control the *position* rather than the *velocity* of our mechanism.

In the case of the flywheel *velocity* controller we could achieve good control performance with feedforward alone. However, it is very hard to predict how much voltage will cause a certain total change in *position* (time can turn even small errors in velocity into very big errors in position). In this case, we cannot rely on feedforward control alone - as with the vertical arm, we will need a feedback controller.

Unlike in the case of the vertical arm, though, there is no voltage required to keep the mechanism at the setpoint once it's there. As a consequence, it is often possible to effectively control a turret without any feedforward controller at all, relying only on the output of the feedback controller (if the mechanism has a lot of friction, this may not work well and both a feedforward and feedback controller may be needed). Simple position control in the absence of external forces is one of the only cases in which pure feedback control works well.

Controlling a mechanism with only feedback can produce reasonable results in cases where no *control effort* is required to keep the *output* at the *setpoint*. On a turret, this can work acceptably - however, it may still run into problems when trying to follow a moving setpoint, as it relies entirely on the controller transients to control the mechanism's intermediate motion between position setpoints.

We saw in the feedforward-only example above that an accurate feedforward can track slow, smooth velocity setpoints quite well. Combining a feedforward controller with the feedback controller gives the smooth velocity-following of a feedforward controller with the stable long-term error elimination of a feedback controller.

Reasons for Non-Ideal Performance

This simulation does not include any motion profile generation, so acceleration setpoints are not very well-defined. Accordingly, the *kA* term of the feedforward equation is not used by the controller. This means there will be some amount of delay/lag inherent to the feedforward-only response.

A Note on Feedforward and Static Friction

For the sake of simplicity, the simulations above omit the K_s term from the WPILib SimpleMotorFeedforward equation. On actual mechanisms, however, this can be important - especially if there's a lot of friction in the mechanism gearing. A turret with a lot of static friction will be very hard to control accurately with feedback alone - it will get "stuck" near (but not at) the setpoint when the loop output falls below K_s.

To measure K_s manually, slowly increase the voltage to the mechanism until it starts to move. The value of K_s is the largest voltage applied before the mechanism begins to move.

It can be mildly difficult to *apply* the measured K_s to a position controller without motion profiling, as the WPILib SimpleMotorFeedforward class uses the velocity setpoint to determine the direction in which the K_s term should point. To overcome this, either use a motion profile, or else add K_s manually to the output of the controller depending on which direction the mechanism needs to move to get to the setpoint.

32.2.9 Tuning a Vertical Arm Position Controller

In this section, we will tune a simple position controller for a vertical arm. The same tuning principles explained below will work also for almost all position-control scenarios under the load of gravity.

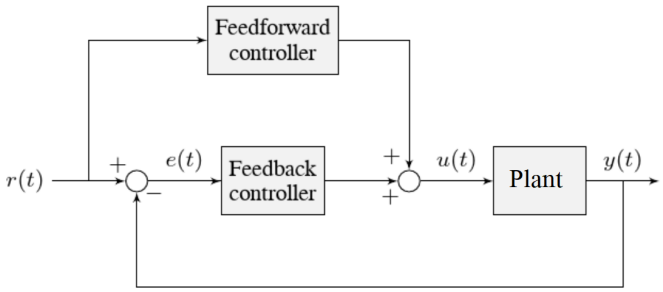
Arm Model Description

Vertical arms are commonly used to lift gamepieces from the ground up to a scoring position. Other similar examples include shooter hoods and elevators.

Our "vertical arm" consists of:

- A mass on a stick, under the force of gravity, pivoting around an axle.
- A motor and gearbox driving the axle to which the mass-on-a-stick is attached

For the purposes of this tutorial, this plant is modeled with the same equation used by WPILib's 臂前馈, with additional adjustment for sensor delay and gearbox inefficiency. The simulation assumes the plant is controlled by feedforward and feedback controllers, composed in this fashion:



Where:

- The plant's *output* $y(t)$ is the arm's rotational position
- The controller's *setpoint* $r(t)$ is the desired angle of the arm
- The controller's *control effort*, $u(t)$ is the voltage applied to the motor driving the arm

Picking the Control Strategy for a Vertical Arm

Applying voltage to the motor causes a force on the mechanism that drives the arm up or down. If there is no voltage, gravity still acts on the arm to pull it downward. Generally, it is desirable to fight this effect, and keep the arm at a specific angle.

The tutorials below will demonstrate the behavior of the system under pure feedforward, pure feedback (PID), and combined feedforward-feedback control strategies. Follow the instructions to learn how to manually tune these controllers, and expand the "tuning solution" to view an optimal model-based set of tuning parameters. Even though WPILib tooling can provide you with optimal gains, it is worth going through the manual tuning process to see how the different control strategies interact with the mechanism.

Pure Feedforward Control

Interact with the simulation below to examine how the turret system responds when controlled only by a feedforward controller.

备注: 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.

备注: 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 臂前馈 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.

备注: 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.

重要: 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 滤波器

备注: 用于生成本节各种演示图的数据可在 download:'here <resources/filterdemo.csv>'找到。

本节描述了 WPILib 的许多滤波器，这些滤波器可用于降低噪音/使输入平滑。

32.3.1 过滤器简介

滤波器是现代技术中最常用的工具之一，在机器人的信号处理和控制的中有广泛的应用。理解过滤器的概念对于理解 WPILib 提供的各种类型的过滤器的实用性至关重要。

什么是滤波器？

备注: 为了本文的方便，我们将假定所有数据均为一维的时间序列数据。显然，滤波器的概念并非仅仅如此，但对信号和滤波完整，严格的讨论超出了本文档的讨论。

那么，什么才是过滤器呢？简而言之，过滤器是从输入流到输出流的映射。也就是说，过滤器输出的值（原则上）不仅可以取决于输入的当前值，还可以取决于过去和将来的值的整个集合（当然，实际上，WPILib 提供的滤波器只能数据流上实时实现；因此，它们只能取决于过去的输入值，而不能取决于将来的值）。这是一个重要的概念，因为通常我们使用滤波器来去除或减轻信号中不需要的动态特性。当我们过滤信号时，我们要修改信号随时间的变化。

使用滤波器的效果

降噪

滤波器最典型的用途之一就是降低噪声。降低噪声的滤波器称为低通滤波器（因为它允许低频“通过”，并阻止高频）。当前 WPILib 中包含的大多数滤波器实际上都是低通滤波器。

速率限制

滤波器通常也用于降低信号变化的速率。这与降噪密切相关，并且降低噪声的滤波器也往往会限制信号变化的速率。

边缘探测

与低通滤波器相对应的是高通滤波器，它只允许高频通过输出。高通滤波器很难用直觉理解，但它被普遍用于边缘检测，因为高通滤波器将反映输入的突然变化而忽略慢变化，可以用于决定大幅不连续的信号。

相位滞后

实时低通滤波器不可避免的一个负面影响是“相位滞后”的引入。正如前面提到的，实时滤波器只能依赖于信号的过去值 (我们不能通过时间旅行来获得未来值)，因此当输入开始变化时，滤波后的值需要一段时间才能“赶上”。降噪越大，引入的延迟就越大。在许多方面，这是实时过滤基本的权衡，并且应该是您的过滤器设计的主要考虑因素。

有趣的是，高通滤波器引入了相位超前，而不是相位滞后，因为它们加剧了输入值的局部变化。

32.3.2 线性滤波器

WPILib 支持的第一种 (也是最常用的) 过滤器是线性过滤器，或者更具体地说，线性定常 (LTI) 过滤器。

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*.

有两种广义的脉冲响应: 无限脉冲响应 (IIR) 和有限脉冲响应 (FIR)。

无限脉冲响应具有无限的“支持”——也就是说，它们在一个无穷大的区域上是非零的。广义上来说，这意味着它们也有无限的“内存”——一旦一个值出现在输入流中，它将永远影响所有后续输出。从严格的信号处理角度来看，这通常是不希望看到的，但是具有无限脉冲响应的滤波器往往非常容易计算，因为它们可以用简单的递归关系来表示。

有限冲激响应具有有限的“支持”-也就是说，它们在有界区域上不为零。“典型”FIR 滤波器是一个固定的移动平均值-也就是说，只需将输出设置为等于过去 n 个输入的平均值即可。FIR 滤波器往往比 IIR 滤波器具有更理想的属性，但计算成本更高。

Linear filters are supported in WPILib through the `LinearFilter` class (Java, C++, , `Python`).

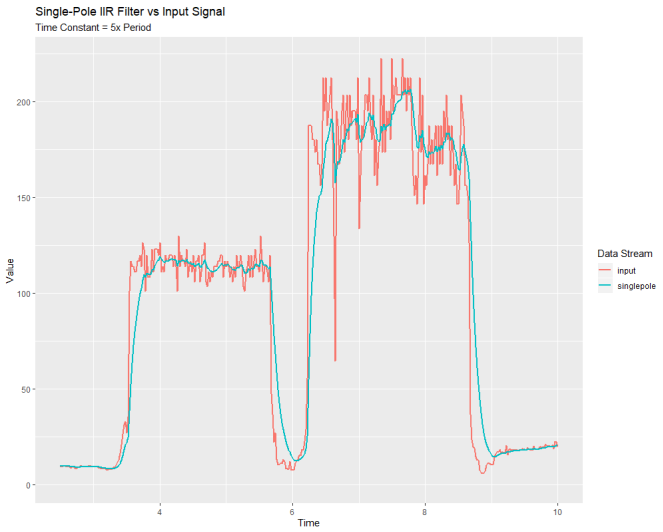
创建一个 LinearFilter

备注： c++“LinearFilter“类被模板化为用于输入的数据类型。

备注： 因为过滤器有“内存”，所以每个输入流都需要它自己的过滤器对象。不要尝试对多个输入流使用相同的过滤器对象。

虽然可以直接实例化“LinearFilter”类以构建自定义过滤器，但使用提供的工厂模式之一要方便得多（并且更常见），而不是：

单极 IIR



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);
```

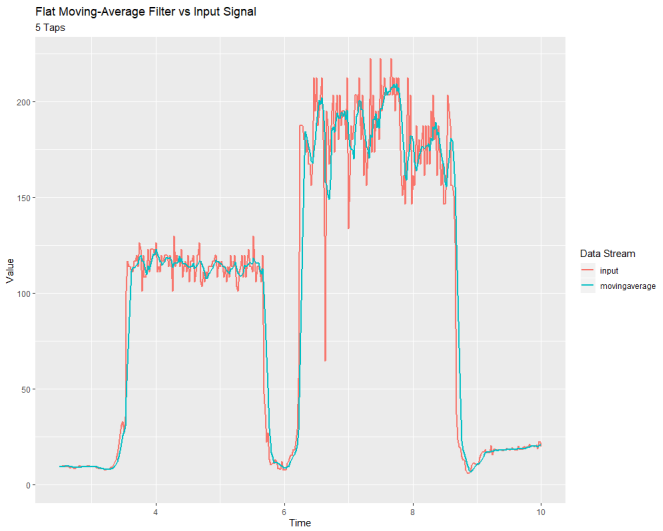
PYTHON

```
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)
```

参数“时间常数”决定了滤波器的脉冲响应的“特征时间尺度”。滤波器将抵消在明显短于此的时间范围内发生的任何信号动态。相关地，它也是引入的相位滞后:ref:‘phase lag <docs/software/advanced-controls/filters/introduction:Phase Lag>‘的近似时间尺度。该时间标度的倒数乘以 2，即为滤波器的“截止频率”。

参数“period”是过滤器的 calculate () 方法将被调用的时间段。对于绝大多数实施方式，这将是 0.02 秒的标准主机器人循环周期。

移动平均值



“movingAverage’’工厂模式创建一个简单的平面移动平均滤波器。这是最简单的低通 FIR 滤波器，在许多与单极 IIR 滤波器相同的环境中很有用。它的计算成本较高，但通常表现得更好。

JAVA

```
// Creates a new flat moving average filter
// Average will be taken over the last 5 samples
LinearFilter filter = LinearFilter.movingAverage(5);
```

C++

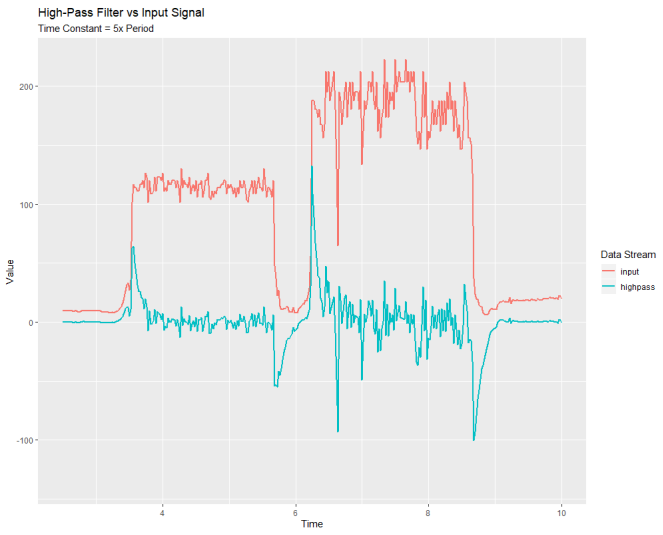
```
// Creates a new flat moving average filter
// Average will be taken over the last 5 samples
frc::LinearFilter<double> filter = frc::LinearFilter<double>::MovingAverage(5);
```

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)
```

“抽头”参数是将包含在固定移动平均值中的样本数。这与上面的“时间常数”类似-有效时间常数是抽头数乘以调用 calculate () 的时间段。

高通



highPass 工厂模式创建一个简单的一阶无限冲激响应高通滤波器。这是“singlePoleIIR” _ 的“对手”。

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)
```

“时间常数”参数确定了滤波器的脉冲响应的“特征时间尺度”；滤波器将消除在时间尺度上明显长于此的任何信号动态。相关地，它也是引入的相位超前*phase lead*. 的近似时间表。该时间刻度的倒数乘以 2 ，即为滤波器的“截止频率”

参数“period”是过滤器的 calculate () 方法将被调用的时间段。对于绝大多数实施方式，这将是 0.02 秒的标准主机器人循环周期。

使用 LinearFilter

备注： 为了使创建的过滤器服从指定的时标参数，必须在指定时间段定期调用其 calculate () 函数 *。如果由于某种原因，必须在 calculate () 调用中发生重大失误，则应在进一步使用之前调用过滤器的 reset () 方法。

创建过滤器后，使用起来很容易-只需使用最新输入调用 calculate () 方法即可获取过滤后的输出：

JAVA

```
// Calculates the next value of the output
filter.calculate(input);
```

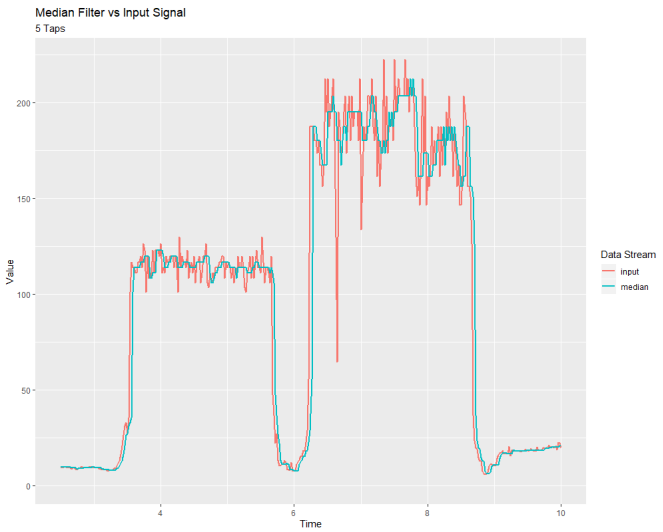
C++

```
// Calculates the next value of the output
filter.Calculate(input);
```

PYTHON

```
# Calculates the next value of the output
filter.calculate(input)
```

32.3.3 中值过滤器



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.

中值滤波器最有用的是从输入流中删除偶尔的异常值。这使其特别适合于过滤来自距离传感器的输入，这些输入容易受到偶尔的干扰。与移动平均值不同，中值过滤器将完全不受少量异常值的影响，无论多么极端。

The median filter is supported in WPILib through the `MedianFilter` class (Java, C++, , Python).

创建一个 MedianFilter

备注： C ++“MedianFilter’’类是用于输入的数据类型的模板。

备注： 因为过滤器具有“内存”，所以每个输入流都需要其自己的过滤器对象。*不要 * 尝试对多个输入流使用相同的过滤器对象。

创建一个 MedianFilter 很简单：

JAVA

```
// Creates a MedianFilter with a window size of 5 samples
MedianFilter filter = new MedianFilter(5);
```

C++

```
// Creates a MedianFilter with a window size of 5 samples
frc::MedianFilter<double> filter(5);
```

PYTHON

```
from wpimath.filter import MedianFilter

# Creates a MedianFilter with a window size of 5 samples
filter = MedianFilter(5)
```

使用 MedianFilter

创建过滤器后，使用起来很容易-只需使用最新输入调用 calculate（）方法即可获取过滤后的输出：

JAVA

```
// Calculates the next value of the output
filter.calculate(input);
```

C++

```
// Calculates the next value of the output
filter.Calculate(input);
```

PYTHON

```
# Calculates the next value of the output
filter.calculate(input)
```

32.3.4 回转速率限制器

FRC | reg |中过滤器的常见用法用于软化控制输入的行为（例如，驱动程序控件的操纵杆输入）。不幸的是，一个简单的低通滤波器不适合该工作。低通滤波器将软化输入流对突然变化的响应，但同时也会淘汰精细的控制细节并引入相位滞后。更好的解决方案是直接限制控制输入的变化率。这是通过 * 压摆率限制器 * 执行的-限制信号最大变化率的滤波器。

回转速率限制器可以被认为是一种原始的运动曲线。事实上，回转速率限制器是由 WPILib 支持的梯形运动曲线:ref:*Trapezoidal Motion Profile <docs/software/advanced-controls/controllers/trapezoidal-profiles:Trapezoidal Motion Profiles in WPILib>* 的一阶等价物。它恰恰是梯形运动的极限情况下，当加速度约束被允许趋向于无穷大。因此，转换速率限制器是一个很好的选择，应用实际的运动剖面流的速度设定值 (或电压，通常是近似成比例的速度)。对于控制位置的输入流，通常最好使用适当的梯形剖面。

Slew rate limiting is supported in WPILib through the `SlewRateLimiter` class (Java, C++, Python).

创建一个 SlewRateLimiter

备注： C ++ 的 “SlewRateLimiter” 类以输入的单位类型为模板。有关 C ++ 单元的更多信息，请参见:ref:*docs/software/basic-programming/cpp-units:The C++ Units Library*.

备注： 因为过滤器具有 “内存”，所以每个输入流都需要其自己的过滤器对象。*不要 * 尝试对多个输入流使用相同的过滤器对象。

创建 SlewRateLimiter 很简单:

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)
```

使用 SlewRateLimiter

创建过滤器后，使用起来很容易-只需使用最新输入调用 calculate（）方法即可获取过滤后的输出：

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

备注： 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 几何类别

本节介绍 WPILib 的几何类别。

32.4.1 平移、旋转和姿态

翻译

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.

您可以使用 "getDistance（Translation2d other）" 来获取到另一个 "Translation2d" 对象的距离，该距离可以通过毕达哥拉斯定理返回到另一个 Translation2d 的距离。

备注： `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.

旋转

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.

备注： `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.

备注： `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.

位姿

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 转变

Translation2d

在 Translation2d 上的操作对由 Translation2d 表示的向量执行操作。

- 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.
- 旋转：通过使用 “rotateBy”，可以绕着原点逆时针旋转 “Translation2d”。这相当于将向量乘以矩阵：math:*begin{bmatrix} costheta & -sintheta \ sintheta & costheta end{bmatrix}*
- Additionally, you can rotate a `Translation2d` by 180 degrees by using `unaryMinus` in Java, or the unary `-` operator in C++/Python.

Rotation2d

“Rotation2d’’ 的转换只是对由 “Rotation2d“表示的角度度量的算术运算。

- `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 and 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''代表一个``相对''转换。它具有平移和旋转组件。通过 “Transform2d” 变换 “Pose2d”，将通过姿势的旋转来旋转变换的平移分量，然后将旋转的平移分量和旋转分量添加到该位姿。换句话说，``Pose2d.plus(Transform2d)` 返回 $\begin{bmatrix} x_p \\ y_p \\ \theta_p \end{bmatrix} + \begin{bmatrix} cos\theta_p & -sin\theta_p & 0 \\ sin\theta_p & cos\theta_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$

“Twist2d” 表示沿弧线的距离变化。通常，这个类用于表示传动系统的运动，其中 x 组件是向前行驶的距离，y 组件是向侧面行驶的距离 (左正)，而:math: ‘theta ‘组件是航向的变化。我们可以在第 10 章，‘此处 <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>‘找到寻找指数型位姿 (沿扭转的曲率向前移动后的新位姿) 背后的基本数学原理。

备注： 对于非完整的传动系统，“Twist2d’’ 的 y 分量应始终为 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

备注: 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.

备注: 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

备注： 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.

备注： 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.

备注: 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.

警告： 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.

警告： 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:

备注： A drivetrain routine is shown below, but the same motions will occur on any mechanism.

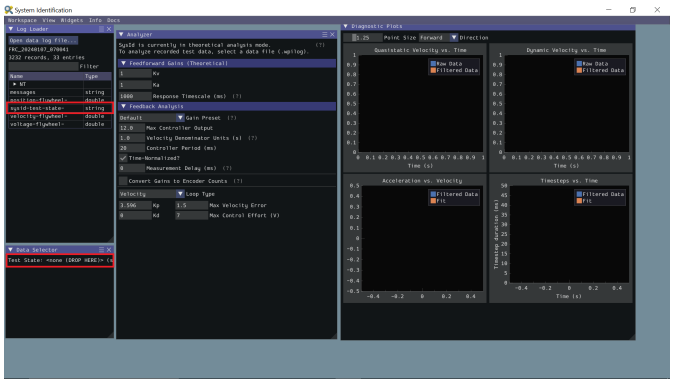
After all four tests have been completed, use the `DataLogTool` to retrieve the log file from the roboRIO.

32.5.4 Loading Data

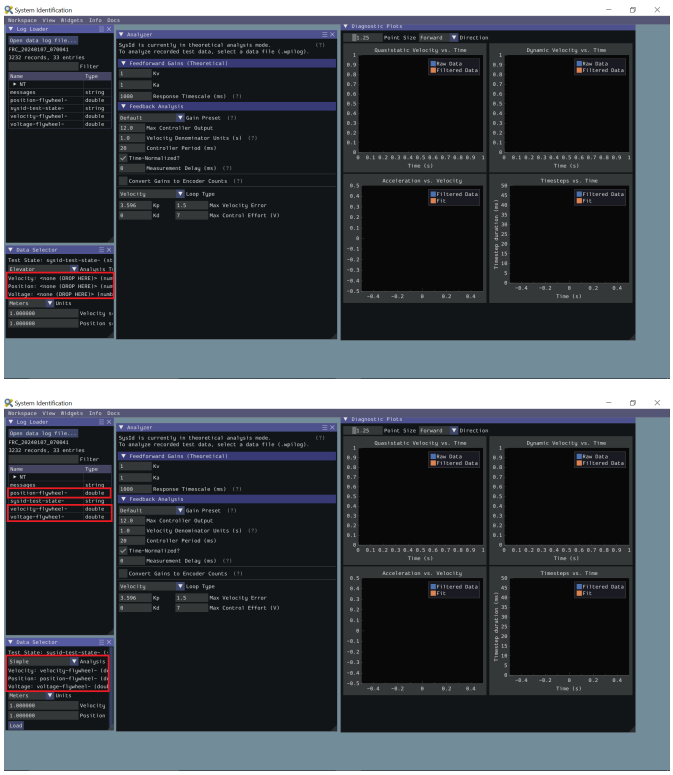
After downloading the WPILog containing the tests from the roboRIO, go to the `Log Loader` pane in SysId and click `Open data log file...`.

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.

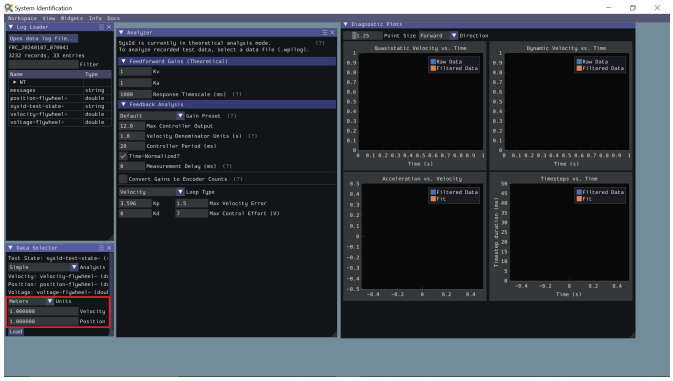
备注： SysIdRoutine will name the entry "sysid-test-state-mechanism", where "mechanism" is the name passed to the `Mechanism` contructor or the subsystem name.



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.

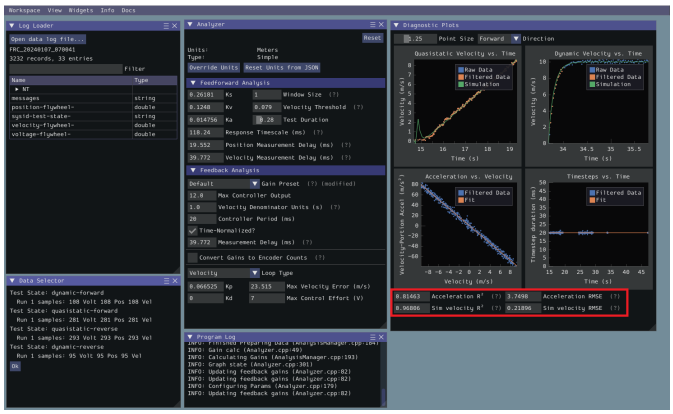


Ensure the correct analysis type has been selected, then click the `Load` button and move on to checking the fit diagnostics in the `Diagnostics` pane.

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*.



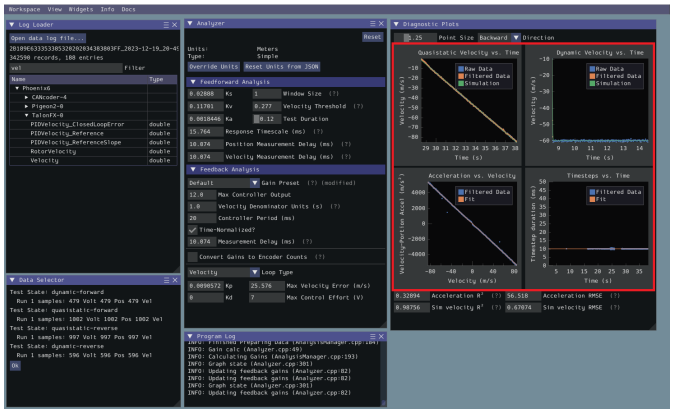
The acceleration r-squared is the fraction of the variance in measured acceleration (used as the independent variable in the SysId regression) explained by the linear model. This can be quite variable, because acceleration is very susceptible to system noise. Mechanisms tend to vibrate quite a bit, so 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

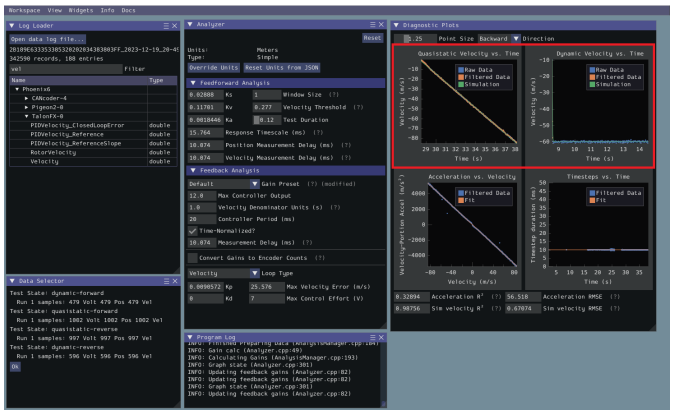
SysId also produces several diagnostic plots to help users evaluate the quality of their model fit.



Time-Domain Plots

备注： 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:



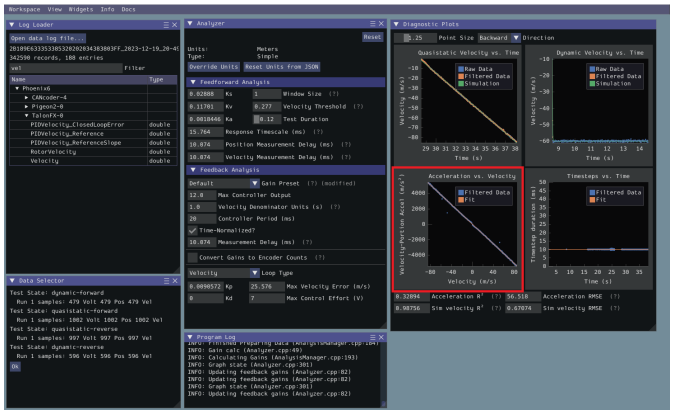
The velocity time domain plots contain three sets of data: Raw Data, Filtered Data, and Simulation. The Raw Data is the recorded data from your robot, the Filtered Data is the data after a median filter has been applied to the data, and the Simulation represents the velocity predictions of a model based 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.

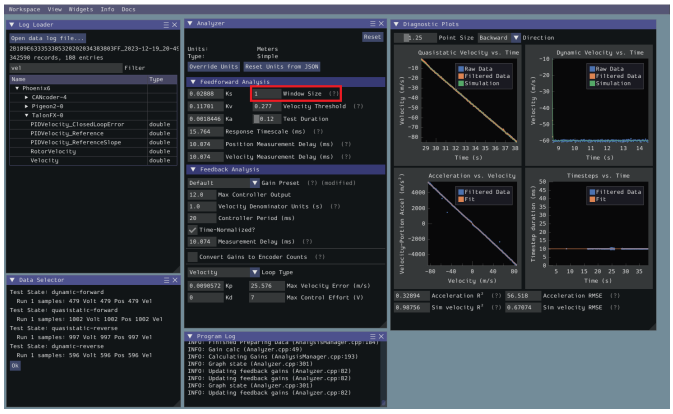
Deviation from this behavior is a sign of an *error*, either in your robot setup, analysis settings, or your test procedure.

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.



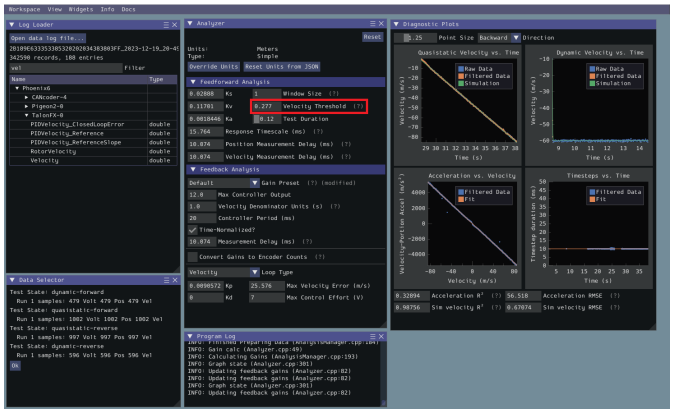
However, if your robot or mechanism has low mass compared to the motor power, this may "eat" what little meaningful acceleration data you have. In these cases kA will tend towards zero and can be ignored for feedforward purposes. However, if kA cannot be accurately measured, the calculated feedback gains are likely to be inaccurate, and manual tuning may be required.

Common Failure Modes

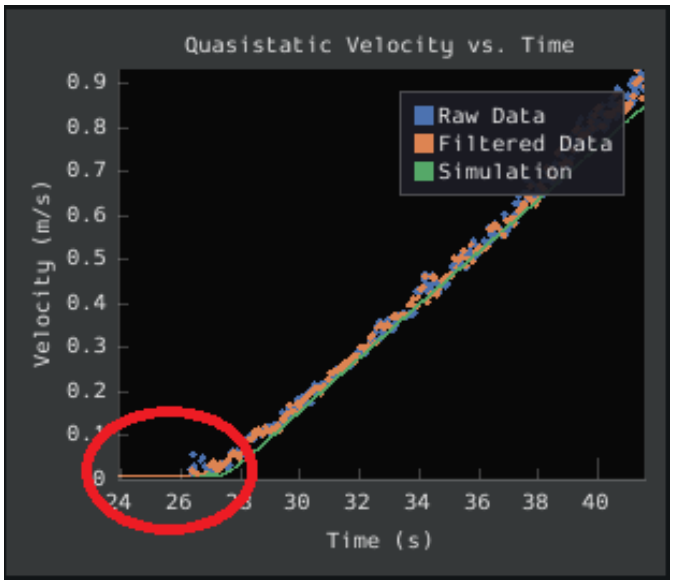
When something has gone wrong with the identification, diagnostic plots and console output provide crucial clues as to *what* has gone wrong. This section describes some common failures encountered while running the system identification tool, the identifying features of their diagnostic plots, and the 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.



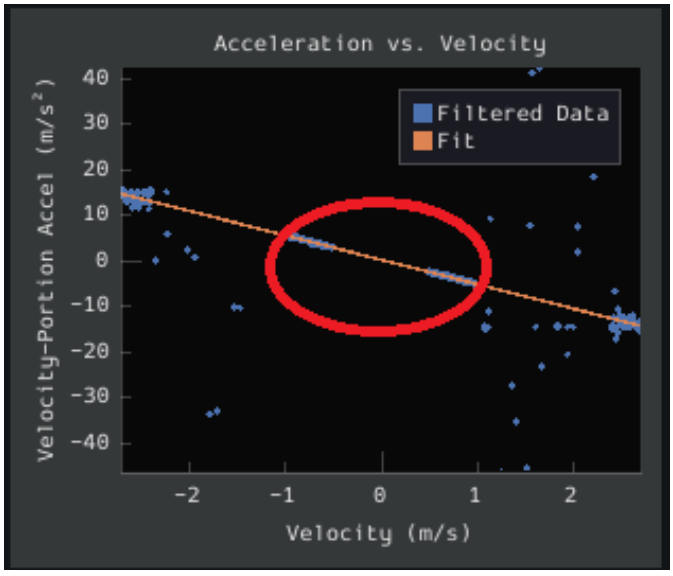
Velocity Threshold Too Low



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

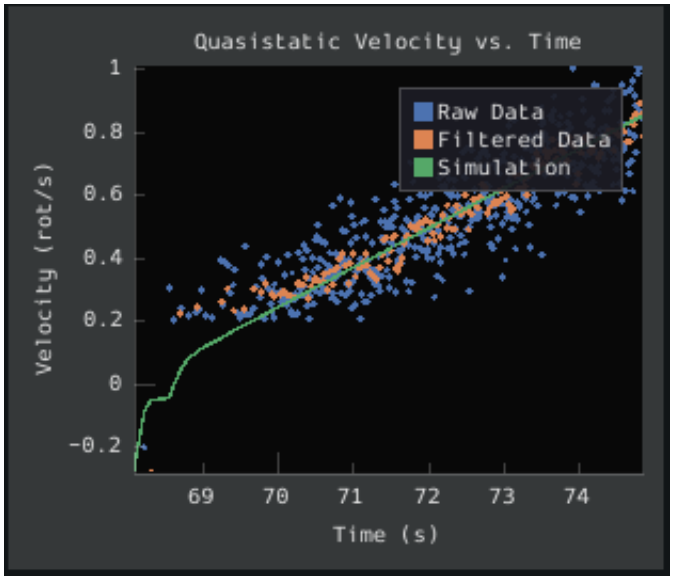


While not nearly as problematic as a too-low threshold, a velocity threshold that is too high will result in a large "gap" in the acceleration-versus-velocity plot.

To solve this, decrease the velocity threshold and re-analyze the data.

Noisy Velocity Signals

备注： 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).

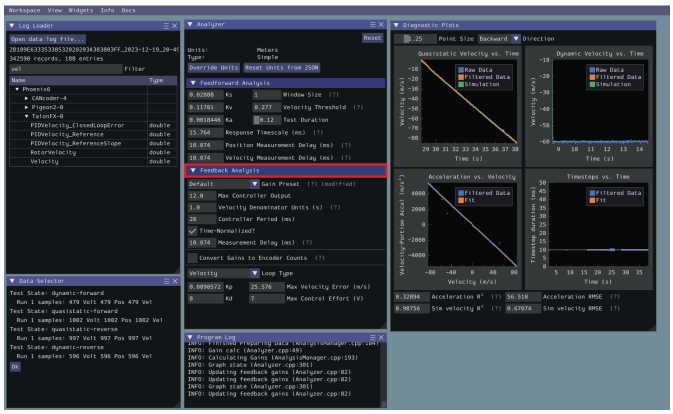
32.5.6 Analyzing Data

Feedforward Analysis

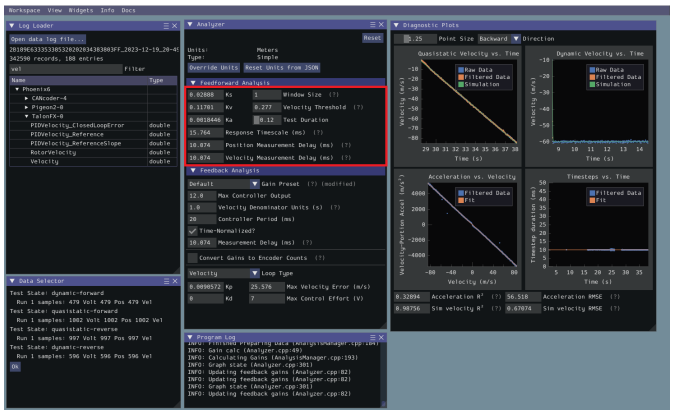
备注： For information on what the calculated feedback gains mean, see *The Permanent-Magnet DC Motor Feedforward Equation*. For information on using the calculated feedback gains in code, see *feedforward control*.

Click the dropdown arrow on the *Feedforward* Section.

备注： If you would like to change units, you will have to press the *Override Units* button and fill out the information on the popup.



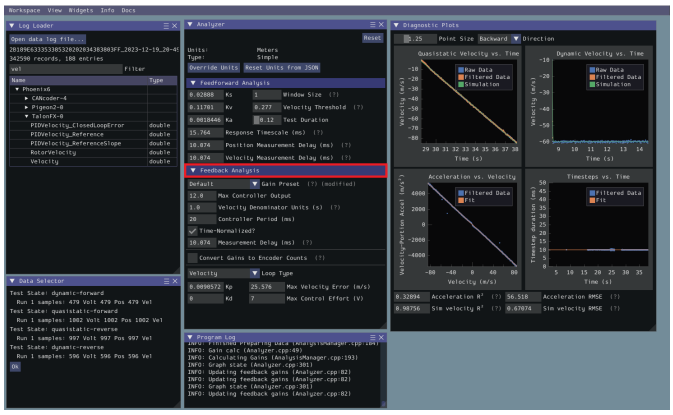
The computed mechanism system parameters will then be displayed.



Feedback Analysis

重要： 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.



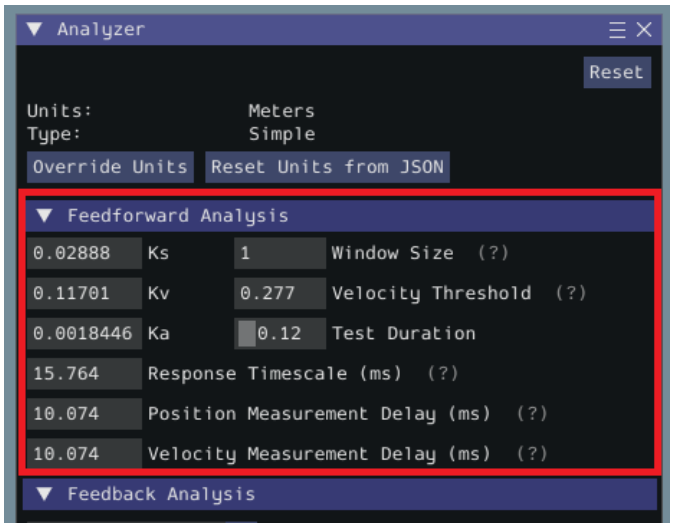
This view can be used to calculate optimal feedback gains for a PD or P controller for your mechanism (via *LQR*).

Enter Controller Parameters

备注： 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.

备注： 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

备注： 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*.

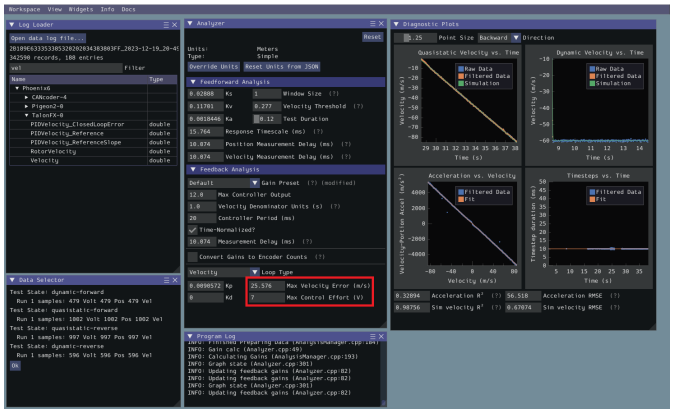
The following only applies if the user decides to implement their own custom filtering settings (e.g. adding a moving average filter to a WPILib PID loop or changing smart motorcontroller measurement period and/or measurement window size) as the measurement delay must be recalculated. Here is the general 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.

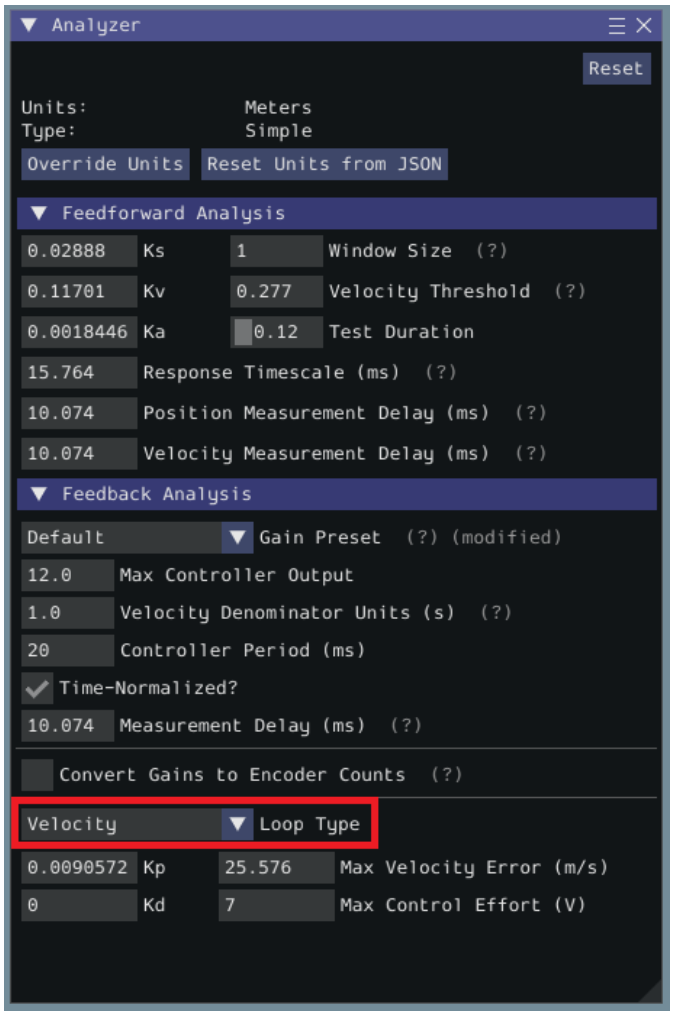


As a rule, smaller values for the *Max Acceptable Error* and larger values for the *Max Acceptable Control Effort* will result in larger gains - this will result in larger control efforts, which can grant better setpoint-tracking but may cause more violent behavior and greater wear on components.

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.

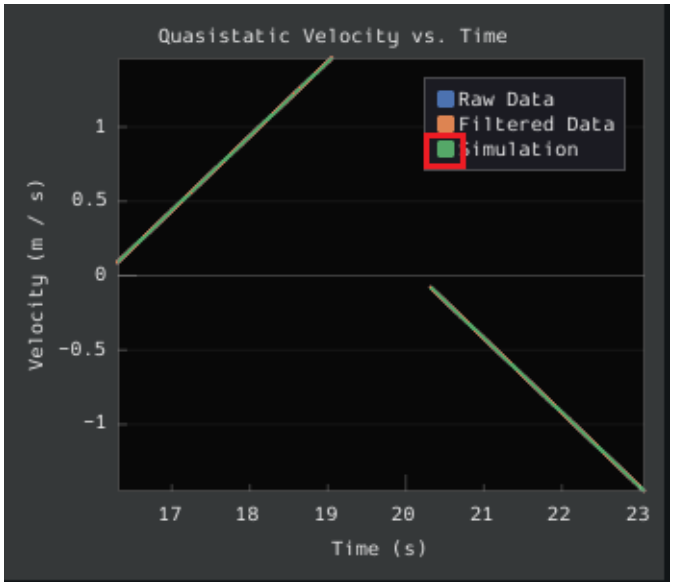
ImGui Tips

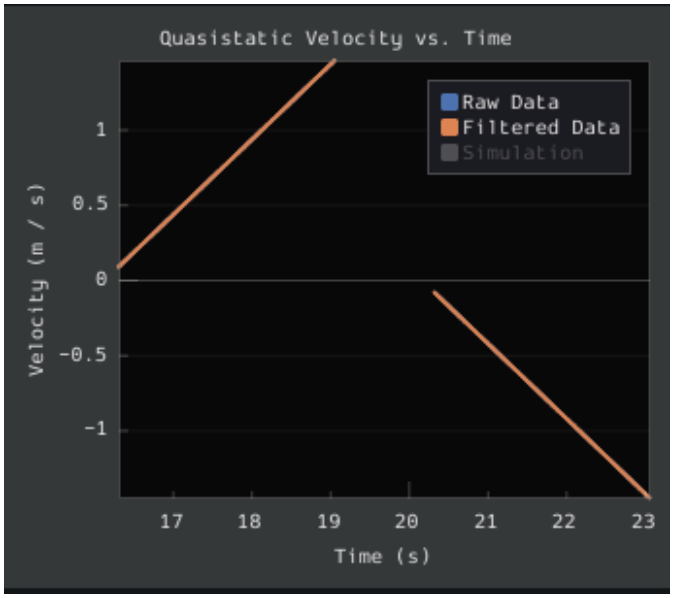
The following are essentially handy features that come with the ImGui framework that SysId uses:

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.

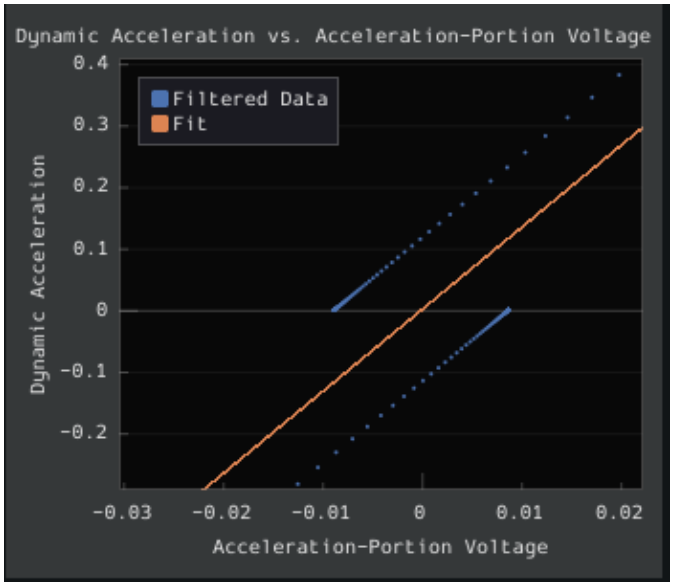




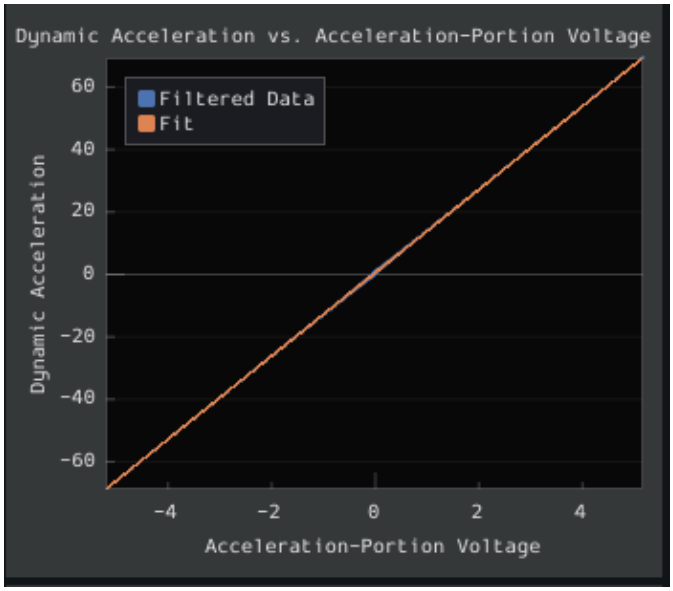
Auto Sizing Plots

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:



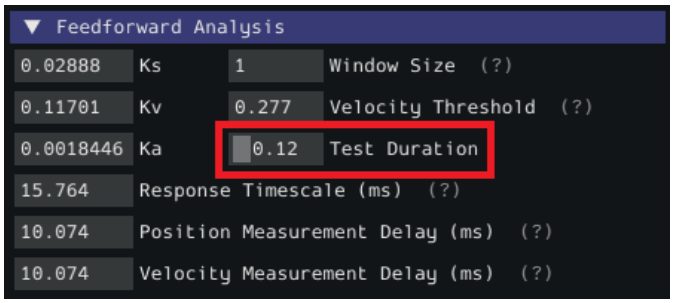
After double clicking, it is automatically resized:



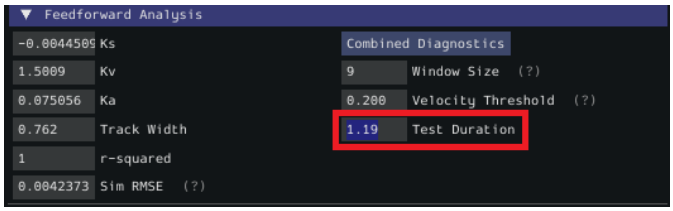
Setting Slider Values

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 控制器

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 WPILib 中的 PID 控制

备注： This article focuses on in-code implementation of PID control in WPILib. For a conceptual explanation of the working of a PIDController, see *PID 简介*

备注： 关于通过以下指南来实现 PID 控制*command-based framework*，参见*通过 PID 子系统和 PID 指令进行 PID 控制*.

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.

使用 PIDController 类

构造一个 PIDController

备注： 虽然 “PIDController“可以异步使用，但它不提供任何线程安全功能-确保线程安全操作完全留给用户使用，因此仅建议高级团队使用异步方法。

为了使用 WPILib 的 PID 控制功能，用户必须首先构造具有所需增益的 “PIDController” 对象：

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)
```

可以向构造函数提供一个可选的第四个参数，指定控制器将运行的时间段。PIDController 对象主要用于与机器人主循环同步使用，因此该值默认为 20ms。

使用反馈回路输出

备注： PIDController 假定以与配置的周期一致的时间间隔定期调用 calculate () 方法。否则，将导致意外的循环行为。

使用构造的 PIDController 很简单：只需从机器人主循环中调用 calculate () 方法（例如，机器人的 autonomousPeriodic () 方法）：

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))
```

检查错误

备注： getPositionError () 和 getVelocityError () 的命名是假设循环正在控制位置-对于控制速度的循环，它们分别返回速度误差和加速度误差。

测量的过程变量的当前错误由 getPositionError () 函数返回，而其派生由 getVelocityError () 函数返回：

指定和检查公差

备注： 如果仅指定位置公差，则速度公差默认为无穷大。

备注： 如上所述，“位置”是指过程变量的测量值，“速度”是指其导数-因此，对于速度环，它们实际上分别是速度和加速度。

有时，知道控制器是否跟踪了设定值到给定的容忍范围内是有用的——例如，确定一个命令是否应该结束，或者 (在遵循一个运动轮廓时) 是否运动被阻碍，需要重新规划。

为此，我们首先必须使用’‘setTolerance() ‘‘方法指定公差; 然后，我们可以用’‘atSetpoint() ‘‘方法检查它。

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()
```

重置控制器

有时需要清除 “PIDController 的内部状态（最重要的是，积分累加器），因为它可能不再有效（例如，``PIDController ``已禁用，然后重新启用）。这可以通过调用 reset() “方法来完成。

设置最大积分器值

备注： 积分器将不稳定性和迟滞引入反馈回路系统。强烈建议团队避免使用积分增益除非绝对没有其他解决方案——通常，问题可以解决一个积分器可以更好的通过使用更为精确的解决*feedforward*.

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)
```

设置连续输入

警告： 如果你的机械装置不能完全连续旋转运动 (例如，一个没有滑环的炮塔，它的电线在旋转时扭曲)，*不要 * 开启连续输入，除非你安装了额外的安全装置，以防止机械装置移动超过其限制!

警告： 连续输入函数 * 不会 * 自动包装您的输入值-请确保您的输入值，在使用此功能时，永远不会超出指定的范围!

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.

要配置一个 " PIDController "来自动执行此操作，请使用 "enableContinuousInput() "方法:

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)
```

钳位控制器输出

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 WPILib 中的前馈控制

备注： 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 提供了许多类来帮助用户实现对其机制的精确前馈控制。在许多方面，准确的前馈比反馈对机制的有效控制更为重要。由于大多数 FRC | reg |机构严格遵循易于理解的系统方程式，从准确的前馈开始既容易又对精确且鲁棒的机构控制非常有利。

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 当前提供以下三个帮助程序类用于前馈控制：

- *SimpleMotorFeedforward* (Java, C++, `Python`)
- *ArmFeedforward* (Java, C++, `Python`)
- *ElevatorFeedforward* (Java, C++, `Python`)

SimpleMotor 前馈

备注： 在 C ++ 中，“SimpleMotorFeedforward’’ 类以用于距离测量的单位类型为模板，可以是角度或线性的。传递的增益 * 必须 * 具有与距离单位一致的单位，否则将引发编译时错误。“kS’’ 的单位应为“伏特’’，“kV’’ 的单位应为“伏特 * 秒/距离’’，“kA’’ 的单位应为“伏特 * 秒/秒 ^ 2 /距离。有关 C ++ 单元的更多信息，请参见：docs / software / basic-programming / cpp-units：C ++ 单元库。

备注： Java 前馈组件将以由用户提供的前馈增益的单位确定的单位来计算输出。用户 * 必须 * 务必保持单元一致，因为 WPILibJ 没有类型安全的单元系统。

备注： 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.

“SimpleMotorFeedforward’’ 类计算由永磁直流电机组成的机构的前馈，这些电机除了摩擦和惯性外没有其他负载，例如飞轮和机器人驱动器。

要创建“SimpleMotorFeedforward’’，只需使用所需的增益即可构造它：

备注： 可以省略“kA’’ 增益，如果是，则默认为零。对于许多机构，尤其是惯性很小的机构，没有必要。

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)
```

要计算前馈，只需使用所需的电动机速度和加速度调用 calculate () 方法即可：

备注： 可以在 calculate () 调用中省略加速度参数，如果是，则默认为零。如果没有明确定义的加速度设定值，则应执行此操作。

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)
```

臂前馈

备注： 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 *C++ 单位库*.

备注： Java 前馈组件将以由用户提供的前馈增益的单位确定的单位来计算输出。用户 * 必须 * 务必保持单元一致，因为 WPILibJ 没有类型安全的单元系统。

备注： 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.

“ArmFeedforward’’ 类可计算由永磁直流电动机直接控制的臂的前馈，并具有外部的摩擦力，惯性和质量。这是 FRC 中大多数武器的准确模型。

要创建一个 “ArmFeedforward“，只需使用所需的增益构造它即可：

备注： 可以省略 “kA’’ 增益，如果是，则默认为零。对于许多机构，尤其是惯性很小的机构，没有必要。

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)
```

要计算前馈，只需使用所需的手臂位置，速度和加速度调用 calculate () 方法即可：

备注： 可以在 calculate () 调用中省略加速度参数，如果是，则默认为零。如果没有明确定义的加速度设定值，则应执行此操作。

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)
```

抬升架前馈

备注： 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 *C++ 单位库*.

备注： Java 前馈组件将以由用户提供的前馈增益的单位确定的单位来计算输出。用户 * 必须 * 务必保持单元一致，因为 WPILibJ 没有类型安全的单元系统。

备注： 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.

“ElevatorFeedforward’’ 类可为电梯计算前馈，该电梯由永磁直流电动机组成，该电动机由电梯的摩擦力，惯性和质量加载。这是 FRC 中大多数电梯的准确模型。

要创建 “ElevatorFeedforward’’，只需使用所需的增益构造它即可：

备注： 可以省略 “kA’’ 增益，如果是，则默认为零。对于许多机构，尤其是惯性很小的机构，没有必要。

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)
```

要计算前馈，只需使用所需的电动机速度和加速度调用 calculate () 方法即可：

备注： 可以在 calculate () 调用中省略加速度参数，如果是，则默认为零。如果没有明确定义的加速度设定值，则应执行此操作。

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)
```

使用前馈控制机制

备注： 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.

前馈控制可以完全单独使用，而无需反馈控制器。这就是所谓的"开环"控制，对于许多机构（尤其是机器人驱动器），都可以令人满意。可以使用"SimpleMotorFeedforward''来控制机器人驱动器，如下所示：

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 前馈与 PID 控制相结合

备注: 本文介绍了结合前馈/ PID 控制和 WPILib 提供的库类的代码内实现。以及详细描述所涉及概念的文档。

前馈和反馈控制器可以单独使用，但组合在一起时最有效。值得庆幸的是，将这两种控制方法结合起来 * 非常 * 简单明了-只需将它们的输出相加即可。

将前馈与 PIDController 一起使用

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)
```

此外，前馈是与反馈完全独立的功能，因此没有理由在同一控制器对象中进行处理，因为这违反了关注点分离。WPILib 带有多个帮助程序类，可以为常见的 FRC | reg |计算准确的前馈电压。机制-有关更多信息，请参阅:ref:*docs/software/advanced-controls/controllers/feedforward:Feedforward Control in WPILib*.

通过 PID 使用前馈组件

备注: 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.

前馈/ PID 组合控制的更完整示例是什么样子？考虑前馈页面上的驱动器示例 <docs / software / advanced-controls / controllers / feedforward：将前馈用于控制机制 >。我们可以很容易地修改它以包括反馈控制（带有"SimpleMotorFeedforward"组件）：

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)
    )
```

其他机制类型也可以类似地处理。

32.6.4 WPILib 中的梯形运动轮廓

备注： 本文介绍了梯形运动剖面的代码生成。文档将更详细地描述所涉及的概念。

备注： 有关在基于命令的框架:ref:*command-based framework <docs/software/commandbased/what-is-command-based:What Is "Command-Based" Programming?>* ，请参阅*通过 TrapezoidProfileSubsystems 和 TrapezoidProfileCommands 进行运动分析*.

备注： 单独使用的“TrapezoidProfile”类在与自定义控制器（例如具有内置 PID 功能的“智能”电机控制器）组合使用时最有用。要将其与 WPILib 的 PIDController 集成，请参阅*将运动分析和 PID 控制与 ProfiledPIDController 相结合*.

尽管前馈和反馈控制提供了实现给定设定值的便捷方法，但我们仍然经常面临为我们的机构生成设定值的问题。尽管天真地立即将某种机制控制到所需状态的方法可能会起作用，但它通常不是最佳的。为了改善对机制的处理，我们通常希望命令机制达到一定的“设定值”序列，以便在其当前状态和所需目标状态之间平滑地内插。

To help users do this, WPILib provides a `TrapezoidProfile` class (Java, C++, Python).

创建梯形轮廓

备注： 在 C ++ 中，“TrapezoidProfile”类在用于距离测量的单位类型（可能是角度的或线性的）上模板化。传入的值 * 必须 * 具有与距离单位一致的单位，否则将引发编译时错误。有关 C ++ 单元的更多信息，请参阅：docs / software / basic-programming / cpp-units：C ++ 单元库。

约束条件

备注: 各种前馈帮助类*feedforward helper classes* 提供了用于计算最大同时可达到的速度和机构加速度的方法。这些对于为您的 “TrapezoidProfile’’ 计算适当的运动约束非常有用。

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)
```

起始和结束状态

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)
```

汇总

备注： C ++ 通常能够推断内部类的类型，因此可以将简单的初始化列表（没有类名）作为参数发送。为了清楚起见，完整的类名包含在下面的示例中。

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))
```

使用 “TrapezoidProfile“

采样配置文件

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))
```

使用状态

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)
```

完整用法示例

备注: 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

```java
package edu.wpi.first.wpilibj.examples.elevatortrapezoidprofile;

import edu.wpi.first.math.controller.SimpleMotorFeedforward;
import edu.wpi.first.math.trajectory.TrapezoidProfile;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;

public class Robot extends TimedRobot {
  private static double kDt = 0.02;

  private final Joystick m_joystick = new Joystick(1);
  private final ExampleSmartMotorController m_motor = new␣
↪ExampleSmartMotorController(1);
  // Note: These gains are fake, and will have to be tuned for your robot.
  private final SimpleMotorFeedforward m_feedforward = new SimpleMotorFeedforward(1,␣
↪1.5);

  // Create a motion profile with the given maximum velocity and maximum
  // acceleration constraints for the next setpoint.
  private final TrapezoidProfile m_profile =
      new TrapezoidProfile(new TrapezoidProfile.Constraints(1.75, 0.75));
  private TrapezoidProfile.State m_goal = new TrapezoidProfile.State();
  private TrapezoidProfile.State m_setpoint = new TrapezoidProfile.State();

  @Override
  public void robotInit() {
    // Note: These gains are fake, and will have to be tuned for your robot.
    m_motor.setPID(1.3, 0.0, 0.7);
  }

  @Override
  public void teleopPeriodic() {
    if (m_joystick.getRawButtonPressed(2)) {
      m_goal = new TrapezoidProfile.State(5, 0);
    } else if (m_joystick.getRawButtonPressed(3)) {
      m_goal = new TrapezoidProfile.State();
    }

    // Retrieve the profiled setpoint for the next timestep. This setpoint moves
    // toward the goal while obeying the constraints.
    m_setpoint = m_profile.calculate(kDt, m_setpoint, m_goal);

    // Send setpoint to offboard controller PID
    m_motor.setSetpoint(
        ExampleSmartMotorController.PIDMode.kPosition,
        m_setpoint.position,
        m_feedforward.calculate(m_setpoint.velocity) / 12.0);
  }
}
```

C++

```
#include <numbers>

#include <frc/Joystick.h>
#include <frc/TimedRobot.h>
#include <frc/controller/SimpleMotorFeedforward.h>
#include <frc/trajectory/TrapezoidProfile.h>
#include <units/acceleration.h>
#include <units/length.h>
#include <units/time.h>
#include <units/velocity.h>
#include <units/voltage.h>

#include "ExampleSmartMotorController.h"

class Robot : public frc::TimedRobot {
 public:
  static constexpr units::second_t kDt = 20_ms;

  Robot() {
    // Note: These gains are fake, and will have to be tuned for your robot.
    m_motor.SetPID(1.3, 0.0, 0.7);
  }

  void TeleopPeriodic() override {
    if (m_joystick.GetRawButtonPressed(2)) {
      m_goal = {5_m, 0_mps};
    } else if (m_joystick.GetRawButtonPressed(3)) {
      m_goal = {0_m, 0_mps};
    }

    // Retrieve the profiled setpoint for the next timestep. This setpoint moves
    // toward the goal while obeying the constraints.
    m_setpoint = m_profile.Calculate(kDt, m_setpoint, m_goal);

    // Send setpoint to offboard controller PID
    m_motor.SetSetpoint(ExampleSmartMotorController::PIDMode::kPosition,
                        m_setpoint.position.value(),
                        m_feedforward.Calculate(m_setpoint.velocity) / 12_V);
  }

 private:
  frc::Joystick m_joystick{1};
  ExampleSmartMotorController m_motor{1};
  frc::SimpleMotorFeedforward<units::meters> m_feedforward{
      // Note: These gains are fake, and will have to be tuned for your robot.
      1_V, 1.5_V * 1_s / 1_m};

  // Create a motion profile with the given maximum velocity and maximum
  // acceleration constraints for the next setpoint.
  frc::TrapezoidProfile<units::meters> m_profile{{1.75_mps, 0.75_mps_sq}};
  frc::TrapezoidProfile<units::meters>::State m_goal;
  frc::TrapezoidProfile<units::meters>::State m_setpoint;
};

#ifndef RUNNING_FRC_TESTS
```

(续下页)

(接上页)

```
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)

        # 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 将运动分析和 PID 控制与 ProfiledPIDController 相结合

备注: 有关在基于命令的框架:ref:*command-based framework <docs/software/commandbased/what-is-command-based:What Is "Command-Based" Programming?>* framework，请参阅*在基于指令中将运动分析和 PID 结合*.

在上一篇文章中，我们看到了如何使用 TrapezoidProfile 类来创建和使用梯形运动轮廓。该文章中的示例代码演示了如何使用“智能”电机控制器的外部 PID 控制功能手动组合“TrapezoidProfile”类。

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.

使用 ProfiledPIDController 类

备注: 在 C ++ 中，“ProfiledPIDController”类在用于距离测量的单位类型（可能是角度的或线性的）上模板化。传入的值 * 必须 * 具有与距离单位一致的单位，否则将引发编译时错误。有关 C ++ 单元的更多信息，请参阅：docs / software / basic-programming / cpp-units：C ++ 单元库。

备注: ProfiledPIDController 的许多功能实际上与 PIDController 的功能相同。因此，本文将仅涵盖经过实质性更改以适应运动分析功能的功能。有关标准“PIDController’’功能的信息，请参阅*WPILib 中的 PID 控制*.

构造一个 ProfiledPIDController

备注: C ++ 通常能够推断内部类的类型，因此可以将简单的初始化列表（没有类名）作为参数发送。为了清楚起见，完整的类名包含在下面的示例中。

创建“ProfiledPIDController’’几乎与创建 PIDController *creating a PIDController*. 完全相同。唯一的区别是需要提供一组梯形轮廓约束 <docs/software/advanced-controls/controllers/trapezoidal-profiles:Constraints>‘, 这些约束将自动转发到内部生成的“TrapezoidProfile“实例：

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))
```

目标与设定点

标准 “PIDController’’ 和 “ProfiledPIDController’’ 之间的主要区别在于控制环的实际 * 设置点 * 不是由用户直接指定的。而是由用户指定目标位置或状态，并根据当前状态和目标之间生成的运动曲线自动计算控制器的设定点。因此，尽管用户端调用看起来大致相同：

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))
```

指定的“目标”值（如果需要非零速度，则可以是位置值或“TrapezoidProfile.State”）* 不一定 * 一定是回路的 * 当前 * 设定值-而是一旦生成的配置文件终止，*最终 * 设定点。

获取/使用设定值

由于 ProfiledPIDController 目标不同于设定值，因此通常需要轮询控制器的当前设定值（例如，获取:ref:*feedforward <docs/software/advanced-controls/controllers/combining-feedforward-feedback:Using Feedforward Components with PID>*）。这可以通过 getSetpoint（）方法来完成。

然后，可以将返回的设定值用于以下示例：

JAVA

```
double lastSpeed = 0;
double lastTime = Timer.getFPGATimestamp();

// Controls a simple motor's position using a SimpleMotorFeedforward
// and a ProfiledPIDController
public void goToPosition(double goalPosition) {
  double pidVal = controller.calculate(encoder.getDistance(), goalPosition);
  double acceleration = (controller.getSetpoint().velocity - lastSpeed) / (Timer.
↪getFPGATimestamp() - lastTime);
  motor.setVoltage(
      pidVal
      + feedforward.calculate(controller.getSetpoint().velocity, acceleration));
  lastSpeed = controller.getSetpoint().velocity;
  lastTime = Timer.getFPGATimestamp();
}
```

C++

```
units::meters_per_second_t lastSpeed = 0_mps;
units::second_t lastTime = frc2::Timer::GetFPGATimestamp();

// Controls a simple motor's position using a SimpleMotorFeedforward
// and a ProfiledPIDController
void GoToPosition(units::meter_t goalPosition) {
  auto pidVal = controller.Calculate(units::meter_t{encoder.GetDistance()},␣
↪goalPosition);
  auto acceleration = (controller.GetSetpoint().velocity - lastSpeed) /
      (frc2::Timer::GetFPGATimestamp() - lastTime);
  motor.SetVoltage(
       pidVal +
      feedforward.Calculate(controller.GetSetpoint().velocity, acceleration));
  lastSpeed = controller.GetSetpoint().velocity;
  lastTime = frc2::Timer::GetFPGATimestamp();
}
```

PYTHON

```
from wpilib import Timer
from wpilib.controller import ProfiledPIDController
from wpilib.controller import SimpleMotorFeedforward


def __init__(self):

    # Assuming encoder, motor, controller are already defined
    self.lastSpeed = 0
    self.lastTime = Timer.getFPGATimestamp()

    # Assuming feedforward is a SimpleMotorFeedforward object
    self.feedforward = SimpleMotorFeedforward(ks=0.0, kv=0.0, ka=0.0)

def goToPosition(self, goalPosition: float):

    pidVal = self.controller.calculate(self.encoder.getDistance(), goalPosition)
    acceleration = (self.controller.getSetpoint().velocity - self.lastSpeed) / (Timer.
↪getFPGATimestamp() - self.lastTime)

    self.motor.setVoltage(
        pidVal
        + self.feedforward.calculate(self.controller.getSetpoint().velocity,␣
↪acceleration))

    self.lastSpeed = controller.getSetpoint().velocity
    self.lastTime = Timer.getFPGATimestamp()
```

完整用法示例

A more complete example of `ProfiledPIDController` usage is provided in the ElevatorProfilePID example project (Java, C++, Python):

JAVA

```
package edu.wpi.first.wpilibj.examples.elevatorprofiledpid;

import edu.wpi.first.math.controller.ElevatorFeedforward;
import edu.wpi.first.math.controller.ProfiledPIDController;
import edu.wpi.first.math.trajectory.TrapezoidProfile;
import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;

@SuppressWarnings("PMD.RedundantFieldInitializer")
public class Robot extends TimedRobot {
  private static double kDt = 0.02;
  private static double kMaxVelocity = 1.75;
  private static double kMaxAcceleration = 0.75;
  private static double kP = 1.3;
  private static double kI = 0.0;
  private static double kD = 0.7;
  private static double kS = 1.1;
  private static double kG = 1.2;
  private static double kV = 1.3;

  private final Joystick m_joystick = new Joystick(1);
  private final Encoder m_encoder = new Encoder(1, 2);
  private final PWMSparkMax m_motor = new PWMSparkMax(1);

  // Create a PID controller whose setpoint's change is subject to maximum
  // velocity and acceleration constraints.
  private final TrapezoidProfile.Constraints m_constraints =
      new TrapezoidProfile.Constraints(kMaxVelocity, kMaxAcceleration);
  private final ProfiledPIDController m_controller =
      new ProfiledPIDController(kP, kI, kD, m_constraints, kDt);
  private final ElevatorFeedforward m_feedforward = new ElevatorFeedforward(kS, kG,␣
↪kV);

  @Override
  public void robotInit() {
    m_encoder.setDistancePerPulse(1.0 / 360.0 * 2.0 * Math.PI * 1.5);
  }

  @Override
  public void teleopPeriodic() {
    if (m_joystick.getRawButtonPressed(2)) {
      m_controller.setGoal(5);
    } else if (m_joystick.getRawButtonPressed(3)) {
      m_controller.setGoal(0);
    }

    // Run controller and update motor output
```

(续下页)

(接上页)

```
    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);
    }

    // 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;
```

(续下页)

(接上页)

```
  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)

        # 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

> **警告:** 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 WPILib 的轨迹生成和跟踪

本节描述了 WPILib 支持，用于生成参数化样条曲线轨迹，并使用典型的 FRC 机器人驱动来遵循这些轨迹。

32.7.1 轨迹生成

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.

本文介绍了如何生成轨迹。本系列的下几篇文章将介绍如何实际遵循生成的轨迹。在深入研究轨迹世界之前，您的机器人必须具备一些条件：

- 一种测量机器人每一边的位置和速度的方法。编码器是实现这一点的最佳方式；然而，其他方法可能包括光流量传感器等。
- 一种测量机器人底盘的角度或角速度的方法。陀螺仪是做到这一点的最好方法。虽然角速率可以计算使用编码器速度，但这种方法不推荐，因为车轮会刮磨。

If you are looking for a simpler way to perform autonomous navigation, see *the section on driving to a distance*.

样条函数

样条曲线是指在点之间插入的一组曲线。可以将其视为连接点，但曲线除外。WPILib 支持两种类型的样条：艾尔米特三次方和艾尔米特五次方。

- 艾尔米特三次方：对于大多数用户，建议使用此选项。使用这些样条线生成轨迹涉及指定所有点的（x，y）坐标以及起点和终点的航向。内部航路点的航向会自动确定，以确保始终保持连续的曲率（航向的变化率）。
- 艾尔米特五次方：这是一个更高级的选项，要求用户为所有航路点指定（x，y）坐标和航向。如果对艾尔米特三次方样条曲线生成的轨迹不满意，或者希望对内部点的航向进行更好的控制，则应使用此方法。

样条线用作生成轨迹的工具；但是，样条本身没有有关速度和加速度的任何信息。因此，不建议您直接使用样条线类。为了生成具有速度和加速度的平滑路径，必须生成轨迹。

创建轨迹配置

必须创建配置才能生成轨迹。该配置除了开始速度和结束速度外，还包含有关特殊约束，最大速度，最大加速度的信息。该配置还包含有关是否应该反转轨迹（机器人沿航路点向后移动）的信息。应该使用“TrajectoryConfig”类来构造配置。此类的构造函数采用两个参数，即最大速度和最大加速度。创建对象时，其他字段（`startVelocity`，`endVelocity`，`reversed`，`constraints`）默认为合理值（`0`，`0`，`false`，`{}`）。如果你想修改任何这些字段的值，则可以调用以下方法：

- `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++）

备注： “reversed’’ 属性仅表示机器人是否向后移动。如果您指定了四个航路点 a，b，c 和 d，则当“reversed”标记设置为“true”时，机器人仍将以相同的顺序通过航路点。这也意味着在提供航路点时必须考虑机器人的方向。例如，如果您的机器人正对着联盟站墙并向后移动到某个场地元素，则起始航路点应旋转 180 度。

生成轨迹

用于生成轨迹的方法是“generateTrajectory（…）“。此方法有四个重载。其中两个使用夹紧的三次样条曲线，另外两个使用五次样条曲线。对于每种类型的样条，都有两种构造轨迹的方法。最简单的方法是接受“Pose2d”对象的重载。

对于钳制三次样条曲线，此方法接受两个“Pose2d”对象，一个对象为起点，另一个为终点。该方法接受代表内部航路点的“Translation2d’’ 对象的向量。这些内部航路点的航向是自动确定的，以确保连续的曲率。对于五次样条曲线，该方法仅获取“Pose2d”对象的列表，每个“Pose2d”代表一个点并指向该字段。

更复杂的重载接受样条曲线的“控制向量”。在使用 Pathweaver 生成轨迹时可以使用此方法，在该轨迹中，您可以控制每个点处切线向量的大小。ControlVector 类由两个 double 数组组成。每个数组表示一个维度（x 或 y），其元素表示该点的导数。例如，“x’’ 数组的元素 0 处的值表示 x 坐标（第 0 导数），元素 1 处的值表示 x 维度上的一阶导数，依此类推。

使用钳制三次样条曲线时，数组的长度必须为 2（0 和 1 的导数），而使用五次样条曲线时，数组的长度应为 3（0、1 和 2 的导数）。除非您确切地知道自己在做什么，否则强烈建议您使用第一种更简单的方法来手动生成轨迹。（即不使用 Pathweaver JSON 文件时）。

这是为 2018 年游戏 FIRST Power Up 使用艾尔米特的三次方生成轨迹的示例：

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
    )
```

备注： The Java code utilizes the Units utility, for easy unit conversions.

备注： 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.

警告： 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)),
```

(续下页)

(接上页)

```
   new TrajectoryConfig(Units.feetToMeters(3.0), Units.feetToMeters(3.0)));

var concatTraj = trajectoryOne.concatenate(trajectoryTwo);
```

C++

```
auto trajectoryOne = frc::TrajectoryGenerator::GenerateTrajectory(
   frc::Pose2d(0_m, 0_m, 0_rad),
   {frc::Translation2d(1_m, 1_m), frc::Translation2d(2_m, -1_m)},
   frc::Pose2d(3_m, 0_m, 0_rad), frc::TrajectoryConfig(3_fps, 3_fps_sq));

auto trajectoryTwo = frc::TrajectoryGenerator::GenerateTrajectory(
   frc::Pose2d(3_m, 0_m, 0_rad),
   {frc::Translation2d(4_m, 4_m), frc::Translation2d(5_m, 3_m)},
   frc::Pose2d(6_m, 0_m, 0_rad), frc::TrajectoryConfig(3_fps, 3_fps_sq));

auto concatTraj = m_trajectoryOne + m_trajectoryTwo;
```

PYTHON

```
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 轨迹约束

在:ref:‘previous article <docs/software/advanced-controls/trajectories/trajectory-generation:Generating the trajectory>‘中，您可能已经注意到在生成轨迹时没有添加自定义约束。自定义约束允许用户基于位置和曲率对轨迹上点的速度和加速度施加更多的限制。

例如，自定义约束可以将轨迹的速度保持在某一区域的某一阈值以下，或者在转弯附近使机器人减速以达到稳定的目的。

WPILib-Provided 约束

WPILib 包括一组预定义的约束，用户在生成轨迹时可以利用这些约束。WPILib 提供的约束列表如下：

- “CentripetalAccelerationConstraint”：限制机器人沿轨迹行进时的向心加速度。这可以帮助机器人在急转弯时减速。
- “DifferentialDriveKinematicsConstraint”：限制转弯时机器人的速度，以使差速驱动机器人的车轮都不会超过指定的最大速度。
- “DifferentialDriveVoltageConstraint”：限制差动驱动机器人的加速度，以使命令电压不超过指定的最大值。
- “EllipticalRegionConstraint”：仅在赛场上的椭圆区域中施加约束。
- “MaxVelocityConstraint”：施加最大速度约束。这可以与 “EllipticalRegionConstraint” 或 “RectangularRegionConstraint” 组成，以仅在特定区域限制机器人的速度。
- “MecanumDriveKinematicsConstraint”：限制转弯时机器人的速度，以使麦克纳姆驱动机器人的车轮都不会超过指定的最大速度。
- “RectangularRegionConstraint”：仅在字段的矩形区域中施加约束。
- “SwerveDriveKinematicsConstraint”：限制转弯时机器人的速度，以使转弯驱动机器人的车轮都不会超过指定的最大速度。

备注： “DifferentialDriveVoltageConstraint” 只确保使用:ref:‘feedforward model <docs/software/advanced-controls/controllers/feedforward:SimpleMotorFeedforward>‘的理论电压命令不会超过指定的最大值。如果机器人在跟踪过程中偏离基准，指令电压可能会高于指定的最大电压。

创建自定义约束

用户可以通过实现 “TrajectoryConstraint” 接口来创建自己的约束。

JAVA

```
@Override
public double getMaxVelocityMetersPerSecond(Pose2d poseMeters, double␣
↪curvatureRadPerMeter,
                                            double velocityMetersPerSecond) {
  // code here
}

@Override
public MinMax getMinMaxAccelerationMetersPerSecondSq(Pose2d poseMeters,
                                                     double curvatureRadPerMeter,
                                                     double velocityMetersPerSecond) {
  // code here
}
```

C++

```
units::meters_per_second_t MaxVelocity(
const Pose2d& pose, units::curvature_t curvature,
units::meters_per_second_t velocity) override {
  // code here
}

MinMax MinMaxAcceleration(const Pose2d& pose, units::curvature_t curvature,
                          units::meters_per_second_t speed) override {
  // code here
}
```

PYTHON

```
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:
        ...
```

MaxVelocity 方法应返回给定姿势，曲率和轨迹的原始速度的最大允许速度，且无任何限制。MinMaxAcceleration"方法应返回给定姿势，曲率和约束速度的最小和最大允许加速度。

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 操纵轨迹

生成轨迹后，您可以使用某些方法从中检索信息。在编写代码以追踪这些轨迹时，这些方法将非常有用。

获取轨迹的总持续时间

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()
```

轨迹采样

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)
```

Trajectory::Sample 结构包含有关采样点的几条信息：

- t：从轨迹开始到采样点为止的时间。
- velocity：采样点的速度。
- acceleration：采样点的加速度。
- pose：采样点的姿态 (x, y, heading)
- “曲率”：采样点的曲率（航向的变化率，相对于沿着轨迹的距离）。

注意：可以通过将速度乘以曲率来计算采样点的角速度。

获取轨迹的所有状态（高级）

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 转变轨迹

可以使用 “relativeTo” 和 “transformBy” 方法将轨迹从一个坐标系转换为另一个坐标系，并在坐标系内移动。这些方法对于在空间内移动轨迹或在另一个参考系中重新定义已经存在的轨迹很有用。

备注： 这些方法都不能改变原始轨迹的形状。

“relativeTo“方法

“relativeTo“方法用于在另一个参考系中重新定义一个已经存在的轨迹。该方法采用一个参数：一个姿势（通过 “Pose2d” 对象），该姿势是相对于当前坐标系定义的，代表新坐标系的原点。

例如，可以使用 “relativeTo’’ 方法在坐标系 B 中重新定义在 A 坐标系中定义的轨迹，该坐标系的原点在坐标系 A 中为（2、2、30 度）。

JAVA

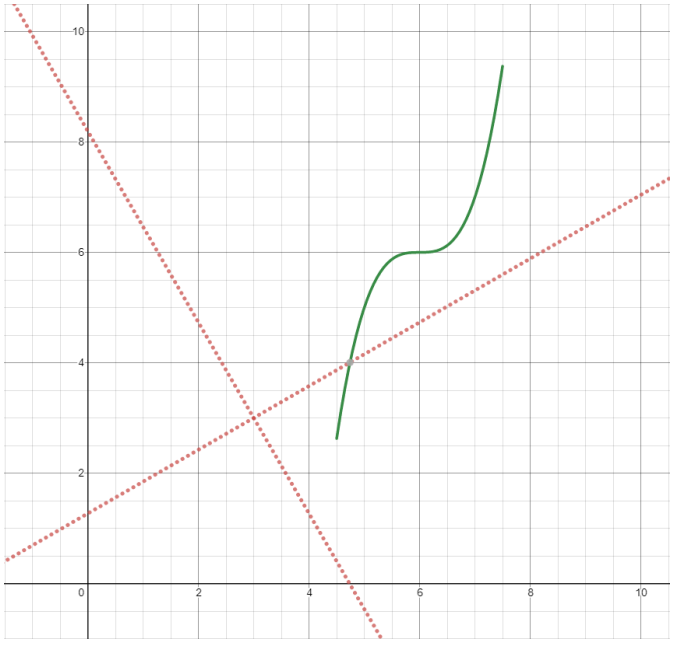
```
Pose2d bOrigin = new Pose2d(3, 3, Rotation2d.fromDegrees(30));
Trajectory bTrajectory = aTrajectory.relativeTo(bOrigin);
```

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)
```



在上图中，原始轨迹（上面代码中的 “aTrajectory“）已在坐标系 A 中定义，由黑轴表示。相对于原始坐标系位于（2,2）和 30° 的红色轴代表坐标系 B. 在 “aTrajectory“上调用 “relativeTo“ 将重新定义轨迹中的所有姿势，使其相对于坐标系 B（红色轴）。

transformBy 方法

可以使用 “transformBy’’ 方法在坐标系内移动（即平移和旋转）轨迹。此方法有一个参数：一个转换（通过 “Transform2d” 对象），该转换将轨迹的当前初始位置映射到同一轨迹的所需初始位置。

例如，您可能想使用 “transformBy’’ 方法将以（2、2、30 度）开始的轨迹转换为以（4、4、50 度）开始的轨迹。

JAVA

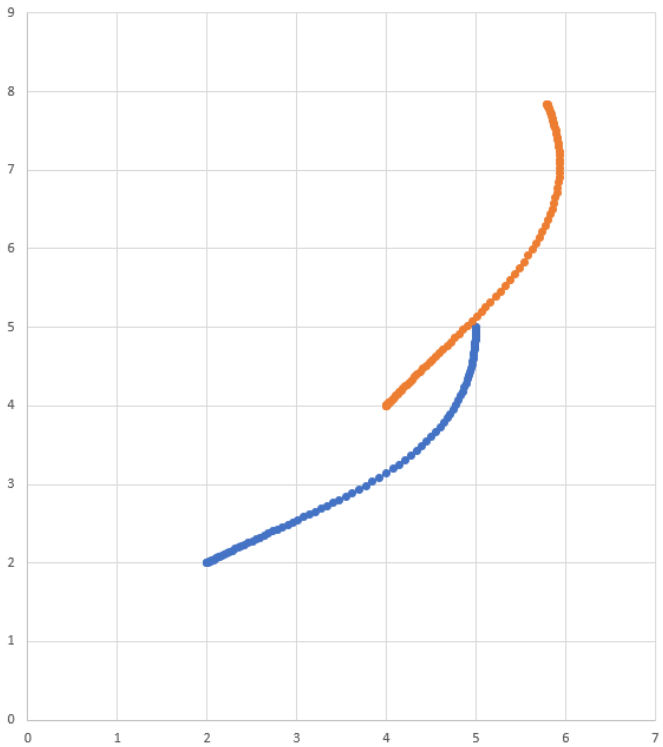
```
Transform2d transform = new Pose2d(4, 4, Rotation2d.fromDegrees(50)).minus(trajectory.
↪getInitialPose());
Trajectory newTrajectory = trajectory.transformBy(transform);
```

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)
```



在上图中，以（2，2）和 30° 开始的原始轨迹显示为蓝色。在应用上面的变换之后，合成轨迹的起始位置在 50° 处更改为（4，4）。生成的轨迹以橙色显示。

32.7.5 斜坡控制器

Ramsete 控制器是 WPILib 内置的轨迹跟踪器。该跟踪器可用于通过对较小干扰的校正来准确跟踪轨迹。

构造 Ramsete 控制器对象

Ramsete 控制器应该初始化为两个增益，即’‘b ‘‘和’‘zeta ‘‘。较大的值’‘b ‘‘使收敛更猛烈，就像一个比例项，而较大的值’‘zeta ‘‘在响应中提供更多的阻尼。这些控制器增益只决定控制器将如何输出调整的速度。它不影响机器人的实际速度跟踪。这意味着这些控制器增益通常与机器人无关。

备注： 当所有单位均以米为单位时，对 b 和 zeta 的增益分别为 2.0 和 0.7 进行了反复测试，以产生理想的结果。这样，“RamseteController’’ 的零参数构造函数就存在，其增益默认为这些值。

JAVA

```
// Using the default constructor of RamseteController. Here
// the gains are initialized to 2.0 and 0.7.
RamseteController controller1 = new RamseteController();

// Using the secondary constructor of RamseteController where
// the user can choose any other gains.
RamseteController controller2 = new RamseteController(2.1, 0.8);
```

C++

```
// Using the default constructor of RamseteController. Here
// the gains are initialized to 2.0 and 0.7.
frc::RamseteController controller1;

// Using the secondary constructor of RamseteController where
// the user can choose any other gains.
frc::RamseteController controller2{2.1, 0.8};
```

PYTHON

```
from wpimath.controller import RamseteController

# Using the default constructor of RamseteController. Here
# the gains are initialized to 2.0 and 0.7.
controller1 = RamseteController()

# Using the secondary constructor of RamseteController where
# the user can choose any other gains.
controller2 = RamseteController(2.1, 0.8)
```

调整速度

Ramsete 控制器返回“已调整的速度”，以便当机器人跟踪这些速度时，它可以精确地达到目标点。应使用新目标定期更新控制器。目标包括期望的姿势，期望的线速度和期望的角速度。此外，机器人的当前位置也应定期更新。控制器使用这四个参数返回调整后的线速度和角速度。用户应命令机器人以这些线性和角速度实现最佳轨迹跟踪。

备注： 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)
```

这些计算应在每次循环迭代时执行，并具有更新的机器人位置和目标

使用调整后的速度

调整后的速度为“ChassisSpeeds’’类型，其中包含“vx’’（向前方向的线速度），“vy’’（横向方向的线速度）和“omega’’‘（围绕机器人框架中心的角速度）。由于 Ramsete 控制器是用于非完整机器人（不能侧向移动的机器人）的控制器，因此调整后的速度对象的“vy’’为零。

可以使用适用于您的传动系统类型的运动学类将返回的调整速度转换为可用速度。例如，可以使用“DifferentialDriveKinematics’’对象将差速器的调整速度转换为左右速度。

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.

基于命令的框架中的 Ramsete

For the sake of ease for users, a `RamseteCommand` class is built in to WPILib. For a full tutorial on implementing a path-following autonomous using RamseteCommand, see *Trajectory Tutorial*.

32.7.6 全向驱动控制器

全向驱动器控制器是一种轨迹跟踪器，适用于具有全向传动系统（例如，swerve 底盘，麦克纳姆轮等）的机器人。可以使用它来校正微小的扰动，从而准确地跟踪轨迹。

构造全向驱动控制器

全向驱动控制器应使用 2 个 PID 控制器和 1 个简略 PID 控制器实例化。

备注： 有关 PID 控制的更多信息，请参见*WPILib 中的 PID 控制*。

---

2 个 PID 控制器是分别应校正相对于磁场的 x 和 y 方向误差的控制器。例如，如果前两个参数分别是“PIDController(1, 0, 0)“ 和 “PIDController(1.2, 0, 0)“，则完整驱动控制器将在 x 方向上为 x 方向上每米的误差增加每秒 1 米的附加误差。在 y 方向上为 y 方向上每米的误差增加每秒 1.2 米的附加误差。

最后一个参数是用于机器人旋转的“ProfiledPIDController”。由于完整传动系统的旋转动力学在 x 和 y 方向上与运动解耦，用户可以设置自定义的航向参考，同时遵循轨迹。这些标题引用是根据“ProfiledPIDController“中的参数设置的概要。

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.
```

调整速度

完整的驱动控制器返回“已调整的速度”，以便当机器人跟踪这些速度时，它可以精确地达到目标点。应使用新目标定期更新控制器。目标包括所需的姿势，线速度和航向。

备注： 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)
)
```

使用调整后的速度

调整后的速度为 “ChassisSpeeds’’ 类型，其中包含 “vx’’（向前方向的线速度），“vy’’（横向方向的线速度）和 “omega’’（围绕机器人框架中心的角速度）。

可以使用适用于您的传动系统类型的运动学类将返回的调整速度转换为可用速度。在下面的示例代码中，我们将假定一个弧形驱动机器人；但是，除了使用 “MecanumDriveKinematics’’ 外，运动学代码与麦克纳姆驱动机器人完全相同。

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)
```

由于这些转弯模块状态仍然是速度和角度，因此您将需要使用 PID 控制器来设置这些速度和角度。

32.7.7 故障排除

排除完全故障

有很多事情会导致你的机器人做完全错误的事情。下面的清单包含了一些常见的错误。

- 我的机器人不动了
  - 你真的在输出给你的马达输出吗?
  - driver station 是否出现了 MalformedSplineException? 如果是，请转到下面有关 MalformedSplineException 的部分。
  - 您的轨迹是否太短或有单位错误?
- 我的机器人沿着轨迹移动却不停摆动，朝向另一个方向。
  - 轨迹的起点和终点的朝向是否有错误?
  - 你的机器人陀螺仪是否重置了错误的方向?
  - 你是否将 reverse flag 错误设置?
  - 陀螺仪角度是设定顺时针为正吗？如果是这样，则应该反过来。

- 我的机器人只是沿着直线行驶，即使它应该转弯。
  - 你的陀螺仪是否设置正确并返回好的数据?
  - 你是否使用正确的单位将陀螺仪航向传递到里程表对象?
  - 轨道宽度正确吗？它的单位正确吗?
- 我的 driver station 显示 “MalformedSplineException”，但是机器人不动。
  - 你是否将 reverse flag 错误设置?
  - 你是否有两个航路点非常接近且朝向相反?
  - 你是否有两个具有相同（或几乎相同）坐标的航路点?
- 我的机器人移动得太远了。
  - 你的编码器单位转换设置正确吗?
  - 你的编码器连接了吗?
- 我的机器人基本上能做正确的事情，但有一点不准确。
  - 转到下一部分。

解决性能不佳的问题

备注: 本节主要涉及对轨迹跟踪性能不佳，例如一米的误差，进行故障排除，而不是像编译错误，机器人不停转向并朝错误的方向或”MalformedSplineException’’那样的灾难性故障。

备注: 本节专为差速驱动机器人的而设计，但是大多数想法都可以适用于转向驱动或麦克纳姆转向。

不良的轨迹跟踪性能可能难以排除。虽然轨迹生成器和跟随器的目的是易于使用和卓越的表现，有情况下，机器人的表现不如预期。轨迹产生器和跟随器有很多需要调节的旋钮和运动部件，因此很难知道从哪里开始，特别是很难从机器人的大致运动中找到轨迹问题的根源。

因为很难定位异常轨迹生成器和追踪器的层次，所以对于一般跟踪性能较差的情况 (如机器人偏离几英尺或超过 20 度)，建议采用一种系统的、逐层的方法。以下是你应该做的步骤的顺序；遵循这个顺序是很重要的，这样您就可以孤立不同步骤的效果。

备注: The below examples put diagnostic values onto *NetworkTables*. The easiest way to graph these values is to *use Shuffleboard’s graphing capabilities*.

核查里程表

如果您的里程表不正确，则 Ramsete 控制器可能会发生异常，因为它会根据您的里程表认为机器人的位置来修改机器人的目标速度。

备注: *Sending your robot pose and trajectory to field2d* can help verify that your robot is driving correctly relative to the robot trajectory.

1. 设置代码以记录每次里程表更新后机器人的位置:

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. 布置一条与您的机器人平行的卷尺，然后沿着卷尺将您的机器人推出约一米。沿 Y 轴布置一个卷尺并重新开始，将您的机器人沿 X 轴推一米，沿 Y 轴推一米，形成一个粗糙的弧度。

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.

核查前馈

如果前馈不良，则机器人各侧的 P 控制器也将无法跟踪，并且 differentialDriveVoltageConstraint 不会精确地限制机器人的加速度。我们主要想关闭车轮的 P 控制器，以便我们可以单独测试前馈。

1. 首先，必须为每个车轮设置禁用 P 控制器。将每个控制器的 P 增益设置为 0。在 RamseteCommand 示例中，您可以将 kPDriveVel 设置为 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. 最后，我们需要记录所需的车轮速度和实际的车轮速度（如果您使用 Shuffleboard，或者您的绘图软件具有该功能，则应将实际速度和所需速度记录在同一张图表上）

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) {
```

(续下页)

(接上页)

```
        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. 在各种轨迹（曲线和直线）上运行机器人，并通过查看 NetworkTables 中的图表来检查实际速度是否跟踪所需速度。
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.

核查 P 增益

如果您完成了上一步，并且问题消失了，那么可能会在后续步骤之一中找到您的问题。在此步骤中，我们将核查车轮的 P 控制器是否已正确调整。如果您使用的是 Java，那么我们要关闭 Ramsete，以便我们可以自己查看 PF 控制器。

1. 除了必须将 P 增益重新设置为之前的非零值之外，您必须重新使用记录实际速度与所需速度的上一步中的所有代码（如果使用 Java，则禁用 Ramsete 的代码）。
2. 在各种轨迹上再次运行机器人，并检查您的实际图形与所需图形之间是否相符。
3. 如果图形看起来不好（即实际速度与期望的速度有很大不同），则应尝试调整 P 增益并重新运行测试轨迹。

核查约束

备注： 确保此步骤的 P 增益不为零，并且仍在前面的步骤中添加了日志记录代码。如果您使用的是 Java，则应删除代码以禁用 Ramsete。

如果您的准确性问题在上述所有步骤中仍然存在，则说明您的约束可能存在问题。以下是调整不良时不同可用约束出现问题的列表。

一次测试一个约束！删除其他约束，调整剩下的一个约束，并对每个要使用的约束重复该过程。以下清单假定您一次仅使用一个约束。

- DifferentialDriveVoltageConstraint：
  - 如果您的机器人加速非常慢，则该约束的最大电压可能太低。
  - If your robot doesn't reach the end of the path then your system identification data may problematic.

- DifferentialDriveKinematicsConstraint：
  - 如果机器人的前进方向错误，则可能是最大动力传动系统侧向速度太低或太高。唯一方法是调整最大速度并查看会发生什么。
- CentripetalAccelerationConstraint：
  - 如果您的机器人的航向错误，那么这可能是罪魁祸首。如果您的机器人似乎转弯不充分，则应增加最大向心加速度，但如果机器人转弯时转弯太快，则应减小最大向心加速度。

检查轨迹航路点

您的轨迹本身可能不太容易追踪。尝试移动航路点（以及航路点的标题，如果适用），并减少急转弯。

32.8 使用 WPILib 进行基于状态空间和模型的控制

本节介绍并描述了 WPILib 对状态空间控制的支持。

32.8.1 空间状态控制简介

备注： 本文来自于 'Controls Engineering in FRC|reg| <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>'，已获得作者 Tyler Veness 的许可。

从 PID 到基于模型的控制

当调优 PID 控制器时，我们关注于摆弄与当前、过去和未来*error* (P、I 和 D 项) 相关的控制器参数，而不是底层系统状态。虽然这种方法在很多情况下都有效，但它是一种不完整的世界观。

基于模型的控制重点在于开发我们试图控制的*system* (机制) 的精确模型。这些模型帮助反馈控制器根据系统的物理响应选择 gains 1，而不是通过测试得到的任意比例*gain*。这使我们不仅能够提前预测系统的反应，而且可以在没有物理机器人的情况下测试控制器，节省调试简单 bug 的时间。

备注： 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.

词汇表

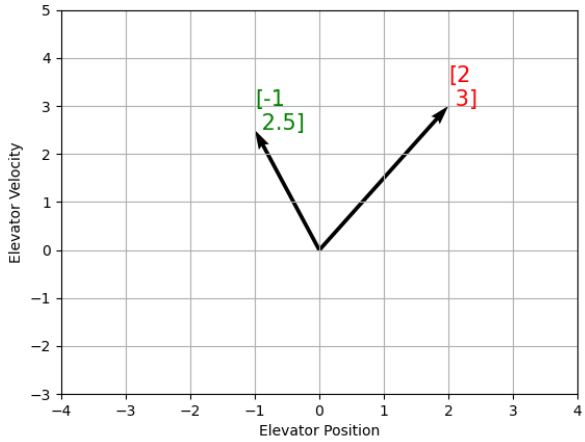
对于本文中使用的背景词汇，清参阅:ref:*Glossary <docs/software/advanced-controls/controls-glossary:Controls Glossary>*.

线性代数导论

为了简短，直观地介绍线性代数的核心概念，我们建议‘3Blue1Brown 线性代数系列 <https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab>‘__ 的第 1 章至第 4 章（向量，它们甚至是什么?，线性组合，跨度和基向量，线性变换和矩阵，以及矩阵乘法作为组合）。

什么是状态空间？

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 [position, velocity]:



在此图像中，表示状态空间中状态的向量是箭头。从现在开始，这些向量将仅由向量尖端处的一个点表示，但请记住，向量的其余部分仍然存在。

除了 state‘之外，:term:‘inputs 1 and :term:‘outputs 2‘也被表示为向量。由于从当前状态和输入到状态变化的映射是一个方程式系统，因此很自然地以矩阵形式编写它。该矩阵方程式可以用状态空间符号表示。

什么是状态空间符号？

状态空间符号是一组矩阵方程，描述了系统随着时间的变化。这些方程将状态 $\dot{\mathbf{x}}$‘□ : *term* : ‘*output* **y**‘□□□□□□□□□□□□□□□□□□□□□ : *math* :mathbf{x}‘ and *input* vector **u**。

状态空间控制可以处理连续时间和离散时间系统。在连续时间情况下，系统状态:math:*mathbf{dot{x}}*‘的变化率表示为当前状态:*math:*mathbf{x}‘ 和输入:math:‘mathbf{u}‘的线性组合。

相反，离散时间系统基于当前状态 $\mathbf{x}_k$‘□□□□□□□□□□□□□ : *math* :mathbf{x}_{k+1}‘和输入:math:*mathbf{u}_k*，其中 k‘□□□□□□□□ : *math* : ‘$k+1$ 是下一个时间步。

无论是连续时间形式还是离散时间形式，*output* 向量:math:*mathbf{y}* 都表示为当前:term:*state* 和:term:‘input‘的线性组合。在许多情况下，输出是系统状态的子集，没有来自当前输入的贡献。

在对系统进行建模时，我们首先导出连续时间表示形式，因为运动方程很自然地写为系统状态的变化率，即系统当前状态和输入的线性组合。我们将这种表示形式转换为机器人上的离散时间，因为我们在此以离散时间步长而不是连续地更新系统。

以下两组方程式是连续时间和离散时间状态空间符号的标准形式：

$$\text{Continuous: } \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$$\text{Discrete: } \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$
$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k$$

A	system matrix	**x**	state vector
B	input matrix	**u**	input vector
C	output matrix	**y**	output vector
D	feedthrough matrix		

连续时间状态空间系统可以通过称为离散化的过程转换为离散时间系统。

备注： 在离散时间形式中，系统状态在两次更新之间保持不变。这意味着我们只能在状态估计值更新后尽快对干扰做出反应。更快地更新我们的估计值可以在一定程度上帮助改善性能。如果需要比机械手主循环更快的更新速度，则可以使用 WPILib 的 “Notifier” 类。

备注： 虽然系统的连续时间矩阵和离散时间矩阵 A，B，C 和 D 具有相同的名称，但它们并不等效。连续时间矩阵描述状态的变化率:math:*mathbf{x}*，而离散时间矩阵描述系统在下一时间步的状态是当前状态和输入的函数。

重要： WPILib 的 LinearSystem 采用连续时间系统矩阵，并在必要时将其内部转换为离散时间形式。

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.

我们可以将该方程转换为状态空间符号。我们可以创建一个具有一个状态（速度），一个*input*（电压）和 output‘（速度）的系统。回想一下速度的一阶导数是加速度，我们可以编写如下公式，用:math:mathbf{x}‘代替速度，用 **x**‘□□□□□□□□□□ : *math* :mathbf{u}‘代替电压:math:*mathbf{V}* ：

$$\mathbf{x} = \left[\frac{-K_v}{K_a}\right]\mathbf{x} + \left[\frac{1}{K_a}\right]\mathbf{u}$$

The output and state are the same, so the output equation is the following:

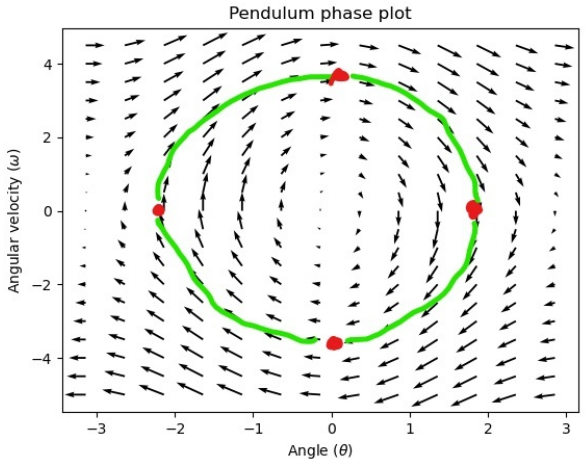
$$\mathbf{y} = \left[1\right]\mathbf{x} + \left[0\right]\mathbf{u}$$

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.

可视化状态空间响应：相位图

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.

要追踪系统可能会穿越状态空间的潜在轨迹，请选择一个起点并遵循周围的箭头。在此示例中，我们可能从:math:‘[-2, 0]‘开始。从那里开始，速度随着我们垂直摆动而增加，并开始降低，直到达到相反的极端。围绕原点旋转的循环无限期地重复。



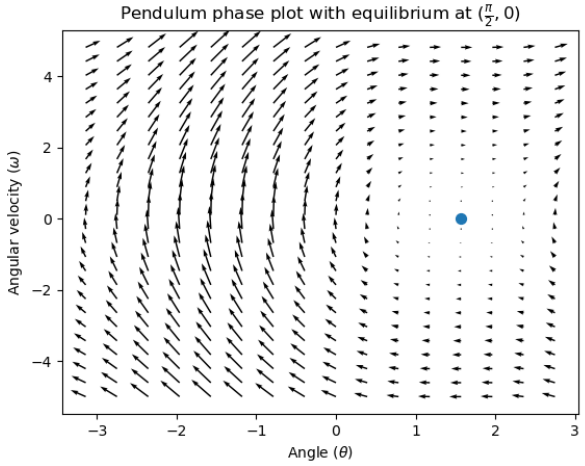
注意，在相像的边缘附近，X 轴绕着:math:*pi* 弧度逆时针旋转，并且:math:*pi* 弧度顺时针旋转将在同一点结束。

有关微分方程和相画像的更多信息，请参见 ‘3Blue1Brown’ s Differential Equations video <https://www.youtube.com/watch?v=p_di4Zn4wz4>‘，它们在 15:30 左右为摆相空间设置动画效果非常出色。

可视化前馈

这个阶段描述显示了系统的“开环”响应——也就是说，如果我们让状态自然发展，它将如何反应。例如，如果要平衡水平摆（在状态空间中的:math:*(frac{pi}{2}, 0)*‘处），我们将需要以某种方式应用控制:*term:‘input* 以抵消摆向下摆动的开环趋势。这就是前馈正在尝试做的——以使我们的相位图在状态空间中的*reference* 位置（或设定点）处具有平衡。

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 $(\frac{\pi}{2}, 0)$:



反馈控制

在直流电动机的情况下，只需要一个数学模型和系统的所有当前状态 (即角速度) 的知识，我们就可以根据未来的电压输入预测所有未来的状态。但是，如果系统受到我们的方程所没有模拟的任何干扰，比如负载或意外摩擦，电机的角速度会随着时间的推移偏离模型。为了解决这个问题，我们可以使用反馈控制器给电机校正命令。

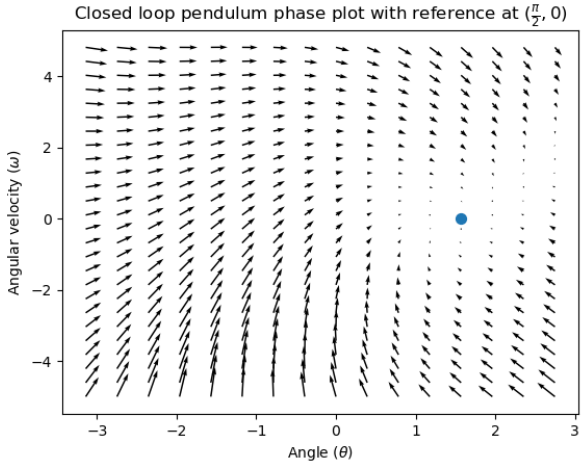
PID 控制器是反馈控制的一种形式。状态空间控制通常使用以下控制法则，其中 $\mathbf{K}$ 是一些控制器：term：*gain‘*矩阵，*:math:*mathbf{r}‘ 是*reference* 状态，而 $\mathbf{x}$ 是状态空间中的当前状态。这两个向量的差 $\mathbf{r} - \mathbf{x}$，是:term:*error*。

$$mathbf u = \ mathbf K r - x $$

This *control law* is a proportional controller for each state of our system. Proportional controllers create software-defined springs that pull our system’s state toward our reference state in state-space. In the case that the system being controlled has position and velocity states, the *control law* above will behave as a PD controller, which also tries to drive position and velocity error to zero.

让我们举例说明这种控制律的实际作用。我们将从上方使用钟摆系统，其中摆动的钟摆环绕状态空间中的原点。:math:‘mathbf{K}‘是零矩阵（一个全为零的矩阵）的情况就像选择 P 和 D 增益为零-没有控制输入将被应用，并且相画像看起来与上面的相同。

要添加一些反馈，我们可以任意选择 **K** [2, 2]，其中摆的:term:*input* 是角加速度。这个 K 意味着对于每一个位置 error‘弧度，角加速度是为 2 弧度每二次方秒；类似地，我们以 2 弧度每二次方秒的加速度对于每弧度每二次方秒的:term:‘error。尝试从状态空间中的某个地方向内跟随箭头-无论初始条件如何，状态将稳定在:term:*reference*，而不是用纯粹的前馈无休止地循环。



但是我们如何为我们的系统选择一个最优的:term:*gain‘矩阵 K？虽然我们可以手动选择：:term:‘gains 1* 并模拟系统响应或像 PID 控制器一样在机器人上对其进行调整，但现代控制理论有一个更好的答案：线性二次调节器（LQR）。

线性二次调节器

因为基于模型的控制意味着我们可以在给定初始条件和未来控制输入的情况下预测系统的未来状态，所以我们可以选择数学上最优的:term:*gain* 矩阵:math:*mathbf{K}*。要做到这一点，我们首先必须定义 “好” 或 “坏” 的:math:*mathbf{K}* 会是什么样子。我们通过将误差平方和控制输入随时间的变化相加来做到这一点，这将给我们一个数字，表示我们的控制律有多 “糟糕”。如果我们使这个和最小化，我们就得到了最优控制律。

LQR：定义

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*.

备注： LQR 设计的 **Q** 和 **R** 矩阵不需要离散化，但是为连续时间和离散时间:term:*systems 1* 计算的 **K** 将有所不同。

LQR: 调试

就像可以通过调节 PID 增益来调整 PID 控制器一样，我们也想改变控制律平衡误差和输入的方式。例如，一艘太空船可能希望将其消耗的燃料减至最小，以达到给定的参考值，而高速机械臂可能需要对干扰做出快速反应。

我们可以使用 **Q** 和 **R** 矩阵在 LQR 中加权错误并控制工作量。在我们的成本函数（描述了控制律将如何“糟糕”地执行）中，**Q** 和 **R** 分别权衡我们的误差和控制输入。在上面的飞船示例中，我们可以使用带有相对较小数字的:math:*mathbf{Q}* 来表明我们不想过度影响误差，而我们的:math:‘mathbf{R}‘可能很大，表明燃料消耗是不可取的。

使用 WPILib，LQR 类可以获取所需的最大状态偏移和控制量的向量，并根据布赖森规则在内部将它们转换为完整的 Q 和 R 矩阵。我们经常使用小写的:math:*mathbf{q}* 和 **r**‘□□□□□□□□□□ : $math$:mathbf{Q}‘和:math:‘mathbf{R}‘指代矩阵。

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.

例如，对于具有位置和速度状态的抬升系统，我们可以使用以下 Q 和 R。

JAVA

```
// Example system -- must be changed to match your robot.
LinearSystem<N2, N1, N1> elevatorSystem = LinearSystemId.identifyPositionSystem(5, 0.
↪5);
LinearQuadraticRegulator<N2, N1, N1> controller = new␣
↪LinearQuadraticRegulator(elevatorSystem,
      // q's elements
      VecBuilder.fill(0.02, 0.4),
      // r's elements
      VecBuilder.fill(12.0),
      // our dt
      0.020);
```

C++

```
// Example system -- must be changed to match your robot.
   LinearSystem<2, 1, 1> elevatorSystem =␣
↪frc::LinearSystemId::IdentifyVelocitySystem(5, 0.5);
   LinearQuadraticRegulator<2, 1> controller{
      elevatorSystem,
      // q's elements
      {0.02, 0.4},
      // r's elements
      {12.0},
      // our dt
      0.020_s};
```

PYTHON

```
from wpimath.controller import LinearQuadraticRegulator_2_1
from wpimath.system.plant import LinearSystemId


# Example system -- must be changed to match your robot.
elevatorSystem = LinearSystemId.identifyPositionSystemMeters(5, 0.5)
controller = LinearQuadraticRegulator_2_1(
   elevatorSystem,
   # q's elements
   (0.02, 0.4),
   # r's elements
   (12.0,),
   # our dt
   0.020,
)
```

LQR：示例应用

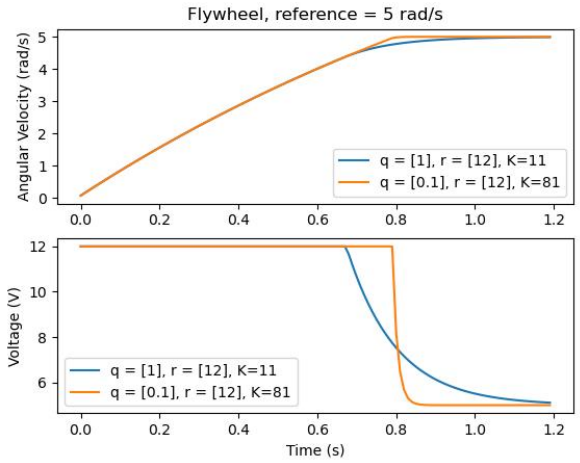
Let’s apply a Linear-Quadratic Regulator to a real-world example. Say we have a flywheel velocity system determined through system identification to have $K_v = 1 \frac{\text{volts}}{\text{radian per second}}$ and $K_a = 1.5 \frac{\text{volts}}{\text{radian per second squared}}$. Using the flywheel example above, we have the following linear *system*:

$$\mathbf{x} = \left[\frac{-K_v}{K_a}\right] v + \left[\frac{1}{K_a}\right] V$$

我们任意选择所需的状态偏移（最大误差):math:*q = [0.1text{rad/sec}]*, [12 volts]‘ :*math* :mathbf{r}‘ 。在 20ms 的时间步长下离散化后，我们得到的:term:*gain‘为:math:*mathbf{K} = ~81‘。这个:term:*gain* K 作为 PID 回路对飞轮速度的比例分量。

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**.

下图显示了飞轮的角速度和施加的电压随时间的变化，具有两个不同的*gain*。我们可以看到更高的:term:‘gain‘将如何使系统更快地到达参考点（t = 0.8 秒时），同时使我们的电机在 12V 时保持更长时间的饱和状态。这与将 PID 控制器的 P 增益增加约 8 倍完全相同。



LQR 和测量延迟补偿

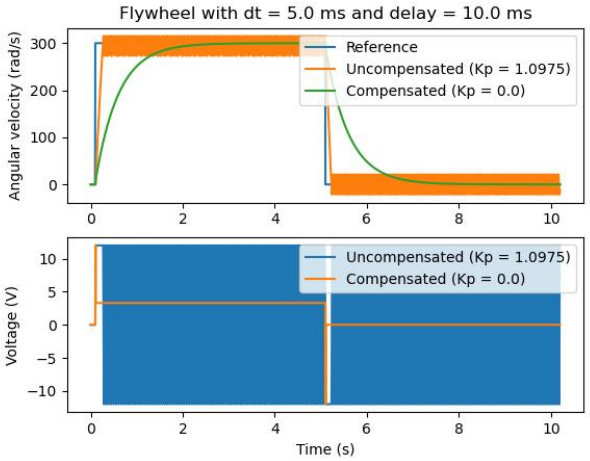
通常，我们的传感器会因测量而延迟。例如，通过 CAN 的 SPARK MAX 电动机控制器可能具有与速度测量相关的长达 30ms 的延迟。

这种滞后现象意味着我们的反馈控制器将基于过去的状态估计来生成电压命令。如下图所示，这通常会给我们的系统带来不稳定性和振荡。

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 $\mathbf{K}$, trading off controller performance for stability. The below formula, which adjusts the *gain* matrix to account for delay, is also used in system identification.

$$\mathbf{K_{compensated}} = \mathbf{K} \cdot (\mathbf{A} - \mathbf{BK})^{\text{delay}/dt}$$

Multiplying $\mathbf{K}$ by $\mathbf{A} - \mathbf{BK}$ essentially advances the gains by one timestep. In this case, we multiply by $(\mathbf{A} - \mathbf{BK})^{\text{delay}/dt}$ to advance the gains by measurement's delay.



备注： 这样可以将:math:‘mathbf{K}‘减小为零，从而有效地禁用反馈控制。

备注： SPARK Max 电机控制器使用 40 抽头 FIR 滤波器，其延迟为 19.5ms，默认情况下每 20ms 发送一次状态帧。

以下代码显示了如何针对传感器输入延迟来调整 LQR 控制器的 K 增益：

JAVA

```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++

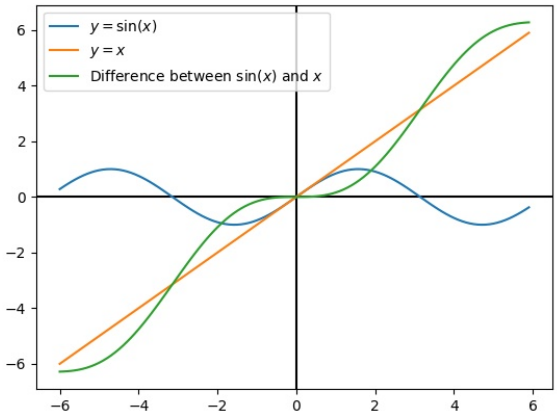
```cpp
// Adjust our LQR's controller for 25 ms of sensor input delay. We
// provide the linear system, discretization timestep, and the sensor
// input delay as arguments.
controller.LatencyCompensate(elevatorSystem, 20_ms, 25_ms);
```

PYTHON

```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)
```

线性化

线性化是一种工具，用于近似非线性函数和状态空间系统的线性处理。在二维空间中，线性函数是直线，而非线性函数是曲线。一个常见的非线性函数及其对应的线性近似例子是 $y = \sin x$，$y = x$ 在 0 附近时，可以近似使用这个函数。当接近 $x = 0$‘□□ $: math :$ ‘$y =$ 0□0.5□□□□□□□□ $: math$:sin{x} approx x‘精确到 0.02 以内，但是很快就失去了精度。在下面的图中，我们看到:math:*y =sin{x}*, $y = x$‘□□ $: math :$ ‘x‘□ $: math$:sin{x}‘的近似值和真实值之间的差值。



我们还可以使用非线性 dynamics‘将状态空间系统线性化。为此，我们在状态空间中选择一个点:math:mathbf{x}‘ 并将其用作非线性函数的输入。像上面的示例一样，这对于系统线性化的点附近的状态效果很好，但是可以迅速偏离该状态。

32.8.2 状态空间控制器演练

备注: 建议您在阅读本教程之前阅读:ref:*docs/software/advanced-controls/state-space/state-space-intro:Introduction to state-space control*。

本教程的目的是提供有关为飞轮实现状态空间控制器的“端到端”指令。遵循本教程，读者将学习如何：

1. Create an accurate state-space model of a flywheel using *system identification* or *CAD* software.
2. 实现卡尔曼滤波器以无延迟地过滤编码器速度测量值。
3. 实现一个 LQR <docs/software/advanced-controls/state-space/state-space-intro:The Linear-Quadratic Regulator>‘反馈控制器，该反馈控制器与基于模型的前馈结合时将产生电压:term:‘inputs 1，来驱动飞轮到:term:*reference*。

本教程适用于没有大量编程专业知识的团队。虽然 WPILib 库在实现其状态空间控制功能的方式上提供了显着的灵活性，但是紧跟本教程中概述的实现方式，应该为团队提供可以在各种状态空间系统中重用的基本结构。

The full example is available in the state-space flywheel (Java/C++/Python) and state-space flywheel system identification (Java/C++/Python) example projects.

为什么要使用状态空间控制？

因为状态空间控制关注于创建系统的精确模型，所以我们可以准确地预测:term:*model* 将如何响应控制 inputs 1。这允许我们在不接触物理机器人的情况下模拟我们的机制，以及轻松地选择我们知道将有效的 gains 2。拥有模型还可以使我们创建无滞后的滤波器，例如卡尔曼滤波器，以最佳地过滤传感器读数。

飞轮建模

Recall 连续状态空间系统是使用以下方程组建模的：

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

Where *x-dot* is the rate of change of the *system*’s *state*, **x** is the system’s current state, **u** is the *input* to the system, and **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*.

建立状态空间系统的第一步是选择系统的状态。我们可以选择任何想要的状态——如果需要，我们可以选择完全不相关的状态——但是选择重要的状态是有帮助的。我们可以在我们的状态中包含 hidden states 1 (比如电梯速度，如果我们只能测量它的位置)，并让卡尔曼滤波器估计它们的值。记住，我们选择的状态将被反馈控制器 (通常是:ref:*Linear-Quadratic Regulator <docs/software/advanced-controls/state-space/state-space-intro:The Linear-Quadratic Regulator>*，因为它是最优的) 驱动到它们各自的 references 2。

对于飞轮，我们只关心一种状态：它的速度。尽管我们可以选择对其加速进行建模，但是对于我们的系统而言，不需要包含此状态。

接下来，我们识别系统的 inputs 1。可以将输入视为可以放入系统以更改其状态的事物。对于飞轮（以及 FRC | reg |中的许多其他单联机构)，我们只有一个输入: 施加到电机的电压。通过选择电压作为我们的输入 (而不是像电机占空比这样的东西)，我们可以补偿电池电压下降，电池负载增加。

连续时间状态空间系统将 x-dot‘或:term:‘system’s 状态的瞬时变化率，与:term:*state* 和 inputs 1 成正比。因为我们的状态是角速度，**x** 将是飞轮的角加速度。

接下来，我们将把飞轮建模为一个连续时间状态空间系统。WPILib 的线性系统将在内部将其转换为离散时间。回顾更多关于连续时间和离散时间系统的:ref:*state-space notation <docs/software/advanced-controls/state-space/state-space-intro:What is State-Space Notation?>* 。

使用系统识别建模

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 **a**.

$$V = kV \cdot \mathbf{v} + kA \cdot \mathbf{a}$$
$$\mathbf{a} = \mathbf{v} = \left[\frac{-kV}{kA}\right] v + \left[\frac{1}{kA}\right] V$$

其中:math:*mathbf{v}* 为飞轮速度，**a**‘□ : $math$:mathbf{dot{v}}‘ 为飞轮加速度，V 为电压。用 **x** 的标准约定重写此状态向量，用:math:‘mathbf{u}‘的标准约定重写输入向量，我们发现:

$$\mathbf{x} = \left[\frac{-kV}{kA}\right] \mathbf{x} + \left[\frac{1}{kA}\right] \mathbf{u}$$

状态空间符号的第二部分将系统的当前 state‘和:term:‘inputs 1 与 output‘联系起来。对于飞轮，我们的输出向量:math:mathbf{y}‘(或者传感器可以测量的东西) 是飞轮的速度，它也是 state‘向

量:math:mathbf{x}‘的一个元素，因此，我们的输出矩阵是 $\mathbf{C} = \begin{bmatrix}1\end{bmatrix}$，系统馈通矩阵是:math:*mathbf{D} = begin{bmatrix}0 end{bmatrix}*。用连续时间状态空间符号写出这个结果。

$$\mathbf{x} = \begin{bmatrix}\frac{-kV}{kA}\end{bmatrix}\mathbf{x} + \begin{bmatrix}\frac{1}{kA}\end{bmatrix}\mathbf{u}$$
$$\mathbf{y} = \begin{bmatrix}1\end{bmatrix}\mathbf{x} + \begin{bmatrix}0\end{bmatrix}\mathbf{u}$$

“线性系统” 类包含容易创建使用:term:‘system identification‘的状态空间系统的方法。这个例子展示了一个 kV 为 0.023 kA 为 0.001 的飞轮模型:

Java

```
  // Volts per (radian per second)
  private static final double kFlywheelKv = 0.023;

  // Volts per (radian per second squared)
  private static final double kFlywheelKa = 0.001;

  // The plant holds a state-space model of our flywheel. This system has the␣
↪following properties:
  //
  // States: [velocity], in radians per second.
  // Inputs (what we can "put in"): [voltage], in volts.
  // Outputs (what we can measure): [velocity], in radians per second.
  //
  // The Kv and Ka constants are found using the FRC Characterization toolsuite.
  private final LinearSystem<N1, N1, N1> m_flywheelPlant =
      LinearSystemId.identifyVelocitySystem(kFlywheelKv, kFlywheelKa);
```

C++

```
#include <frc/system/plant/LinearSystemId.h>
```

```
  // Volts per (radian per second)
  static constexpr auto kFlywheelKv = 0.023_V / 1_rad_per_s;

  // Volts per (radian per second squared)
  static constexpr auto kFlywheelKa = 0.001_V / 1_rad_per_s_sq;

  // The plant holds a state-space model of our flywheel. This system has the
  // following properties:
  //
  // States: [velocity], in radians per second.
  // Inputs (what we can "put in"): [voltage], in volts.
  // Outputs (what we can measure): [velocity], in radians per second.
  //
  // The Kv and Ka constants are found using the FRC Characterization toolsuite.
  frc::LinearSystem<1, 1, 1> m_flywheelPlant =
      frc::LinearSystemId::IdentifyVelocitySystem<units::radian>(kFlywheelKv,
                                                                 kFlywheelKa);
```

Python

```
# Volts per (radian per second)
kFlywheelKv = 0.023

# Volts per (radian per second squared)
kFlywheelKa = 0.001
```

```
        # The plant holds a state-space model of our flywheel. This system has the following properties:
        #
        # States: [velocity], in radians per second.
        # Inputs (what we can "put in"): [voltage], in volts.
        # Outputs (what we can measure): [velocity], in radians per second.
        #
        # The Kv and Ka constants are found using the FRC Characterization toolsuite.
        self.flywheelPlant = (
            wpimath.system.plant.LinearSystemId.identifyVelocitySystemRadians(
                kFlywheelKv, kFlywheelKa
            )
        )
```

利用飞轮的惯性矩和齿轮传动进行建模

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).

备注: 对于 WPILib 的状态空间类，齿轮传动被写为输入的输出-也就是说，如果飞轮的旋转速度慢于电动机的旋转速度，则该数字应大于 1。

备注: 线性系统类使用:ref:'the C++ Units Library <docs/software/basic-programming/cpp-units:The C++ Units Library>'来防止单元混淆和断言维数。

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
```

(续下页)

(接上页)

```
↪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

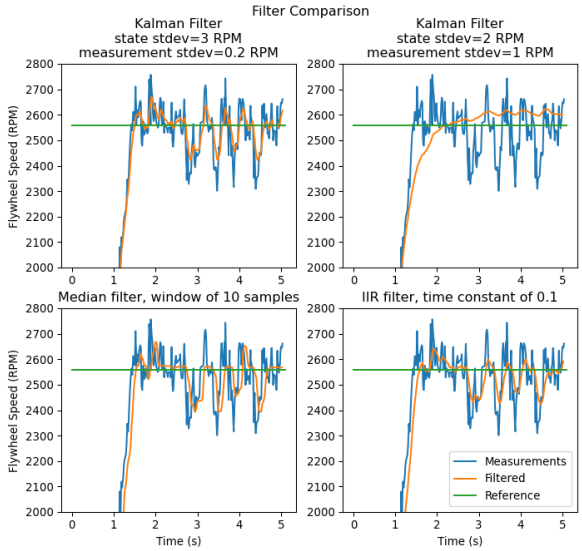
```
kFlywheelMomentOfInertia = 0.00032  # kg/m^2

# 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,
        )
```

卡尔曼滤波器：观察飞轮状态

卡尔曼滤波器用于过滤我们的速度测量使用我们的状态空间模型，以产生一个状态估计:math:*mathbf{hat{x}}*。由于我们的飞轮模型是线性的，我们可以使用卡尔曼滤波估计飞轮的速度。WPILib 的卡尔曼滤波器采用线性系统 (我们在上面发现的)，以及模型和传感器测量值的标准差。我们可以通过调整这些权重来调整我们的状态估计的“平滑度”。较大的状态标准差会导致过滤器“不信任”我们的状态估计值，更倾向于新的测量值，而较大的测量标准差则相反。

对于飞轮，我们的状态标准偏差为 3 rad/s，测量标准偏差为 0.01 rad/s。这些值由用户来选择——这些权重产生了一个过滤器，它可以容忍一些噪音，但是它的状态估计可以快速地对 *a* 飞轮的外部干扰做出反应——应该进行调整，以创建一个适合特定飞轮的过滤器。将状态、测量值、输入、引用和输出随时间变化的图形化是调整卡尔曼滤波器的一种很好的可视化方法。



上图显示了两个不同调谐的卡尔曼滤波器，以及一个:ref:*single-pole IIR filter <docs/software/advanced-controls/filters/linear-filter:Linear Filters>* 和一个:ref:*docs/software/advanced-controls/filters/median-filter:Median Filter*。用射手在约 5 秒钟内收集了此数据，并且有四个球穿过射手（从四个速度下降中可以看出）。尽管没有关于选择良好状态和测量标准偏差的硬性规则，但通常应该对其进行调整以使模型具有足够的信任度，从而在对外部干扰做出快速反应的同时抑制噪声。

由于反馈控制器使用卡尔曼滤波器估计的*x-hat* 来计算误差，因此控制器只会对滤波器的状态估计值变化迅速做出反应。在上图中，橙色曲线（状态标准偏差为 3.0，测量标准偏差为 0.2）产生了一个过滤器，该过滤器对干扰快速反应，同时抑制了噪声，而品红色过滤器几乎不受速度下降的影响。

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
```

(续下页)

(接上页)

```
          // 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,
        )
```

由于卡尔曼滤波器在:ref:‘docs/software/advanced-controls/state-space/state-space-observers:Predict step‘步骤中使用我们的状态空间模型，因此我们的模型尽可能准确是很重要的。验证这一点的一种方法是记录飞轮随时间变化的输入电压和速度，然后通过仅调用 Kalman 滤波器上的预测来重放此数据。然后，可以调节 kV 和 kA 增益（或惯性矩和其他常数），直到模型与记录的数据紧密匹配为止。

线性二次调节器和设备反转前馈

线性二次调节器 找到了一个反馈控制器来将我们的飞轮*system* 驱动到:term:*reference*。因为我们的飞轮只有一个状态，所以我们的 LQR 选择的控制律将采用:math:*mathbf{u = K (r - x)}* 的形式，其中:math:*mathbf{K}‘是一个 1x1 矩阵；换句话说，由 LQR 选择的控制律只是比例控制器，或仅具有 P 增益的 PID 控制器。该增益由我们的 LQR 根据我们通过它的状态偏移和控制努力来选择。更多有关 LQR 控制器调优的信息可在:ref:‘LQR application example <docs/software/advanced-controls/state-space/state-space-intro:LQR: example application>*. 中找到。

在给定 kS, kV 和 kA 常数的情况下, 类似于 SimpleMotorFeedforward 可以用来生成:ref:*feedforward <docs/software/advanced-controls/state-space/state-space-intro:Visualizing Feedforward>* 电压输入, 而在状态空间系统下, Plant Inversion 前馈类可以生成前馈电压输入。由 LinearSystemLoop 类生成的电压命令是前馈和反馈输入的总和。

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.
        )
```

整合在一起：LinearSystemLoop

LinearSystemLoop 结合了我们先前创建的系统，控制器和观察器。所示的构造函数还将实例化 “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
        )
```

一旦我们有了 “LinearSystemLoop’’，剩下的唯一要做的就是实际运行它。为此，我们将使用新的编码器速度测量值定期更新我们的卡尔曼滤波器，并对其应用新的电压命令。为此，我们首先设置*reference*，然

后使用当前飞轮速度进行“校正”，将卡尔曼滤波器“预测”到下一个时间步，然后应用使用“getU”生成的输入‘。

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.
```

(续下页)

(接上页)

```
    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;
```

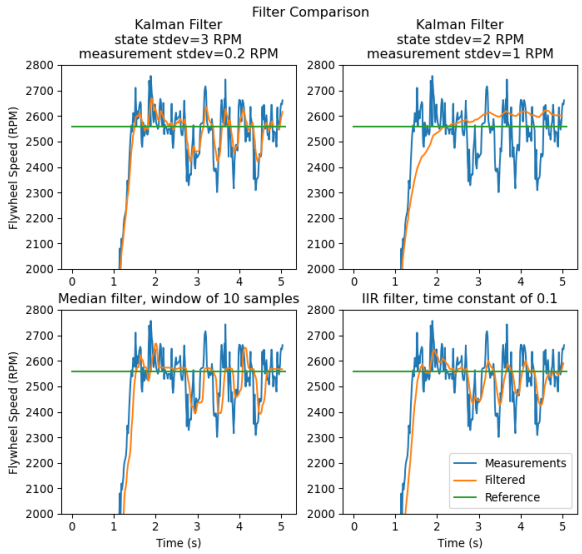
PYTHON

```
error = lqr.R() - x
error[0] = wpimath.angleModulus(error[0])
u = lqr.K() * error
```

32.8.3 状态观察者和卡尔曼滤波器

状态观察者将关于系统行为的信息和外部度量结合起来，以估计系统的真实状态。线性系统常用的观测器是卡尔曼滤波器。卡尔曼滤波器比其他:ref:‘filters <docs/software/advanced-controls/filters/index:Filters>‘更有优势，因为它们将来自一个或多个传感器的测量结果与系统的状态空间模型相融合，以最佳估计系统的状态。

这张图片显示了飞轮速度随时间的测量，通过各种不同的过滤器运行。注意，一个调谐良好的卡尔曼滤波器在飞轮旋转时没有显示测量滞后，同时仍然拒绝噪声数据，并对球通过时的干扰做出快速反应。更多关于过滤器的信息可以在:ref:*filters section <docs/software/advanced-controls/filters/index:Filters>* 找到。



高斯函数

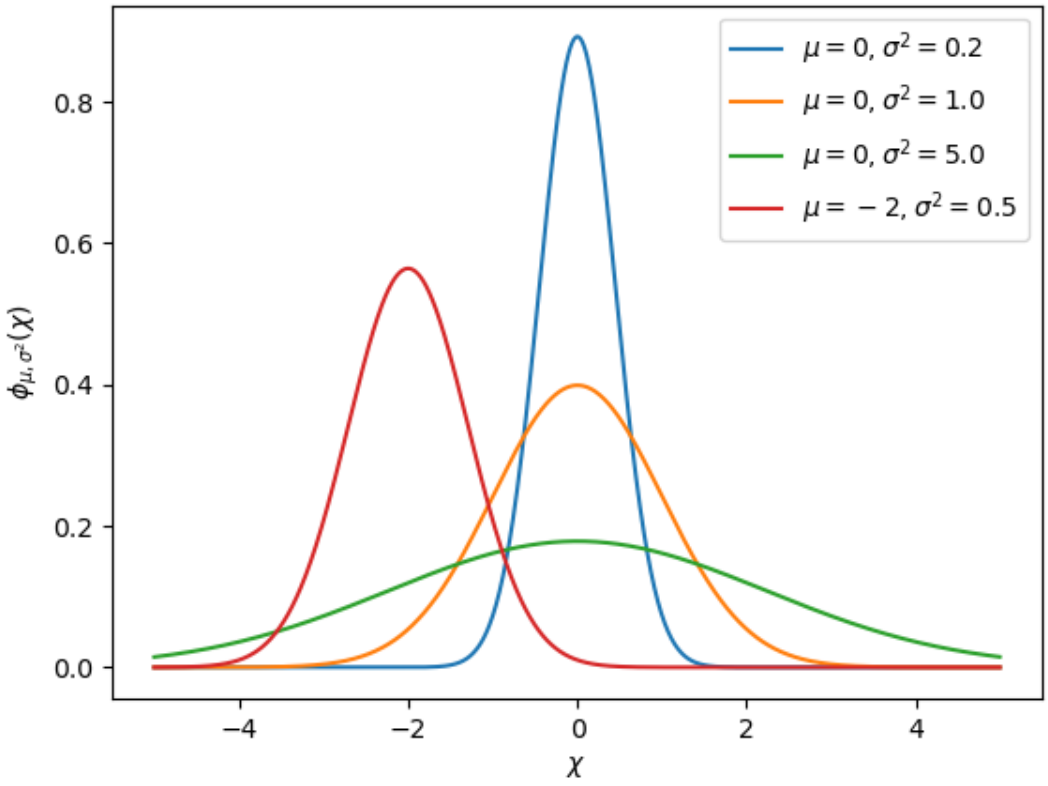
Kalman filters utilize a *Gaussian distribution* to model the noise in a process[1]. In the case of a Kalman filter, the estimated *state* of the system is the mean, while the variance is a measure of how certain (or uncertain) the filter is about the true *state*.

方差和协方差的概念是卡尔曼滤波器功能的核心。协方差是对两个随机变量相关程度的度量。在一个只有一个状态的系统中，协方差矩阵就是简单的:math:*mathbf{text{cov}(x_1, x_1)}*，或者是一个包含状态 x_1‘□□□：*math* :mathbf{text{var}(x_1)}‘ 的矩阵。该方差的大小是描述当前状态估计的高斯函数的标准差的平方。协方差相对较大的值可能表示有噪声的数据，而较小的协方差可能表示过滤器对它的估计更有信心。记住，方差或协方差的“大”值和“小”值与所使用的基本单位有关——例如，如果:math:*mathbf{x_1}‘是以米计量，:math:*mathbf{text{cov}(x_1, x_1)}‘ 将以米的平方计量。

协方差矩阵可写成如下形式:

$$□ = \begin{bmatrix} \operatorname{cov}(x_1, x_1) & \operatorname{cov}(x_1, x_2) & \dots & \operatorname{cov}(x_1, x_n) \\ \operatorname{cov}(x_2, x_1) & \operatorname{cov}(x_2, x_2) & \dots & \operatorname{cov}(x_1, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{cov}(x_n, x_1) & \operatorname{cov}(x_n, x_2) & \dots & \operatorname{cov}(x_n, x_n) \end{bmatrix}$$

[1] In a real robot, noise comes from all sorts of sources. Stray electromagnetic radiation adds extra voltages to sensor readings, vibrations and temperature variations throw off inertial measurement units, gear lash causes encoders to have inaccuracies when directions change…all sorts of things. It’s important to realize that, by themselves, each of these sources of “noise” aren’t guaranteed to follow any pattern. Some of them might be the “white noise” random vibrations you’ve probably heard on the radio. Others might be “pops” or single-loop errors. Others might be nominally zero, but strongly correlated with events on the robot. However, the *Central Limit Theorem* shows mathematically that regardless of how the individual sources of noise are distributed, as we add more and more of them up their combined effect eventually is distributed like a Gaussian. Since we do not know the exact individual sources of noise, the best choice of a model we can make is indeed that Gaussian function.


$\mu = 0, \sigma^2 = 0.2$
$\mu = 0, \sigma^2 = 1.0$
$\mu = 0, \sigma^2 = 5.0$
$\mu = -2, \sigma^2 = 0.5$
0.8
0.6
0.4
0.2
0.0
$\phi_{\mu,\sigma^2}(\chi)$
−4
−2
0
2
4
χ

卡尔曼滤波器

重要: 对卡尔曼滤波器的实际作用有一个直观的认识是很重要的。'Kalman and Bayesian Filters in Python by Roger Labbe <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>'一书提供了贝叶斯滤波器极佳的可视化和互动性介绍。WPILib 中的卡尔曼滤波器使用线性代数来美化数学，但其思想与一维情况相似。我们建议通读第 4 章，直观了解这些过滤器在做什么

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).

下面给出了离散时间卡尔曼滤波器的方程:

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	□	process noise intensity vector
P	error covariance matrix	**Q**	process noise covariance matrix
K	Kalman gain matrix	**R**	measurement noise covariance matrix

状态估计 **x'□ : $math$:mathbf{P}'' 描述高斯函数的均值和协方差，高斯函数描述了我们的滤波器对系统真实状态的估计。

处理和测量噪声协方差矩阵

过程和测量噪声协方差矩阵:math:*mathbf{Q}* 和 **R** 描述了我们的每个状态和测量的方差。记住，对于高斯函数，方差是函数标准差的平方。在 WPILib 中，Q 和 R 是对角矩阵，它们的对角线包含各自的方差。例如，卡尔曼滤波状态 $\begin{bmatrix}\text{position}\\ \text{velocity}\end{bmatrix}$ 和具有状态标准差 $\begin{bmatrix}0.1\\ 1.0\end{bmatrix}$ 和测量标准差 $\begin{bmatrix}0.01\end{bmatrix}$ 的测量 $\begin{bmatrix}\text{position}\end{bmatrix}$ 将具有下面的:math:*mathbf{Q}* 和 **R** 矩阵:

$$Q = \begin{bmatrix}0.01 & 0\\ 0 & 1.0\end{bmatrix}, R = \begin{bmatrix}0.0001\end{bmatrix}$$

误差协方差矩阵

误差协方差矩阵 **P**‘□□□□□□□□□□□□ : *math* :mathbf{hat{x}}‘。非正式地，**P** 描述了我们对估计 state‘的确定性。如果:math:mathbf{P}‘很大，真实状态的不确定性就很大。相反，元素更小的:math:‘mathbf{P}‘意味着我们真实状态的不确定性更小。

当我们向前预测模型时，**P** 会随着我们对系统真实状态的确定性降低而增加。

预测步骤

在预测中，我们的状态估计是根据线性系统动力学:math:*mathbf{dot{x}} = Ax + Bu*‘更新的。此外，我们的误差协方差:math:mathbf{P}‘ 随着过程噪声协方差矩阵:math:*mathbf{Q}*‘的增大而增大，:math:mathbf{Q}‘的值越大，我们的误差协方差 **P** 增长得越快。这个 **P** 用于校正步骤，以权重模型和测量。

校正步骤

在校正步骤中，我们的状态估计值将更新为包含新的测量信息。这个新信息通过卡尔曼增益 **K**‘□□□□□□ : *math* :mathbf{hat{x}}‘进行加权，较大的 **K**. 值对进入测量值的权重更高，较小的 **K** 值对我们的状态预测的权重更高。因为 **K**‘□ : *math* :mathbf{P}‘相关，**P**‘□□□□ : *math* :mathbf{K}‘就越高，重量测量也越重。例如，如果一个过滤器预测了很长一段时间，那么大的:math:‘mathbf{P}‘将会赋予新信息很大的权重。

最后，误差协方差 **P** 减少，以增加我们对状态估计的信心。

调整卡尔曼过滤器

WPILib 的卡尔曼滤波类的构造器采用线性系统、过程噪声标准差和测量噪声标准差向量。通过用每个状态或测量的标准差或方差的平方填充对角线，这些矩阵转换为:math:*mathbf{Q}* 和 **R**‘□□□□□□□□□□□□□□□□□□(□□ : *math* :mathbf{Q}‘中相应的条目)，过滤器将更加不信任传入的度量。类似地，增加一个州的标准偏差将更信任传入的测量结果。对于测量标准偏差也是如此——减少一个条目将使过滤器更加信任输入的测量值对应的状态，而增加它将降低对测量值的信任。

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++

```
#include <numbers>

#include <frc/DriverStation.h>
#include <frc/Encoder.h>
#include <frc/TimedRobot.h>
#include <frc/XboxController.h>
#include <frc/controller/LinearQuadraticRegulator.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/estimator/KalmanFilter.h>
#include <frc/motorcontrol/PWMSparkMax.h>
#include <frc/system/LinearSystemLoop.h>
#include <frc/system/plant/DCMotor.h>
#include <frc/system/plant/LinearSystemId.h>
#include <units/angular_velocity.h>
```

```
  // The observer fuses our encoder data and voltage inputs to reject noise.
  frc::KalmanFilter<1, 1, 1> m_observer{
      m_flywheelPlant,
      {3.0},   // How accurate we think our model is
      {0.01},  // How accurate we think our encoder data is
      20_ms};
```

Python

```
        # The observer fuses our encoder data and voltage inputs to reject noise.
        self.observer = wpimath.estimator.KalmanFilter_1_1_1(
            self.flywheelPlant,
            [3],  # How accurate we think our model is
            [0.01],  # How accurate we think our encoder data is
            0.020,
        )
```

Footnotes

32.8.4 Pose Estimators

WPILib includes pose estimators for differential, swerve and mecanum drivetrains. These estimators are designed to be drop-in replacements for the existing *odometry* classes that also support fusing latency-compensated robot pose estimates with encoder and gyro measurements. These estimators can account for encoder drift and noisy vision data. These estimators can behave identically to their corresponding odometry classes if only `update` is called on these estimators.

Pose estimators estimate robot position using a state-space system with the states $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$, which can represent robot position as a `Pose2d`. WPILib includes `DifferentialDrivePoseEstimator`, `SwerveDrivePoseEstimator` and `MecanumDrivePoseEstimator` to estimate robot position. In these, users call `update` periodically with encoder and gyro measurements (same as the odometry classes) to update the robot's estimated position. When the robot receives measurements of its field-relative

position (encoded as a `Pose2d`) from sensors such as computer vision or V-SLAM, the pose estimator latency-compensates the measurement to accurately estimate robot position.

Here's how to initialize a `DifferentialDrivePoseEstimator`:

JAVA

```
  private final DifferentialDrivePoseEstimator m_poseEstimator =
      new DifferentialDrivePoseEstimator(
          m_kinematics,
          m_gyro.getRotation2d(),
          m_leftEncoder.getDistance(),
          m_rightEncoder.getDistance(),
          new Pose2d(),
          VecBuilder.fill(0.05, 0.05, Units.degreesToRadians(5)),
          VecBuilder.fill(0.5, 0.5, Units.degreesToRadians(30)));
```

C++

```
  frc::DifferentialDrivePoseEstimator m_poseEstimator{
      m_kinematics,
      m_gyro.GetRotation2d(),
      units::meter_t{m_leftEncoder.GetDistance()},
      units::meter_t{m_rightEncoder.GetDistance()},
      frc::Pose2d{},
      {0.01, 0.01, 0.01},
      {0.1, 0.1, 0.1}};
```

Add odometry measurements every loop by calling `Update()`.

JAVA

```
    m_poseEstimator.update(
        m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
↪getDistance());
```

C++

```
  m_poseEstimator.Update(m_gyro.GetRotation2d(),
                         units::meter_t{m_leftEncoder.GetDistance()},
                         units::meter_t{m_rightEncoder.GetDistance()});
```

Add vision pose measurements occasionally by calling `AddVisionMeasurement()`.

JAVA

```
    // Compute the robot's field-relative position exclusively from vision measurements.
    Pose3d visionMeasurement3d =
        objectToRobotPose(m_objectInField, m_robotToCamera, m_cameraToObjectEntry);

    // Convert robot pose from Pose3d to Pose2d needed to apply vision measurements.
    Pose2d visionMeasurement2d = visionMeasurement3d.toPose2d();
```

C++

```
  // Compute the robot's field-relative position exclusively from vision
  // measurements.
  frc::Pose3d visionMeasurement3d = ObjectToRobotPose(
      m_objectInField, m_robotToCamera, m_cameraToObjectEntryRef);

  // Convert robot's pose from Pose3d to Pose2d needed to apply vision
  // measurements.
  frc::Pose2d visionMeasurement2d = visionMeasurement3d.ToPose2d();

  // Apply vision measurements. For simulation purposes only, we don't input a
  // latency delay -- on a real robot, this must be calculated based either on
  // known latency or timestamps.
  m_poseEstimator.AddVisionMeasurement(visionMeasurement2d,
                                       frc::Timer::GetFPGATimestamp());
```

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 调试状态空间模型和控制器

检查标志

状态空间控制器错误的最常见原因之一是信号被翻转。例如，WPILib 中包含的模型期望正电压会导致正加速度，反之亦然。如果施加正电压不能使该机构向前加速，或者如果“向前”移动会使编码器（或其他传感器读数）减小，则应将其反转，以使正电压输入导致编码器读数为正。例如，如果我对差速器传动系统应用：math：*[12，12] ^ T‘*（*左右电动机的全速前进*）的：*term*：*‘input*，则我的车轮应将机器人“向前”推进（沿 + X 轴局部），并使我的编码器读取正速度。

重要： 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.

图的重要性

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.

备注： 默认情况下，NetworkTables 的更新速率限制为 10hz。为了进行测试，可以使用以下代码段绕过它，以最高 100hz 的速度提交数据。该代码应定期运行以强制发布新数据。

危险： 这将通过 NetworkTables 发送额外的数据（最高 100hz），这可能导致用户代码和机械手仪表板的滞后。这也将提高网络利用率。在比赛中禁用此功能通常是个好主意。

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()
```

补偿输入滞后

通常，某些传感器输入数据（即速度读数）可能会由于智能电机控制器倾向于执行的车载过滤而被延迟。默认情况下，LQR 的 K 增益不假定输入延迟，因此引入数十毫秒量级的显着延迟可能会导致不稳定。为了解决这个问题，可以降低 LQR 的 K 增益，以牺牲性能为代价。有关如何以数学上严格的方式补偿此延迟的代码示例，请参见*here*。

32.9 控制术语表

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.

控制工作
: *Control signal*

控制律
: 一种数学公式，在给定当前状态的情况下，产生 inputs 1 来驱动 system'达到期望的:term:'state。一个常见的例子是控制律:math:*mathbf{u} = mathbf{K(r - x)}*

controller
: 与: 术语: "装置" 一起用于位置或负反馈，通过驱动: 术语: "参考" 信号和: 术语: "输出" 之间的差值到零，从而达到所期望的: 术语: "系统状态"。

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.

动力学
物理学的一个分支，涉及在力作用下物体的运动。在现代控制中，系统根据其动态发展。

derivative
A mathematical operation which evaluates the "rate-of-change" of a function at a given point. See derivative on Wikipedia for more info.

错误
Reference 1 减去: term:*output*'或*:term:*'*state*.

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.

增益
A scalar value that relates the magnitude of an input signal to the magnitude of an output signal. For example, `gain` in `output = gain * input`. A gain greater than one would amplify an input signal, while a gain less than one would dampen an input signal. A negative gain would negate the input signal.

Gaussian distribution
A special mathematical function that describes distributions of averages. The graph of a Gaussian function is a "bell curve" shape. This function is described by its mean (the location of the "peak" of the bell curve) and variance (a measure of how "spread out" the bell curve is). See Gaussian distribution on Wikipedia for more info.

gradient
The *derivative*, but applied to a function with multiple inputs. As a result, the output is both the magnitude of the rate of change, and the vector direction along which it occurs.

隐藏状态
一种不能直接测量的*state*，但其*dynamics* 可以与其他状态相关。

输入
An input to the *plant* (hence the name) that can be used to change the *plant's state*.

- 例如：飞轮有 1 个输入：驱动它的电动机电压。
- 例如：动力传动系统可能有 2 个输入：左右电动机的电压。

输入通常用变量 **u**, 表示，这是一个列向量，每个输入对 system 都具有一个输入。

least-squares regression
A curve-fitting technique which picks a curve to minimizes the *square* of the error be-

tween the fitted curve, and the actual measured data. See ordinary least-squares regression on Wikipedia for more info.

LQR
Linear-Quadratic Regulator - A feedback control scheme which seeks to operate a system in a "most optimal"or "lowest cost"manner, in the sense of minimizing the square of some "cost function" that represents a combination of system error and control effort. This requires an accurate mathematical model of the system being controlled, and function describing the "cost" of any given system state. See LQR on Wikipedia for more info.

测量
测量是用传感器从 plant'或物理系统测量的:term:'outputs 1 。

模型
反映物理某些方面的一组数学方程::term:'system' s 行为。

观察者
在控制理论中，一种系统，通过测量一个给定实体的 "输入" 和 "输出" 来估计其内部的状态。WPILib 包括一个用于观察线性系统的卡尔曼滤波器类，以及用于非线性系统的扩展卡尔曼滤波器类和非 scentedkalmanfilter 类。

orthogonal
Having the property of being independent, or lacking mutual influence. For example, two lines are orthogonal if moving any number of units along one line causes zero displacement along the other line. In a *cartesian coordinate system*, orthogonal lines are often said to have 90-degree angles between each other.

output
测量传感器。可以有更多的测量，而不是状态。这些输出用于卡尔曼滤波器的 "正确" 步骤。

- 例如：飞轮可能用:term:'output'输出 1 来表示其速度。
- 例如：动力传动系统可能使用 solvePNP 和 V-SLAM 在野外找到其 x / y /航向位置。可以进行 6 种测量（solvePNP x / y / heading 和 V-SLAM x / y / heading）和 3 种状态（robot x / y / heading）。

系统的输出通常使用变量：mathbf {y}'表示，该列向量每个/ term 输出有一个条目（或我们可以测量的东西）。例如，如果我们的：term：*system'具有速度和加速度的状态，但是我们的传感器只能测量速度，那么我们的：term：'output'向量将仅包括：term：'system* 的速度。

phase portrait
A graph of a function' s value and its *derivative* as they change in time, given some initial starting conditions. They are useful for analyzing system behavior (stable/unstable operating points, limit cycles, etc.) given a certain set of parameters or starting conditions. See phase portrait on Wikipedia for more info.

PID
Proportional-Integral-Derivative - A feedback controller which calculates a *control signal* from a weighted sum of the *error*, the rate of change of the error, and an accumulated sum of previous errors. See PID controller. on Wikipedia for more info.

设备
术语: 被控制的执行机构的 "系统" 或集合。

过程变量
该术语用于描述 PID 控制环境下的*plant* 的输出。

r-squared
A statistical measurement of how well a model predicts a set of data, representing the fraction of the observed variation in the independent variable that is accurately predicted by the model. The value typically runs from 0.0 (a terrible fit, equivalent to just

guessing the average value of your independent variable) to 1.0 (a perfect fit). See Coefficient_of_determination on Wikipedia for more info.

参考
期望的状态。这个值用作控制器误差计算的参考点。

上升时间
在应用:term:*step input*‘后，*:term:‘system* 最初到达*reference* 所需的时间。

RMSE
Root Mean Squared Error - Statistical measurement of how well a curve is fit to a set of data. It is calculated as the square root of the average (mean) of the squares of all the errors between the actual sample and the curve fit. It has units of the original input data. See Root Mean Squared Error on Wikipedia for more info.

设定点
该术语用于描述 PID 控制器的*reference* 。

沉淀时间
:term:‘system‘在:term:‘step input‘后的:term:‘reference‘处设置所需的时间。

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.

状态
system 的特征（例如速度），可用于确定:term:‘system’ s 的未来行为。在状态空间表示法中，系统的状态被写为描述其在状态空间中位置的列向量。

- 例如：动力传动系统可能具有状态 $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$ 来描述其在现场的位置。
- 例如：抬升系统可能具有以下状态：$\begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix}$ 来描述其当前的高度和速度。

A system’ s 1 的状态通常由变量 **x**‘□□□□□□□□□□□□□□ : *term* : ‘*state*. 有一个条目。

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.

稳态误差
Error after *system* reaches equilibrium.

阶跃输入
A:term: ‘system ‘:term: ‘input ‘即:math: ‘0 ‘for:math: ‘t < 和一个大于:math: ‘0 ‘的常数:math: ‘t geq 0 ‘。对于:math: ‘t geq 0 ‘，step input 为:math: ‘1 ‘，称为单位阶跃输入。

阶跃响应
The response of a *system* to a *step input*.

系统
一个包含:term: ‘plant ‘的术语，它与:term: ‘controller ‘和:term: ‘observer ‘交互，被视为一个单独的实体。从数学上讲，a:term: ‘system ‘通过:term: ‘state ‘的线性组合将:term: ‘input ‘映射到:term: ‘output ‘。

系统识别
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.

系统反应
对于给定的：term：*input*，：term：‘system‘随时间的行为。

voltage
The measurement of how much an electric field is "pushing" electrons through a circuit. It is sometimes called "Electromotive Force", or "EMF". It is measured in units of "Volts". It always is defined between *two* points in a circuit. If one electron travels between two points that have one volt of EMF between them, it will have been accelerated to the point of having $\frac{1}{6241509074000000000}$ joules of energy.

viscous drag
The force generated from an object moving *relatively* slowly through non-turbulent fluid. In this region, the force is roughly proportional to the *velocity* of the object. It describes the most common type of "air resistance" an FRC robot would encounter, as well as losses in a gearbox from displacing grease. See Drag (physics) on Wikipedia for more info.

x-dot
$\dot{\mathbf{x}}$, 或者 x-dot: state‘向量:math:mathbf{x}‘的导数，如果 term:*system* 只有一个速度:term:*state*，那么:math:‘dot{mathbf{x}}‘表示系统的加速度。

x-hat
$\hat{\mathbf{x}}$, 或者 x-hat: 由:term:*observer‘估计的系统估计:term:‘state* 。

33 便捷功能

这个部分包括了一些可以与其它高级编程特征通用的便捷功能

33.1 自定义频率的调度功能

TimedRobot 的 addPeriodic() 方法允许用户以比默认 TimedRobot 定期更新速率（20 毫秒）更快的速度运行自定义方法。以前，团队必须制作一个通告程序，以比 20 毫秒的 TimedRobot 循环周期（不建议这样做更频繁地运行 TimedRobot）更频繁地运行反馈控制器。现在，用户可以比主机器人循环更频繁地运行反馈控制器，但可以与 TimedRobot 周期性功能同步运行，从而消除了潜在的线程安全问题。

addPeriodic() (Java）/ AddPeriodic()（C ++）方法采用 lambda（运行功能），要求时间段以及对一个一般开始时间的可选偏移量。相对于其他 TimedRobot 周期性方法，可选的第三个参数对于在不同的时隙中调度函数很有用。

备注： 在 Java 中，时间段和偏移量的单位是秒。在 C ++ 中，单位库可用于指定任何时间单位的时间段和偏移量。

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() {
```

(续下页)

(接上页)

```
        if (m_joystick.getRawButtonPressed(1)) {
            if (m_controller.getSetpoint() == 0.0) {
                m_controller.setSetpoint(30.0);
            } else {
                m_controller.setSetpoint(0.0);
            }
        }
    }
  }
```

C++ (Header)

```
class Robot : public frc::TimedRobot {
 private:
  frc::Joystick m_joystick{0};
  frc::Encoder m_encoder{1, 2};
  frc::Spark m_motor{1};
  frc::PIDController m_controller{1.0, 0.0, 0.5, 10_ms};

  Robot();

  void TeleopPeriodic() override;
};
```

C++ (Source)

```
void Robot::Robot() {
  AddPeriodic([&] {
    m_motor.Set(m_controller.Calculate(m_encoder.GetRate()));
  }, 10_ms, 5_ms);
}

void Robot::TeleopPeriodic() {
  if (m_joystick.GetRawButtonPressed(1)) {
    if (m_controller.GetSetpoint() == 0.0) {
      m_controller.SetSetpoint(30.0);
    } else {
      m_controller.SetSetpoint(0.0);
    }
  }
}
```

在这个例子中，teleopPeriodic() 方法每 20ms 运行一次，控制器每 10ms 运行一次。当 teleopPeriodic() 运行时有 5ms 的偏移量以避免它们的时隙冲突。（比如 teleopPeriodic() 每 0 ms，20 ms，40 ms 运行一次；控制器每 5ms，15ms，25ms)

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.

备注: 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(); }
```

警告: 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`.

> **警告：** 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>();
```

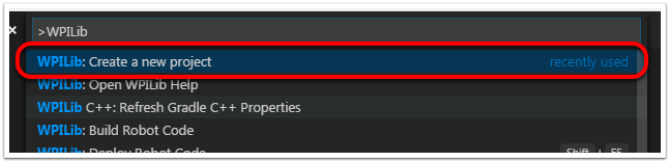
备注: In Java, the parameter expects a method reference to a constructor accepting an `EventLoop` instance and a `BooleanSupplier`. Due to the lack of method references, this parameter is defaulted in C++ as long as a constructor of the form `Type(frc::EventLoop*, std::function<bool()>)` exists.

---

34 WPILib 示例项目

警告： 虽然我们要尽力保持 WPILib 示例的功能，但并不意味着可以直接使用示例代码。如果要使代码在用户的机器人上工作，至少要更改特定于机器人的常量。为了演示的目的，许多需要实际测试得出的常数的值都被“伪造”了。我们强烈鼓励用户编写自己的代码（从零开始或从现有模板开始），而不是复制示例代码。

WPILib 示例项目演示了许多库功能和使用模式。项目范围从单一功能的简单演示到完整的具有竞争能力的机器人程序。输入 Ctrl + Shift + P 键，然后选择 'WPILib: Create a new project' 并选择示例，即可在 VS Code 中使用所有这些示例。



34.1 基本范例

这些示例演示了基本的机器人功能。对于刚开始熟悉机器人编程但机器人功能受到严重限制的入门团队，它们很有用。

- **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 控制示例

这些示例演示了通用机器人控件的 WPILib 实现。可能存在传感器，但不是这些示例的重点概念。

- **DifferentialDriveBot** (Java, C++, Python): Demonstrates an advanced differential drive implementation, including encoder-and-gyro odometry through the `DifferentialDriveOdometry` class, and composition with PID velocity control through the `DifferentialDriveKinematics` and `PIDController` classes.
- **Elevator with profiled PID controller** (Java, C++, Python): Demonstrates the use of the `ProfiledPIDController` class to control the position of an elevator mechanism.
- **Elevator with trapezoid profiled PID** (Java, C++, Python): Demonstrates the use of the `TrapezoidProfile` class in conjunction with a "smart motor controller" to control the position of an elevator mechanism.
- **Gyro Mecanum** (Java, C++, Python): Demonstrates field-oriented control of a mecanum robot through the `MecanumDrive` class in conjunction with a gyro.
- **MecanumBot** (Java, C++, Python): Demonstrates an advanced mecanum drive implementation, including encoder-and-gyro odometry through the `MecanumDriveOdometry` class, and composition with PID velocity control through the `MecanumDriveKinematics` and `PIDController` classes.
- **PotentiometerPID** (Java, C++, Python): Demonstrates the use of the `PIDController` class and a potentiometer to control the position of an elevator mechanism.
- **RamseteController** (Java, C++, Python): Demonstrates the use of the `RamseteController` class to follow a trajectory during the autonomous period.
- **SwerveBot** (Java, C++, Python): Demonstrates an advanced swerve drive implementation, including encoder-and-gyro odometry through the `SwerveDriveOdometry` class, and composition with PID position and velocity control through the `SwerveDriveKinematics` and `PIDController` classes.
- **UltrasonicPID** (Java, C++, Python): Demonstrates the use of the `PIDController` class in conjunction with an ultrasonic sensor to drive to a set distance from an object.

34.3 传感器实例

这些示例演示了使用 WPILib 进行传感器读取和数据处理。可能存在机制控制，但不是这些示例强调的概念。

- •Axis Camera Sample (Java, C++): 演示了 OpenCV 和 Axis 网络摄像机的使用，在捕获的视频上覆盖一个矩形，并将其输出到仪表盘
- **Power Distribution CAN Monitoring** (Java, C++, Python): Demonstrates obtaining sensor information from a Power Distribution module over CAN using the `PowerDistribution` class.
- **Duty Cycle Encoder** (Java, C++, Python): Demonstrates the use of the `DutyCycleEncoder` class to read values from a PWM-type absolute encoder.
- **DutyCycleInput** (Java, C++, Python): Demonstrates the use of the `DutyCycleInput` class to read the frequency and fractional duty cycle of a *PWM* input.
- **Encoder** (Java, C++, Python): Demonstrates the use of the `Encoder` class to read values from a quadrature encoder.
- **Gyro** (Java, C++, Python): Demonstrates the use of the `AnalogGyro` class to measure robot heading and stabilize driving.
- **Intermediate Vision** (Java, C++, Python): Demonstrates the use of OpenCV and a USB camera to overlay a rectangle on a captured video feed and stream it to the dashboard.
- **AprilTagsVision** (Java, C++): Demonstrates on-roboRIO detection of AprilTags using an attached USB camera.
- **Ultrasonic** (Java, C++, Python): Demonstrates the use of the `Ultrasonic` class to read data from an ultrasonic sensor in conjunction with the `MedianFilter` class to reduce signal noise.
- **SysIdRoutine** (Java, C++): Demonstrates the use of the SysIdRoutine API to gather characterization data for a differential drivetrain.

34.4 基于指令的示例

这些示例演示了:ref:‘Command-Based framework <docs/software/commandbased/what-is-command-based:What Is “Command-Based” Programming?>‘的使用。

- **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++): 一套完整的 WPI 演示机器人代码：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++): 演示使用 TrajectoryGenerator 和 MecanumControllerCommand 类用麦克纳姆轮实现轨迹的生成与追踪
- **RamseteCommand** (Java, C++, Python): Demonstrates trajectory generation and following with a differential drive using the `TrajectoryGenerator` and `RamseteCommand` classes. A matching step-by-step tutorial can be found *here*.
- **Select Command Example** (Java, C++, Python): Demonstrates the use of the `SelectCommand` class to run one of a selection of commands depending on a runtime-evaluated condition.
- •SwerveControllerCommand(Java, C++): 使用 TrajectoryGenerator 和 SwerveControllerCommand 类演示轨迹生成和转向驱动。

34.5 状态空间示例

这些示例演示了:ref:'State-Space Control <docs/software/advanced-controls/state-space/state-space-intro:Introduction to State-Space Control>'的使用。

- **StateSpaceFlywheel** (Java, C++, Python): Demonstrates state-space control of a flywheel.
- **StateSpaceFlywheelSysId** (Java, C++, Python): Demonstrates state-space control using SysId's System Identification for controlling a flywheel.
- •StateSpaceElevator (Java, C++): 演示电梯的状态空间控制
- •Statespacarm (Java, C++): 演示机械臂的状态空间控制
- •Statespacedrivessimulation(Java, C++): 结合 RAMSETE 路径跟踪控制器和 Field2d 类演示差速传动系统的状态空间控制

34.6 模拟物理实例

这些示例说明了物理模拟的用法。

- **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.
- •Statespacedrivessimulation(Java, C++): 结合 RAMSETE 路径跟踪控制器和 Field2d 类演示差速传动系统的状态空间控制

- •StateSpaceDrivesSimulation (Java, C++): 结合 RAMSETE 路径跟踪控制器和 Field2d 类演示了差速传动系统的状态空间控制

34.7 其他例子

这些示例演示了不属于上述任何类别的其他 WPILib 功能。

- **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++): 演示了 HAL（硬件抽象层）的使用（不使用 WPILib 的其余部分）这个例子是针对进阶用户的（仅限 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 支持资源 page.

- Cross The Road Electronics (CTRE)
- Kauai Labs (navX)
- Limelight (additional examples, called tutorials, can be found on the left)
- PhotonVision
- REV Robotics

36

硬件-基础

36.1 连接线路的最佳实践

小技巧: 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 振动/冲击

FRC® 机器人在振动和冲击负荷方面的环境异常恶劣。尽管许多 FRC 专用电子设备已在这些条件下进行了机械坚固性的广泛测试，但某些组件如无线电设备，并不是专门为在移动平台上使用而设计的。采取措施减少这些组件受到的冲击和振动，可能有助于减少故障。一些可以减少机械故障的建议：

- 隔振 - 确保使用隔振器来隔离任何产生过度振动的组件，比如压缩机。这会有助于减少机器整体上的振动，否则，振动会使紧固件松动，并导致某些电子组件过早失效。
- 保险杠 - 使用保险杠来尽量多的覆盖您所设计的机器人。虽然规则上要求在机器人的各个边角处覆盖特定的保险杠，但是最大程度的使用保险杠会大大提高所有碰撞被保险杠缓冲的概率。与直接撞击机器人坚硬的表面相比，保险杠大大的降低了在冲撞时所承受的力量，进而减少了电子设备所受到的冲击，进而降低了相关故障发生的可能性。
- 避震安装 - 您可以选择对某些或全部电子组件进行避震安装，以进一步减小他们在机器人冲撞过程中受到的力量。这对于机器人的无线电以及其他电子组件，例如协处理器尤其有用，因为它们也许并不是专门为了适用于移动的平台而设计的，将这些组件安装到隔振器，弹簧，泡沫或其他柔性材料上都可以减少这些组件所受到的冲击力。

36.1.2 冗余

不幸的是，在 FRC 控制系统中，很少有冗余是有用的。有时候，利用冗余可能提高可靠性。一个主要示例是将桶型连接器连接到无线电，还有提供 PoE 连接。这样就可以确保如果其中有一根电缆损坏或者脱落，另一根电缆将保持无线电的供电。留意其他有机会使用冗余的区域，以确保在您连接和编写机器人代码的时候使用它。

36.1.3 端口保护

使用“端口保护程序”或“尾纤”可以大大减少机器人或 Driver station 上经常插拔的任何连接（例如 DS 操作杆，DS 以太网，roboRIO USB 线和以太网线）端口损坏的可能性。这种类型的设备可以起到双重作用，既可以减少电子设备上看到的端口周期数，又可以将连接重新定位到更方便的位置。确保安装好端口保护程序（请参阅下一项）以避免端口损坏。

36.1.4 电线管理和应力消除

机器人的可靠性和维护性最关键的要素之一是良好的导线管理和应力消除。好的电线管理包括以下几个部分：

- 确保电缆长度正确。任何多余的导线长度都需要多加思考。如果由于 COTS 电缆上的额外长度而必须有多余的电线，请在固定其余电线之前，使用单独的电缆扎带将多余的电线固定在小捆中。
- 确保将电缆固定在连接点附近，并留有足够的松弛空间，以免对连接器造成压力。这称为应力消除，可以最大程度地减小电缆在连接点（这是常见的应力集中地）拔出或导线断开的可能性。
- 在任何可动部件附近固定电缆。确保所有的电线可以稳妥运作且不受移动组件的影响，即使移动组件弯曲或出轨。
- 如有必要，可以额外找几个点固定电缆以保持布线整洁。注意不要固定点太多；如果电线固定的位置太多，可能会使故障排除和维护更加困难。

36.1.5 文档

创建描述机器人连接处的文档是简化维护的一种好方法。创建此类文档的方法有很多种，从完整的接线图到 excel 图表，再到将哪些功能附加到哪些通道的快捷清单，不一而足。许多团队还将这些列表与标签集成在一起（请参阅下一项）。

如果不小心电线意外切断了，或者机械出现了故障，或者组件烧坏了，如果您有一些文档告诉您所连接的地方，维修起来会容易得多，而不必从头梳理电线（即使您的接线整齐！）。

36.1.6 标签

标签是补充上述接线文档的好方法。标记电线和电子产品有许多不同的策略，各有其优缺点。电子产品的标签和电线的标志可以手工制作，也可以使用标签制造商（有些也可以制作热缩标签）制成，或者可以使用不同颜色的电工胶带或标记来表示不同的东西。无论选择哪种方法，请确保您了解如何使用它帮助完善您的文档，并确保团队中的每个人都熟悉它。

36.1.7 检查所有接线和连接

机器人上的所有接线完成后，请仔细检查每个连接并拉动以确保一切牢固。此外，请确保没有散线“须”从任何连接点伸出或未绝缘的连接裸露在外。如果在测试过程中任何连接松动，或发现任何“须”，请重新接线并确保完成后请第二个人检查。

不良连接多数来自螺钉或螺母螺栓一类。对于机器人上的任何此类连接（例如电池连接，主断路器，PDP，roboRIO），请确保紧固件已拧紧。对于螺母型连接，请确保电线/端子用手转不动；如果您可以通过抓住端子并扭绞来旋转电池导线或主断路器连接，则连接不够牢固。

另一个常见的故障源是 PDP 末端的保险丝。确保这些保险丝完全固定好；您需要施加的力量比您认为足够完全入座要更多。如果保险丝正确安装，则可能很难或无法用手将其卸下。

诸如 SB-50 连接器之类的卡扣式连接应使用线夹或电缆扎带固定，以确保它们在撞击过程中不会松动。

36.1.8 尽早并经常地重新检查

在打完一两场比赛（或进行非常激烈的测试）之后，请尽可能彻底地重新检查整个电气系统。机器人遇到的前几个冲击可能会松开紧固件或暴露出问题所在。

创建一个检查表以定期重新检查电气连接。作为一个非常初步地开始，应每 1-3 场比赛检查一次旋转紧固件，例如电池和 PDP 连接。例如 WAGO 和 Weidmuller 连接器，弹簧型连接每个赛事可能只需要检查一次。确保团队知道谁负责填写清单，以及他们将如何记录该清单。

36.1.9 电池维护

小心保管好电池！电池电量不足会很容易导致机器人在比赛中无法正常工作或根本无法工作。给所有电池贴上标签，以帮助跟踪活动期间的使用情况。许多团队还在此标签上包含诸如电池寿命之类的信息。

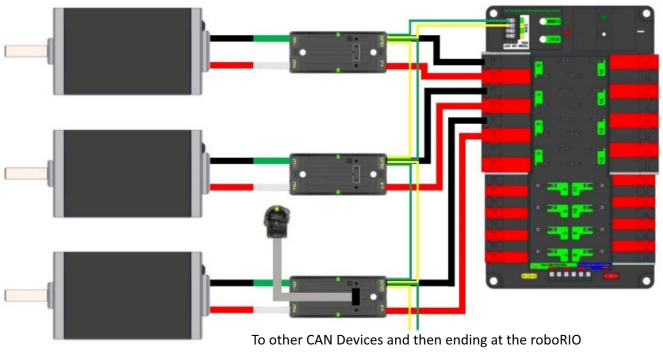
- 切勿用电线提起或搬运电池！电线扯着电池有可能损坏端子和极板之间的内部连接，从而大大增加内部电阻并降低性能。
- 把所有摔过的电池标记成损坏，直到进行完整的测试为止。除上述地端子连接外，掉落的电池可能损坏单个蓄电池单元。这种损坏可能无法通过简单的电压测试记录下来，而是隐藏起来直到将电池置于负载下为止。
- 均匀循环电池。这有助于确保电池有最大的充电和休息时间，并确保它们被均匀使用（相等的充电/放电循环次数）
- 尽可能装入测试电池来监控健康状况。有许多商业产品可供队伍用作测试电池，包括至少一种专门为 FRC 设计的产品。负载测试可以通过测量内部电阻来指示电池的运行状况。这种测试方法远比万用表提供的简单空载电压值相比更接近实际表现、更有意义。

36.1.10 检查 DS 日志

每次比赛后，查看 DS 日志以查看电池电压和电流使用情况。透过建立机器人在这些项目上的正常范围，您就可以在潜在问题（严重的电池，电动机故障，机械冲突）变为严重故障之前发现它们。

36.2 CAN 接线基础

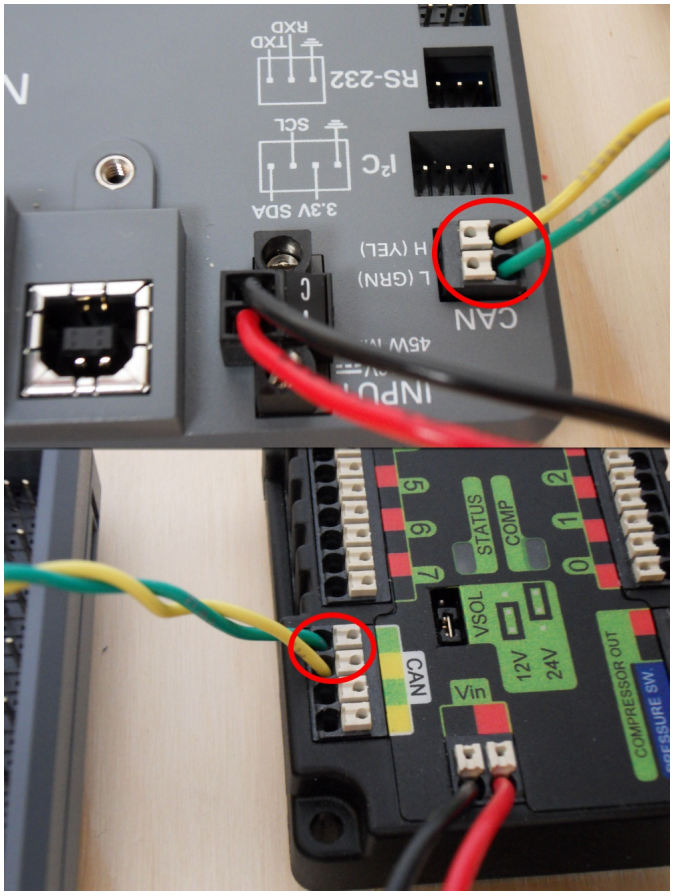
CAN is a two wire network that is designed to facilitate communication between multiple devices on your robot. It is recommended that CAN on your robot follow a "daisy-chain" topology. This means that the CAN wiring should usually start at your roboRIO and go into and out of each device successively until finally ending at the *PDP*.



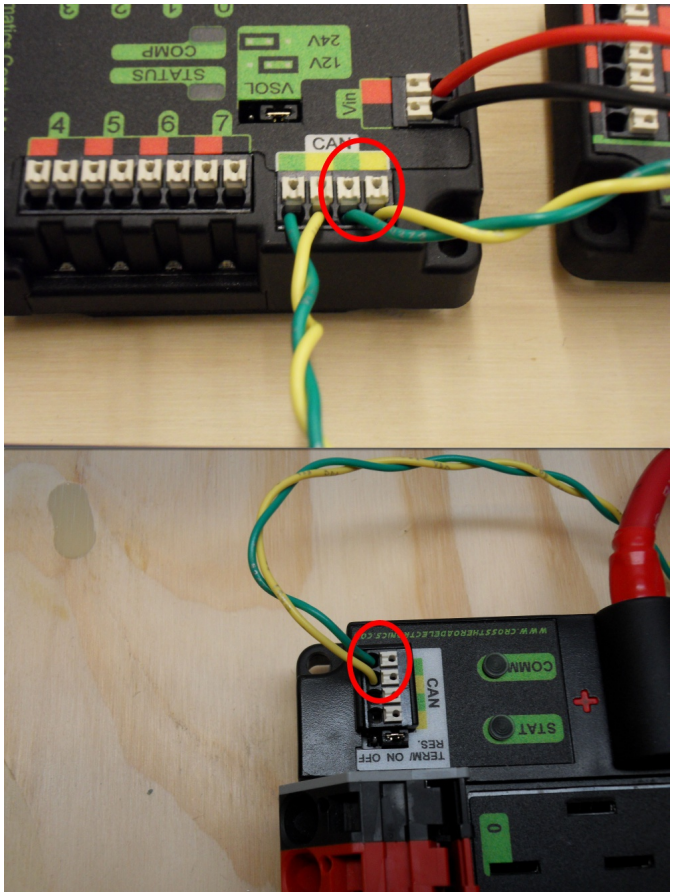
36.2.1 标准接线

CAN 通常使用黄色和绿色的导线布线，黄色表示 CAN-High 信号，绿色则表示 CAN-Low 信号。大多设备都使用黄色和绿色的配色方案用来指导应如何插入导线。

从 roboRIO 到 PCM 的 CAN 接线。



从 PCM 到 PDP 的 CAN 接线。



36.2.2 终端

布线从 roboRIO 处开始，在 PDP 处结束是被建议的，因为 CAN 网络需要用 120:math:*Omega*‘电阻终止，而这两个器件都内置了这些电阻。*PDP* 随附的 *CAN* 总线终端电阻的跳线处于 “*ON*” 位置。建议将跳线保持在此位置，并将任何额外的 *CAN* 节点放置在 *roboRIO* 和 *PDP* 之间（保证 *PDP* 位于总线末端）。如果您希望将 *PDP* 放置在总线中间（使用两对 *PDP CAN* 端子），将跳线移至 “*OFF*” 位置，并将自己的 *120 :math:*Omega‘ 终端电阻放置在 CAN 总线链的末端。

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*.

提示： For pneumatics safety & mechanical requirements, consult this year’ s Robot Construction rules. For mechanical design guidelines, the FIRST Pneumatics Manual is located here

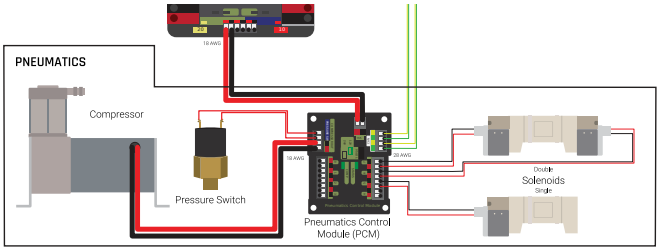
---

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.

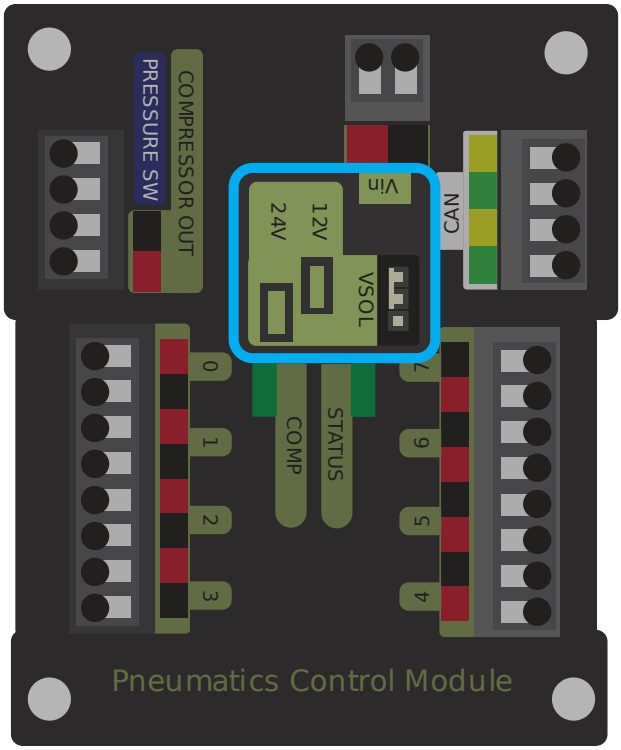
36.3.4 The Pressure Switch

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*.

提示: For pneumatics safety & mechanical requirements, consult this year's Robot Construction rules. For mechanical design guidelines, the FIRST Pneumatics Manual is located here

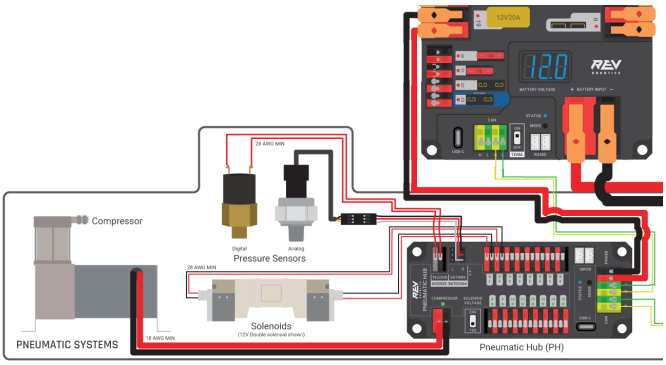
---

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.

> **警告:** 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.

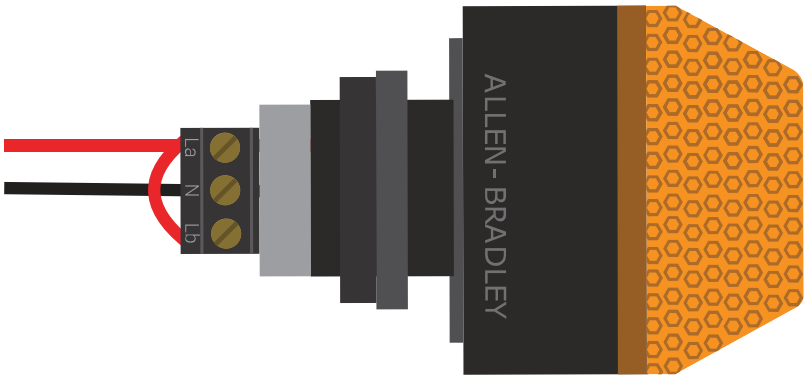
36.4.6 Solenoid Voltage Switch

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 状态指示灯快速参考

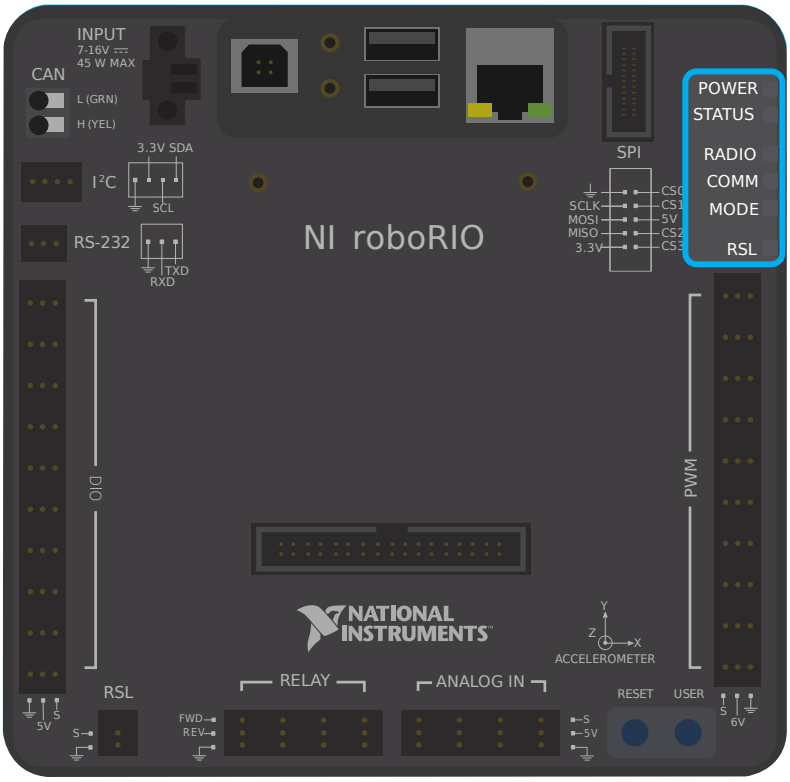
许多 FRC 控制系统的元件有状态指示灯，可被用于快速诊断机器人出现的问题。这个指南包含每一个元件的指示灯和它所描述的状态。图片和文字来自于 Innovation FIRST 和 Cross the Road Electronics。

36.5.1 机器人状态信号灯 (RSL)



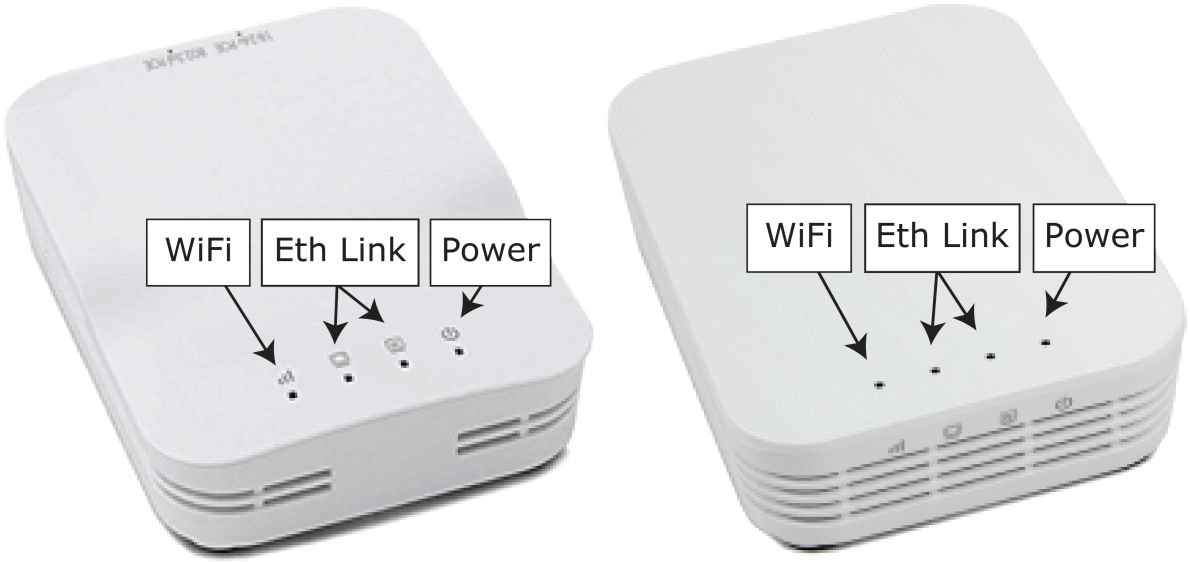
常亮	机器人通电并处于 Disabled 状态
闪烁	机器人通电并处于 Enabled 状态
不亮	机器人处于关机状态，roboRIO 没有通电，或是 RSL 信号灯电路没有正确连接

36.5.2 roboRIO



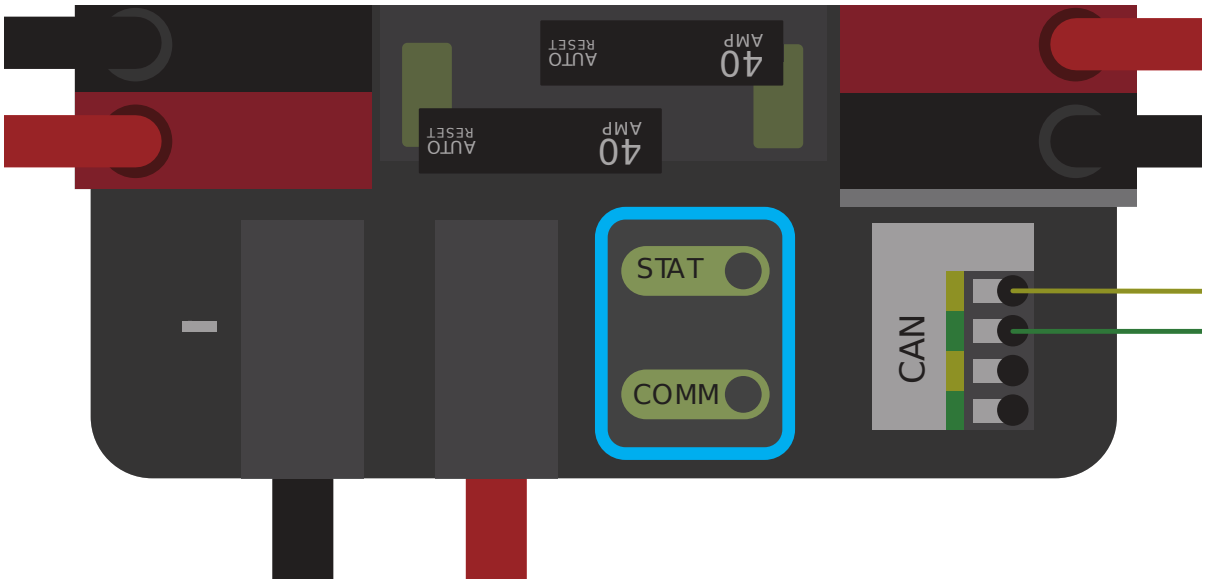
Power	绿色 (Green)	电源正常
	橙黄色 (Amber)	欠压保护被激活，所有输出都关闭了。
	红色 (Red)	电源故障，查找短路点
指示	在设备正在启动的时候常亮，过一段时间就会熄灭	
	双闪	软件错误，重新刷写 roboRIO 固件
	三闪	安全模式，尝试重启 roboRIO，如果无效请重新刷写 roboRIO 固件
	闪烁四次	在启动时软件崩溃两次，重启 roboRIO，如果无效请重新刷写固件
	一直闪烁或者常亮	无法恢复的错误
路由器	现在没有被使用	
通信	不亮	没有通信
	红灯常亮	与操控台取得通信，但是没有运行的代码
	红灯闪烁	紧急停止被触发
	绿色常亮	与操作台通信良好
模式	不亮	输出被关闭（机器人处于 Disabled 状态，欠压，等……）
	橙色	自动模式激活
	绿色 (Green)	远程操作模式激活
	红色 (Red)	测试模式激活
信号灯	**‘参考前文 <#robot-signal-light-rsl>‘_**	

36.5.3 OpenMesh 路由器



电源	蓝色	开机或通电
	蓝色闪烁	通电
Eth 连接	蓝色	连接
	蓝色闪烁	流量现状
无线网络	不亮	桥接模式，未链接或非 FRC 固件
	红色 (Red)	AP，未连接
	黄色/橙色	AP，已连接
	绿色 (Green)	桥接模式，已连接

36.5.4 配电板



PDP 指示/通信 LEDs

LED	频闪	慢
绿色 (Green)	无故障-机器人已启用	无故障-机器人已禁用
橙色	NA	Sticky 故障
红色 (Red)	NA	没有 CAN 通信

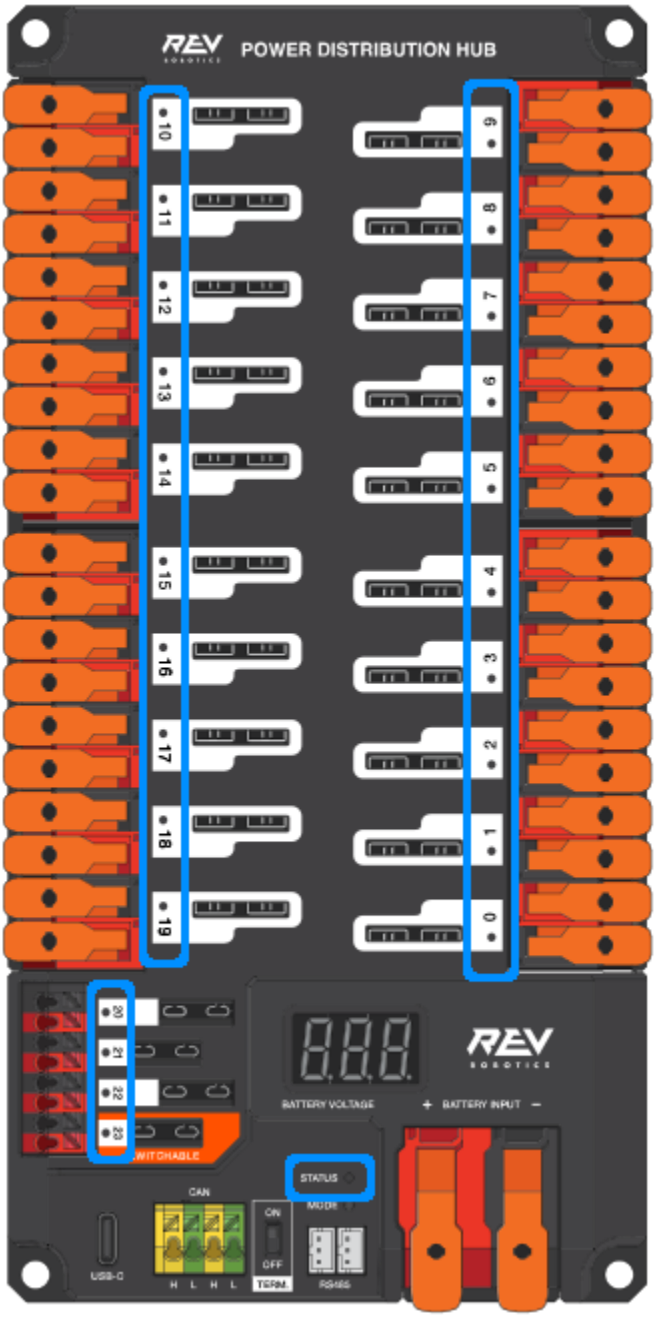
小技巧： 如果 PDP LED 显示不止一种颜色，请参阅下面的 PDP LED 特殊状态表。有关解决 PDP 故障的更多信息，请参见《PDP 用户手册》。

备注： Note that the No *CAN* Comm fault will occur if the PDP cannot communicate with the roboRIO via CAN Bus.

PDP 特殊状态

LED 颜色	问题
红色/橙色	受损的硬件
绿色/橙色	在引导程序中
无 LED	无电源/极性错误

36.5.5 Power Distribution Hub



备注： These led patterns only apply to firmware version 21.1.7 and later

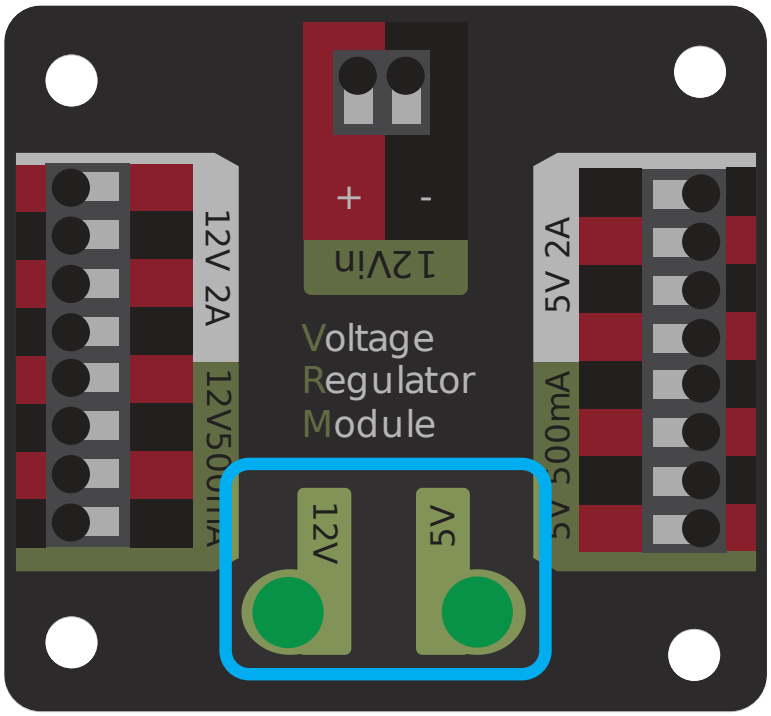
36.5.6 PDH Status LED

LED Color	状态
Blue Solid	Device on but no communication established
绿色常亮	Main Communication with roboRIO established
Magenta Blinking	Keep Alive Timeout
Solid Cyan	Secondary Heartbeat (Connected to REV Hardware Client)
Orange/Blue Blinking	Low Battery
Orange/Yellow Blinking	CAN Fault
Orange/Cyan Blinking	Hardware Fault
Orange/Red Blinking	Fail Safe
Orange/Magenta Blinking	Device Over Current

Channel LEDs

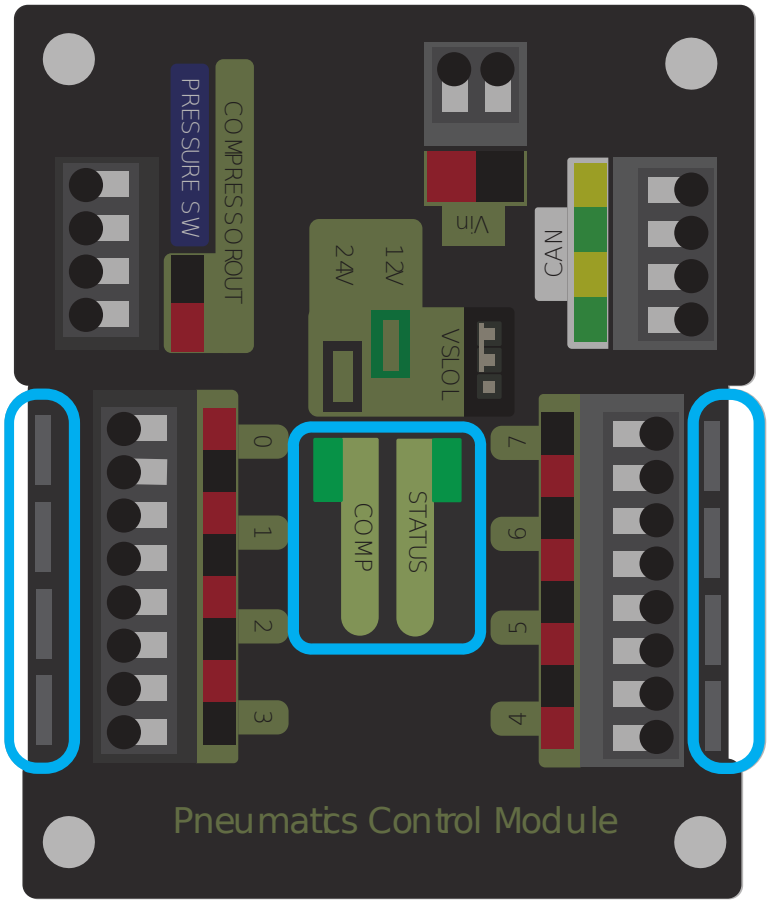
LED Color	状态
不亮	Channel has voltage and is operating as expected
红灯常亮	Channel has NO voltage and there is an active fault. Check for tripped or missing circuit breaker / fuse
红灯闪烁	Sticky fault on the channel. Check for tripped circuit breaker / fuse.

36.5.7 稳压器模块



VRM 上的指示 LED 标识两个电源的状态。如果电源正常运行，则 LED 应当亮绿色。如果 LED 不亮或暗淡，则输出可能短路或消耗过多电流。

36.5.8 气动控制模块（PCM）



PCM 状态指示 LED 灯

LED	频闪	慢	长
绿色 (Green)	无故障机器人已启用	Sticky 故障	NA
橙色	NA	Sticky 故障	NA
红色 (Red)	NA	没有 CAN 通讯或电磁阀故障（闪烁电磁阀索引）	压缩机故障

小技巧: 如果 PCM LED 显示不止一种颜色，请参阅下面的 PCM LED 特殊状态表。有关解决 PCM 故障的更多信息，请参见《PCM 用户手册》。

备注： 请注意，仅当设备无法与任何其他设备通信，PCM 和 PDP 可以彼此通信，而 roboRIO 不能彼此通信时，才会发生没有 CAN 通信故障。

PCM LED 特殊状态表

LED	问题
红色/橙色	受损的硬件
绿色/橙色	在引导程序中
无 LED	无电源/极性错误

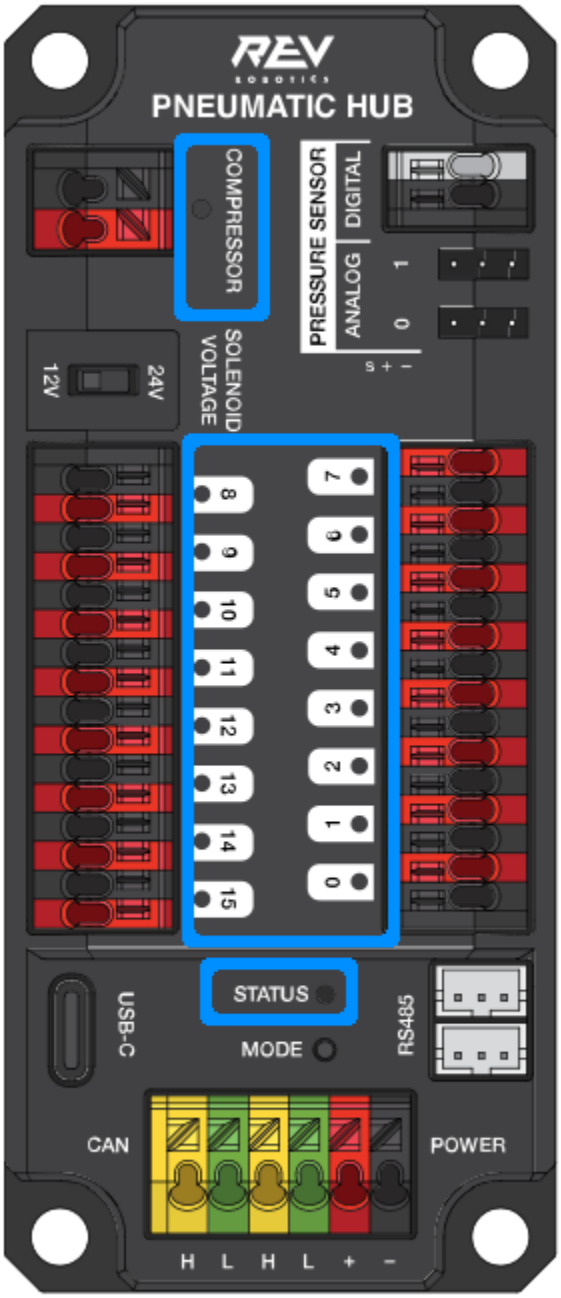
PCM Comp LED

这是 Compressor LED。当压缩机输出处于激活状态（压缩机当前处于开启状态）时，此 LED 呈绿色；当压缩机输出未处于活动状态时，该 LED 处于熄灭状态。

PCM 电磁阀通道 LED

如果启用了电磁阀通道，则这些 LED 呈红色点亮，如果禁用则不点亮。

36.5.9 Pneumatic Hub



备注： These led patterns only apply to firmware version 21.1.7 and later

PH Status LED

LED Color	状态
Blue Solid	Device on but no communication established
绿色常亮	Main Communication established
Magenta Blinking	Keep Alive Timeout
Solid Cyan	Secondary Heartbeat (connected to REV HW Client)
Orange/Blue Blinking	Hardware Fault
Orange/Yellow Blinking	CAN Fault
Orange/Red Blinking	Fail Safe
Orange/Magenta Blinking	Device Over Current
Orange/Green Blinking	Orange/Green Blinking

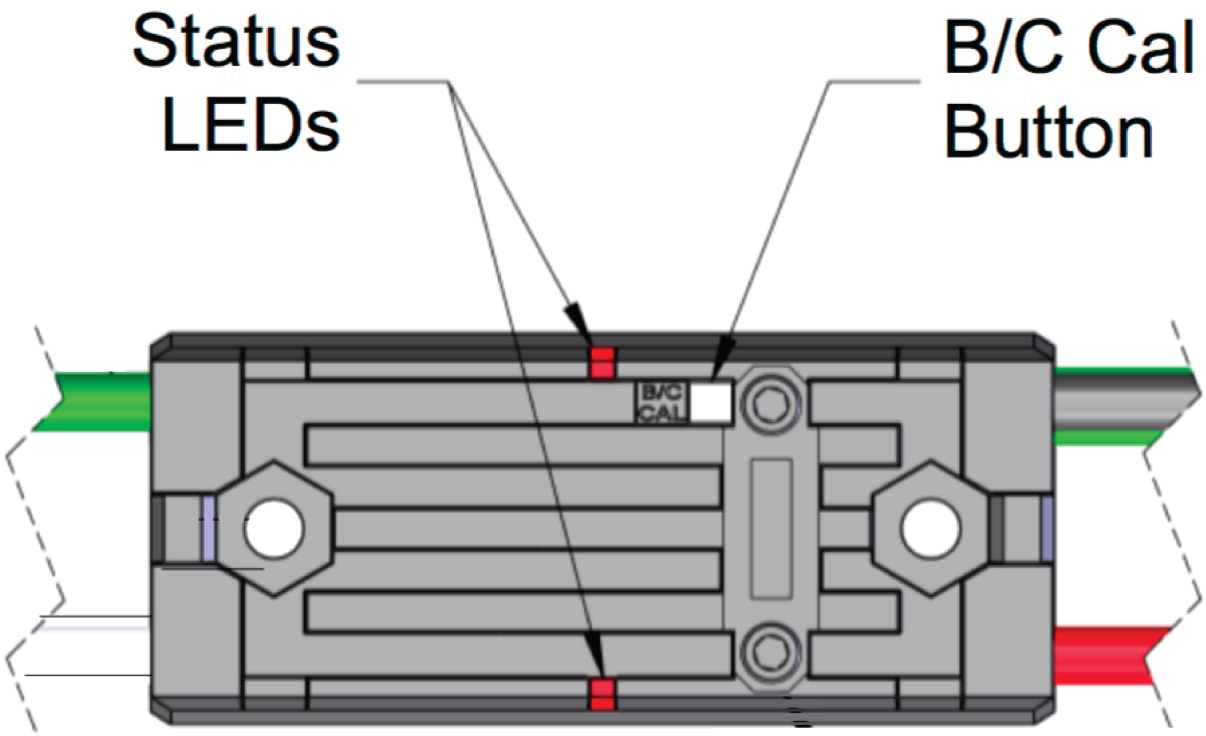
Compressor LED

LED Color	状态
绿色常亮	Compressor On
Black Solid	Compressor Off

Solenoid LEDs

LED Color	状态
绿色常亮	Solenoid On
Black Solid	Solenoid Off

36.5.10 Talon SRX 和 Victor SPX 和 Talon FX 电机控制器



正常操作期间的指示 LED

LEDs	颜色	Talon SRX 状态
都	绿色闪烁	应用前进油门时。闪烁速率与占空比成正比。
都	红色闪烁	应用了前进油门。闪烁速率与占空比成正比。
一无所有	一无所有	Talon SRX 没有被供电
LED 交替	关/橙色	检测到 CAN 总线，禁用了机器人
LED 交替	关/慢红	未检测到 CAN 总线/ PWM
LED 交替	关/快红	检测到故障
LED 交替	红色/橙色	受损的硬件
LED 频闪朝向（M-）	关/红色	正向限位开关或正向软限位
LED 频闪朝向（M +）	关/红色	反向限位开关或反向软限位
仅 LED1（最接近 M + / V +）	绿色/橙色	在引导程序中
LED 频闪朝向（M +）	关/橙色	热故障/关断（仅限 Talon FX）

校准期间的指示 LED

指示 LED 闪烁代码	Talon SRX 状态
闪烁红色/绿色	校正模式
绿色闪烁	成功校准
红色闪烁	失败校准

B/C CAL 闪烁代码

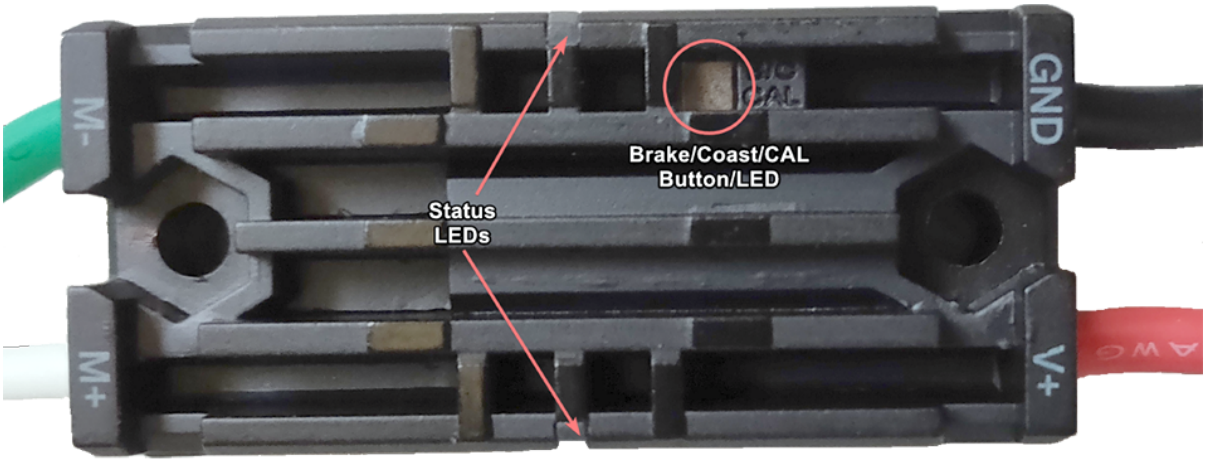
B/C CAL 按钮颜色	Talon SRX 状态
稳定的红色	刹车模式
不亮	滑行模式

36.5.11 SPARK-MAX 电机控制器

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 Victor-SP 电机控制器

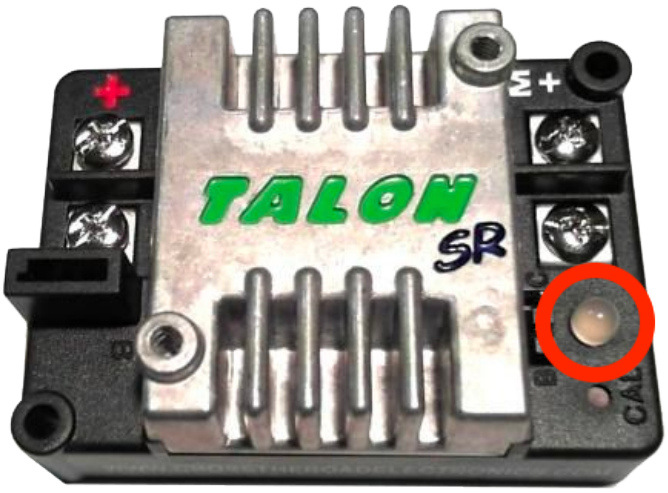


制动/惯性/校准按钮/ LED-如果控制器处于制动模式，则为红色；如果控制器处于惯性模式，则为关

状态

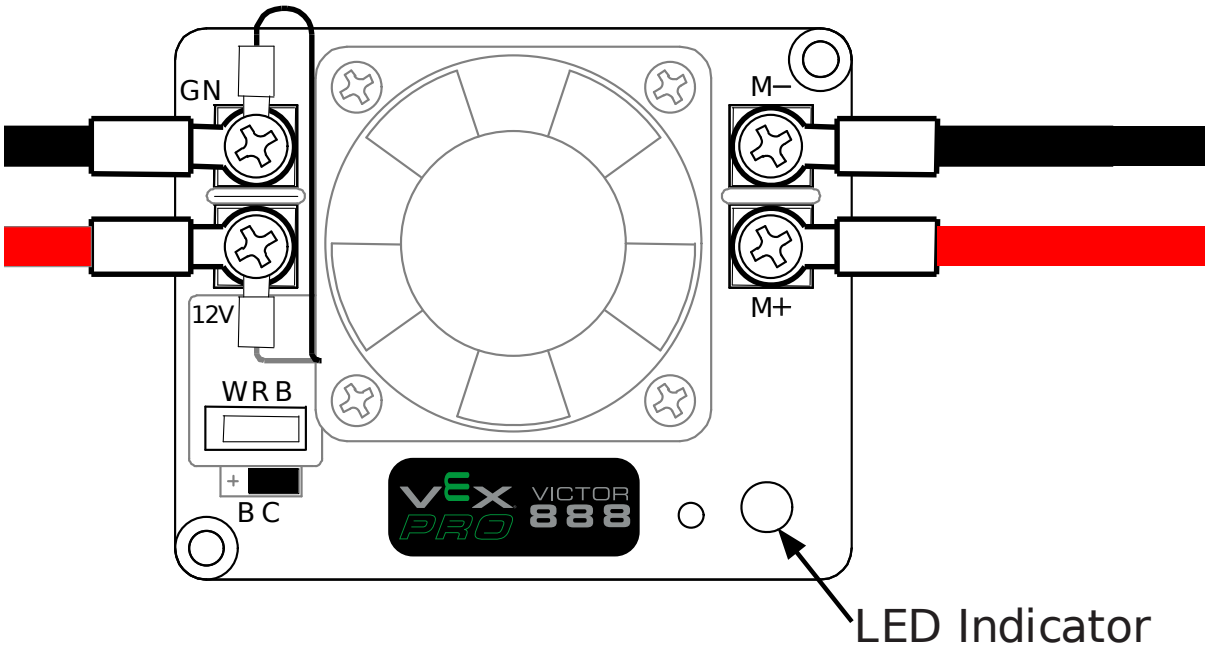
绿色 (Green)	稳定	全前向输出
	闪烁	正比输出电压
红色 (Red)	稳定	全反向输出
	闪烁	正比输出电压
橙色	稳定	FRC 机器人禁用，PWM 信号丢失或信号在死区范围内（+/- 4％输出）
红色/绿色	闪烁	准备进行校准。几次绿色闪烁表示校准成功，而多次红色表示校准失败。

36.5.14 Talon 电机控制器



绿色 (Green)	稳定	全前向输出
	闪烁	正比输出电压
红色 (Red)	稳定	全反向输出
	闪烁	与输出电压成正比
橙色	稳定	没有连接 CAN 设备
	闪烁	禁用状态，PWM 信号丢失，FRC 机械手禁用或信号在死区范围内（+/- 4%输出）
不亮		Talon 无输入电源
红色/绿色	闪烁	准备进行校准。几次绿色闪烁表示校准成功，而多次红色表示校准失败。

36.5.15 Victor888 电机控制器

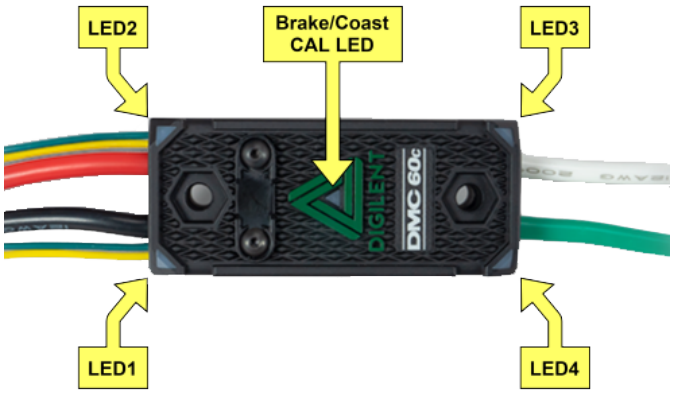


绿色 (Green)	稳定	全前向输出
	闪烁	成功校准
红色 (Red)	稳定	全反向输出
	闪烁	未成功校准
橙色	稳定	中性/制动
红色/绿色	闪烁	校准模式

36.5.16 Jaguar 电机控制器

LED 状态	模块状态
正常运行条件	
稳定的黄色	空档（速度设置为 0）
快速闪烁绿色	前进
快速闪烁红色	反向
稳定的绿色	全速前进
稳定的红色	全速倒车
故障状态	
缓慢闪烁黄色	伺服或网络链接丢失
快速闪烁黄色	无效 CAN ID
缓慢闪烁红色	电压，温度或限位开关故障情况
缓慢闪烁红色和黄色	目前故障状态
校准或 CAN 状态	
快速闪烁红色和绿色	校准模式激活
闪烁红色或黄色	校准模式失败
闪烁绿色和黄色	校准模式成功
缓慢闪烁绿色	CAN ID assignment mode
快速闪烁黄色	当前的 CAN ID（计数闪烁以确定 ID）
闪烁黄色	CAN ID 无效（即设置为 0）等待分配有效 ID

36.5.17 Digilent DMC-60



DMC60C 包含四个 RGB（红色，绿色和蓝色）LED 和一个 “制动/滑行 CAL” LED。四个 RGB LED 位于角落，用于指示正 常操作期间以及发生故障时的状态。制动/滑行 CAL LED 位于三角形的中心，该三角形位于外壳的中心，用于指示当前的制动/滑行设置。当中心指示灯熄灭时，设备将在惯性停车模式下运行。当中心 LED 点亮时，设备将以制动模式运行。按下三角形的中心，然后释放按钮，可以切换制动/滑行模式。

开机时，RGB LED 蓝色点亮并持续变亮。这持续约五秒钟。在此期间，电机控制器将不会响应输入信号，也不会启用输出驱动器。初始上电完成后，设备开始正常运行，并且 RGB LED 上显示的内容取决于所施加的输入信号以及当前故障状态。假设未发生任何故障，则 RGB LED 的功能如下：

应用 PWM 信号	LED 状态
没有输入信号或无效的输入脉冲宽度	顶部（LED1 和 LED2）和底部（LED3 和 LED4）之间交替显示红色和熄灭的 LED。
中性输入脉冲宽度	所有 4 个 LED 均亮橙色。
正输入脉冲宽度	LED 呈顺时针圆形闪烁绿色（LED1→LED2→LED3→LED4→LED1）。LED 更新速率与输出的占空比成正比，并随着占空比的增加而增加。在 100％占空比下，所有 4 个 LED 均呈绿色亮起。
反输入脉冲宽度	LED 以逆时针圆形模式闪烁红色（LED1→LED4→LED3→LED2→LED1）。LED 更新速率与输出的占空比成正比，并随着占空比的增加而增加。在占空比为 100％时，所有 4 个 LED 均亮红色。

CAN 总线控制状态	LED 状态
未检测到输入信号或 CAN 总线错误	顶部（LED1 和 LED2）和底部（LED3 和 LED4）之间交替显示红色和熄灭的 LED。
在最近的 100ms 或指定的 modeNoDrive 最后一个控制帧内未接收到 CAN 控制帧（输出禁用）	顶部（LED1 和 LED2）和底部（LED3 和 LED4）之间交替显示橙色和熄灭的 LED。
最近 100 毫秒内收到的有效 CAN 控制帧。指定的控制模式导致中性占空比应用于电机输出	所有 4 个 LED 均呈橙色常亮。
最近 100 毫秒内收到的有效 CAN 控制帧。指定的控制模式导致正占空比为电动机输出	LED 呈顺时针圆形闪烁绿色（LED1→LED2→LED3→LED4→LED1）。LED 更新速率与输出的占空比成正比，并随着占空比的增加而增加。在 100% 占空比下，所有 4 个 LED 均呈绿色亮起。
最近 100 毫秒内收到的有效 CAN 控制帧。指定的控制模式导致负占空比为电机输出	LED 以逆时针圆形模式闪烁红色（LED1→LED4→LED3→LED2→LED1）。LED 更新速率与输出的占空比成正比，并随着占空比的增加而增加。在占空比为 100% 时，所有 4 个 LED 均亮红色。

故障颜色指示灯

当检测到故障情况时，输出占空比降至 0%，并发出故障信号。然后，输出将保持禁用状态 3 秒钟。在此期间，板载 LED（LED1-4）用于指示故障情况。通过在顶部的 LED（LED1 和 LED2）和底部的 LED（LED3 和 LED4）之间切换来指示故障状态，这些 LED 呈红色亮起并且熄灭。底部 LED 的颜色取决于当前正在激活的故障。下表描述了底部 LED 的颜色如何映射到当前活动的故障。

颜色	过高温度	电压不足
绿色 (Green)	激活	不亮
蓝色	不亮	激活
青色/水色	激活	激活

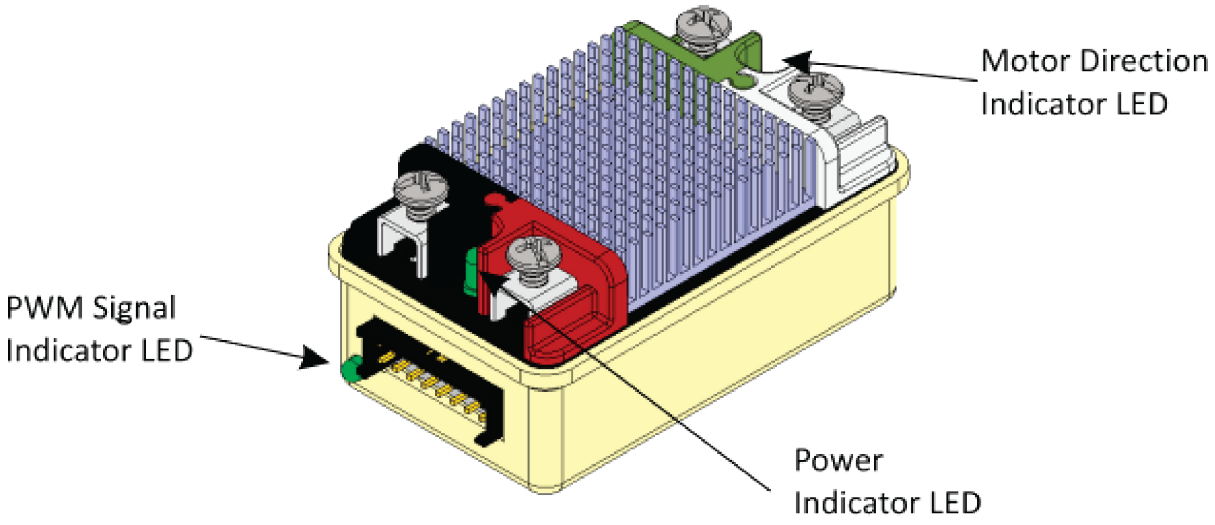
中断/滑行模式

当中心指示灯熄灭时，设备将以惯性停车模式运行。当中心 LED 点亮时，设备将以制动模式运行。按下三角形的中心然后释放按钮可以切换制动/滑行模式。

36.5.18 Venom 电机控制器

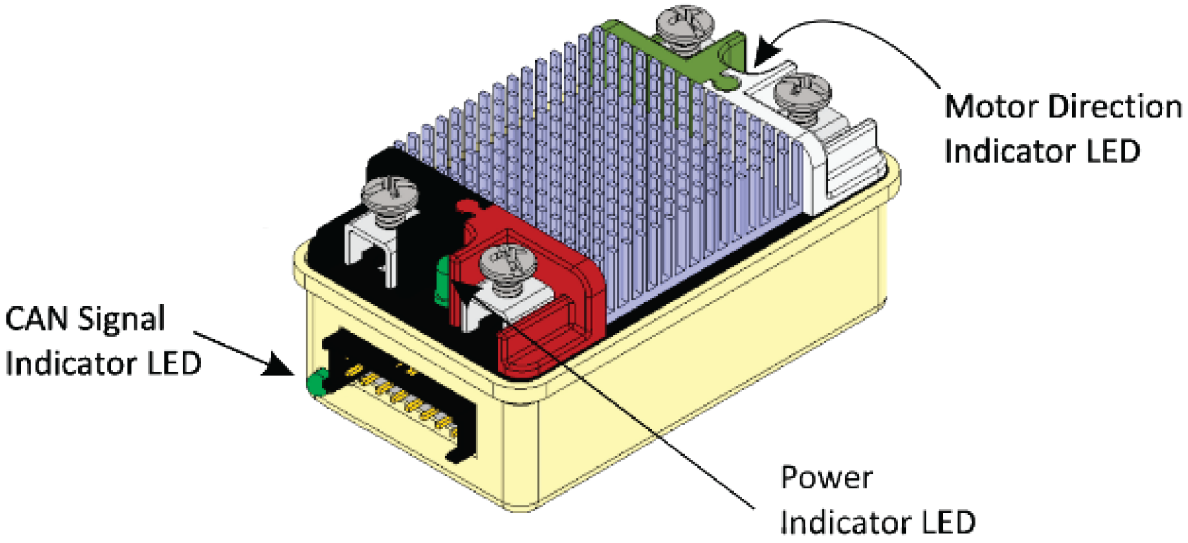
LED Pattern	Description
	Venom is initializing. This state should last less than 40ms after power up.
	The 'Identify Device' feature is active. This pattern in used to locate a particular Playing With Fusion device when multiple are installed on a robot. See the Motor Configuration section for more information
	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)



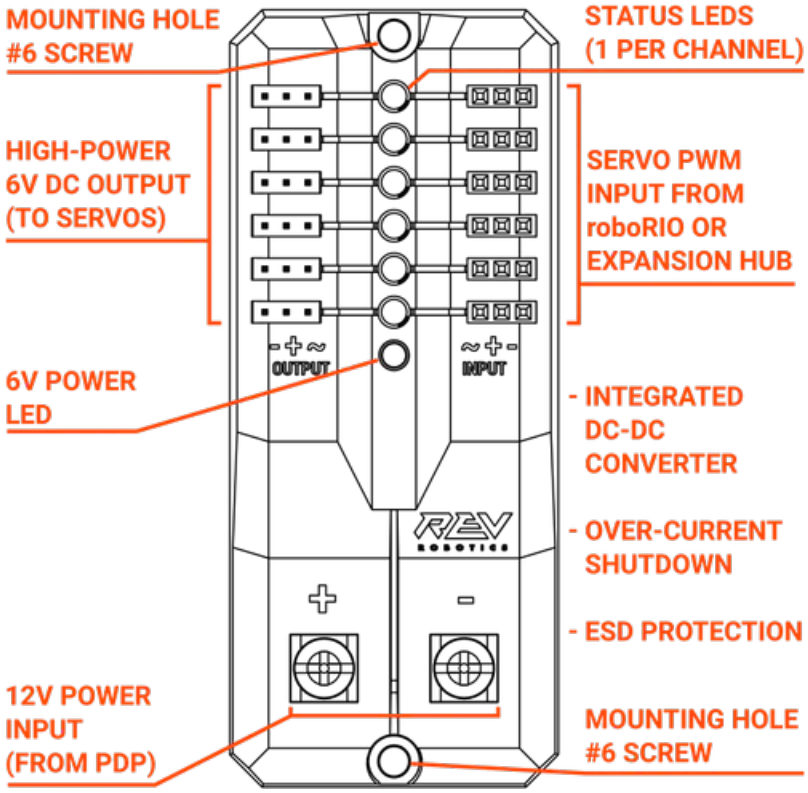
电源 LED	不亮	没有电源供应
	红色 (Red)	供电
电机 LED	红色 (Red)	前进方向
	绿色 (Green)	倒车方向
PWM 信号 LED	红色 (Red)	检测不到有效 PWM 信号
	绿色 (Green)	检测到有效 PWM 信号

36.5.20 Mindsensors SD540C (CAN 总线)



电源 LED	不亮	没有电源供应
	红色 (Red)	供电
电机 LED	红色 (Red)	前进方向
	绿色 (Green)	倒车方向
CAN 信号 LED	快速闪烁	没有连接 CAN 设备
	不亮	连接到 roboRIO 和开启 Driver station

36.5.21 REV Robotics Servo 电源模块



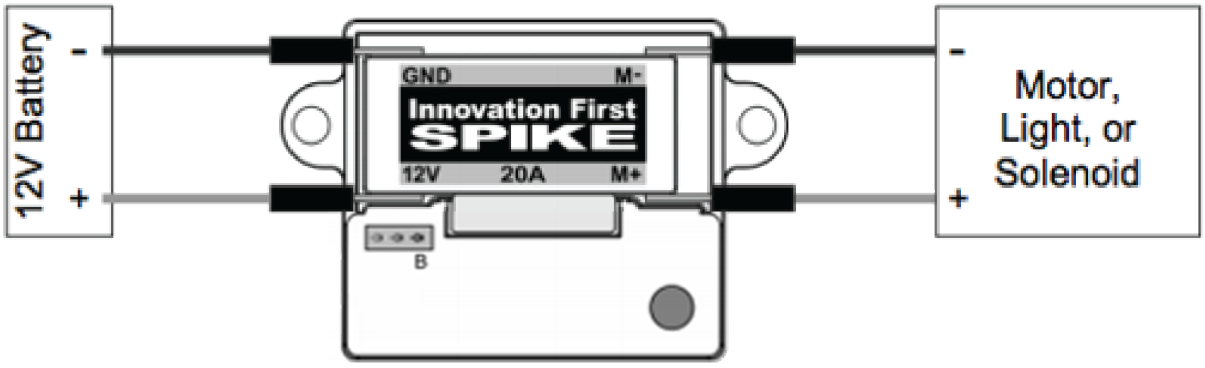
指示 LED

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.

状态	模式
无信号	橙黄色闪烁
左/反向信号	稳定的红色
中间/中性信号	稳定的橙黄色
右/前信号	稳定的绿色

- 接通电源后，6V 电源 LED 熄灭，暗淡或闪烁 = 过电流关闭

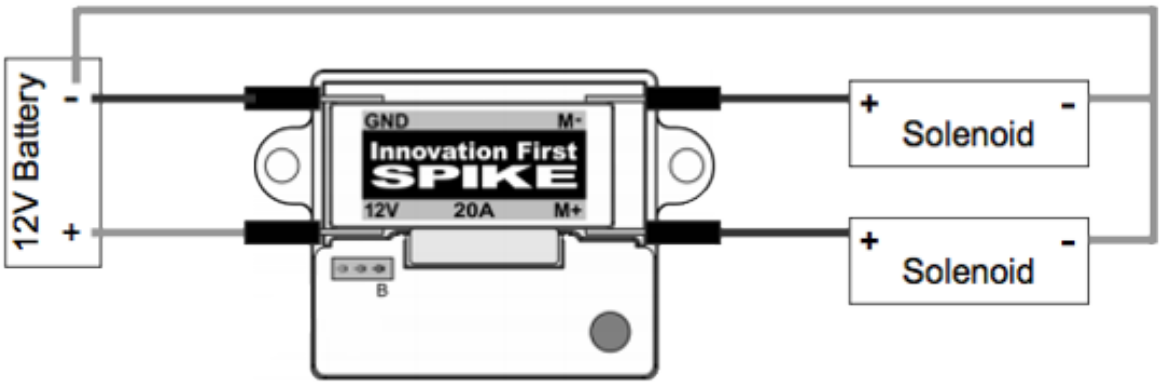
36.5.22 峰值继电器配置为电机、照明或电磁开关



输入		输出		指示器	马达功能
前进（白色）	倒车（红色）	M+	M-		
不亮	不亮	GND	GND	橙色	关/制动状态（默认）
激活	不亮	+12v	GND	绿色 (Green)	电机向一个方向旋转
不亮	激活	GND	+12v	红色 (Red)	电机反向旋转
激活	激活	+12v	+12v	不亮	关/制动状态

备注： “制动状态”是指由于电动机输入短路而导致的电动机动态停止。进入关闭状态时，此状态不是可选的。

36.5.23 峰值继电器配置为一个或两个螺线管



输入		输出		指示器	马达功能
前进（白色）	倒车（红色）	M+	M-		
不亮	不亮	GND	GND	橙色	两个电磁阀均关闭（默认）
激活	不亮	+12v	GND	绿色 (Green)	连接到 M + 的电磁阀为 ON
不亮	激活	GND	+12v	红色 (Red)	连接到 M -的电磁阀为 ON
激活	激活	+12v	+12v	不亮	两个电磁阀都打开

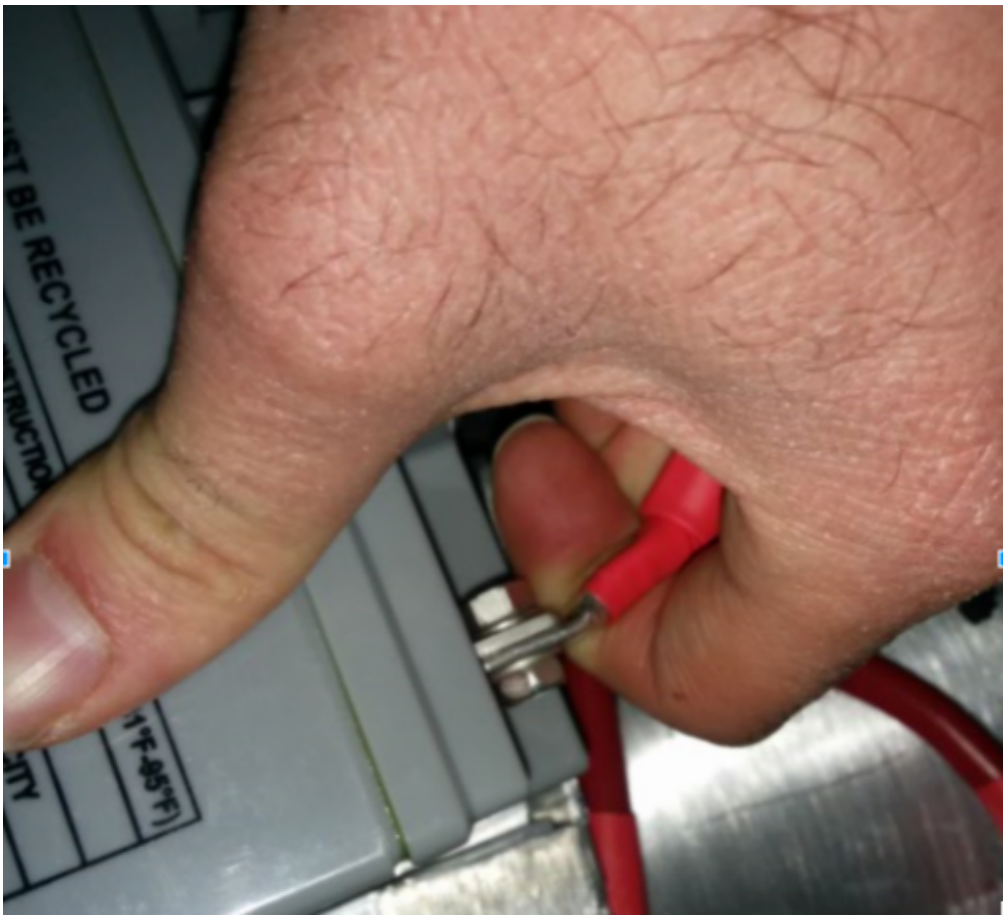
36.5.24 CANCoder Encoder

LED Color	LED Brightness	CAN Bus detection	Magnet Field Strength	Description
不亮	不亮			CANCoder is not powered
Yellow/Gree	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 机器人的预防式故障排除

备注： 在 *FIRST*® 机器人竞赛中，机器人在场地行驶时承受很大的压力。确保连接的牢固、零件稳妥的固定在合适的地方上以及一切都已安装好是非常重要的，好让机器人在场上活蹦乱跳时不会散架。

36.6.1 检查电池连接

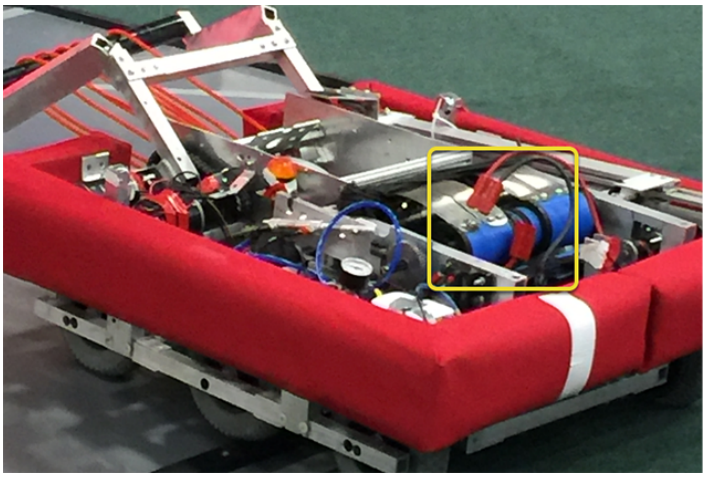


在这些示例中，应该覆盖电池连接的胶带已被移除，以说明发生了什么情况。在您的机器人上，连接应该被遮盖。

摆动电池线束插头。感到松动通常是因为螺丝松动，或者有时压接没有完全闭合。您只能以此发现非常严重的问题，因为通常电工胶带会使连接变硬，觉得连接仍然结实。使用电压表或电池喙将有助于解决此问题。

以 90 度的力向电池电缆施加相当大的力，以尝试移动电缆离开电池的方向，如果成功，则连接不够紧，应重新进行连接。该文章 <docs/hardware/hardware-basics/robot-battery:Robot Battery Basics> 具有更多详细的电池信息。

36.6.2 将电池固定到机器人上



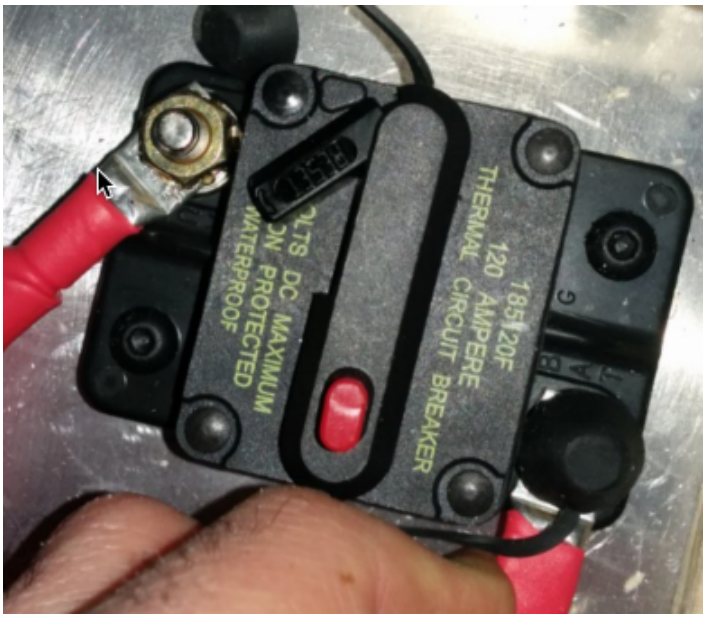
In almost every event we see at least one robot where a not properly secured battery connector (the large Anderson) comes apart and disconnects power from the robot. This has happened in championship matches on the Einstein and everywhere else. Its an easy to ensure that this doesn't happen to you by 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 固定电池连接器和主电源线

松开的机器人侧电池连接器（较大的 Anderson SB）可以在更换电池时拉紧主电源线。如果主电源线松动，则该“拉力”可以完全回到连接到 120 安培断路器或配电面板（PDP）的压接突耳，弯曲该突耳，随着时间的推移，会导致突耳端部断裂从疲劳。放置一对将主电源线连接到机箱的绑带，并用螺栓固定机器人侧的电池连接器可以防止这种情况，并且可以更轻松地连接电池。

36.6.4 主断路器（120 安培断路器）

备注： 确保将螺母牢固拧紧，并且将断路器安装在刚性元件上。



施加较大的扭曲力以尝试旋转压接的接线片。如果凸耳旋转，则螺母不够紧。拧紧螺母后，再次尝试旋转凸耳，以进行重新测试。

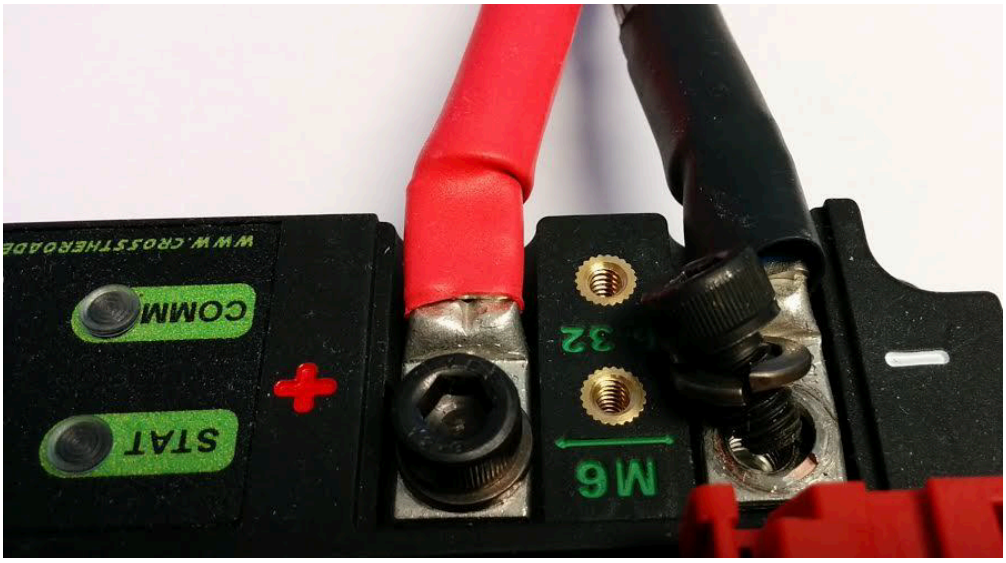
原始螺母具有星形锁定功能，该功能会随着时间的流逝而磨损：尤其是如果您的机器人侧电池连接器未安装在机箱上，则可能需要每隔几下匹配项进行检查。

螺母通常是不常见的 1 / 4-28 螺纹：如果更换螺母，请确保这是正确的。

由于金属螺柱只是模制在外壳中，因此有时您可能会折断螺柱。不要强调，只需更换组件即可。

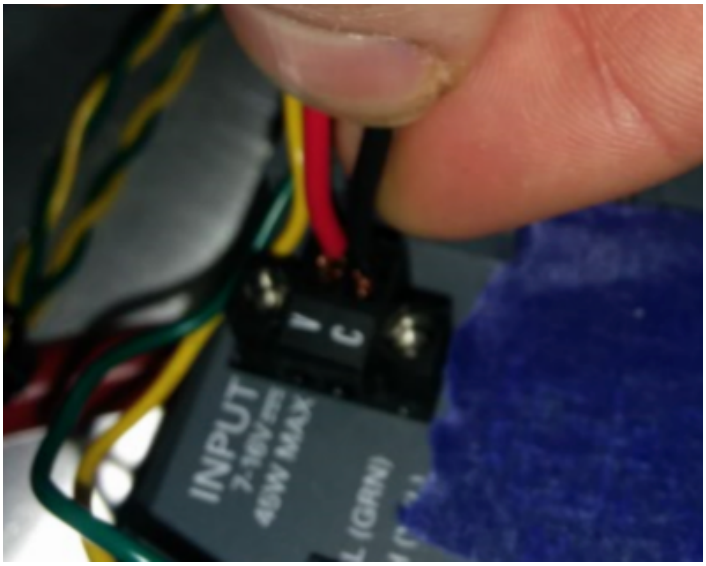
当经受多个比赛季节时，主断路器容易受到振动和使用引起的疲劳损坏，并且在受到冲击时会开始打开。每次触发热熔断器功能时，它会逐渐变得更容易跳闸。每个赛季，许多老队伍都会更换一个新的主断路器进行比赛，并随身携带备件。

36.6.5 配电板（PDP）



确保将弹簧垫圈放在 PDP 螺钉的下面，但肉眼很难确认，有时甚至根本看不到。您可以拿开外壳进行检查。也可以将红色和黑色电线抓成一把，有时会帮助您会发现真正失去连接的地方。

36.6.6 拖拽测试



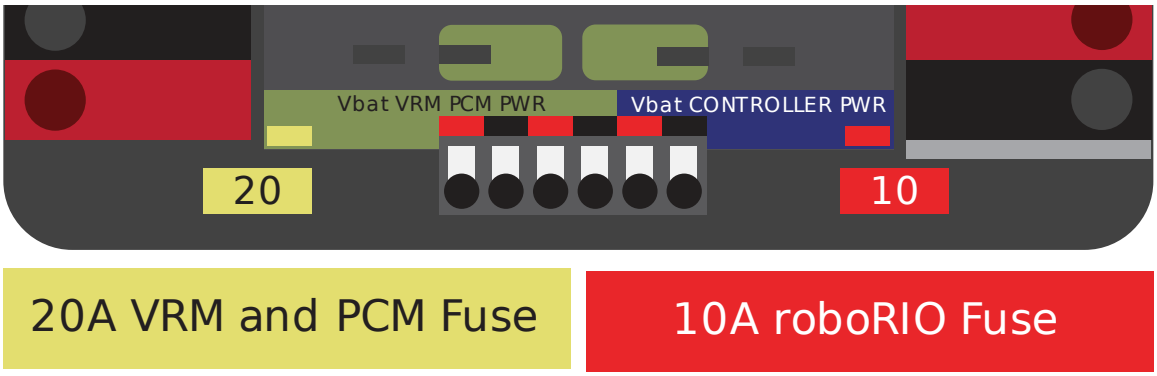
如图拖拽 Weidmuller 触点和电源、压缩机输出、roboRIO 电源连接器及路由器电源的连接处是至关重要的。确保没有任何连接断开。

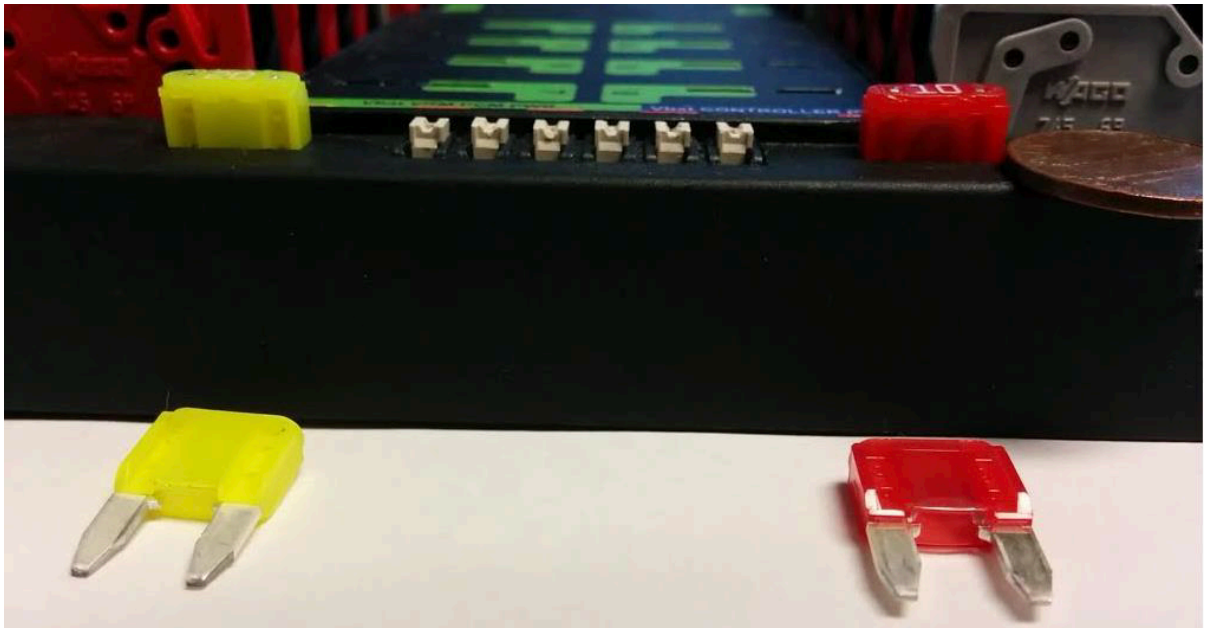
在相近和导线过长（剥开了额外长度的导线）的 Weidmuller 连接处寻找可能或即将发生的短路。

铲形接头也可能由于不适当的压接而失效，因此也应进行拖拽测试。

36.6.7 插片保险丝

确保将 20A 保险丝（黄色）放在左侧，将 10A 保险丝（红色）放在右侧。





警告： 注意确保保险丝完全插入保险丝插座中。保险丝应至少插至下图所示深度（不同品牌的保险丝具有不同的引线长度）。几乎不可能徒手卸下保险丝（不使用钳子）。如果这一步出错，机械人/路由器可能会出现间歇性的连接问题。

如果您可以用手卸下插片保险丝，则说明它们没有完全插入。确保将它们完全固定在 PDP 中，以免它们在机器人操作过程中弹出。

36.6.8 roboRIO 的碎屑

碎屑是指由机械加工产生的石材、金属或其他材料的屑屑。通常机器人在控制系统部件安装到位后仍需修改。roboRIO 的电路板涂有涂层，但这并不能保证金属碎屑绝不会让内部的走线或组件短路。在这种情况下，必须格外小心检查 roboRIO 或其他任何组件不会出现任何碎屑。特别是裸露的 3 针接口，这可以让碎屑内部。用手电筒大约扫过四个面通常足以发现非常糟糕的问题区域。

36.6.9 路由器公头

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 以太网电缆

如果 RIO 接路由器的以太网电缆缺少将连接器锁定的固定夹，请换一根电缆。这是一个每次比赛中都会发生几次的普遍问题。确保电缆连接牢固。夹子很容易会断掉，特别是在通过狭窄路径时会钩住某些东西然后折断。

36.6.11 电缆松动

必须拧紧电缆，尤其是路由器电源和以太网电缆。路由器电源线并没有很大的摩擦力（即使使用正确的插头），如果松弛电缆自由摆动，电线就会掉落。

以太网电缆也很重，如果可以自由摆动，则塑料夹可能不足以将以太网针连接器固定在电路中。

36.6.12 重现 Pit 区中的问题

除了在给机器人充电和连接时正常地晃动电线外，建议拿起机器人的一侧拿起并放手。在场地中驾驶，特别是对抗防守方，通常会非常暴力，这有助于确保没有东西会脱落。机器人在 Pit 区失败总比在比赛中失败好。

进行此测试时，请务必使用以太网进行网络连接，而不要使用 USB 连接，否则无法测试所有的关键路径。

36.6.13 检查固件和版本

机器人检查员会做这些检查，但是您也应该进行检查，这可以帮助机器人检查员找到问题，他们也会对此对您欣赏有加。并且它能保证您使用的是最新的、已修正错误的代码。您不会希望因为机器人上的控制系统软件过期而输掉比赛。

36.6.14 Drivers Station 检查

我们经常看到 Drivers Station 出现问题。你应该：

- 始终将笔记本电脑的电源线带到现场，无论电池的电池质量如何，保障您可以在场地内充电。
- 检查电源和睡眠设置，关闭睡眠和休眠模式、屏幕保护程序等。
- 关闭 USB 设备的电源管理（设备管理器）
- 关闭以太网端口的电源管理（设备管理器）
- 关闭 Windows Defender
- 关闭防火墙
- 在场地外，请关闭除 DS /仪表板以外的所有应用程序。
- 确认开始菜单（右下角）中的应用程序托盘中没有多余的东西在运行

36.6.15 方便的工具

机器人内部不见得有足够的光线、至少没有足够的光线来检查关键连接点，因此请考虑使用手持式 LED 手电筒检查机器人上的连接。可从家里的车库或任何硬件/汽车商店购买。

WAGO 工具是用于重做 Weidmuller 绞线连接的好工具。通常我会给队伍展示一次，然后在插入绞合线时让他们使用 WAGO 工具压下白色柱塞来完成剩下的工作。WAGO 工具的角度在这种事情上有优秀的发挥。

36.7 机器人电池基础

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.

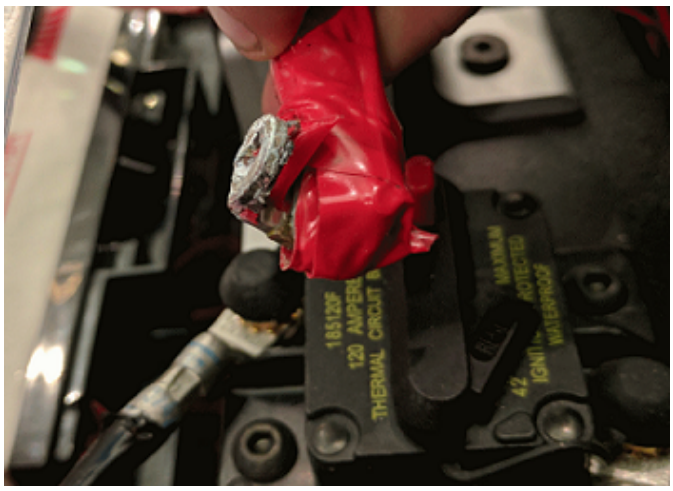
36.7.1 COTS 电池

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 电池安全与处理

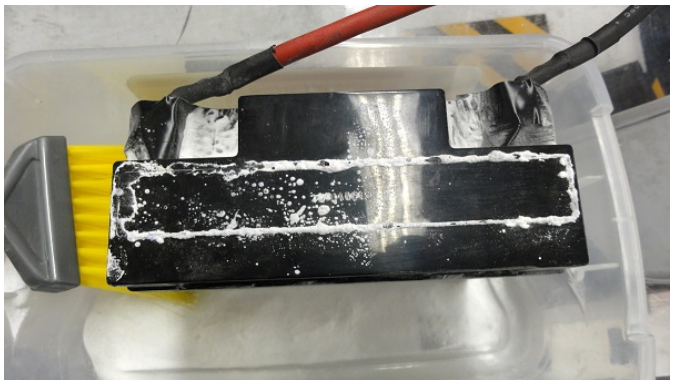
健康的电池始终“打开”，并且终端始终“通电”。如果极性短接（例如，扳手或气雾剂可能掉落并桥接两个裸端子之间的间隙），则所有存储的能量将以危险的电弧释放。这种风险带来了广泛的最佳实践，例如覆盖存储中的端子，一次仅露出一个端子或在其中一个极性上进行揭露和工作，保持 SB 触点完全插入连接器等。

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*.



跌落电池可能会使内板弯曲并导致性能问题，产生凸起，甚至使电池盒打开裂开。尽管大多数 FRC 电池使用吸收性玻璃毡 [AGM] 或凝胶技术来提高安全性和性能，但当电池被刺穿时，仍可能泄漏少量电池酸。这是 FIRST 建议团队使用电池溢出套件的原因之一。

最后，某些没有“维护模式”功能的较旧的电池充电器可能会“过度充电”电池，从而使部分电池酸沸腾。



损坏的电池应尽快被安全处置。所有销售大型 SLA 电池（如汽车电池）的零售商都应能够为您处理。他们可能会收取少量费用，或提供少量“核心费用退款”，具体取决于您所在州的法律。

危险： 请勿尝试“修复”损坏的或无法正常工作的电池。

36.7.3 电池构造与工具

电池引线

电池引线必须为铜，最小尺寸（横截面）为 6 AWG（16mm2，7 SWG），最大长度为 12，颜色标记为极性并要带有 Anderson SB 连接器。标准 6AWG 铜引线带有粉红色/红色 SB50 电池引线在零件套件中，由 FRC 供应商出售。

引线电缆

允许镀锡，退火或涂层的铜。请勿使用 CCA（覆铜铝），铝或其他非铜基金属。导体金属通常与其他电缆额定值一起印在绝缘层的外部。

几乎所有机器人都可以使用 6AWG 的线径，并适合标准 SB50 触点。少数团队采用较大的线径以获得边际性能优势。

股数较高的线材（有时以“Flex”或“weld wire”出售）具有较小的弯曲半径，这使布线更容易，并且疲劳极限更高。没有股数要求，但与 19 / 0.0372（19 股连接）相比，84/25（84 股“柔性”连接线）和 259/30（259 股“焊接线”）都更 * 容易 * 金属丝）。

绝缘必须按照游戏手册进行颜色编码：自 2021 年起，+ 12Vdc 电线必须为带条纹的红色，白色，棕色，黄色或黑色，接地线（回线）必须为黑色或蓝色。没有明确的绝缘温度额定值要求，但是绝缘变黑或损坏意味着需要更换电线：即插即用，105°C 足够，更低的温度适用于几乎所有机器人。没有绝缘额定电压的要求，越低越好，绝缘越薄越好。

SB 连接器

Anderson SB 连接器可以是标准的粉红色/红色 SB50，也可以是其他 Anderson SB 连接器。*强烈建议* 团队使用粉红色/红色 SB50，以实现互操作性：其他颜色和尺寸的外壳将无法相互配合，并且您将无法借用电池或充电器。

请按照制造商的说明压接触点，然后将导线组装到 Anderson SB 连接器中。小型平头螺丝刀可以帮助插入触点（按在触点上，而不是插在电线绝缘上），或者如果触点插入错误的插槽或朝上，则可以帮助松开内部闩锁。

电池接线片

可通过在线商店和通过电气供应商获得# 10 螺栓（或 M5）电池接线片的压缩接线片（“压接接线片”）（约 0.2 英寸或约 5mm 的孔直径），按 AWG（或 mm2）和可接受的电线尺寸出售。接线柱直径（“螺栓尺寸”，“孔直径”）。高端供应商还将在其接线片目录中区分标准（⒡19）和 Flex（> 80）股数。一些供应商除提供直角接线片外，还提供更常见的直式样式，请按照制造商的说明压接凸耳。

螺丝接线端子是合法的，但我们不建议这样做。如果使用螺钉式接线端子，请使用正确的尖端尺寸螺丝刀拧紧端子。经常检查端子的密封性，因为它们可能会随着时间的流逝而松动。

电池引线接线柱连接

10 或 M5 螺母和螺栓将电池引线接线片连接到电池凸耳。

> **警告：** 接线片和接线片必须直接铜与铜得接触：请勿放置任何类型的垫圈将它们分开。

某些电池的包装中带有凸耳螺栓：他们是可以使用的，也可以用强度更高的合金钢螺栓代替。除了尼龙锁紧（“nylock”）螺母外，最好添加一个功能性锁紧垫圈，例如# 10 星型垫圈或 nordlock 垫圈系统。每个连

接中只能使用一种样式的锁紧垫圈。即使制造商在包装中提供了开口环锁紧垫圈，您也无需使用它们。

这些连接必须非常紧密，以确保可靠性。接线片在运行过程中的任何移动都可能会中断机器人的电源，从而导致机器人重新启动以及持续 30 秒或更长时间的现场断开连接。

为了安全起见，此连接也必须被完全覆盖。可以使用电工胶带，但建议在整个连接过程中使用热收缩。高收缩率（最低 3：1，建议 4：1）将使热收缩更容易进行。内衬热收缩胶是允许的。确保 * 所有 * 铜被覆盖！如果显示出一些铜，则必须用电工胶带“修补”热收缩。

电池充电器

有许多适用于 12V SLA 电池的优质 COTS “智能” 电池充电器，每个电池的额定电流为 6A 或更小，具有 “维护模式” 功能。额定电压超过 6A 的充电器不允许进入 FRC 维修区。

比赛中使用的充电器必须是 Anderson SB 连接器。使用合适尺寸的接线螺母或螺钉端子将 COTS SB 连接器电池引线连接到充电器引线上，既快速又简单（请确保用热收缩或胶带覆盖任何裸露的铜）。如果小组具有压接功能，则 SB 连接器触头也可用于较小的电线尺寸。

> **警告：** 安装 SB 后，请先用万用表仔细检查充电器的极性，然后再插入第一个电池。

一些 FRC 供应商出售的充电器带有预先连接的红色 SB50 连接器。

电池评估工具

充电器

如果您的电池充电器具有维护模式指示灯，例如绿色 LED，则可以使用该指示灯来告诉您是否准备就绪。某些充电器会定期在 “充电” 和 “就绪” 之间循环。这是一种 “维护” 行为，这有时与电池明明冷却并能够接受更多电荷有关。

驾驶室显示和记录

插入机器人并连接至驱动程序工作站便携式计算机后，电池电压将显示在 NI Driver Station 软件上。

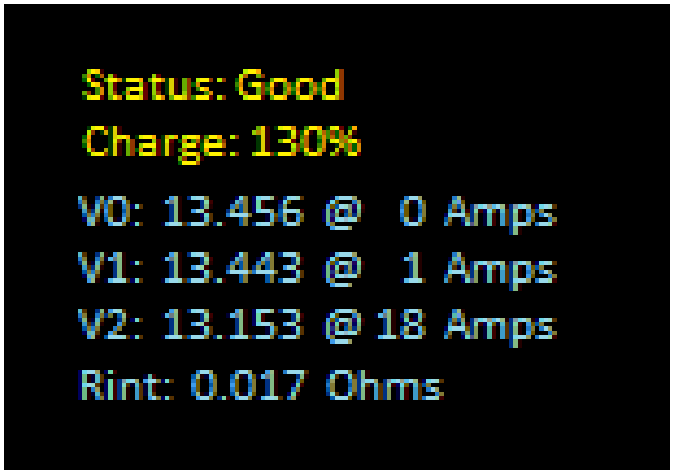
完成驾驶会话后，您可以在 Log Viewer 中查看电池电压。<docs/software/driverstation/driver-station-log-viewer:Using the Graph Display>‘

手持式 ** 电压表 ** 或 ** 万用表 **

从断开连接的电池的 SB 连接器上的探针读取的电压读数将为您提供 Voc（电压开路或 “浮动电压”）处于 “未加载” 状态的快照。通常，不建议使用 Voc 来了解电池的健康状况：开路电压不如内部电阻和负载测试仪（或电池分析仪）提供的特定负载下的电压组合有用。

负载测试器

电池负载测试仪可以用作确定电池详细准备情况的快速方法。它可能提供以下信息：开负载电压，负载电压，内部电阻和充电状态。这些指标可用于快速确认电池已准备就绪，甚至可以帮助确定电池的某些长期问题。



理想的内部电阻应小于 0.015 欧姆。大多数电池的制造商规格为 0.011 欧姆。如果电池电量高于 0.020 欧姆，则最好考虑不要在比赛中使用该电池。

如果电池在较高的测试电流负载下显示出明显较低的电压，则可能无法完成充电，或者可能需要停用。

36.7.4 了解电池电压

“12V 电池”不是 12.0V。

充满电后，电池的开路电压（Voc）可以在 12.7 至 13.5 伏之间。开路电压是在“未连接 * 的情况下测量的。

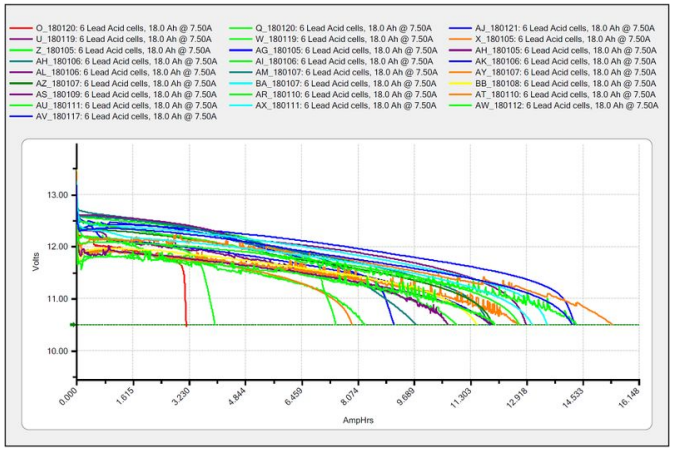
一旦连接了负载（例如机器人），并且有大量电流流过，电池电压就会下降。因此，如果您用电压表检查电池，并且其读数为 13.2，然后将其连接至机器人并打开电源，则其读数会更低，在 Driver Station 显示屏上可能为 12.9。这些数字将随每个电池和特定的机器人而有所不同，请参见下面的特性描述。一旦您的机器人开始运行，它将拉动更多的电流，并且电压将进一步下降。

比赛前，应在闲置的机器人上读取 12.5V 的电池并进行充电。务必在机器人开始达到节电安全阈值之前将电池更换（在驾驶员站显示屏上停留在低电压下），因为频繁进入低电压范围可能会造成电池永久损坏；这种行为可能会在各种 Voc 状态下发生，具体取决于电池的运行状况，电池制造商和机器人的设计。电池充电状态应保持 50%以上，以延长电池寿命。

电池电压和电流还取决于温度：凉爽的电池就是快乐的电池。

电池特性

电池分析仪可用于详细检查和比较电池性能。



它将提供一段时间内电池性能的图表。该测试需要花费大量时间（大约两个小时），因此不太适合在比赛中进行测试。建议每年对每个电池进行此测试，以监视和跟踪其性能。这将决定应如何使用它：比赛，练习，测试或处置。

在标准 7.5 安培的测试负载下，竞赛电池应至少具有 11.5 安培的小时额定值。少于该限制的任何内容仅应用于实践或其他要求不高的用例。

电池寿命

电池的额定正常充电/充电周期约为 1200 次。FRC 匹配所需的高电流将使用寿命缩短到约 400 个周期。这些循环旨在实现相对较低的放电，从大约 13.5 伏降至 12 伏或 12.5 伏。深度循环使用电池（将其完全耗尽）会损坏电池。

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.

电池应远离极热和极冷的地方。这通常意味着将电池存放在恒温区域：教室壁橱通常很好，停车场运输容器的危险性更大。

36.7.5 电池最佳使用方法

- 比赛时只能使用充电的电池。如果您的电池电量耗尽，请向经验丰富的团队寻求帮助！没有人希望看到由于电池损坏或未充电而使机器人在场上死亡。
- 强烈建议团队在压接过程中使用具有适当额定值的工具和严格的质量控制做法（向当地资深团队或商业电工寻求帮助），或使用供应商制造的电池引线。
- 比赛结束后，请等待电池冷却后再充电：机壳不应摸起来很热，通常十五分钟就可以了。
- 团队应考虑每年购买几节新电池以帮助保持其电池新鲜。淘汰赛可能需要很多电池，并且可能没有足够的时间来充电。

- 多库电池充电器可让您一次为多个电池充电。许多团队会为其电池和充电器建立一个机器人手推车，以便于运输和存放。
- 最好永久性地至少标识每个电池：团队编号，年份和唯一标识符。
- 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.

- 使用电池标志（电池连接器中放有一块塑料）是指示电池已充电的常用方法。电池标志也可以轻松地被 3D 打印出。
- 可以购买 SB3 触点的手柄或进行 3D 打印，以帮助避免在连接或断开电池连接时拉扯引线。请勿使用这些手柄搬运电池。

- 一些团队用旧的安全带或其他扁平尼龙缝制电池携带带，以固定在电池周围来防止导线携带。

- 扎带边缘夹可与 90 度压接凸耳一起使用，以减轻电池引线的应力。

37

硬件教程

备注： 该部分的页面包含只能从文档的 web 版本中查看的媒体。

37.1 机器人电机的应用

设计机器人时，每个团队往往在机器人电动驱动系统的选择上面临抉择。选择不正确的电动机会降低性能，有时甚至会因过多的电流消耗而使电机失效。在本系列视频中，WPI 教授 Ken Stafford 教授了电机的工作原理，如何设计以最佳性能运行的系统以及机器人系统的示例设计。

37.2 传感与传感器

没有传感器和传感的机器人会是真正的无线电控制载具。传感器允许机器人了解机械系统的内部操作，以及与机器人周围环境互动的能力。在这些视频中，WPI 教授 Craig Putnam 描述了一些传感器的类别，它们是如何使用的，并提供了针对于您的应用场景的传感器最优选择的指导。

37.3 气动组件

气动组件常常未被充分应用在机器人上。事实上，与使用电机相比，气动具有许多优点。在此视频中，Ken Stafford 教授描述了气动系统的特性，在机器人上的应用，以及在应用时如何计算合适的气动系统的尺寸。

37.4 动力传递

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 传感器

38.1 传感器概述-硬件

备注： This section covers sensor hardware, not the use of sensors in code. For a software sensor guide, see *Sensor Overview - Software*.

为了有效，机器人通常必须能够收集有关其周围环境的信息，这一点至关重要。向机器人提供有关其环境状态的反馈的设备称为“传感器”。FRC |reg| 有大量可用的传感器。团队，用于测量从现场定位到机器人定位再到电机/机械定位的所有内容。利用传感器是在现场取得成功的绝对关键技能。尽管大多数 FRC 游戏确实具有可以由“盲人”机器人完成的任务，但是最好的机器人在很大程度上依靠传感器来尽可能快，可靠地完成游戏任务。

此外，传感器对于机器人的安全性可能非常重要-如果使用不当，许多机器人机构都有可能自行损坏。传感器提供了防止这种情况的保护措施，例如，如果机械装置遇到硬停止，则允许机器人禁用电动机。

38.1.1 传感器的种类

FRC 中使用的传感器通常可以通过两种不同的方式进行分类：按功能和按通信协议。前一种分类与机器人设计有关。后者用于接线和编程。

传感器功能

传感器可以提供有关机器人状态的各种不同方面的反馈。FRC 常用的传感器功能包括：

- 接近开关 <proximity-switches>
  - 机械接近开关（“限位开关”）
  - 磁性接近开关
  - 电感式接近开关
  - 光电接近开关
- 距离传感器
  - 超声波传感器 <ultrasonics-hardware>‘

  - 三角测距仪 <triangulating-rangefinders>‘
  - LIDAR <lidar>‘
- 轴旋转传感器
  - 编码器 <encoders-hardware>‘
  - 电位器 <analog-potentiometers-hardware>‘
- 加速度计 <accelerometers-hardware>‘
- 陀螺仪 <gyros-hardware>‘

通讯协议的传感器

为了使传感器有用，它必须能够与 roboRIO “对话”。传感器可以通过几种主要方法将其读数传达给 roboRIO：

- 模拟输入 <analog-inputs-hardware>‘
- 数字输入 <digital-inputs-hardware>‘
- Serial bus <serial-buses>‘

通常，对通过模拟和数字输入进行通信的传感器的支持非常简单，而通过串行总线进行的通信则更为复杂。

38.2 模拟输入 - 硬件

备注： This section covers analog input hardware. For a software guide to analog inputs, see *Analog Inputs - Software*.

模拟信号 analog signal 是一种信号，其值可以位于连续间隔内的任何位置。这与数字信号形成鲜明对比，数字信号 <digital-inputs-hardware> 只能采用几个离散值之一。roboRIO 的模拟输入端口允许测量 0V 至 5V 范围内的模拟信号。

实际上，无法使用数字设备比如计算机（roboRIO）来测量模拟信号是否为 “真”。因此，模拟输入实际上被当作 12 位数字信号来测量。这么高的测量分辨率已经足够了[1]。

模拟输入通常（但不总是）用于其测量值在一定范围内连续变化的传感器，例如超声波测距仪 <ultrasonics-hardware> 和电位计 <analog-potentiometers-hardware>，因为它们可以通过输出与测量值成比例的电压进行通信。

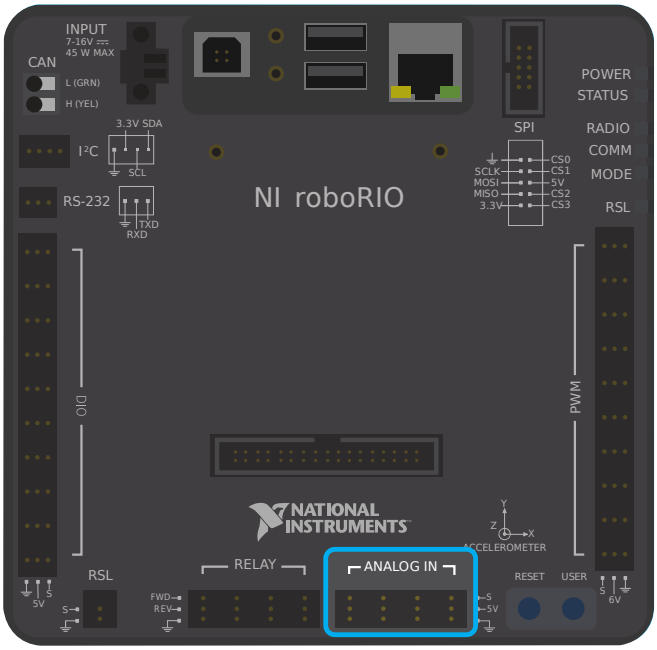
[1] 一个 12 位分辨率的信号可以产生 2^{12} 即 4096 个不同的值。对于 5V 范围内的信号，这是大约 1.2 mV 或 0.0012V 的有效分辨率。实际测量精度为正负 50mV，因此这些离散的信号值并不会限制测量精度。

38.2.1 连接到 roboRIO 的模拟输入端口

备注： 通过“MXP”扩展端口还可以使用另外四个模拟输入。要使用它们，需要和 MXP 连接的分线板上的端口相连。

警告： 在对传感器进行接线之前，请务必先阅读正在使用的传感器的技术文档，以确保将正确的导线连接到每个引脚。否则可能会损坏传感器或 roboRIO。

警告： **切勿 ** 将传感器的电源引脚直接连接到 roboRIO 任意端口上的接地引脚！这将触发 roboRIO 上的保护功能，并可能导致意外行为。

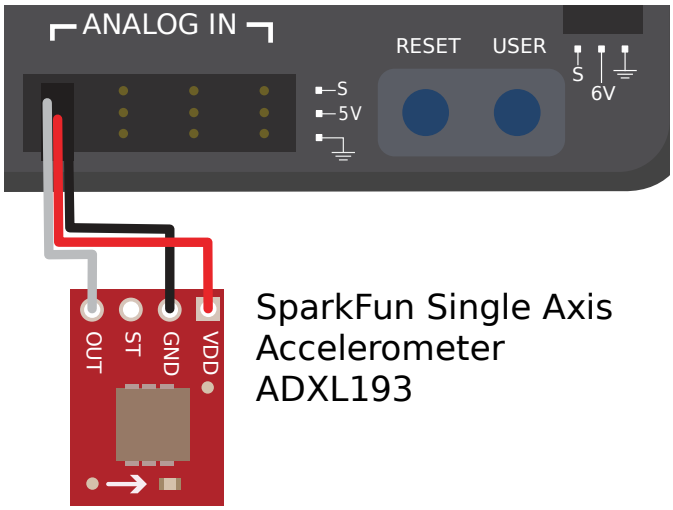


如上图所示，roboRIO 内置 4 路模拟输入端口（0-3 号）。每一个端口有三根引脚——信号引脚（S），电源引脚（V）和地线引脚（⏚）。电源引脚和地线引脚 [2]_ 用于给传感器外设供电，电压是 5V。信号引脚则是实际测量信号的引脚。

将传感器连接到单个模拟输入端口

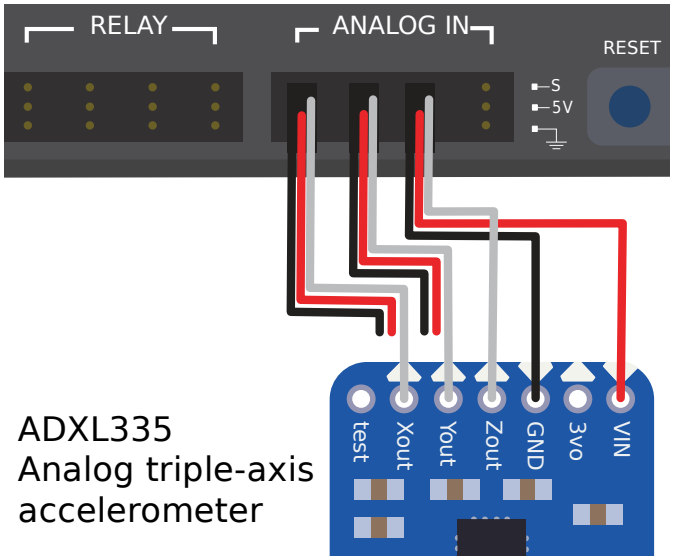
备注： 有的传感器（如电位器 <analog-potentiometers-hardware>) 可能会共享电源和地线引脚。

连接到模拟输入端口的大多数传感器将具有三根线——信号，电源和地线——精确地对应于模拟输入端口的三个引脚。它们应进行相应连接。



将传感器连接到多个模拟输入端口

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 脚注

38.3 模拟电位器 - 硬件

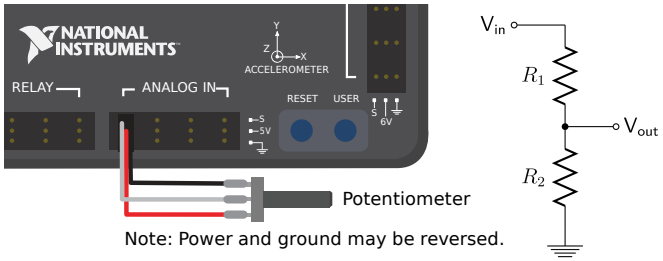
备注: This section covers analog potentiometer hardware. For a software guide to analog potentiometers, see *Analog Potentiometers - Software*.

警告: 电位计通常具有受机械限制的行程范围。用户应注意其机械装置不要使电位计旋转超过最大行程，因为这会损坏或破坏电位计。

除*quadrature encoders*，另一种常用于测量 FRC|reg| 机器人旋转的方式是使用模拟电位器。一个模拟电位器实际上是一个可调电阻 - 当电位器的轴旋转时，电阻的阻值也随之变化（通常是线性的）。应用 ‘voltage divider <https://en.wikipedia.org/wiki/Voltage_divider>‘__ 上可以通过测量电阻两端的电压来测出电阻的阻值，从而可以计算电位器旋转轴的旋转角度。

38.3.1 连接一个模拟电位器

就如其名，模拟电位器需要连接到 roboRIO 的模拟输入口 <analog-inputs-hardware>。但是，若要了解如何精确连接电位计，了解其内部电路非常重要。



左上方的图片显示了一个典型的电位计。就像 RIO 的模拟输入口一样，它共有三个引脚。中间引脚是信号引脚，而外部两引脚可以是电源或地线任意一个。

如前所述，电位器是分压器，如右图所示。随着电位计轴的旋转，电阻 R1 和 R2 改变；但是，它们的总和保持不变 [1]_。因此，整个电位计上的电压保持恒定（对于 roboRIO，这将是 5 伏），但是信号轴与电源或地线引脚之间的电压随轴旋转而线性变化。

由于电路是对称的，因此也是可逆的——允许用户选择在行程的哪一端测得的电压为零，在哪一端为 5 伏。若要反转传感器的方向，只需将其供电引脚反向接线即可！在将电线焊接到触点之前，请务必用万用表检查电位计的方向性，以确保其方向正确。

38.3.2 脚注

38.4 数字输入 - 硬件

备注： This section covers digital input hardware. For a software guide to digital inputs, see *Digital Inputs - Software*.

数字输入 是一个可能处于几种离散状态之一的信号。在大多数情况下，信号就是电线中的电压，数字信号只有两种状态——高或低（分别表示为 1 和 0，或 true 和 false）。

roboRIO 的内置数字输入输出端口（DIO）以 5V 供电，因此“高”对应于 5V 信号，“低”对应于 0V 信号[1][2]。

38.4.1 连接到 roboRIO 的数字输入端口

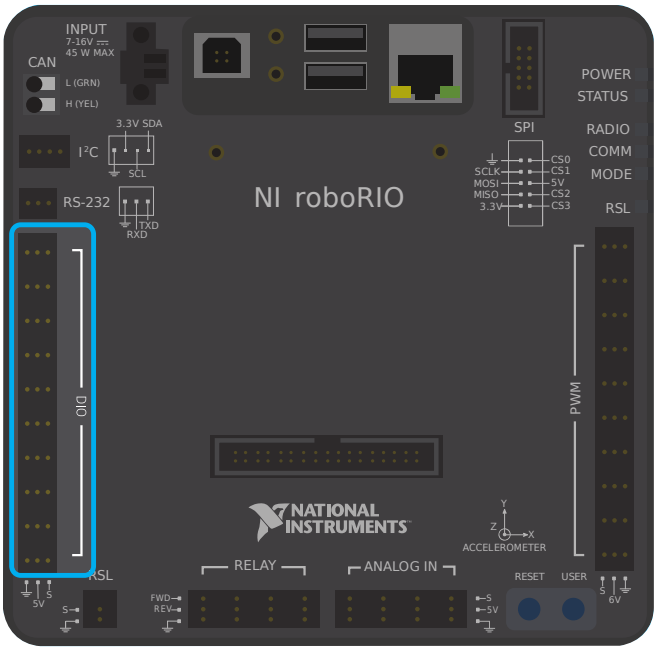
备注： 通过“MXP”扩展端口还可以使用额外的 DIO 端口。要使用它们，需要和 MXP 连接的分线板上的端口相连。

警告： 在对传感器进行接线之前，请务必 * 先 * 阅读正在使用的传感器的技术文档，以确保将正确的导线连接到每个引脚。否则可能会损坏传感器或 roboRIO。

警告： **切勿 ** 将传感器的电源引脚直接连接到 roboRIO 任意端口上的接地引脚！这将触发 roboRIO 上的保护功能，并可能导致意外行为。

[1] More precisely, the signal reads “high” when it rises above 2.0V, and reads “low” when it falls back below 0.8V - behavior between these two thresholds is not guaranteed to be consistent.

[2] The roboRIO also offers 3.3V logic via the “MXP” expansion port; however, the use of this is far less common than the 5V.



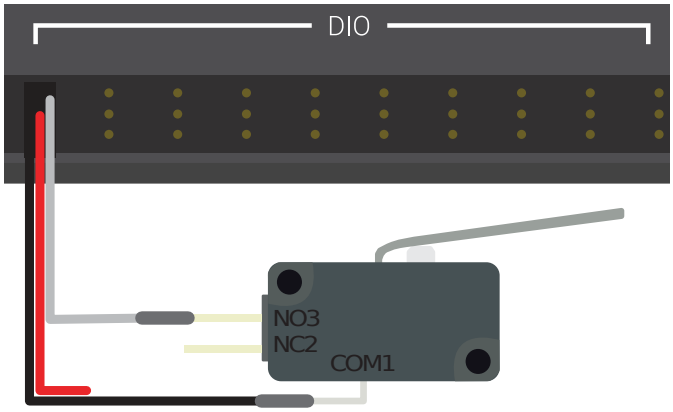
如上图所示，roboRIO 内置 10 路 DIO 端口（0-9 号）。每一个端口有三根引脚——信号引脚（S），电源引脚（V）和地线引脚（□）。电源引脚和地线引脚 [3]_ 用于给传感器外设供电，电压是 5V。电源引脚对应高电平（5V），地线引脚对应低电平（0V）。信号引脚则是实际测量信号的引脚（当用于输出时，该引脚对外发送信号）。

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.

将简易开关连接到 DIO 端口

连接到 DIO 端口的最简单的设备就是开关（例如一个限位开关）。当将开关正确连接到 DIO 端口时，电路断开时该端口将读值为"高"，而电路闭合时该端口将显示为"低"。

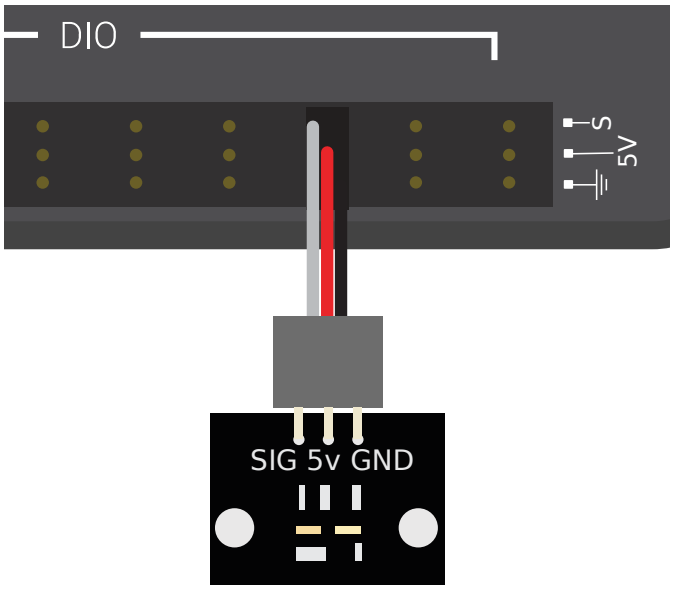
简单的开关不需要供电，因此只有两根线。开关应连接在 DIO 端口的 * 信号 * 和 * 接地 * 引脚之间。当开关电路断开时，信号引脚将浮置，并且上拉电阻将确保其读数为"高"。当开关电路闭合时，它将直接连接到地线，因此显示为"低"。



Limit Switch or Micro Switch

将一个需要供电的传感器连接至 DIO 端口

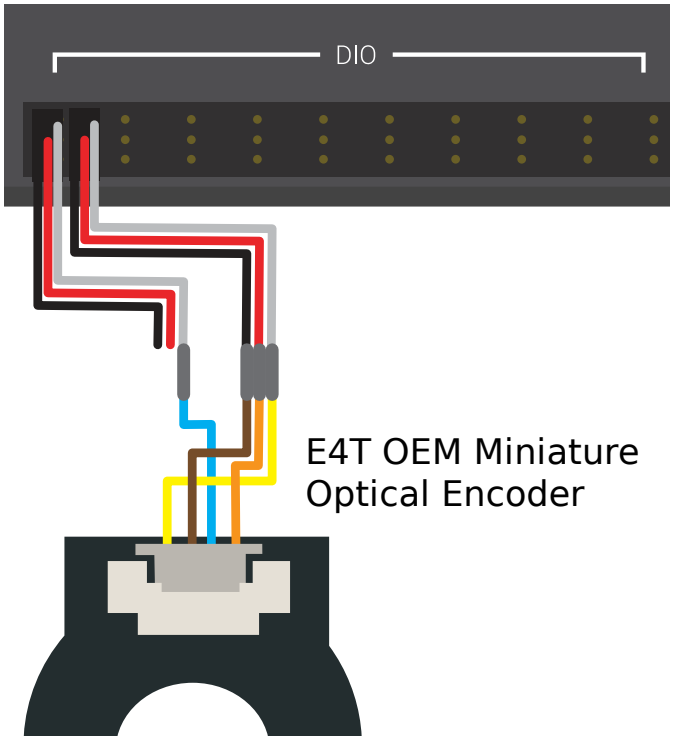
许多数字传感器（例如大多数非接触式接近开关）都需要电源才能正常工作。有源传感器通常具有三根电线-信号线，电源线和地线。这些应连接到 DIO 端口的相应引脚。



WCP Hall Effect Sensor

将传感器连接到多个 DIO 端口

一些传感器（如正交编码器 <encoders-hardware>）可能需要连接到多个 DIO 端口才能工作。通常，这些传感器将仅需要一路供电引脚和多路信号引脚。



38.4.2 脚注

38.5 接近开关-硬件

备注： This section covers proximity switch hardware. For a guide to using proximity switches in software, see *Digital Inputs - Software*.

机器人上最常见的传感任务之一是检测物体（无论是机制，游戏块还是场地元素）何时在机器人上已知点的一定距离内。这种类型的感测是通过“接近开关”完成的。

38.5.1 接近开关操作

接近开关是开关-它们在“断开”状态（电路中没有连接）和“闭合”状态（其中有 *）之间操作电路。因此，接近开关会产生数字信号，因此，它们几乎总是连接到 roboRIO 的:doc:*digital input <digital-inputs-hardware>* 端口。

接近开关可以是“常开”，其中激活开关可以使电路闭合；也可以是“常开”，其中激活开关可以使电路断开。某些开关 * 同时提供 * NO 和连接到同一开关的 NC 电路。实际上，在常开和常闭开关之间的有效区别是在到开关的接线失败的情况下系统的行为，因为接线故障几乎总是会导致开路。NC 开关通常是“更安全的”，因为布线故障会导致系统像按下开关一样工作-因为经常使用开关来防止机械装置自身损坏，因此在这种情况下，可以减轻机械装置损坏的机会接线故障。

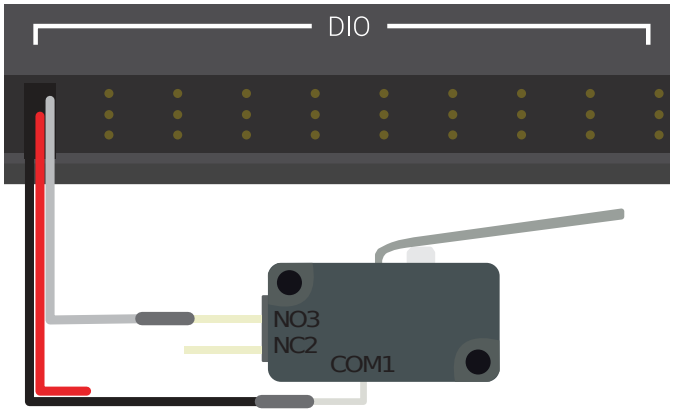
roboRIO 上的数字输入具有上拉电阻，当开关断开时，上拉电阻会使输入为高（1 值），但是当开关闭合时，该值将变为 0，因为现在输入已接地。

38.5.2 接近开关的种类

There are several types of proximity switches that are commonly used in FRC®:

- 机械接近开关（“限位开关”）_
- 磁性接近开关
- 感应接近开关
- 光电接近开关
- 飞行时间接近开关

机械接近开关（“限位开关”）



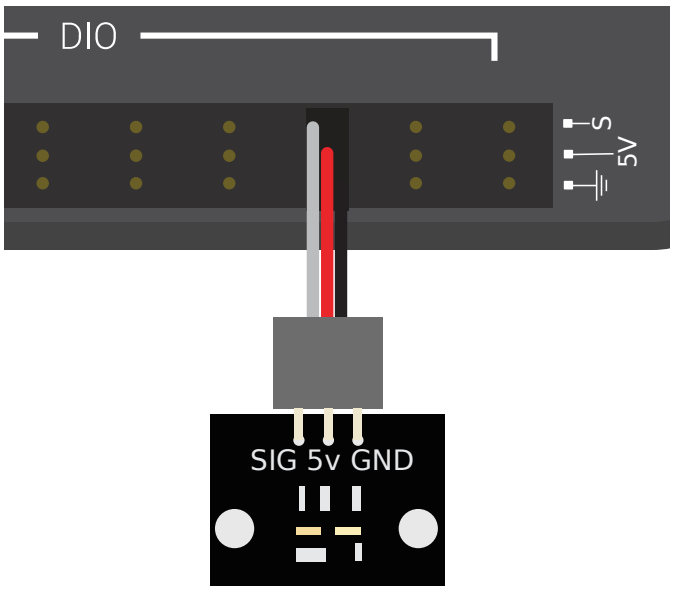
Limit Switch or Micro Switch

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.

限位开关的大小，开关臂的几何形状以及激活开关所需的“投掷”量各不相同。尽管限位开关非常便宜，但其机械致动有时不如无触点替代品可靠。但是，它们也具有极强的通用性，因为它们可以由能够移动开关臂的任何物理物体触发。

See this *article* for writing the software for Limit Switches.

磁性接近开关



WCP Hall Effect Sensor

当磁铁进入传感器的特定范围内时，将激活磁性接近开关。因此，它们是“非接触式”开关-它们不需要与被感测的物体接触。

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.

磁性接近开关可以是“单极”，“双极”或“全极”。单极开关根据磁铁的给定磁极的存在（根据开关的不同，是南极还是北极）来激活和停用。双极开关从一个极的附近激活，而从相对极的附近去激活。全极性开关将在任一极存在时激活，并在没有磁体的情况下禁用。

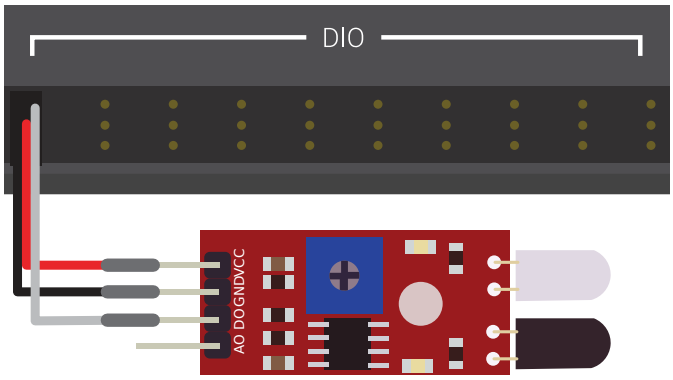
尽管磁性接近开关通常比机械接近开关更可靠，但是它们要求用户将磁体安装在要感测的物体上-因此，它们大多用于感测机构的位置。

电感式接近开关

当任何类型的导体进入传感器的特定范围内时，感应式接近开关就会激活。像磁性接近开关一样，它们是“无触点”开关。

电感式接近开关用于许多与磁性接近开关相同的目的。根据应用的性质，它们的更一般的性质（在任何导体而不是仅在磁体的存在下激活）可以是帮助或阻碍。

光电接近开关



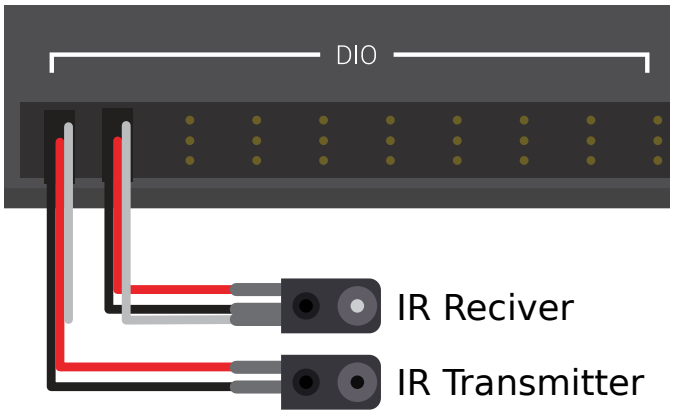
IR Digital Obstacle Avoidance Sensc

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.

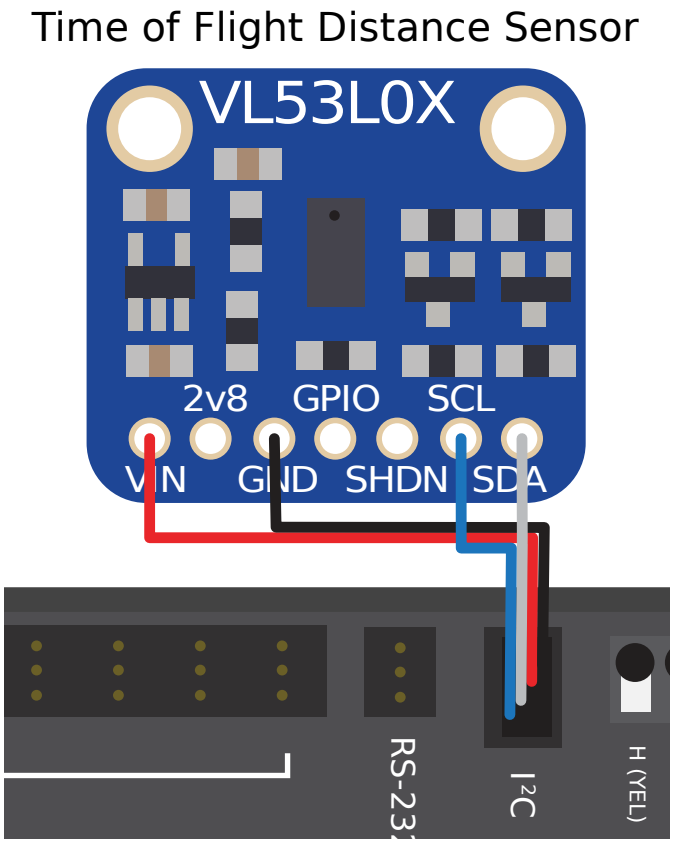
Since photoelectric proximity switches rely on measuring the amount of reflected light, they are often inconsistent in their triggering range between different materials - accordingly, most photoelectric sensors have an adjustable activation point (typically controlled by turning a

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.



飞行时间接近开关



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 编码器 - 硬件

备注： This section covers encoder hardware. For a software guide to encoders, see *Encoders - 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 编码器的种类

There are three main ways encoders connect physically that are typically used in FRC:

- 轴编码器
- 轴上编码器
- 磁编码器

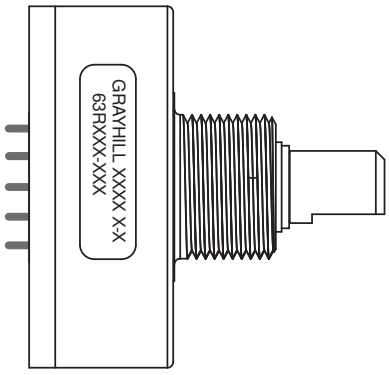
这些编码器的安装方式有所不同。除了这些类型的编码器，许多队伍还将正交编码器集成到机械设计中。

There are also three main ways the encoder data is communicated that are typically used in FRC:

- *Quadrature encoders*
- *Duty Cycle encoders*
- *Analog encoders*

备注： Some encoders may support more then one communication method

Shafted Encoders



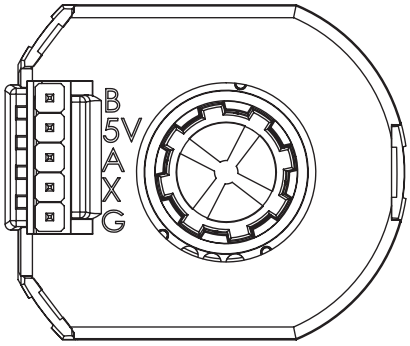
Shafted encoders have a sealed body with a shaft protruding out of it that must be coupled rotationally to a mechanism. This is often done with a helical beam coupling, or, more cheaply, with a length of flexible tubing (such as surgical tubing or pneumatic tubing), fastened with cable ties and/or adhesive at either end. Many commercial off-the-shelf FRC gearboxes have purpose-built mounting points for shafted encoders.

Examples of shafted encoders:

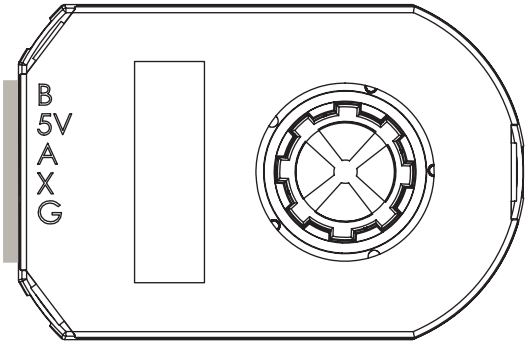
- Grayhill 63r
- US Digital MA3

On-shaft Encoders

AM10 Series Modular Incremental Encoder



AMT103



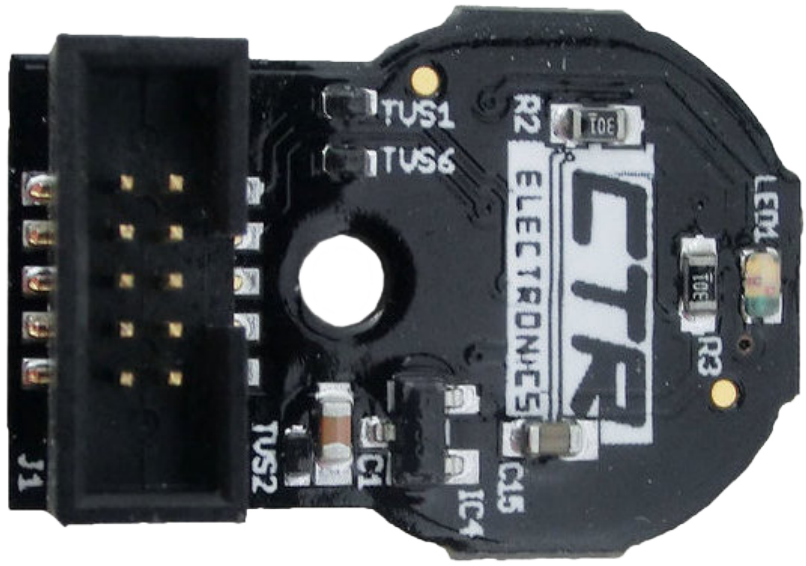
AMT102

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



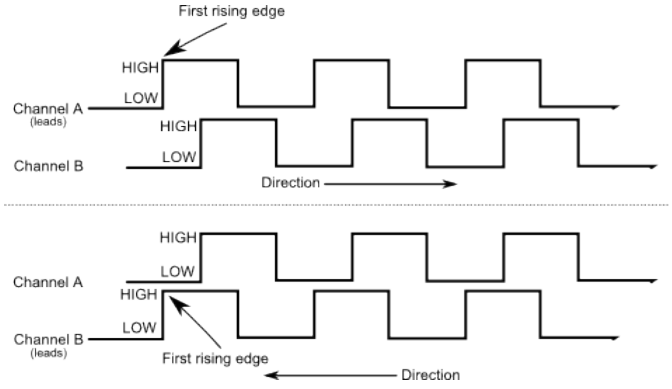
Magnetic encoders require no mechanical coupling to the shaft at all; rather, they track the orientation of a magnet fixed to the shaft. While the no-contact nature of magnetic encoders can be handy, they often require precise construction in order to ensure that the magnet is positioned correctly with respect to the encoder.

Examples of magnetic encoders:

- CTRE Mag Encoder
- Thrifty Absolute Magnetic Encoder
- Team 221 Lamprey2

Quadrature Encoders

术语“正交”是指测量/编码运动的方法。正交编码器会产生两个相位相差为 90 度的方波脉冲，如下图所示：



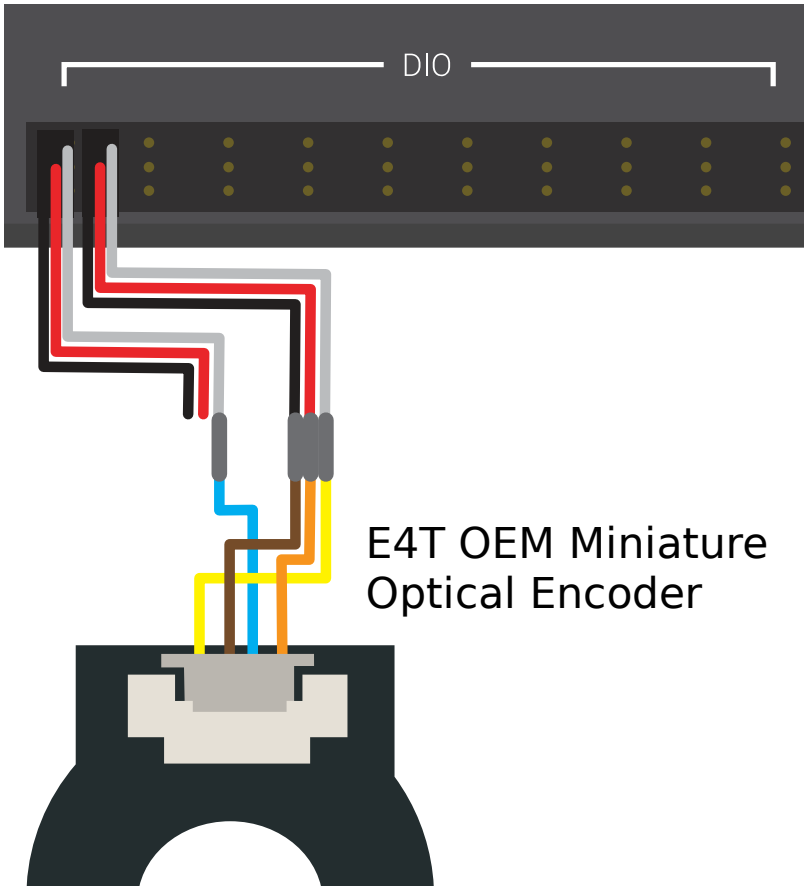
因此，在这两个信号通道之间，每个周期有四个“边”（因此为“quad”）。通过判断两个不同相位的信号脉冲，哪个脉冲在另一个脉冲的前面，就可以明确确定运动方向，

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

警告： 首字母缩写词 “CPR” 和 “PPR” 被不同的来源 * 同时 * 使用，以表示每转的两个边沿 * 和 * 每转的循环数，因此仅首字母缩写还不足以说明两者中的哪一个是何时给定的价值。如有疑问，请查阅您特定编码器的技术手册。

当编码器使用数字脉冲测量旋转时，测量精度受每给定旋转运动量的脉冲数限制。这被称为编码器的 “分辨率”，并且通常以两种不同方式之一进行测量：每转的边沿或每转的圈数。

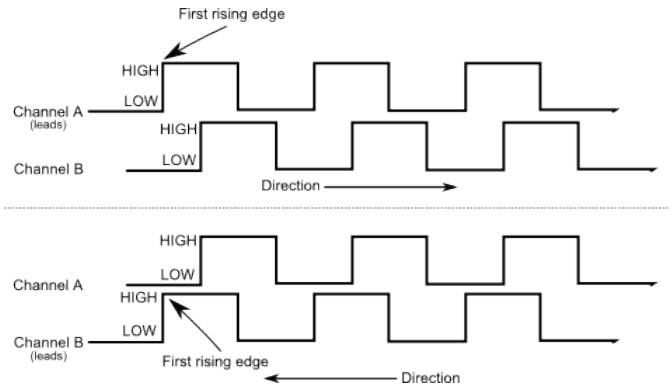
*每转边沿 * 是指编码器轴每转两个通道从高到低或从低到高的过渡总数。一个完整的周期包含 * 四个 * 边。

*每转周期 * 是指编码器轴每转两个通道的 * 完整周期 * 的总数。一个完整的周期是 * 一 * 转。

因此，以每转边缘表示的分辨率的值是以每转周期表示的相同分辨率的值的四倍。

通常，编码器每转边的分辨率应比定位中可接受的最小误差稍好。因此，如果您想知道正负一度的机制，则应使用分辨率稍高于每转 360 个边缘的编码器。

Duty Cycle Encoders

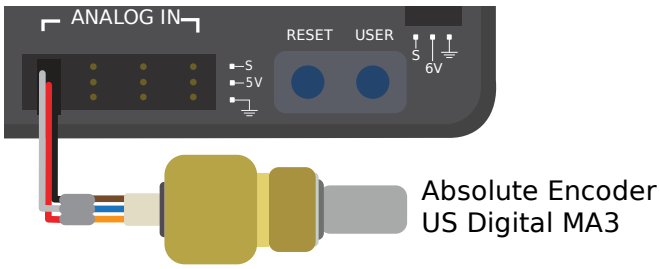


Duty cycle encoders connect to a single digital input on the roboRIO. They output a pulse where the length of a pulse is proportional to the absolute position of the encoder.

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 陀螺仪-硬件

备注： This section covers gyro hardware. For a software guide to gyros, see *Gyroscopes - Software*.

陀螺仪（简称“陀螺仪”）是测量旋转速率的设备。这些对于稳定机器人驱动或通过总和（累加）速率测量以获得总角位移的测量来测量航向或倾斜特别有用。

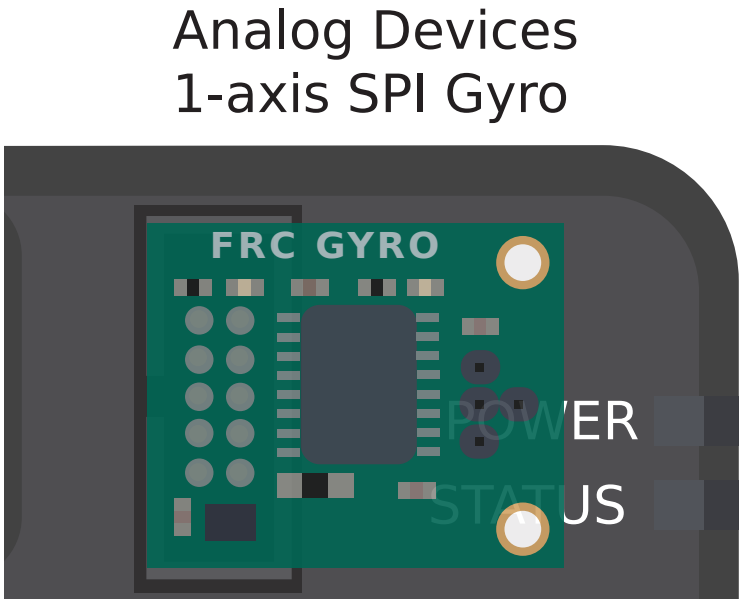
Several popular FRC® devices known as *IMUs* (Inertial Measurement Units) combine 3-axis gyros, accelerometers and other position sensors into one device. Some popular examples are:

- Analog Devices ADIS16448 和 ADIS 16470 IMUs
- CTRE Pigeon IMU
- Kauai Labs NavX

38.7.1 陀螺仪的种类

FRC 中通常使用两种类型的陀螺仪：单轴陀螺仪，三轴陀螺仪和 IMU，这通常包括 3 轴陀螺仪。

单轴陀螺仪



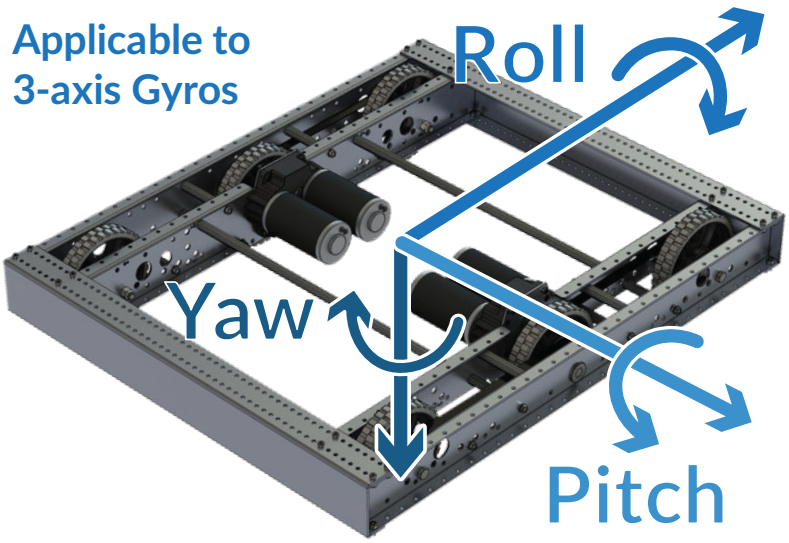
As per their name, single-axis gyros measure rotation rate around a single axis. This axis is generally specified on the physical device, and mounting the device in the proper orientation so that the desired axis is measured is highly important. Some single-axis gyros can output an analog voltage corresponding 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.

The Analog Devices ADXRS450 FRC Gyro Board that has been in FIRST Choice in recent years is a commonly used single axis gyro.

三轴陀螺仪

三轴陀螺仪测量围绕所有三个空间轴（通常标记为 x，y 和 z）的旋转速率。围绕这些轴的运动称为俯仰，偏航和横滚。

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.



备注： 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.

外围三轴陀螺仪可以简单地输出三个模拟电压（并因此连接到*analog input ports*，或（更常见的）它们可以与 roboRIO 的:doc:‘serial buses <serial-buses>‘中的一种通信。

38.8 超声波测距仪-硬件

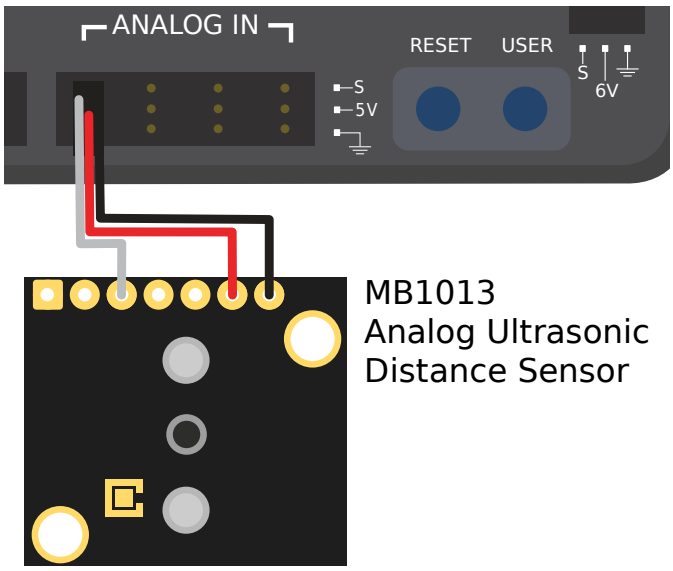
备注： This section covers ultrasonic sensor hardware. For a software guide to ultrasonics, see *Ultrasonics - Software*.

超声波测距仪是 FRC | reg |中最常用的一些测距仪。它们便宜，易于使用且相当可靠。超声波测距仪的工作原理是发出高频声音脉冲，然后测量从目标弹回后回声到达传感器所需的时间。根据测得的时间和空气中的声速，可以计算到目标的距离。

38.8.1 超声波测距仪的种类

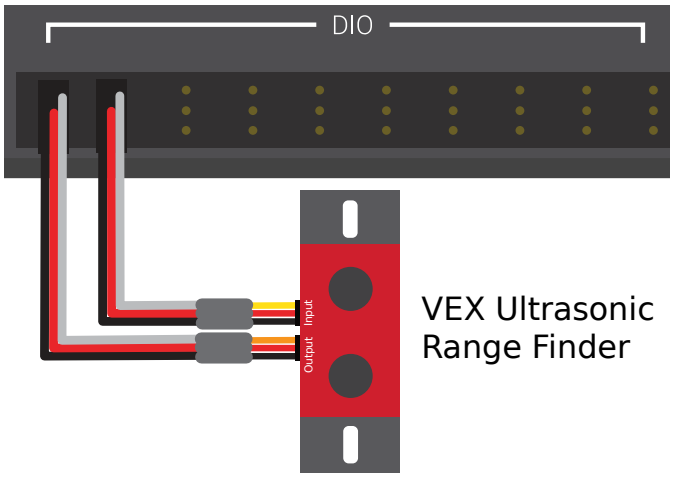
尽管所有超声波测距仪均按照上述“ping 响应”原理运行，但它们与 roboRIO 通讯的方式可能会有所不同。

模拟超声波测距仪



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*.

响应式超声波测距仪



如上所述，所有超声波都是具有功能性的 ping 响应设备，一种“ping 响应”超声波测距仪可配置为连接到*both a digital input and a digital output*。数字输出用于发送 ping，而输入则用于读取响应。

连续超声波

一些更复杂的超声波传感器可能会通过*serial buses*, 与 RIO 通信，比如 RS-232。

38.8.2 注意事项

超声波传感器通常很容易使用，但是有一些警告。超声波通过测量脉冲与其回波之间的时间来工作，因此它们通常仅测量距其范围内“最近”目标的距离。因此，为工作选择合适的传感器非常重要。超声波传感器的文档通常会包含“光束图案”的图片，该图片显示超声波将在其中检测到目标的“窗口”的形状-选择传感器时请特别注意。

超声波传感器还容易受到其他超声波传感器的干扰。为了最大程度地减少这种情况，roboRIO 可以“循环”方式运行 ping 响应超声波-但是，在竞争中，无法确定会不会受到来自其他机器人上安装的传感器的干扰。

最后，超声波可能无法检测吸收声波或以八仙过海各显神通地重定向声波的物体。因此它最适用于检测坚硬的扁平物体。

38.9 加速度计 - 硬件

加速度传感器是常用于测量机器人运动的加速度。

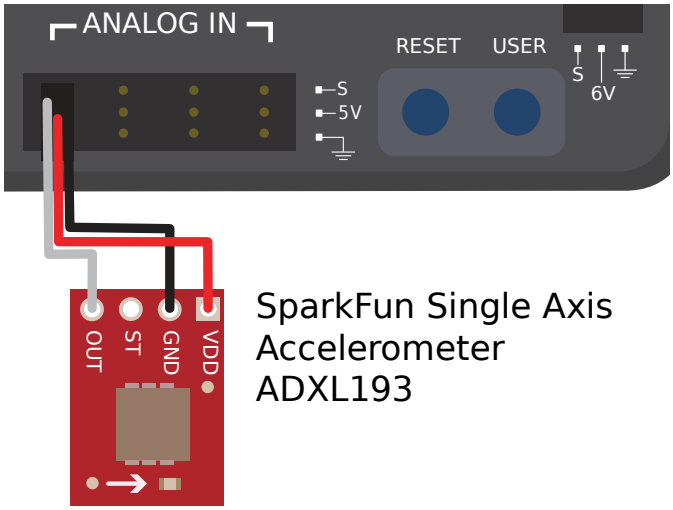
原则上，精确的测量加速度读数可以通过二重计分得到位置信息（这与二重积分从陀螺仪传感器测量的角速度变化率获取角度朝向信息十分类似）- 然而实际上可用于 FRC|reg| 比赛的加速度传感器的测量精度还是远远不足以估算位置的。尽管如此，加速度计依然在 FRC 比赛中有着十分大的用处。

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 加速度计的种类

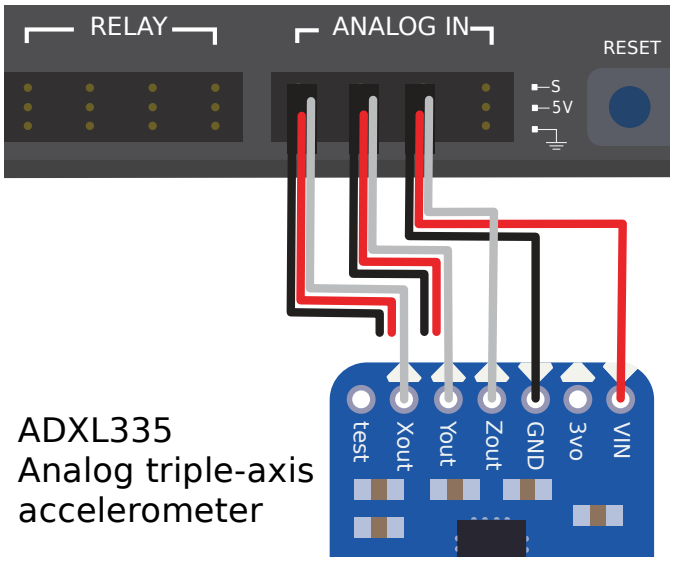
FRC 中有以下三种常见的加速度计：单轴加速计，多轴加速度计和 IMU 惯性测量单元。

单轴加速度计



根据他们的名字，单轴加速计测量的沿一个轴向上的加速度读数。该轴通常是由机械结构的设计决定，并且以正确的方向安装该加速度，以测量加速度读数在该轴向上的读数十分重要。单轴加速度计通常输出与加速度读数正相关的模拟电压信号，因此应当连接到 roboRIO 的模拟输入端口 <analog-inputs-hardware>。

多轴加速度计

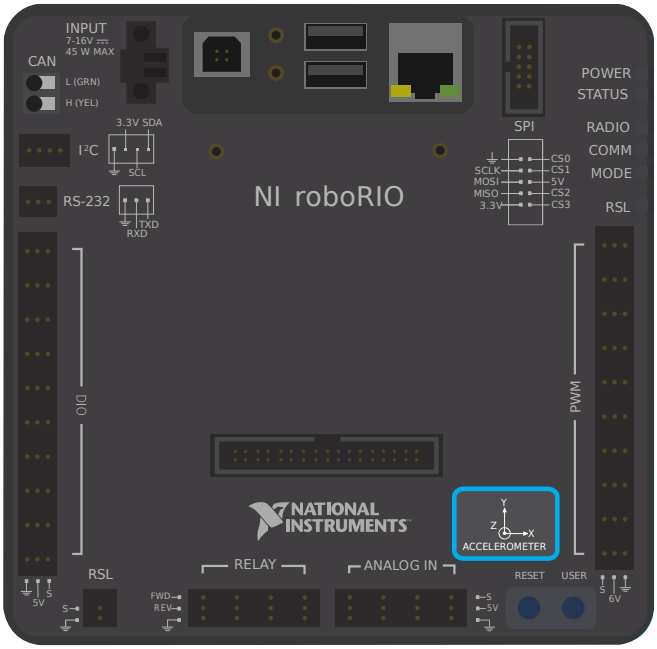


多轴加速度计可测量三维空间中多个轴向的加速度。roboRIO 的内置加速度计是一个三轴加速度计。

外围多轴加速度计可以简单地输出多个模拟电压（从而连接到模拟输入端口 <docs/hardware/sensors/analog-inputs-hardware:Connecting a sensor to multiple ana-

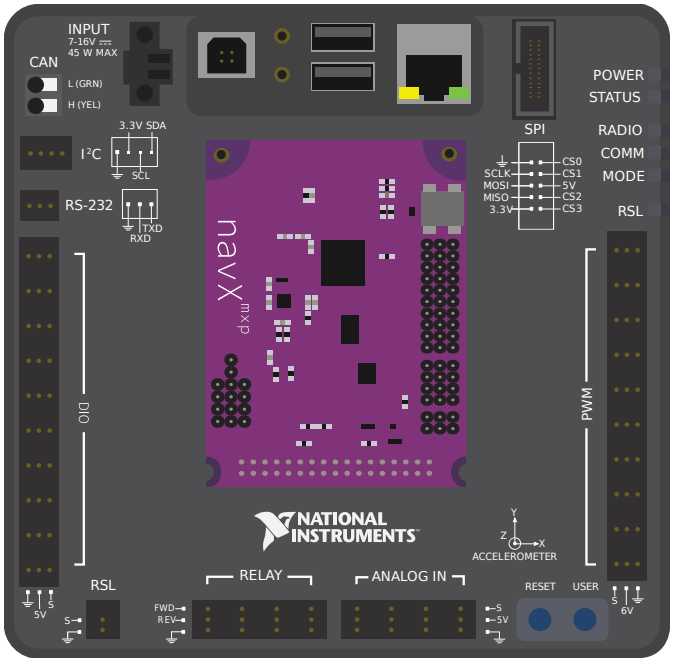
log input ports>，或者（更常见的是）它们可以与 roboRIO 的串行总线 <serial-buses> 之一通信。

roboRIO 板载加速度计



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.

IMU（惯性测量单元）



几种 FRC 提供的传感器（称为 IMU 或者惯性测量单元）整合了加速度计和陀螺仪。这些传感器包括:

- Analog Devices ADIS16448 和 ADIS 16470 IMUs
- CTRE Pigeon IMU
- Kauai Labs NavX

38.10 LIDAR - 硬件

LIDAR（light detection and ranging 光检测和测距）传感器是各种测距仪，在 FRC|reg| 中的使用越来越多。

LIDAR 传感器的工作原理与:doc:‘ultrasonics <ultrasonics-hardware>‘类似，但使用光代替声音。脉冲激光，传感器测量直到脉冲回弹的时间。

38.10.1 LIDAR 的类型

当前 FRC 中通常使用两种类型的 LIDAR 传感器：1 维 LIDAR 和 2 维 LIDAR。

1 维 LIDAR

一维（1D）LIDAR 传感器的工作原理与超声传感器非常相似-它沿其前面的一条线或多或少地测量到最近物体的距离。一维 LIDAR 传感器通常比超声波更可靠，因为它们的“光束轮廓”更窄并且不易受到干扰。上图是‘Garmin LIDAR-Lite Optical Distance Sensor <https://www.andymark.com/products/lidar-lite-3>‘__。

1 维 LIDAR 传感器通常输出与所测距离成比例的模拟电压，因此连接到 roboRIO 的:doc:*analog input <analog-inputs-hardware>* 端口或者:doc:‘roboRIO’ s serial buses <serial-buses>‘的其中一个。

2 维 LIDAR

2 维（2D）LIDAR 传感器可测量平面上所有方向的距离。通常，这可以通过将 1 维 LIDAR 传感器简单地放置在以恒定速度旋转的转盘上（或多或少）来实现。

由于 2D LIDAR 传感器本质上需要将大量数据发送回 roboRIO，因此它们几乎总是连接到 roboRIO 的其中一个*serial buses*。

38.10.2 注意事项

LIDAR 传感器确实有一些常见的缺点：

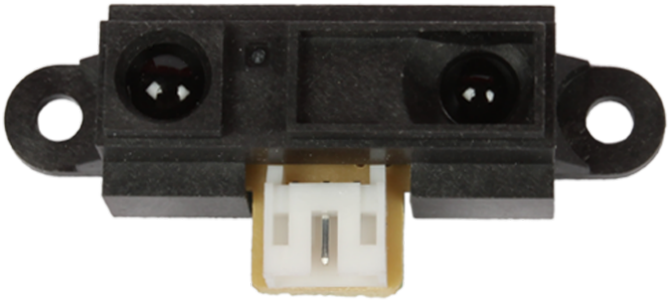
像超声波一样，激光雷达依靠发射的脉冲反射回传感器。因此，激光雷达关键取决于材料在激光波长中的反射率。FRC 场墙由聚碳酸酯制成，在红外波长下它往往是透明的（这在 FRC 的使用中通常是合法的）。因此，LIDAR 往往难以检测到场障。

2D LIDAR 传感器（在 FRC 使用的合法价格范围内）往往噪声很大，处理其测量数据（称为“点云”）可能涉及许多复杂软件。此外，专门为 FRC 制造的 2D LIDAR 传感器很少，因此软件支持趋于匮乏。

由于 2D LIDAR 传感器依靠转盘工作，因此其更新速度受到转盘旋转速度的限制。对于 FRC 合法价格范围内的传感器，这通常意味着它们不会特别快地更新其值，这在机器人（或目标）移动时可能会受到限制。

此外，由于 2 维 LIDAR 传感器的 * 角度 * 分辨率受到限制，所以当目标距离较远时，点云的 * 空间 * 分辨率会变差。

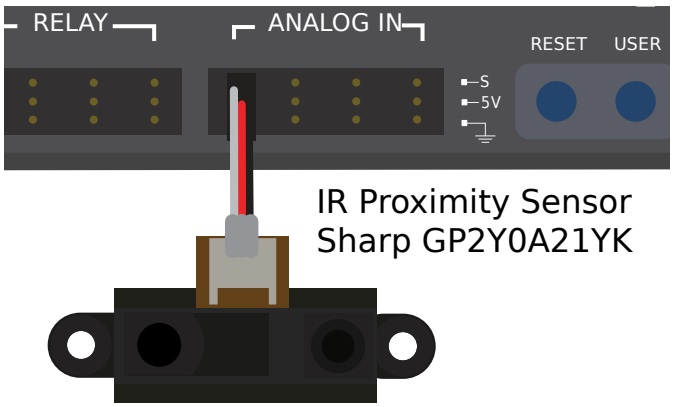
38.11 三角测距仪



三角测距仪（通常称为 “IR 测距仪”，因为它们通常在红外波段内起作用）是 FRC | reg |中使用的另一种常见测距仪。上面显示的传感器是 ‘common Sharp-brand sensor <https://www.sparkfun.com/products/242>‘__

与:doc:‘LIDAR <lidar>‘不同，三角测距仪不测量脉冲发射与反射接收之间的时间。相反，大多数红外测距仪的工作原理是以微小角度发射恒定光束，并测量反射光束的位置。反射光束与发射器的接触点越近，物体越靠近传感器。

38.11.1 使用红外测距仪



红外测距仪通常输出与到目标的距离成正比的模拟电压，并连接到 RIO 上的:doc:*analog input <analog-inputs-hardware>* 端口。

38.11.2 注意事项

红外测距仪具有与 1 维激光雷达传感器类似的缺点-它们对目标在发射激光波长中的反射率非常敏感。

此外，尽管在近距离测量时，红外测距仪往往比 LIDAR 传感器提供更好的分辨率，但它们通常对目标的方向差异更敏感，尤其是如果目标是高反射性的（例如镜子）。

38.12 串行总线

除了:doc:*digital <digital-inputs-hardware>* 和:doc:‘analog <analog-inputs-hardware>‘作为输入，roboRIO 还提供了几种与外围设备进行串行通信的方法。

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.

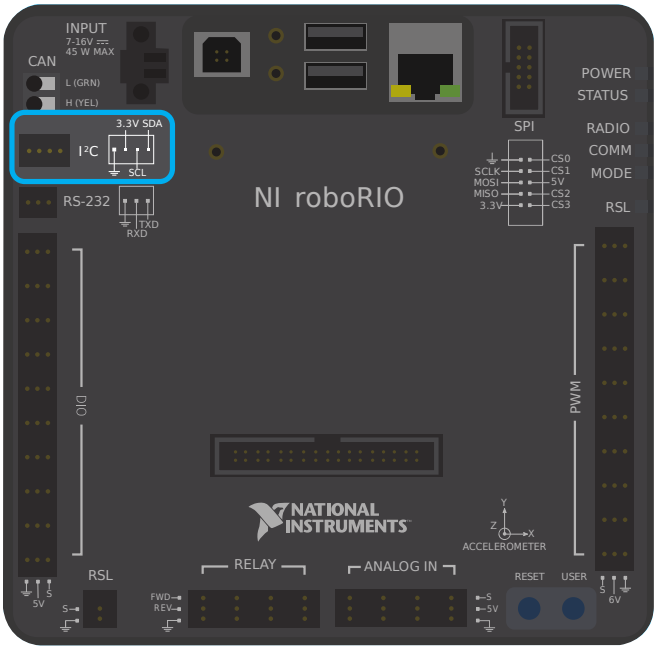
38.12.1 支持的串行总线类型

roboRIO 支持许多基本类型的串行通信:

- *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

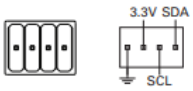


Table 5. I²C Port Signal Descriptions

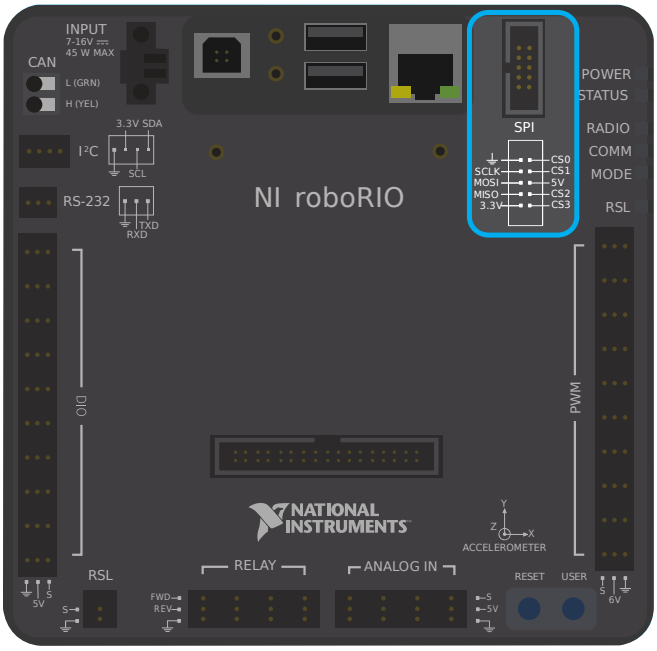
Signal Name	Direction	Description
GND	—	Reference for digital lines and +3.3 V power output.
3.3V	Output	+3.3 V power output.
SCL	Input or Output	I²C lines with 3.3 V output, 3.3 V/ 5 V-compatible input. Refer to the *I²C Lines* section for more information.
SDA	Input or Output	

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.

> **警告:** 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

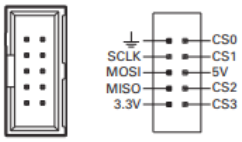


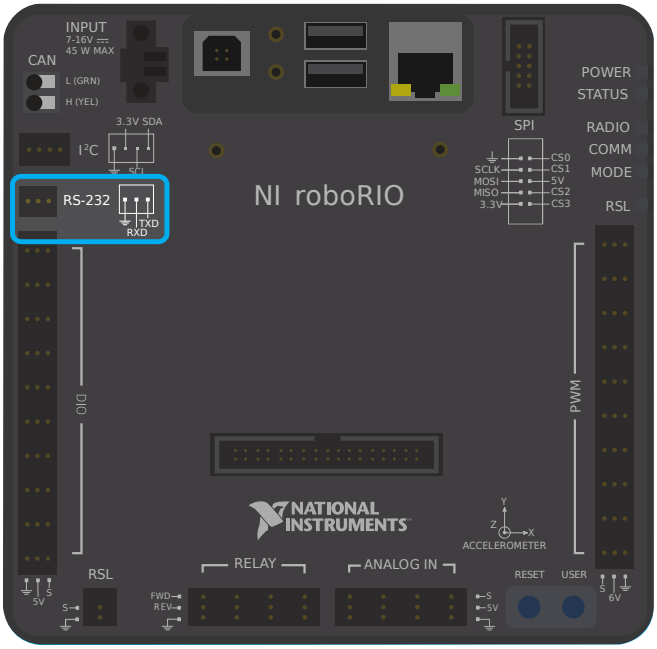
Table 12. SPI Port Signal Descriptions

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).

SPI 总线也可以通过‘MXP expansion port‘_ 使用。MXP 端口提供独立的时钟，输入/输出线和一个附加的 CS。

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

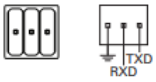


Table 6. RS-232 Serial Port Signal Descriptions

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.

要通过 RS-232 与外围设备通信，应将每个引脚连接到设备上相应的引脚。

RS-232 总线也可以通过 'MXP expansion port'_ 使用。

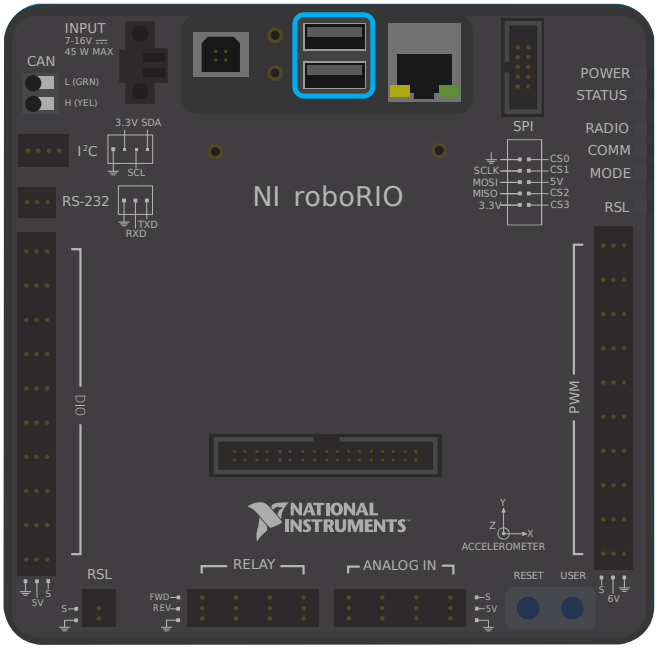
The roboRIO RS-232 serial port uses RS-232 signaling levels (+/- 15v). The MXP serial port uses CMOS signaling levels (+/- 3.3v).

备注： 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 网络仪表板*.

38.12.5 USB Client

roboRIO 上的 USB 端口之一是 USB-B 或 USB 客户端端口。可以使用标准 USB 电缆将其连接到设备，例如 Driver Station 计算机。

38.12.6 USB 主机



roboRIO 上的两个 USB 端口是 USB-A 或 USB 主机端口。可以使用例如相机或传感器的标准 USB 线将它们连接到设备。

38.12.7 MXP 扩展端口

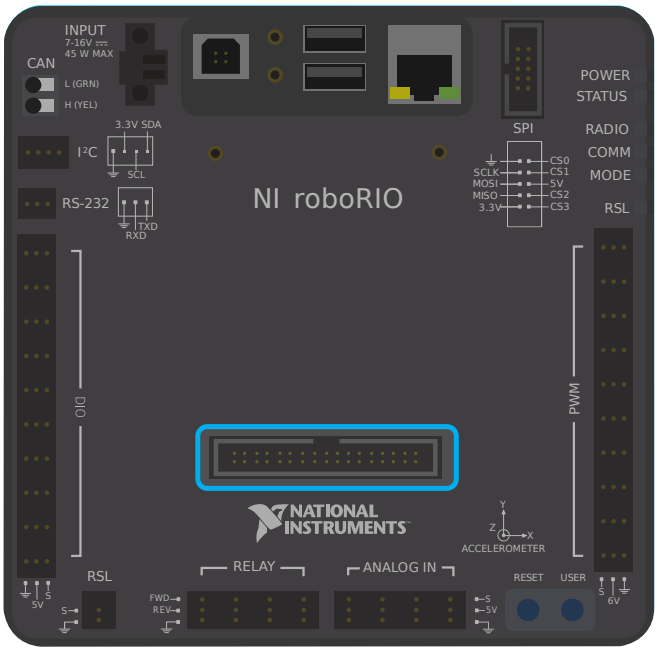
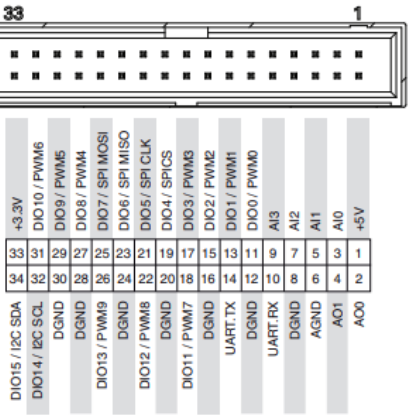


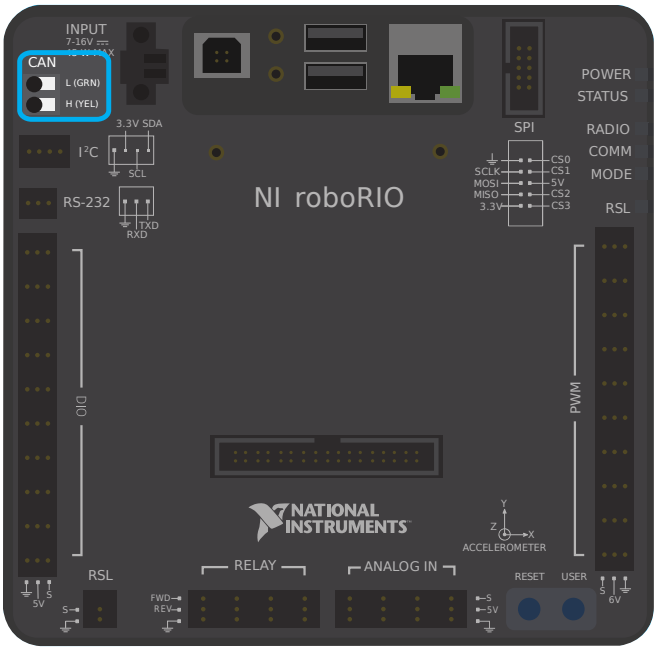
Figure 4. MXP Pinout



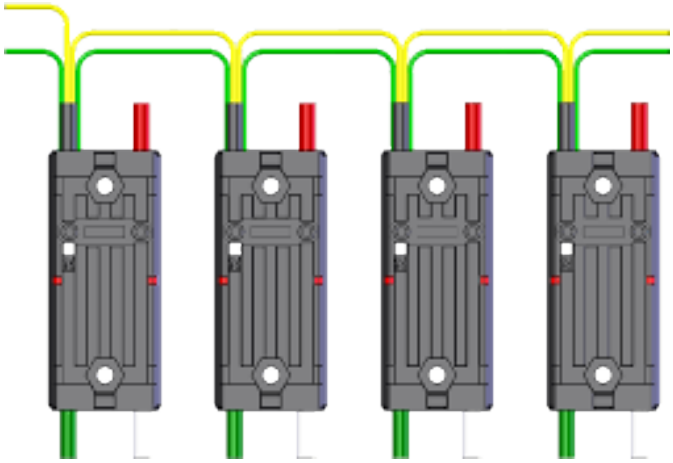
Several of the serial buses are also available for use through the roboRIO's *MXP* Expansion Port. This port allows users to make use of many additional *digital* and *analog* inputs, as well as the various serial buses.

为了方便起见，许多外围设备都直接连接到 MXP 端口，不需要用户进行任何接线。

38.12.8 CAN 总线



此外，roboRIO 支持通过 CAN 总线与外围设备进行通信。但是，由于 FRC CAN 协议非常特殊，因此很少有外围传感器支持该协议（尽管它广泛用于电机控制器）。使用 CAN 总线协议的优点之一是可以采用菊花链方式连接设备，如下所示。如果断开链中任何设备的电源，数据信号仍将能够到达链中的所有设备。



几个 CAN 主要使用的传感器。一些示例包括:

- CAN Based Time-of-Flight Range/Distance Sensor from playingwithfusion.com
- 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)*

有关使用与 CAN 总线连接的设备的更多信息，请参见以下文章:ref:*using can devices <docs/software/can-devices/using-can-devices:Using CAN Devices>*。

39

从 Romi 开始

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.

小技巧： A course that teaches programming using the Romi Robot is available via Thinkscape. Information on this course is available here

---

39.1 Romi 硬件和装配体

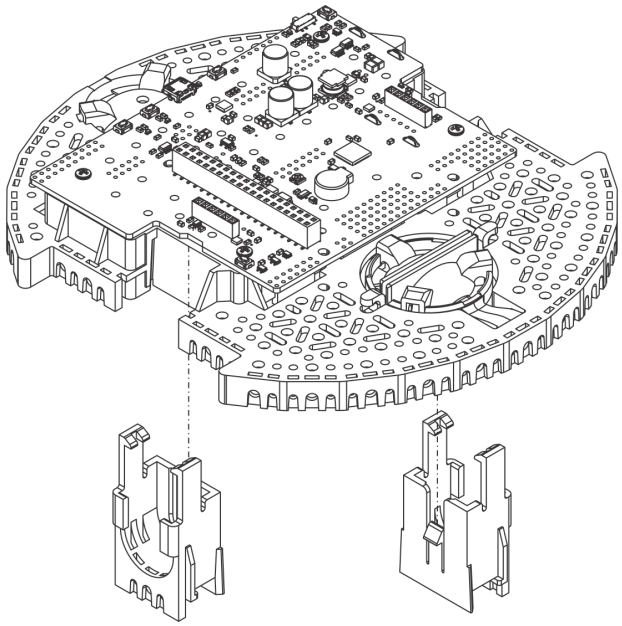
要开始使用 Romi，您需要具备必要的硬件。

1. Romi Kit from Pololu –订购可免运费
2. 树莓派 <https://www.amazon.com/gp/product/B07BFH96M3/>‘__ - 3B+ 或 4
3. 8GB(及以上) 的 Micro - SD 卡 <https://www.amazon.com/dp/B073K14CVB/>‘__
4. Micro SD 读卡器 <https://www.amazon.com/gp/product/B0779V61XB/>‘__ -如果你还没有的话
5. 6 AA 电池 <https://www.amazon.com/gp/product/B07TW9T8JW/>‘__ -最好是可充电的 (别忘了充电器)

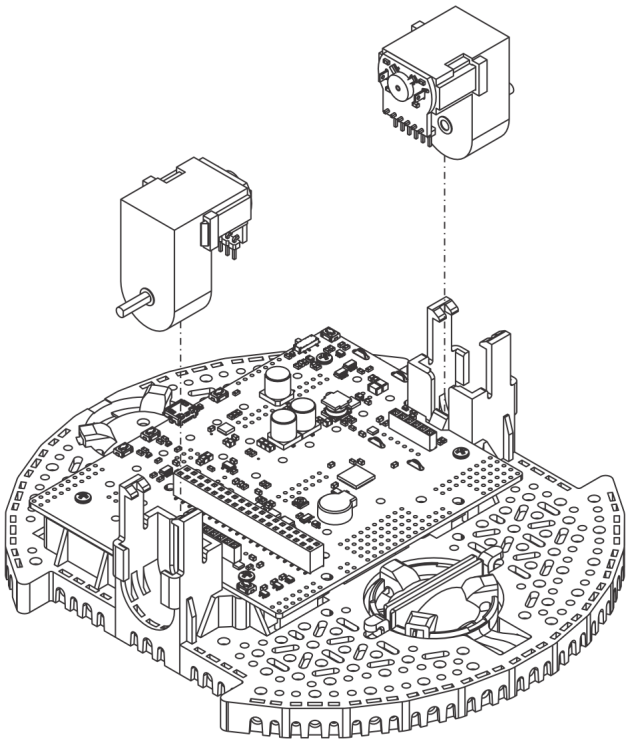
39.1.1 装配

Romi 机器人套件是预先焊接的，只需要在使用前组装好。一旦你收集了所有的材料，你可以开始组装:

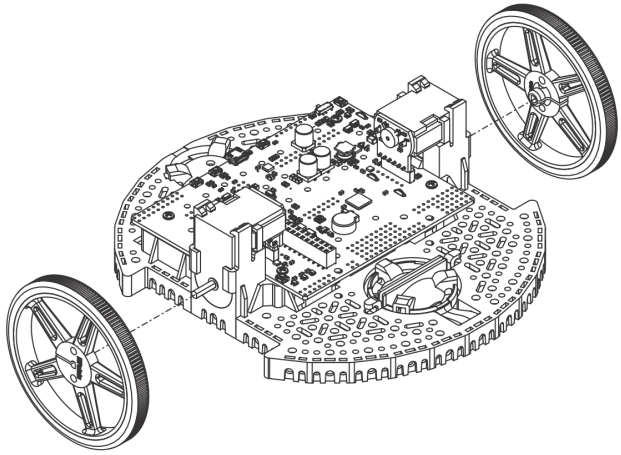
1. 按照指示将电机夹与底盘对齐，并将其牢固地压入底盘，直到电机夹的底部与底盘底部持平 (您可能会听到几声咔哒声)。



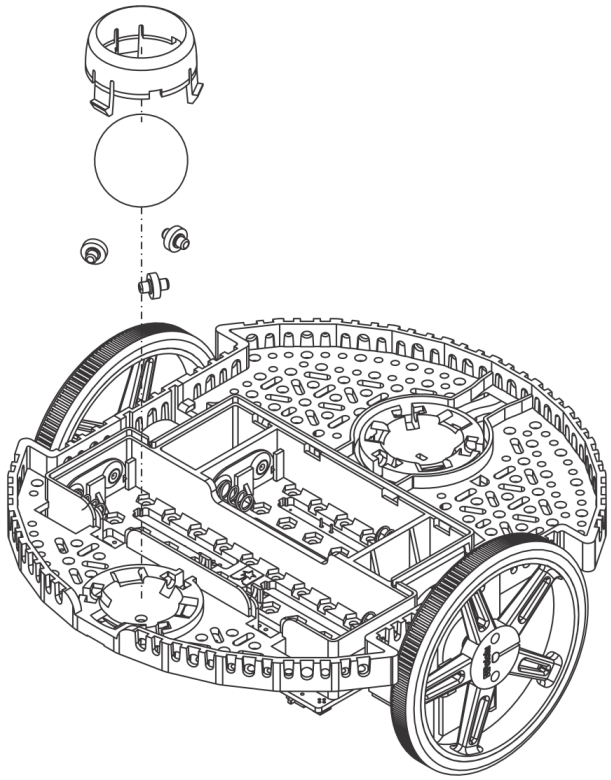
2. 将微型塑料齿轮马达推入马达夹，直到它们卡住。注意，电机阻塞夹释放，因此，如果您稍后需要拆卸电机支架，您将首先需要拆卸电机。迷你塑料齿轮马达和有扩展的电机轴结合以得到正交编码器的位置反馈。



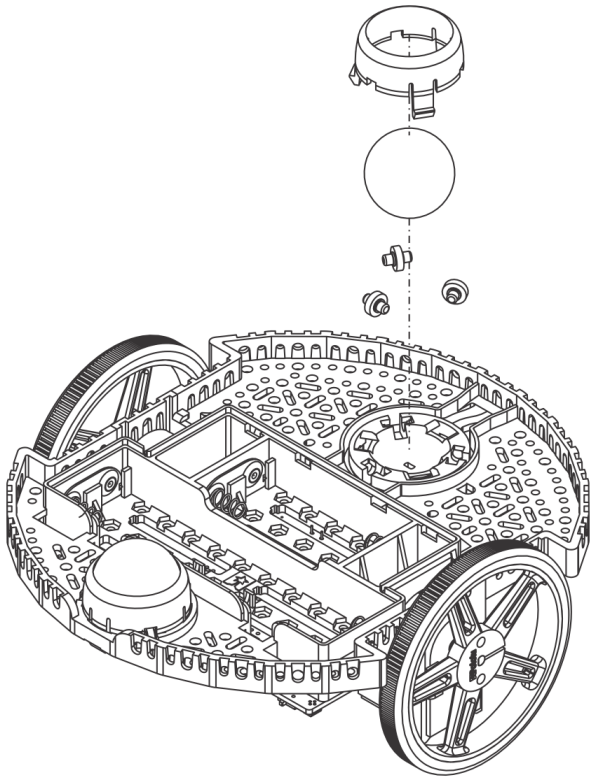
3. 将轮组压在电机的输出轴上，直到电机轴与轮组的外表面齐平。通过使电机的 D 轴的平坦部分与车轮正确地排列来将轮组和底盘设置在同一个平坦的表面。然后，放下底盘，将电机轴压入车轮，直到它接触到表面。



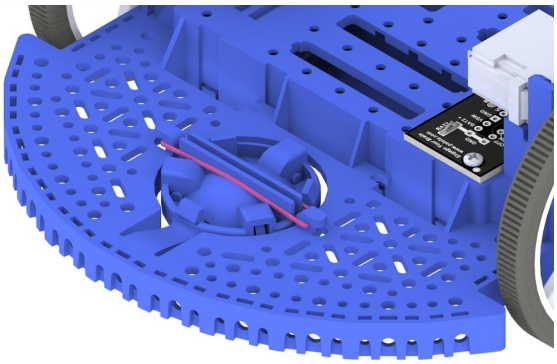
4. 将底盘倒过来，将后滚珠脚轮的三个滚轮放入底盘的开口内。将 1 英寸塑料球放在三个滚轮的顶部。然后推滚珠脚轮保留夹并在球和进入底盘时滑入到他们各自的位置。



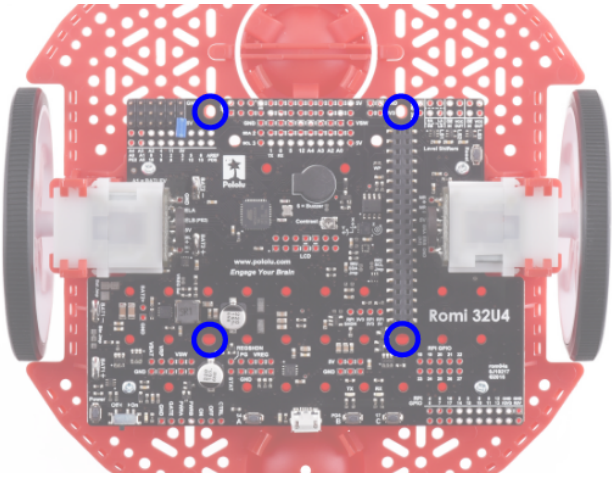
5. 重复为前面的球的脚轮，所以有一个脚轮在前面和后面的机器人。



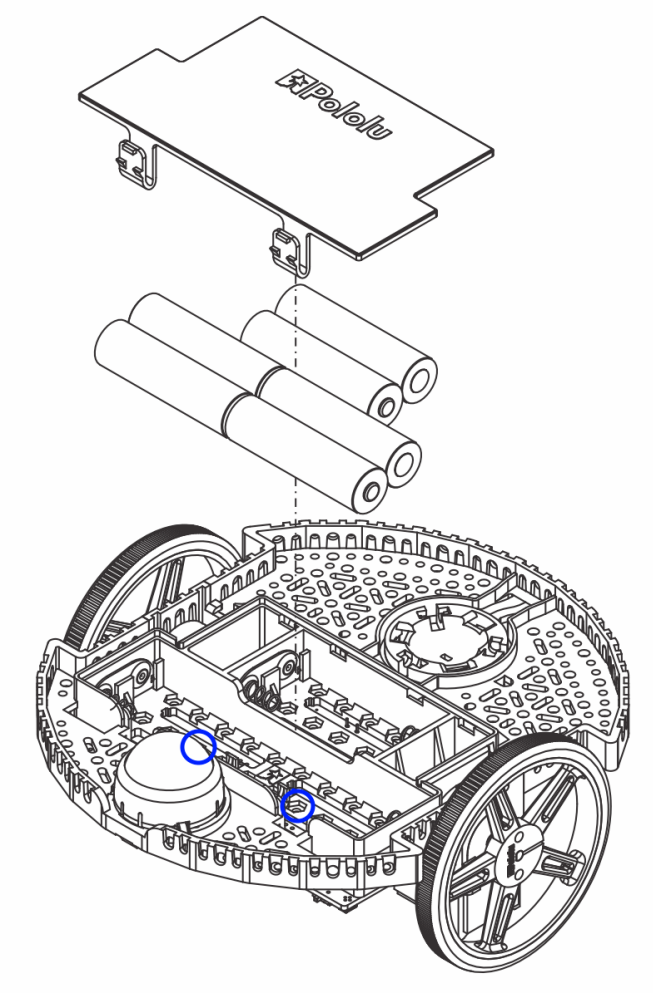
6. Optional: The front ball caster is supported by a flexible arm that acts as a suspension system. If you want to make it stiffer, you can wrap a rubber band around the two hooks located on either side of the ball caster on the top side of the chassis.



7. Install the standoffs to support the Raspberry Pi board. Two standoffs (thread side down) mount in the holes on the side of the Romi board closest to the "Romi 32U4" label as shown in the picture. The nuts for these standoffs are inside the battery compartment. The other two standoffs go into the holes on the opposite side of the board. To attach them, you will need a needle-nose pliers to hold the nut while you screw in the standoffs. The circled holes in the image below show where the standoffs should go.

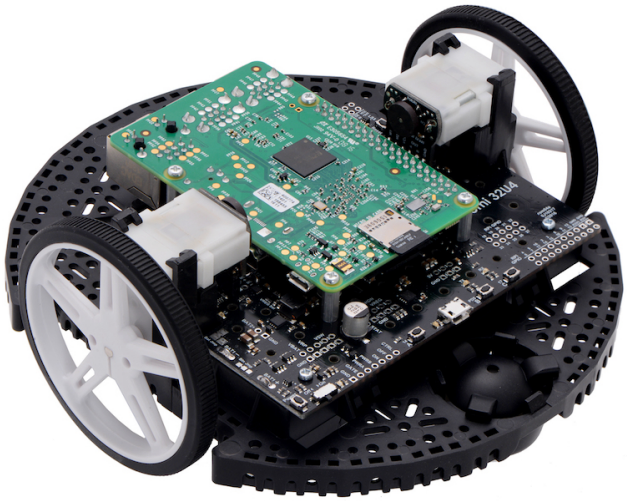


8. 底盘工作需要 4 或 6 节 AA 电池 (我们建议使用可充电 AA 镍氢电池)。电池的正确方向是由电池形状的孔在 Romi 底盘以及底盘本身的 + 和-指示器指示。



9. 将树莓派板倒置，小心对准 Pi 上的 2x20 针连接器与 Romi 上的 2x20 针插座。用均匀的压力推，注意不要弯曲任何插销。插入后，使用所提供的螺丝将树莓派板固定到上一步安装的支架上。

备注： 其中两个螺丝需要在电池隔间内的一个六角形孔中放置一个螺母。位置用上图中的蓝色圆圈表示。



你的 Romi 底盘的组装现在完成了!

39.2 成像你的 Romi

Romi 有 2 个微处理器板:

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.

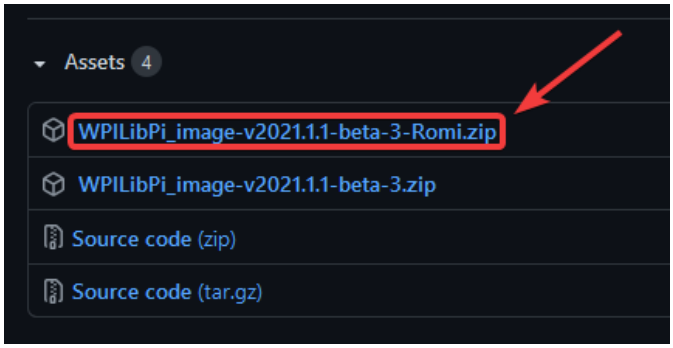
39.2.1 树莓版

下载

覆盆子 Pi 固件是基于 WPILibPi(以前为 FRCVision) 的，必须下载并写入覆盆子 Pi micro SD 卡。点击下方的 Assets 可以看到可用的图像文件:

Romi WPILibPi

请确保下载的是 Romi 版本，而不是 WPILibPi 的标准版本。Romi 版本的后缀是-Romi。下面的图片是一个例子



成像

安装映像的过程在这里描述:WPILibPi 安装。

无线网络设置

完成以下步骤，让树莓派可以和 Romi 一起使用:

1. 打开 Romi 通过滑动 Romi 32U4 板上的电源开关到 on 位置。第一次使用新映像启动时，需要大约 2-3 分钟来引导，同时调整文件系统的大小并重新启动。随后它将在不到一分钟的时间内启动。
2. 使用您的电脑，连接到 Romi WiFi 网络使用 SSID WPILibPi-<number>(其中 <number> 是基于树莓派的序列号) 和 WPA2 密码 WPILib2021!

备注: 注意: 如果在一个有多个运行 wpilibpi 的树莓派的环境中打开树莓派的电源，一个特定的树莓派的 SSID 也会通过耳机端口发出声音。默认的 SSID 也被写入/boot/default- SSID .txt 文件，可以通过将 SD 卡 (通过读卡器) 插入计算机并打开引导分区来读取。

---

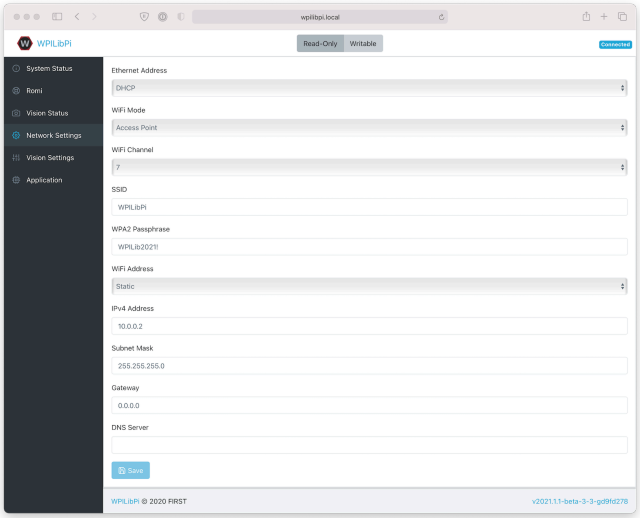
3. 打开 Web 浏览器，然后通过 “http://10.0.0.2/’’ 或 “http：//wpilibpi.local/’’ 连接到 Raspberry Pi 仪表板。

备注: Note: 默认情况下，映像以只读方式启动，因此有必要单击 Writable 按钮进行更改。完成更改后，单击 Read-Only 按钮以防止内存损坏。

---

4. 在仪表板 web 页面的顶部选择 Writable。
5. 通过在 WPA2 Passphrase 字段中设置新密码，更改 Romi 的默认密码。
6. 按页面底部的 Save 按钮保存更改。
7. Change the network SSID to a unique name if you plan on operating your Romi on a wireless network with other Romis.
8. 用你设置的新密码重新连接 Romi 的 WiFi 网络。

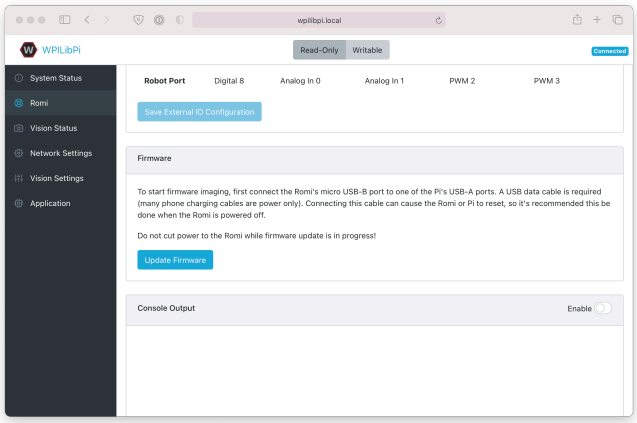
在完成所有更改后，请确保将指示板设置为只读。



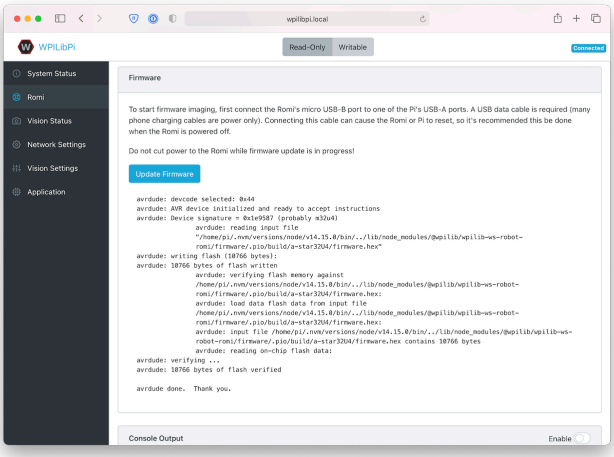
39.2.2 32U4 控制板

树莓派现在可以用来写入固件镜像到 32U4 控制板。

1. 把罗米关掉
2. 将树莓派的一个 USB 接口连接到 32U4 控制板的 micro USB 接口。
3. 打开 Romi 并连接到它的 Wifi 网络，并像前面的步骤一样连接到 web 仪表板。
4. 在 Romi 配置界面中，按“Update Firmware”按钮。



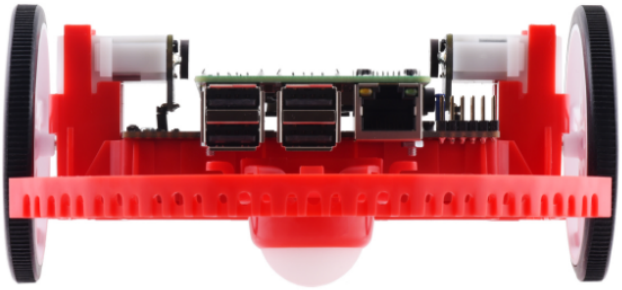
将出现一个控制台，显示固件部署过程的日志。一旦固件部署到 32U4 控制板，消息 avrdude 就完成了。谢谢你！就会出现。



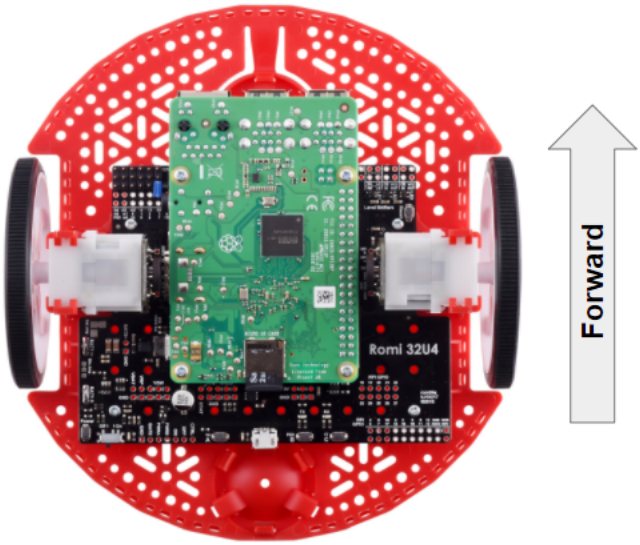
39.3 了解你的 ROMI

39.3.1 驱动器惯例

ROMI 的正面是树莓派 USB 接口, GPIO 插口和万向轮所在地。



在所有 ROMI 文档中，提到向前行驶时使用上述“前方”的定义。



39.3.2 Hardware, Sensors, and GPIO

ROMI 有以下几个内置硬件/外围设备:

- 2 个带编码器的齿轮减速电机
- 1 个惯性测量单元 (IMU)
- 3 个 led 灯 (绿、黄、红)
- 3 个按钮 (标注 A、B、C)
- 5x configurable GPIO channels (EXT)
- Buzzer

备注: The Buzzer is currently not supported by WPILib.

Motors, Wheels, and Encoders

ROMI 上使用的电机有 120:1 齿轮减速，在 4.5V 下的空载输出速度为 150 RPM。自由电流为 0.13 安培，失速电流为 1.25 安培。失速扭矩为 25 盎司-in (0.1765 N-m)，但内置的安全离合器在较低扭矩时可能开始打滑。

车轮的直径为 70mm(2.75”)。它们的轨道宽度为 141 毫米 (5.55 英寸)。

编码器直接连接到电机输出轴，每转速分辨率 (CPR) 有 12 个计数。在提供的齿轮传动比下，每轮转数为 1440。

The motor *PWM* channels are listed in the table below.

Channel	Romi 硬件组件
PWM 0	Left Motor
PWM 1	Right Motor

备注： 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	Romi 硬件组件
DIO 4	左编码器正交通道 A
DIO 5	左编码器正交信道 B
DIO 6	右编码器正交通道 A
DIO 7	右编码器正交信道 B

备注： 注意: 默认情况下，当 ROMI 移动时编码器开始计数。

惯性测量单元

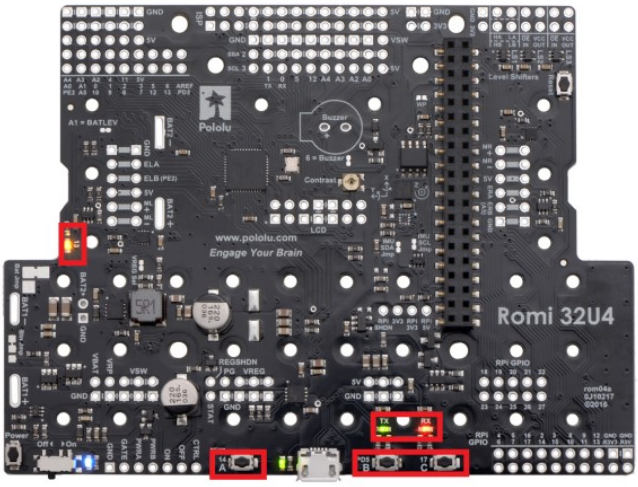
ROMI 包括意法半导体 LSM6DS33 惯性测量单元 (IMU)，其中包含一个三轴陀螺仪和一个三轴加速度计。

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.

ROMI Web UI 还提供了一种方法，在使用机器人代码之前校准陀螺仪和测量它的零偏移量。

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.

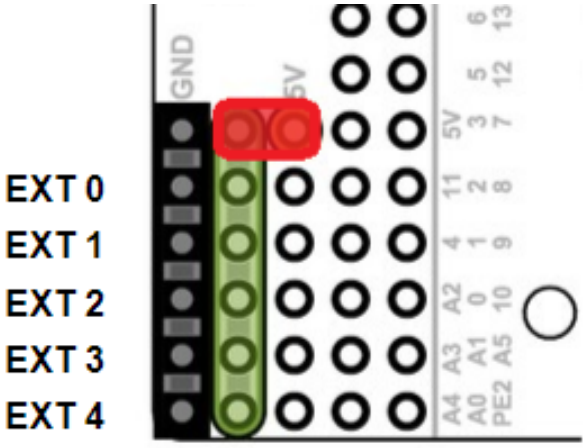


DIO 通道	Romi 硬件组件
DIO 0	按钮 A(只输入)
DIO 1	按钮 B（输入），绿色 LED（输出）
DIO 2	按钮 C（输入），红色 LED（输出）
DIO 3	黄色 LED（仅输出）

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*.

GPIO 通道通过一个 3-pin, servo style interface 暴露，有接地、电源和信号的连接 (接地距离单板边缘最近，信号距离单板内部最近)。

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.

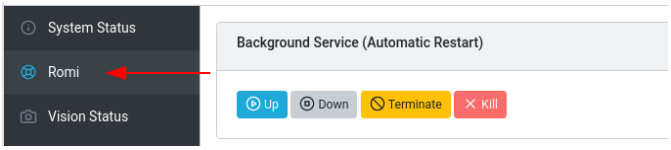
- PWM Motor Controllers (i.e. `Spark`)
- `Encoder`
- `AnalogInput`
- `DigitalInput`
- `DigitalOutput`
- `Servo`
- `BuiltInAccelerometer`

The following classes are provided by the Romi Vendordep.

- `RomiGyro`
- `RomiMotor`
- `OnboardIO`

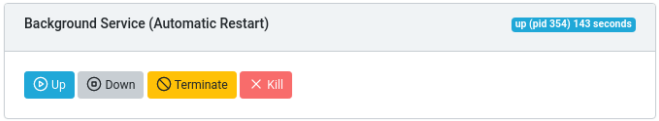
39.5 ROMI Web 界面

ROMI Web UI 作为 WPILibPi Raspberry Pi 图像的一部分进行安装。通过单击主 WPILibPi Web UI 导航栏上的 ROMI 选项卡可以访问它。



本节的其余部分将介绍 ROMI Web UI 的各个部分，并描述相关的功能。

39.5.1 后台服务状态



ROMI Web UI 的这一部分提供了有关当前运行的 ROMI Web 服务的信息 (它允许 WPILib 与 ROMI 对话)。UI 提供了控制来打开/关闭服务，并显示 web 服务的当前正常运行时间。

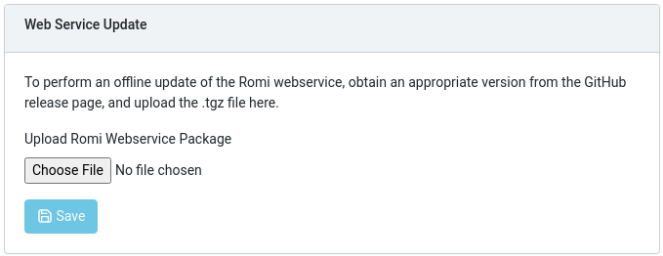
备注： 注意: 用户不需要经常使用本节中的功能，但它可以用于解决问题

39.5.2 Romi 状态

这个部分介绍 Romi 的相关信息，包括服务版本、电池电压、当前安装在 Romi 32U4 板上的固件与当前 web 服务版本是否兼容等。

备注： 如果固件不兼容，请参阅:doc: ‘Imaging your Romi 1 ‘章节。

39.5.3 Web 服务更新

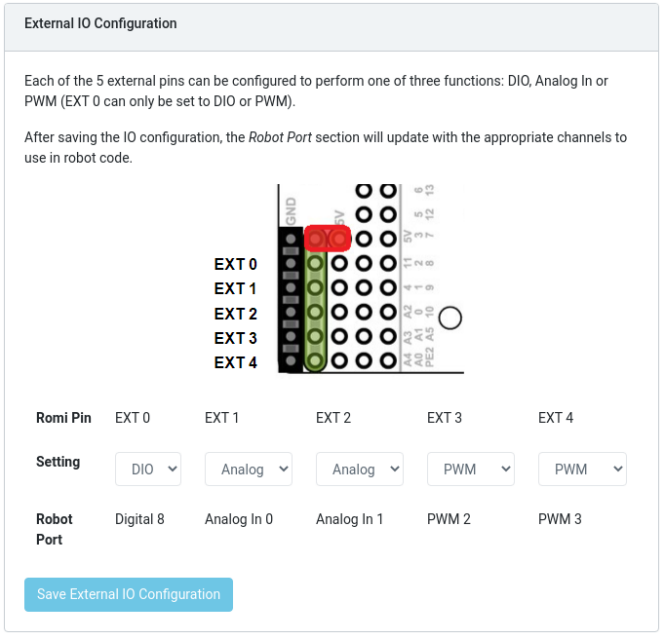


备注： 注意:Raspberry Pi 必须在可写模式下才能工作。

Romi WPILibPi 映像随附了 Romi Web 服务的最新版本（在发布时）。为了支持升级到 Romi Web 服务的较新版本，本节允许用户上传可以通过 Romi Web 服务 ‘GitHub releases 页面 <https://github.com/wpilibsuite/wpilib-ws-robot-romi/releases>‘__ 获得的预构建捆绑包。

要执行升级，请从 GitHub 发布页面下载适当的.tgz 文件。接下来，选择下载的.tgz 文件并单击 Save。更新后的 web 服务包将被上传到 Raspberry Pi，并进行安装。过一会儿，Romi Status 部分就会使用最新的版本信息更新自己。

39.5.4 外部 IO 配置



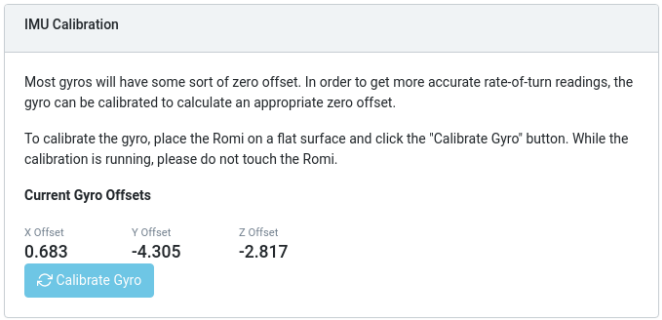
这一部分允许用户在 Romi 上配置 5 个外部 GPIO 通道。

备注： 注意:Raspberry Pi 必须在可写模式下才能工作。

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.

机器人端口行为每个配置好的 GPIO 通道提供了适当的 WPILib 映射。例如，EXT 0 被配置为数字 IO 通道，并且可以在 WPILib 中作为数字输入 (或数字输出) 通道 8 访问。

39.5.5 IMU 校准



备注： 注意:Raspberry Pi 必须在可写模式下才能工作。

本节允许用户在 Romi 上校准陀螺。陀螺通常有某种零偏移误差，并且校准允许 Romi 计算偏移量并在计算中使用它。

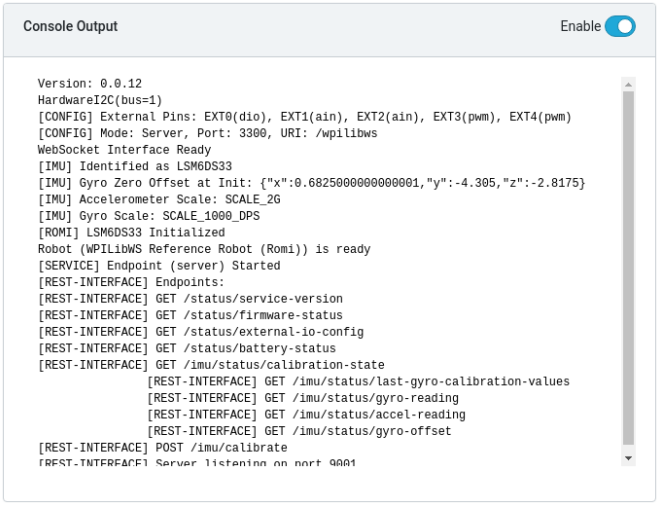
开始校准时，把 Romi 放在一个平坦、稳定的表面上。然后，点击校准陀螺按钮。将出现一个进度条，显示当前的校准过程。校准完成后，最新的偏移值将显示在屏幕上，并向 Romi web 服务注册。

这些偏移值保存到磁盘，并在重新引导之间保持不变。

39.5.6 固件

备注： 注意: 请参阅成像 Romi 部分

39.5.7 控制台输出



启用后，本节允许用户查看 Romi web 服务提供的原始控制台输出。这对于解决 Romi 问题非常有用，或者只是为了了解更多关于幕后发生的事情。

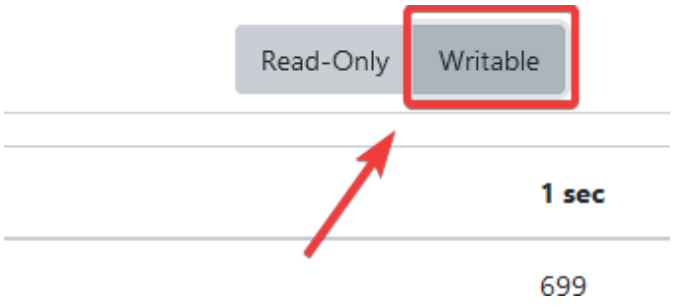
39.5.8 桥接模式

桥接模式允许您的 Romi 机械手连接到 WiFi 网络，而不是充当接入点（AP）。这在远程学习环境中特别有用，因为您可以在使用 Romi 的同时使用互联网而无需额外的硬件。

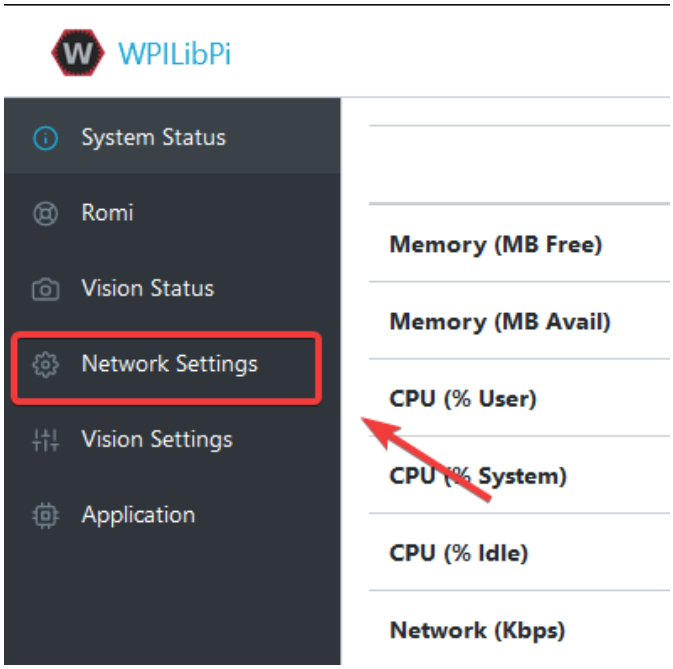
备注： 在受限的网络环境（教育机构）中，桥接模式不太可能正常工作。

---

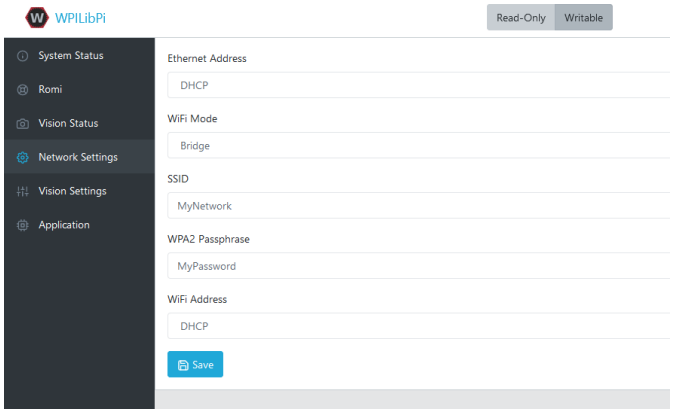
1. 在顶部菜单中启用“可写”。



2. 点击“网络设置”。



3. 必须应用以下网络设置：



- **以太网**：DHCP
- ** WiFi 模式 **：桥接
- ** SSID **：您网络的 SSID（名称）
- ** WPA2 密码 **：您的 wifi 网络的密码
- ** WiFi 地址 **：DHCP

应用设置后，请重新启动 Romi。现在，当连接到指定的网络时，您应该能够在 Web 浏览器中导航到“wpilibpi.local”。

无法访问罗米

如果 Romi 的网桥设置正确，而您无法访问它，我们有一些解决方法。

- 以太网进入罗米
- 重塑 Romi

某些受限制的网络可能会干扰 Romi 解析的主机名，您可以使用 'Angry IP Scanner <https://angryip.org/>'__ 查找 IP 地址来解决此问题。

警告： 在对网络中的设备执行 ping 操作时，Angular IP Scanner 被某些防病毒软件标记为间谍软件！这是一个安全的应用程序！

39.6 编程 Romi

为 Romi 编写程序与为普通的 FRC 机器人编写程序非常相似，事实上，所有相同的工具 (Visual Studio Code、Driver Station、SmartDashboard 等) 都可以在 Romi 上使用。

39.6.1 创建 Romi 程序

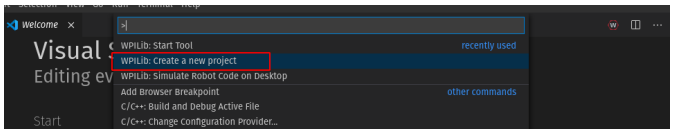
Creating a new program for a Romi is like creating a normal FRC program, similar to the *Zero To Robot* programming steps.

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.

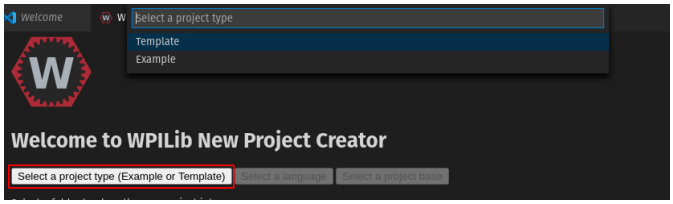
备注： In order to program the Romi using C++, a compatible C++ desktop compiler must be installed. See *Robot Simulation - Additional C++ Dependency*.

创建一个新的 WPILib Romi 项目

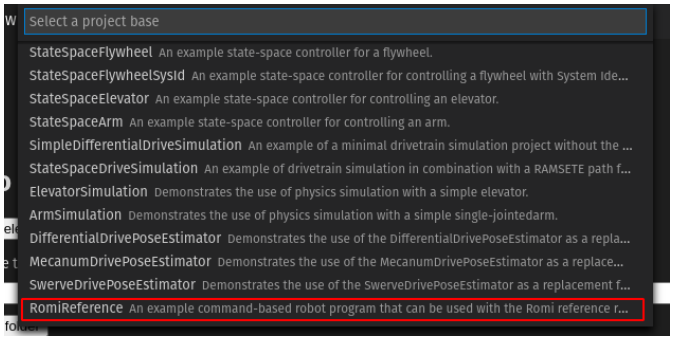
使用 Ctrl+Shift+P 打开 Visual Studio 代码命令面板，并在提示符中键入 New Project。选择 Create a new project 命令。



这将弹出“新建项目创建器”窗口。在这里，单击 Select a project type (Example 或 Template)，并从出现的提示符中选择 Example。



接下来，将出现一个示例列表。滚动列表以找到 RomiReference 示例。



填写 New Project Creator 中的其他字段，并单击 Generate Project 创建新的 robot 项目。

运行一个 Romi 程序

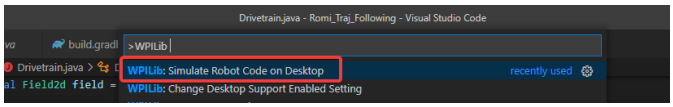
Once the robot project is generated, it is essentially ready to run. The project has a pre-built `Drivetrain` class and associated default command that lets you drive the Romi around using a joystick.

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.

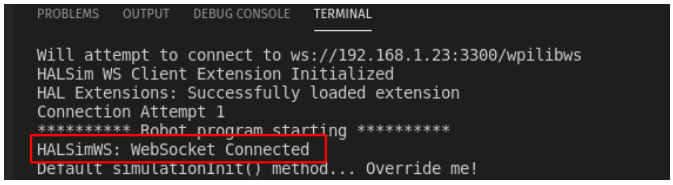
To run a Romi program, first, ensure that your Romi is powered on. Next, connect to the `WPILibPi-<number>` WiFi network broadcast by the Romi. If you changed the Romi network settings (for example, to connect it to your own WiFi network) you may change the IP address that your program uses to connect to the Romi. To do this, open the `build.gradle` file and update the `wpi.sim.envVar` line to the appropriate IP address.

```
//Sets the websocket client remote host.
wpi.sim.envVar("HALSIMWS_HOST", "10.0.0.2")
wpi.sim.addWebsocketsServer().defaultEnabled = true
wpi.sim.addWebsocketsClient().defaultEnabled = true
```

Now to start your Romi robot code, open the WPILib Command Palette (type `Ctrl+Shift+P`) and select “Simulate Robot Code”, or press `F5`.



如果一切顺利，您应该在控制台输出中看到一行读取 HALSimWS: WebSocket Connected 的内容。



您的 ROMI 代码现在正在运行!

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

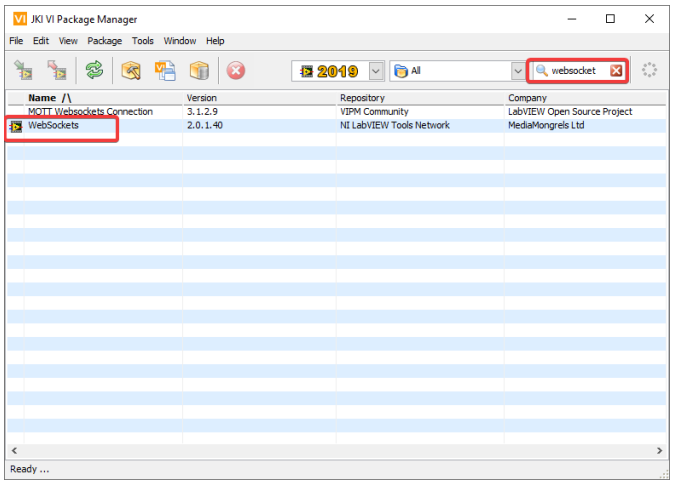
Creating a new program for a Romi is no different than creating a normal FRC |reg| program, similar to the *Zero To Robot* programming steps. Initially, you may wish to create a separate project for use on just the Romi as the Romi hardware may be connected to different ports than on your roboRIO robot.

The Romi Robot used *PWM* ports 0 and 1 for left and right side respectively.

Installing the WebSockets VI

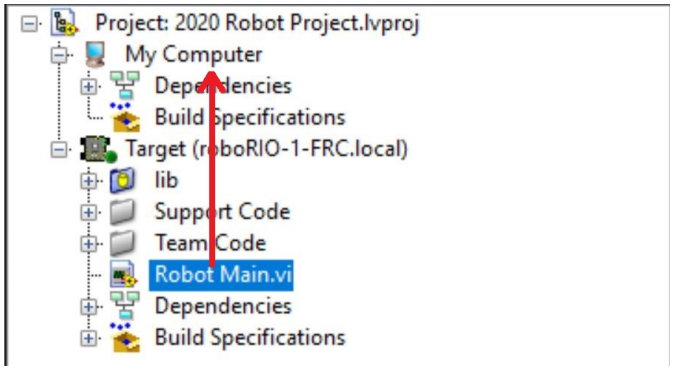
One aspect where a Romi project differs from a regular FRC |reg| robot project is that the code is not deployed directly to the Romi. Instead, a Romi project runs on your development computer, and leverages the WPILib simulation framework to communicate with the Romi robot. WebSockets is the protocol 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*.



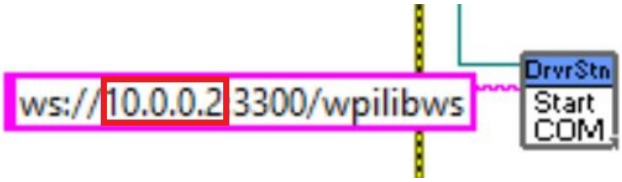
Changing the Project Target

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.



Running a Romi Program

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.

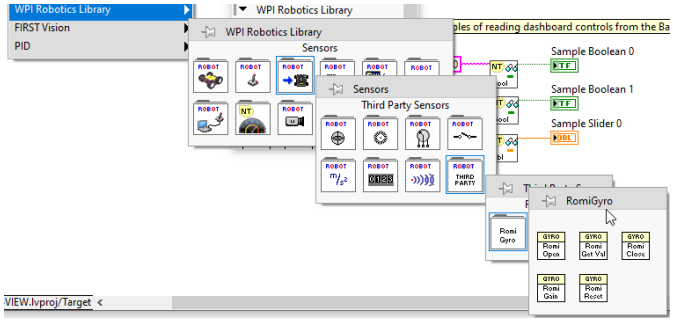
It is recommended to run the Driver Station software on the same computer as the LabVIEW code. Once your program successfully connects to the Driver Station, it will automatically notify the Driver Station that the code is running on the Desktop, allowing the Driver Station to connect without you changing 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.

备注: 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

Getting Started with XRP

The XRP is a small and inexpensive robot designed for learning about programming FRC robots. All the same tools used for programming full-sized FRC robots can be used to program the XRP. The XRP comes with two drive motors with integrated wheel encoders. It also includes an *IMU* sensor that can be used for measuring headings and accelerations. Using it is as simple as writing a robot program, and running it on your computer. It will command the XRP to follow the steps in the program.

The XRP provides a similar use case as the *Romi* with similar functionality, albeit using a lower power processor and is overall lower in cost.

40.1 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

备注： 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

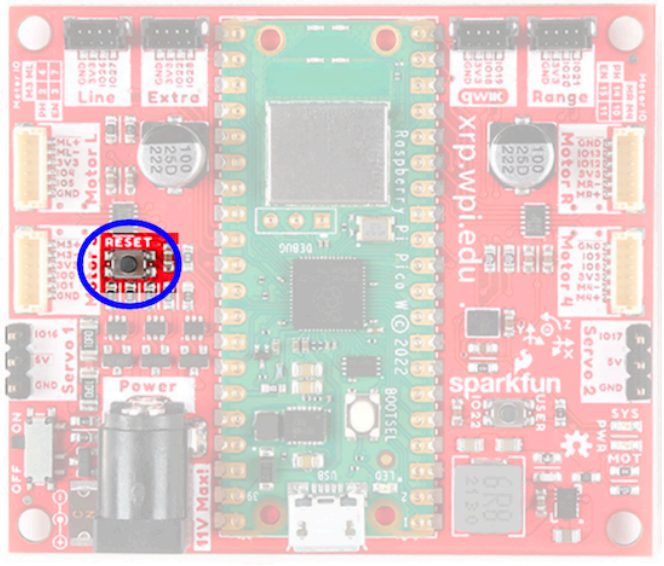
The XRP firmware must be downloaded and written to the Pico W. Click on `Assets` at the bottom of the description to see the available image files:

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.

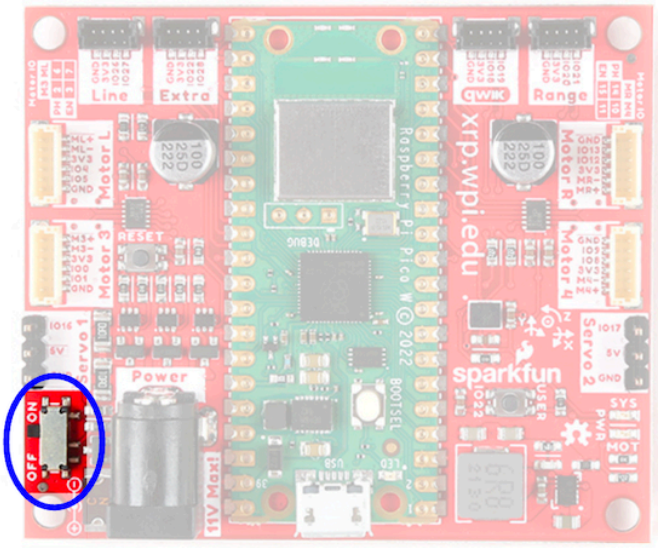


4. The board will temporarily disconnect from your computer, and then reconnect as a USB storage device named `RPI-RP2`.
5. Drag the `.uf2` firmware file into the `RPI-RP2` drive, and it will automatically update the firmware.
6. Once complete, the `RPI-RP2` USB storage device will disconnect. At this 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`.

备注: 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.

备注: 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.

备注: 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

备注: 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.

备注: 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

备注： 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

备注： 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`

备注： The PWM motor controller classes (e.g. `Spark`) and `Servo` are not supported. The XRP requires use of specialized `XRPMotor` and `XRPServo` classes.

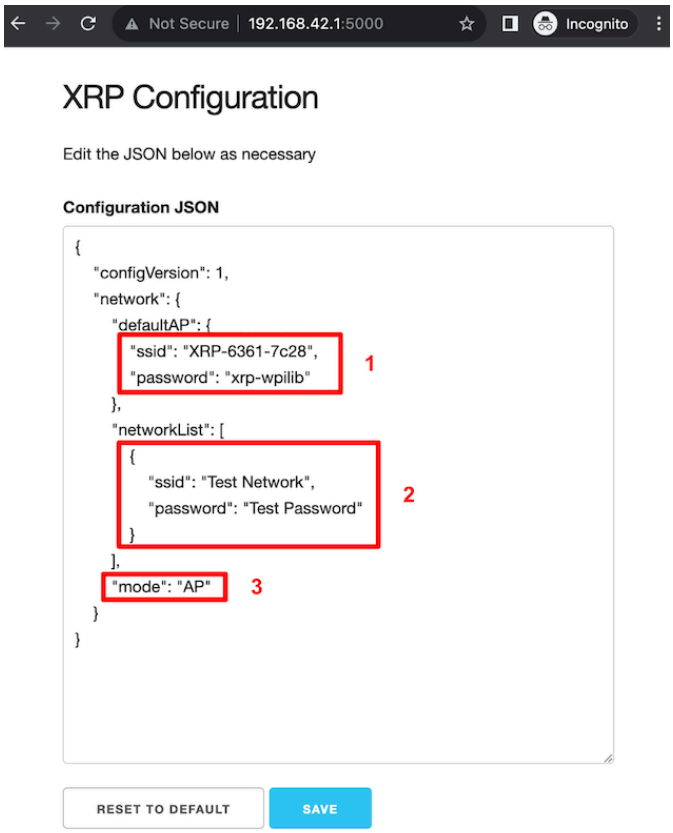
The following classes are provided by the XRP Vendordep (built-in to WPILib).

- `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.

备注： 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

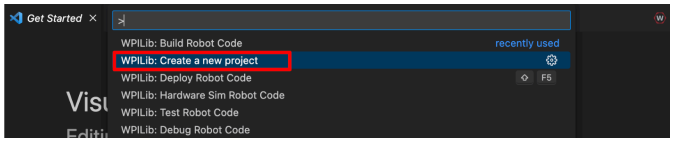
Creating a new program for an XRP is like creating a normal FRC program, similar to the *Zero To Robot* programming steps.

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.

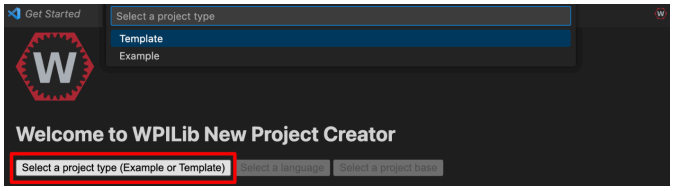
备注： In order to program the XRP using C++, a compatible C++ desktop compiler must be installed. See *Robot Simulation - Additional C++ Dependency*.

Creating a New WPILib XRP Project

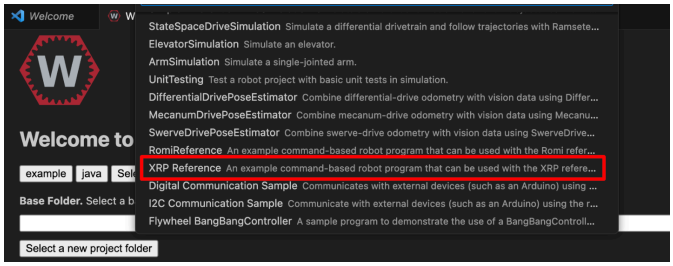
Bring up the Visual Studio Code command palette with `Ctrl+Shift+P`, and type "New project" into the prompt. Select the "Create a new project" command:



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

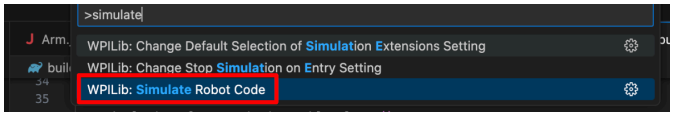
Once the robot project is generated, it is essentially ready to run. The project has a pre-built `Drivetrain` class and associated default command that lets you drive the XRP around using a joystick.

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!

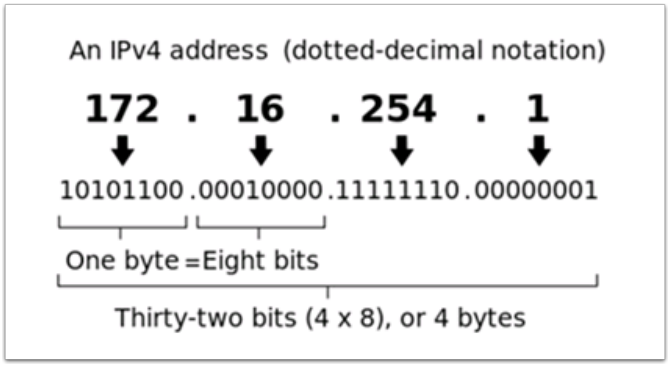
41 网络介绍

本节概述了基本的机器人上与 driver station 和 roboRIO 之间的通信有关的配置和用法。

41.1 网络基础

41.1.1 什么是 IP 地址？

IP 地址是一串唯一的数字字符串，由英文句点分隔标识网络上的每个设备。每个 IP 地址分为 4 个部分（八位字节），范围从 0-255。



如上所示，这意味着每个 IP 地址都是 32 位地址，这意味着有 2:sup:‘32‘个地址，或者可能有近 4300000000 个地址。然而，其中大多数都公开用于 web 服务器之类的东西。

这带来了我们 IP 寻址的 “第一个关键点”：网络上的每个设备都必须具有唯一的 IP 地址。两个设备不能具有相同的 IP 地址，否则会发生冲突。

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.

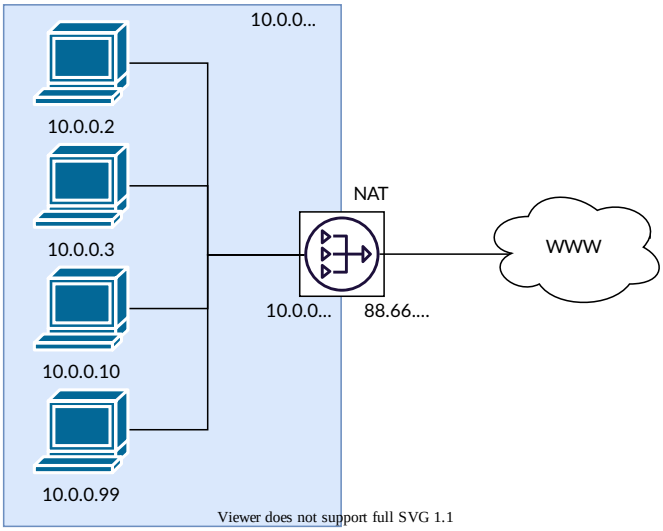
41.1.2 公有 IP 地址与私有 IP 地址

为了有效使用 IP 地址，实施了“保留 IP 范围”的想法。简而言之，这意味着有些 IP 地址范围永远不会分配给 Web 服务器，而只会用于本地网络，例如您家中的那些。

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:

类	位元	起始地址	结束地址	地址数
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

这些保留范围使我们可以为整个房屋分配一个“未保留的 IP 地址”，然后使用保留范围内的多个地址将多台计算机连接到 Internet。家庭互联网路由器上的一个称为 NAT（网络地址转换）的过程负责处理以下过程：跟踪哪个私有 IP 正在请求数据，使用公共 IP 从 Internet 请求该数据，然后传递返回的数据。数据返回到请求它的专用 IP。这使我们可以为许多本地网络使用相同的保留 IP 地址，而不会引起任何冲突。下面显示了此过程的图像。



备注： 对于 FRC | reg |网络，我们将使用“10.0.0.0”范围。此范围允许我们对 IP 地址使用“10.TE.AM.xx”格式，而使用 B 级或 C 级网络将仅允许一部分团队遵循该格式。对于 FRC Team 1750，这种格式的示例为 “10.17.50.1’’。

41.1.3 这些地址如何分配？

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).

动态地

动态分配 IP 地址意味着我们让网络上的设备管理 IP 地址分配。这是通过动态主机配置协议（DHCP）完成的。DHCP 具有许多组件，但是在本文档的范围内，我们会将其视为自动管理网络的服务。每当您将新设备插入网络时，DHCP 服务都会看到该新设备，然后为其提供可用的 IP 地址以及该设备进行通信所需的其他网络设置。这可能意味着有时我们不知道每个设备的确切 IP 地址。

啥是 DHCP 服务器?

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.

这意味着，如果遇到需要 DHCP 服务器为网络设备分配 IP 地址的情况，这就像找到最近的家用路由器并将其插入一样简单。

静态

静态分配 IP 地址意味着我们正在手动告诉网络上的每个设备我们希望它拥有哪个 IP 地址。通过每个设备上的设置进行此配置。通过禁用网络上的 DHCP 并手动分配地址，我们的好处是可以知道网络上每个设备的确切 IP 地址，但是由于我们手动设置了每个设备，并且没有服务跟踪使用的 IP 地址，因此必须自己对此进行跟踪。在静态设置 IP 地址时，我们必须注意不要分配重复的地址，并且必须确保我们在每个设备上正确设置了其他网络设置（例如子网掩码和默认网关）。

41.1.4 什么是本地链接？

如果设备没有 IP 地址，则无法在网络上通信。如果我们拥有一台设置为从 DHCP 服务器动态获取其地址的设备，但是网络上没有 DHCP 服务器，这可能会成为一个问题。例如，当您将一台笔记本电脑直接连接到 roboRIO 且两者都设置为动态获取 IP 地址时。这两个设备都不是 DHCP 服务器，并且由于它们是网络上仅有的两个设备，因此不会自动为它们分配 IP 地址。

本地链接地址为我们提供了一组标准地址，如果设置为动态获取的设备无法获取地址，则可以“回退”到这些地址。如果发生这种情况，设备将在“169.254.xx.yy”地址范围内为自己分配一个 IP 地址；这是本地链接地址。在上面的 roboRIO 和计算机示例中，两台设备都将意识到尚 未为其分配 IP 地址，而是为自己分配了本地链接地址。一旦为它们都分配了“169.254.xx.yy”范围内的地址，即使它们设置为动态并且 DHCP 服务器未分配地址，它们也将位于同一网络中并且能够通信。

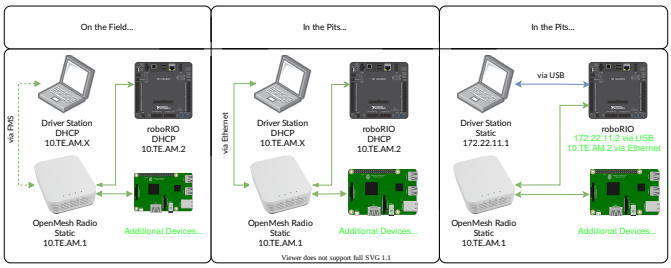
41.1.5 FRC 的 IP 寻址

查阅:doc:‘IP Networking Article <ip-configurations>‘获取更多资讯。

混合动态配置和静态配置

在场上，只要设备没有静态设置在“10.TE.AM.xx”范围内，并且只要没有 IP 地址冲突，该字段就分配 DHCP 地址，团队就不会在以上部分中发现任何问题。

在 Pits 区中，出于以下原因，团队可能会遇到混合使用静态和 DHCP 设备的问题。如上所述，如果没有服务器，DHCP 设备将回退到本地链接地址（“169.254.xx.yy’’）。对于静态设备，IP 地址将始终相同。如果不存在 DHCP 服务器，并且 roboRIO，驱动程序站和便携式计算机回退到链接本地地址，则位于 10.TE.AM.xx 范围内的静态设置的设备将位于不同的网络中，并且对于使用链接本地地址。下面提供了对此的直观描述：



警告: 通过 USB 连接到 roboRIO 时，需要配置:ref:*docs/networking/networking-utilities/portforwarding:Port Forwarding* 来连接 OpenMesh 路由器设备（在上面显示的绿色网络上）。

Available Network Ports

Please see R704 of the 2024 Game Manual for information regarding available network ports.

41.1.6 mDNS

mDNS 或多播域名系统是一种协议，它使我们可以从 DNS 的功能中受益，而无需在网络上安装 DNS 服务器。为了更清楚地说明这一点，让我们退后一步，谈谈什么是 DNS。

什么是 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.

示例 2：在您的家庭网络上，您有一个要从笔记本电脑连接到的名为 “MYCOMPUTER’’ 的服务器。您的网络使用 DHCP，因此您不知道“MYCOMPUTER”的 IP 地址，但是 DNS 允许您仅使用“MYCOMPUTER”名称进行连接。此外，每当 DHCP 分配刷新时，“MYCOMPUTER” 可能会以不同的地址结尾，但是由于您是通过使用 “MYCOMPUTER” 名称而不是特定的 IP 地址进行连接的，因此 DNS 记录已更新，您仍然能够连接。

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.

FRC 的 DNS

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 为我们提供了与传统 DNS 相同的优势，但是仅以不需要服务器的方式实现。每当用户要求使用友好名称连接到设备时，mDNS 都会发出一条消息，要求具有该名称的设备标识自己。然后，具有该名称的设备将发送包括其 IP 地址的返回消息，以便网络上的所有设备都可以更新其信息。mDNS 使我们可以将 roboRIO 称为 “roboRIO-TEAM-FRC.local’’ 并将其连接到 DHCP 网络上。

备注： 如果用于 FRC 的设备不支持 mDNS，则会为该设备分配 10.TE.AM.20-10.TE.AM.255 范围内的 IP 地址，但我们不知道要连接的确切地址，我们将无法像以前一样使用友好名称。在这种情况下，设备将需要具有静态 IP 地址。

---

mDNS-原理

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-提供者

要使用 mDNS，必须在您的 PC 上安装 mDNS 工具。以下是每个主要平台的一些常见 mDNS 工具：

Windows:

- **NI mDNS Responder:** 用 NI FRC Game Tools 安装。
- **Apple Bonjour:** iTunes 内安装

OSX:

- **Apple Bonjour:** 默认安装

Linux:

- **nss-mDNS/Avahi/Zeroconf:** 默认在某些 Linux 变体（例如 Ubuntu 或 Mint）上安装并启用。可能需要在其他产品（例如 Arch）上安装或启用

mDNS - 防火墙

备注: 根据您的 PC 配置，可能不需要进行任何更改，此部分用于帮助进行故障排除。

为了正常工作，必须允许 mDNS 穿过防火墙。因为网络流量来自 mDNS 实施，而不是直接来自 Driver Station 或 IDE，所以仅允许那些应用程序通过是不够的。解决 mDNS 防火墙问题的主要方法有两种：

- 为 mDNS 实施添加应用程序/服务例外（NI mDNS Responder 为 `C:\Program Files\National Instruments\Shared\mDNS Responder\nimdnsResponder.exe`）
- 为往返于 UDP 5353 的流量添加端口例外。IP 范围：
  - `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-浏览器支持

只要安装了 mDNS 提供程序，大多数 Web 浏览器就应该能够使用 mDNS 地址访问 roboRIO Web 服务器。这些浏览器包括 Microsoft Edge，Firefox 和 Google Chrome。

41.1.7 USB

如果使用 USB 接口，则无需进行网络设置（您将需要 ref:‘docs/zero-to-robot/step-2/frc-game-tools:Installing the FRC Game Tools‘以提供 roboRIO USB 驱动）。roboRIO 驱动程序将自动配置主机（您的计算机）的 IP 地址，并且 roboRIO 和上面列出的软件应该能够找到并利用您的 roboRIO。

41.1.8 以太网/无线

The 给你的路由器编程 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 总结

IP 地址使我们能够与网络上的设备进行通信。对于 FRC，如果我们连接到 DHCP 服务器或静态分配了这些地址，则这些地址将在 10.TE.AM.xx 范围内，或者在本地链接的 “169.254.xx.yy’’ 范围内如果设备设置为 DHCP，但是不存在服务器。有关 IP 地址如何工作的更多信息，请参见微软的这篇文章 this

如果网络上的所有设备均支持 mDNS，则可以将所有设备设置为 DHCP 并使用其友好名称进行引用（例如 “roboRIO-TEAM-FRC.local’’）。如果某些设备不支持 mDNS，则需要将其设置为使用静态地址。

如果将所有设备都设置为使用 DHCP 或静态 IP 分配（具有正确的静态设置），则通讯应在 Pits 区和场地内均可进行，而无需进行任何更改。如果混合使用某些静态设备和某些 DHCP 设备，则静态设备将在场地内进行连接，但不会在 Pits 区中进行连接。可以通过以下方式解决此问题：将所有设备设置为静态设置，或者保留当前设置并在 Pits 区中提供 DHCP 服务器。

41.2 IP 配置

备注： 本文档描述了场地和 Pits 区在事件中使用的 IP 配置，潜在问题和解决方法配置。

41.2.1 TE.AM IP 表示法

在本文档的许多地方，符号 TE.AM 用作 IP 的一部分。此表示法是指将您的四位数团队号码分成两对数字，用于 IP 地址八位位组。

举例：`10.TE.AM.2`

Team 12 - `10.0.12.2`

Team 122 - `10.1.22.2`

Team 1212 - `10.12.12.2`

Team 1202 - `10.12.2.2`

Team 1220 - `10.12.20.2`

Team 3456 - `10.34.56.2`

41.2.2 场地内

本节介绍了连接到 Field Network 进行比赛时的联网

场地内的 DHCP 配置

The Field Network runs a *DHCP* server with pools for each team that will hand out addresses in the range of `10.TE.AM.20` to `10.TE.AM.199` with a subnet mask of `255.255.255.0`, and a default gateway of `10.TE.AM.4`. When configured for an event, the Team Radio runs a DHCP server with a pool for devices onboard the robot that will hand out addresses in the range of 10.TE.AM.200 to 10.TE.AM.219 with a subnet mask of 255.255.255.0, and a gateway of 10.TE.AM.1.

- OpenMesh OM5P-AN or OM5P-AC radio - Static `10.TE.AM.1` programmed by Kiosk
- roboRIO - DHCP `10.TE.AM.2` assigned by the Robot Radio
- Driver Station - DHCP ("Obtain an IP address automatically") 10.TE.AM.X assigned by field
- IP camera (if used) - DHCP `10.TE.AM.Y` assigned by Robot Radio
- Other devices (if used) - DHCP `10.TE.AM.Z` assigned by Robot Radio

场地内静态配置

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.

- OpenMesh 路由器-静态 "10.TE.AM.1'' 由 Kiosk 编程。
- roboRIO-静态 "10.TE.AM.2'' 是一个合理的选择，子网掩码为 "255.255.255.0''（默认）
- 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`.
- IP 摄像头（如果使用）-静态 "10.TE.AM.11'' 将是一个合理的选择，子网 "255.255.255.0'' 应该可以
- 其他设备-静态 "10.TE.AM.6-.10''或 ".12-.19''(如果没有摄像头,则为.11)子网 "255.255.255.0''

41.2.3 Pits 区内

备注： ** 2018 年新增功能：**赛事配置中，机器人无线电的有线端现在运行着一个 DHCP 服务器。

Pits 区内 DHCP 配置

- OpenMesh radio - Static `10.TE.AM.1` programmed by Kiosk.
- roboRIO - `10.TE.AM.2`, assigned by Robot Radio
- Driver Station - DHCP ("Obtain an IP address automatically"), `10.TE.AM.X`, assigned by Robot Radio
- IP camera (if used) - DHCP, `10.TE.AM.Y`, assigned by Robot Radio
- Other devices (if used) - DHCP, `10.TE.AM.Z`, assigned by Robot Radio

Pits 区内静态配置

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 roboRIO 网络故障排除

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 使用 mDNS 检查 roboRIO

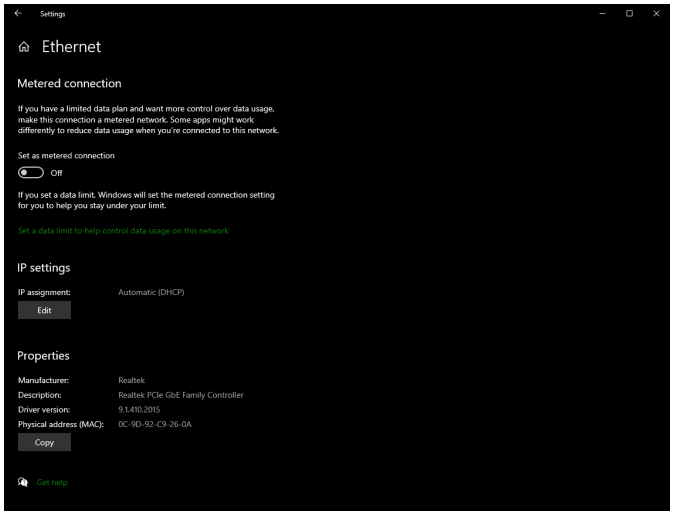
识别 roboRIO 网络问题的第一步是隔离如果其是应用程序问题或常规网络问题的话。为此，请单击 ** 开始-> 输入 cmd-> 按 Enter **以打开命令提示符。输入 "ping roboRIO-####-FRC.local"，其中 #### 是您的团队编号（没有前导零），然后按 Enter 键。如果 ping 操作成功，则可能是特定应用程序出现了问题，请在应用程序中验证您的团队编号配置，然后检查防火墙配置。

41.3.2 ping roboRIO IP 的地址

如果没有响应，请尝试使用上述命令提示符对 "10.TE.AM.2' ' (: ref: *TE.AM IP 符号 <docs/networking/networking-introduction/ip-configurations:TE.AM IP Notation>*) 进行 ping 操作。如果这行得通，则说明您在 PC 上解析 mDNS 地址时遇到问题。两种最常见的原因是系统上没有安装 mDNS 解析器，而网络上没有试图使用常规 DNS 解析.local 地址的 DNS 服务器。

- 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*.
- 使 用: ref: *FRC 无 线 电 配 置 实 用 程 序 <docs/zero-to-robot/step-3/radio-programming:Programming your Radio>*，将您的计算机与其他任何网络断开连接，并确保已将 OM5P-AN 配置为访问点。从系统中删除任何其他路由器将有助于验证是否没有 DNS 服务器引起此问题。

41.3.3 平失败



如果直接 ping IP 地址失败，则 PC 的网络配置可能有问题。PC 应该配置为“自动”。要检查这一点，请单击“开始” -> “设置” -> “网络和 Internet”。根据您的网络，选择“Wifi”或“以太网”。然后单击您所连接的网络。向下滚动到“IP 设置”，然后单击“编辑”，并确保选择了“自动（DHCP）”选项。

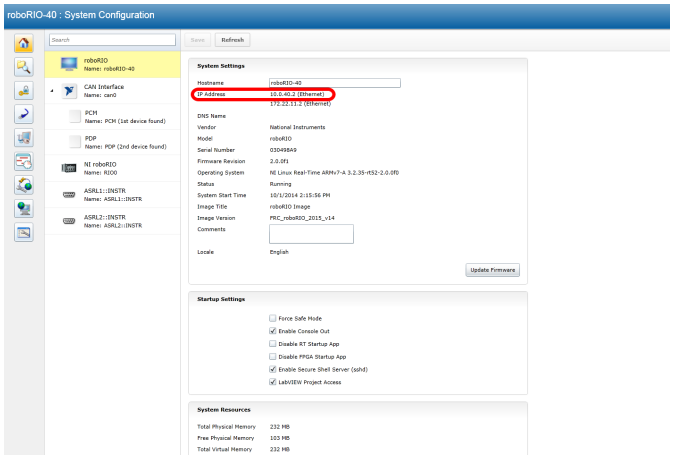
41.3.4 USB 连接故障排除

如果您尝试对 USB 连接进行故障排除，请尝试 ping roboRIO 的 IP 地址。只要仅将一个 roboRIO 连接到 PC，就应将其配置为 172.22.11.2。如果无法 ping 通，请确保已连接 roboRIO 并为其供电，并且已安装 NI FRC 游戏工具。游戏工具会安装 USB 连接所需的 roboRIO 驱动程序。

如果此 ping 成功，但.local ping 失败，则可能是 roboRIO 主机名配置错误，或者您已连接到尝试解析.local 地址的 DNS 服务器。

- Verify that your roboRIO has been imaged for your team number: *roboRIO 1 roboRIO 2*. This sets the hostname used by mDNS.
- : ref: 禁用所有其他网络适配器 *<docs/networking/networking-introduction/roborio-network-troubleshooting:Disabling Network Adapters>*

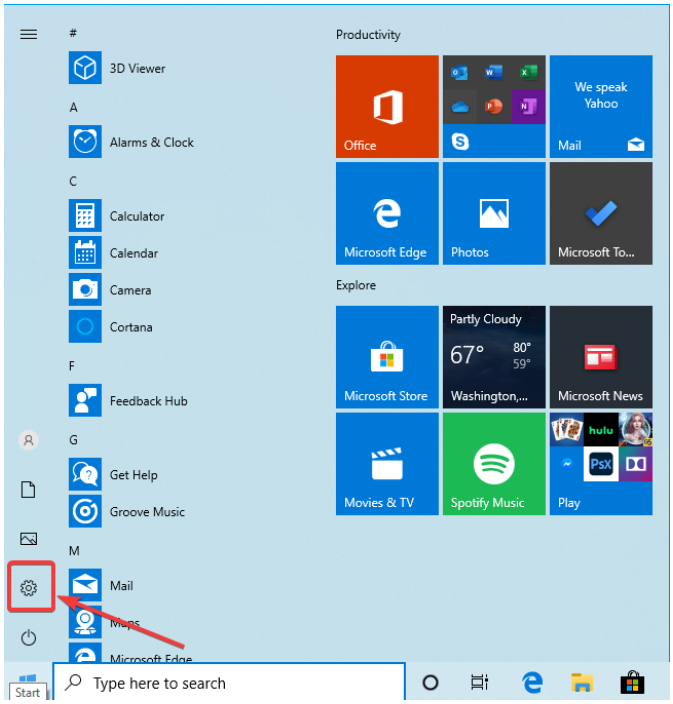
41.3.5 以太网连接



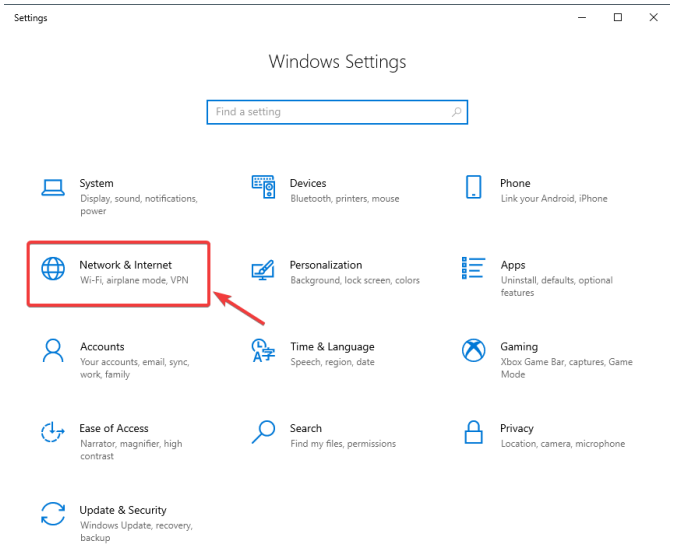
如果要对以太网连接进行故障排除，首先确保你可以使用 USB 连接连接到 roboRIO 可能会有所帮助。使用 USB 连接，打开 roboRIO Web 面板 <docs/software/roborio-info/roborio-web-dashboard:roboRIO Web Dashboard>，并验证 roboRIO 在以太网接口上是否具有 IP 地址。如果要直接绑定到 roboRIO，则应为自定义的 “169。。。*’’ 地址，如果已连接到 OM5P-AN 无线电，则应为 “10’’ 形式的地址.TE.AM.XX“，其中 TEAM 是您的四位数 FRC 团队编号。如果此处唯一的 IP 地址是 USB 地址，请验证物理 roboRIO 以太网连接。

41.3.6 禁用网络适配器

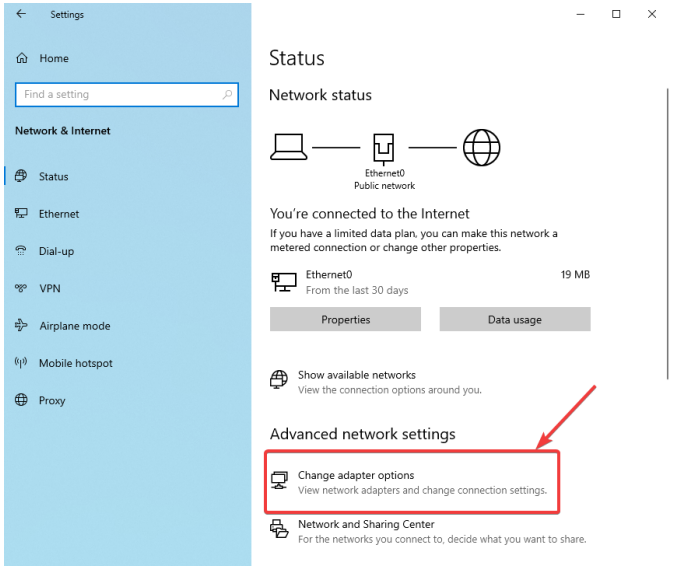
这与通过物理按钮关闭适配器或使 PC 进入飞行模式并不总是相同。以下步骤提供有关如何禁用适配器的更多详细信息。



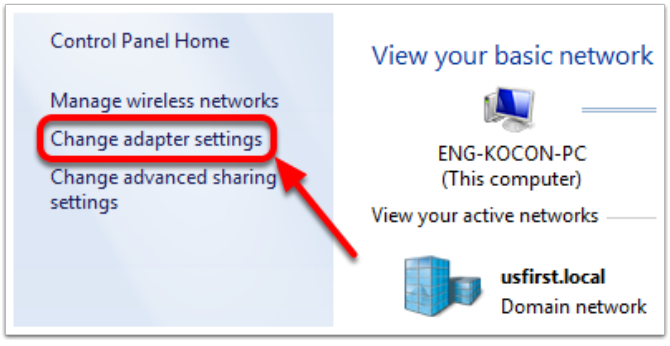
通过单击设置图标打开设置应用程序。



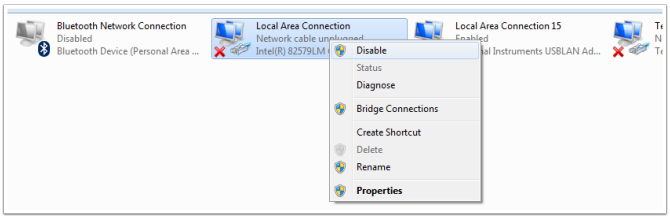
选择“网络和 Internet”类别。



单击“更改适配器选项”。



在左窗格上，单击“更改适配器设置”。



对于除与无线电连接的适配器以外的每个适配器，请右键单击适配器，然后从菜单中选择“禁用”。

41.3.7 代理

- 代理。启用代理可能会导致 roboRIO 网络出现问题。

41.4 Windows 防火墙配置

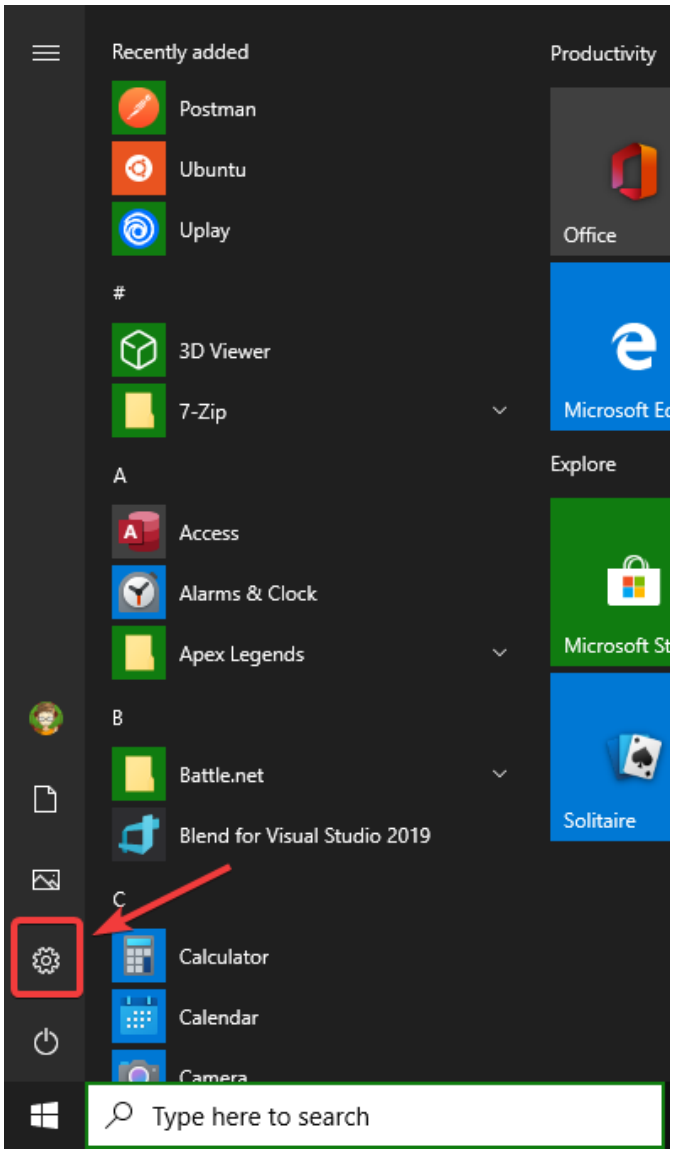
FRC | reg |中使用的许多编程工具由于各种原因需要网络访问。根据确切的配置，Windows 防火墙可能会干扰其中一个或多个程序的访问。

41.4.1 禁用 Windows 防火墙

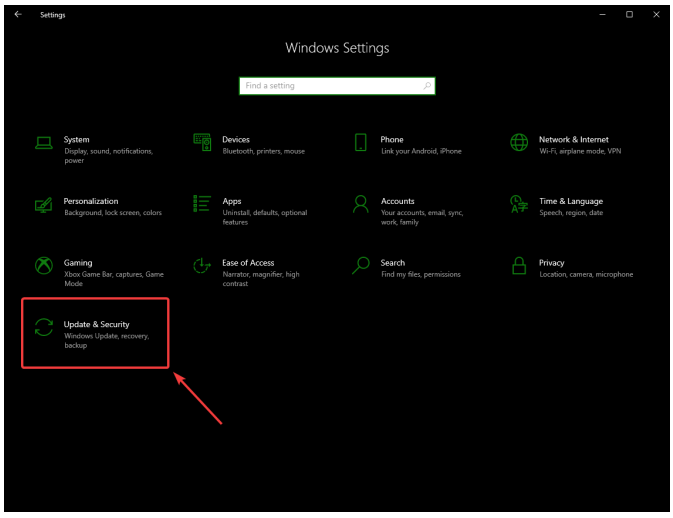
重要： 禁用防火墙需要具有 PC 管理员权限。另外请注意，对于连接到 Internet 的计算机，建议不要禁用防火墙。

最简单的解决方案是禁用 Windows 防火墙。团队应注意，如果连接到 Internet，这确实会使 PC 可能更容易受到恶意软件攻击。

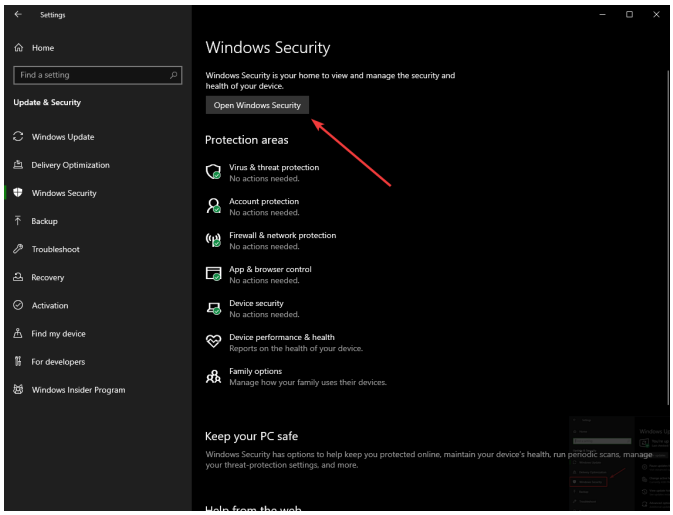
点击 *Start -> Settings*



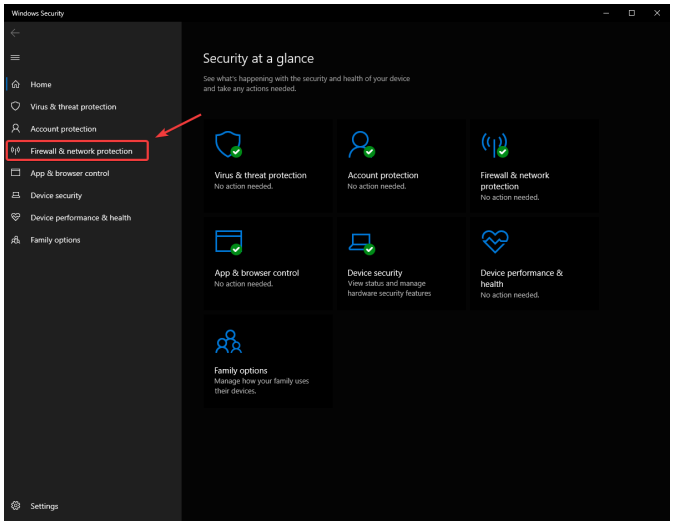
点击 *Update & Security*



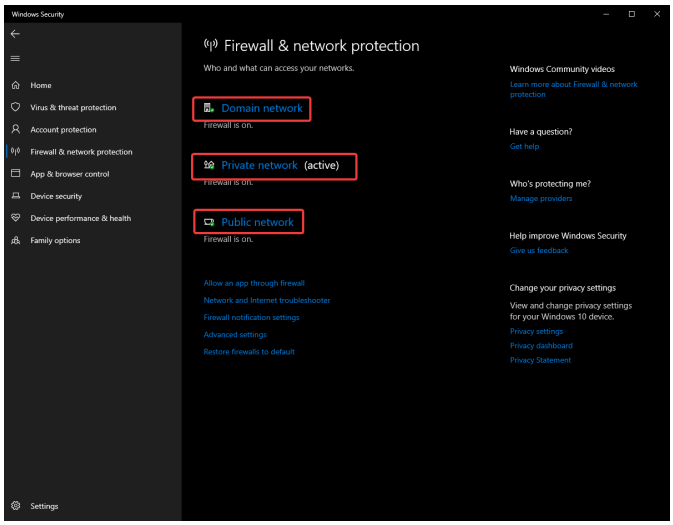
在右边界面选择:guilabel:*Open Windows Security*



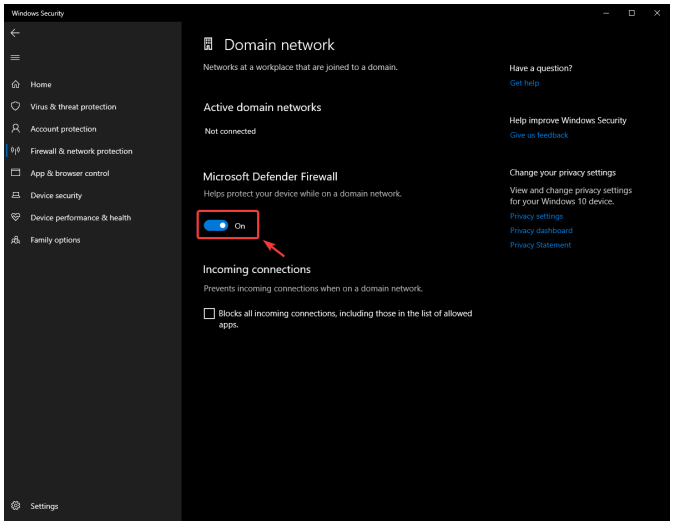
左边界面选择:guilabel:*Firewall and network protection*



单击突出显示的 ** 每个 ** 选项



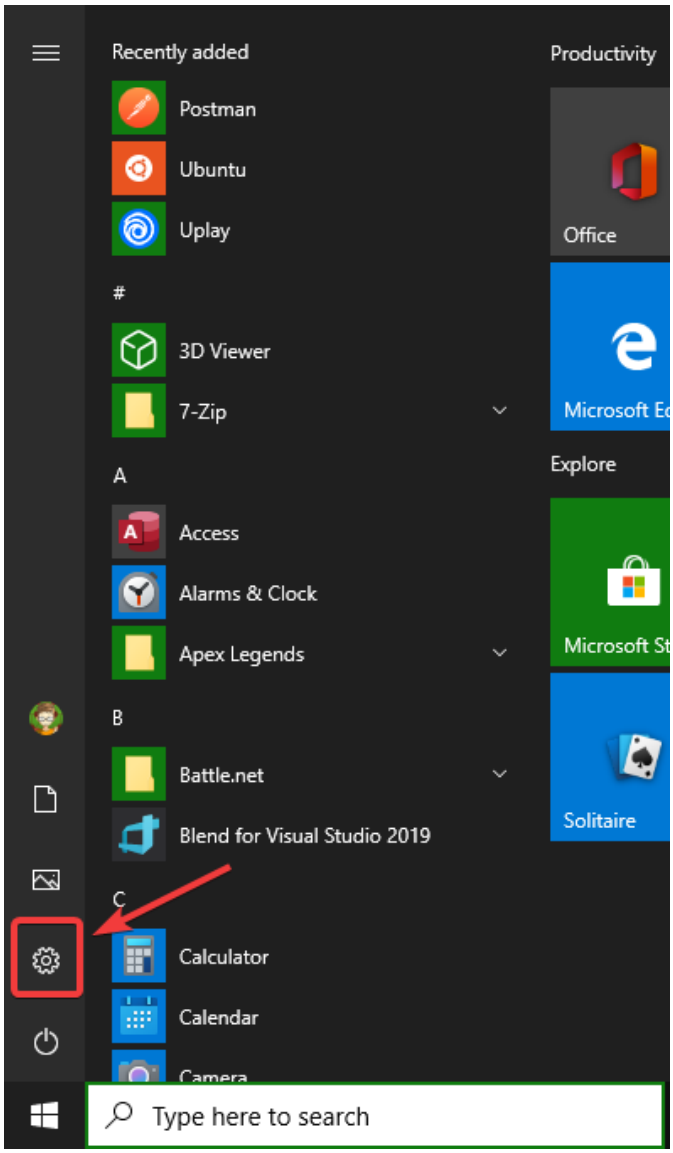
然后单击 ** On **开关将其关闭。



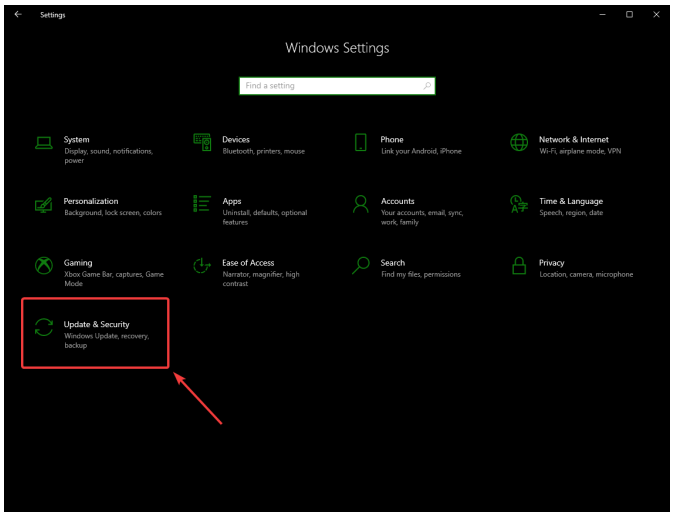
41.4.2 白名单程序

或者，您可以为遇到问题的任何 FRC 程序向防火墙添加例外。

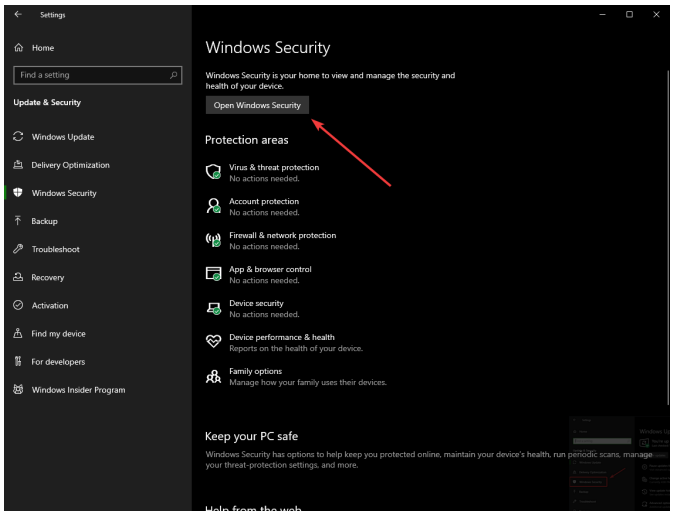
点击 *Start -> Settings*



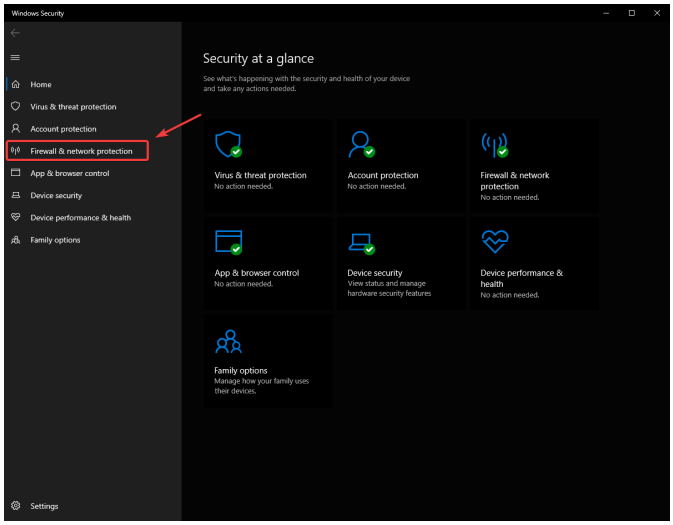
点击 *Update & Security*



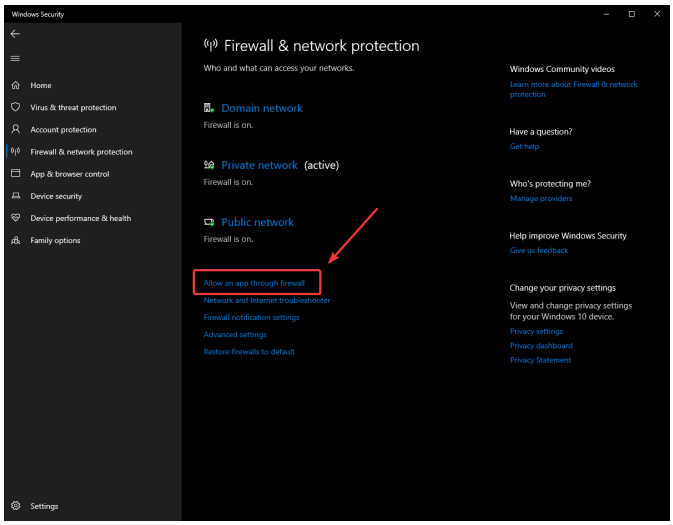
在右边界面选择:guilabel:*Open Windows Security*



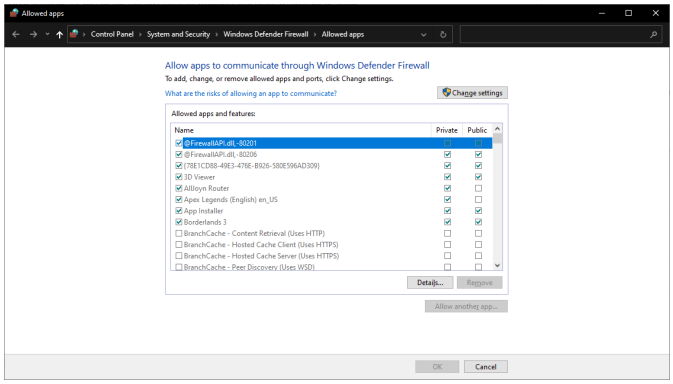
左边界面选择:guilabel:*Firewall and network protection*



窗口下方，选择:guilabel:*Allow an app through firewall*



对于您遇到问题的每个 FRC 程序，请确保它出现在列表中，并且在 3 列的每一列中都有一个选中标记。如果您需要更改设置，则需要先单击右上角的 *Change settings* 按钮，然后再更改设置。如果该程序根本不在列表中，请单击:guilabel:‘Allow another program…‘按钮，然后浏览至该程序的位置以将其添加。



41.5 测量带宽使用

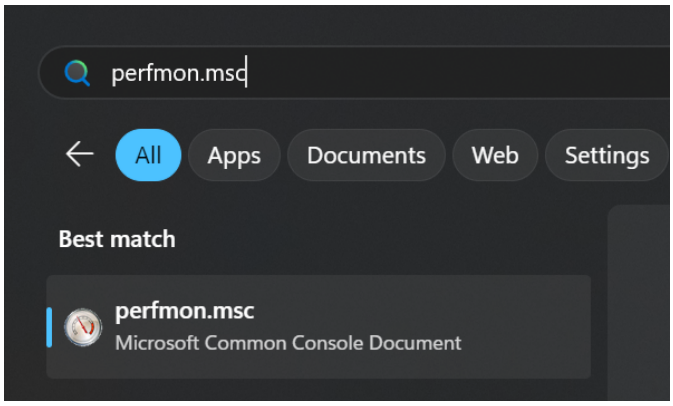
On the FRC® Field each team is allocated limited network bandwidth (see R704 in the 2024 manual). The FMS Whitepaper provides more information on determining the bandwidth usage of the Axis camera, but some teams may wish to measure their overall bandwidth consumption. This document details how to make that measurement.

备注： 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)

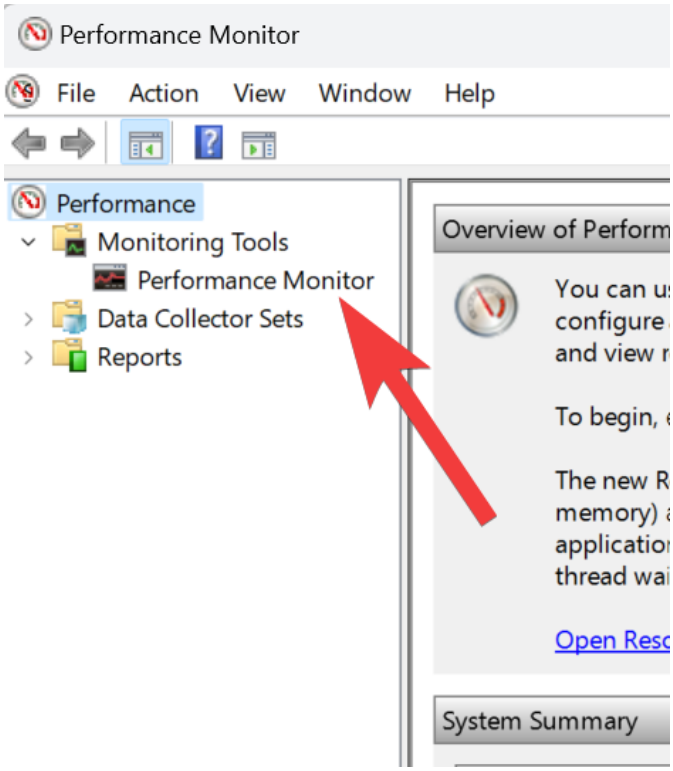
Windows contains a built-in tool called the Performance Monitor that can be used to monitor the bandwidth usage over a network interface.

运行性能监视器



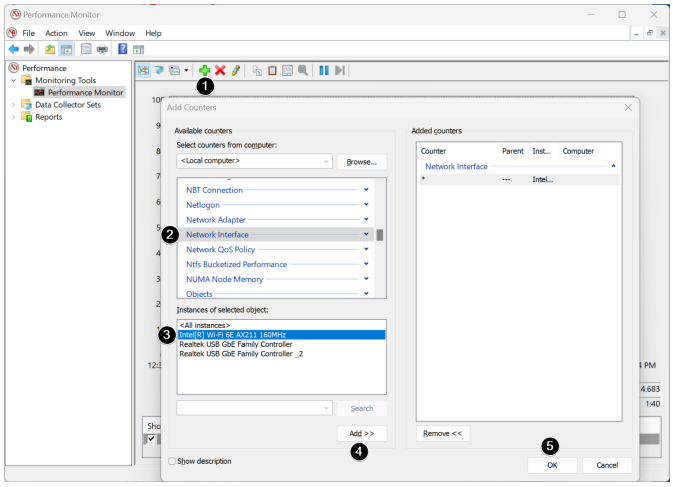
Open the Start menu and in the search box, type `perfmon.msc` and press Enter.

打开实时监控器



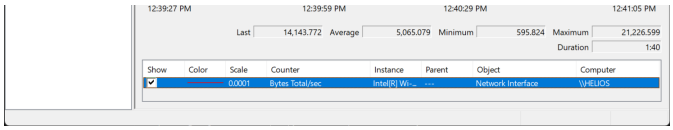
In the left pane, click *Performance Monitor* to display the real-time monitor.

添加网络计数器



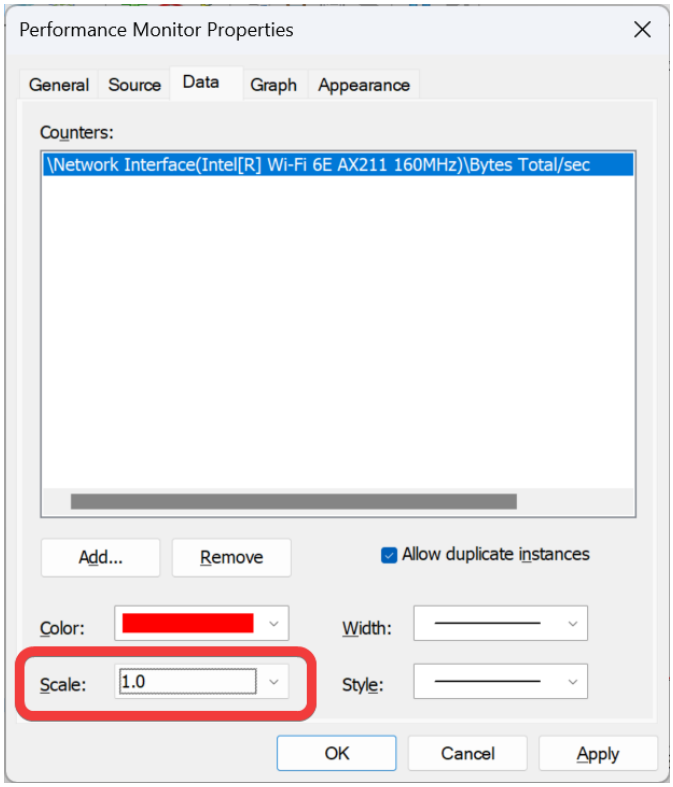
1. 点击屏幕顶部附近的绿色加号以添加一个计数器
2. 在左上窗格中，找到并单击“Network Interface”（网络接口）以选中它
3. 在左下方窗格中，找到所需的网络接口（或使用“所有实例”监视所有接口）
4. Click *Add »* to add the counter to the right pane.
5. Click *OK* to add the counters to the graph.

删除额外计数器



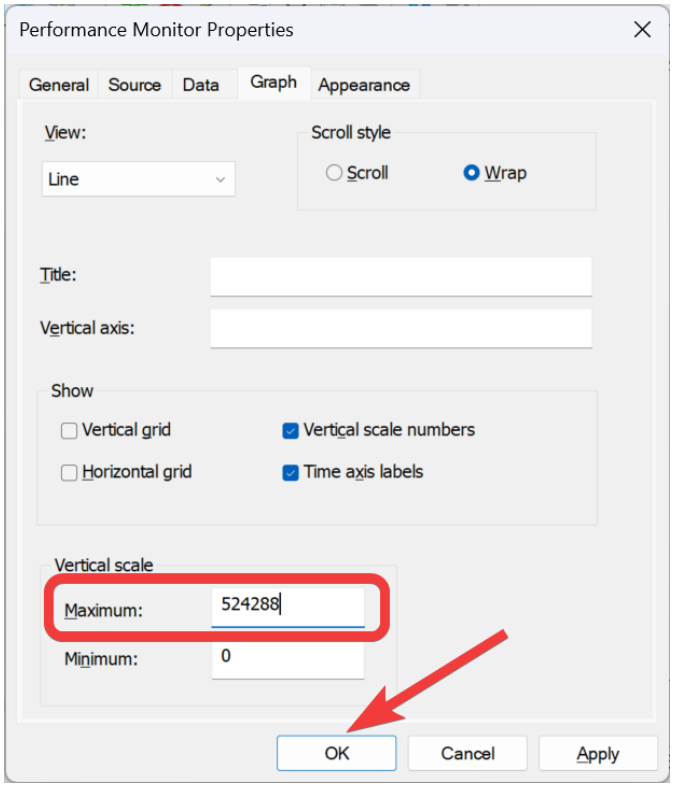
In the bottom pane, select each counter other than `Bytes Total/sec` and press the `Delete` key. The `Bytes Total/sec` entry should be the only entry remaining in the pane.

配置数据属性



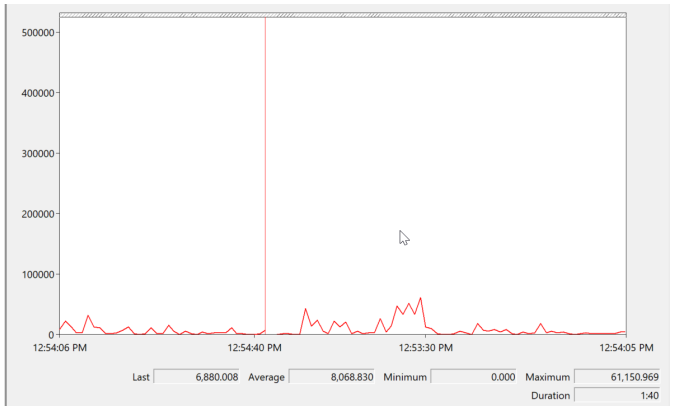
Press `Ctrl+Q` to bring up the Properties window. Click on the dropdown next to `Scale` and select `1.0`. Then click on the *Graph* tab.

配置图形特征



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.

查看带宽使用情况

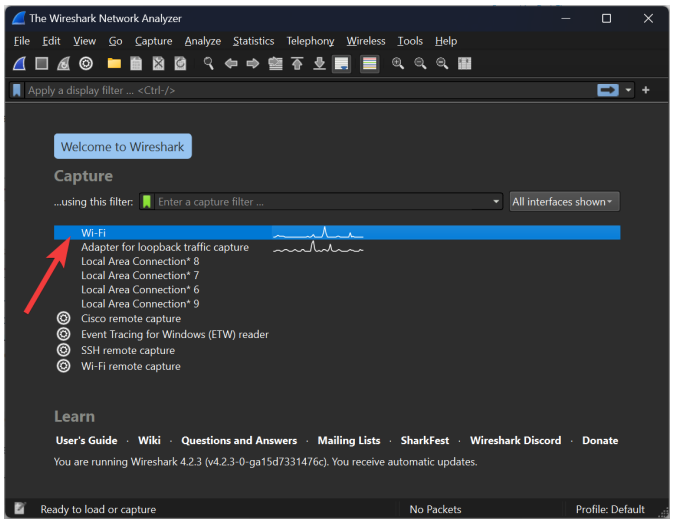


You may now connect to your robot as normal over the selected interface (if you haven't done so already). The graph will show the total bandwidth usage of the connection, with the bandwidth cap at the top of the graph. The Last, Average, Min and Max values are also displayed at the bottom of the graph. 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 使用 Wireshark 测量带宽使用情况

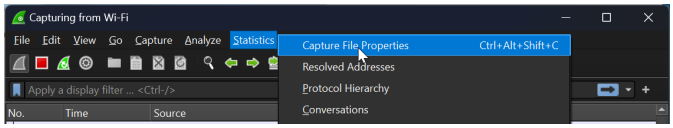
If you can not use performance monitor, you will need to install a 3rd party program to monitor bandwidth usage. One program that can be used for this purpose is Wireshark. Download and install the latest version of Wireshark for your version of Windows from this page. After installation is complete, locate and open Wireshark. Connect your computer to your robot, open the Driver Station and any Dashboard or custom programs you may be using.

选择界面并开始捕获



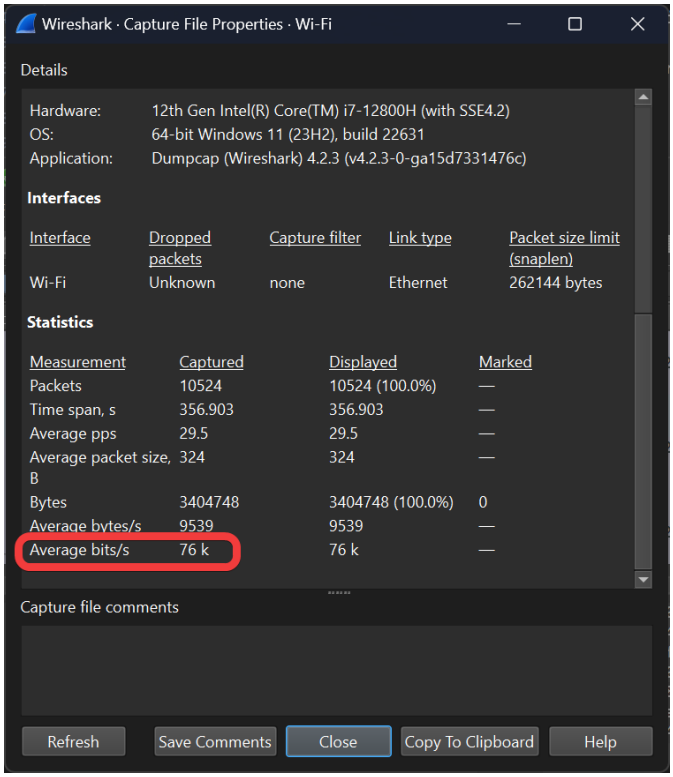
In the Wireshark program on the left side, double click the interface you are using to connect to the robot.

Open Capture File Properties



Let the capture run for at least 1 minute, then click *Statistics* then *Capture File Properties*.

查看带宽使用情况



Average bandwidth usage, in bits/second is displayed near the bottom of the window.

41.6 修改 OM5P-AC 路由器

OM5P-AC 路由器可能会在 FRC |reg| 环境中遭受不确定的冲击和震动。如果路由器在外壳底层受到很大的压力，则可能通过将路由器底部的金属层接触到电路板底部的一些裸露的金属引线而短路来导致路由器重新启动。本文详细介绍了如何修改路由器，以防止出现这种情况。

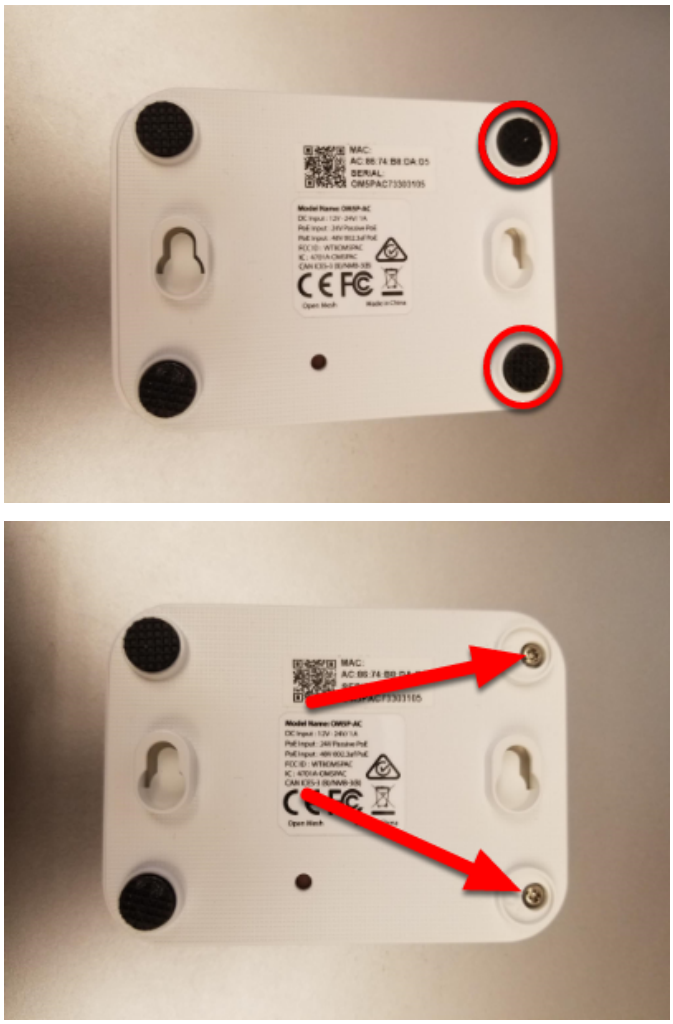
警告： 路由器会在外壳底部受到很大的压力时才会以这种方式重新启动。大多数 FRC 路由器重新启动问题可以以某种形式追溯到电源路径。我们建议通过策略性地固定路由器设备来减轻这种风险，而不是打开和改装路由器（并且这存在损坏精密内部组件的风险）：

- 避免使用路由器底部的“固定片”功能。
- 您可能希望在安装路由器的位置能吸收一些震动。用钩环紧固件将路由器安装到机器人上，或者用少量弯曲（塑料或金属薄板等）将机器人安装在机器人表面上，可能会大大减少路由器所承受的外力干涉。

41.6.1 打开路由器

备注： OpenMesh OM5P-AC 并非设计为用户可维修的设备。用户自行承担执行此修改的责任。确保缓慢而谨慎地工作，以免损坏内部组件，例如路由器天线电线。

壳体螺丝



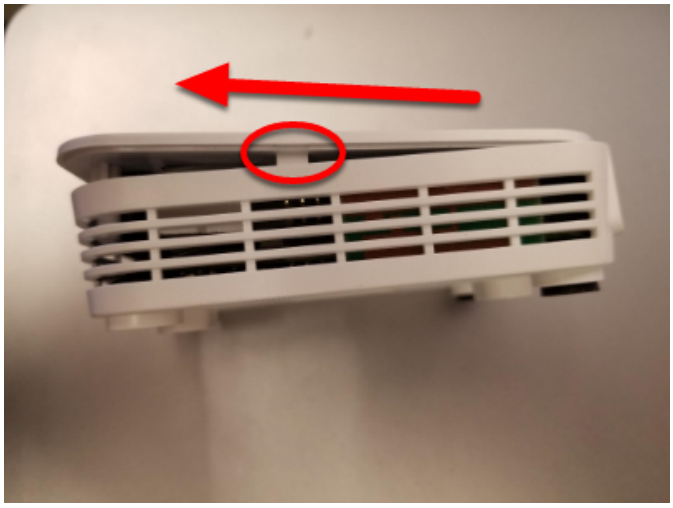
找到路由器正面的两个橡胶支脚，然后使用指甲，小平头螺丝刀等将其撬出路由器。使用小的十字螺丝刀，卸下支脚下面的两个螺钉。

侧面锁

路由器盖子上每个长边的中间附近都有一个小闩锁（您可以在下一张照片中更清楚地看到这些闩锁）。使用指甲或非常薄的工具，沿着盖子和外壳之间的间隙从前向后朝路由器的中央滑动，当您靠近路由器的中央时，应该会听到一声小声音。在另一侧重复（注意：这样做时不难将第一侧重新锁上，在继续操作之前，请确保两侧均未上锁）。如上图所示，路由器盖子现在应在前侧稍微打开。

取下盖子

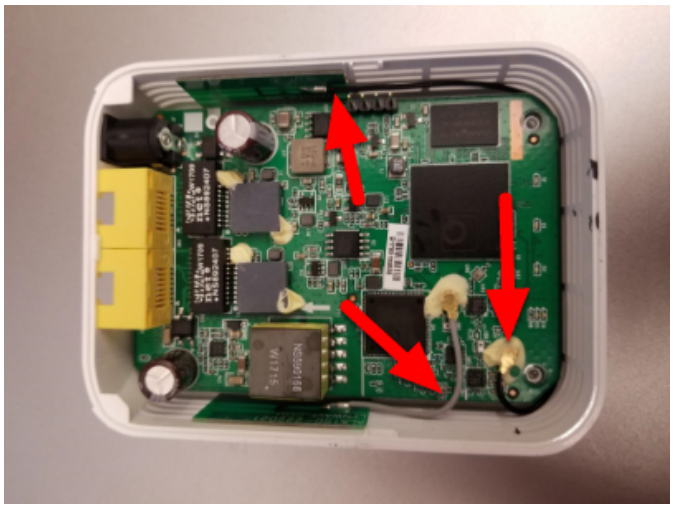
警告： 由于散热垫的缘故，在卸下面板时，面板可能会粘在盖子上。卸下盖子时，请看一下收音机的通风孔，以查看电路板是否随其一起提起来了；如果是，则可能需要插入一个小的工具以将电路板压下，以将其与盖子分离。我们建议使用一个小螺丝刀或类似的工具，该螺丝刀可穿过通气孔，并通过枪管插孔侧的前角，就在螺丝孔上方。您可以向下滚动至图片，并取下盖子，以查看该区域板的外观。



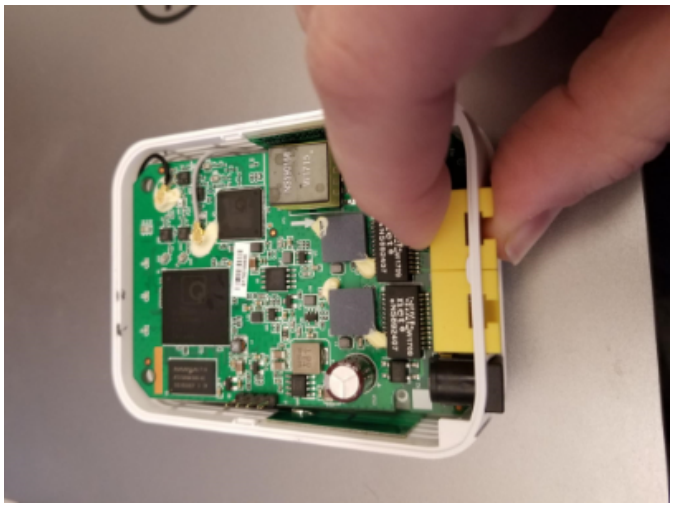
要开始取下盖子，请将其向前滑动（略微抬起），直到螺钉固定器碰到机箱正面为止（这样做时，您可能需要在闩锁区域上多加力气）。

接下来，如图所示，在继续提起的同时，将盖子从公头侧稍微旋转一点。这样可以将盖子从右上角可见的小三角形上解开。继续朝此方向稍微旋转，同时将左上角推向公头（在此步骤中请勿尝试进一步提起），以在左上角解开类似的功能。然后将盖子完全抬离身体。

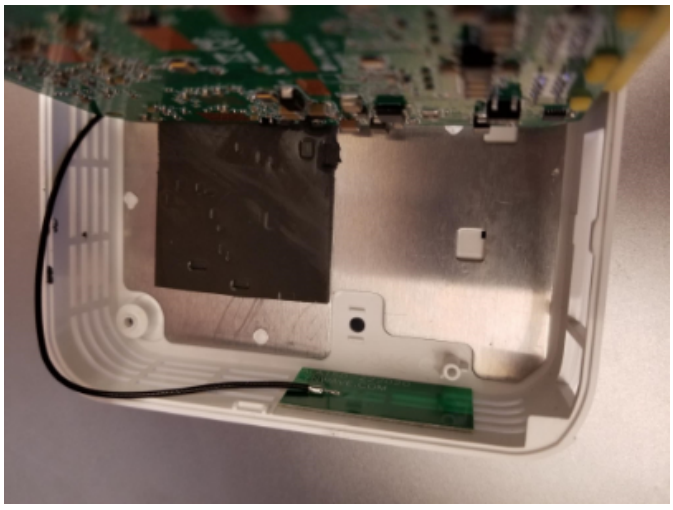
移除板子



警告： 注意上图所示的天线线。这些电线及其连接器易碎，执行后续步骤时请注意不要损坏它们。



要卸下板，我们建议您用手指抓住一个或两个网络端口（如图所示），然后向内（朝无线电设备的前部）推入并向上推，直到网络端口和机筒插孔脱离机壳。



将电路板向上倾斜（朝向短的灰色天线电缆），露出下面的金属层。

备注： 当您执行此步骤时，您可能会注意到板的底侧有一个小的复位按钮，该按钮大于机壳上的孔。请注意，在安装了 FRC 固件的情况下，按下重置按钮无效，并且不允许修改路由器的外壳。

41.6.2 贴胶带

在网络端口/公头开口内部的区域中，在金属层上缠上一条胶带。这样可以防止电路板下面的裸露引线在该板上短路。

41.6.3 重新组装路由器

按照将其打开的说明的相反步骤重新组装路由器：

- 将板放回原处，确保其与前部附近的螺孔对齐并牢固就位
- 从右向左移动盖子，将盖子滑到左后固定功能部件上。照顾这个区域的电容器
- 旋转盖子，向下按，然后向右滑动后固定住
- 用力向下按盖子的前部/中部以固定闩锁
- 装回前脚中的 2 个螺钉
- 装回前脚

42 网络工具

42.1 端口转发

此类提供了一种将本地端口转发到另一个主机/端口的简便方法。这给了连接到 roboRIO USB 端口的计算机一个访问以太网连接设备的方法。此类充当原始 TCP 端口转发器，这意味着您可以转发诸如 SSH 之类的连接。

42.1.1 转发远程端口

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)
```

重要： 您 ** 不能 ** 使用小于 1024 的端口作为本地转发端口。同样重要的是要注意，您不能使用完整的 URL（http://wpilibpi.local），而只能使用 IP 地址或 DNS 名称。

42.1.2 移除转发的端口

要停止在指定端口上的转发，只需调用 “remove（int port）”，port 为端口号，即可。如果在未转发的端口上调用 “remove（）“，则不会发生任何事情。

JAVA

```
@Override
public void robotInit() {
   PortForwarder.remove(8888);
}
```

C++

```
void Robot::RobotInit {
   wpi::PortForwarder::GetInstance().Remove(8888);
}
```

PYTHON

```
wpinet.PortForwarder.getInstance().remove(8888)
```

43 为 frc-docs 做出贡献

43.1 文档贡献规则

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.

重要： *FIRST*® 保留被提供文档和图像的所有的权利。文章 / 更新的积分将记录在 GitHub commit history.

43.1.1 使命宣言

WPILib 的任务是让 *FIRST* 机器人团队专注于编写游戏专用软件，而不是专注于硬件细节—— “raise the floor, don’t lower the ceiling”。我们致力于使具有有限编程知识和/或指导经验的团队尽可能成功，同时不妨碍具有更高级编程能力的团队的能力。我们在库中直接支持零件控制系统组件套件。我们还努力保持每个语言（java，C++，和 NI 的 LabVIEW）的主要特征之间的奇偶性，这样团队就不会因为选择特定的编程语言而处于不利的地位。

这些文档为所有 *FIRST* 机器人竞赛团队提供了一个学习场所。对该项目的捐赠必须遵循这些核心原则。

- 社区主导的文档。文档来源是公开托管的，社区可以参与捐赠
- 结构化、格式好、清晰的文档。从源代码和发布的角度来看，文档应该是干净且易于阅读的
- 息息相关。文档应该集中注意在 * FIRST *机器人大赛上。

查阅:ref:‘docs/contributing/frc-docs/style-guide:Style Guide‘了解如何规范化您的文档。

43.1.2 发布过程

frc-docs 使用特殊的发行过程来处理主站点 “/ stable /’’ 和开发站点 “/ latest /’’。该流程将在下面详细说明。

During Season:

- Commit made to `main` branch
  - 更新网站上的 `/stable/`` 和/latest/“

End of Season:

- Repository is tagged with year, for archival purposes

Off-Season:

- `stable` branch is locked to the last on-season commit
- Commit made to `main` branch
  - 指更新文档站点中的 “/latest/“

43.1.3 创建一个 PR

应该在 GitHub 上的 ‘frc-docs <https://github.com/wpilibsuite/frc-docs>‘__ 仓库中进行 PR。它们应指向 “main” 分支，并指向 “not” “stable”。

43.1.4 增加新内容

感谢您为 ‘frc-docs <https://github.com/wpilibsuite/frc-docs>‘__ 项目做出贡献！开始之前，希望您了解下几件事！

文章放在哪呢?

新文章的位置可能是一个很主观的问题。属于现存主题分类的独立文章应放在提及的主题类别中（有关模拟的文档应放在模拟部分中）。但是，当文章结合或引用两个单独的现有部分时，事情可能会变得非常复杂。在这种情况下，我们建议作者在打开 PR 之前先在存储库中打开一个 repository，以进行讨论。

备注: 所有新文章在合并到 repository 之前都将经过审核过程。审核过程将由 WPILib 团队的成员来完成。新文章必须在受 * FIRST *支持的官方软件和硬件上。关于非官方图书馆或传感器的文件不会被接受。该过程可能需要一些时间进行审核，请耐心等待。

在哪里放置章节?

章节是相当棘手的，因为它们包含大量的内容。我们建议作者在打开 PR 之前先打开以 “issue <https://github.com/wpilibsuite/frc-docs/issues> ‘__ 来聚集讨论。

链接其他文章

在该文章引用另一篇文章中描述的内容的情况下，作者应尽最大努力在第一次引用时链接到该文章。

想象一下，在传动系统教程中我们具有以下内容：

```
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.
```

请注意如何只链接 Simulation 的第一个实例，这是作者应遵循的结构。有时链接文章的内容主题不同。如果您引用了本文中不同类型的内容，则应链接到每个新引用一次（除非作者认为其他情况适当）。

43.2 格式指南

本文档包含针对 frc-docs 项目的各种 RST / Sphinx 特定准则。有关与各种 WPILib 代码项目相关的指南，请参见 ‘the WPILib GitHub <https://github.com/wpilibsuite/styleguide>‘__

43.2.1 文件名

仅使用小写字母数字字符和 “- ” （减号）。

对于具有相同软件/硬件名称的文档，请在文档名称的末尾附加 “硬件” 或 “软件” 。IE，“超声波-hardware.rst’’

后缀名为.rst 的文件名。

备注: If you are having issues editing files with the `.rst` extension, the recommended text editor is VS Code with the rST extension.

43.2.2 文本

All text content should be on the same line. If you need readability, use the word-wrap function of your editor.

对于这些术语，请依据以下情况：

- roboRIO (不是 RoboRIO、roboRio 或 RoboRio)
- LabVIEW (不是 labview 或 LabView)
- Visual Studio Code (VS Code) (而非 vscode、VScode、vs code, etc)
- macOS (而非 Mac OS, Mac OSX, Mac OS X, Mac, Mac OS, etc.)
- GitHub (不是 github, Github, etc)

- PowerShell (不可用 powershell, Powershell, etc)
- Linux (不可用 linux)
- Java (不可用 java)

将 ASCII 字符集用于英语文本。对于特殊字符（例如希腊符号），请使用标准字符实体集 ‘standard character entity sets <https://docutils.sourceforge.io/docs/ref/rst/definitions.html#character-entity-sets>‘_。

对单个方程式使用” .. math ::”，对于行内方程式使用’ ’ :math:’ ’。可以在 ‘here <https://www.reed.edu/academic_support/pdfs/qskills/latexcheatsheet.pdf>‘_ 找到有用的 LaTeX 公式备忘单。

对于文件名，函数和变量名，请使用文字。

注册商标的使用 * FIRST * **|reg| 和 FRC \ |reg| 应遵循 ‘this page <https://www.firstinspires.org/brand>‘__ 中的政策。具体而言，在可能的情况下（即不嵌套在其他标记中或文档标题中），商标的首次使用应使用 |reg|** 符号，并且 *FIRST* 的所有实例均应以斜体显示。|reg| 可以通过在文档顶部使用 “.. include:: <isonum.txt>“ ，然后使用 “*FIRST*|reg|“ 或“FRC|reg|“来添加符号。

常用术语应添加到 docs / software / frc-glossary：FRC 术语表中。您可以通过使用：term：‘deprecated‘来引用词汇表中的项目。

43.2.3 空白

缩进

缩进应该 * 始终 * 匹配先前的缩进级别 * 除非 * 正在创建新的内容块。

内容指令的缩进作为新行 “.. toctree ::“应该是 3 个空格。

空行

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.

内部空白

段落中使用一个空格。

43.2.4 标题

标题应采用以下结构。标题下划线应与标题本身匹配相同数量的字符。

1. “=“用于文档标题。每篇文章的使用次数 * 不得 * 超过 * 一次 *。
2. “- ° 用于文档章节
3. “^“用于文档子章节
4. “~” 代表文档子子章节
5. 如果您需要使用任何较低级别的结构，那么您应该有些事情出错了。

标题使用标题大小写字体。

43.2.5 列表

列表在每个缩进级别之间应有一个新行。最高的缩进应具有“0’’缩进，并且随后的子列表应具有从前一个缩进的第一个字符开始的缩进。

```
- Block one
- Block two
- Block three

  - Sub 1
  - Sub 2

- Block four
```

43.2.6 代码块

所有代码块都应指定一种编程语言。

1. 例外：必须保留格式且没有编程语言的内容。使用“text’’代替。

对于 C ++ 和 Java 示例代码，请遵循 ‘WPILib style guide <https://github.com/wpilibsuite/styleguide/>‘_ 。例如，在 C ++ 和 Java 中使用两个空格进行缩进。

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 警告

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.

用途

```
.. warning:: This is a warning!
```

不

```
.. warning::
   This is a warning!
```

43.2.10 链接

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 *Analog ultrasonics*.

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 *Build Instructions* 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 *Ultrasonics - 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!
```

但是，在某些必须多次引用同一链接情况下，则接受以下语法。

```
Hi there, `this is a link`_ and it's pretty cool!

..  _this is a link: https://example.com
```

43.2.11 图片

图像应使用“1”新行，将内容和指令分隔开。

所有图像（包括矢量）的大小应小于“500’’ kb 字节。请使用较小的分辨率和更有效的压缩算法。

```
.. image:: images/my-article/my-image.png
   :alt: Always add alt text here describing the image.
```

图片文档

Image files should be stored in the document directory, sub-directory of `document-name/images`.

他们应遵循“short-description.png’’ 的命名方案，其中图像名称是对图像显示内容的简短描述。该字符应少于“24’’ 个字符。

它们应该是.png 或.jpg 图片扩展名。出于存储和可访问性的考虑，.gif 是不可接受的。

备注： 可访问性很重要！图像应该用 “:alt:“指令标记。

```
.. image:: images/my-document/my-image.png
   :alt: An example image
```

矢量图

SVG 文件通过 Sphinx 扩展 “svg2pdfconverter“支持。

就像使用其他任何图像一样，只需简单地使用它们即可。

备注： 确保矢量中的任何嵌入图像都不会使矢量膨胀到超过 500KB 的限制。

```
.. image:: images/my-document/my-image.svg
   :alt: Always add alt text here describing the image.
```

Draw.io 图

Draw.io（也称为 ‘diagrams.net <https://app.diagrams.net/>‘__）图通过嵌入了.drawio 元数据的“svg“文件来支持，从而允许 “svg“文件充当图的源文件，并像普通的矢量图形文件一样呈现。

只需像使用任何其他矢量图像或任何其他图像一样使用它们。

```
.. image:: diagrams/my-document/diagram-1.drawio.svg
   :alt: Always add alt text here describing the image.
```

Draw.io 文件

Draw.io 文件遵循与普通图像几乎相同的命名方案。为了跟踪嵌入了.drawio 元数据的文件，请在扩展名之前的文件名末尾附加.drawio，这意味着文件名应为“document-title-1.drawio.svg“等。此外，图表应存储在文档目录中名为 “diagrams’’ 的子文件夹中。

有关将图另存为带有元数据的.svg 的细节，请查看:ref:*docs/contributing/frc-docs/drawio-saving-instructions:Draw.io Saving Instructions*。

> **警告：** 确保不修改 “diagrams” 文件夹中的任何文件或在 draw.io 以外的任何程序中以.drawio.svg 结尾的文件，否则可能会破坏文件的元数据，使其无法编辑。

43.2.12 文档扩展名

文件扩展名应使用代码格式。例如使用：

```
``.png``
```

代替：

```
.png
".png"
"``.png``"
```

43.2.13 目录（TOC）

每个类别都应包含一个 “index.rst”。该索引文件应包含 “1’’ 的 “maxdepth’’。子类别是可接受的，只要 “maxdepth’’ 为 1。

The category `index.rst` file can then be added to the root index file located at `source/index.rst`.

43.2.14 示例

```
Title
=====
This is an example article

.. code-block:: java

   System.out.println("Hello World");

Section
-------
This is a section!
```

43.2.15 务必留意!

此列表并不详尽，管理员保留进行更改的权利。更改将反映在本文档中。

43.3 Build Instructions

本文档包含有关如何构建 frc-docs 网站的 HTML，PDF 和 EPUB 版本的信息。frc-docs 使用 Sphinx 作为文档生成器。本文档还假定您具有 Git 和控制台命令的基本知识。

43.3.1 Prerequisites

确保已安装 Git，并使用 `git clone https://github.com/wpilibsuite/frc-docs.git`. 克隆了 frc-docs 存储库。

Text Editors / IDE

For development, we recommend that you use VS Code along with the reStructuredText extension. However, any text editor will work.

By default, the reStructuredText extension enables linting with all doc8 features enabled. As frc-docs does not follow the line length lint, add the following to your VS Code `settings.json` to disable line length linting.

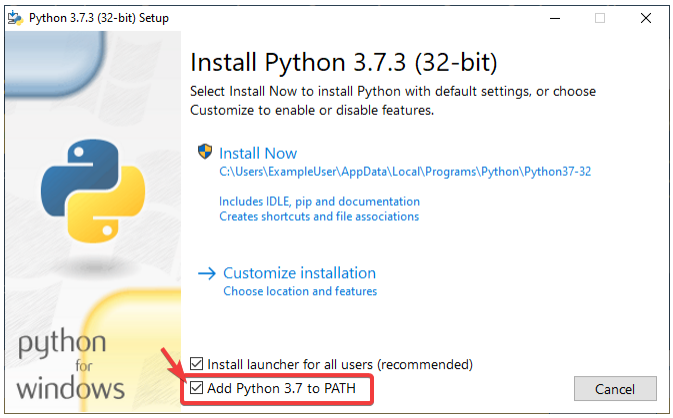
```
"restructuredtext.linter.doc8.extraArgs": [
   "--ignore D001"
]
```

Windows

备注： MikTeX and `rsvg-convert` are not required for building HTML, they are only required for Windows PDF builds.

- Python 3.9
- Perl
- MiKTeX (Only needed for PDF builds)
- rsvg-convert (Only needed for PDF builds)

通过在安装 Python 时选择开关 **Add Python to PATH** 确保 Python 在您的路径中。



安装 Python 之后，打开 Powershell。然后导航到 frc-docs 目录。运行以指令：`pip install -r source/requirements.txt`

Install the missing MikTex packages by navigating to the frc-docs directory, then running the following command from 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 Building

打开 Powershell Window 或终端，然后直接导航到克隆的 frc-docs 目录。

```
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

备注： Lint Check 将不会检查 Windows 上的行尾，因为存在行尾错误。有关更多信息，请参见 this issue for more information.

鼓励您检查对 linter 所做的任何更改。如果编译器没有通过，这 **will** 使编译器失败。要检查，请运行 `.\make lint`

Link Check

链接检查器确保文档中的所有链接都能解析。如果编译器没有通过，这 **will** 使编译器失败。要检查，请运行 “。make linkcheck“

Image Size Check

Please run `.\make sizecheck` to verify that all images are below 500KB. This check **will** fail CI if it fails. Exclusions are allowed on a case by case basis and are added to the `IMAGE_SIZE_EXCLUSIONS` list in the configuration file.

Redirect Check

已移动或重命名的文件必须在 `source` 中的 `redirects.txt` 文件中有新位置（或用 404 替换）。

重定向编写器会自动将重命名/移动的文件添加到重定向文件中。运行 `.\make rediraffewritediff`

备注： 如果移动了文件的同时并对其进行了实质性更改，则重定向编写器不会将其添加到 `redirects.txt` 文件中，并且需要手动更新 `redirects.txt` 文件。

重定向检查器确保所有文件都有有效的重定向。如果编译器没有通过，这 **will** 使编译器失败。要进行检查，请运行 `.\make rediraffecheckdiff`，以验证所有文件都已重定向。此外，可能需要运行一个 HTML 以确保所有文件正确重定向。

构建 HTML

输入指令 `.\make html` 以生成 HTML 内容。该内容位于存储库根目录的 `build/html` 目录中。

43.3.3 Building PDF

警告： 请注意，Windows 上的 PDF 程序可能会导致 SVG 内容的图像失真。这是因为 Windows 上缺少 librsvg2-bin 支持。

输入指令 `.\make latexpdf` 以生成 PDF 内容。PDF 位于存储库根目录的 `build/latex` 目录中。

43.3.4 Building EPUB

输入指令 `.\make epub` 以生成 EPUB 内容。EPUB 位于存储库根目录的 `build/epub` 目录中。

43.3.5 添加 Python 第三方库

重要： 在以任何方式修改 frc-docs 依赖关系之后，必须通过从目录的根目录运行 “poetry export -f requirements.txt –output source / requirements.txt –with-hashhes” 来重新生成 “requirements.txt”。回购。

---

frc-docs 使用 Poetry <https://python-poetry.org/>‘__ 来管理其依赖项，以确保构建可复制。

备注： 诗不是构建和贡献于 frc-docs 内容所必需的。*仅 * 用于依赖项管理。

安装诗歌

Ensure that Poetry is installed. Run the following command: `pip install poetry`.

添加依赖

Add the dependency to the `[tool.poetry.dependencies]` section of `pyproject.toml`. Make sure to specify an exact version. Then, run the following command: `poetry lock --no-update`.

更新顶级依赖关系

Update the dependency’s version in the `[tool.poetry.dependencies]` section of `pyproject.toml`. Then, run the following command: `poetry lock --no-update`.

更新隐藏的依赖项

Run the following command: `poetry lock`.

43.4 Draw.io 保存说明

警告： 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.

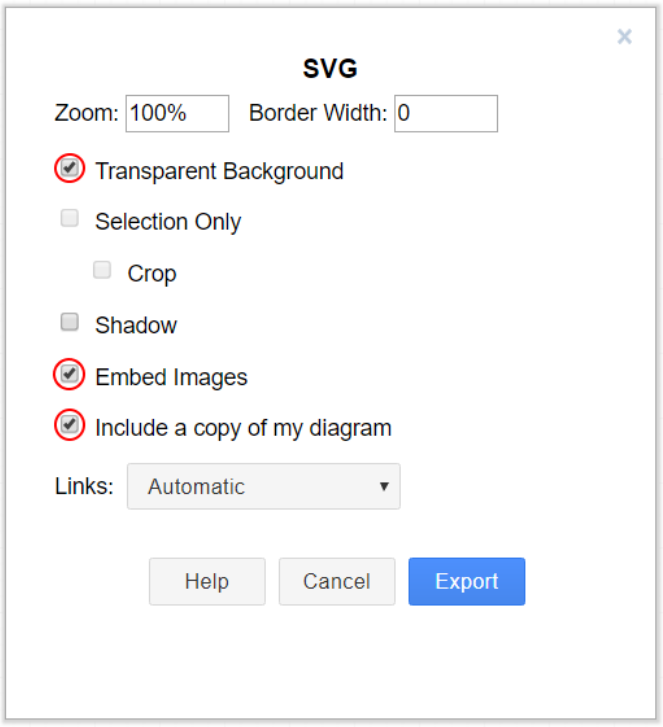
将 Draw.io（也称为 'diagrams.net <https://app.diagrams.net/>'__）保存为 "svg'' 文件时，支持 draw.io 源文件的嵌入式 XML 元数据（通常存储为 ".drawio'' '')。这使这些图像既可以用作图表的源文件，也可以在以后进行编辑，并以普通的 svg 文件的形式呈现。

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.

此方法适用于 draw.io 桌面版本和 'diagrams.net <https://app.diagrams.net/>'__ 上的 Web 版本。

要导出，请转到 "File - Export as - SVG…"。确保启用了 "Include a copy of my diagram"以嵌入图元数据，并且启用了 "Embed Images"，以便嵌入图中的图像文件，以便它们在文档中呈现。另外，标记 "Transparent Background" 选项以确保背景正确显示。

导出菜单应如下所示：



然后只需单击 "Export''，然后选择要保存文件的位置并保存它。

备注： 保存时，检查是否跟从了:ref:'docs/contributing/frc-docs/style-guide:Draw.io Files'的指南。

43.5 翻译

frc-docs 使用基于 Web 的 'Transifex <https://www.transifex.com>'__ 程序来帮助翻译。frc-docs 已被翻译成西班牙语-墨西哥（es_MX），法语-加拿大（fr_CA）和土耳其语-土耳其（tr_TR）。中文-中国（zh_CN），希伯来语-以色列（he_IL）和葡萄牙语-巴西（pt_BR）的翻译正在进行中。会说流利的英语和指定语言之一的翻译者将为翻译做出贡献。即使翻译完成，也需要对其进行更新，以跟上 frc-docs 中的更改。

43.5.1 工作流程

以下是翻译 frc-docs 的一些步骤。

1. 注册 'Transifex <https://www.transifex.com/>'__，并要求加入 'frc-docs project <https://www.transifex.com/wpilib/frc-docs>'__，然后询问您愿意为之付出努力的语言。
2. 加入 GitHub'discussions <https://github.com/wpilibsuite/allwpilib/discussions>'__！这是与 WPILib 团队进行直接沟通的方式。您可以使用它以快速，简化的方式向我们提问。
3. 在被授予访问 frc-docs 翻译项目的权限之前，可能会与您联系并询问涉及贡献语言的问题。
4. 化掉语言的隔阂!

43.5.2 链接

链接必须保留其原始语法。要翻译链接，您可以将 TRANSLATE ME 文本（将被英文标题替换）替换为适当的翻译。

这是原始文本的示例

```
For complete wiring instructions/diagrams, please see the :doc:`Wiring the FRC␣
↪Control System Document <Wiring the FRC Control System document>`.
```

当 "Wiring the FRC Control System Document"被翻译。

```
For complete wiring instructions/diagrams, please see the :doc:`TRANSLATED TEXT
↪<Wiring the FRC Control System document>`.
```

另外一个例子如下

```
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 翻译内容上线

每天都会从 Transifex 提取翻译并自动发布。

43.5.4 准确度

翻译内容应与原文准确无误。如果英文文本需要改善，请在 frc-docs repository 中开始一个 PR 或 issue。然后可以在合并时翻译它们。

43.6 主要翻译

43.6.1 中文

- 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
- 堂晋徐
- 志鹏邢
- 怡静陆
- 智翔杨
- 楚涵张
- 璨宇汤
- 陈 Sherry

43.6.2 法语

- 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 葡萄牙语

- 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 西班牙语

- 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 土耳其语

- 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 希伯来语

- Aric Radzin
- Dalton Smith
- Itay Ziv
- Ofek Ashery
- Shai Grossman
- Starlight220
- Yotam Shlomi

44 使用 allwpilib 开发

重要： 这个文档仅供 WPILIB 开发者参考，不是在正式比赛中的编程指南

allwpilib 是指向存储库的各种文档的链接列表。

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 核心代码库

44.3 NetworkTables

索引