
FIRST Robotics Competition

WPILib

09 May 2024

Sıfırdan Robota

1	Giriş	3
2	Adım 1: Robotunuzu İnşa Etme	5
3	Adım 2: Yazılımın Kurulması	39
4	Adım 3: Robotunuzu Hazırlama	73
5	Adım 4: Robotunuzu Programlama	93
6	Donanım Bileşenine Genel Bakış	117
7	Yazılım Bileşenine Genel Bakış	147
8	WPILib Nedir?	157
9	2024 Overview	159
10	VS Code'a Genel Bakış	173
11	Dashboards	201
12	Telemetry	335
13	FRC LabVIEW Programlama	349
14	FRC Python Programming	385
15	Hardware APIs	391
16	CAN Aygıtları	483
17	Temel Programlama	499
18	Destek Kaynakları	575
19	FRC Sözlüğü	577
20	Driver Station	585

21 RobotBuilder	615
22 Robot Simülasyonu	689
23 OutlineViewer	725
24 roboRIO Team Number Setter	727
25 Görüntü İşleme	729
26 Komut Tabanlı Programlama	793
27 Kinematik ve Odometri	897
28 NetworkTables	927
29 Path Planning	987
30 roboRIO	1025
31 Gelişmiş GradleRIO	1039
32 Gelişmiş Kontroller	1061
33 Kullanışlı Özellikler	1219
34 WPILib Örnek Projeler	1227
35 Third Party Example Projects	1233
36 Donanım - Temel Prensipler	1235
37 Donanım Eğitimleri	1289
38 Sensörler	1291
39 Romi'ye Giriş	1331
40 Getting Started with XRP	1357
41 Ağa Giriş	1369
42 Ağ Araçları	1401
43 frc-docs'a Katkıda Bulunmak	1405
44 Allwpilib ile geliştirme	1425
Dizin	1427

FIRST® Robotics Competition Kontrol Sistemi Dokümantasyonu'na hoşgeldiniz! Bu site yarışma robotunuzu kodlamanız için gereken her şeyi barındırıyor!

Sol-alt menüde çeşitli dillere yapılan gönüllü çevirilere ulaşabilirsiniz.

Geri Dönen Takımlar

Eğer tekrardan katılacak bir takımsanız, 2023-2024 arası olan değişikliklere, sık karşılaşılan sorunlara ve güncelleme için hızlı başlangıç kılavuzuna göz gezdirmeyi unutmayınız.

[Değişiklik Günlüğü](#)

[Bilinen Sorunlar](#)

[Hızlı Başlangıç](#)

Yeni Takımlar

Sıfırdan Robot'a eğitimi size temel bir robotu hazırlama, kablolama ve programlama konusunda rehberlik edecek!

[Go to Zero-to-Robot](#)

Hardware Overview

Takımların kullanabileceği donanım bileşenlerine genel bakış.

[Go to Hardware Overview](#)

Software Overview

Takımların kullanabileceği yazılım bileşenlerine ve araçlarına genel bakış.

[Go to Software Overview](#)

Programming Basics

Takıma programlama süreci boyunca faydalı olacak dokümanlar.

[View articles](#)

Advanced Programming

Veteran takımlara hitabeden belgeler. Bu belgelerde Yol Planlama ve Kinematik gibi içerikler bulunuyor.

[View articles](#)

Hardware

Takımlar için donanım eğitimleri ve içerik mevcuttur.

[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

FRC Driver Station, Dashboards, roboRIO Görüntüleme Aracı ve daha fazlası gibi temel araçlar.

[View articles](#)

Example Projects

Bu kısım, takımların VS kodunu referans alarak mevcut projelere örnek gösterir.

[View articles](#)

Status Light Quick Reference

Çeşitli FRC donanımlarındaki durum ışıkları için hızlı başvuru kılavuzu.

[View article](#)

3rd Party libraries

Robot projenize CTRE ve REV gibi 3. taraf kitaplıkları ekleme eğitimi.

[View article](#)

FIRST® Robotics Competition kontrol sistemi ve WPILib yazılım paketleri resmi belgeler ana sayfasına hoş geldiniz. Bu sayfa, FRC|reg|'in kontrol sistemini (kablolama, konfigürasyon ve yazılım dahil) ve ayrıca WPILib kitaplıkları ve araçları için birincil kaynaktır.

1.1 Programlamada yeni misiniz?

Bu sayfalar, WPILib kitaplıklarının ve FRC Kontrol Sisteminin özelliklerini kapsar ve desteklenen programlama dillerini kullanmanın temellerini açıklamaz. Desteklenen programlama dillerini öğrenmeye yönelik kaynaklar istiyorsanız, aşağıdaki önerilere göz atın:

Not: Programlama dili bilgisi olmadan işleyen bir temel robot elde etmek için bu Zero-to-Robot bölümüyle devam edebilirsiniz. Bunun ötesine geçmek için, programlamayı seçtiğiniz dile aşina olmanız gerekecek.

1.1.1 Java

- [Code Academy](#)
- [Head First Java 2nd Edition](#) başlangıç seviyesi dostu ve Java programlamaya giriş (ISBN-10: 0596009208).

1.1.2 C++

- [LearnCPP](#)
- [Programming: Principles and Practice Using C++ 2nd Edition](#) C++ dilinin yaratıcısından başlangıç seviyesi bir giriş.(ISBN-10: 0321992784).
- [C++ Primer Plus 6th Edition](#) (ISBN-10: 0321776402).

1.1.3 LabVIEW

- [NI Learn LabVIEW](#)

1.1.4 Python

- [List of various guides to learn Python](#)

1.2 Zero to Robot - Sıfırdan Robota

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.

Adım 1: Robotunuzu İnşa Etme

Mevcut kontrol sistemine genel bir bakış [burada bulunabilir](#).

2.1 Introduction to FRC Robot Wiring

Not: 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

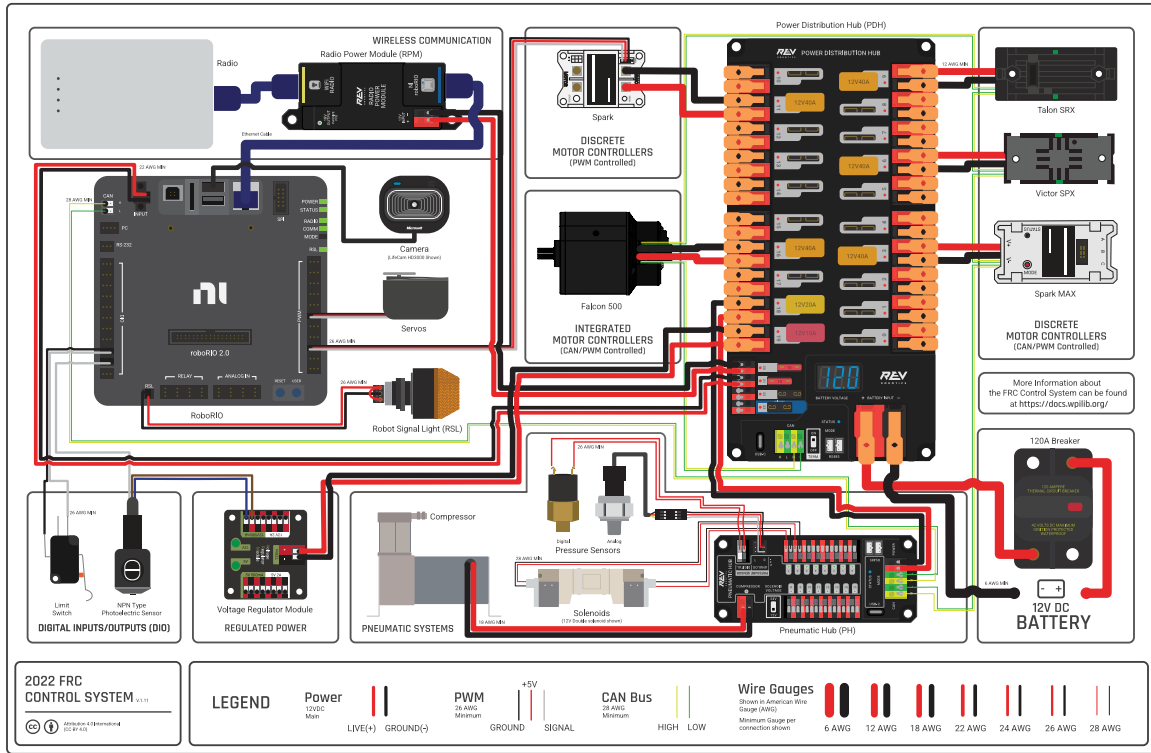
REV

CTR

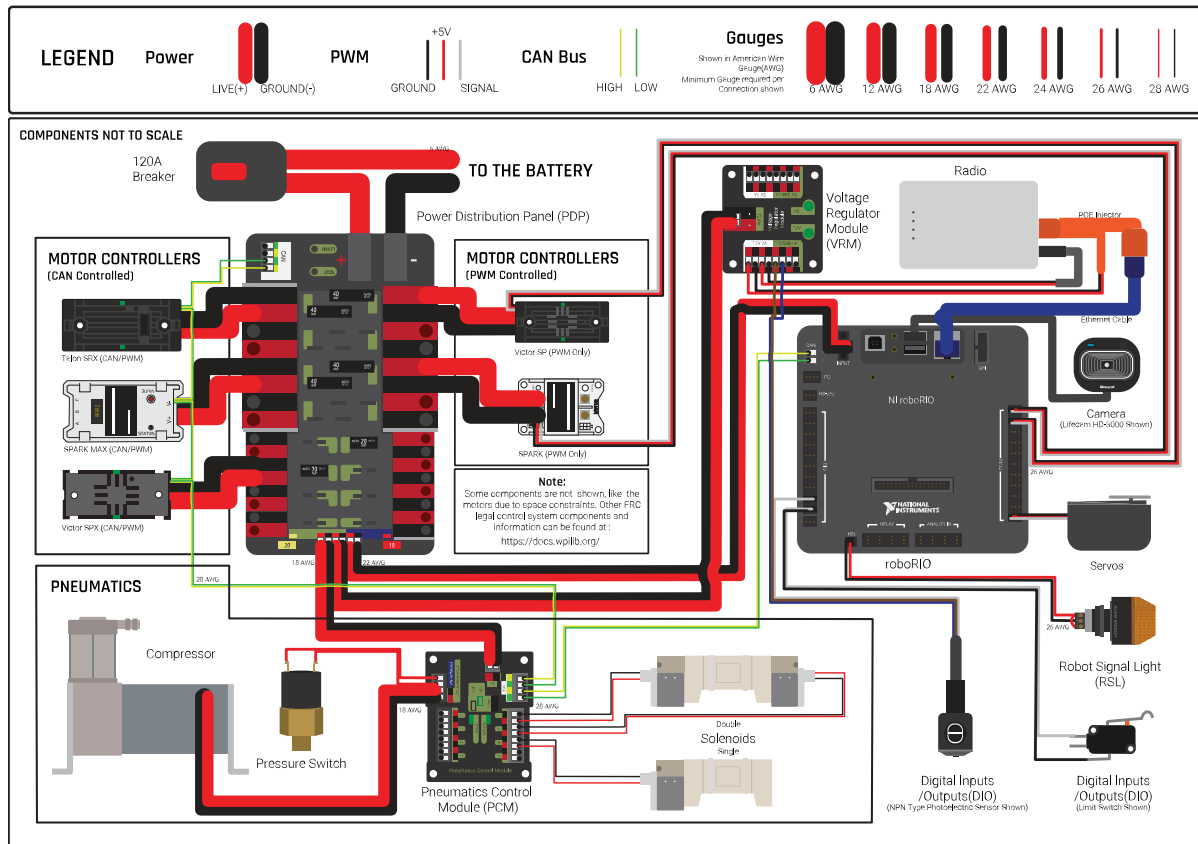
2.1.2 Gather Materials

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](#))



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



Şekil 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.

2.1.5 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.

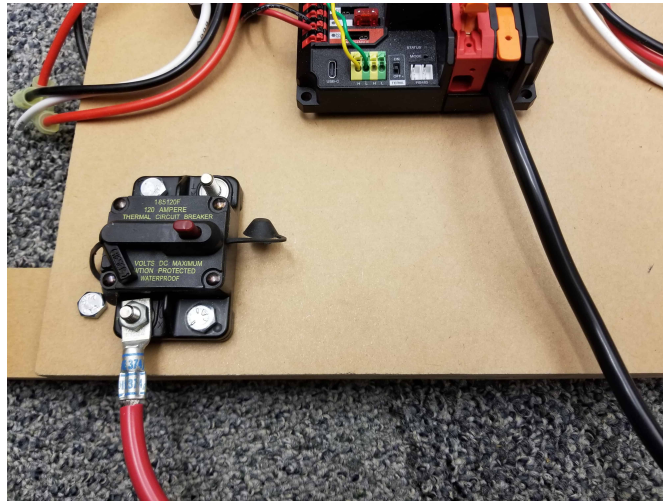
2.1.6 Attach Robot Side Battery Connector

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.

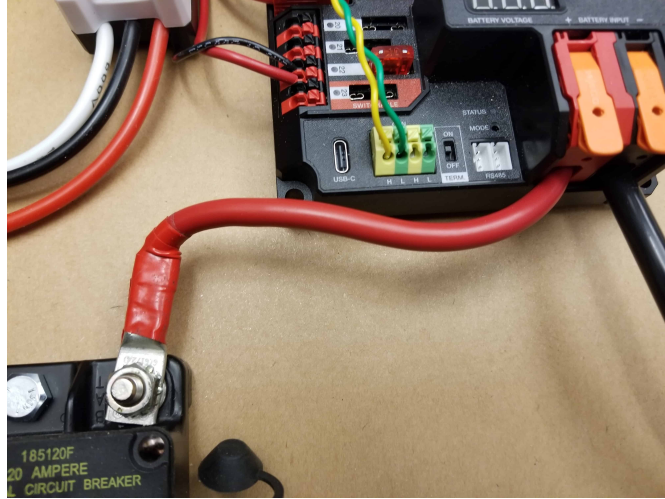


Requires: Battery Connector, 6 AWG (16 mm^2) terminal lugs, 7/16" (11 mm) Box end
Attach terminal lug to positive (red) wire of battery connector. Strip .75" off the black wire.



2.1.7 Wire Breaker to Power Distribution

REV

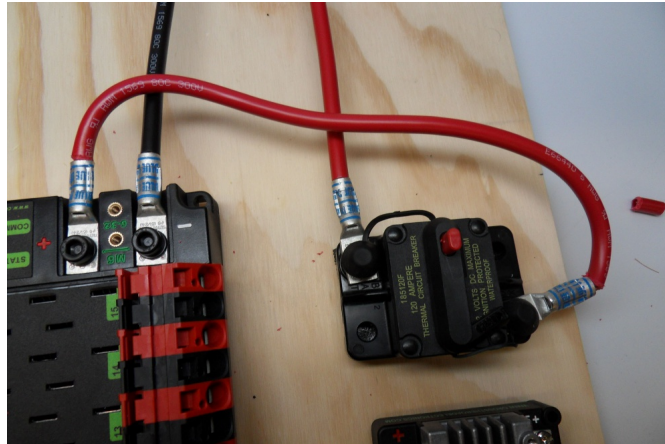


Requires: 6 AWG (16 mm^2) red wire, 1x 6 AWG (16 mm^2) terminal lugs, 7/16" (11 mm) wrench

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

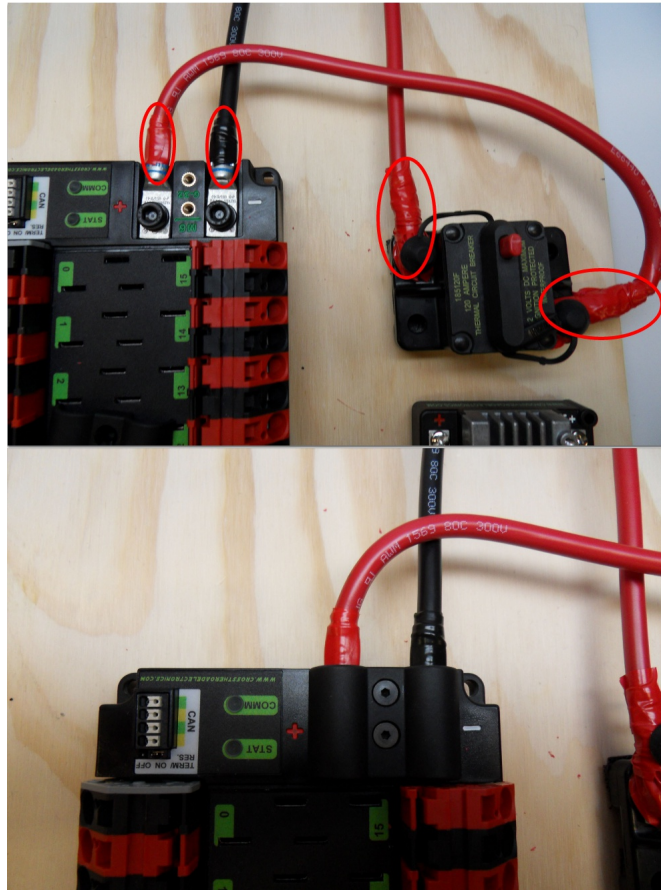
REV



Requires: Electrical tape

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

CTR

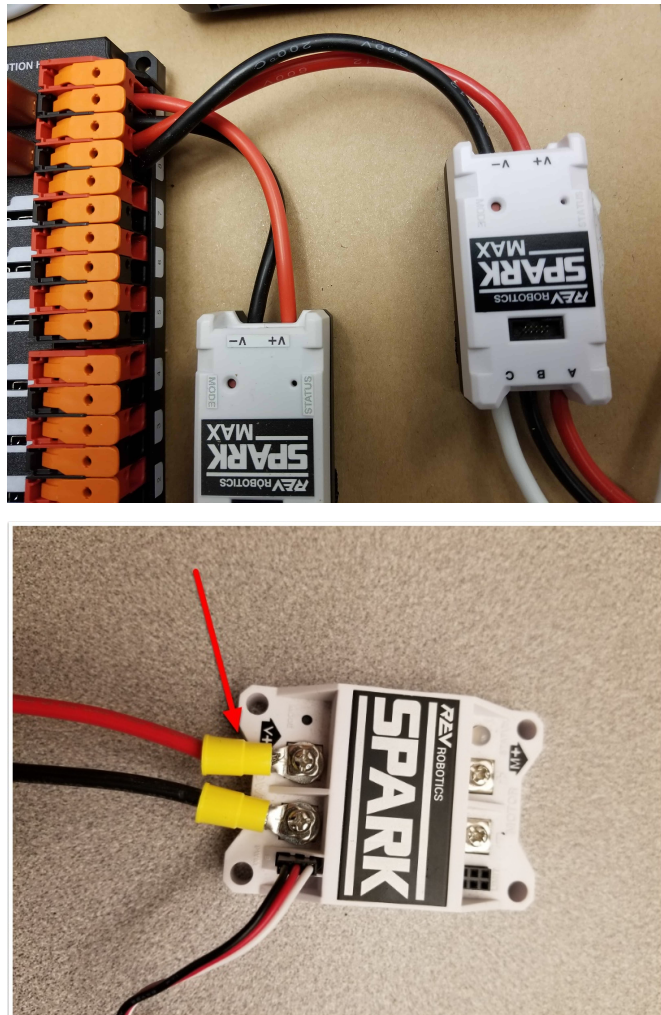


Requires: 1/16" Allen, Electrical tape

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):

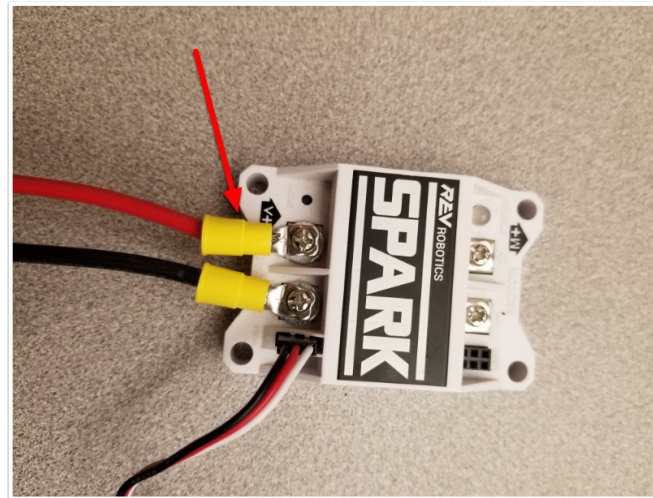
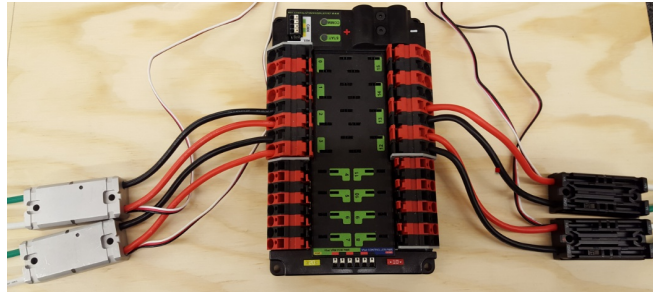
1. Cut red and black wire to appropriate length to reach from one of the Wago terminal pairs to the input side of the motor controller (with a little extra for the length that will be inserted into the terminals on each end)
2. Strip one end of each of the wires, then insert into the Wago terminals.
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 ($0.25 - 6 \text{ mm}^2$), strip 11-12 mm ($\sim 7/16''$)
- Large Wago connector: Accepts 6 - 12 AWG ($4 - 16 \text{ mm}^2$), strip 12-13 mm ($\sim 1/2''$)

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 ($4 - 6 \text{ mm}^2$) wire, 10 or 12 AWG ($4 - 6 \text{ mm}^2$) 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:

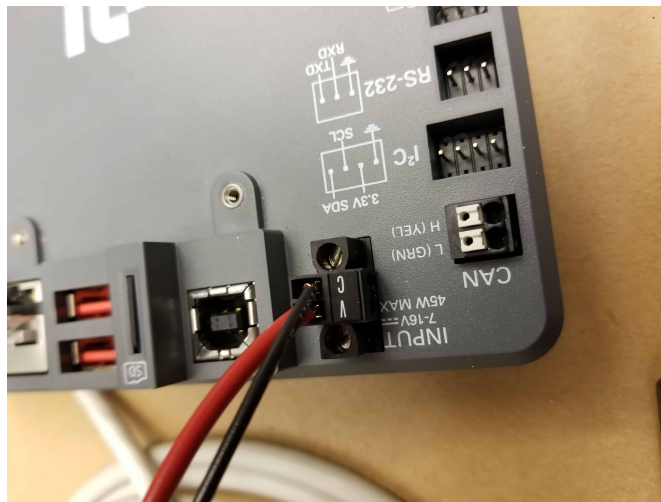
- Wire should be 16 AWG (1.5 mm^2) to 24 AWG (0.25 mm^2) (consult rules to verify required gauge for power wiring)
- Wire ends should be stripped approximately 5/16 (~8 mm)
- To insert or remove the wire, press down on the corresponding “button” to open the terminal

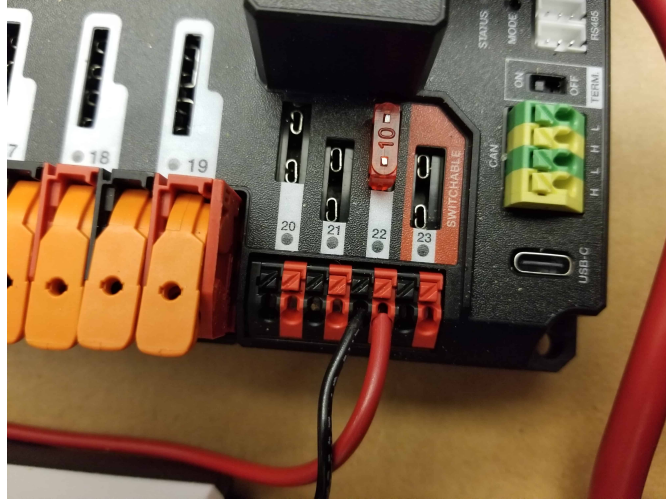
After making the connection check to be sure that it is clean and secure:

- 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

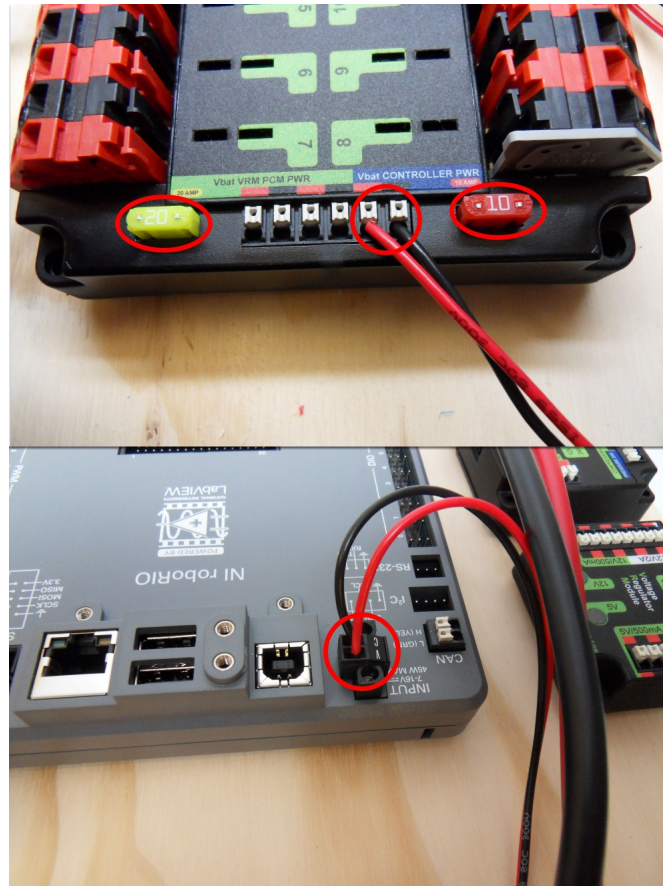




Requires: 10A mini fuse, Wire stripper, very small flat screwdriver, 18 AWG (1 mm^2) Red and Black

1. Insert the 10A fuse into the PDH in one of the non-switchable fused channels (20-22).
2. Strip $\sim 5/16''$ ($\sim 8 \text{ mm}$) 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 robRIO. 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 robRIO (red to V, black to C). Also make sure that the power connector is screwed down securely to the robRIO.

CTR

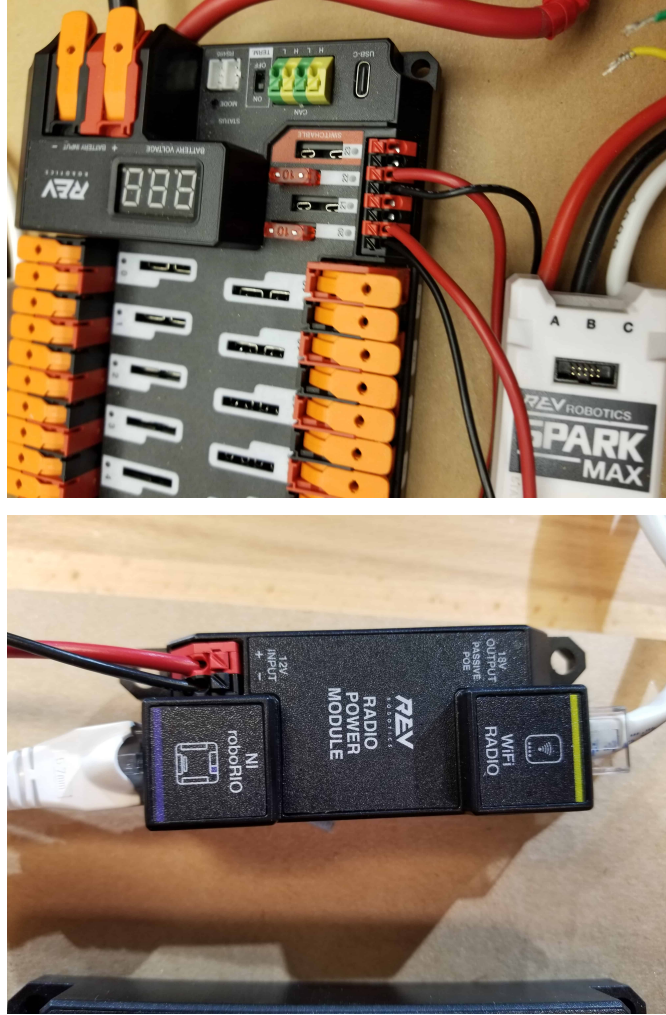


Requires: 10A/20A mini fuses, Wire stripper, very small flat screwdriver, 18 AWG (1 mm^2) Red and Black

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 $\sim 5/16"$ ($\sim 8 \text{ mm}$) 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 $\sim 5/16''$ ($\sim 8 \text{ mm}$) 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 $\sim 5/16''$ ($\sim 8 \text{ mm}$) from the end of the wire.
5. Connect the wire to the RPM 12V Input terminals.

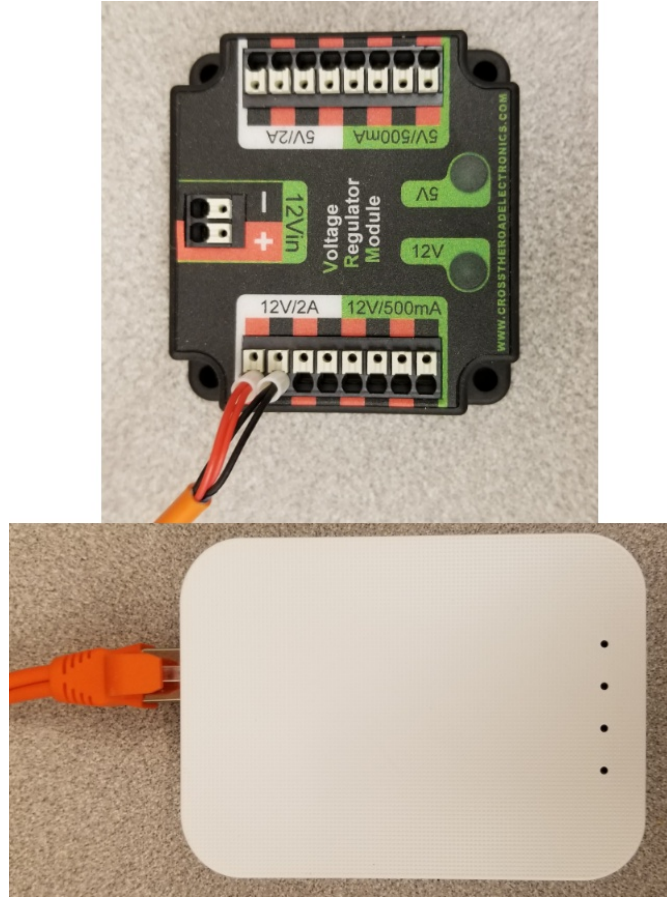
CTR



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

1. Strip $\sim 5/16"$ ($\sim 8 \text{ mm}$) 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 $\sim 5/16"$ ($\sim 8 \text{ mm}$) from the end of the wire.
5. Connect the wire to the VRM 12Vin terminals.

Uyarı: 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.

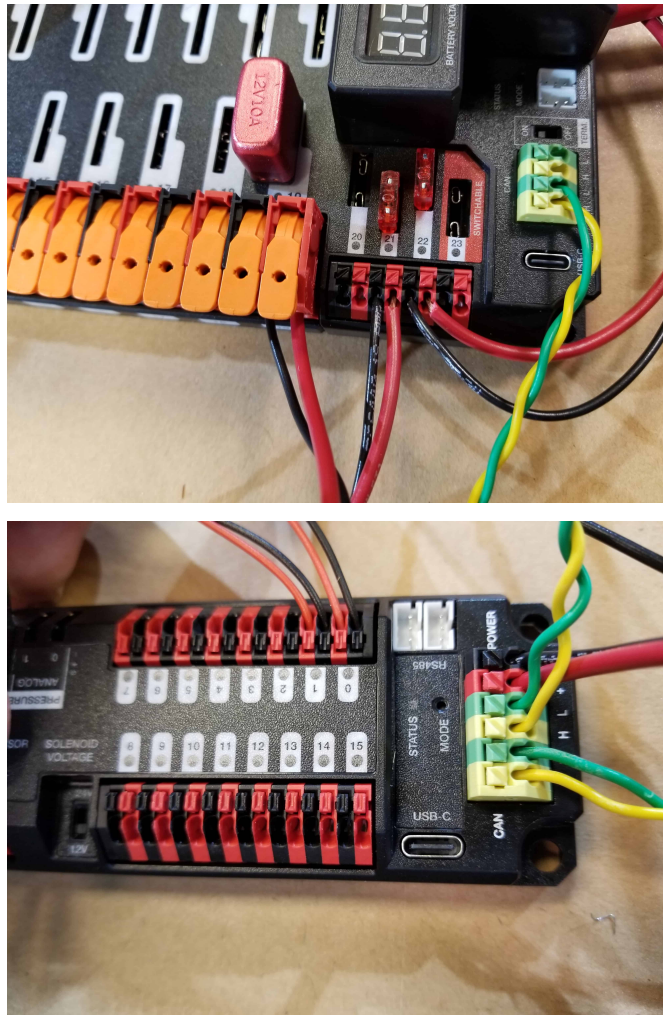


Requires: Small flat screwdriver (optional), Rev radio PoE cable

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



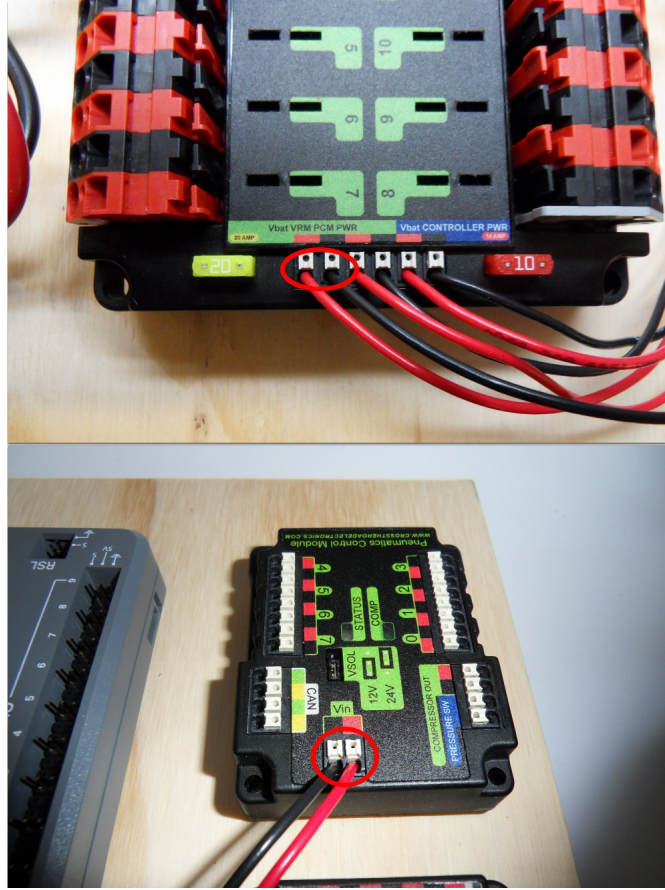
Requires: Wire stripper, small flat screwdriver (optional), 18 AWG (1 mm^2) red and black wire

Not: 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 $\sim 5/16''$ ($\sim 8 \text{ mm}$) 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 \pm . 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 $\sim 5/16''$ ($\sim 8 \text{ mm}$) 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

Not: The PCM is an optional component used for controlling pneumatics on the robot.

1. Strip $\sim 5/16''$ ($\sim 8\text{ mm}$) 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 $\sim 5/16''$ ($\sim 8\text{ mm}$) 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



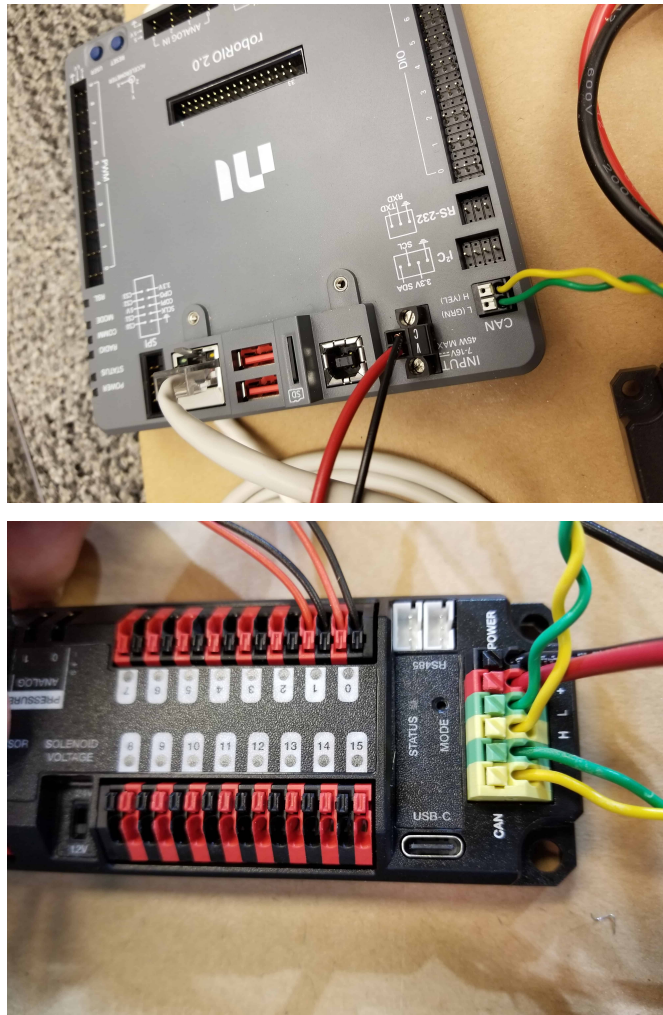
Requires: Ethernet cable

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

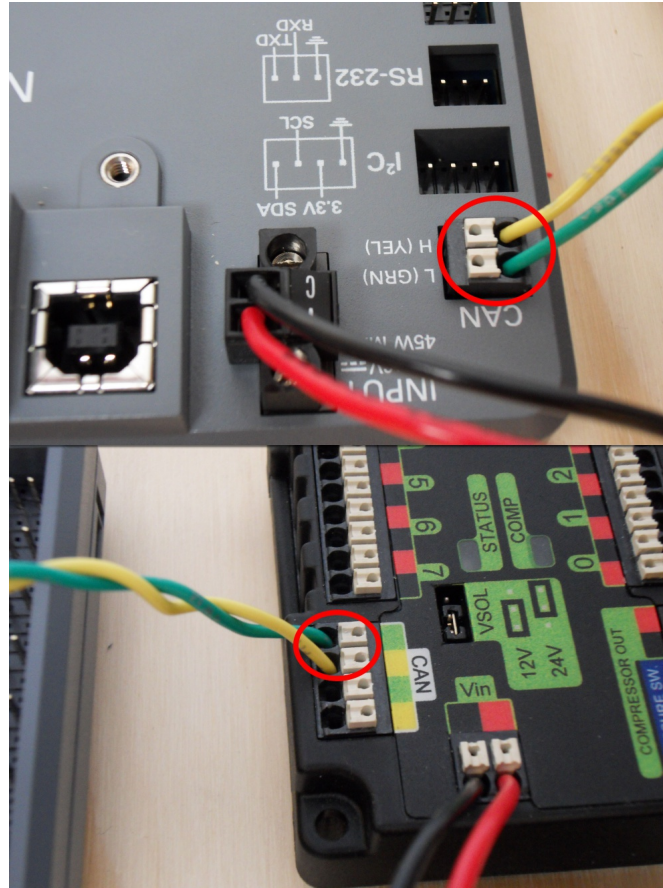


Requires: Wire stripper, small flat screwdriver (optional), yellow/green twisted CAN cable

Not: 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 $\sim 5/16''$ (~ 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 $\sim 5/16''$ (~ 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



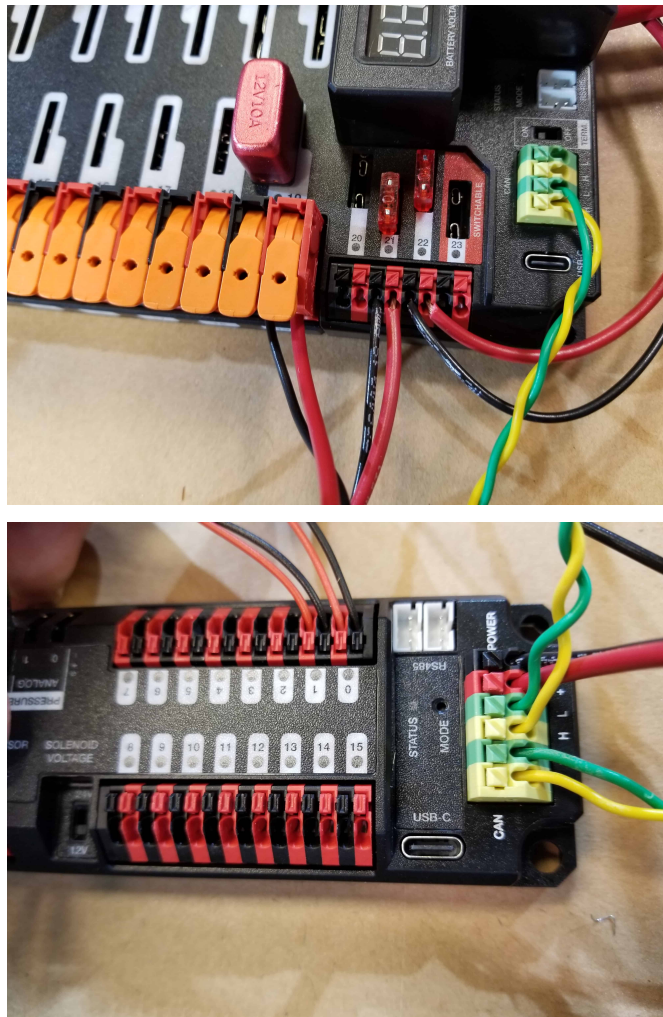
Requires: Wire stripper, small flat screwdriver (optional), yellow/green twisted CAN cable

Not: 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 $\sim 5/16''$ (~ 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 $\sim 5/16''$ (~ 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



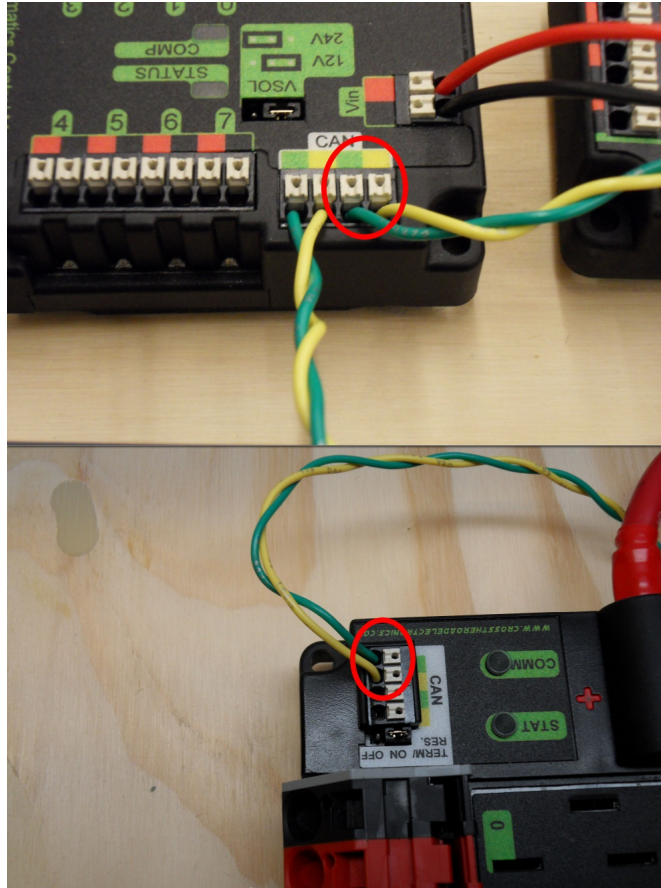
Requires: Wire stripper, small flat screwdriver (optional), yellow/green twisted CAN cable

Not: 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 $\sim 5/16''$ (~ 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 $\sim 5/16''$ (~ 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.

Not: 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

Not: 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 $\sim 5/16''$ (~ 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 $\sim 5/16''$ (~ 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.

Not: 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



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.

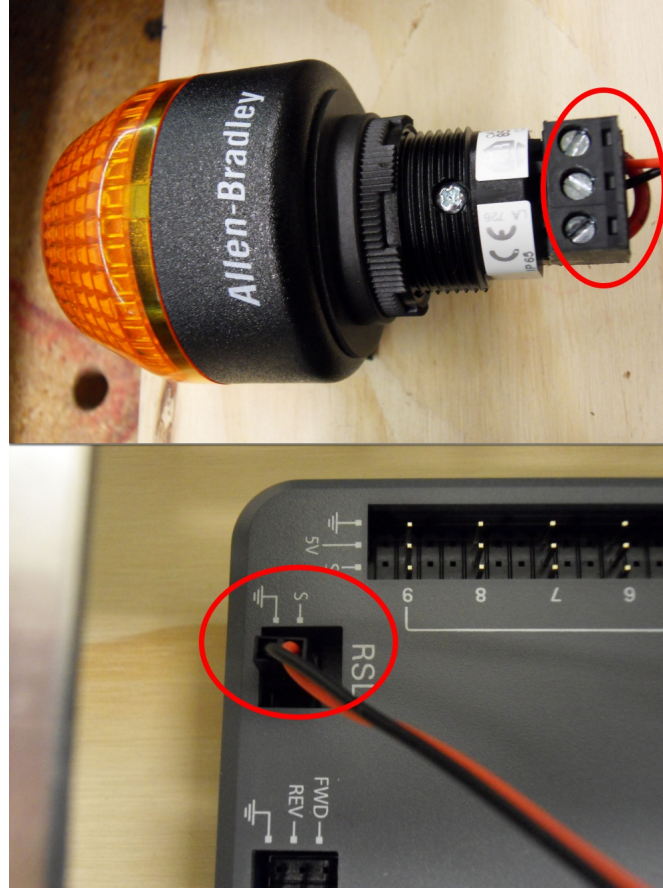
CAN

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 zip tie, 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.

Tüyo: 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

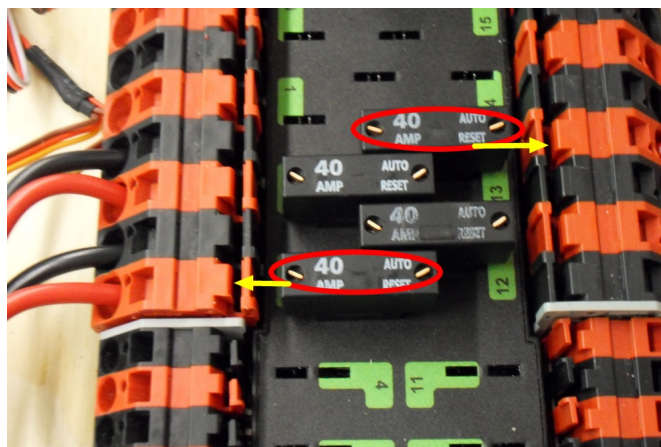


Requires: 4x 40A circuit breakers

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



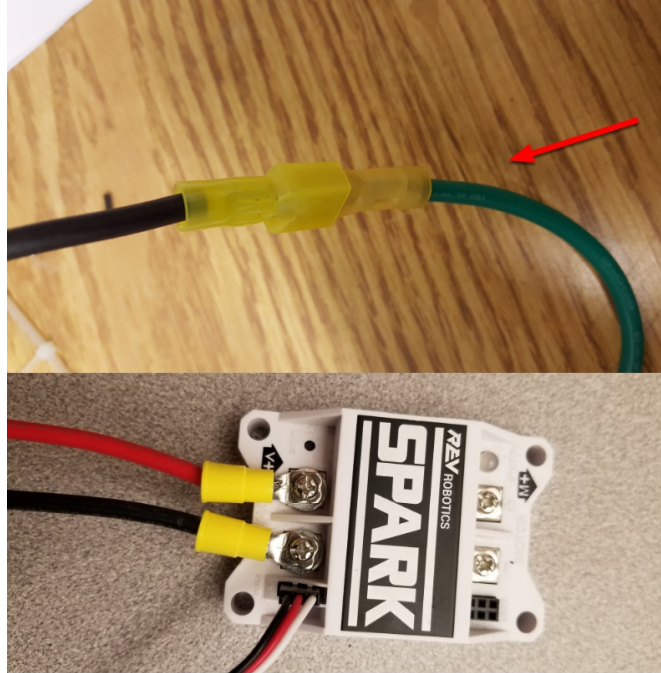
Requires: 4x 40A circuit breakers

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



Tehlike: 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!

Tüyo: 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

Tüyo: 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.

Tüyo: 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!

Adım 2: Yazılımın Kurulması

Mevcut kontrol sistemi yazılımına genel bir bakış bulunabilir :doc: *burada* </docs/controls-overviews/control-system-software>.

3.1 Çevrimdışı Kurulum Hazırlığı

Bu makale, FRC® Control System software'in çevrimdışı kurulumunu yapmanız gerektiğinde toplamak isteyeceğiniz bileşenlere yönelik yönergeler/bağlantılar içerir.

Tüyo: Bu belge, çevrimdışı bilgisayarlara veya birden çok bilgisayara yüklemeyi kolaylaştırmak için aşağıdaki belgelerdeki tüm indirme bağlantılarını derler. İnternete bağlı tek bir bilgisayara kurulum yapıyorsanız, bu sayfayı atlayabilirsiniz.

Not: Bu araçların yüklenme sırası Java ve C ++ ekipleri için önemli değildir. LabVIEW, FRC Oyun Araçları veya 3. Taraf kütüphanelerinden önce kurulmalıdır.

3.1.1 Dökümantasyon

This documentation can be downloaded for offline viewing. The link to download the PDF can be found [here](#).

3.1.2 Yükleyiciler

Tüm takımlar

- [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 Takımları

- [LabVIEW Base Installer](#) (Note: Click on link for “Individual Offline Installers”)

Java/C++ Takımları

- [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

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.

Not: 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 3. Taraf Kitaplıkları/Yazılımları

WPILib'e bağlanan mevcut 3. taraf yazılımların bir dizini şu adreste bulunabilir: [3. Taraf Kütüphaneleri](#).

3.2 FRC için LabVIEW'i yükleme (yalnızca LabVIEW)

Not: 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](#).

İndirme ve yükleme süreleri, bilgisayar ve internet bağlantı özelliklerine göre büyük ölçüde değişecektir, ancak bu işlemin büyük bir dosya indirme ve yükleme içerdiğini ve muhtemelen tamamlanmasının en az bir saat alacağını unutmayın.

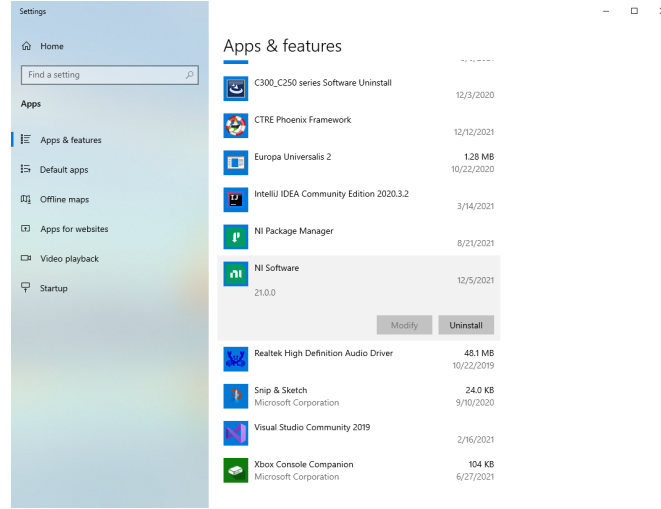
3.2.1 Requirements

- Windows 10 or higher (Windows 10, 11)

3.2.2 Eski Sürümleri Kaldırma (Önerilir)

Not: Eğer cRIO'ları programlamaya devam etmek istiyorsanız, FRC | reg | için LabVIEW kurulumunu sürdürmeniz gerekecektir. 2014. LabVIEW for FRC 2014 lisansı uzatıldı. Bu sürümlerin tek bir bilgisayarda bir arada bulunabilmesi gerekirken, bu kapsamlı bir şekilde test edilmiş bir yapılandırma değildir.

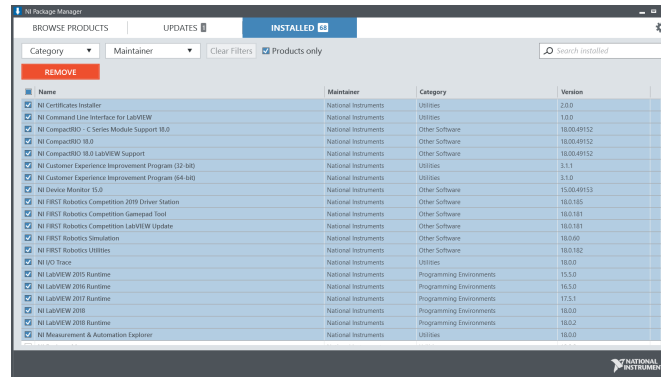
Before installing the new version of LabVIEW it is recommended to remove any old versions. The new version will likely co-exist with the old version, but all testing has been done with FRC 2024 only. Make sure to back up any team code located in the "User\LabVIEW Data" directory before un-installing.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



Kaldırılacak Bileşenleri Seçin

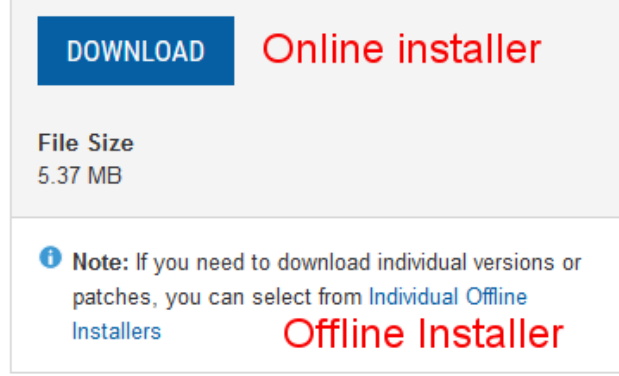
Görünen iletişim kutusunda tüm girişleri seçin. Bunu yapmanın en kolay yolu, “Products Only” onay kutusunun seçimini kaldırmak ve “Name” in solundaki onay kutusunu seçmektir. Remova’a tıklayın. Kaldırıcının tamamlanmasını ve sorduğunda yeniden başlamasını bekleyin.

Uyarı: Bu talimatlar, başka hiçbir NI yazılımının kurulu olmadığını varsayar. Yüklü başka NI yazılımınız varsa, kaldırılmaması gereken yazılımın işaretini kaldırmanız gerekir.



3.2.3 LabVIEW yükleyicisini edinme

Download the [LabVIEW for FRC 2024 installer](#) from NI. Be sure to select the correct version from the drop-down.



Çevrimdışı diğer makinelere yüklemek isterseniz, İndir düğmesine tıklamayın, ****Individual Offline Installers-Bireysel Çevrimdışı Yükleyiciler **** 'e tıklayın ve ardından tam yükleyiciyi indirmek için İndir'e tıklayın.

Not: 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 LabVIEW Kurulumu

NI LabVIEW bir lisans gerektirir. Her sezonun lisansı, bir sonraki yılın 31 Ocak tarihine kadar geçerlidir (ör. 2020 sezonu lisansının süresi 31 Ocak 2021'de sona erer)

Takımların, ilgili yazılıma eşlik eden kısıtlamalara ve lisans koşullarına tabi olarak ve yazılımı yalnızca takım üyelerinin veya danışmanların yalnızca FRC için kullanması koşuluyla, yazılımı gerektiği kadar takım bilgisayarına kurmalarına izin verilir. LabVIEW'i kullanma hakları, yalnızca ilgili yazılımın kurulumu sırasında gösterilen lisans sözleşmelerinin hükümlerine tabidir.

Yüklemeyi Başlatma

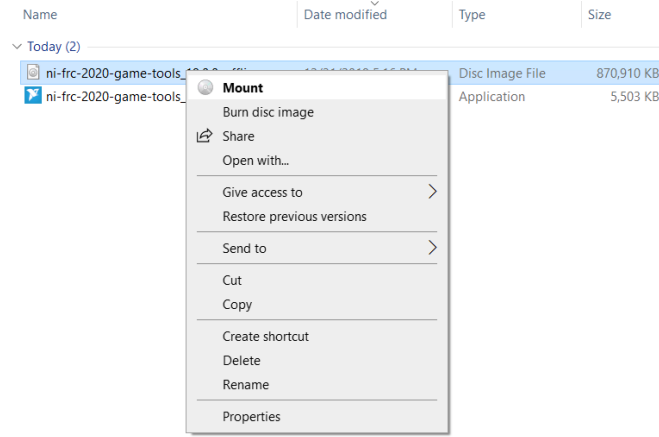
Online Installer

Yükleme işlemini başlatmak için indirilen exe dosyasını çalıştırın. Yes Bir Windows Güvenlik istemi ise *Yes-Evet* e tıklayın.

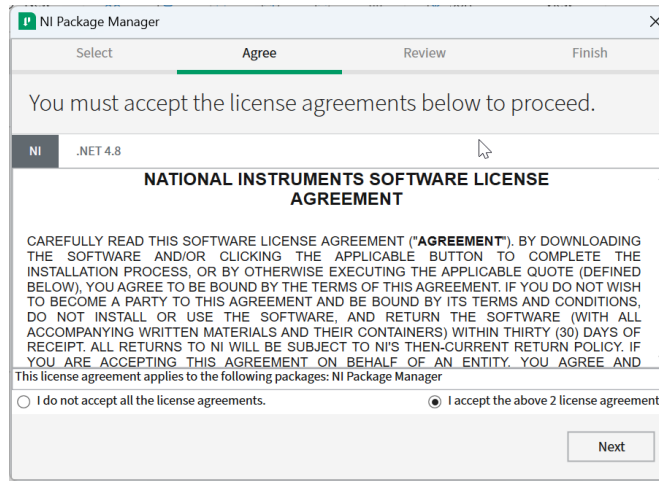
Offline Installer (Windows 10+)

Right click on the downloaded iso file and select mount. Run install.exe from the mounted iso. Click "Yes" if a Windows Security prompt

Not: 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.

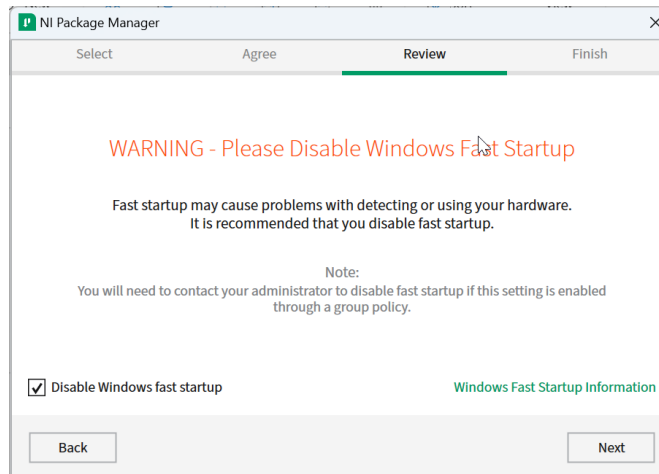


NI Package Manager Lisansı



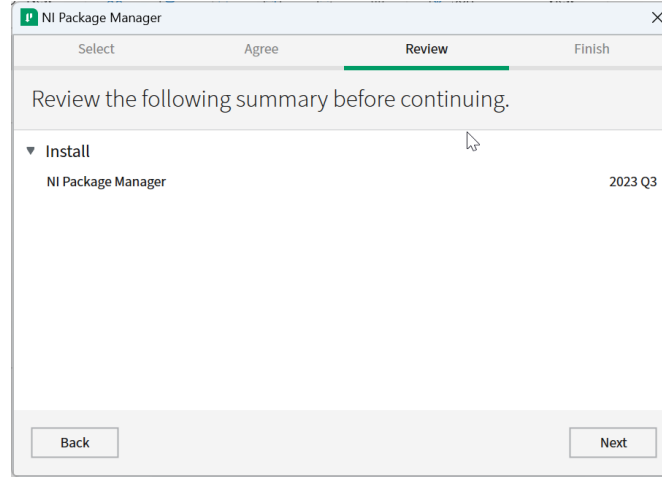
Bu ekranı görürseniz, tıklayın *Next*

Windows Hızlı Başlangıç'ı devre dışı bırakın



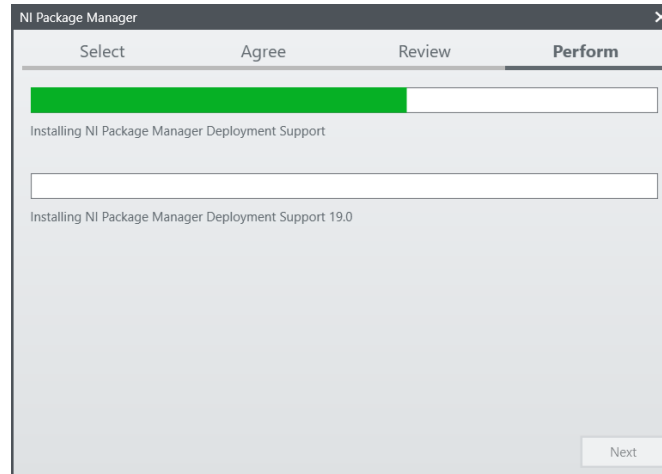
Bu ekranı görürseniz, tıklayın *Next*

NI Package Manager İncelemesi



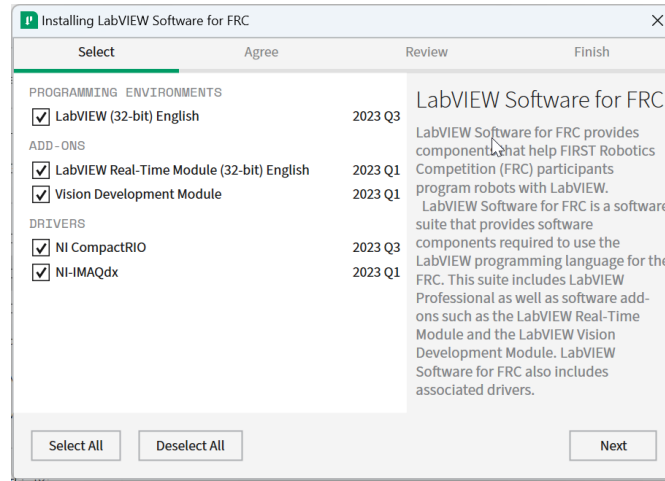
Bu ekranı görürseniz, tıklayın *Next*

NI Package Manager Kurulumu



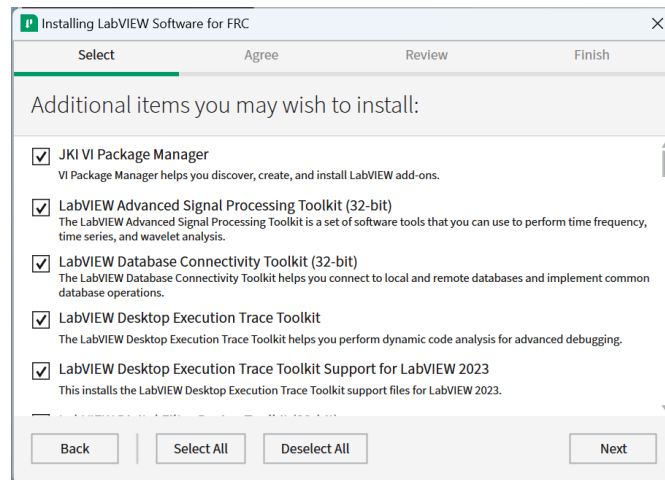
NI Paket Yöneticisinin yükleme ilerlemesi bu pencerede izlenecektir

Ürün listesi



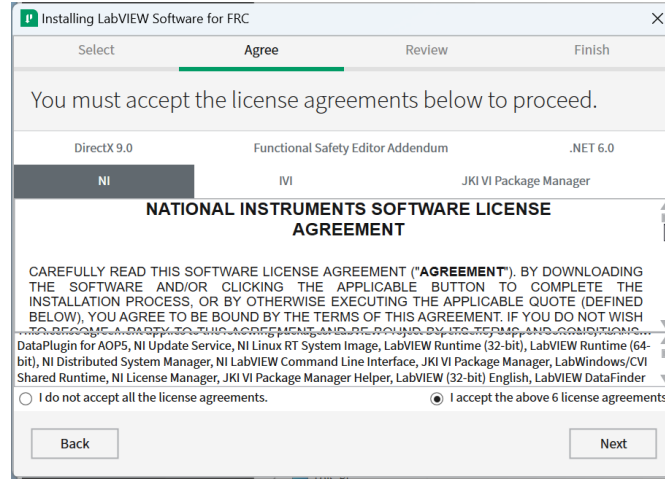
Tıklayın *Next*

Ek Paketler



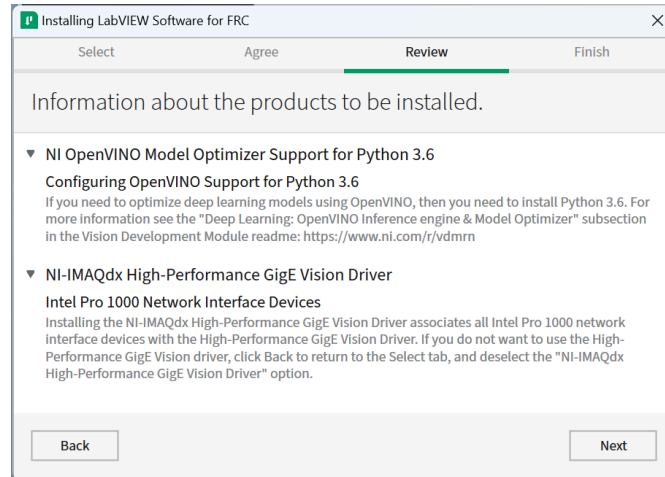
Tıklayın *Next*

Lisans anlaşmaları



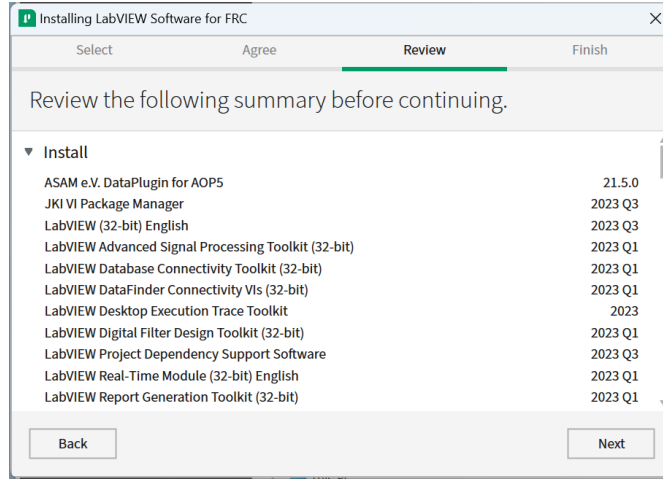
"I accept..." ögesini işaretleyin ve ardından *Next* tıklayın.

Ürün Bilgisi



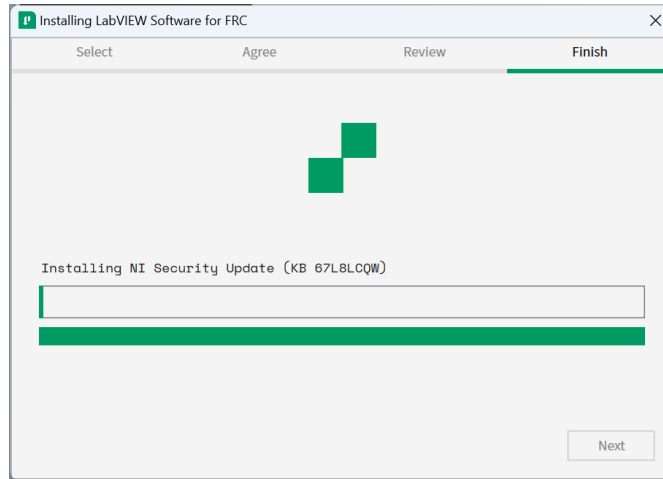
Tıklayın *Next*

Yüklemeyi başlat



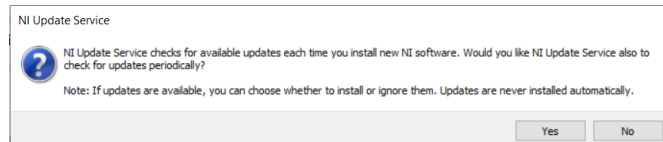
Tıklayın *Next*

Genel ilerleme



Genel yükleme ilerlemesi bu pencerede izlenecektir

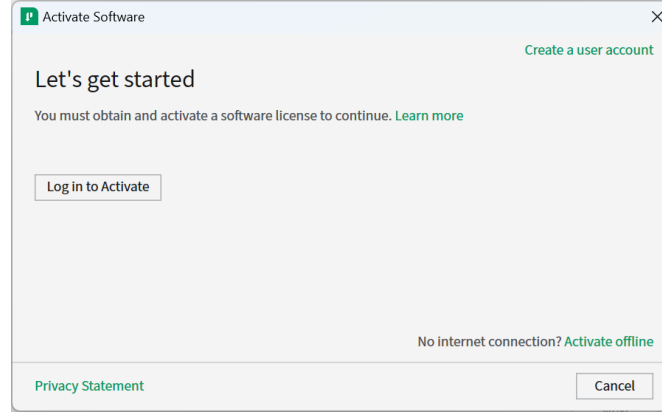
3.2.5 NI Update Hizmeti



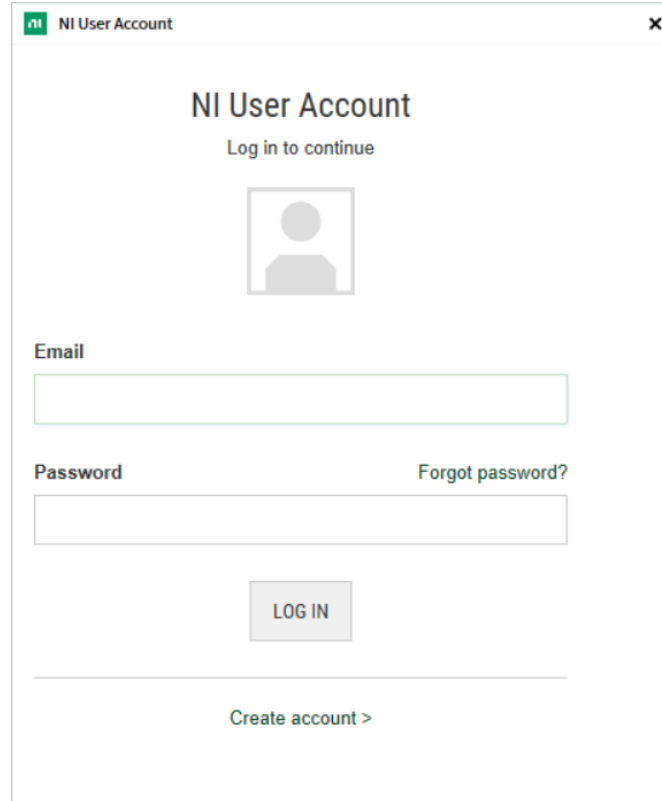
NI güncelleme servisini etkinleştirip etkinleştirmemeniz sorulacaktır. Güncelleme hizmetini etkinleştirmemeyi seçebilirsiniz.

Uyarı: Genel iletişim kanallarımız (FRC Blog, Takım Güncellemeleri veya E-posta duyuruları) aracılığıyla FRC tarafından yönlendirilmedikçe bu güncellemelerin yüklenmesi önerilmez.

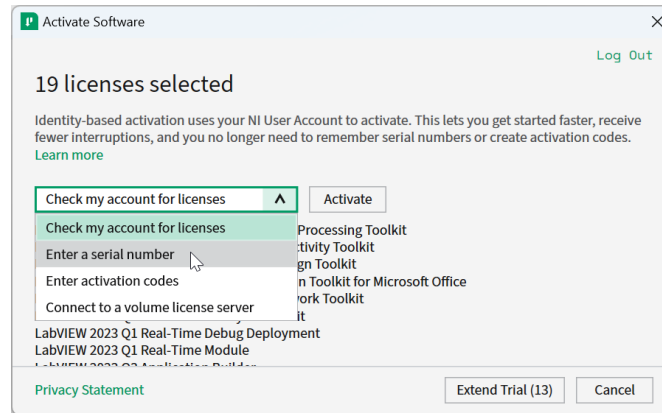
NI Aktivasyon Sihirbazı



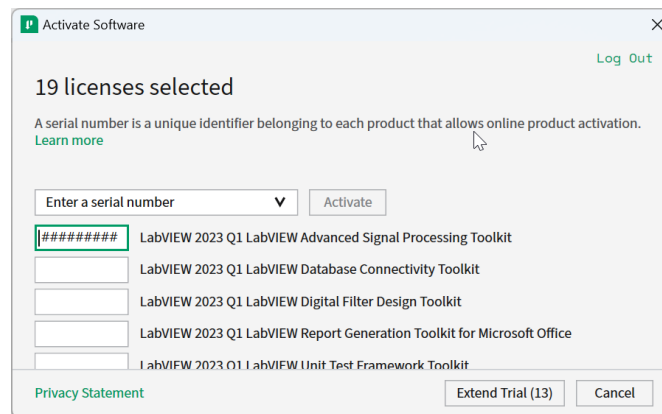
Click the *Log in to Activate* button.



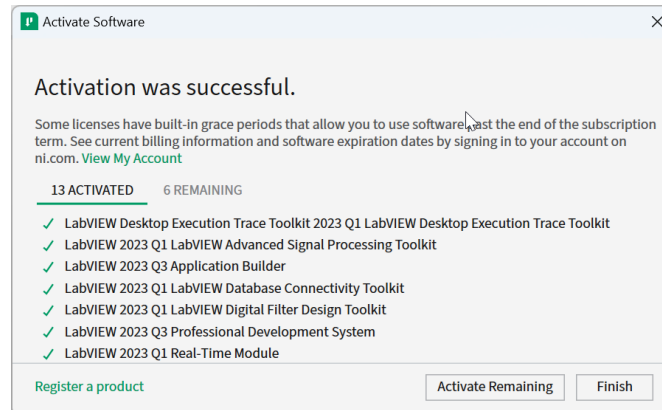
ni.com hesabınıza giriş yapın. Bir hesabınız yoksa, ücretsiz bir hesap oluşturmak için seçin *Create account* .



From the drop-down, select enter a serial number

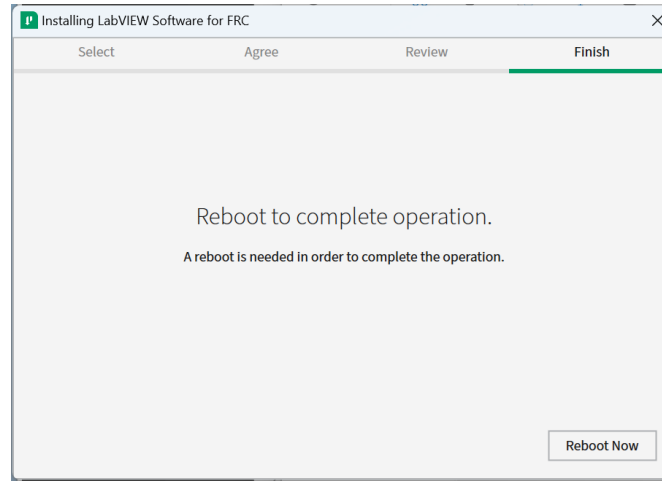


Enter the serial number in all the boxes. Click *Activate*.



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*.

Restart



Select *Reboot Now* after closing any open programs.

3.3 FRC Game Tools Araçlarını Yükleme

FRC ® Game Tools aşağıdaki yazılım bileşenlerini içerir:

- LabVIEW Güncellemesi
- FRC Sürücü İstasyonu
- FRC roboRIO Imaging Tool and Images

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

Not: No components from the LabVIEW Software for FRC package are required for running either the Driver Station or Imaging Tool.

3.3.1 Gereksinimler

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

[DOWNLOAD](#) **Online installer**

File Size
5.37 MB

Note: If you need to download individual versions or patches, you can select from [Individual Offline Installers](#) **Offline Installer**

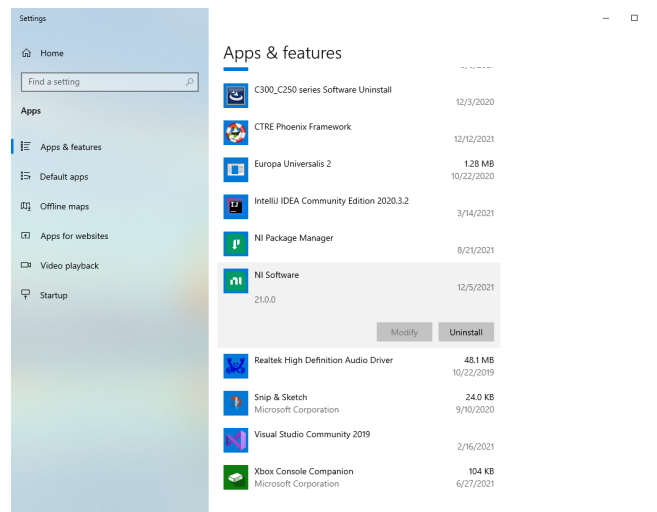
Çevrimdışı başka makinelere kurmak isterseniz, tam yükleyiciyi indirmek için *Download* e tıklamadan önce *Individual Offline Installers* seçeneğini tıklayın.

3.3.2 Eski Sürümleri Kaldırma (Önerilir)

Önemli: LabVIEW ekipleri bu adımı zaten tamamladı, tekrar etmeyin. LabVIEW ekipleri *Kurulum* bölümüne atlamalıdır.

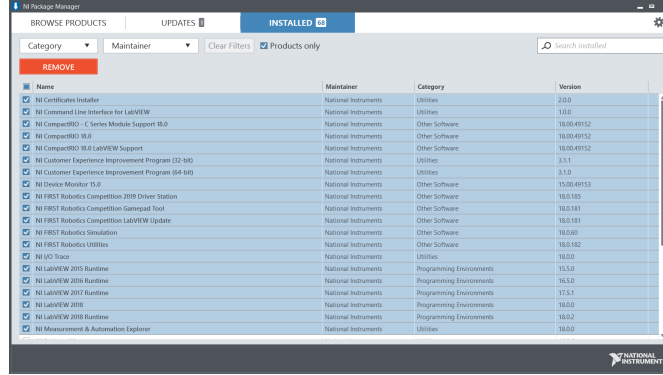
Before installing the new version of the FRC Game Tools it is recommended to remove any old versions. The new version will likely co-exist with the old version (note that the DS will overwrite old versions), but all testing has been done with FRC 2024 only. Then click Start >> Add or Remove Programs.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

Not: 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.



Kaldırılacak Bileşenleri Seçin

Görünen iletişim kutusunda tüm girişleri seçin. Bunu yapmanın en kolay yolu *Products Only* onay kutusunun seçimini kaldırmak ve “Name-Ad” in solundaki onay kutusunu seçmektir. Tıklayın *Remove*. İstenirse kaldırıcının tamamlanmasını ve yeniden başlamasını bekleyin.



3.3.3 Kurulum

Önemli: Game Tools yükleyicisi, .NET Framework 4.6.2'nin güncellenmesi veya yüklenmesi gerektiğini sorabilir. Kurulumu tamamlamak için, istenirse yeniden başlatma da dahil olmak üzere ekrandaki komut istemlerini izleyin. Ardından, gerekirse yükleyiciyi yeniden başlatarak FRC Game Tools kurulumuna devam edin.

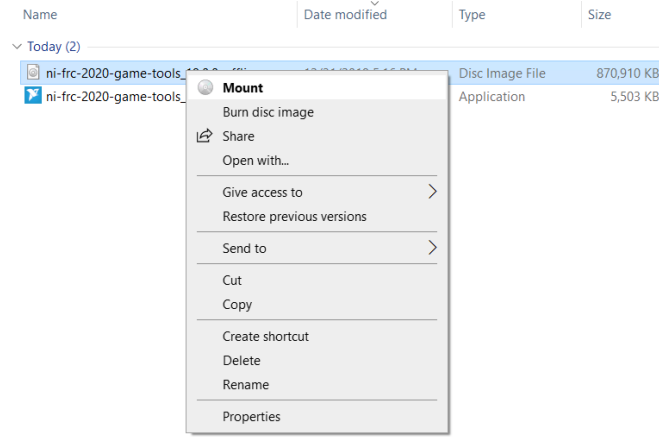
Extraction-Çıkarma

Online

Yükleme işlemini başlatmak için indirilen yürütülebilir dosyayı çalıştırın. Bir Windows Güvenlik istemi görünürse *Yes* e tıklayın.

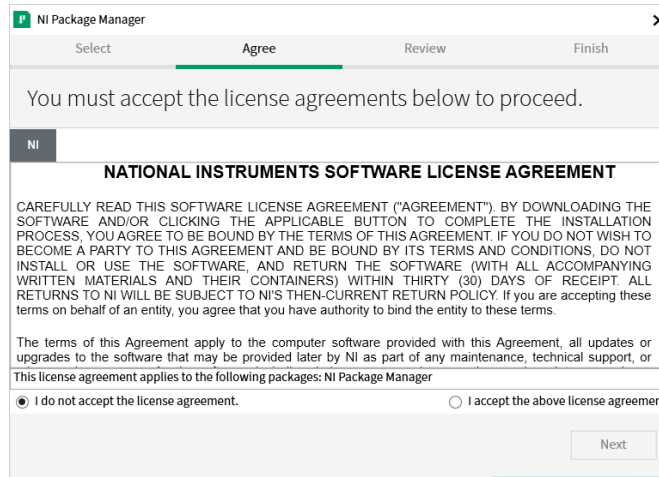
Offline (Windows 10+)

İndirilen iso dosyasına sağ tıklayın ve şunu seçin *mount* . Takılı iso'dan *install.exe* dosyasını çalıştırın. Bir Windows Güvenlik istemi görünürse *Yes* e tıklayın.



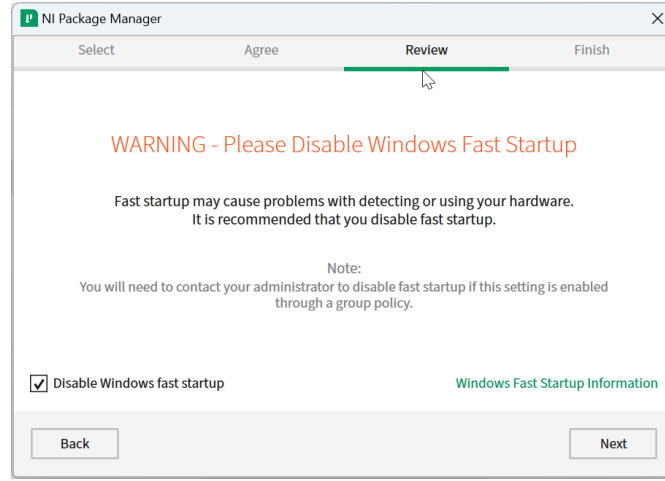
Not: Other installed programs may associate with iso files and the *mount* option may not appear. If that software does not give the option to mount or extract the iso file, then install 7-Zip and use that to extract the iso.

NI Package Manager Lisansı



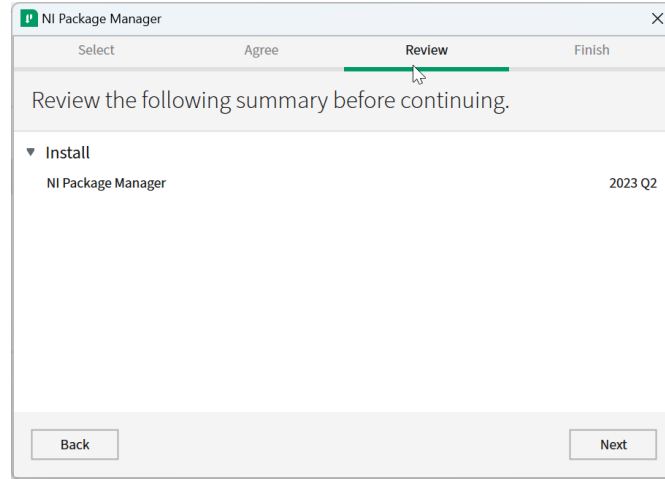
Bu ekranı görürseniz, tıklayın *Next*. Bu ekran, NI Paket Yöneticisi Lisans sözleşmesini kabul ettiğinizi onaylar.

Windows Fast Startup - Hızlı Başlangıç özelliğini kapatın.



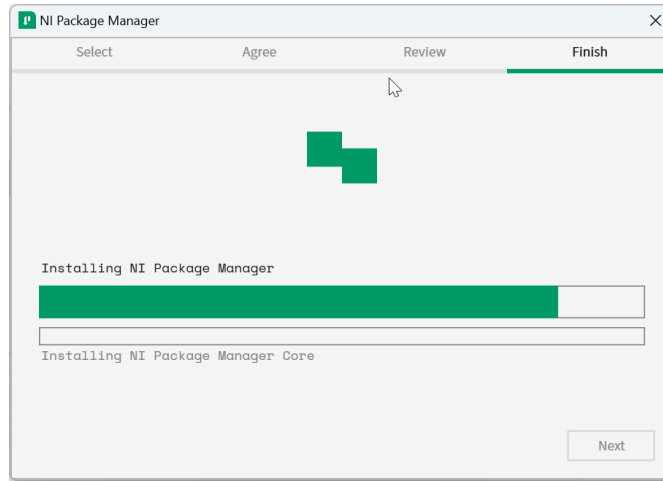
Windows Hızlı Başlatma roboRIO'yu görüntülemek için gereken NI sürücülerinde sorunlara neden olabileceğinden, bu ekranı olduğu gibi bırakmanız önerilir. Devam edin ve tıklayın *Next*.

NI Package Manager İncelemesi



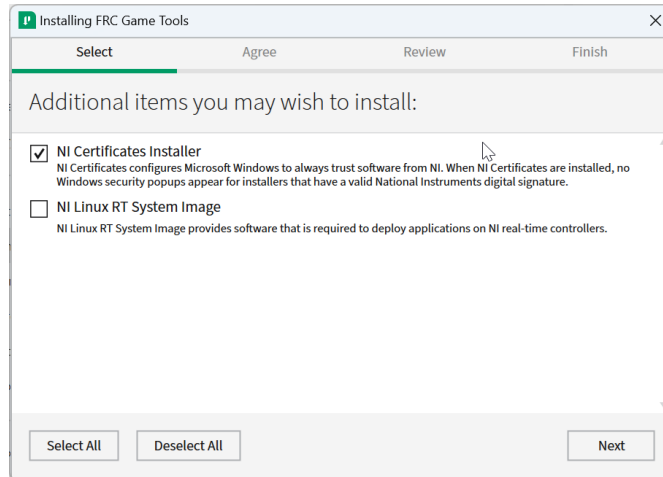
Bu ekranı görürseniz, tıklayın *Next*.

NI Package Manager Kurulumu



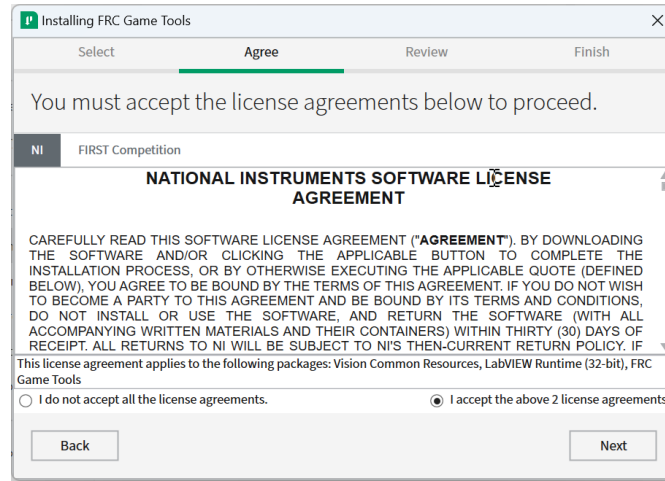
NI Paket Yöneticisinin kurulum ilerlemesi bu pencerede izlenecektir.

Ek Yazılım



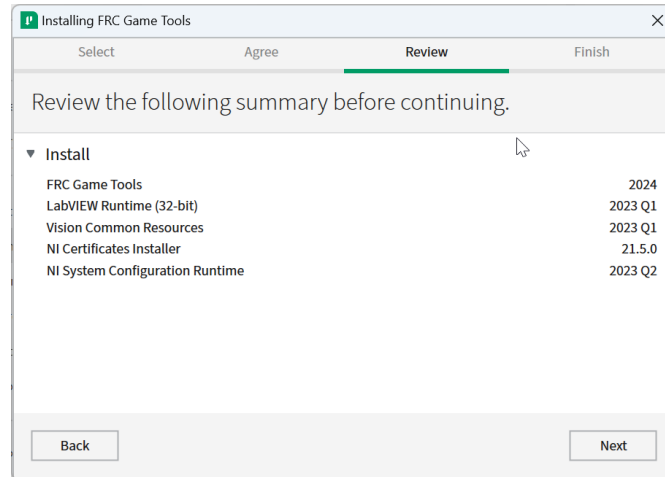
Bu ekranı görürseniz, tıklayın *Next*.

Lisans Sözleşmeleri



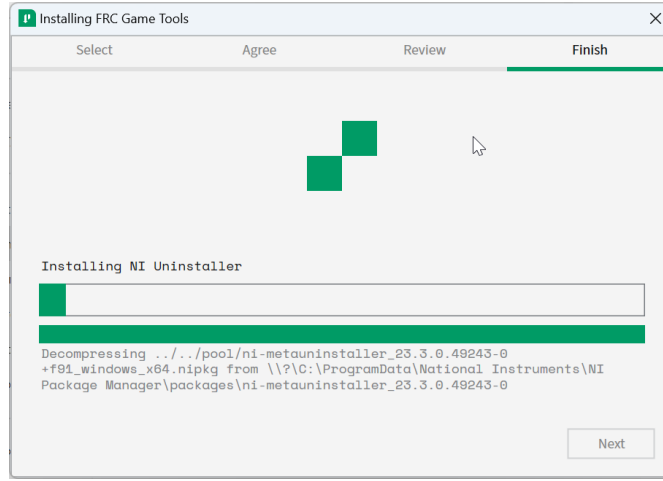
Seçin *I accept...* ve ardından tıklayın *Next*.

Değerlendirme Özeti



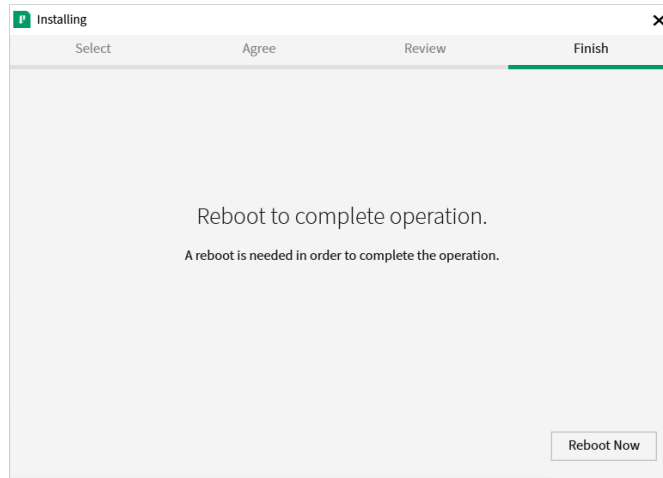
Tıklayın *Next*.

İlerleme Detayları



Bu ekran kurulum sürecini gösterir, devam edin ve bittiğinde *Next* ye basın.

3.3.4 Kurulumu Tamamlamak için Yeniden Başlatın



İstenirse, açık programları kapattıktan sonra *Reboot Now* seçeneğini seçin.

3.4 WPILib Kurulum Kılavuzu

This guide is intended for Java and C++ teams. LabVIEW teams can skip to [FRC için LabVIEW'i yükleme \(yalnızca LabVIEW\)](#). 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 Önkoşullar

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.

Uyarı: 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 ✎ 🗑

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)

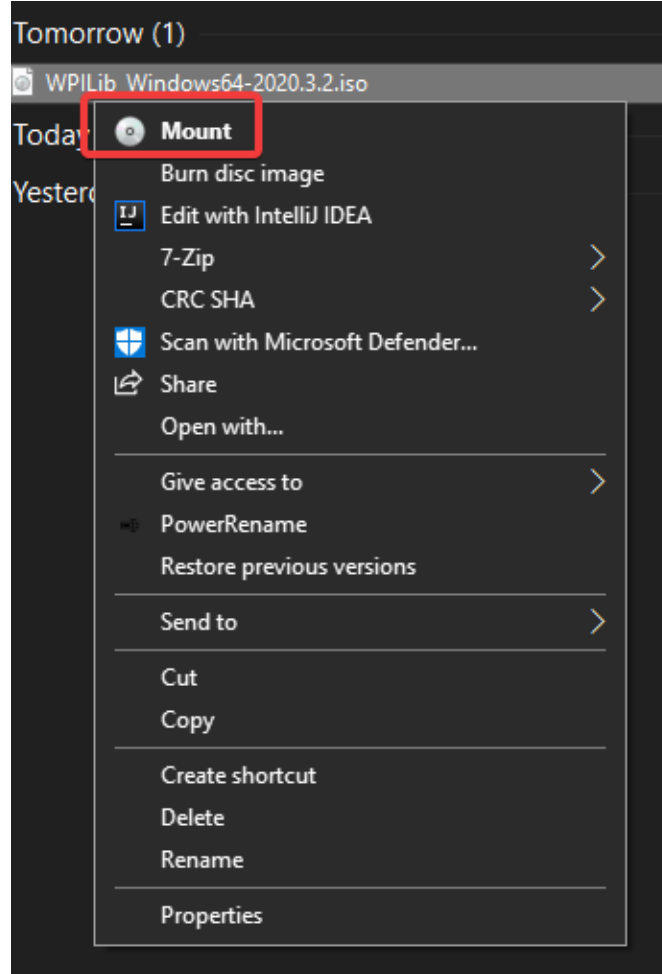
Then click on the correct binary for your OS and architecture to begin the download.

3.4.3 Yükleyiciyi Çıkarma

WPILib yükleyicisini indirdiğinizde, Windows için `iso`, Linux için `.tar.gz` disk görüntü dosyası ve MacOS için bir DMG olarak dağıtılır.

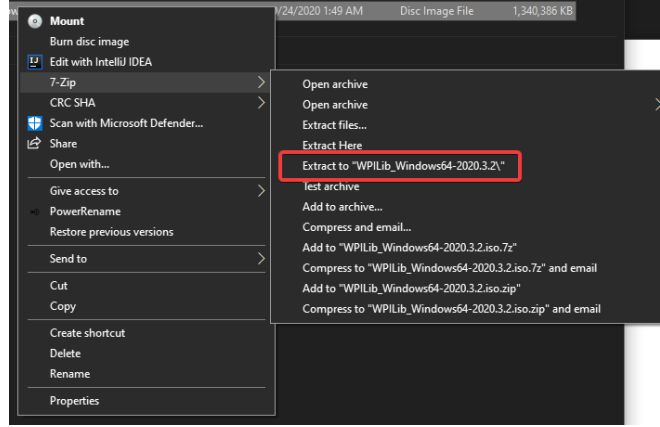
Windows 10+

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



Not: 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`.

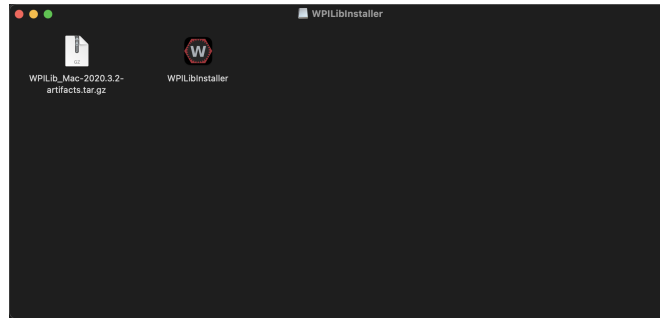
Not: After launching the installer, Windows may display a window titled “Windows protected your PC”. Click *More info*, then select *Run anyway* to run the installer.

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.

Önemli: 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 kullanıcıları, indirilen DMG ye çift tıklayabilir ve ardından uygulamayı başlatmak için `WPILibInstaller` ögesini seçebilir.



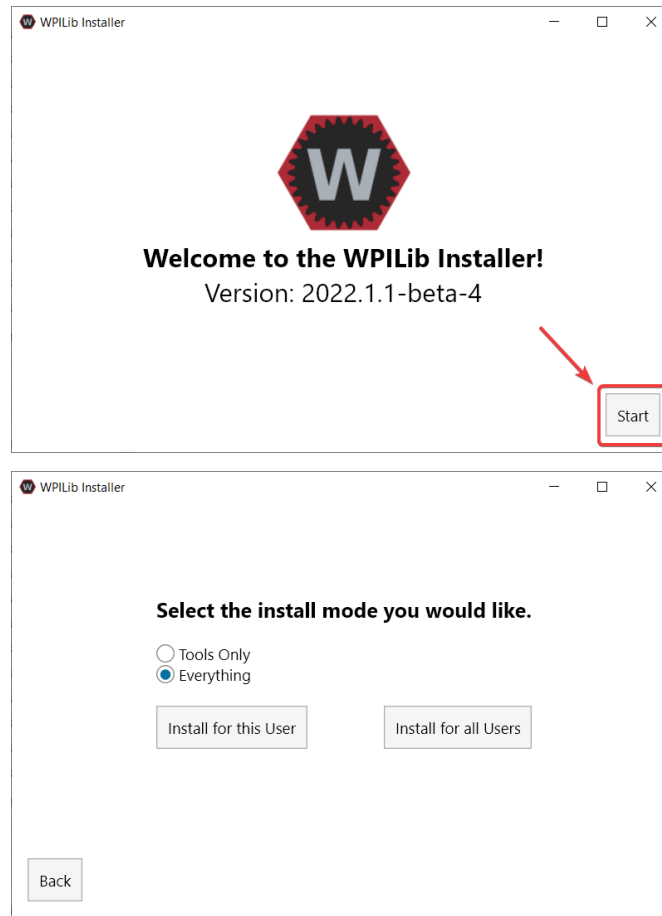
Linux

Linux kullanıcıları indirilen `.tar.gz` dosyasını çıkartmalı ve ardından WPILibInstaller uygulamasını başlatmalıdır. Ubuntu, dosya gezginindeki yürütülebilir dosyaları paylaşılan kitaplıklar olarak değerlendirir, bu nedenle çift tıklamak onları çalıştırmaz. Aşağıdaki komutları, `<version>` yerine yüklediğiniz sürümle değiştirerek bir terminalde çalıştırın.

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

3.4.4 Yükleyiciyi Çalıştırma

Yükleyiciyi açtığınızda, aşağıdaki ekran ile karşılaşacaksınız. Devam edin ve basın *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.

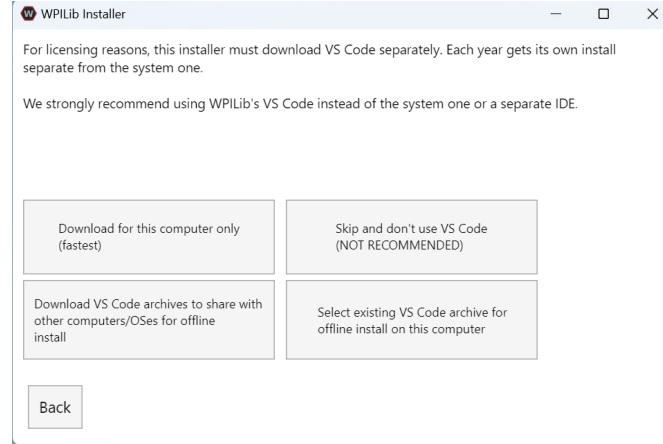
İki düğme göreceksiniz *Install for this User* ve *Install for all Users*. *Install for this User* yalnızca mevcut kullanıcı hesabına yükler ve yönetici ayrıcalıkları gerektirmez. Ancak, *Install*

for all Users , tüm sistem hesapları için araçları yükler ve *will-yönetici erişimi gerektirir *. Install for all Users , macOS ve Linux için bir seçenek değildir.

Not: If you select Install for all Users, Windows will prompt for administrator access through UAC during installation.

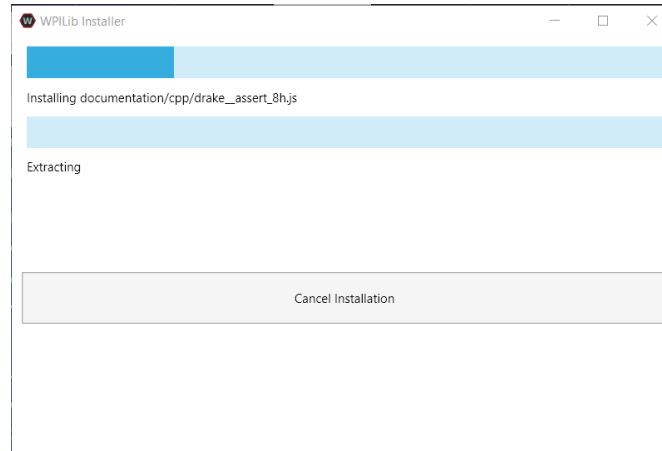
Size uygun seçeneği seçin ve aşağıdaki kurulum ekranını göreceksiniz.

Bu sonraki ekran VS Kodunun indirilmesini içerir. Ne yazık ki, lisanslama nedenlerinden dolayı VS Code yükleyiciyle birlikte paketlenemez.

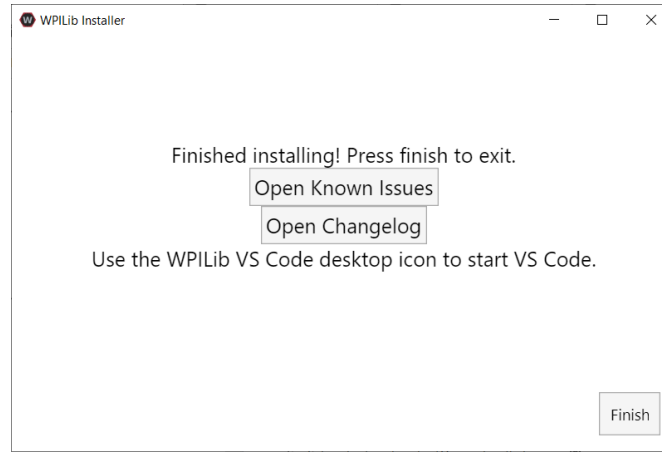


- Download for this computer only
 - Bu, VS Code'u yalnızca aynı zamanda en küçük indirme olan mevcut platform için indirir.
- Skip and don't use VS Code
 - VS Code'u yüklemeyi atlar. Gelişmiş kurulumlar veya konfigürasyonlar için kullanışlıdır. Genellikle tavsiye edilmez.
- Select existing VS Code archive for offline install on this computer
 - Bu seçeneğin seçilmesi, daha önce yükleyici tarafından indirilmiş VS Code'un önceden var olan bir zip dosyasını seçmenize olanak tanıyan bir komut istemini açacaktır. Bu seçenek, makinenize VS Code'un önceden yüklenmiş bir kopyasını seçmenize **not-izin vermez**.
- Create VS Code archives to share with other computers/OSes for offline install
 - Bu seçenek, tüm platformlar için VS Code'un bir kopyasını indirir ve kaydeder; bu, yükleyicinin kopyasını paylaşmak için yararlıdır.

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.



Kurulum tamamlandıktan sonra, bitmiş ekran ile karşılaşacaksınız.



Önemli: 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!

Tebrikler, WPILib geliştirme ortamı ve araçları artık bilgisayarınızda yüklü! Yükleyiciden çıkmak için Bitir'e basın.

3.4.5 Yükleme sonrası

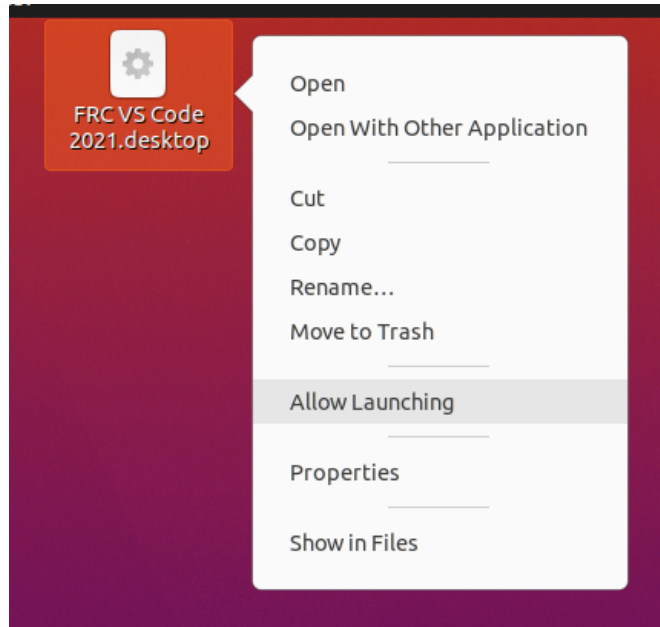
Bazı işletim sistemleri, kurulumu tamamlamak için bazı son eylemler gerektirir.

macOS

Kurulumdan sonra, yükleyici WPILib VS Code klasörünü açar. VS Code uygulamasını yuvaya sürükleyin. WPILibInstaller görüntüsünü masaüstünden çıkarın.

Linux

Bazı Linux sürümleri (ör. Ubuntu 20.04) masaüstü kısayoluna başlatma yeteneği vermenizi gerektirir. Masaüstü simgesine sağ tıklayın ve Başlatmaya İzin Ver'i seçin.

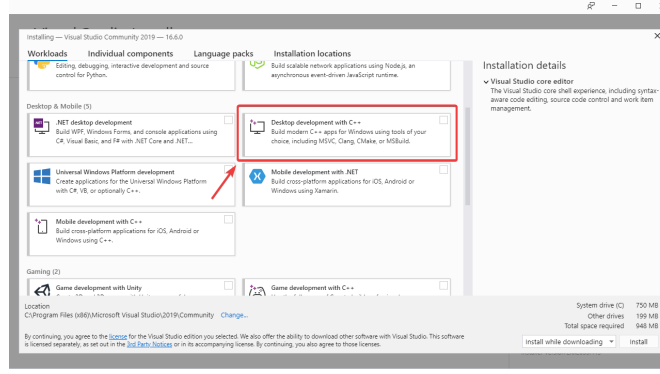


Not: Installing desktop tools and rebooting will create a folder on the desktop called YYYY WPILib Tools, where YYYY is the current year. Desktop tool shortcuts are not available on Linux and macOS.

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 Ne Yüklenir?

Çevrimdışı Yükleyici aşağıdaki bileşenleri yükler:

- **Visual Studio Code** - 2019 ve sonraki robot kodu geliştirme için desteklenen IDE. Çevrimdışı yükleyici, makinenizde zaten VS Kodunuz olsa bile, WPILib geliştirme için VS Code'un ayrı bir kopyasını ayarlar. Bu, WPILib kurulumunun çalışmasını sağlayan bazı ayarların, diğer projeler için VS Code kullanırsanız mevcut iş akışlarını bozabileceği için yapılır.
- ****C ++ Derleyici **** - roboRIO için C ++ kodu oluşturmaya yönelik araç zincirleri
- ****Gradle **** - C ++ veya Java robot kodu oluşturmak/dağıtmak için kullanılan belirli Gradle sürümü
- ****Java JDK / JRE **** - Java robot kodu oluşturmak ve Java tabanlı Araçlardan herhangi birini (Gösterge Tabloları, vb.) Çalıştırmak için kullanılan Java JDK / JRE'nin belirli bir sürümü. Bu, mevcut herhangi bir JDK kurulumuyla yan yana bulunur ve JAVA_HOME değişkeninin üzerine yazmaz
- **WPILib Tools** - SmartDashboard, Shuffleboard, RobotBuilder, OutlineViewer, PathWeaver, Glass, SysId, Data Log Tool, roboRIO Team Number Setter, AdvantageScope
- **WPILib Bağımlılıkları** - OpenCV vb.
- **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 Kaldırma

WPILib, bu yılki WPILib'i kurmadan önce önceki bir sürümü kaldırmaya gerek kalmaması için farklı yıllar boyunca farklı klasörlere yüklenecek şekilde tasarlanmıştır. Ancak, istenirse aşağıdaki talimatlar WPILib'i kaldırmak için kullanılabilir.

Windows

1. Delete the appropriate wpilib folder (c:\Users\Public\wpilib\YYYY where YYYY is the year to uninstall)
2. C:\Users\Public\Public Desktop konumundaki masaüstü simgelerini silin.

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 Sorun giderme

Yükleyicinin başarısız olması durumunda, lütfen yükleyici havuzunda bir sorun açın. Burada bir bağlantı mevcuttur <<https://github.com/wpilibsuite/wpilibinstaller-avalonia>> . Yükle-yici, hatanın nedeni hakkında bir mesaj vermelidir, lütfen bunu sorununuzun açıklamasına ekleyin.

3.5 Python Installation Guide

This guide is intended for Python teams. Java and C++ teams can skip to *WPILib Kurulum Kılavuzu*. LabVIEW teams can skip to *FRC için LabVIEW'i yükleme (yalnızca 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

Not: 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.

Uyarı: 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 you can install to the user site-packages directory:

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

macOS

Not: 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

Not: 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.

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

Uyari: 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.

Ayrıca bakınız:

[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.

Ayrıca bakınız:

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

Not: 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 Sonraki adımlar

Tebrikler! 2. adımı tamamladınız ve şimdi çalışan bir yazılım geliştirme ortamına sahip olabilirsiniz! Bu öğreticinin 3. adımı, donanımı programlayabilmeniz için güncellemeyi kapsar, 4. Adım ise VS Code Integrated Development Environment (IDE) 'de bir robotun programlanmasını gösterir. Daha fazla bilgi için, IDE'yi tanımak için *[VS Code section](#)* okuyabilirsiniz.

Okunması tavsiye edilen belirli makaleler şunlardır:

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

Ek olarak, ekibinizin robotu için geçerli olan ekstra konfigürasyon yapmanız gerekebilir. Lütfen gerekli belgeleri bulmak için arama özelliğini kullanın.

Not: Üçüncü taraf CAN motor denetleyicileri kullanan ekiplerin, kodlamak için fazladan adımlar gerektiğinden, *[Installing 3rd Party Libraries](#)* makalesine bakması önemlidir. bu cihazlar için.

Adım 3: Robotunuzu Hazırlama

4.1 Imaging your roboRIO 2

Not: 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.

Önemli: 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.

Uyarı: 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

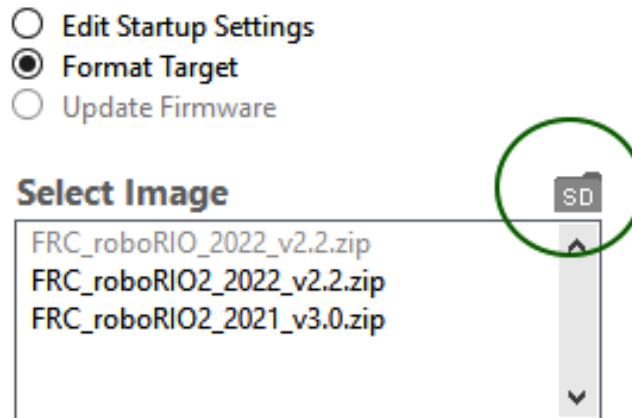
Distribution Panel or REV Power Distribution Hub. Make sure the power wires to the roboRIO are secure and that the connector is secure firmly to the roboRIO (4 total screws to check).

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.

Not: 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*

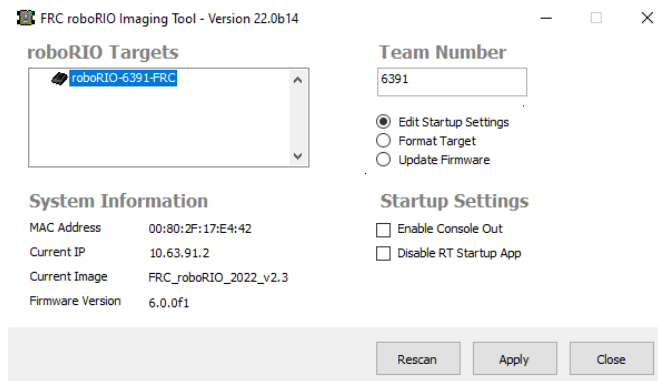
Writing the image with Raspberry Pi Imager

- 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*

Uyarı: 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 roboRIO 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

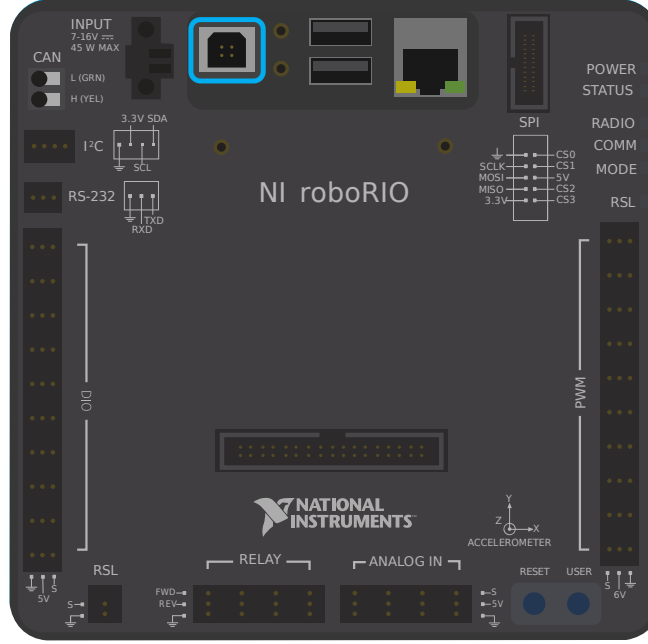
Uyarı: RoboRIO'nuzu görüntülemeyen önce, [FRC Game Tools](#) kurulumunu tamamlamış olmanız gerekir. Ayrıca roboRIO gücünün Power Distribution Paneline uygun şekilde kablolanmış olması gerekir. RoboRIO'ya giden güç kablolarının sağlam olduğundan ve konektörün roboRIO'ya sıkıca sabitlendiğinden emin olun (kontrol edilecek toplam 4 vida).

Not: The roboRIO 2 uses different imaging instructions. The imaging instructions for the NI roboRIO 2.0 are [here](#).

4.2.1 RoboRIO'yu yapılandırma

RoboRIO Imaging Tool, roboRIO'nuzu en son yazılımla formatlamak için kullanılacaktır.

USB Bağlantısı



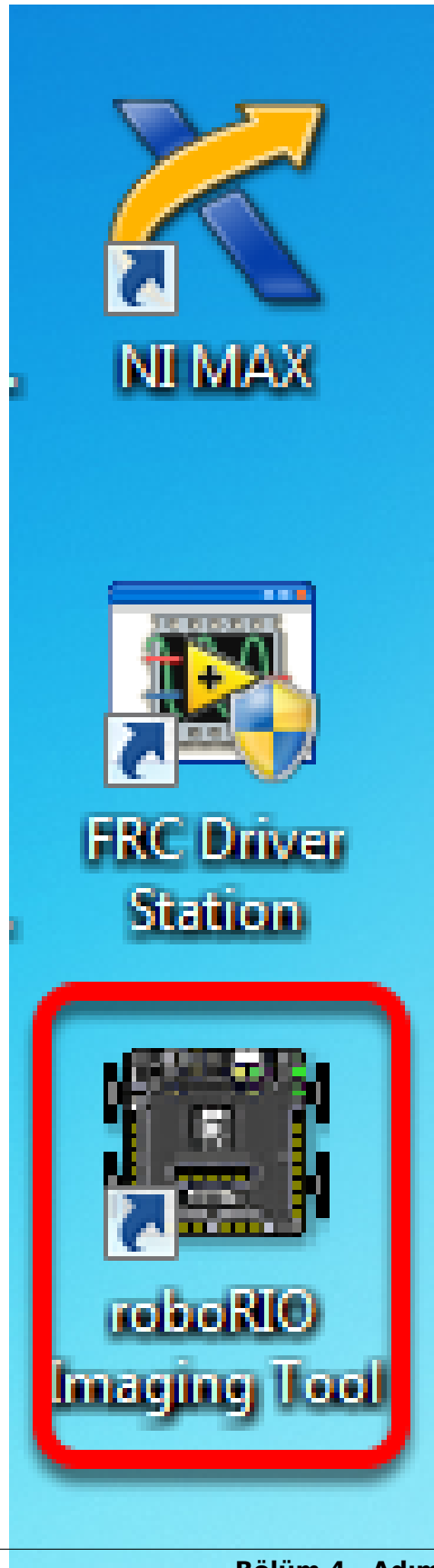
RoboRIO USB Aygıtı bağlantı noktasından bilgisayara bir USB kablosu bağlayın. Bu, genellikle bir yazıcı USB kablosu olarak bulunan, USB Type A erkek (standart PC ucu) ile Tip B erkek kablo (2 kesik köşeli kare) gerektirir.

Not: RoboRIO yalnızca USB bağlantısı aracılığıyla imajlanmalıdır. Ethernet bağlantısını kullanarak imaj atmaya çalışmanız önerilmez.

Sürücü Kurulumu

Aygıt sürücüsü otomatik olarak kurulmalıdır. Ekranın sağ alt köşesinde bir “New Device” açılır penceresi görürseniz, devam etmeden önce sürücü kurulumunun tamamlanmasını bekleyin.

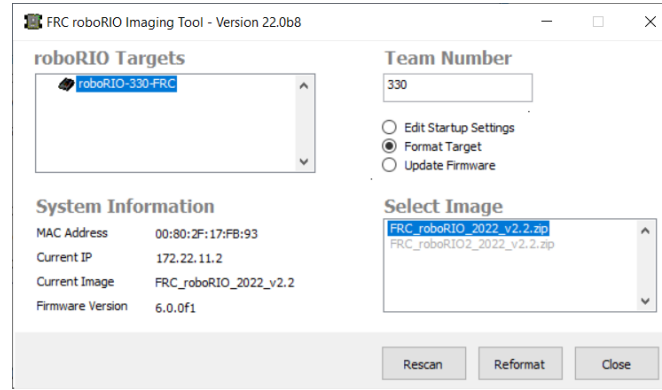
4.2.2 Imaging Tool Aracını Başlatma



RoboRIO imaj atma aracı ve en son imaj dosyası, NI FRC|reg| Game Tool ile yüklenir. Masaüstündeki kısayola çift tıklayarak görüntüleme aracını başlatın. RoboRIO'nuzu görüntülemeye zorluk yaşıyorsanız, simgeye sağ tıklayıp bunun yerine Yönetici Olarak Çalıştır'ı seçmeniz gerekebilir.

Not: The roboRIO imaging tool is also located at C:\Program Files (x86)\National Instruments\LabVIEW 2023\project\roboRIO Tool

4.2.3 roboRIO Imaging Tool - İmaj atma aracı

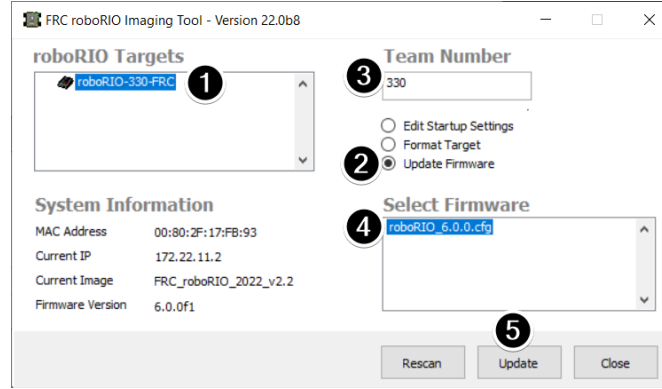


Başlatıldıktan sonra roboRIO Imaging Tool, mevcut roboRIO'ları tarayacak ve sol üst kutuda bulunanları gösterecektir. Sol alttaki kutu, halihazırda seçili roboRIO için bilgi ve ayarları gösterecektir. Sağdaki bölme, roboRIO ayarlarını değiştirmek için kontroller içerir:

- **Edit Startup Settings** - Bu seçenek, roboRIO'yu imajlamadan roboRIO'nun başlangıç ayarlarını (sağ bölmedeki ayarlar) yapılandırmak istediğinizde kullanılır.
- **Format Target** - Bu seçenek, roboRIO'ya yeni bir imaj yüklemek (veya mevcut görüntüyü yeniden atmak) istediğinizde kullanılır. Bu en yaygın seçenektir.
- **Update Firmware** - Bu seçenek, roboRIO aygıt yazılımını güncellemek için kullanılır. Bu sezon için, görüntüleme aracı roboRIO aygıt yazılımının 5.0 veya üzeri sürümünü gerektirecek.

Updating Firmware - Donanım Yazılımını Güncelleme

Uyarı: 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 donanım yazılımının 2019 veya sonraki görüntüyle çalışması için en az v5.0 olması gerekir. RoboRIO'nuz en az 5.0 sürümüyse (görüntüleme aracının sol alt kısmında gösterildiği gibi) güncelleme yapmanıza gerek yoktur.

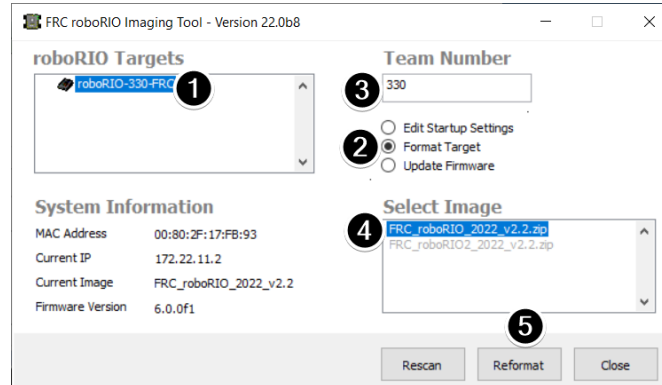
Not: roboRIO firmware has had different version numbering schemes over the years. It isn't necessary to update the firmware if it has version 5, 6, 8, 22.5, 23.5 or variations of those version numbers (e.g. 8.8.0f0 is a variation of 8). The firmware is only utilized in *safe mode*, it is not used in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

roboRIO donanım yazılımını güncellemek için:

1. Sol üst bölmede roboRIO'nuzun seçildiğinden emin olun.
2. Select *Update Firmware* in the top right pane
3. Enter a team number in the *Team Number* box
4. Sağ alttaki en son üretici yazılımı dosyasını seçin
5. Click the *Update* button

4.2.4 roboRIO'ya İmaj atmak

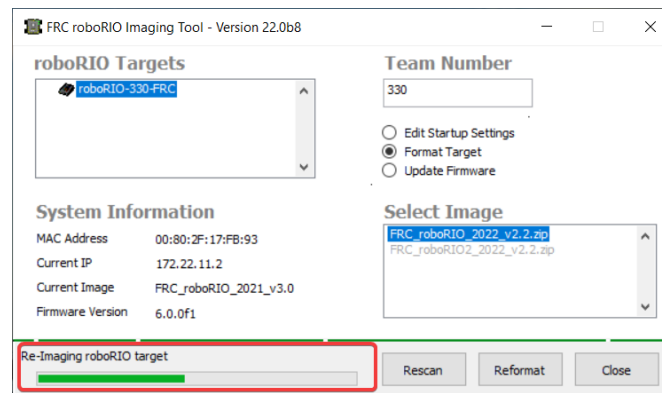
Uyarı: The roboRIO image is different then the firmware, and must be updated yearly.



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

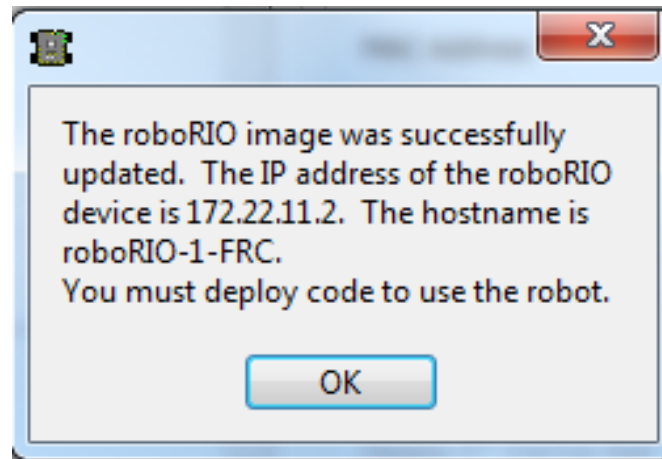
1. Sol üst bölmede roboRIO'nun seçildiğinden emin olun
2. Select *Format Target* in the right pane
3. Enter a team number in the *Team Number* box
4. Kutudaki en son imaj sürümünü seçin.
5. Click *Reformat* to begin the imaging process.

4.2.5 İmaj Atma İlerlemesi



İmaj atma işlemi yaklaşık 3-10 dakika sürecektir. Pencerenin sol alt kısmındaki ilerleme çubuğu, ilerlemeyi gösterecektir.

4.2.6 İmaj Atma İşlemi Tamamlandı



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 Sorun giderme

roboRIO'nuza imaj atamıyorsanız, sorun giderme adımları şunları içerir:

- RoboRIO Imaging Tool'u başlatmak için Masaüstü simgesine sağ tıklayarak Yönetici-Administrator olarak çalıştırmayı deneyin.
- RoboRIO web sayfasına bir web tarayıcısı ile ``http://172.22.11.2/`` adresinden erişmeyi deneyin ve / veya NI ağ adaptörünün Kontrol Panelindeki Ağ Adaptörleri listenizde göründüğünü doğrulayın. Değilse, NI FRC Game Tools'u yeniden yüklemeyi veya farklı bir PC deneyin.
- : ref:Diğer tüm ağ bağdaştırıcılarını devre dışı bırak <docs/networking/networking-introduction/roborio-network-troubleshooting:Disabling Network Adapters>
- Güvenlik duvarınızın kapalı olduğundan emin olun.
- 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*
- Reset düğmesini en az 5 saniye basılı tutarak roboRIO'yu Safe Mode - Güvenli Modda başlatmayı deneyin.
- Try a different USB Cable
- Farklı bir bilgisayar deneyin
- 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 Radyonuzu Programlama

Bu kılavuz size FRC ® FRC olaylarının dışında kullanım için robotunuzun kablosuz köprüsünü yapılandırmak için Radyo Yapılandırma Yardımcı Programı yazılımı.

4.3.1 Gereksinimler

FRC Radio Configuration Utility Programı, makinenizdeki ağ ayarlarını yapılandırmak için Yönetici ayrıcalıkları gerektirir. Program gerekli ayrıcalıkları otomatik olarak istemelidir (Yönetici olmayan bir hesaptan çalıştırılırsa bir parola gerektirebilir), ancak sorun yaşıyorsanız, bir Yönetici hesabından çalıştırmayı deneyin.

Aşağıdaki bağlantılardan en son FRC Radio Configuration Utility Yükleyicisini indirin:

[FRC Radio Configuration 24.0.1](#)

[FRC Radio Configuration 24.0.1 Israel Version](#)

Not: _IL sürümü İsrail ekipleri içindir ve İsrail’de kullanım için kısıtlanmış kanallara sahip OM5PAC ürün yazılımının bir sürümünü içerir.

Yazılımı kullanmaya başlamadan önce:

1. *Diğer tüm ağ bağdaştırıcılarını devre dışı bırak*
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.

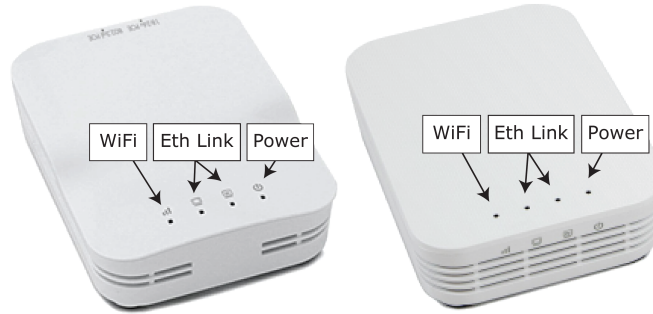
Uyarı: OM5P-AN ve AC, D-Link DAP1522 ile aynı elektrik fişini kullanır, ancak bunlar 12V radyolardır. Radyoyu VRM üzerindeki 12V 2A terminallerine bağlayın (merkez pin pozitif).

4.3.2 Uygulama Notları

Varsayılan olarak, Radyo Yapılandırma Yardımcı Programı, telsizi kablosuz arayüz üzerinden telsizden çıkan trafikte 4Mbps bant genişliği sınırını uygulayacak şekilde programlayacaktır. Ev yapılandırmasında (AP modu) bu, istemci başına bir limit değil, toplamdır. Bu, birden fazla istemciye video akışının tavsiye edilmediği anlamına gelir.

Bu program Windows 7, 8 ve 10’da test edilmiştir. Diğer işletim sistemlerinde çalışabilir, ancak test edilmemiştir.

Programlanmış Yapılandırma



Radio Configuration Utility Programı, çalıştırıldığında radyoya bir dizi yapılandırma ayarı programlar. Bu ayarlar tüm modlarda (etkinliklerde dahil) telsiz için geçerlidir. Bunlar şunları içerir:

- ``10.TE.AM.1`` için statik bir IP ayarlayın
- Gelecekteki programlama ihtiyaçları için 192.168.1.1 ‘in kablolu tarafında alternatif bir IP ayarı.
- Kablolu bağlantı noktalarını köprüleyin, böylece birbirlerinin yerine kullanılabilirler.
- Yukarıdaki grafikte belirtilen LED konfigürasyonu.

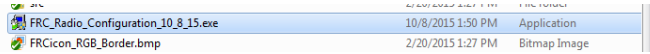
- Kablosuz arayüzün giden tarafında 4Mb/s bant genişliği sınırı (evde kullanım için devre dışı bırakılabilir)
- İç paket önceliklendirme için QoS kuralları (dahili arabelleği ve bant genişliği sınırına ulaşıldığında hangi paketlerin dikkate alınmayacağını etkiler). Bu kurallar:
 - Robot Kontrolü ve Durumu (UDP 1110, 1115, 1150)
 - Robot TCP & *NetworkTables* (TCP 1735, 1740)
 - Toplu (Diğer tüm trafik). (BW sınırı devre dışı bırakılırsa devre dışı bırakılır)
- *DHCP* server enabled. Serves out:
 - 10.TE.AM.11 - 10.TE.AM.111 kablolu tarafta
 - 10.TE.AM.138 - 10.TE.AM.237 on the wireless side
 - 255.255.255.0 alt ağ maskesi
 - Yayın adresi 10.TE.AM.255
- DNS server enabled. DNS server IP and domain suffix (.lan) are served as part of the DHCP.

Yalnızca evde:

- SSID, birden fazla ağı ayırt etmek için ekip numarasına eklenmiş bir “Robot Adı” içerebilir.
- Güvenlik duvarı seçeneği, saha güvenlik duvarı kurallarını taklit etmek için etkinleştirilebilir (açık bağlantı noktaları Oyun Kılavuzunda bulunabilir)

Uyarı: Yapılandırmayı manuel olarak değiştirmek mümkün değildir.

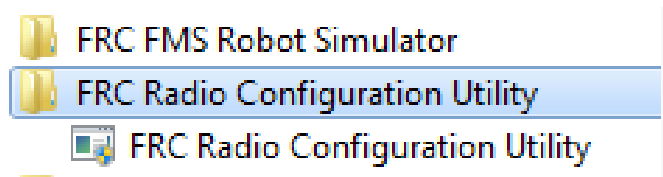
4.3.3 Yazılımı Yükleyin



Yükleyiciyi başlatmak için FRC_Radio_Configuration_VERSION.exe üzerine çift tıklayın. Kurulumu tamamlamak için talimatları izleyin.

Kurulum sırasında eğer zaten mevcut değilse Npcap kurulumunu içerecektir. Npcap yükleyici, yüklemeyi yapılandırmak için bir dizi onay kutusu içerir. Seçenekleri varsayılanlar olarak bırakmalısınız.

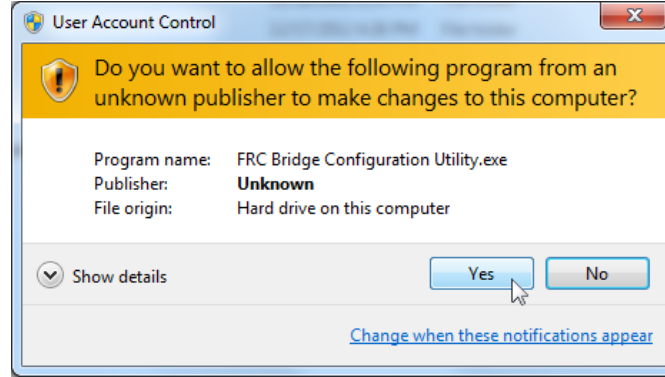
4.3.4 Yazılımı başlatın



Programı başlatmak için Başlat menüsünü veya masaüstü kısayolunu kullanın.

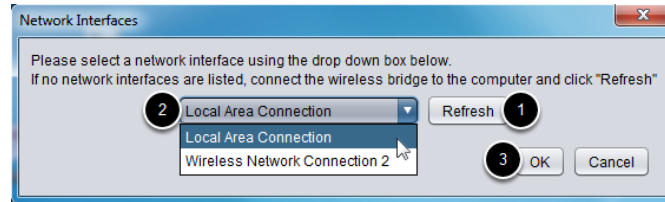
Not: Programı bulmanız gerekirse, C:\Program Files (x86)\FRC Radio Configuration Utility konumuna kurulum. 32 bit makineler için yol ``C:\Program Files\FRC Radio Configuration Utility``dır.

4.3.5 Eğer sorulursa programın değişiklik yapmasına izin verin



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

4.3.6 Ağ arayüzünü seçin.



Yapılandırma yardımcı programının kablosuz cihaz ile iletişim kurmak için kullanacağı ethernet arayüzünü seçmek için açılır pencereyi kullanın. Windows makinelerde, ethernet arayüzleri tipik olarak "Yerel Alan Bağlantısı" olarak adlandırılır. Yapılandırma programı kablosuz bağlantı üzerinden bir köprü programlayamaz.

1. Hiçbir ethernet arabirimi listelenmemişse, kullanılabilir arabirimleri yeniden taramak için *Refresh* ye tıklayın.
2. Açılır listeden kullanmak istediğiniz arayüzü seçin
3. Tıklayın *OK*.

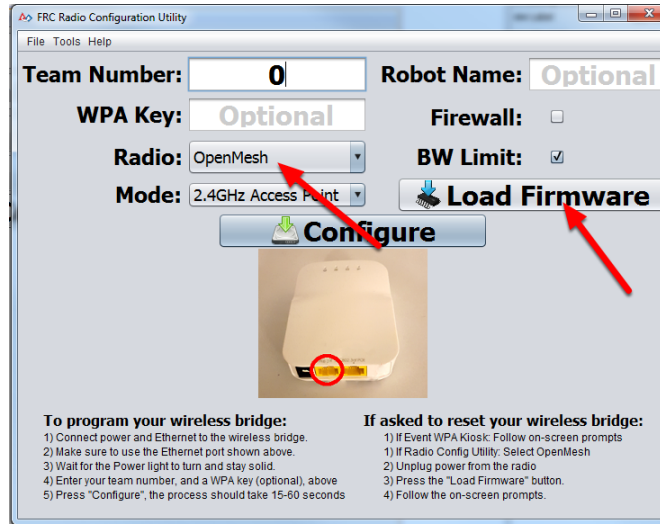
4.3.7 Açık Mesh Firmware Notu

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.

Ürün yazılımını güncellemeniz veya yeniden yüklemeniz gerekmiyorsa, sonraki adımı atlayın.

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

4.3.8 Open Mesh radyoya FRC Donanım Yazılımını Yükleme



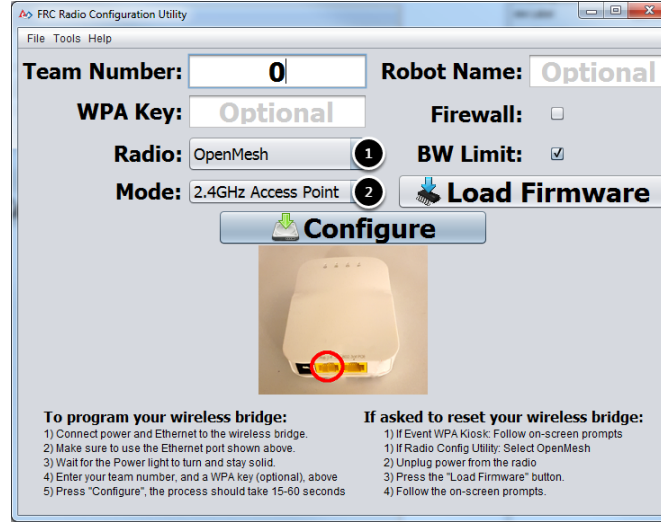
FRC aygıt yazılımını yüklemeniz (veya cihazı sıfırlamanız) gerekirse, bunu FRC Radio Configuration Utility Programını kullanarak yapabilirsiniz.

1. Yazılımı kurmak, programı başlatmak ve Ethernet arayüzünü seçmek için yukarıdaki talimatları izleyin.
2. Radio açılır menüsünde Open Mesh radyosunun seçildiğinden emin olun.
3. Radyonun bilgisayara Ethernet üzerinden bağlı olduğundan emin olun.
4. Radyodan gücü kesin. (Bir PoE kablosu kullanıyorsanız, bu aynı zamanda PC'ye giden kabloyu çıkarabilirsiniz, buda doğru bir yönetimdir)
5. Firmware Yükle düğmesine basın
6. İstendiğinde, radyo gücünü bağlayın. Yazılım radyoyu algılamalı, aygıt yazılımını yüklemeli ve tamamlandığında sizi uyarmalıdır.

Uyarı: NPF adı ile ilgili bir hata görürseniz, radyoyu programlamak için kullanılan adaptör dışındaki tüm adaptörleri devre dışı bırakmayı deneyin. Yalnızca bir adaptör bulunursa, araç onu kullanmaya çalışmalıdır. Daha fazla bilgi için : ref: *Ağ Adaptörlerini Devre Dışı Bırakma* <docs/networking/networking-introduction/roborio-network-troubleshooting:Disabling Network Adapters> içindeki adımlara bakın.

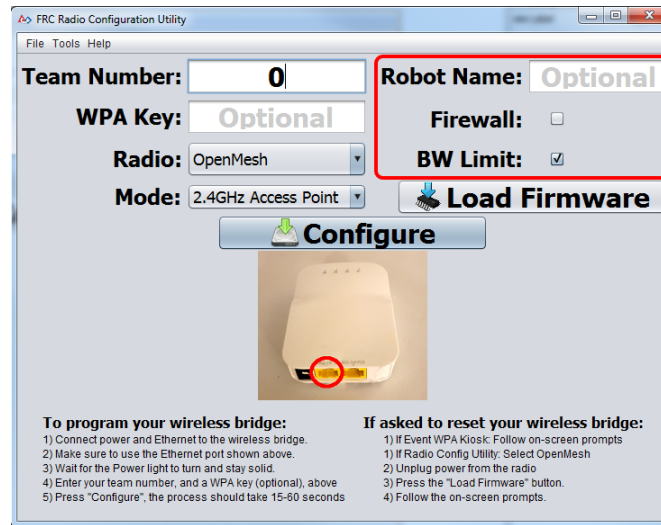
Teams may also see this error with Operating Systems configured for languages other than US English. If you experience issues loading firmware or programming on a foreign language OS, try using an English OS, such as on the KOP provided PC or setting the Locale setting to "en_us" as described on [this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

4.3.9 Radyo ve Çalışma Modunu Seçin



1. Açılır listeyi kullanarak yapılandırmakta olduğunuz radyoyu seçin.
2. Yapılandırmak istediğiniz işletim modunu seçin. Çoğu durumda, varsayılan olarak 2,4 GHz Erişim Noktası seçimi yeterli olacaktır. Bilgisayarlarınız destekliyorsa, 5GHz birçok ortamda daha az yoğun olduğu için 5GHz AP modu önerilir.

4.3.10 Seçenekler'i seçin



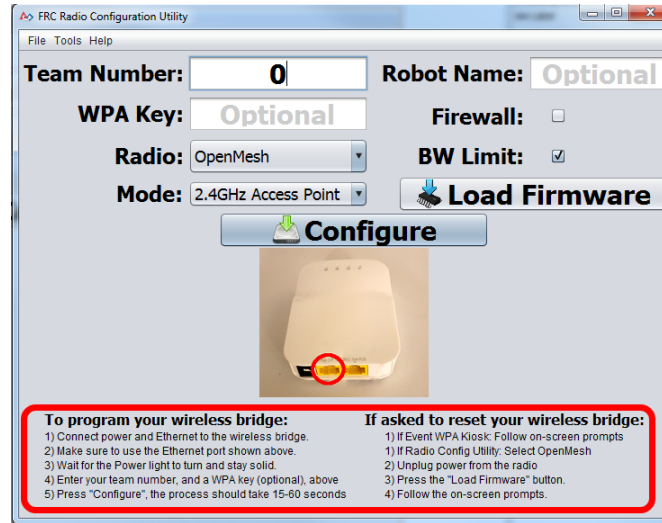
Seeneklerin varsayılan deęerleri, oęu takımın kullanım durumuyla eőleőecek őekilde seilmiőtir, ancak bu seenekleri kendi senaryonuza gre zelleőtirmek isteyebilirsiniz:

1. **Robot Adı:** Bu, telsiz tarafından kullanılan SSID'ye eklenen bir dizedir. Bu, aynı ekip numarasına sahip birden ok aęa sahip olmanıza ve bunları yine de ayırt edebilmenize olanak tanır.
2. **Firewall-Gvenlik Duvarı:** Bu kutu iőaretlenirse, radyo gvenlik duvarı, FRC alanında bulunan gvenlik duvarının baęlantı noktası engelleme davranıőını taklit etmeye alıőacak őekilde yapılandırılacaktır. Aık baęlantı noktalarının listesi iin ltfen FRC Oyun Kılavuzuna bakın.
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.

Not: Gvenlik Duvarı ve BW Sınırı yalnızca Aık Aę telsizleri iin geerlidir. Bu seeneklerin D-Link radyoları zerinde hibir etkisi yoktur.

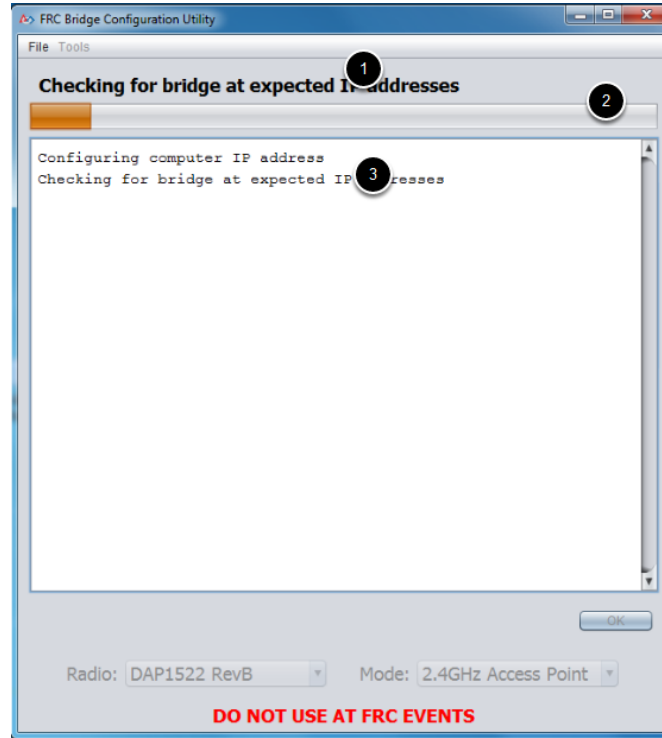
Uyarı: "Firewall-Gvenlik Duvarı" seeneęi, modemi alan gvenlik duvarını taklit edecek őekilde yapılandırır. Bu, bu seenek etkinleőtirildięinde kodu kablosuz olarak daęıtamayacaęınız anlamına gelir. Bu, yarıőmalarda bulunabilecek engellenmiő baęlantı noktalarını simle etmek iin kullanıőlıdır.

4.3.11 Yapılandırma İőleminin Baőlatılması



Kablosuz kprnz hazırlamak, kprnn yapılandırılacaęı ayarları girmek ve yapılandırma srecini baőlatmak iin ekrandaki talimatları izleyin. Bu ekran talimatları, seilen kpr modeli ve iőletim moduna uyacak őekilde gncellenir.

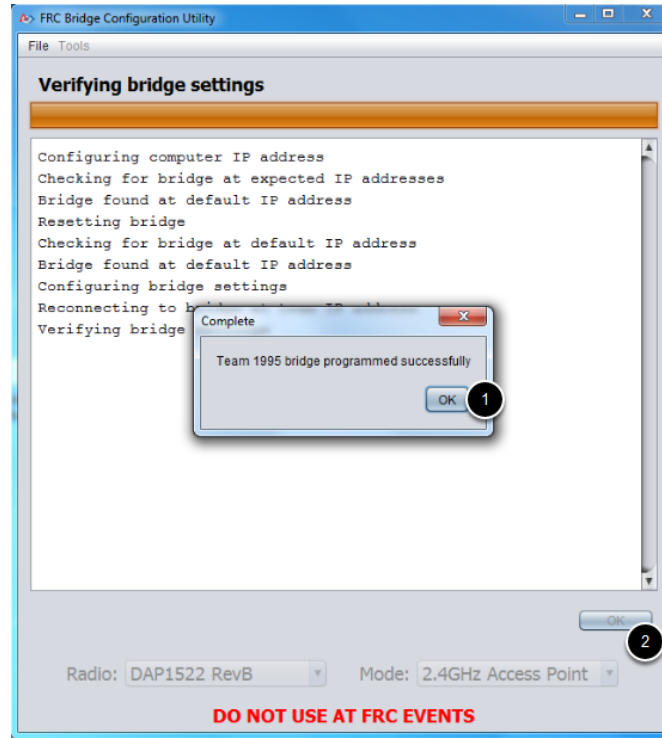
4.3.12 Yapılandırma İlerlemesi



Yapılandırma işlemi boyunca pencere şunları gösterecektir:

1. Şu anda yürütülen adım.
2. Yapılandırma sürecinin genel ilerlemesi.
3. Şimdiye kadar tüm adımlar uygulandı.

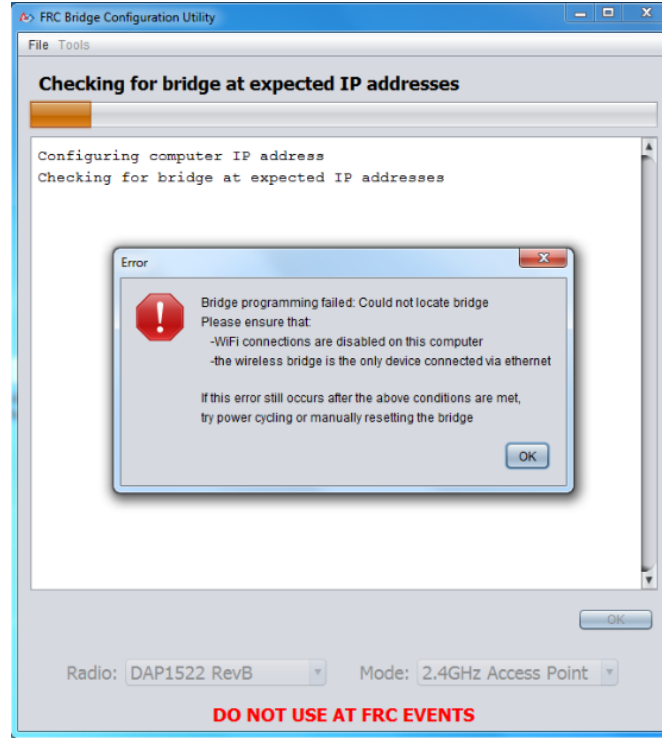
4.3.13 Yapılandırma Tamamlandı



Yapılandırma tamamlandığında:

1. Diyalog penceresinde guilabel: OK tuşuna basın.
2. Ayarlar ekranına dönmek için ana pencerede :guilabel: OK tuşuna basın.

4.3.14 Yapılandırma Hataları



Yapılandırma işlemi sırasında bir hata oluşursa, sorunu gidermek için hata mesajındaki talimatları izleyin.

4.3.15 Sorun giderme

- *Disable all other network adapters.*
- Make sure you wait long enough that the power light has stayed solid for 10 seconds.
- Make sure you have the correct network interface, and only one interface is listed in the drop-down.
- Make sure your firewall is turned off.
- Doğrudan bilgisayarınızdan kablolu köprüye bağlayın ve bilgisayarınıza ethernet arabirimiyle başka hiçbir aygıtın bağlı olmadığından emin olun.
- Ensure the ethernet is plugged into the port closest to the power jack on the wireless bridge.
- 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 ncap and the radio tool and then reinstall the radio tool in order to get back to a known configuration.

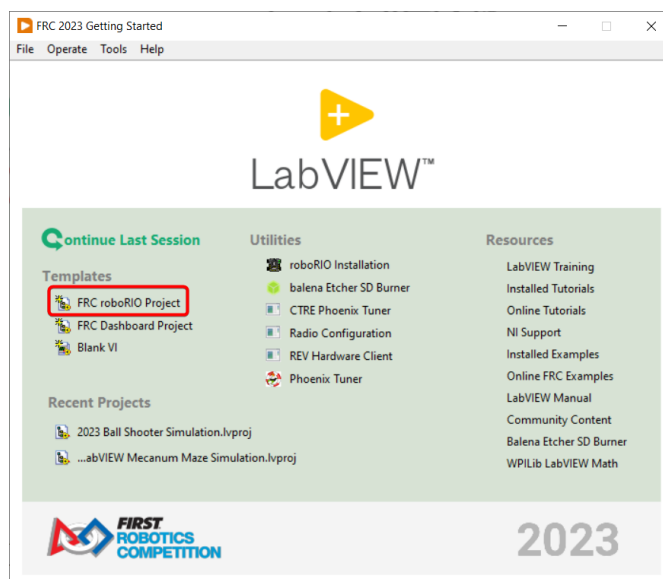
- If all else fails, try a different computer.

Adım 4: Robotunuzu Programlama

5.1 Creating your Test Drivetrain Program (LabVIEW)

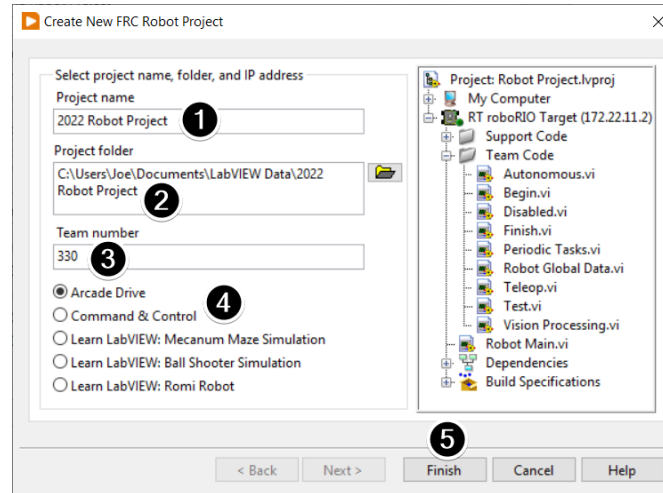
Not: This document covers how to create, build and load a basic FRC® LabVIEW program for a drivetrain onto a roboRIO. Before beginning, make sure that you have installed LabVIEW for FRC and the FRC Game Tools and that you have configured and imaged your roboRIO as described in the [Zero-to-Robot tutorial](#).

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.

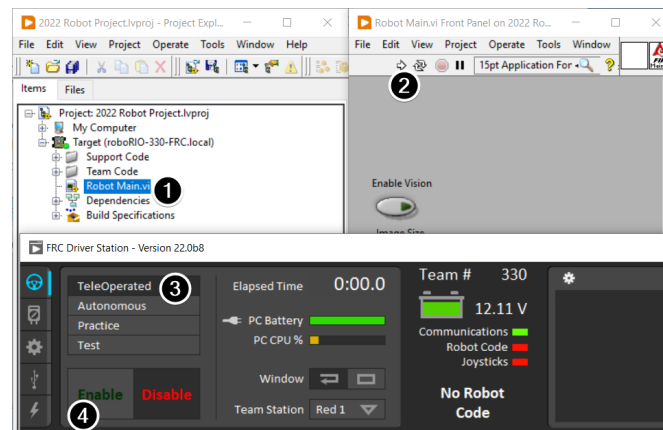
5.1.2 Configuring Project



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

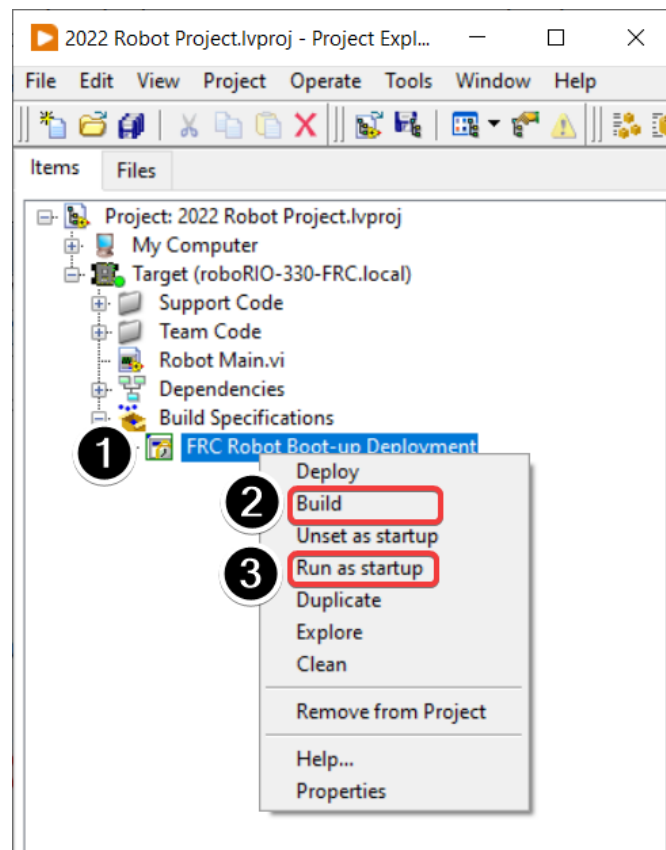


Not: Note that a program deployed in this manner will not remain on the roboRIO after a power cycle. To deploy a program to run every time the roboRIO starts follow the next step, Deploying the program.

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)

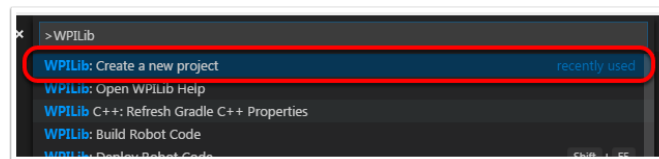
Once everything is installed, we're ready to create a robot program. WPILib comes with several templates for robot programs. Use of these templates is highly recommended for new users; however, advanced users are free to write their own robot code from scratch. This article walks through creating a project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

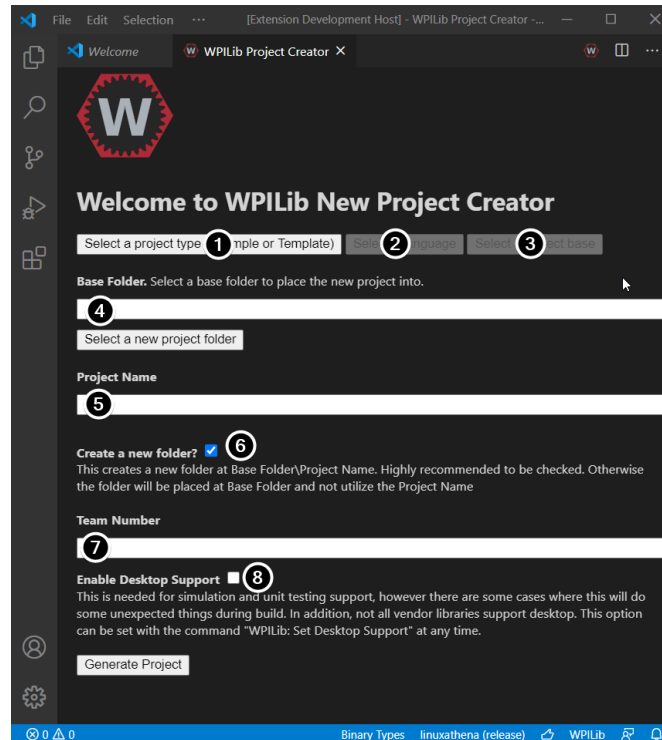
Önemli: This guide includes code examples that involve vendor hardware for the convenience of the user. In this document, [PWM](#) refers to the motor controller included in the KOP. The CTRE tab references the Talon FX motor controller (Falcon 500 motor), but usage is similar for TalonSRX and VictorSPX.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

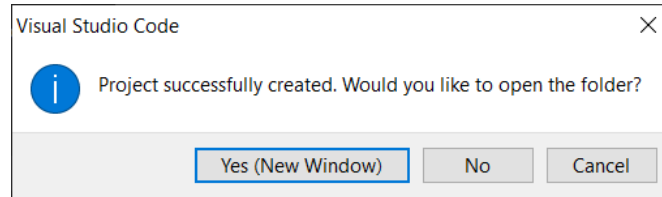
Once all the above have been configured, click “Generate Project” and the robot project will be created.

Not: Any errors in project generation will appear in the bottom right-hand corner of the

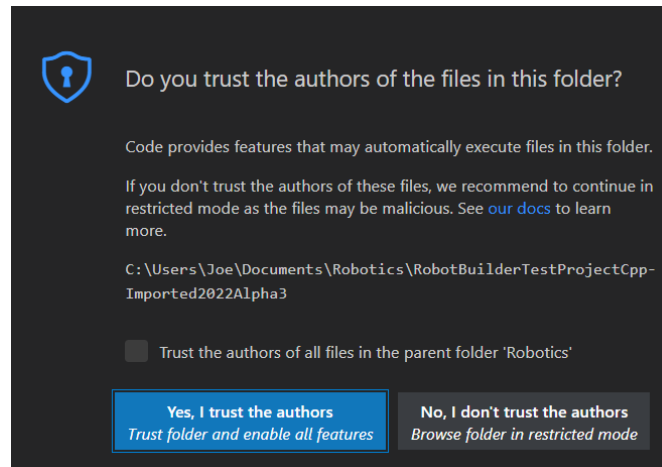
screen.

Uyarı: 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

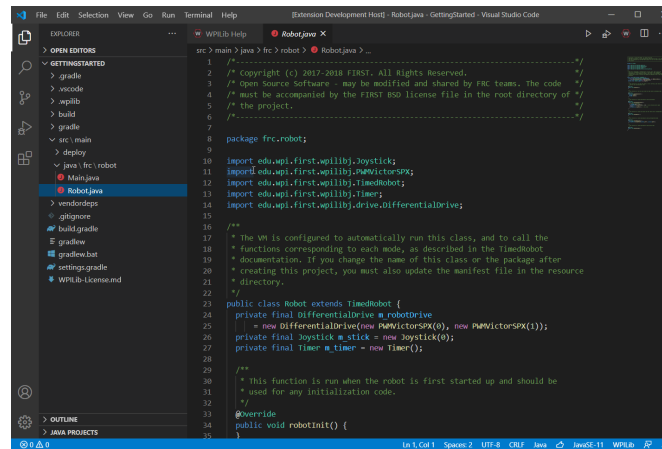


After successfully creating your project, VS Code will give the option of opening the project as shown above. We can choose to do that now or later by typing `Ctrl+K` then `Ctrl+O` (or just `Command+O` on macOS) and select the folder where we saved our project.



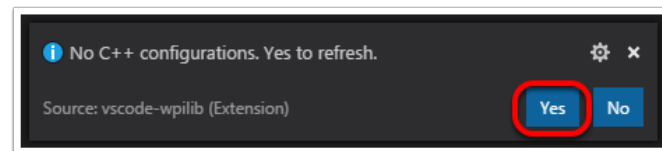
Click *Yes I trust the authors*.

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.

Ayrıca bakınız:

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).

Not: the Python example below is from <https://github.com/robotpy/examples/tree/main/GettingStarted>

JAVA

```
1 // Copyright (c) FIRST and other WPILib contributors.
2 // Open Source Software; you can modify and/or share it under the terms of
3 // the WPILib BSD license file in the root directory of this project.
4
5 package edu.wpi.first.wpilibj.examples.gettingstarted;
6
7 import edu.wpi.first.util.sendable.SendableRegistry;
8 import edu.wpi.first.wpilibj.TimedRobot;
9 import edu.wpi.first.wpilibj.Timer;
10 import edu.wpi.first.wpilibj.XboxController;
11 import edu.wpi.first.wpilibj.drive.DifferentialDrive;
12 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
13
14 /**
15  * The VM is configured to automatically run this class, and to call the functions
16  * ↪corresponding to
17  * ↪each mode, as described in the TimedRobot documentation. If you change the name of
18  * ↪this class or
19  * ↪the package after creating this project, you must also update the manifest file in
20  * ↪the resource
21  * ↪directory.
22  */
23 public class Robot extends TimedRobot {
24     private final PWMSparkMax m_leftDrive = new PWMSparkMax(0);
25     private final PWMSparkMax m_rightDrive = new PWMSparkMax(1);
26     private final DifferentialDrive m_robotDrive =
27         new DifferentialDrive(m_leftDrive::set, m_rightDrive::set);
28     private final XboxController m_controller = new XboxController(0);
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

26 private final Timer m_timer = new Timer();
27
28 public Robot() {
29     SendableRegistry.addChild(m_robotDrive, m_leftDrive);
30     SendableRegistry.addChild(m_robotDrive, m_rightDrive);
31 }
32
33 /**
34  * This function is run when the robot is first started up and should be used for
35  * any initialization code.
36  */
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 }
44
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. */
52 @Override
53 public void autonomousPeriodic() {
54     // Drive for 2 seconds
55     if (m_timer.get() < 2.0) {
56         // Drive forwards half speed, make sure to turn input squaring off
57         m_robotDrive.arcadeDrive(0.5, 0.0, false);
58     } else {
59         m_robotDrive.stopMotor(); // stop robot
60     }
61 }
62
63 /** This function is called once each time the robot enters teleoperated mode. */
64 @Override
65 public void teleopInit() {}
66
67 /** This function is called periodically during teleoperated mode. */
68 @Override
69 public void teleopPeriodic() {
70     m_robotDrive.arcadeDrive(-m_controller.getLeftY(), -m_controller.getRightX());
71 }
72
73 /** This function is called once each time the robot enters test mode. */
74 @Override
75 public void testInit() {}
76
77 /** This function is called periodically during test mode. */
78 @Override
79 public void testPeriodic() {}
80 }

```

C++

```

1  // Copyright (c) FIRST and other WPILib contributors.
2  // Open Source Software; you can modify and/or share it under the terms of
3  // the WPILib BSD license file in the root directory of this project.
4
5  #include <frc/TimedRobot.h>
6  #include <frc/Timer.h>
7  #include <frc/XboxController.h>
8  #include <frc/drive/DifferentialDrive.h>
9  #include <frc/motorcontrol/PWMSparkMax.h>
10
11 class Robot : public frc::TimedRobot {
12 public:
13     Robot() {
14         wpi::SendableRegistry::AddChild(&m_robotDrive, &m_left);
15         wpi::SendableRegistry::AddChild(&m_robotDrive, &m_right);
16
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     }
24
25     void AutonomousInit() override { m_timer.Restart(); }
26
27     void AutonomousPeriodic() override {
28         // Drive for 2 seconds
29         if (m_timer.Get() < 2_s) {
30             // Drive forwards half speed, make sure to turn input squaring off
31             m_robotDrive.ArcadeDrive(0.5, 0.0, false);
32         } else {
33             // Stop robot
34             m_robotDrive.ArcadeDrive(0.0, 0.0, false);
35         }
36     }
37
38     void TeleopInit() override {}
39
40     void TeleopPeriodic() override {
41         // Drive with arcade style (use right stick to steer)
42         m_robotDrive.ArcadeDrive(-m_controller.GetLeftY(),
43                                   m_controller.GetRightX());
44     }
45
46     void TestInit() override {}
47
48     void TestPeriodic() override {}
49
50 private:
51     // Robot drive system
52     frc::PWMSparkMax m_left{0};
53     frc::PWMSparkMax m_right{1};
54     frc::DifferentialDrive m_robotDrive{
55         [&](double output) { m_left.Set(output); },

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

56     [&](double output) { m_right.Set(output); });
57
58     frc::XboxController m_controller{0};
59     frc::Timer m_timer;
60 };
61
62 #ifndef RUNNING_FRC_TESTS
63 int main() {
64     return frc::StartRobot<Robot>();
65 }
66 #endif

```

PYTHON

```

1  #!/usr/bin/env python3
2  #
3  # Copyright (c) FIRST and other WPILib contributors.
4  # Open Source Software; you can modify and/or share it under the terms of
5  # the WPILib BSD license file in the root directory of this project.
6  #
7
8  import wpilib
9  import wpilib.drive
10
11
12 class MyRobot(wpilib.TimedRobot):
13     def robotInit(self):
14         """
15         This function is called upon program startup and
16         should be used for any initialization code.
17         """
18         self.leftDrive = wpilib.PWMSparkMax(0)
19         self.rightDrive = wpilib.PWMSparkMax(1)
20         self.robotDrive = wpilib.drive.DifferentialDrive(
21             self.leftDrive, self.rightDrive
22         )
23         self.controller = wpilib.XboxController(0)
24         self.timer = wpilib.Timer()
25
26         # We need to invert one side of the drivetrain so that positive voltages
27         # result in both sides moving forward. Depending on how your robot's
28         # gearbox is constructed, you might have to invert the left side instead.
29         self.rightDrive.setInverted(True)
30
31     def autonomousInit(self):
32         """This function is run once each time the robot enters autonomous mode."""
33         self.timer.restart()
34
35     def autonomousPeriodic(self):
36         """This function is called periodically during autonomous."""
37
38         # Drive for two seconds
39         if self.timer.get() < 2.0:
40             # Drive forwards half speed, make sure to turn input squaring off

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
41         self.robotDrive.arcadeDrive(0.5, 0, squareInputs=False)
42     else:
43         self.robotDrive.stopMotor() # Stop robot
44
45     def teleopInit(self):
46         """This function is called once each time the robot enters teleoperated mode."
47         ↪ ""
48
49     def teleopPeriodic(self):
50         """This function is called periodically during teleoperated mode."""
51         self.robotDrive.arcadeDrive(
52             -self.controller.getLeftY(), -self.controller.getRightX()
53         )
54
55     def testInit(self):
56         """This function is called once each time the robot enters test mode."""
57
58     def testPeriodic(self):
59         """This function is called periodically during test mode."""
60
61     if __name__ == "__main__":
62         wpilib.run(MyRobot)
```

Now let's look at various parts of the code.

5.2.6 Imports/Includes

PWM

Java

```
1 import edu.wpi.first.util.sendable.SendableRegistry;
2 import edu.wpi.first.wpilibj.TimedRobot;
3 import edu.wpi.first.wpilibj.Timer;
4 import edu.wpi.first.wpilibj.XboxController;
5 import edu.wpi.first.wpilibj.drive.DifferentialDrive;
6 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
```

C++

```
5 #include <frc/TimedRobot.h>
6 #include <frc/Timer.h>
7 #include <frc/XboxController.h>
8 #include <frc/drive/DifferentialDrive.h>
9 #include <frc/motorcontrol/PWMSparkMax.h>
```

Python

```

8 import wpilib
9 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     }
```

```

50 private:
51     // Robot drive system
52     frc::PWMSparkMax m_left{0};
53     frc::PWMSparkMax m_right{1};
54     frc::DifferentialDrive m_robotDrive{
55         [&](double output) { m_left.Set(output); },
56         [&](double output) { m_right.Set(output); }};
57
58     frc::XboxController m_controller{0};
59     frc::Timer m_timer;
60 };

```

Python

```

12 class MyRobot(wpiilib.TimedRobot):
13     def robotInit(self):
14         """
15         This function is called upon program startup and
16         should be used for any initialization code.
17         """
18         self.leftDrive = wpiilib.PWMSparkMax(0)
19         self.rightDrive = wpiilib.PWMSparkMax(1)
20         self.robotDrive = wpiilib.drive.DifferentialDrive(
21             self.leftDrive, self.rightDrive
22         )
23         self.controller = wpiilib.XboxController(0)
24         self.timer = wpiilib.Timer()
25
26         # We need to invert one side of the drivetrain so that positive voltages
27         # result in both sides moving forward. Depending on how your robot's
28         # gearbox is constructed, you might have to invert the left side instead.
29         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++

```
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
    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

```
13 class MyRobot(wpilib.TimedRobot):
14     def robotInit(self):
15         """
16         This function is called upon program startup and
17         should be used for any initialization code.
18         """
19         self.leftDrive = ctre.WPI_TalonFX(1)
20         self.rightDrive = ctre.WPI_TalonFX(2)
21         self.robotDrive = wpilib.drive.DifferentialDrive(
22             self.leftDrive, self.rightDrive
23         )
24         self.controller = wpilib.XboxController(0)
25         self.timer = wpilib.Timer()
26
27         # We need to invert one side of the drivetrain so that positive voltages
28         # result in both sides moving forward. Depending on how your robot's
29         # gearbox is constructed, you might have to invert the left side instead.
30         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);
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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.

Not: 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.

Not: 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. */
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

52  @Override
53  public void autonomousPeriodic() {
54      // Drive for 2 seconds
55      if (m_timer.get() < 2.0) {
56          // Drive forwards half speed, make sure to turn input squaring off
57          m_robotDrive.arcadeDrive(0.5, 0.0, false);
58      } else {
59          m_robotDrive.stopMotor(); // stop robot
60      }
61  }

```

C++

```

25  void AutonomousInit() override { m_timer.Restart(); }
26
27  void AutonomousPeriodic() override {
28      // Drive for 2 seconds
29      if (m_timer.Get() < 2_s) {
30          // Drive forwards half speed, make sure to turn input squaring off
31          m_robotDrive.ArcadeDrive(0.5, 0.0, false);
32      } else {
33          // Stop robot
34          m_robotDrive.ArcadeDrive(0.0, 0.0, false);
35      }
36  }

```

PYTHON

```

31  def autonomousInit(self):
32      """This function is run once each time the robot enters autonomous mode."""
33      self.timer.restart()
34
35  def autonomousPeriodic(self):
36      """This function is called periodically during autonomous."""
37
38      # Drive for two seconds
39      if self.timer.get() < 2.0:
40          # Drive forwards half speed, make sure to turn input squaring off
41          self.robotDrive.arcadeDrive(0.5, 0, squareInputs=False)
42      else:
43          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

```

63  /** This function is called once each time the robot enters teleoperated mode. */
64  @Override
65  public void teleopInit() {}
66
67  /** This function is called periodically during teleoperated mode. */
68  @Override
69  public void teleopPeriodic() {
70      m_robotDrive.arcadeDrive(-m_controller.getLeftY(), -m_controller.getRightX());
71  }

```

C++

```

38  void TeleopInit() override {}
39
40  void TeleopPeriodic() override {
41      // Drive with arcade style (use right stick to steer)
42      m_robotDrive.ArcadeDrive(-m_controller.GetLeftY(),
43                               m_controller.GetRightX());
44  }

```

PYTHON

```

45  def teleopInit(self):
46      """This function is called once each time the robot enters teleoperated mode."
47      ↪ ""
48
49  def teleopPeriodic(self):
50      """This function is called periodically during teleoperated mode."""
51      self.robotDrive.arcadeDrive(
52          -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

```

73  /** This function is called once each time the robot enters test mode. */
74  @Override
75  public void testInit() {}
76

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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

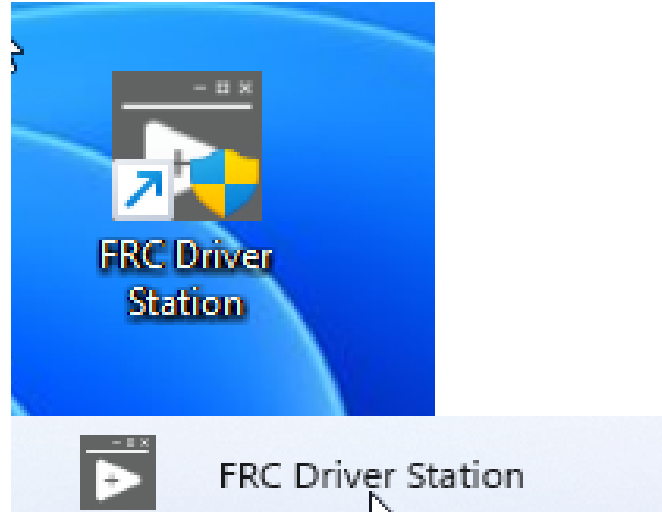
LabVIEW

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.

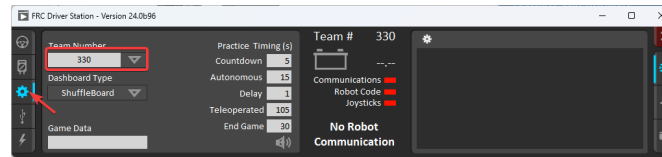
The roboRIO should be powered on and connected to the PC over Ethernet or USB.

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.

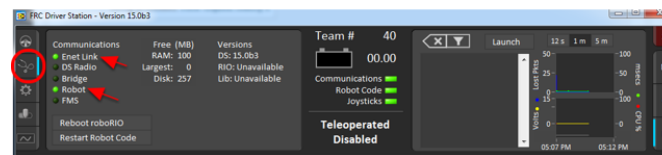
5.3.4 Setting Up the Driver Station



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.

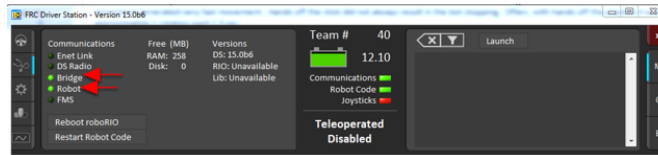
PCs will typically have the correct network settings for the DS to connect to the robot already, but if not, make sure your Network adapter is set to *DHCP*.

5.3.5 Confirm Connectivity



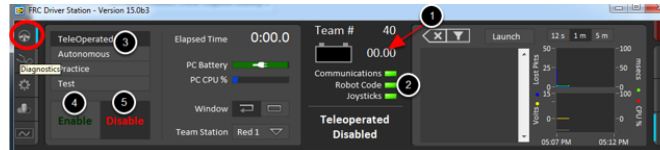
Şekil 1: Tethered

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.



Şekil 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.

Donanım Bileşenine Genel Bakış

Bu belgenin amacı, FRC® 'i oluşturan donanım bileşenlerine kısa bir genel bakış sağlamaktır. Kontrol sistemi. Her bileşen, bileşen işlevinin kısa bir açıklamasını ve daha fazla dokümantasyona bir bağlantı içerecektir.

Not: For wiring instructions/diagrams, please see the *Wiring the FRC Control System* document.

6.1 Kontrol Sistemine Genel Bakış

REV

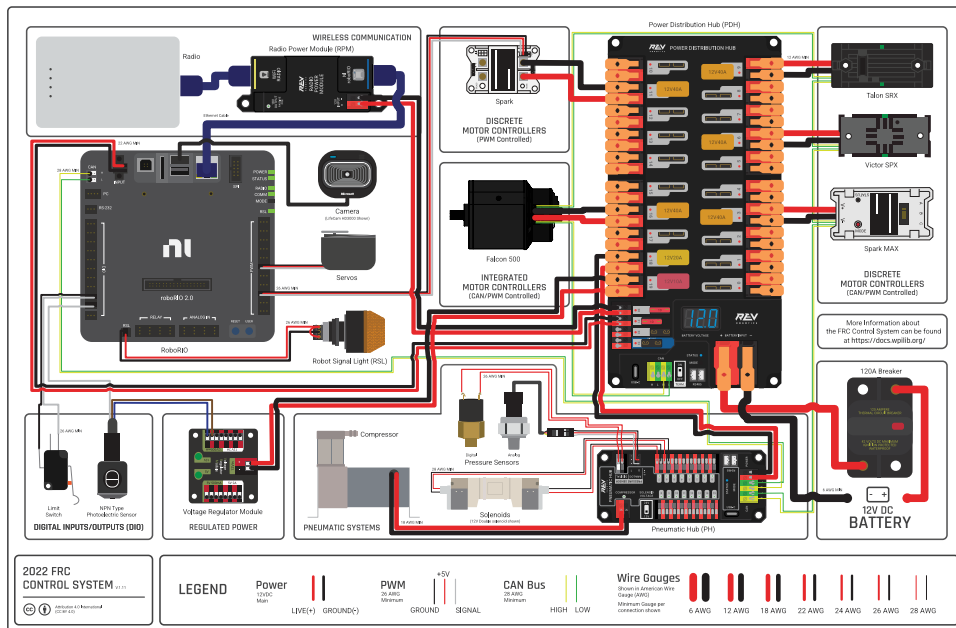


Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

CTRE

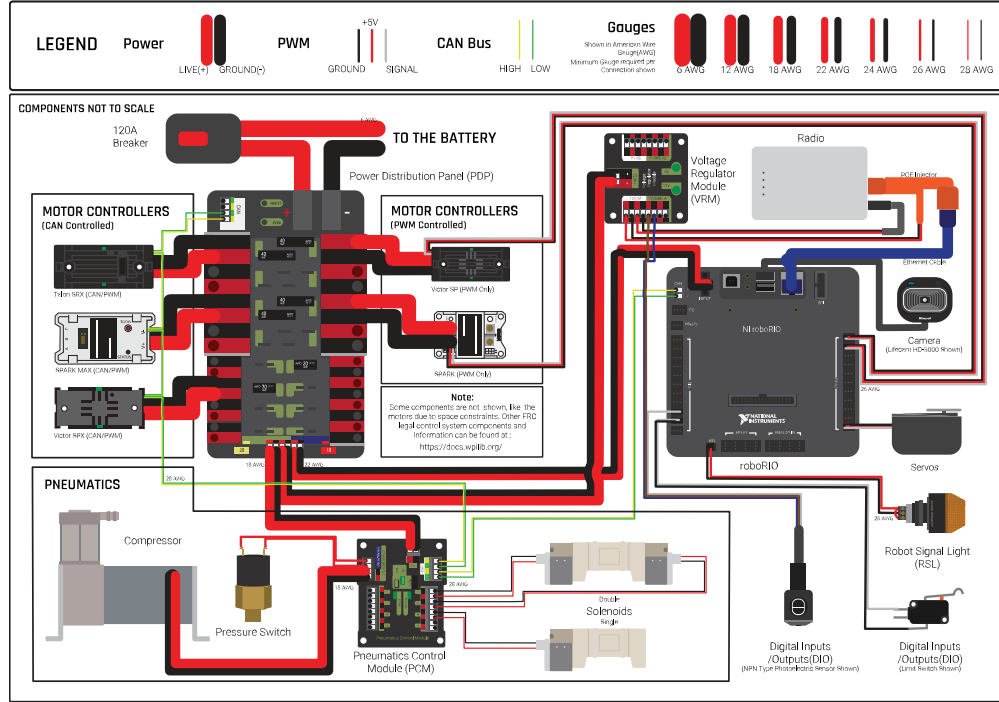
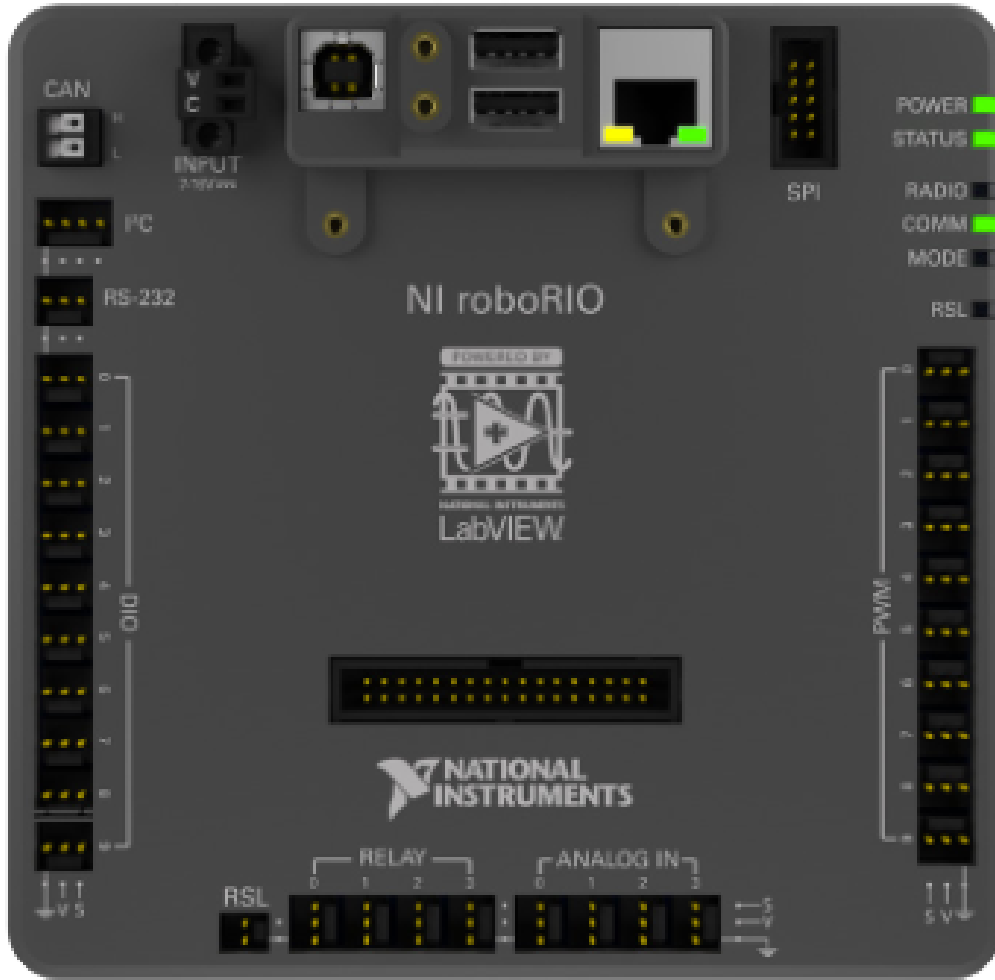


Diagram courtesy of FRC® Team 3161 and Stefen Acepcion.

6.2 NI roboRIO



NI-roboRIO , FRC için kullanılan ana robot denetleyicisidir. RoboRIO, diğer tüm donanımlara komuta eden ekip tarafından oluşturulan kodu çalıştıran robot için “brain-beyin” görevi görür.

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 veya OM5P-AC Radyo-Modem



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](#).

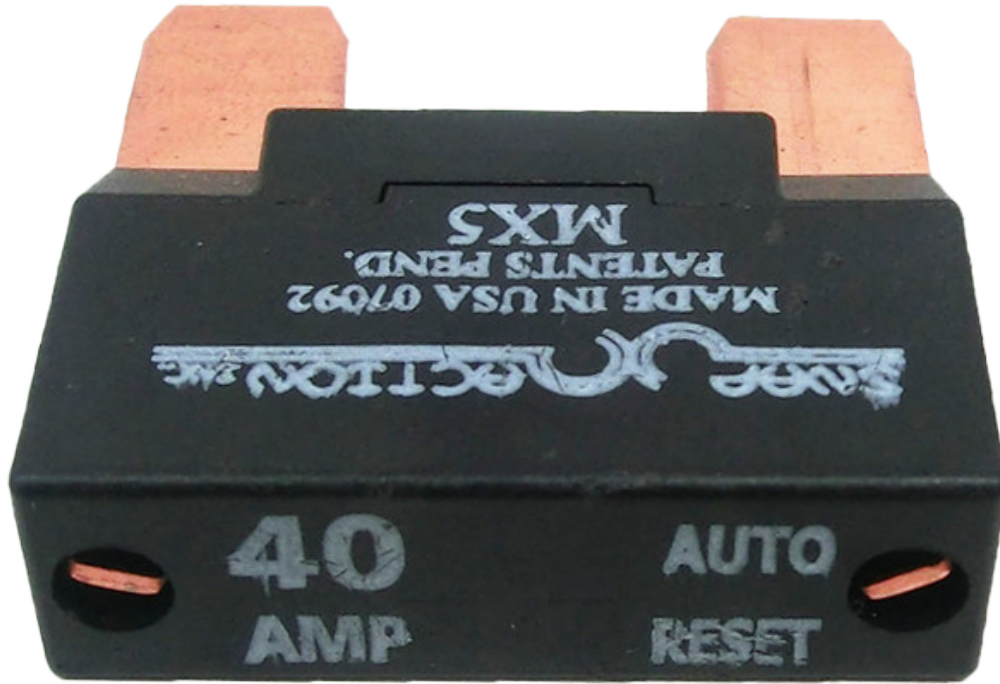
OM5P-AN [artık satın alınamıyor](#). OM5P-AC, OM5P-AN'ye kıyasla biraz daha ağırdır, daha fazla soğutma ızgarasına ve pürüzlü bir yüzey dokusuna sahiptir.

6.8 120A Circuit Breaker - Devre Kesici Anahtar



120A Ana Devre Kesici, robot üzerinde iki görev görür : ana robot güç anah-tarı ve aşağı akışlı robot kabloları ve bileşenleri için bir koruma cihazı. 120A devre kesici, robot pilinin ve Güç Dağıtım kartlarının pozitif terminallerine bağla-nır. Daha fazla bilgi için lütfen Cooper Bussmann 18X Serisi Veri Sayfası'na (PN: 185120F) <https://www.mouser.com/datasheet/2/87/BUS_Tns_DS_18X_CIRCUITBREAKER-515519.pdf> bakın.

6.9 Snap Action Devre Kesici - Sigorta



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 Robot Aküsü



The power supply for an FRC robot is a single 12V 18Ah Sealed Lead Acid (SLA) battery, capable of meeting the high current demands of an FRC robot. For more information, see the [Robot Battery page](#).

Not: Multiple battery part numbers may be legal, consult the [FRC Manual](#) for a complete list.

6.11 Robot Sinyal Işıđı - Robot Signal Light - RSL

Allen-Bradley



Şekil 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.



Şekil 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.

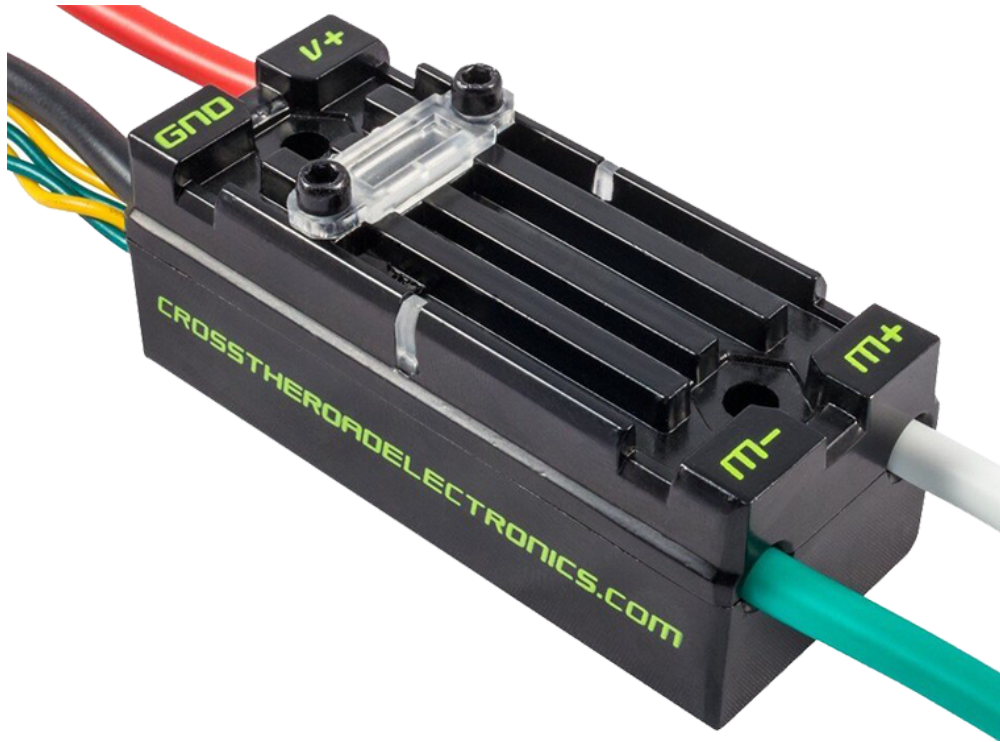
Digital and analog pressure sensor ports are built into the device, increasing the flexibility and feedback functionality of the pneumatic system. The USB-C connection on the Hub works with the REV Hardware Client, allowing users to test pneumatic systems without a need for an additional robot controller.

6.14 Motor Kontrolörleri

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*.

Not: 3rd Party CAN control is not supported from WPILib. See this section on *Üçüncü Taraf CAN Cihazları* for more information.

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 Motor Kontrolörü



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 Motor Kontrolörü



The [SPARK MAX Motor Controller](#) is an advanced brushed and brushless DC motor controller from REV Robotics. When using CAN bus or USB control, the SPARK MAX uses input from limit switches, encoders, and other sensors, including the integrated encoder of the REV NEO Brushless Motor, to perform advanced control modes. The SPARK MAX can be controlled over PWM, CAN or USB (for configuration/testing only). For more information, see the [SPARK MAX User's Manual](#).

6.14.4 TalonFX Motor Kontrol Cihazı



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](#).

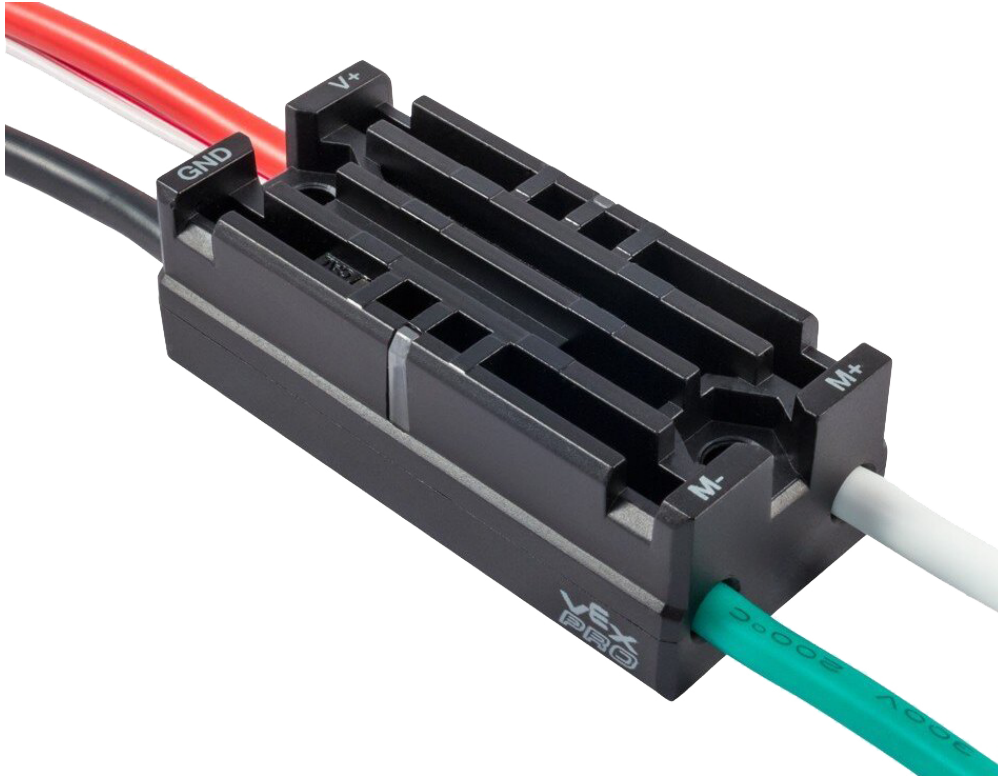
6.14.5 SPARK Motor Kontrolörü



Uyarı: Bu motor kontrolörü FRC kullanımı için hala yasal olsa da, üretici bu ürünün üretimini durdurmuştur.

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 Motor Kontrolörü



Uyarı: Bu motor kontrolörü FRC kullanımı için hala yasal olsa da, üretici bu ürünün üretimini durdurmuştur.

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 Motor Kontrolörü



Uyarı: Bu motor kontrolörü FRC kullanımı için hala yasal olsa da, üretici bu ürünün üretimini durdurmuştur.

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 Motor Kontrolörü/ Victor 884 Motor Kontrolörü



Uyarı: Bu motor kontrolörü FRC kullanımı için hala yasal olsa da, üretici bu ürünün üretimini durdurmuştur.

VEX Robotics'in [Victor 884](#) ve [Victor 888](#) motor kontrolörleri, FRC'de kullanım için değişken hızlı PWM motor kontrolörleridir. Victor 888, FRC'de de kullanılabilen Victor 884'ün yerini alıyor.

6.14.9 Jaguar Motor Kontrolörü



Uyarı: Bu motor kontrolörü FRC kullanımı için hala yasal olsa da, üretici bu ürünün üretimini durdurmuştur.

VEX Robotics'in (daha önce Luminary Micro ve Texas Instruments tarafından yapılmış) **Jaguar Motor Controller**, FRC'de kullanım için değişken hızlı bir motor kontrolörüdür. FRC için Jaguar yalnızca PWM arayüzü kullanılarak kontrol edilebilir.

6.14.10 DMC-60 ve DMC-60C Motor Kontrolörü



Uyarı: Bu motor kontrolörü FRC kullanımı için hala yasal olsa da, üretici bu ürünün üretimini durdurmuştur.

DMC-60, Digilent'in bir PWM motor kontrolörüdür. DMC-60, aşırı ısınmayı ve hasarı önlemek için akım bükme dahil entegre termal algılama, koruma ve daha kolay hata ayıklama için hızı, yönü ve durumu gösteren dört çok renkli LED'i içerir. Daha fazla bilgi için, [DMC-60 referans kılavuzuna bakın](#)

DMC-60C, DMC-60 kontrol cihazına CAN akıllı kontrolör yetenekleri ekler. Üreticinin bu ürünü durdurması nedeniyle, DMC-60C yalnızca PWM ile kullanılabilir. Daha fazla bilgi için bakınız, [DMC-60C Product Page- Ürün sayfası](#)

6.14.11 Venom Motor Kontrolörü



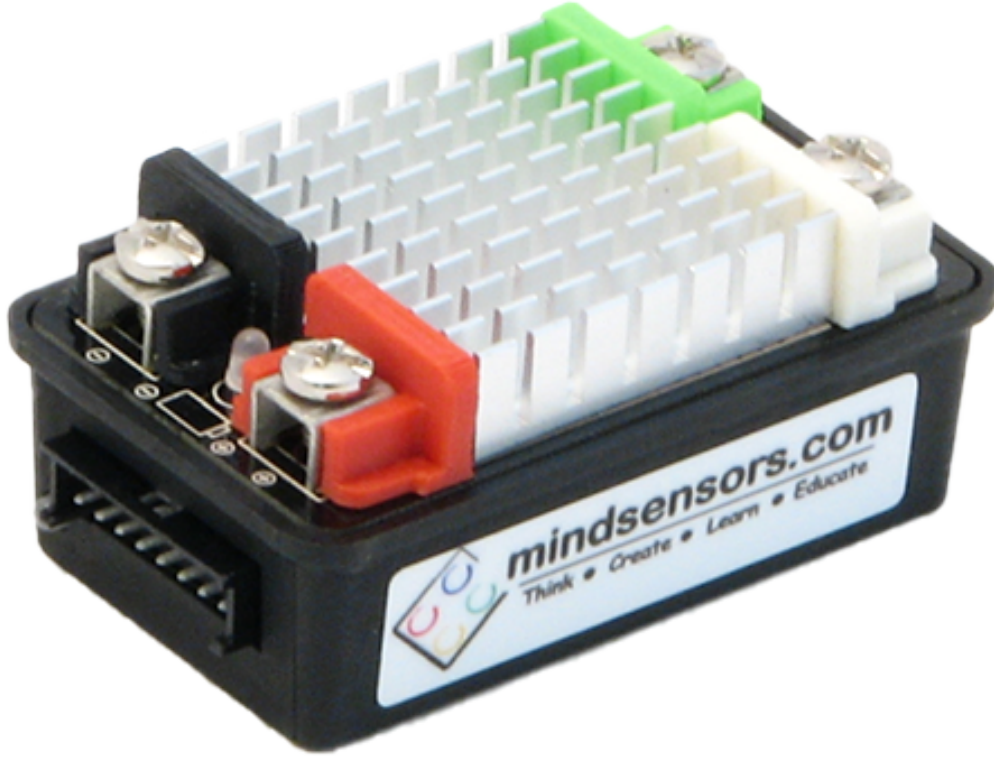
The [Venom Motor Controller](#) from Playing With Fusion is integrated into a motor based on the original [CIM](#). Speed, current, temperature, and position are all measured onboard, enabling advanced control modes without complicated sensing and wiring schemes.

6.14.12 Nidec Dynamo BLDC Motor Kontroller ile birlikte



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 ve SD540C Motor Kontrolörleri



Mindsensors'tan SD540B ve SD540C Motor Kontrolörleri, PWM kullanılarak kontrol edilir. Üretici desteğinin olmaması nedeniyle SD540C için CAN kontrolü artık mevcut değil. Motor hareketini bir veya her iki yönde sınırlamak için limit anahtarları doğrudan SD540'a bağlanabilir. Daha fazla bilgi için [Mindsensors FRC sayfasına](#) bakın

6.15 Spike H-Bridge Rölesi



Uyarı: Bu röle FRC kullanımı için hala yasal olmakla birlikte, üretici bu ürünün üretimini durdurmuştur.

VEX Robotics'ten Spike H-Bridge Relay, motorlara veya diğer özel robot elektronik cihazlarına giden gücü kontrol etmek için kullanılan bir cihazdır. Bir motora bağlandığında, Spike hem ileri hem de geri yönde On/Off kontrolü sağlar. Spike çıkışları bağımsız olarak kontrol edildiğinden 2 adede kadar özel elektronik devreye güç sağlamak için de kullanılabilir. Spike H-Bridge Rölesi roboRIO'nun bir röle çıkışına bağlanmalı ve Güç Dağıtım Panelinden beslenmelidir. Daha fazla bilgi için bkz. [Spike User's Guide](#).

6.16 Servo Güç Modülü



Rev Robotics'in Servo Güç Modülü, servolara sağlanan gücü roboRIO entegre güç kaynağının yapabileceğinin ötesinde genişletebilir. Servo Güç Modülü, 6 kanalda 90W'a kadar 6V güç sağlar. Tüm kontrol sinyalleri doğrudan roboRIO'dan geçer. Daha fazla bilgi için, bkz. [Servo Güç Modülü web sayfası](#).

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 Görsel Atıflar

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.

Yazılım Bileşenine Genel Bakış

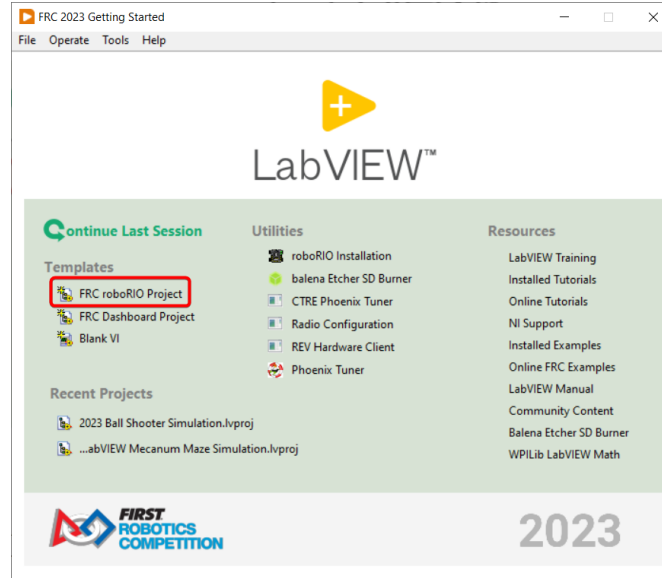
FRC ® yazılım, çok çeşitli zorunlu ve isteğe bağlı bileşenlerden oluşur. Bu öğeler, robot kodunun tasarımı, geliştirilmesi ve hata ayıklamasında size yardımcı olmanın yanı sıra kontrol robotunun çalışmasına yardımcı olmak ve sorun giderme sırasında geri bildirim sağlamak için tasarlanmıştır. Bu belge, her bir yazılım bileşeni için, amacına ilişkin kısa bir genel bakış, uygunsa paket indirme bağlantısı ve varsa diğer belgelere bağlantı sağlar.

7.1 İşletim Sistemi Uyumluluğu

FRC bileşenleri için desteklenen birincil işletim sistemi Windows'tur. Tüm gerekli FRC yazılım bileşenleri Windows 10 & 11 'de test edilmiştir.

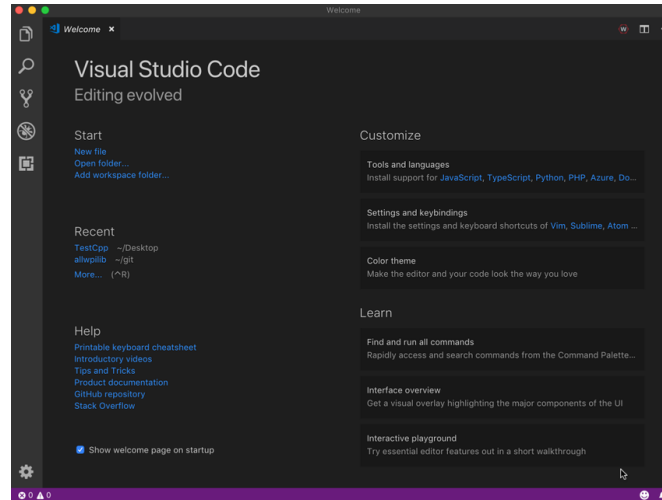
Many of the tools for C++/Java/Python programming are also supported and tested on macOS and Linux. Teams programming in C++/Java/Python should be able to develop using these systems, using a Windows system for the Windows-only operations such as the Driver Station, Radio Configuration Utility, and roboRIO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

7.2 LabVIEW FRC (Yalnızca Windows)



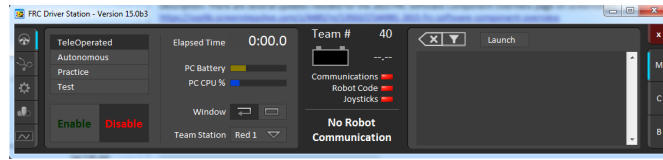
LabVIEW Professional'ın son sürümüne dayanan LabVIEW FRC, bir FRC robotu programlamak için resmi olarak desteklenen üç dilden biridir. LabVIEW grafiksel, veri akışı odaklı bir dildir. LabVIEW programları, VI'lar arasında veri ileten kablolarla birlikte bağlanan ve VI adı verilen bir simgeler koleksiyonundan oluşur. LabVIEW FRC yükleyici, Kickoff Parça Kitinde bulunan bir DVD'de dağıtılır ve ayrıca indirilebilir. Kurulum talimatları dahil olmak üzere LabVIEW FRC yazılımına başlama kılavuzu bulunabilir [here](#).

7.3 Visual Studio Kodu



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](#).

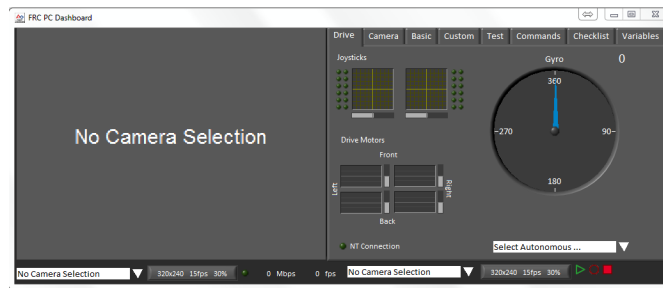
7.4 NI LabVIEW Tarafından Desteklenen FRC Sürücü İstasyonu (Yalnızca Windows)



Yarışma sırasında robotun durumunu kontrol etmek amacıyla kullanılmasına izin verilen tek yazılım budur. Bu yazılım, robotunuza çeşitli giriş cihazlarından veri gönderir. Ayrıca, robot sorunlarını gidermeye yardımcı olmak için kullanılan bir dizi araç içerir. NI LabVIEW Tarafından Desteklenen FRC Sürücü İstasyonu hakkında daha fazla bilgi bulunabilir [here](#).

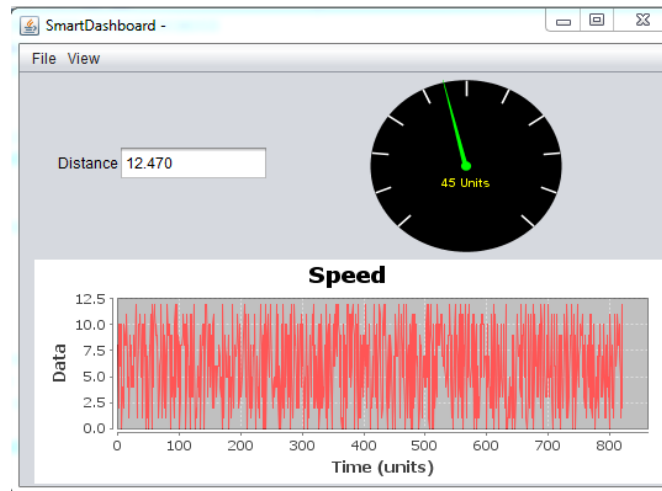
7.5 Gösterge Tablosu Seçenekleri

7.5.1 LabVIEW Dashboard (Yalnızca Windows)



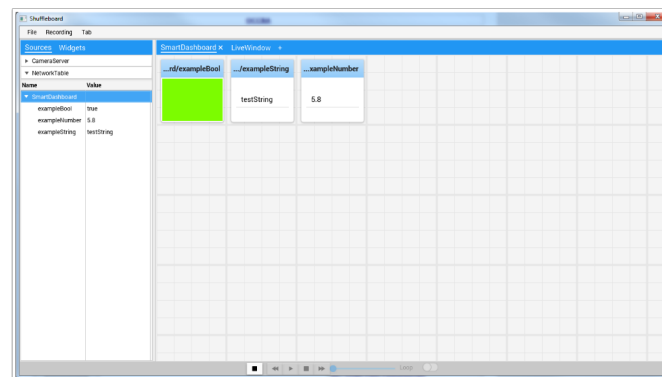
The LabVIEW Dashboard is automatically launched by the FRC Driver Station by default. The purpose of the Dashboard is to provide feedback about the operation of the robot using tabbed display with a variety of built in features. More information about the FRC Default Dashboard software can be found [here](#).

7.5.2 SmartDashboard



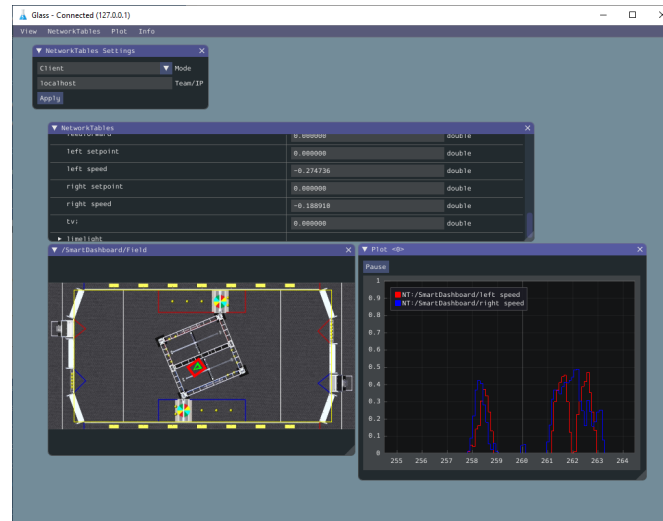
SmartDashboard allows you to view your robot data by automatically creating customizable indicators specifically for each piece of data sent from your robot. Additional documentation on SmartDashboard can be found [here](#).

7.5.3 Shuffleboard



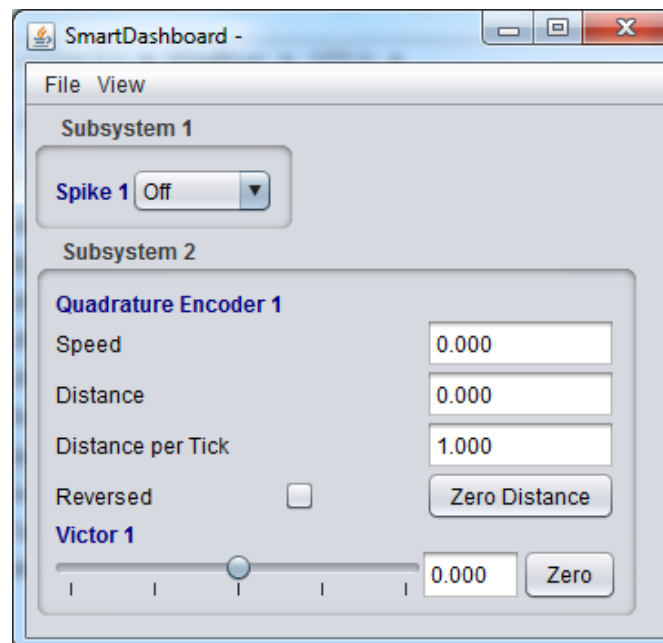
Shuffleboard has the same features as SmartDashboard. It also improves on the setup and visualization of your data with new features and a modern design at the cost of being less resource efficient. Additional documentation on Shuffleboard can be found [here](#).

7.5.4 Glass



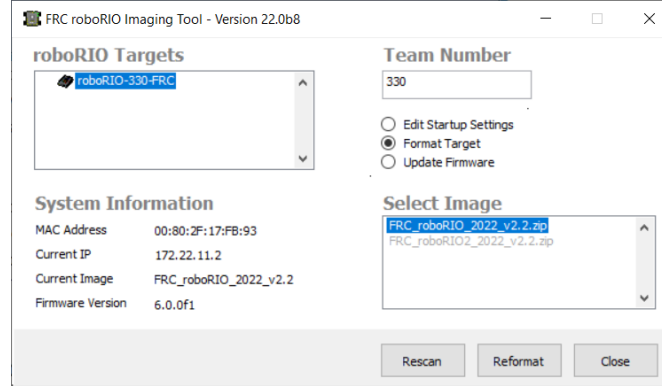
Glass is a Dashboard focused on being a programmer's tool for debugging. The primary advantages are the field view, pose visualization and advanced signal plotting tools.

7.6 LiveWindow



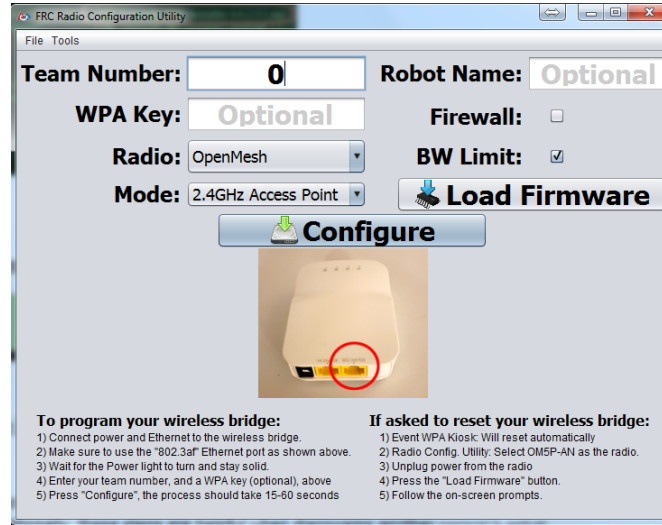
LiveWindow is a feature of SmartDashboard and Shuffleboard, designed for use with the Test Mode of the Driver Station. LiveWindow allows the user to see feedback from sensors on the robot and control actuators independent of the written user code. More information about LiveWindow can be found [here](#).

7.7 FRC roboRIO Imaging Tool (Yalnızca Windows)



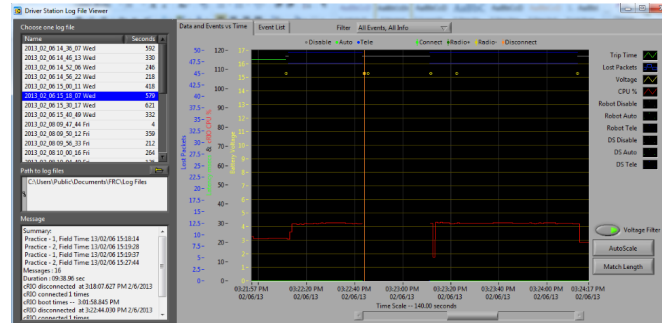
Bu araç, FRC’de kullanılmak üzere bir roboRIO’yu biçimlendirmek ve ayarlamak için kullanılır. Kurulum talimatları bulunabilir [here](#). Bu aracı kullanarak roboRIO’nuzu görüntüleme hakkında ek talimatlar bulunabilir [here](#).

7.8 FRC Radyo Yapılandırma Yardımcı Programı (Yalnızca Windows)



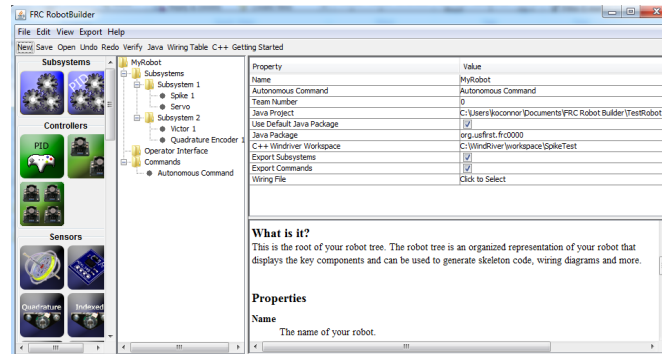
FRC Radyo Yapılandırma Yardımcı Programı, standart modemi evde pratik kullanım için yapılandırmak için kullanılan bir araçtır. Bu araç, FRC oyun alanı deneyimini taklit etmek için uygun ağ ayarlarını ayarlar. FRC Radyo Yapılandırma Yardımcı Programı, bulunabilen bağımsız bir yükleyici tarafından yüklenir [here](#).

7.9 FRC Driver Station Günlük Görüntüleyicisi (Yalnızca Windows)



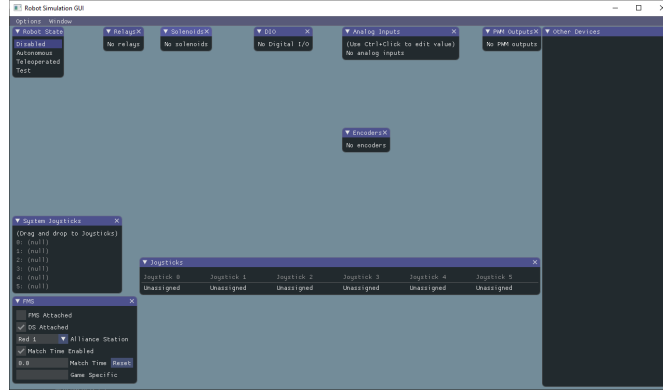
FRC Driver Station Günlük Görüntüleyicisi, FRC Driver Station tarafından oluşturulan günlükleri görüntülemek için kullanılır. Bu günlükler, bir antrenman seansı veya FRC maçı sırasında ne olduğunu anlamak için önemli çeşitli bilgiler içerir. FRC Sürücü İstasyonu Günlük Görüntüleyicisi hakkında daha fazla bilgi ve günlükleri anlamak şu adreste bulunabilir [burada](#).

7.10 RobotBuilder



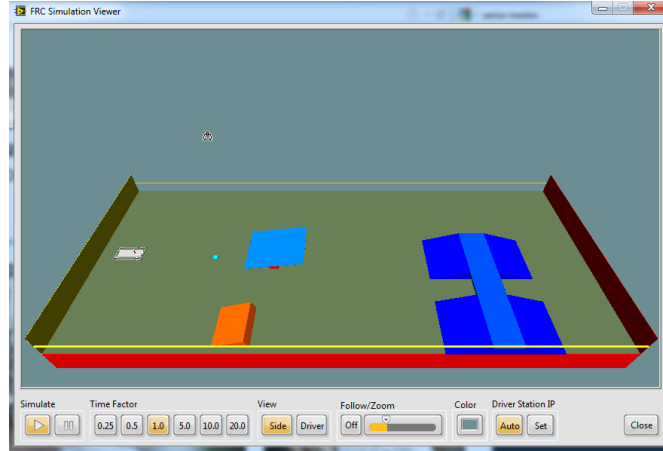
RobotBuilder is a tool designed to aid in setup and structuring of a Command Based robot project for C++ or Java (Python not currently supported). RobotBuilder allows you to enter in the various components of your robot subsystems and operator interface and define what your commands are in a graphical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

7.11 Robot Simülasyonu



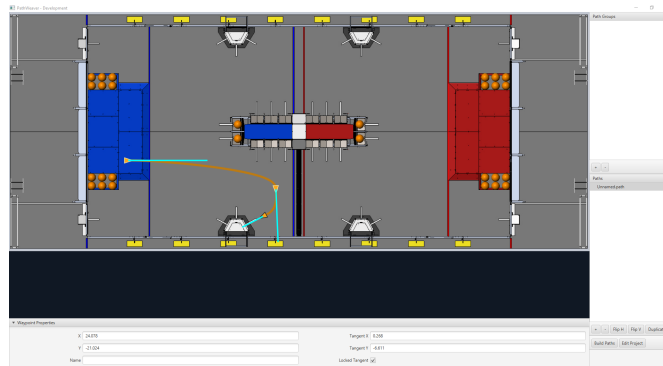
Robot Simulation offers a way for Java, C++, and Python teams to verify their actual robot code is working in a simulated environment. This simulation can be launched directly from VS Code and includes a 2D field that users can visualize their robot's movement on. For more information see the [Robot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

7.12 FRC LabVIEW Robot Simölätörü (Yalnızca Windows)



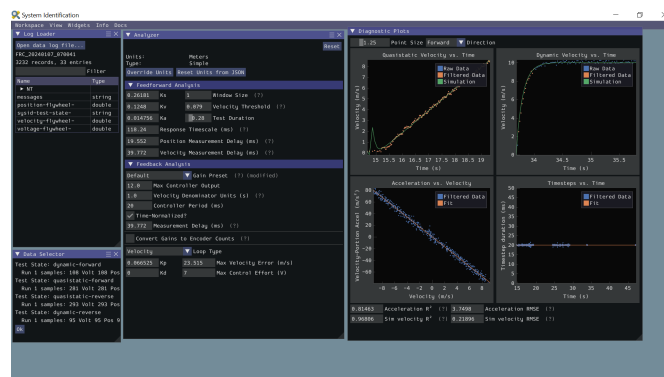
FRC Robot Simölätörü, kodu ve / veya Sürücü İstasyonu işlevlerini test etmek için simüle edilmiş bir ortamda önceden tanımlanmış bir robotu çalıştırmanıza olanak tanıyan LabVIEW programlama ortamının bir bileşenidir. FRC Robot Simölätörünün kullanımına ilişkin bilgiler [burada](#) veya LabVIEW Project Explorer'da Robot Simulation Readme.html dosyasını açarak bulunabilir.

7.13 PathWeaver



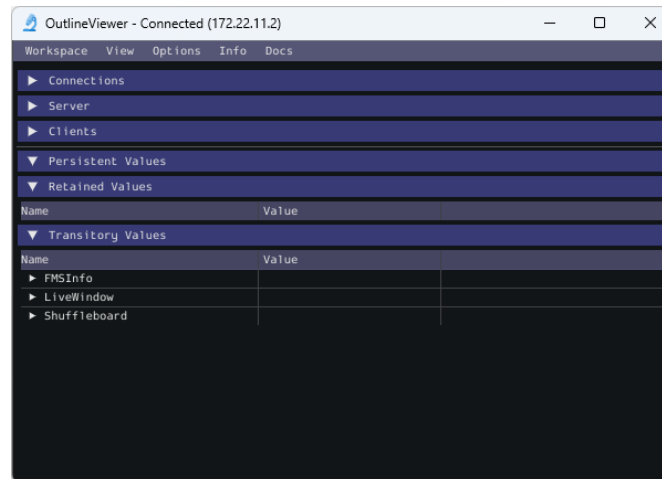
PathWeaver allows teams to quickly generate and configure paths for advanced autonomous routines. These paths have smooth curves allowing the team to quickly navigate their robot between points on the field. For more information see the [PathWeaver section](#).

7.14 System Identification



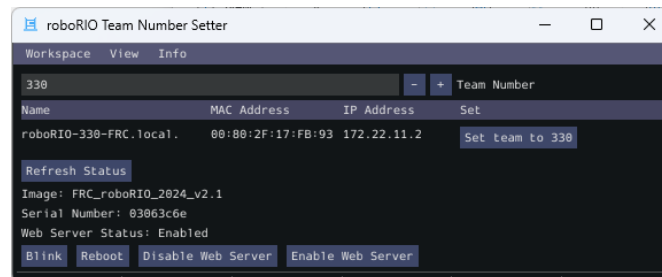
This tool helps teams automatically calculate constants that can be used to describe the physical properties of your robot for use in features like robot simulation, trajectory following, and PID control. For more information see the [System Identification section](#).

7.15 OutlineViewer



OutlineViewer is a utility used to view, modify and add to all of the contents of the Network-Tables for debugging purposes. LabVIEW teams can use the Variables tab of the LabVIEW Dashboard to accomplish this functionality. For more information see the [Outline Viewer section](#).

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](#).

WPILib Nedir?

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 Desteklenen Diller

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 Kaynak kodu ve belgeleme

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:

Java ve C++ kaynak kodu, WPILibJ ve WPILibC kaynak dizinlerinde bulunabilir:

- Java kaynak kodu <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibj/src/main/java/edu/wpi/first>

- C++ kaynak kodu <<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](#)

9.1 Bilinen Sorunlar

Bu makale, FRC® Kontrol Sistemi Yazılımı için bilinen sorunları (ve geçici çözümleri) ayrıntılarıyla anlatmaktadır.

9.1.1 Açık sorunlar

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.

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

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

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

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

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

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

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

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

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

Driver Station Reports Less Free RAM than 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 than 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.

Not: 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 reestablish 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 Network-Tables. 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.

Invalid build due to missing GradleRIO

Issue: Rarely, a user's Gradle cache will get broken and they will get shown errors similar to the following:

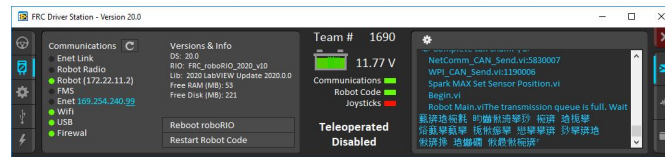
```
Could not apply requested plugin [id: 'edu.wpi.first.GradleRIO', version: '2020.3.2']  
→ as it does not provide a plugin with id 'edu.wpi.first.GradleRIO'
```

Workaround-Çözüm:

Delete your Gradle cache located under `~$USER_HOME/.gradle`. Windows machines may need to enable the ability to [view hidden files](#). This issue has only shown up on Windows so far. Please [report](#) this issue if you get it on an alternative OS.

Sürücü İstasyonu Günlüğünde Çince karakterler

****Issue-Sorun:** * Nadiren, sürücü istasyonu günlüğünde İngilizce metin yerine Çince karakterler görünecektir. Bu, yalnızca Windows, İngilizce dışında bir dile ayarlandığında meydana gelir.



Çözüm: Bilinen iki geçici çözüm vardır:

1. Çince karakterleri kopyalayıp not defterine yapıştırın, İngilizce metin görünecektir.
2. Windows dilini geçici olarak İngilizce olarak değiştirin.

C++ Intellisense - Açılışta Açılan Dosyalar Düzgün Çalışmıyor

Issue-Sorun: C ++ 'da, VS Code başlatıldığında açılan dosyalarda Intellisense, yalnızca uygun olanları değil veya üstbilgi dosyalarını bulamama yerine bir derleme biriminden tüm seçeneklerden gelen önerileri gösteren sorunlar yaşayacaktır. Bu VS Code'daki bir hatadır.

Workaround-Çözüm:

1. VS Code'daki tüm dosyaları kapatın, ancak VS Code'u açık bırakın
2. Varsa .vscode klasöründeki c_cpp_properties.json dosyasını silin
3. Run the "Refresh C++ Intellisense" command in VS Code.
4. Sağ altta platforma benzeyen bir şey görmelisiniz (linuxathena veya windowsx86-64 vb.). Eğer linuxathena değilse tıklayın ve linuxathena (yayın) olarak ayarlayın
5. ~ 1 dakika bekleyin
6. Ana cpp dosyasını açın (bir başlık dosyası değil). Intellisense şimdi çalışıyor olmalı

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
- Smartdashboard will crash on start-up
- Robot Simulation will crash on start-up

Solution-Çözüm: [Media Feature Pack](#) i kurun.

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

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

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

```
python -m robotpy_installer niweb disable
```

Ayrıca bakınız:

[Github issue](#)

9.1.2 Fixed in WPILib 2024.2.1

Visual Studio Code Reports Unresolved Dependency

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

Workaround: This can be safely 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 Önceki Yıllardaki Projeleri İçe Aktarmak

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 Büyük Değişiklikler (Java/C++)

Bu değişiklikler, kitaplıkta yapılan ve kullanıcının bilmesi gereken önemli değişikliklerden *bazılarını* içerir. Bu bütün büyük değişiklikleri içermez, daha fazla değişiklik görmek için bu belgenin diğer kısımlarına bakın.

- 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 *Enabling LiveWindow in Test Mode* to enable it.
- SysId has been rewritten to remove project generation; Replaced with data logging within team robot program

Desteklenen İşletim Sistemleri ve Mimariler:

- Windows 10 & 11, 64 bit. 32 bit ve Arm desteklenmiyor
- Ubuntu 22.04, 64 bit. Glibc > = 2.32 yüklü olan diğer Linux dağıtımları çalışabilir ancak desteklenmemektedir.
- macOS 12 or later, Intel and Arm.

Uyarı: 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

Genel Kütüphane

- 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` from `RobotRelativeSpeeds` 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 interface 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); };`

Son Dakika Değişiklikler

- 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 Simülasyon

- 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

Önemli: 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 Hepsi Bir Arada Yükleyici

- 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 bundled by the installer

9.2.9 Visual Studio Kod Uzantısı

- 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*

10.1 Visual Studio Code Temelleri ve WPILib Uzantısı

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 Hoşgeldiniz Sayfası

|Hoşgeldiniz Ekranı|

Visual Studio Code ilk açıldığında, Hoşgeldiniz sayfası sunulur. Bu sayfada, Visual Studio Code'u özelleştirmenize olanak tanıyan bazı hızlı bağlantıların yanı sıra, IDE'nin temellerini, bazı ipuçlarını ve püf noktalarını öğrenmenize yardımcı olabilecek belgelere ve videolara giden bir dizi bağlantı bulacaksınız.

Sağ üst köşede küçük bir WPILib logosu da görebilirsiniz. Bu, WPILib uzantısı tarafından sağlanan özelliklere erişmenin bir yoludur (aşağıda daha ayrıntılı olarak ele alınmıştır).

10.1.2 Kullanıcı arayüzü

Programa bir göz atmak için en önemli bağlantı muhtemelen temel Kullanıcı Arayüzü belgesidir. Bu belge, kullanıcı arabirimini UI kullanmanın birçok temelini açıklar ve FRC için Visual Studio Code kullanmaya başlamak için ihtiyacınız olan bilgilerin çoğunu sağlar.

10.1.3 Komut Paleti

Komut Paleti, Visual Studio Code'daki (WPILib uzantısından olanlar dahil) hemen hemen her işlev veya özelliğe erişmek veya çalıştırmak için kullanılabilir. Komut Paletine Görünüm menüsünden veya şu tuşlara basılarak erişilebilir `Ctrl+Shift+P` (`Cmd+Shift+P` on macOS). Pencereye metin yazmak, aramayı dinamik olarak ilgili komutlarla daraltacak ve bunları açılır menüde gösterecektir.

Aşağıdaki örnekte “wpilib”, Komut Paleti etkinleştirildikten sonra arama kutusuna yazılır ve listeyi WPILib içeren işlemlere daraltır.

10.1.4 WPILib Uzantısı

[WPILib Komutları]

WPILib uzantısı FRC|reg | projeler ve proje bileşenleri oluşturma, roboRIO'ya kod oluşturma ve indirme ve daha fazlasıyla ilgili özel işlevler. WPILib komutlarına iki yoldan biriyle erişebilirsiniz:

- Komut Paletine “WPILib” yazarak
- Çoğu pencerenin en sağ üst köşesindeki WPILib simgesine tıklayarak. Bu, önceden yazılmış gibi “WPILib” ile Komut Paletini açacaktır

Not: IntelliSense'i garip şekillerde bozduğu bilindiği için VS Code'un FRC kurulumuyla [Visual Studio IntelliCode](#) eklentisinin yüklenmesi **önerilmez**.

Belirli WPILib uzantı komutları hakkında daha fazla bilgi için bu bölümdeki diğer makalelere bakın.

10.2 Visual Studio Code'da WPILib Komutları

Bu belge, WPILib VS Code Uzantısı tarafından sağlanan komutların ve ne yaptıklarının tam bir listesini içerir.

Bu komutlara erişmek için; Komut Paletini açmak için `Ctrl+Shift+P` tuşlarına basın, ardından komut listesini filtrelemek için burada gösterildiği gibi komut adını yazmaya başlayın. Komutu yürütmek için komut adına tıklayın.

- **WPILib: Build Robot Code** - GradleRIO kullanarak açık proje oluşturur.
- **WPILib: Create a new project-WPILib: Yeni bir proje oluşturun** - Yeni bir robot projesi oluşturun
- **WPILib C++: Refresh C++ Intellisense** - C++ Intellisense yapılandırmasına yönelik bir güncellemeyi zorlar.

- **WPILib C++: Select Current C++ Toolchain** - Select the toolchain to use for Intel-lisense (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 -WPILib: Şu anda çalışan görevleri iptal edin** - WPILib uzantısının şu anda çalıştırmakta olduğu tüm görevleri iptal eder.
- **WPILib: Change Auto Save On Deploy Setting** - Bir yükleme yapılırken dosyaların otomatik olarak kaydedilip kaydedilmeyeceğini değiştirir. Bu varsayılan olarak etkinleştirilmiştir.
- **WPILib: Change Auto Start RioLog on Deploy Setting** - RioLog'un yüklemede otomatik olarak başlayıp başlamayacağını değiştirir. Bu varsayılan olarak etkinleştirilmiştir.
- **WPILib: Change Desktop Support Enabled Setting** - Masaüstünde robot kodu oluşturma'nın etkinleştirilip etkinleştirilmeyeceğini değiştirir. Bunu test ve simülasyon amaçları için etkinleştirir. Varsayılan olarak Masaüstü Desteği kapalıdır.
- **WPILib: Change Language Setting** - Şu anda açık olan projenin C++ veya Java olup olmadığını değiştirir.
- **WPILib: Change Run Commands Except Deploy/Debug in Offline Mode Setting-WPILib: Çevrimdışı Modda Dağıtma / Hata Ayıklama Hariç Çalıştırma Komutlarını Değiştirme** - GradleRIO'nun dağıtma / hata ayıklama dışındaki komutlar için Çevrimiçi Modda eğer çalışmıyorsa değiştirin (bağımlılıkları çevrimiçinden otomatik olarak çekmeye çalışır). Varsayılanlar etkin (online mode-çevrimiçi mod).
- **WPILib: Change Run Deploy/Debug Command in Offline Mode Setting-WPILib: Çevrimdışı Mod Ayarında Çalıştırma Dağıtma / Hata Ayıklama Komutunu Değiştirin** - GradleRIO'nun dağıtım / hata ayıklama için Çevrimiçi Modda eğer çalışmıyorsa değiştirin (bağımlılıkları çevrimiçinden otomatik olarak çekmeye çalışacaktır). Varsayılanlar devre dışıdır (çevrimdışı mod).
- **WPILib: Varsayılan Simülasyon Uzantısı Ayarını Seçin** - Simülasyon uzantılarının varsayılan olarak etkinleştirilip etkinleştirilmeyeceğini değiştirin (build.gradle içinde tanımlanan tüm simülasyon uzantıları etkinleştirilecektir)
- **WPILib: Change Skip Tests On Deploy Setting-WPILib: Yükleme Ayarında Testlerin Durumunu Değiştirin** - Yüklemede testlerin atlanıp atlanmayacağını değiştirir. Varsayılan olarak devre dışıdır (testler yüklemeye çalışılır).
- **WPILib: Change Stop Simulation on Entry Setting-WPILib: Giriş Ayarında Simülasyonun Durumunu Değiştir** - Simülasyon çalıştırılırken girişte robot kodunun durdurulup durdurulmayacağını değiştirir. Varsayılan olarak devre dışıdır (girişte durmaz).
- **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-WPILib: Robot Kodunda Hata Ayıklama** - Hata ayıklama modunda robot kodunu oluşturun ve roboRIO'ya yükleyin, sonrasında hata ayıklamaya başlayın.

- **WPILib: Deploy Robot Code-WPILib: Robot Kodunu Yükle** - Robot kodunu oluşturur ve roboRIO'ya yükler.
- **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 computer rather than 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 an existing VS Code Gradle project from 2020-2022. Further documentation is at [importing gradle project](#)
- **WPILib: Install tools from GradleRIO -WPILib: GradleRIO'dan araçlar yükleyin** - WPILib Java araçlarını yükler (örn. SmartDashboard, Shuffleboard, vb.). Bunun varsayılan olarak çevrimdışı yükleyici tarafından yapıldığını unutmayın.
- **WPILib: Manage Vendor Libraries-WPILib: Üretici Kütüphanelerini Yönetin** - 3. şahıs kütüphanelerini yükler/günceller.
- **WPILib: Open API Documentation-WPILib: Açık API Belgeleri** - WPILib Javadocs veya C++ Doxygen belgelerini açar
- **WPILib: Open Project Information -WPILib: Proje Bilgilerini Aç** - Proje bilgilerini içeren bir widget açar (Proje sürümü, uzantı sürümü vb.).
- **WPILib: WPILib Komut Paletini Aç** - Bu komut, bir WPILib Komut Paletini açmak için kullanılır (şuna eşdeğer Ctrl+Shift+P and typing WPILib yazarak)
- **WPILib: Open WPILib Help-WPILib: WPILib Yardımını Aç** - Bu, WPILib belgelerine bağlantı veren basit bir sayfa açar (bu site).
- **WPILib: Reset Ask for WPILib Updates Flag-WPILib Güncellemelerini Sıfırlama** - Bu, mevcut projedeki işareti temizleyerek, daha önce güncelleme yapmamayı seçtiyseniz bir projeyi en son WPILib sürümüne güncellemenize olanak tanır.
- **WPILib: Run a command in Gradle-WPILib: Gradle'da bir komut çalıştırın** - Bu, GradleRIO komut ortamında rastgele bir komut çalıştırmanıza olanak tanır.
- **WPILib: Set Team Number-WPILib: Takım Numarasını Ayarla** - Bir projeyle ilişkili takım numarasını değiştirmek için kullanılır. Bu, yalnızca proje oluştururken başlangıçta belirtilen takım numarasını değiştirmeniz gerektiğinde gereklidir.
- **WPILib: Set VS Code Java Home to FRC Home -WPILib: VS Code Java Home'u FRC Home'a Ayarla** - VS Code Java Home değişkenini FRC uzantısı tarafından keşfedilen Java Home'a yönlendirecek şekilde ayarlar. Bu, IntelliSense ayarlarının WPILib yapı ayarlarıyla uyumlu olduğundan emin olmak için çevrimdışı yükleyici kullanılmıyorsa gereklidir.
- **WPILib: Show Log Folder-WPILib: Log Klasörünü Göster** - WPILib uzantısının dahili logları sakladığı klasörü gösterir. Bu, WPILib geliştiricilerine bir uzantı sorunu bildirirken/hata ayıklarken yararlı olabilir.
- **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-WPILib: RioLog'u Başlat** - Bu, bir robot programından konsol çıkışını görüntülemek için kullanılan RioLog ekranını başlatır.
- **WPILib: Start Tool-WPILib: Başlangıç Aracı** - Bu, WPILib araçlarını (örn. SmartDashboard, Shuffleboard, vb.) VS Code içinden başlatmanıza olanak tanır.

- **WPILib: Test Robot Code-WPILib: Robot Kodunu Test Etme** - Bu, mevcut robot kodu projesini oluşturur ve oluşturulan tüm testleri çalıştırır. Bu, Masaüstü Desteğinin-Desktop Support, Etkin-Enabled olarak ayarlanmasını gerektirir.

10.3 Bir Robot Programı Oluşturmak

Her şey kurulduktan sonra, bir robot programı oluşturmaya hazırız. WPILib, robot programları için çeşitli şablonlarla birlikte gelir. Bu şablonların kullanımı yeni kullanıcılar için şiddetle tavsiye edilir; ancak, ileri düzey kullanıcılar kendi robot kodlarını sıfırdan yazmakta özgürdür.

10.3.1 Temel Sınıf Seçme

WPILib robot programı şablonlarından birini kullanarak bir projeye başlamak için, kullanıcılar önce robotları için bir temel sınıf seçmelidir. Kullanıcılar, robot programının ana akışını kontrol eden birincil Robot sınıfını oluşturmak için bu temel sınıfları alt sınıflandırır. Temel sınıf için üç seçenek mevcuttur:

TimedRobot

Documentation: [Java](#) - [C++](#)

Kaynak: [Java <https://github.com/wpilibsuite/allwpilib/blob/main/wpilibj/src/main/java/edu/wpi/first/wpilibj/RobotBase.java>](https://github.com/wpilibsuite/allwpilib/blob/main/wpilibj/src/main/java/edu/wpi/first/wpilibj/RobotBase.java)
 __ - [C++ <https://github.com/wpilibsuite/allwpilib/blob/main/wpilibc/src/main/native/cpp/TimedRobot.cpp>](https://github.com/wpilibsuite/allwpilib/blob/main/wpilibc/src/main/native/cpp/TimedRobot.cpp)

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++](#))

Not: Bazı bilgilendirici yorumları ve otonom örneği kaldıran bir *TimedRobot Skeleton* şablonu mevcuttur. Zaten *TimedRobot* u biliyorsanız bunu kullanabilirsiniz. Aşağıda gösterilen örnek *TimedRobot Skeleton* içindir.

JAVA

```

7 import edu.wpi.first.wpilibj.TimedRobot;
8
9 /**
10  * The VM is configured to automatically run this class, and to call the functions
11  * corresponding to
12  * each mode, as described in the TimedRobot documentation. If you change the name of
13  * this class or
14  * the package after creating this project, you must also update the build.gradle
15  * file in the

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
13  * project.
14  */
15  public class Robot extends TimedRobot {
16      /**
17       * This function is run when the robot is first started up and should be used for
18       * any
19       * initialization code.
20       */
21      @Override
22      public void robotInit() {}
23
24      @Override
25      public void robotPeriodic() {}
26
27      @Override
28      public void autonomousInit() {}
29
30      @Override
31      public void autonomousPeriodic() {}
32
33      @Override
34      public void teleopInit() {}
35
36      @Override
37      public void teleopPeriodic() {}
38
39      @Override
40      public void disabledInit() {}
41
42      @Override
43      public void disabledPeriodic() {}
44
45      @Override
46      public void testInit() {}
47
48      @Override
49      public void testPeriodic() {}
50
51      @Override
52      public void simulationInit() {}
53
54      @Override
55      public void simulationPeriodic() {}
56  }
```

C++

```
5  #include "Robot.h"
6
7  void Robot::RobotInit() {}
8  void Robot::RobotPeriodic() {}
9
10 void Robot::AutonomousInit() {}
11 void Robot::AutonomousPeriodic() {}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

12
13 void Robot::TeleopInit() {}
14 void Robot::TeleopPeriodic() {}
15
16 void Robot::DisabledInit() {}
17 void Robot::DisabledPeriodic() {}
18
19 void Robot::TestInit() {}
20 void Robot::TestPeriodic() {}
21
22 void Robot::SimulationInit() {}
23 void Robot::SimulationPeriodic() {}
24
25 #ifndef RUNNING_FRC_TESTS
26 int main() {
27     return frc::StartRobot<Robot>();
28 }
29 #endif

```

Periyodik yöntemler varsayılan olarak her 20 ms’de bir çağrılır. Bu, istenen yeni güncelleme oranıyla üst sınıf kurucusunu çağırarak değiştirilebilir.

Tehlike: 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++](#)

Kaynak: Java <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibj/src/main/java/edu/wpi/first/wpilibj/RobotBase.java>> - C++ <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibc/src/main/native/cppcs/RobotBase.cpp>>

The RobotBase class is the most minimal base-class offered, and is generally not recommended for direct use. No robot control flow is handled for the user; everything must be written from scratch inside the startCompetition() method. The template by default showcases how to process the different operation modes (teleop, auto, etc).

Not: Boş bir `startCompetition()` metodu sunan bir `RobotBase` Skeleton şablonu mevcuttur.

Command Robot-Komut Robotu

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

Teams using Command Robot should see the [Command-Based Programming Tutorial](#).

Romi

Romi kullanan takımlar Romi - Timed veya Romi - Command Bot şablonunu kullanmalıdır.

Romi - Zamanlı

Romi - Timed şablonu, bir `arcadeDrive` (`double xaxisSpeed`, `double zaxisRotate`) metodunu ortaya çıkaran bir `RomiDrivetrain` sınıfı sağlar. Bu `arcadeDrive` işlevini beslemek kullanıcıya bağlıdır.

Bu sınıf ayrıca Romi'nin yerleşik kodlayıcılarının alınması ve sıfırlanması için işlevler de sağlar.

Romi - Komut Robotu

Romi - Command Bot şablonu, bir `arcadeDrive` (`double xaxisSpeed`, `double zaxisRotate`) metodunu ortaya çıkaran bir `RomiDrivetrain` alt sistemi sağlar. Bu `arcadeDrive` işlevini beslemek kullanıcıya bağlıdır.

Bu alt sistem aynı zamanda Romi'nin yerleşik kodlayıcılarının alınması ve sıfırlanması için işlevler de sağlar.

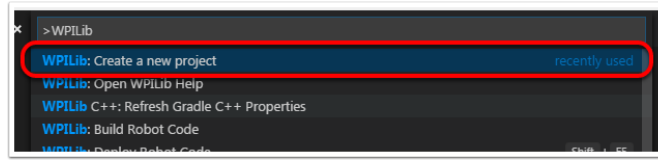
Temel Sınıf Kullanmama

İstenirse, kullanıcılar bir temel sınıfı tamamen çıkarabilir ve programlarını başka herhangi bir programda olduğu gibi basitçe `main()` yöntemine yazabilirler. Bu *kesinlikle* tavsiye edilmez - kullanıcılar robot kodlarını yazarken “tekerleği yeniden icat etmemelidir” - ancak program akışları üzerinde mutlak kontrole sahip olmak isteyenler için desteklenir.

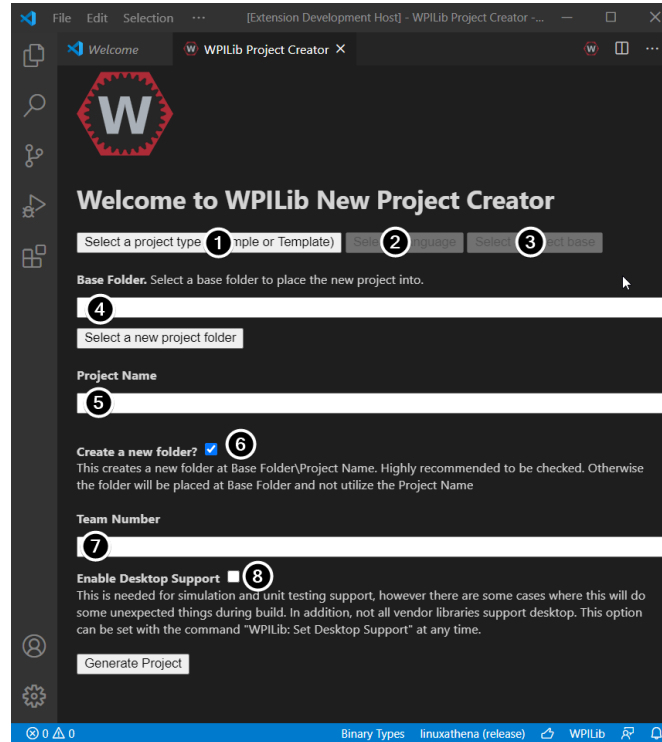
Uyarı: Kullanıcılar, ne yaptıklarından kesinlikle emin olmadıkları sürece bir robot programının `main()` yöntemini *değiştirmemelidir*.

10.3.2 Yeni bir WPILib Projesi Oluşturma

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:



Bu, "New Project Creator Window-Yeni Proje Oluşturma Penceresi:" getirecektir:



Yeni Proje Oluşturma Penceresinin öğeleri aşağıda açıklanmıştır:

1. **Project Type -Proje Türü:** Oluşturmak istediğimiz proje türü. Bu örnek bir proje veya WPILib tarafından sağlanan proje şablonlarından biri olabilir. Robot temel sınıflarının her biri için şablonlar mevcuttur. Ek olarak, *Command-based* projeleri için bir şablon mevcuttur; bu projeler TimedRobot temel sınıfı üzerine inşa edilmiştir, ancak bir dizi ek özellik içerir - bu tür robot programı yeni ekipler için şiddetle tavsiye edilir.
2. **Language -Dil:** Bu proje için kullanılacak dildir (C++ veya Java).
3. **** Base Folder-Temel Klasör **:** Bu bir şablon projeyse, bu kullanılacak şablonun türünü belirtir.

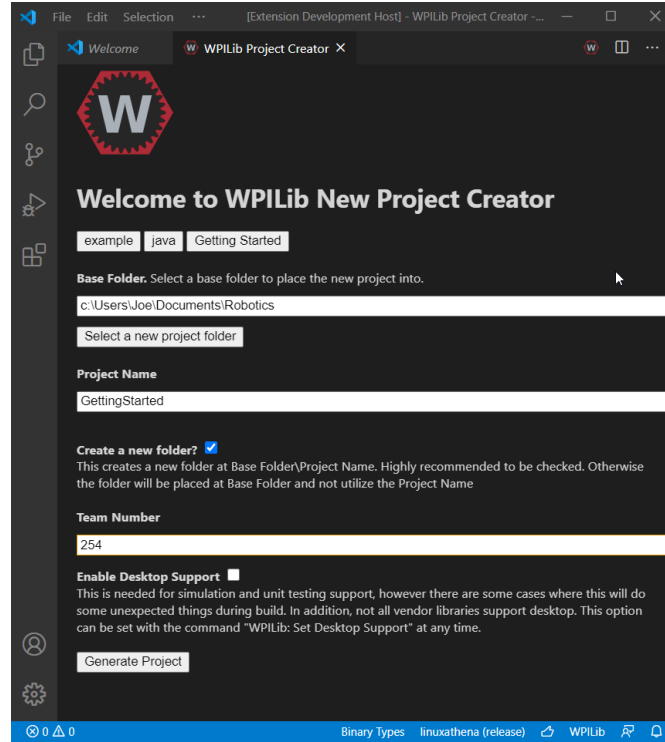
4. **Project Location -Proje Konumu:** Robot projesinin konumlandırılacağı klasörü belirler.
5. **Project Name -Proje Adı:** Robot projesinin adı. Bu ayrıca Create New Folder -Yeni Klasör Oluştur kutusu işaretlendiğinde proje klasörünün verileceği adı da belirtir.
6. **** Create a New Folder-Yeni Klasör Oluştur **:** Bu işaretlenirse, projeyi önceden belirtilen klasör içinde tutmak için yeni bir klasör oluşturulur. *not-işaretli değilse *, proje doğrudan önceden belirtilen klasörde yer alacaktır. Klasör boş değilse ve bu kontrol edilmezse bir hata atılacaktır.
7. **Team Number -Takım Numarası:** Proje içindeki paket isimleri için ve kodu yüklerken robotu bulmak için kullanılacak proje takım numarası.
8. ****Enable Desktop Support-Masaüstü Desteğini Etkinleştir **:** Birim testi ve simülasyonu etkinleştirir. WPILib bunu desteklerken, üçüncü taraf yazılım kitaplıkları desteklemeyebilir. Kitaplıklar masaüstünü desteklemiyorsa, kodunuz derlenmeyebilir veya çökebilir. Birim testi veya simülasyon gerekmedikçe ve tüm kitaplıklar bunu desteklemedikçe, işaretlenmeden bırakılmalıdır.

Yukarıdakilerin tümü yapılandırıldıktan sonra, “Generate Project -Proje Oluştur” u tıklayın ve robot projesi oluşturulacaktır.

Not: Proje oluşturmadaki herhangi bir hata, ekranın sağ alt köşesinde görünecektir.

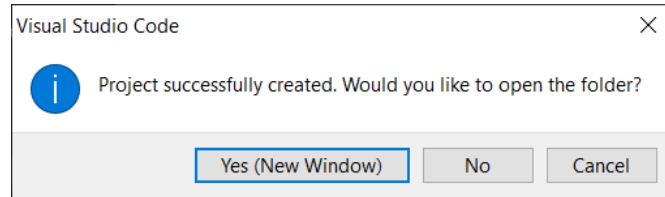
Uyarı: Creating projects on OneDrive is not supported as OneDrive’s caching interferes with the build system. Some Windows installations put the Documents and Desktop folders on OneDrive by default.

Tüm seçenekler seçildikten sonra bir örnek aşağıda gösterilmiştir.

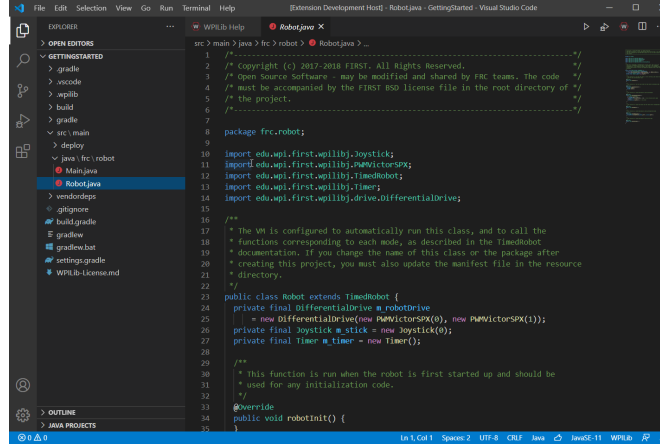


10.3.3 Yeni Projeyi Açmak

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+0** (or just **Command+0** on macOS) and select the folder where we saved our project.

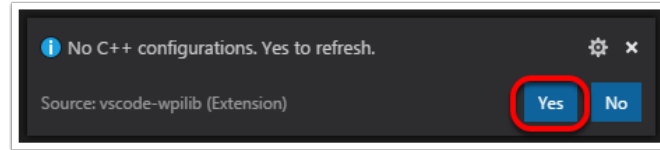


Açıldıktan sonra solda proje hiyerarşisini göreceğiz. Dosyaya çift tıklamak o dosyayı düzenleyicide açacaktır.



10.3.4 C++ Yapılandırmaları (Yalnızca C++)

C++ projeleri için IntelliSense'i kurmak için bir adım daha vardır. Ne zaman bir proje açsak, sağ alt köşede C++ yapılandırmalarını yenilememizi isteyen bir pencere açmalıyız. IntelliSense'i kurmak için "Yes-Evet" i tıklayın.



10.4 3. Taraf Kütüphaneleri

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.

Uyarı: 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.

Not: 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.

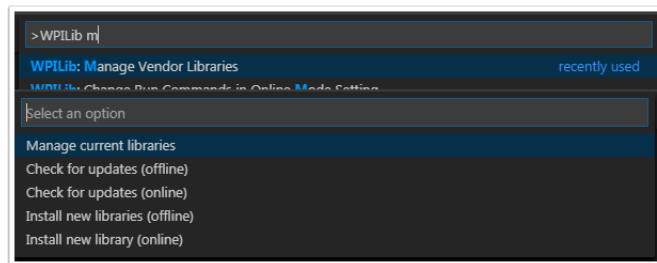
How Does It Work? - Python

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.

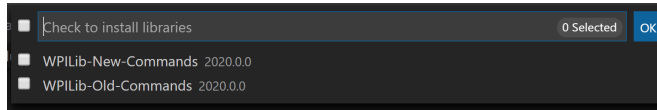
Installing Libraries

Java/C++

VS Code



Çevrimdışı bir sağlayıcı tarafından kurulan bir satıcı kütüphanesini eklemek için, şu tuşa basın: `kbd:Ctrl + Shift + P` ve WPILib yazın veya WPILib Komut Paletini açmak için sağ üstteki WPILib simgesine tıklayın ve yazmaya başlayın: *Manage Vendor Libraries*, ardından menüden *Install new libraries (offline)* seçeneğini seçin.



Her birinin yanındaki kutuyu işaretleyerek projeye eklemek için istenen kitaplıkları seçin ve ardından tıklayın *OK*. JSON dosyası, kütüphaneyi projeye bağımlılık olarak ekleyerek projedeki `vendordeps` klasörüne kopyalanacaktır.

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)

Satıcıların isteğe bağlı olarak doldurabileceği JSON dosyasının bir kısmı, çevrimiçi bir güncelleme konumudur. Bir kitaplıkta uygun bir konum belirtilmişse, çalıştırarak *Check for updates (online)*, kitaplığın daha yeni bir sürümünün uzak konumdan mevcut olup olmadığını kontrol eder.

Removing a Library Dependency

Bir projeden kütüphane bağımlılığını kaldırmak için: `guilabel:Manage Current Libraries` menüsünden *Manage Vendor Libraries* seçeneğini seçin, kaldırılacak kitaplıklar için kutuyu işaretleyin ve tıklayın OK. Bu kitaplıklar projeden bağımlılıklar olarak kaldırılacaktır.

Command-Line

Satıcı URL'sinden bir satıcı kitaplığı bağımlılığı eklemek, bir gradle görevi aracılığıyla komut satırı aracılığıyla da yapılabilir. Proje kökünde bir komut satırı örneği açın ve `radlew vendordep --url = yazın`; burada `` , satıcının JSON URL'sidir. Bu, satıcı kitaplığı bağımlılığı JSON dosyasını projenin `` vendordeps`` klasörüne ekleyecektir. Satıcı kitaplıkları da aynı şekilde güncellenebilir.

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`

Ayrıca bakınız:

[pyproject.toml usage](#)

10.4.3 Kütüphaneler

WPILib Libraries

Command Library

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

Java/C++

New Command Library

Python

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

Romi Library

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

Java/C++

[Romi Vendordep.](#)

Python

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

XRP Library

An XRP Library has been created to contain several helper classes that are used in the XRP - Reference 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>

Not: 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

[Fusion Driver ile oynamak](#)- Venom motor/kontrol cihazı dahil tüm PWF cihazları için kütüphane

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, ve Sensor Fusion kütüphaneleri

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](#) [REVLlib](#) - Library for all REV devices including SPARK Flex, SPARK MAX, and Color Sensor V3

Java/C++

<https://software-metadata.revrobotics.com/REVLlib-2024.json>

Python

Community-supported packages:

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

Topluluk Kütüphaneleri

[PhotonVision](#) - PhotonVision CV yazılımı için Kütüphane

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://brncbotz3481.github.io/YAGSL-Lib/yagsl/yagsl.json>

Python

Not available

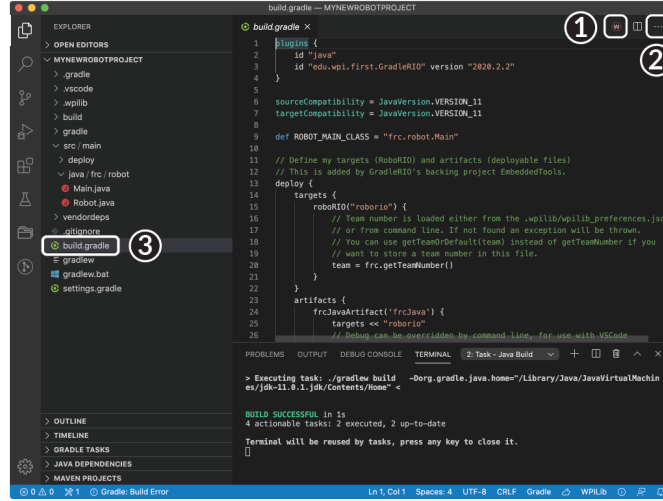
10.5 Robot Kodunun Oluşturulması ve Yüklenmesi

RoboRIO üzerinde çalışabilmek için, robot projeleri (“built”) ile derlenmeli ve yüklenmelidir. Kod, robot kontrolcüsünde yerel olarak derlenmediğinden, bu “çapraz derleme” olarak bilinir.

Bir robot projesi oluşturmak veya yüklemek için şunlardan birini yapın:

1. Komut Paletini açın ve “Build Robot Code-Robot Kodu Oluştur” seçin
2. VS Code penceresinin sağ üst köşesindeki üç nokta ile gösterilen kısayol menüsünü açın ve “Build Robot Code-Robot Kodu Oluştur” seçin.

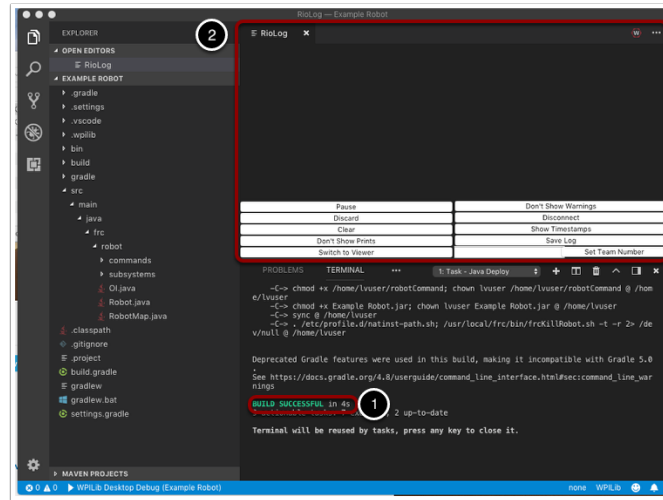
- Proje hiyerarşisinde build.gradle dosyasına sağ tıklayın ve “Build Robot Code-Robot Kodu Oluştur” seçin



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.

Uyarı: 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 Konsol Çıktısını Görüntüleme

Metin tabanlı-text based programların konsol çıktısını görüntülemek için roboRIO bir NetConsole uyarlar. RoboRIO'dan NetConsole çıktısını görüntülemenin iki ana yolu vardır: FRC Driver Station'daki Console Viewer ve VS Code'daki Riolog eklentisi.

Not: RoboRIO'da, NetConsole yalnızca program çıktısı içindir. Sistem konsolu ile etkileşim kurmak istiyorsanız, SSH veya Serial console kullanmanız gerekecektir.

10.6.1 Console Viewer

Console Viewer'ı Açma

|Console Viewer'ı Açma|

Konsol Görüntüleyiciyi açmak için önce FRC|reg| Sürücü İstasyonu. Ardından, mesaj görüntüleyici penceresinin (1) üst kısmındaki dişli çarkı tıklayın ve "Konsolu Görüntüle" yi seçin.

Console Viewer Penceresi

|Console Viewer Penceresi|

Console Viewer penceresi robot programımızın çıktısını yeşil renkte gösterir. Sağ üstteki çark, pencereyi temizleyebilir ve görüntülenen mesajların düzeyini ayarlayabilir.

10.6.2 Riolog VS Code Eklentisi

Riolog eklentisi, NetConsole çıktısını VS Code'da görüntülemek için kullanılabilen bir VS Code görünümüdür (orijinal Eclipse sürümü için atıf : Manuel Stoeckl, FRC1511).

RioLog View'ı Açma

|RioLog View'ı Açma|

Varsayılan olarak, RioLog görünümü her roboRIO dağıtımının sonunda otomatik olarak açılacaktır. RioLog görünümünü manuel olarak başlatmak için, komut paletini açmak ve "RioLog" yazmaya başlamak için: kbd:Ctrl + Shift + P tuşlarına basın, ardından WPILib: RioLog'u Başlat seçeneğini seçin.

Riolog Penceresi

[Riolog Penceresi]

RioLog görünümü üst bölmede görünmelidir. Riolog, konsolu işlemek için bir dizi kontrol içerir:

- ****Pause/Resume Display-Görüntülemeyi Duraklat/Sürdür **** - Bu, görüntülemeyi duraklatır/devam ettirir. Arka planda, yeni paketler alınmaya devam edecek ve devam et düğmesine tıklandığında görüntülenecektir.
- ****Discard/Accept Incoming-Gelenleri İptal Et/Kabul Et **** - Bu, yeni paketlerin kabul edilip edilmeyeceğini değiştirir. Paketler atılırken ekran duraklatılacak ve elde edilen tüm paketler atılacaktır. Düğmeye tekrar tıklamak, paketleri almaya devam edecektir.
- **Clear-Temizle** - Bu, ekranın mevcut içeriğini temizler.
- **Don't Show/Show Prints-Printleri Göster/Gösterme** - Bu, print ifadeleri olarak kategorize edilen mesajları gösterir veya gizler.
- **Switch to Viewer-Görüntüleyiciye Geç** - Bu, kaydedilen log dosyaları için görüntüleyiciye geçer.
- **Don't Show/Show Warnings-Uyarıları Göster/Gösterme** - Bu, uyarı olarak kategorize edilen mesajları gösterir veya gizler.
- **Disconnect/Reconnect-Bağlantıyı Kes/Yeniden Bağlan** - Bu, konsol akışına bağlantıyı keser veya yeniden bağlanır.
- **Show/Don't Show Timestamps-Zaman Damgalarını Göster/Gösterme** - Penceredeki iletilerde zaman damgalarını gösterir veya gizler.
- **Save Log-Log Kaydet** - Log içeriğini kaydedip görüntüleyebileceğiniz veya daha sonra RioLog görüntüleyiciyle açabileceğiniz bir dosyaya kopyalar (bkz. Switch to Viewer above Yukarıdaki Görüntüleyiciye Geç)
- **Set Team Number-Takım Numarasını Ayarla** - Konsol akışına bağlanmak için roboRIO'nun ekip numarasını ayarlar, RioLog yükleme işlemiyle başlatılırsa otomatik olarak ayarlanır.

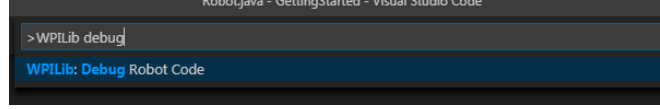
10.7 Bir Robot Programında Debugging -Hata Ayıklama

Kaçınılmaz olarak, bir program beklediğimiz şekilde davranmayabilir. Bu gerçekleştiğinde, programın bunu neden yaptığını yaptığını anlamamız gerekir, böylece yapmasını istediğimiz şeyi yapmasını sağlayabiliriz. Bu tür istenmeyen program davranışına “bug” hata adı verilir ve düzeltme işlemine “debugging” hata ayıklama denir.

Hata ayıklayıcı, bir programda hata ayıklamaya yardımcı olmak için program akışını kontrol etmek ve değişkenleri izlemek için kullanılan bir araçtır. Bu bölümde bir FRC ® için bir hata ayıklama oturumunun nasıl ayarlanacağı açıklanacaktır. robot programı.

Not: Programlarında hata ayıklaması gereken, ancak bir hata ayıklayıcının -debugger nasıl kullanılacağını bilmeyen/öğrenmeye vakti olmayan yeni başlayan kullanıcılar için, genellikle, yalnızca ilgili program durumunu konsola yazdırarak bir programda hata ayıklamak mümkündür. Ancak, öğrencilerin sonunda bir hata ayıklayıcı-debugger kullanmayı öğrenmeleri şiddetle tavsiye edilir.

10.7.1 Hata Ayıklayıcıyı Çalıştırma



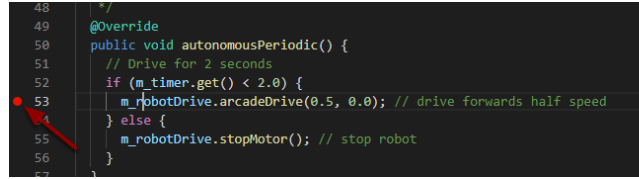
WPILib önceden doldurulmuş Komut paletini açmak için kbd: *Ctrl + Shift + P* ye basın ve `WPILib` yazın veya *WPILib Menu Item* Menü Ögesi'ne tıklayın. Hata ayıklamayı başlatmak için Debug yazın ve Debug Robot Code menü ögesini seçin. Kod roboRIO'ya indirilecek ve hata ayıklamaya başlayacaktır.

10.7.2 Breakpoints-Duraksatma Noktaları

“breakpoint-kesme noktası”, kullanıcının program durumunu inceleyebilmesi için hata ayıklayıcının program yürütmesini duraklatacağı bir kod satırıdır. Bu, kullanıcının programın beklenen davranıştan tam olarak nerede saptığını belirlemek için sorunlu koddaki belirli noktalarda programı duraklatmasına izin verdiği için hata ayıklama sırasında son derece yararlıdır.

Hata ayıklayıcı, karşılaştığı ilk breakpoint da otomatik olarak durur.

Breakpoint-Duraksatma Noktası Belirleme



Kullanıcı programınızda bir kesme noktası ayarlamak için kaynak kodu penceresinin sol kenar boşluğunu (satır numarasının solunda) tıklayın : Küçük kırmızı bir daire, ilgili satırda kesme noktasının ayarlandığını gösterir.

10.7.3 Baskı İfadeleri ile Hata Ayıklama

Programınızda hata ayıklamanın başka bir yolu, kodunuzda yazdırma deyimleri kullanmak ve bunları RioLog'u Visual Studio Code veya Driver Station'da kullanarak görüntülemektir. Özellikle yüksek miktarlarda kullanıldıklarında çok verimli olmadıkları için baskı tabloları dikkatle eklenmelidir. Döngü taşmalarına neden olabileceğinden rekabet için kaldırılmalıdırlar.

JAVA

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

C++

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

10.7.4 Debugging with NetworkTables

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

10.7.5 Daha fazla bilgi edin

- VS Code ile hata ayıklama hakkında daha fazla bilgi edinmek için bu [link](#) bağlantısına bakın.
- Bu VS Code *article* <<https://code.visualstudio.com/docs/editor/editingevolved>> makalesinde bahsedilen özelliklerden bazıları, kodunuzla ilgili sorunları anlamana ve teşhis etmenize yardımcı olacaktır. Hızlı Onarım (sarı ampul) özelliği, neyin içe aktarılacağı dahil olmak üzere çeşitli sorunlarda çok yardımcı olabilir.
- Bu kadar çok sorunda hata ayıklama zorunluluğunu önlemenin en iyi yollarından biri Birim Testi yapmaktır.
- Robotunuzun çalıştığını doğrulamak *Simulation* gerçek robot üzerinde karmaşık hata ayıklama yapmak zorunda kalmayı önlemek için de harika bir yoldur.

10.8 Importing Last Year's Robot Code

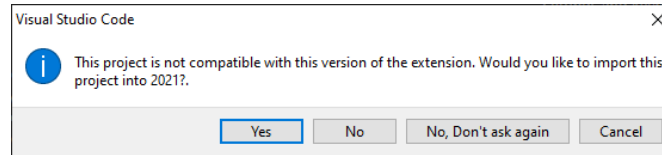
Due to changes in the project, it is necessary to update the build files for a previous years Gradle project. It is also necessary to import vendor libraries again, since last year's vendor libraries must be updated to be compatible with this year's projects.

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.

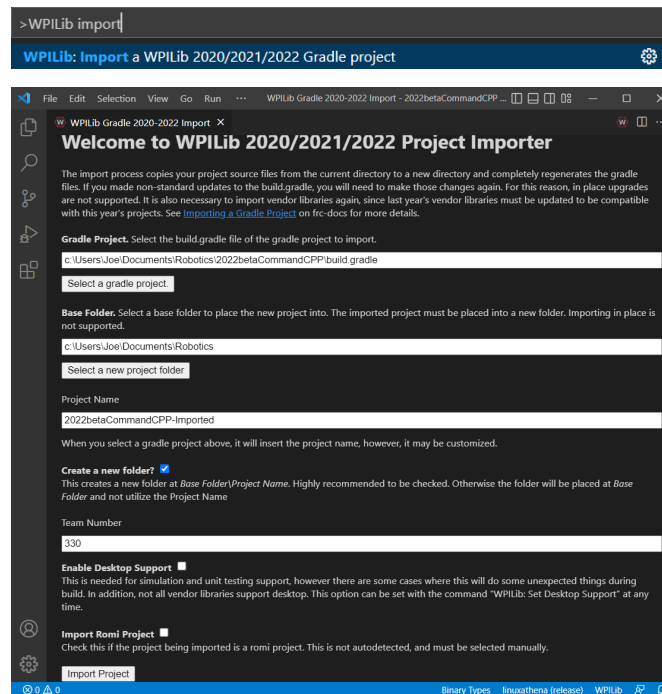
Önemli: The import process copies your project source files from the current directory to a new directory and completely regenerates the gradle files. Additionally, it updates the code for the package changes made in 2023. If you made non-standard updates to the build. gradle, you will need to make those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

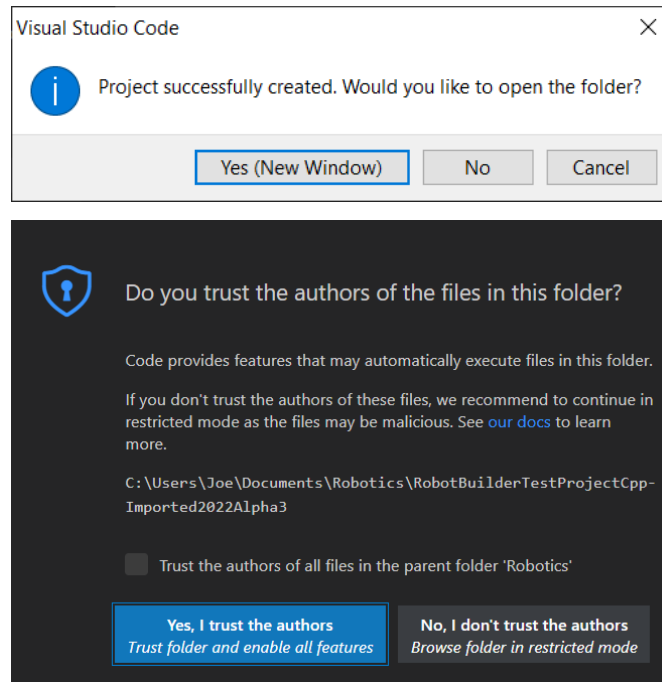
1. **Gradle Project:** Selects the project to be imported. Users should select the build.gradle file in the root directory of the gradle project.
2. **Project Location:** This determines the folder in which the robot project will be located.
3. **Project Name:** The name of the robot project. This also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

Uyarı: 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.

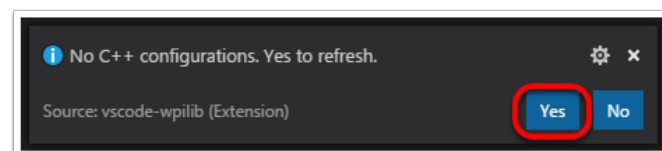
The gradle project will be upgraded and copied into the new project directory. You can then either open the new project immediately using the pop-up below or open it later using the Ctrl+0 (or Command+0 for macOS) shortcut.



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.

11.1 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)

Not: SmartDashboard and Shuffleboard have a long history of aiding FRC teams. However, they do not have a person to maintain them so are not receiving bug fixes or improvements. Notably, Shuffleboard may experience performance issues on some machines under certain scenarios. PRs from external contributors will be reviewed.

LabVIEW Dashboard (Driver / Programming) - easy to use and provides a lot of features straight out of the box like: camera streams, autonomous selection, and joystick feedback. It can be customized using LabVIEW by creating a new Dashboard project. While it *can be used* by Java or C++ teams, they generally prefer SmartDashboard or Shuffleboard which can be customized in their respective language.

SmartDashboard (Driver) - simple and efficient dashboard that uses relatively few computer resources. It does not have the fancy look or some of the features Shuffleboard has, but it displays network tables data with a variety of widgets without bogging down the driver station computer.

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

Glass (Programming) - robot data visualization tool. Its GUI is extremely similar to that of the *Simulation GUI*. In its current state, it is meant to be used as a programmer's tool rather than a proper dashboard in a competition environment, with a focus on high performance real time plotting.

AdvantageScope (Programming) - robot diagnostics, log review/analysis, and data visualization application. It reads the WPILib Data Log (.wpilog) and Driver Station Log (.dslog / .dsevents) file formats, plus live robot data viewing.

11.1.4 Third Party Dashboards

FRC Web Components (Driver) - A web-based dashboard that can be installed as a standalone application, or as a JavaScript package for custom dashboard solutions.

Elastic (Driver) - simple and modern Shuffleboard alternative made by Team 353. It is meant to serve as a dashboard for competition but can also be used for testing. It features draggable and resizable card widgets.

QFRCDashboard (Driver) - described as reliable, high-performance, low-footprint dashboard. QFRCDashboard has been specifically designed to use as few resources as possible.

11.2 Shuffleboard

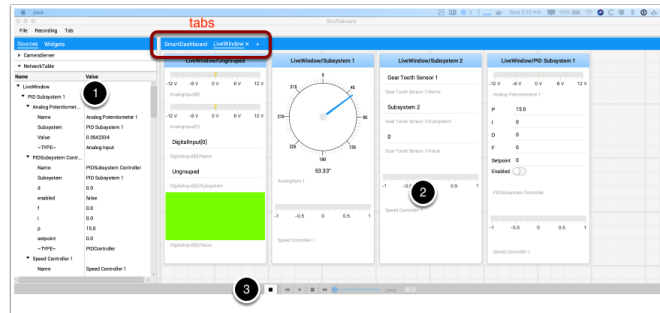
Shuffleboard is a straightforward and easily customizable drivetrain focused dashboard. It displays network tables data using a variety of widgets that can be positioned and controlled with robot code. It includes many extra features like: tabs, recording / playback, and advanced custom widgets.

11.2.1 Shuffleboard - Getting Started

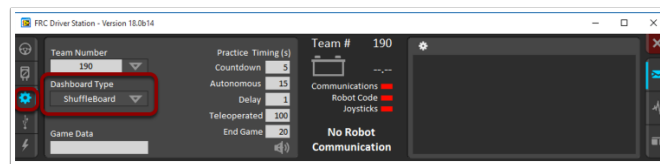
Tour of Shuffleboard

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:

- Graphics is based on **JavaFX**, the Java graphics standard. Each of the components has an associated style sheet so it becomes possible to have different “skins” or “themes” for Shuffleboard. We supply default light and dark themes.
 - Shuffleboard supports **multiple sheets for the display of your data**. In fact you can create a new sheet (shown as a tab in the Shuffleboard window) and indicate if and which data should be autopopulated on it. By default there is a Test tab and a SmartDashboard tab that are autopopulated as data arrives. Other tabs might be for robot debugging vs. driving.
 - Graphical **display elements (widgets) are laid out on a grid** to keep the interface clean and easy to read. You can change the grid size to have more or less resolution in your layouts and visual cues are provided to help you change your layout using drag and drop. Or you can choose to turn off the grid lines although the grid layout is preserved.
 - Layouts are saved and the previous layout is instantiated by default when you run shuffleboard again.
 - There is a **record and playback** feature that lets you review the data sent by your robot program after it finishes. That way you can carefully review the actions of the robot if something goes wrong.
 - **Graph widgets are available for numeric data** and you can drag data onto a graph to see multiple points at the same time and on the same scale.
 - 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:** Here are data sources from which you can choose values from NetworkTables or other sources to display by dragging a value into one of the tabs
 2. **Tab panes:** This is where you data is displayed from the robot or other sources. In this example it is Test-mode subsystems that are shown here in the LiveWindow tab. This area can show any number of tabbed windows, and each window has it's own set of properties like grid size and auto-populate.
 3. **Record/playback controls:** set of media-like controls where you can playback the current session to see historical data



Starting Shuffleboard



You can start Shuffleboard in one of four ways:

1. You can automatically start it when the Driver Station starts by setting the “Dashboard Type” to Shuffleboard in the settings tab as shown in the picture above.
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. You can run it by double-clicking on the shuffleboard.XXX file (where XXX is .vbs on Windows and .py on Linux or macOS) in ~/WPILib/YYYY/tools/ (where YYYY is the year and ~ is C:\Users\Public on Windows). This is useful on a development system that does not have the Driver Station installed such as a macOS or Linux system.
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.

Not: The .vbs (Windows) and .py (macOS/Linux) scripts help launch the tools using the correct JDK.

Getting robot data onto the dashboard

The easiest way to get data displayed on the dashboard is simply to use methods in the SmartDashboard class. For example to write a number to Shuffleboard write:

JAVA

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

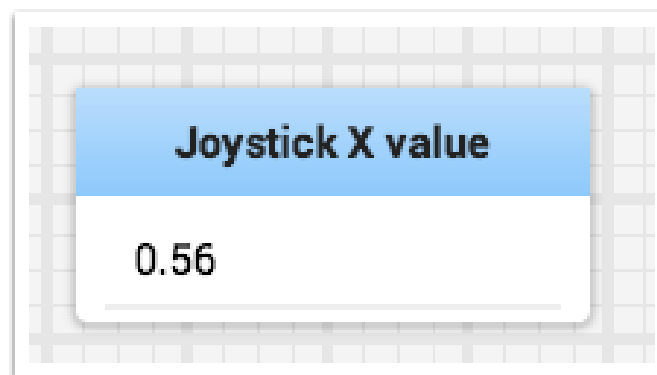
C++

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

PYTHON

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

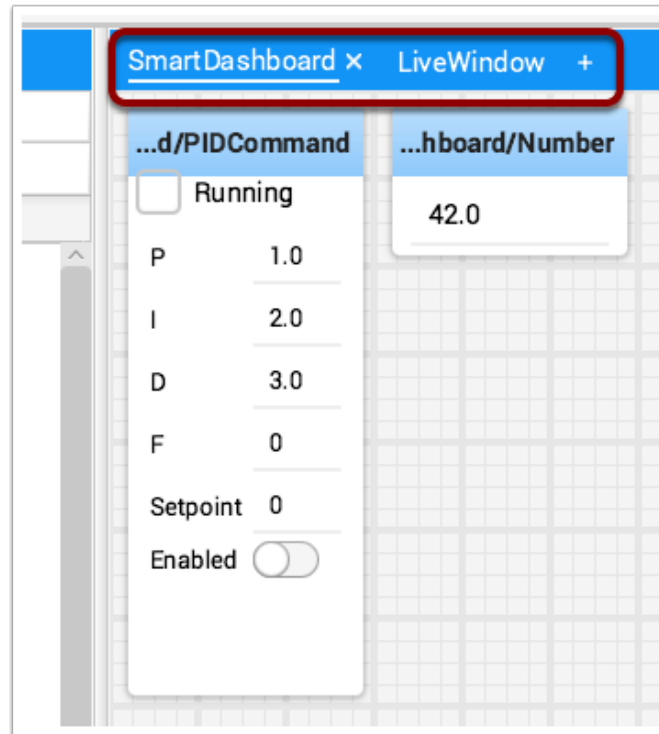
to see a field displayed with the label “Joystick X value” and a value of the X value of the joystick. Each time this line of code is executed, a new joystick value will be sent to Shuffleboard. Remember: you must write the joystick value whenever you want to see an updated value. Executing this line once at the start of the program will only display the value once at the time the line of code was executed.



Displaying data from your robot

Your robot can display data in regular operating modes like Teleop and Autonomous modes but you can also display the status and operate all the robot subsystems when the robot is switched to Test mode. By default you’ll see two tabs when you start Shuffleboard, one for Teleop/Autonomous and another for Test mode. The currently selected tab is underlined as can be seen in the picture below.

Often debugging or monitoring the status of a robot involves writing a number of values to the console and watching them stream by. With Shuffleboard you can put values to a GUI that is



automatically constructed based on your program. As values are updated, the corresponding GUI element changes value - there is no need to try to catch numbers streaming by on the screen.

Displaying values in normal operating mode (autonomous or teleop)

JAVA

```
protected void execute() {
    SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge());
    SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition());
    SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle());
    SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder());
    SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder());
    SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle());
    SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage());
    SmartDashboard.putNumber("RPM", shooter.getRPM());
}
```

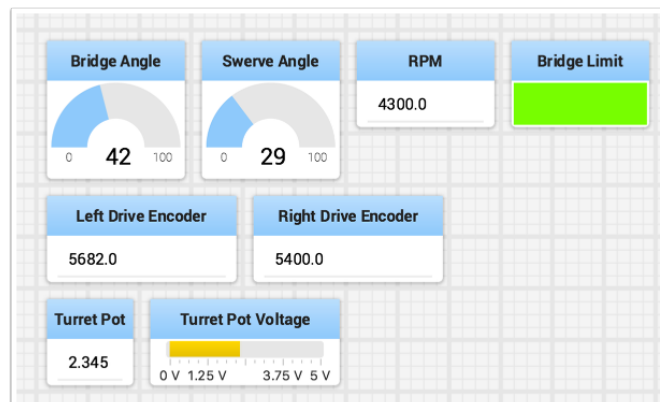
C++

```
frc::SmartDashboard::PutBoolean("Bridge Limit", bridgeTipper.AtBridge());
frc::SmartDashboard::PutNumber("Bridge Angle", bridgeTipper.GetPosition());
frc::SmartDashboard::PutNumber("Swerve Angle", drivetrain.GetSwerveAngle());
frc::SmartDashboard::PutNumber("Left Drive Encoder", drivetrain.GetLeftEncoder());
frc::SmartDashboard::PutNumber("Right Drive Encoder", drivetrain.GetRightEncoder());
frc::SmartDashboard::PutNumber("Turret Pot", turret.GetCurrentAngle());
frc::SmartDashboard::PutNumber("Turret Pot Voltage", turret.GetAverageVoltage());
frc::SmartDashboard::PutNumber("RPM", shooter.GetRPM());
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge())
SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition())
SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle())
SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder())
SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder())
SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle())
SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage())
SmartDashboard.putNumber("RPM", shooter.getRPM())
```

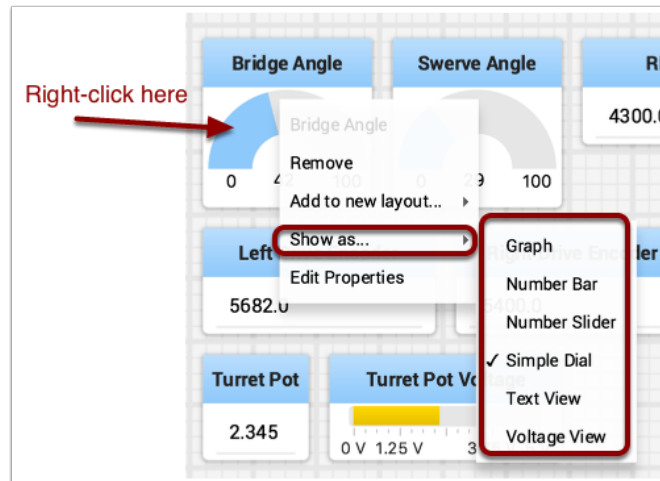


You can write Boolean, Numeric, or String values to Shuffleboard by simply calling the correct method for the type and including the name and the value of the data, no additional code is required.

- Numeric types such as char, int, long, float or double call `SmartDashboard.putNumber("dashboard-name", value)`.
- String types call `SmartDashboard.putString("dashboard-name", value)`
- Boolean types call `SmartDashboard.putBoolean("dashboard-name", value)`

Changing the display type of data

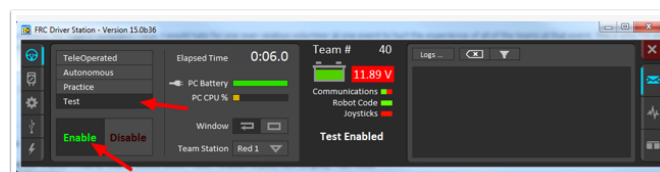
Depending on the data type of the values being sent to Shuffleboard you can often change the display format. In the previous example you can see that number values were displayed as either decimal numbers, a dial to better represent angles, and as a voltage view for the turret potentiometer. To set the display type right-click on the tile and select “Show as...”. You can choose display types from the list in the popup menu.



Displaying data in Test mode

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. The code to display these values is automatically generated by RobotBuilder or manually added to your program and is described in the next article. 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.

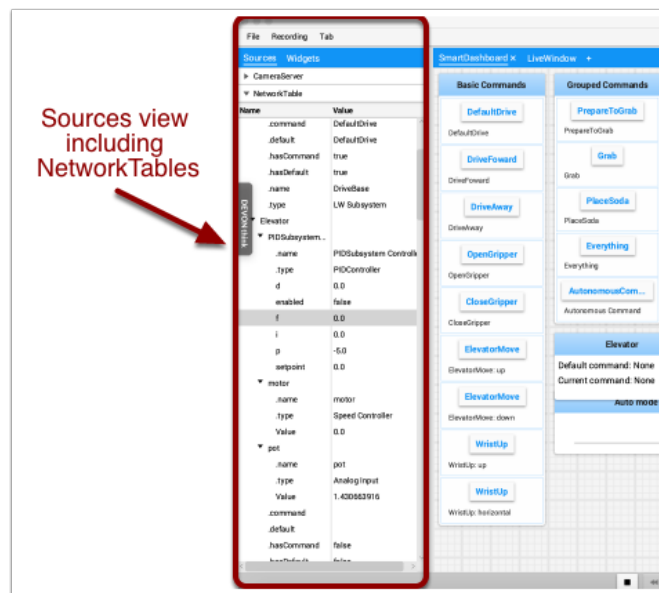
Setting test mode



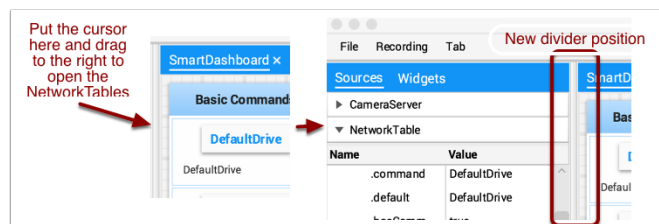
Enable Test Mode in the Driver Station by clicking on the “Test” button and setting “Enable” on the robot. When doing this, Shuffleboard will display the status of any actuators and sensors used by your program organized by subsystem.

Getting data from the Sources view

Normally *NetworkTables* data automatically appears on one of the tabs and you just rearrange and use that data. Sometimes you might want to recover a value that was accidentally deleted from the tab or display a value that is not part of the SmartDashboard / NetworkTables key. For these cases the values can be dragged onto a pane from NetworkTables view under Sources on the left side of the window. Choose the value that you want to display and just drag it to the pane and it will be automatically created with the default type of widget for the data type.



Not: Sometimes the Sources view is not visible on the left - it is possible to drag the divider between the tabbed panes and the Sources so the sources is not visible. If this happens move the cursor over the left edge and look for a divider resizing cursor, then left click and drag out the view. In the two images below you can see where to click and drag, and when finished the divider is as shown in the second image.

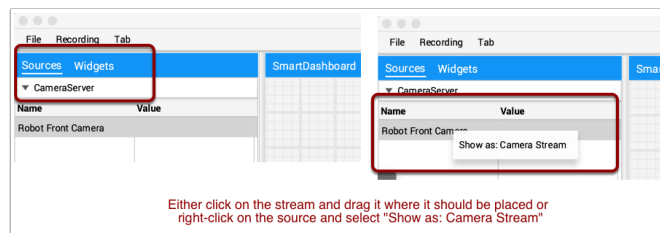


Displaying Camera Streams

Camera streams from the robot can be viewed on a tab in Shuffleboard. This is useful for viewing what the robot is seeing to give a less obstructed view for operators or helping visualize the output from a vision algorithm running on the driver station computer or a coprocessor on the robot. Any stream that is running using the CameraServer API can be viewed in a camera stream widget.

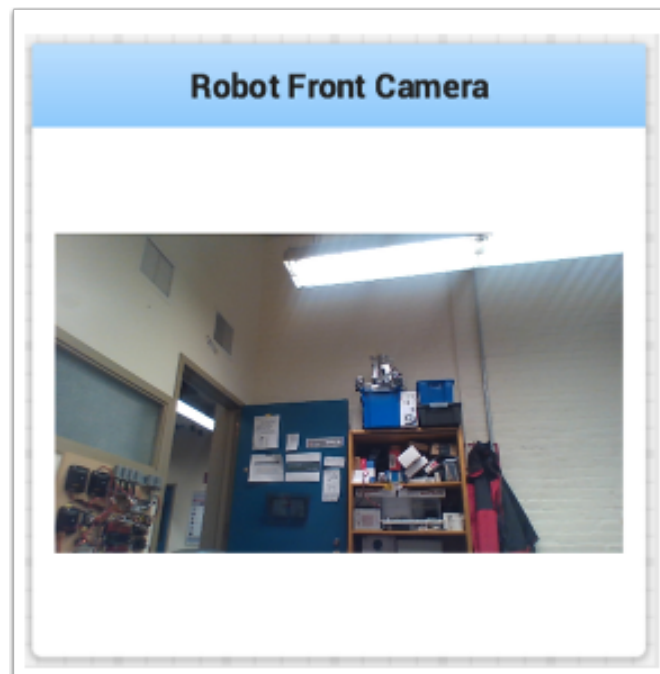
Adding a Camera Stream

To add a camera to your dashboard select “Sources” and view the “CameraServer” source in the left side panel in the Shuffleboard window as shown in the example below. A list of camera streams will be shown, in this case there is only one camera called “Robot Front Camera”. Drag that to the tab where it should be displayed. Alternatively the stream can also be placed on the dashboard by right-clicking on the stream in the Sources list and selecting “Show as: Camera Stream”.



Once the camera stream is added it will be displayed in the window. It can be resized and moved where you would like it.

Not: Be aware that sending too much data from too high a resolution or too high a frame rate will cause high CPU usage on both the roboRIO and the laptop.



Working with widgets

The visual displays that you manipulate on the screen in Shuffleboard are called widgets. Widgets are generally automatically displayed from values that the robot program publishes with NetworkTables.

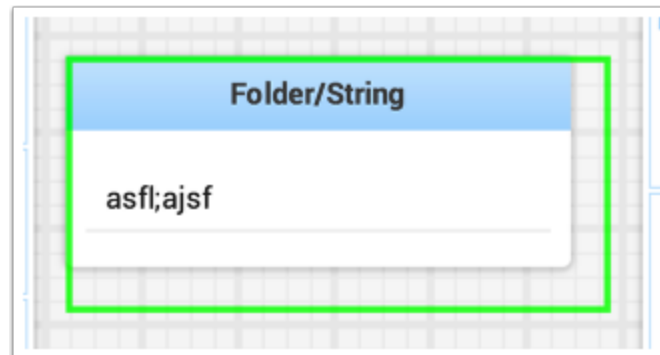
Moving widgets

Widgets can be moved simply with drag and drop. Just move the cursor over the widget, left-click and drag it to the new position. When dragging you can only place widgets on grid squares and the size of the grid will effect the resolution of your display. When dragging a red or green outline will be displayed. Green generally means that there is enough room at the current location to drop the widget and red generally means that it will overlap or be too big to drop. In the example below a widget is being moved to a location where it doesn't fit.



Resizing widgets

Widgets can be resized by clicking and dragging the edge or corner of the widget image. The cursor will change to a resize-cursor when it is in the right position to resize the widget. As with moving widgets, a green or red outline will be drawn indicating that the widget can be resized or not. The example below shows a widget being resized to a larger area with the green outline indicating that there is no overlap with surrounding widgets.



Changing the display type of widgets

Shuffleboard is very rich in display types depending on the data published from the robot. It will automatically choose a default display type, but you might want to change it depending on the application. To see what the possible displays are for any widget, right-click on the widget and select the “Show as...” and from the popup menu, choose the desired type. In the example below are two data values, one a number and the other a boolean. You can see the different types of display options that are available to each. The boolean value has only two possible values (true/false) it can be shown as a boolean box (the red/green color), or text, or a toggle button or toggle switch. The number value can be displayed as a graph, number bar, number slider, dial, text, or a voltage view depending on the context of the value.

Changing the title of widgets

You can change the title of widgets by double-clicking in their title bar and editing the title to the new value. If a widget is contained in a layout, then right-click on the widget and select the properties. From there you can change the widget title that is displayed.

Changing widget properties

You can change the appearance of a widget such as the range of values represented, colors or some other visual element. In cases where this is possible right-click on the widget and select “Edit properties” from the popup menu. In this boolean value widget shown below, the widget title, true color and false color can all be edited.

Working with Lists

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:

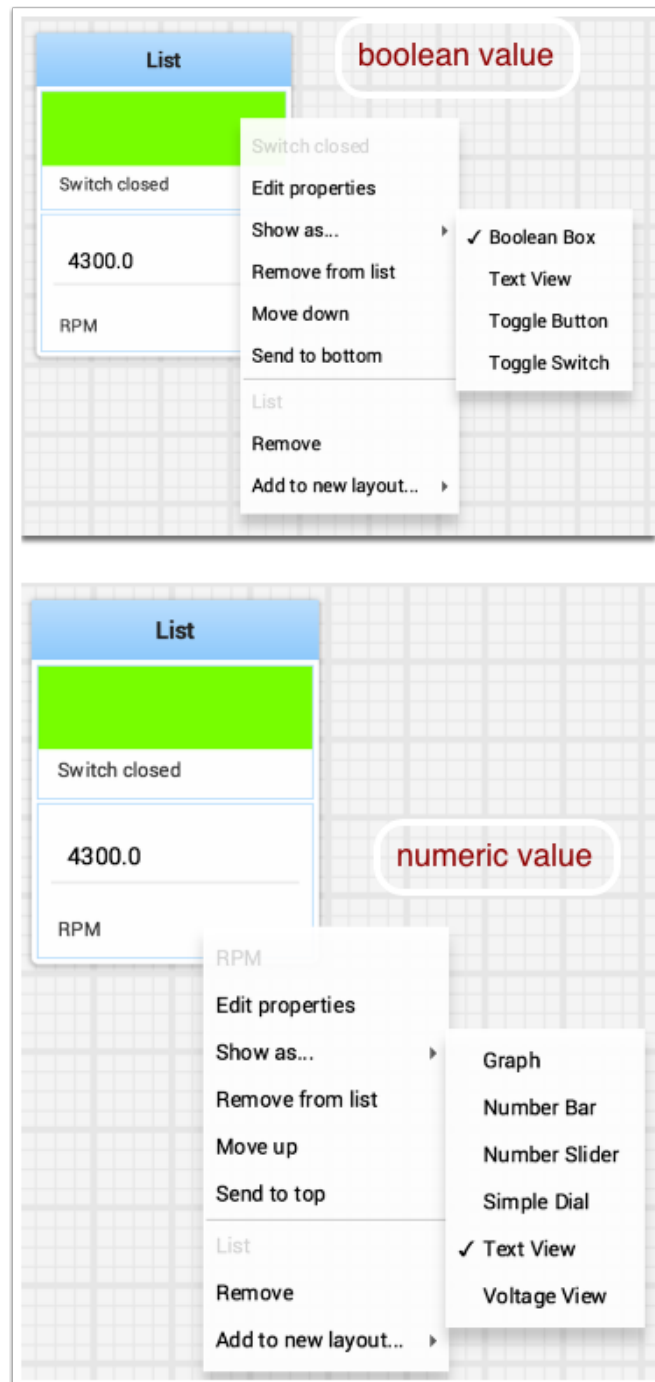
- Tiles in lists don’t have individual header labels; they instead have smaller labels within their list entries.
- Individual tiles placed together create gaps between one another; lists have smaller gaps between tiles.

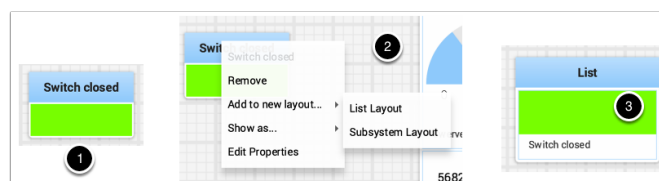
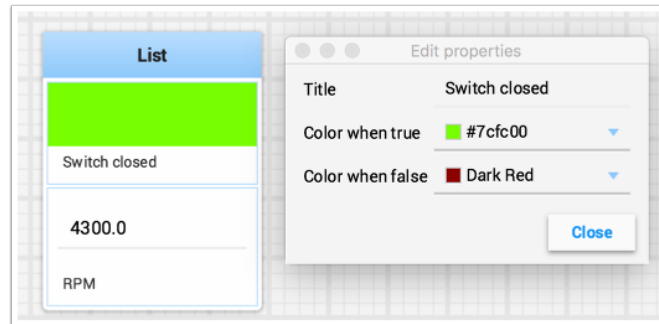
Creating a list

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.





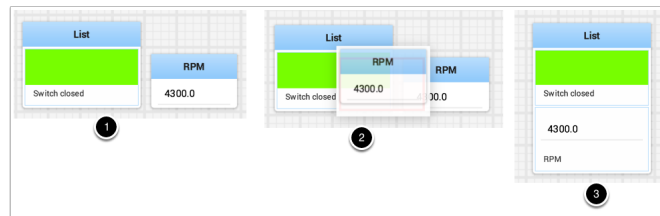
Adding tiles to/removing tiles from a list

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:

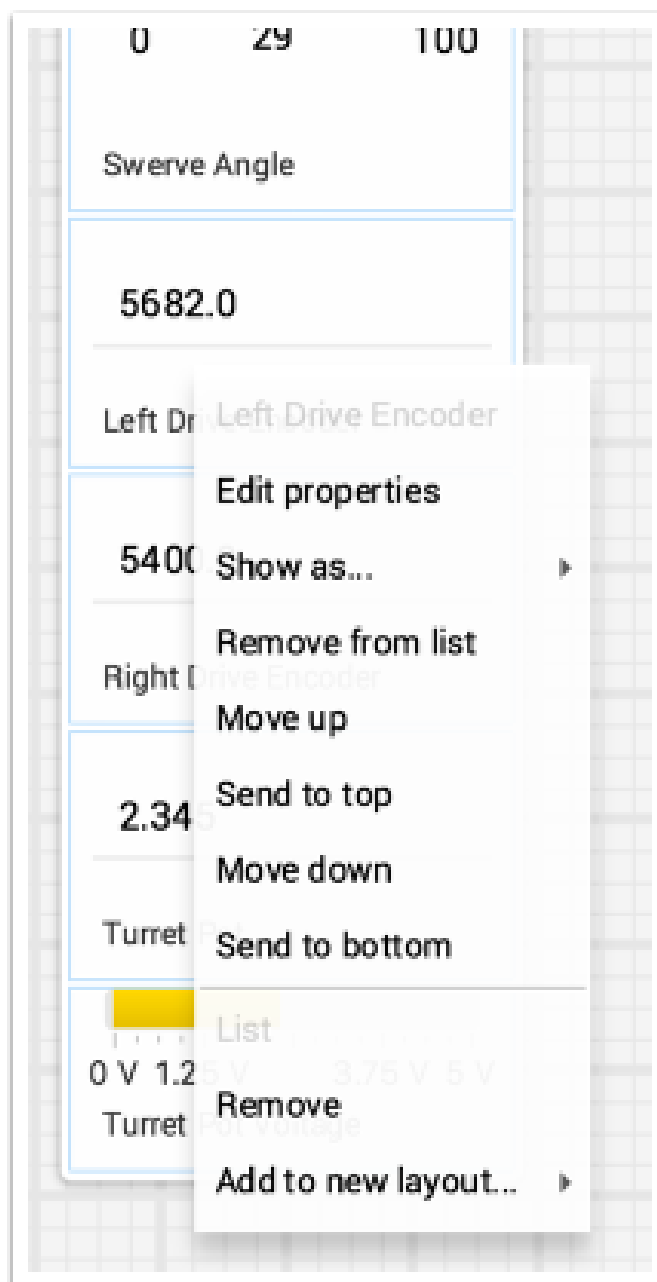
1. Identify the list and the tile within it to be removed.
2. Drag the tile out of the list and place it anywhere with free space.
3. The tile will be removed from the list and placed at that location.



Rearranging tiles in a list

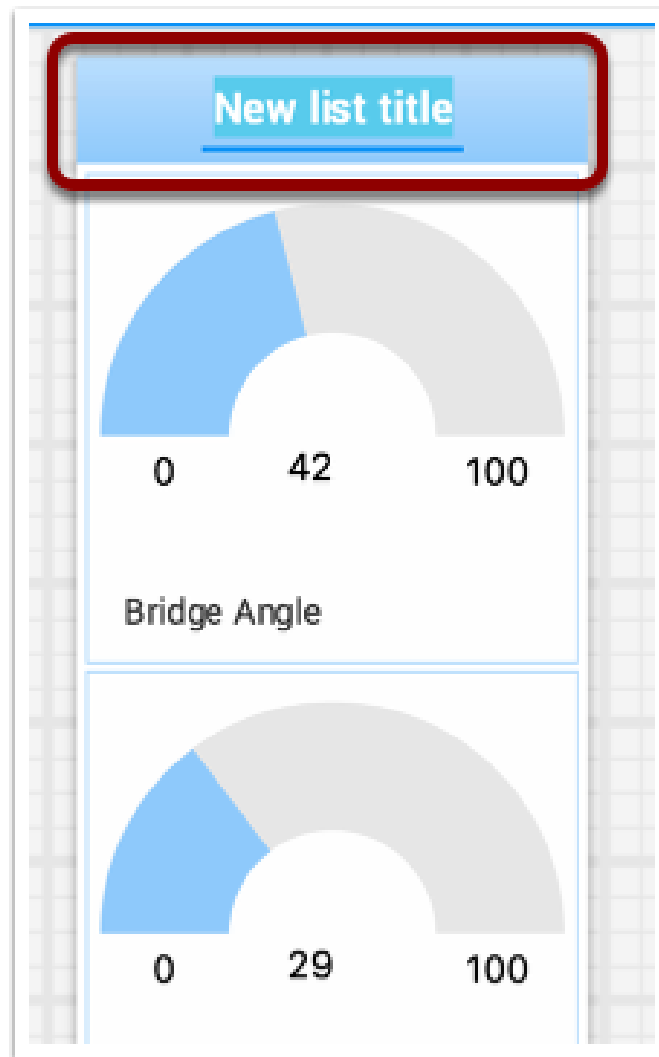
Tiles in a list can be rearranged by right-clicking on the tile and selecting:

- *Move up* moves the tile **towards** the *top* of the list.
- *Move down* moves the tile **towards** the *bottom* of the list.
- *Send to top* moves the tile **to** the *top* of a list.
- *Send to bottom* moves the tile **to** the *bottom* of a list.
- There are two buttons labeled *Remove*, and each button does:
 - The **top** *Remove* button (above the pinline; section of dropdown with grayed-out *tile* label) **deletes** the *tile* from the Shuffleboard layout.
 - The **bottom** *Remove* button (below the pinline; section of dropdown with grayed-out *list* label) **deletes** the *list and all tiles inside it* from the Shuffleboard layout.
 - If you want to take an entry out of a list without deleting it, see [Adding tiles to/removing tiles from a list](#).



Renaming a list

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

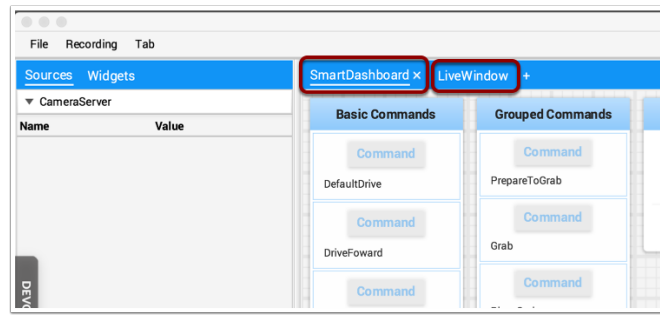


Creating and manipulating tabs

The tabbed layout the Shuffleboard uses help separate different “views” of your robot data and make the displays more useful. You might have a tab the has the display for helping debug the robot program and a different tab for use in competitions. There are a number of options that make tabs very powerful. You can control which data from NetworkTables or other sources appears in each of your tabs using the auto-populate options described later in this article.

Default tabs

When you open Shuffleboard for the first time there are two tabs, labeled SmartDashboard and LiveWindow. These correspond to the two views that SmartDashboard had depending on whether your robot is running in Autonomous/Teleop or Test mode. In shuffleboard both of these views are available any time.



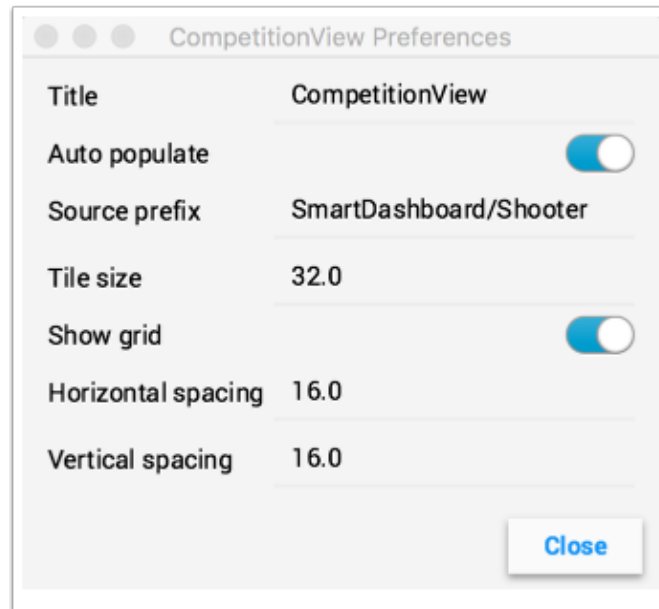
On the SmartDashboard tab all the values that are written using the SmartDashboard.putType() set of methods. On the LiveWindow tab all the autogenerated debugging values are shown.

Switching between tabs

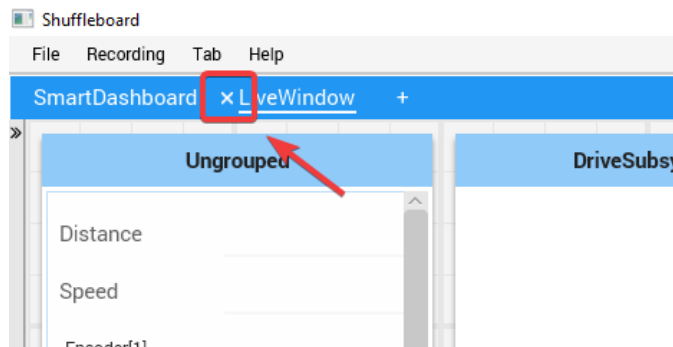
You can switch between tabs clicking on the tab label at the top of the window. In the case above, simply click on SmartDashboard or LiveWindow to see the values that are associated with each tab.

Adding and Hiding Tabs

You can add additional tabs by clicking on the plus(+) symbol just to the right of the last tab. Once you create a new tab you can set the label by double-clicking on the label in the tab and editing it. You can also right-click on the tab or use the Tab menu to bring up the tab preferences and from that window you can change the name by editing the Title field.



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



Setting the tab to auto-populate

One of the most powerful features with tabs is to have them automatically populate new values based on a source prefix that is supplied in the tab Preferences pane. In the above example the Preferences pane has a Source prefix of “SmartDashboard/Shooter” and Auto populate is turned on. Any values that are written using the SmartDashboard class that specifies a sub-key of Shooter will automatically appear on that tab. Note: keys that match more than one Source prefix will appear in both tabs. Because those keys also start with SmartDashboard/ and that’s the Source prefix for the default SmartDashboard tab, those widgets will appear in both panes. To only have values appear in one pane, you can use NetworkTables to write labels and values and use a different path that is not under SmartDashboard. Alternatively you could let everything appear in the SmartDashboard tab making it very cluttered, but have specific tabs for your needs that will be better filtered.

Using the tab grid and spacing

Each tab can have its own Tile size (number of pixels per large square). So some tabs might have coarser resolution for easier layout and others might have a fine grid. The Tile size in the Tab preferences overrides any global settings in the Shuffleboard preferences. In addition, you can specify the padding between the drawing in the widget and the edge of the widget. If you program user interfaces these parameters are usually referred to as horizontal and vertical gap (hgap, vgap).

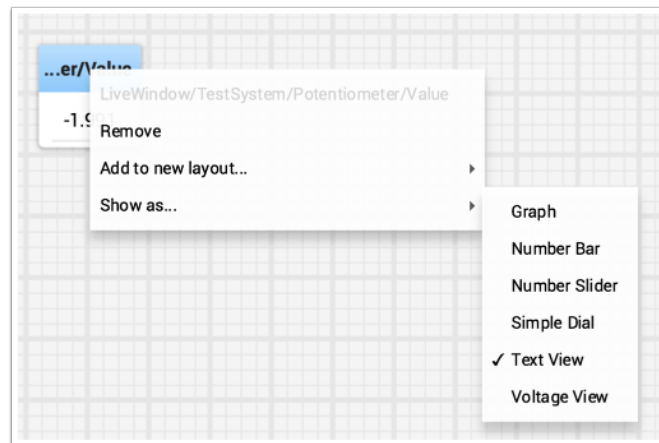
Moving widgets between tabs

Currently there is no way to easily move widgets between tabs without deleting it from one tab and dragging the field from the sources hierarchy on the left into the new pane. We hope to have that capability in a subsequent update soon.

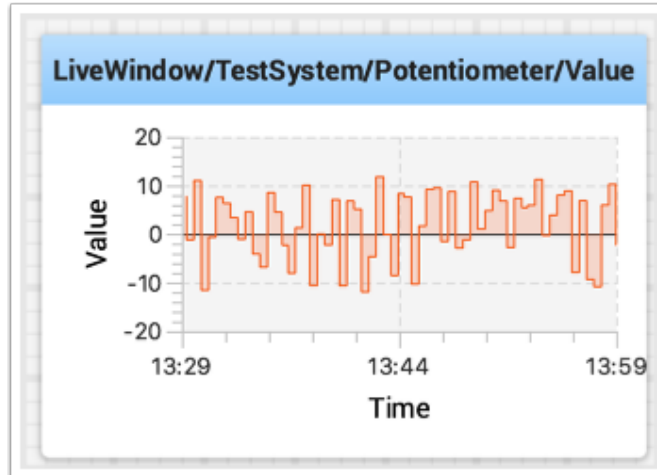
Working with Graphs

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.

To create a graph, choose a numeric value and right-click in the heading and select “Show as...” and then choose graph

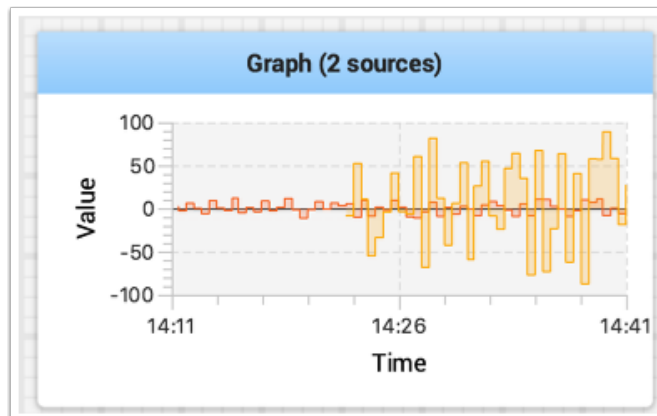


The graph widget shows line plots of the value that you selected. It will automatically set the scale and the default time interval that the graph will show will be 30 seconds. You can change that in the setting for the graph (see below).

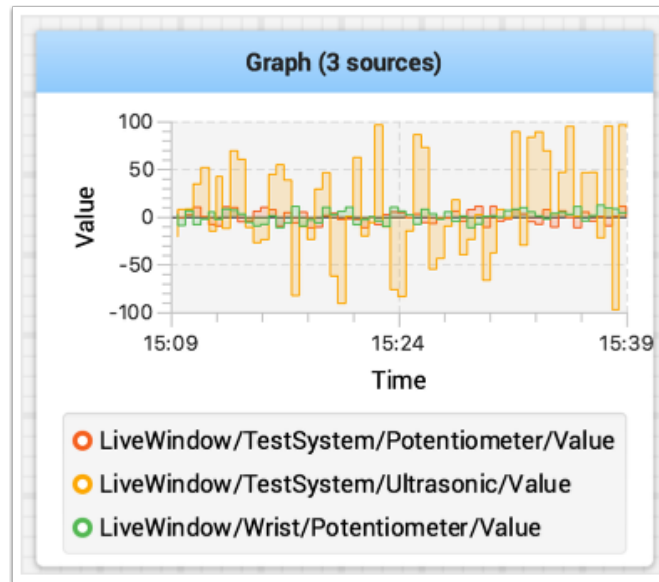


Adding Additional Data Values

For related values it is often desirable to show multiple values on the same graph. To do that, simply drag additional values from the NetworkTables source view (left side of the Shuffleboard window) and drop it onto the graph and that value will be added as shown below. You can continue to drag additional values onto the graph.



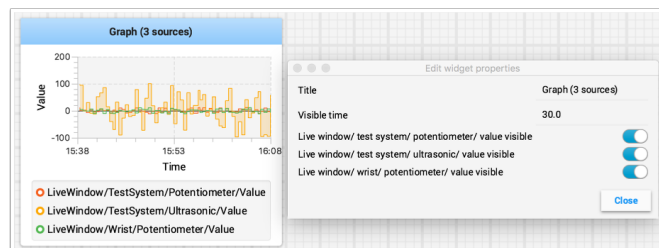
You can resize the graph vertically to view the legend if it is not displayed as shown in the image below. The legend shows all the sources that are used in the plot.



Setting Graph Properties

You can set the number of seconds that are shown in the graph by changing the “Visible time” in the graph widget properties. To access the properties, right-click on the graph and select “Edit properties”.

In addition to setting the visible time the graph can selectively turn sources on and off by turning the switch on and off for each of the sources shown in the properties window (see below).

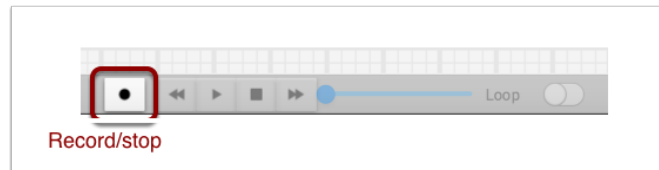


Recording and Playback

Shuffleboard can log all widget updates during a session. Later the log file can be “played back” to see what happened during a match or a practice run. This is especially useful if something doesn’t operate as intended during a match and you want to see what happened. Each recording is captured in a recording file.

Creating a Recording

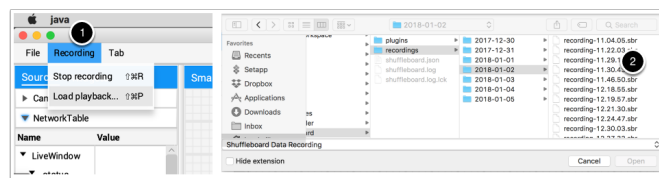
When shuffleboard starts it begins recording to all the NetworkTables values are recorded and continues until stopped by hitting the record/stop button in the recorder controls as shown below. If a new recording is desired, such as when a new piece of code or mechanical system is being tested, stop the current recording if it is running, and click the record button. Click the button again to stop recording and close the recording file. If the button is round (as shown) then click it to start a recording. If the button is a square, then a recording is currently running so click it to stop the recording.



Playing a Recording

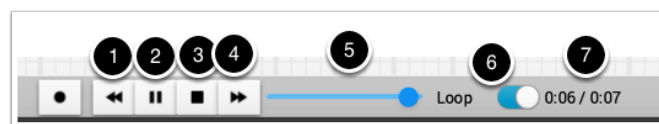
Previous recordings can be played back by:

1. Selecting the “Recording” menu then click “Load playback”.
2. Choose a recording from the from the directory shown. Recordings are grouped by date and the file names are the time the recording was made to help identify the correct one. Select the correct recording from the list.



Controlling the Playback

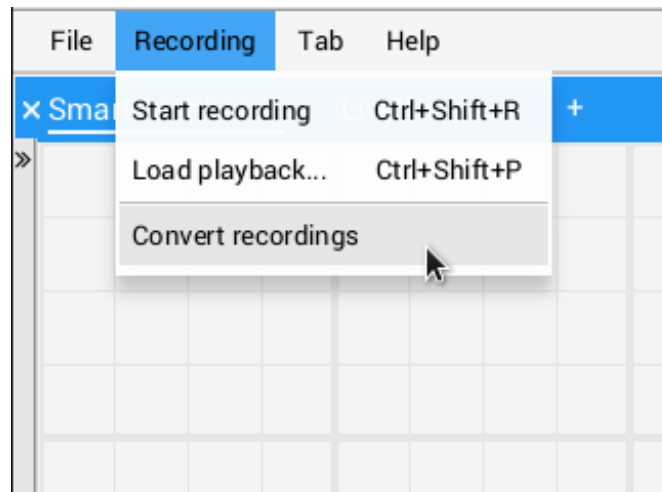
Selecting the recording file will begin playback of that file. While the recording is playing the recording controls will show the current time within the recording as well as the option to loop the recording while watching it. When the recording is being played back the “transport” controls will allow the playback to be controlled.



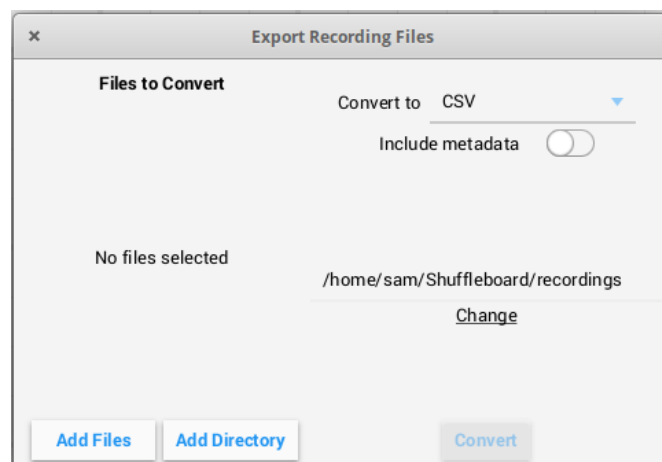
The controls work as follows:

1. The left double-arrow button backs up the playback to the last changed data point
2. The play/pause controls starts and stops the playback
3. The square stop button stops playback and resumes showing current robot values
4. The right double-arrow skips forward to the next changed data value
5. The slider allows for direct positioning to any point in time to view different parts of the recording
6. The loop switch turns on playback looping, that is, the playback will run over and over until stopped
7. The time shows the current point within the recording and the total time of the recording

Converting to Different File Formats



Shuffleboard recordings are in a custom binary format for efficiency. To analyze recorded data without playing it back through the app, Shuffleboard supports data converters to convert the recordings to an arbitrary format. Only a simple CSV converter is shipped with the app, but teams can write custom converters and include them in Shuffleboard plugins.



Multiple recordings can be converted at once. Individual files can be selected with the “Add

Files” button, or all recording files in a directory can be selected at once with the “Add Directory” button.

Converted recordings will be generated in the ~/Shuffleboard/recordings directory, but can be manually selected with the “Change” button.

Different converters can be selected with the dropdown in the top right. By default, only the CSV converter is available. Custom converters from plugins will appear as options in the dropdown.

Additional Notes

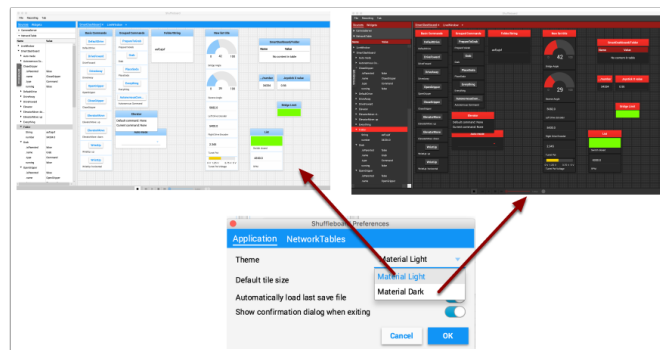
Graphs won’t display properly while scrubbing the timeline but if it is playing through where the graph history can be captured by the graph then they will display as in the original run.

Setting global preferences for Shuffleboard

There are a number of settings that set the way Shuffleboard looks and behaves. Those are on the Shuffleboard Preferences pane that can be accessed from the File menu.

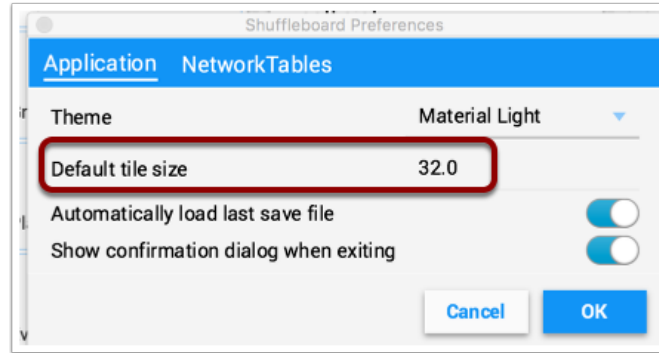
Setting the theme

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.



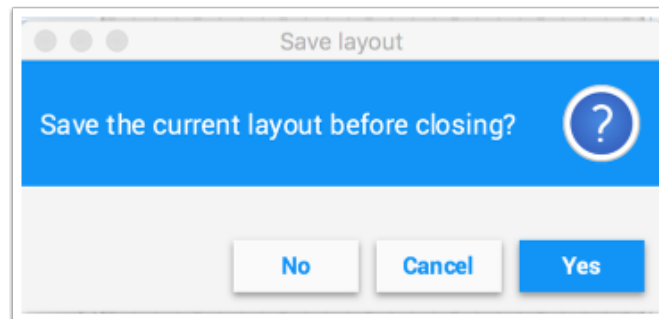
Setting the default tile size

Shuffleboard positions tiles on a grid when you are adding or moving them yourself or when they are auto-populated. You can set the default tile size when for each tab or it can be set globally for all the tabs created after the default setting is changed. Finer resolution in the grid results in finer control over placement of tiles. This can be set in the Shuffleboard Preferences window as shown below.

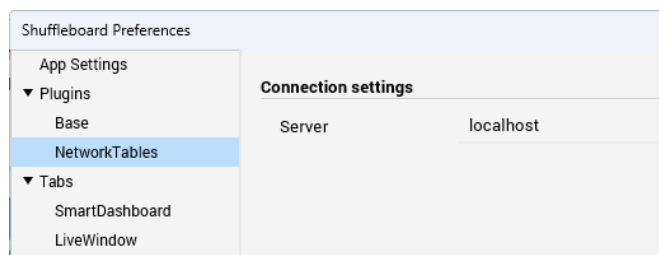


Working with the layout save files

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 lose your changes.



Setting the team number



In order for Shuffleboard to be able to find your NetworkTables server on your robot, specify your team number in the “NetworkTables” tab on the Preferences pane. If you’re running Shuffleboard with a running Driver Station, the Server field will be auto-populated with the correct information. If you’re running on a computer without the Driver Station, you can manually enter your team number or the robotRIO network address.

Shuffleboard FAQ, issues, and bugs

Uyari: Shuffleboard as well as most of the other control system components were developed with Java 11 and will not work with Java 8. Be sure before reporting problems that your computer has Java 11 installed and is set as the default Java Environment.

Frequently Asked Questions

How do I report issues, bugs or feature requests with Shuffleboard?

There is no active maintainer of Shuffleboard, but we are accepting pull requests. Bugs, issues, and feature requests can be added on the Shuffleboard GitHub page by creating an issue. Please try to look at existing issues before creating new ones to make sure you aren't duplicating something that has already been reported or work that is planned. You can find the issues on the [Shuffleboard GitHub page](#).

How can I add my own widgets or other extensions to 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](#).

How can I build Shuffleboard from the source code?

You can get the source code by downloading, cloning, or forking the repository on the GitHub site. To build and run Shuffleboard from the source, make sure that the current directory is the top level source code and use one of these commands:

Application	Command (for Windows systems run the gradlew.bat file)
Running Shuffleboard	<code>./gradlew :app:run</code>
Building the APIs and utility classes for plugin creation	<code>./gradlew :api:shadowJar</code>
Building the complete application jar file	<code>./gradlew :app:shadowJar</code>

11.2.2 Shuffleboard - Layouts with Code

Using tabs

Shuffleboard is a tabbed interface. Each tab organizes widgets in a logical grouping. By default, Shuffleboard has tabs for the legacy SmartDashboard and LiveWindow - but new tabs can now be created in Shuffleboard directly from a robot program for better organization.

Creating a new tab

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")
```

Creating a new tab is as simple as calling a single method on the Shuffleboard class, which will create a new tab on Shuffleboard and return a handle for adding your data to the tab. Calling `getTab` multiple times with the same tab title will return the same handle each time.

Selecting a tab

JAVA

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

C++

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

PYTHON

```
from wpilib.shuffleboard import Shuffleboard  
  
Shuffleboard.selectTab("Tab Title")
```

This method lets a tab be selected by title. This is case-sensitive (so “Tab Title” and “Tab title” are two individual tabs), and only works if a tab with that title exists at the time the method is called, so calling `selectTab("Example")` will only have an effect if a tab named “Example” has previously been defined.

This method can be used to select any tab in Shuffleboard, not just ones created by the robot program.

Caveats

Tabs created from a robot program differ in a few important ways from normal tabs created from the dashboard:

- Not saved in the Shuffleboard save file
- No support for autopopulation
- Users are expected to specify the tab contents in their robot program
- Have a special color to differentiate from normal tabs

Sending data

Unlike SmartDashboard, data cannot be sent directly to Shuffleboard without first specifying what tab the data should be placed in.

Sending simple data

Sending simple data (numbers, strings, booleans, and arrays of these) is done by calling `add` on a tab. This method will set the value if not already present, but will not overwrite an existing value.

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)
```

If data needs to be updated (for example, the output of some calculation done on the robot), call `getEntry()` after defining the value, then update it when needed or in a periodic function

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)
```

Making choices persist between reboots

When configuring a robot from the dashboard, some settings may want to persist between robot or driverstation reboots instead of having drivers remember (or forget) to configure the settings before each match.

Simply using *addPersistent* instead of *add* will make the value saved on the roboRIO and loaded when the robot program starts.

Not: This does not apply to sendable data such as choosers or motor controllers.

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))
```

Sending sensors, motors, etc

Analogous to `SmartDashboard.putData`, any `Sendable` object (most sensors, motor controllers, and `SendableChoosers`) can be added to any tab

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))
```

Retrieving data

Unlike `SmartDashboard.getNumber` and friends, retrieving data from `Shuffleboard` is also done through the `NetworkTableEntries`, which we covered in the previous article.

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)
```

This basic example has a glaring flaw: the maximum speed can be set on the dashboard to a value outside [0, 1] - which could cause the inputs to be saturated (always at maximum speed), or even reversed! Fortunately, there is a way to avoid this problem - covered in the next article.

Configuring widgets

Robot programs can specify exactly which widget to use to display a data point, as well as how that widget should be configured. As there are too many widgets to be listed here, consult the docs for details.

Specifying a widget

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

JAVA

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

C++

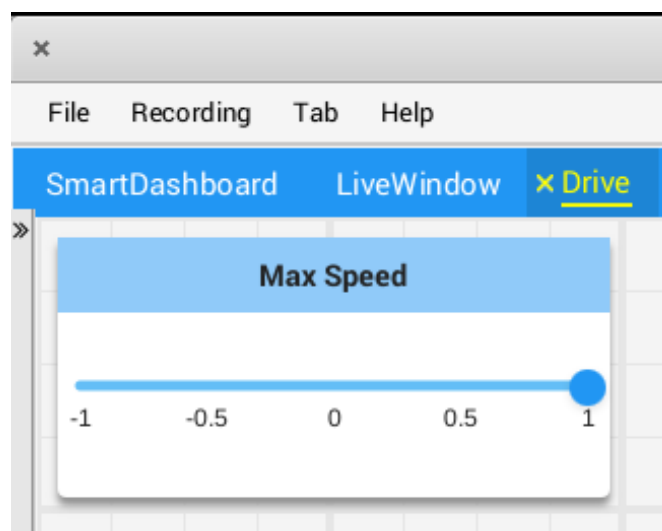
```
frc::Shuffleboard::GetTab("Drive")
    .Add("Max Speed", 1)
    .WithWidget(frc::BuiltInWidgets::kNumberSlider) // specify the widget here
    .GetEntry();
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard
from wpilib.shuffleboard import BuiltInWidgets

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

In this example, we configure the “Max Speed” widget to use a slider to modify the values instead of a basic text field.



Setting widget properties

Since the maximum speed only makes sense to be a value from 0 to 1 (full stop to full speed), a slider from -1 to 1 can cause problems if the value drops below zero. Fortunately, we can modify that using the `withProperties` method

JAVA

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

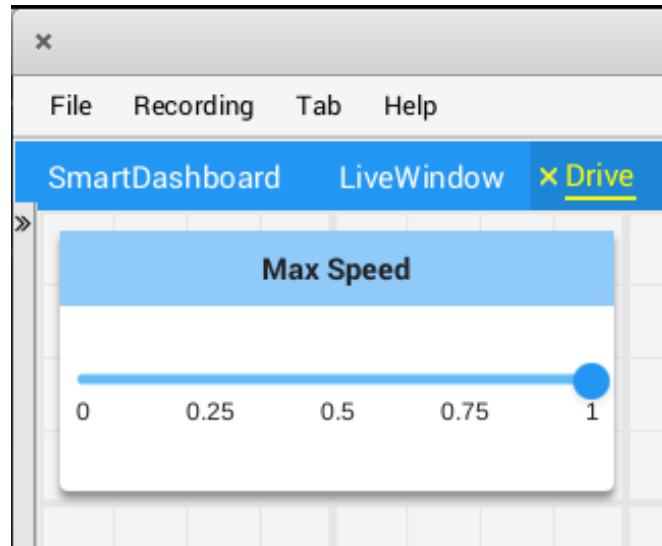
C++

```
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())
```



Notes

Widgets can be specified by name; however, names are case- and whitespace-sensitive (“Number Slider” is different from “Number slider” and “NumberSlider”). For this reason, it is recommended to use the built in widgets class to specify the widget instead of by raw name. However, a custom widget can only be specified by name or by creating a custom `WidgetType` for that widget.

Widget property names are neither case-sensitive nor whitespace-sensitive (“Max” and “max” are the same). Consult the documentation on the widget in the `BuiltInWidgets` class for details on the properties of that widget.

Organizing Widgets

Setting Widget Size and Position

Call `withSize` and `withPosition` to set the size and position of the widget in the tab.

`withSize` sets the number of columns wide and rows high the widget should be. For example, calling `withSize(1, 1)` makes the widget occupy a single cell in the grid. Note that some widgets have a minimum size that may be greater than the specified size, in which case the widget will use the smallest supported size.

`withPosition` sets the row and column of the top-left corner of the widget. Rows and columns are both 0-indexed, so the topmost row is number 0 and the leftmost column is also number 0. If the position of any widget in a tab is specified, every widget should also have its position set to avoid overlapping widgets.

JAVA

```
Shuffleboard.getTab("Pre-round")
    .add("Auto Mode", autoModeChooser)
    .withSize(2, 1) // make the widget 2x1
    .withPosition(0, 0); // place it in the top-left corner
```

C++

```
frc::Shuffleboard::GetTab("Pre-round")
    .Add("Auto Mode", autoModeChooser)
    .WithSize(2, 1)
    .WithPosition(0,0);
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard

(Shuffleboard.getTab("Pre-round")
    .add("Auto Mode", autoModeChooser)
    .withSize(2, 1) # make the widget 2x1
    .withPosition(0, 0)) # place it in the top-left corner
```

Adding Widgets to Layouts

If there are many widgets in a tab with related data, it can be useful to place them into smaller subgroups instead of loose in the tab. Much like how the handle to a tab is retrieved with `Shuffleboard.getTab`, a layout inside a tab (or even in another layout) can be retrieved with `ShuffleboardTab.getLayout`.

JAVA

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

elevatorCommands.add(new ElevatorDownCommand());
elevatorCommands.add(new ElevatorUpCommand());
```

C++

```
wpi::StringMap<std::shared_ptr<nt::Value>> properties{
    std::make_pair("Label position", nt::Value::MakeString("HIDDEN"))
};

frc::ShuffleboardLayout& elevatorCommands = frc::Shuffleboard::GetTab("Commands")
    .GetLayout("Elevator", frc::BuiltInLayouts::kList)
    .WithSize(2, 2)
    .WithProperties(properties);

ElevatorDownCommand* elevatorDown = new ElevatorDownCommand();
ElevatorUpCommand* elevatorUp = new ElevatorUpCommand();

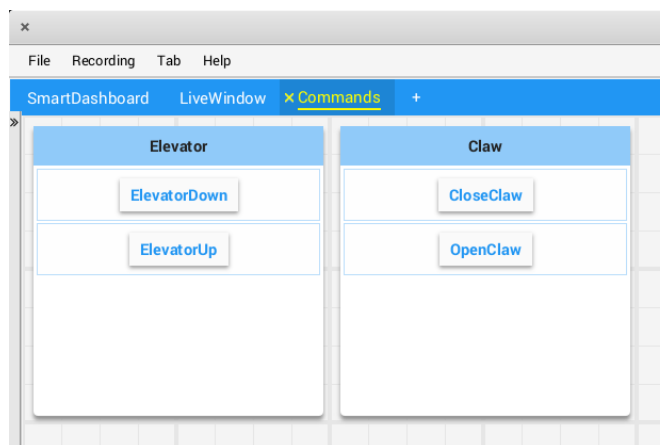
elevatorCommands.Add("Elevator Down", elevatorDown);
elevatorCommands.Add("Elevator Up", elevatorUp);
```

PYTHON

```
from wpilib.shuffleboard import Shuffleboard
from wpilib.shuffleboard import BuiltInLayouts

(elevatorCommands = Shuffleboard.getTab("Commands")
    .getLayout("Elevator", BuiltInLayouts.kList)
    .withSize(2, 2)
    .withProperties(map("Label position", "HIDDEN"))) # hide labels for commands

elevatorCommands.add(ElevatorDownCommand())
elevatorCommands.add(ElevatorUpCommand())
```



11.2.3 Shuffleboard - Advanced Usage

Commands and Subsystems

When using the command-based framework Shuffleboard makes it easier to understand what the robot is doing by displaying the state of various commands and subsystems in real-time.

Displaying Subsystems

To see the status of a subsystem while the robot is operating in either autonomous or teleoperated modes, that is what its default command is and what command is currently using that subsystem, send a subsystem instance to Shuffleboard:

JAVA

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

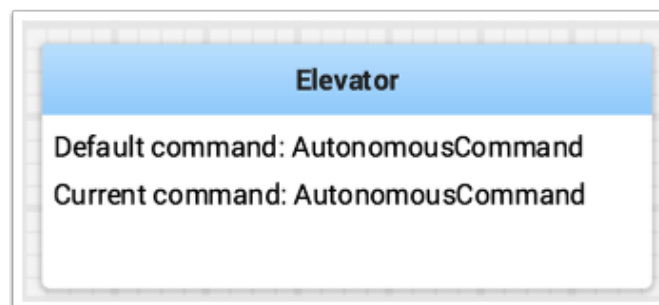
C++

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

PYTHON

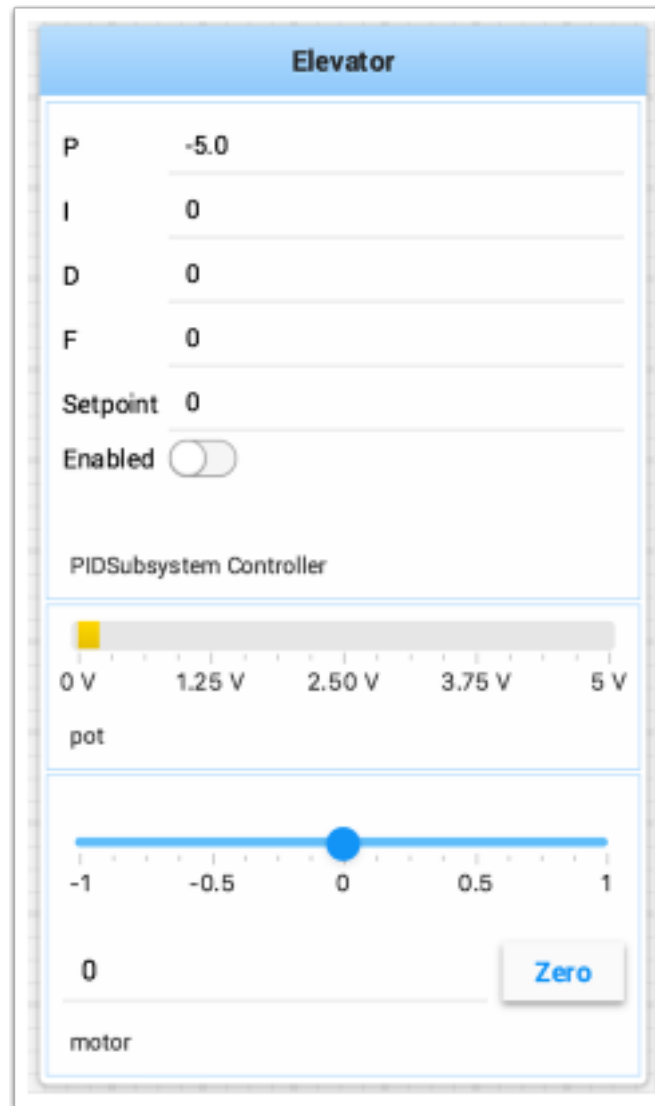
```
from wpilib import SmartDashboard  
SmartDashboard.putData(subsystem-reference)
```

Shuffleboard will display the subsystem name, the default command associated with this subsystem, and the currently running command. In this example the default command for the Elevator subsystem is called `AutonomousCommand` and it is also the current command that is using the Elevator subsystem.



Subsystems in Test Mode

In Test mode (Test/Enabled in the driver station) subsystems may be displayed in the Live-Window tab with the sensors and actuators of the subsystem. This is ideal for verifying of sensors are working by seeing the values that they are returning. In addition, actuators can be operated. For example, motors can be operated using sliders to set their commanded speed and direction. For PIDSubsystems the P, I, D, and F constants are displayed along with the setpoint and an enable control. This is useful for tuning PIDSubsystems by adjusting the constants, putting in a setpoint, and enabling the embedded PIDController. Then the mechanism's response can be observed. This cycle (change parameters, enable, and observe) can be repeated until a reasonable set of parameters is found.



PIDS alt sistemlerinin ayarlanması hakkında daha fazla bilgi bulunabilir [here](#). RobotBuilder'ı kullanmak, alt sistemi Test modunda görüntülemek için kodu otomatik olarak oluşturacaktır. Alt sistemlerin görüntülenmesi için gerekli olan kod aşağıda gösterilmiştir; burada alt sistem adı, alt sistemin adını içeren bir dizedir:

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

Komutları Görüntüleme

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));
```

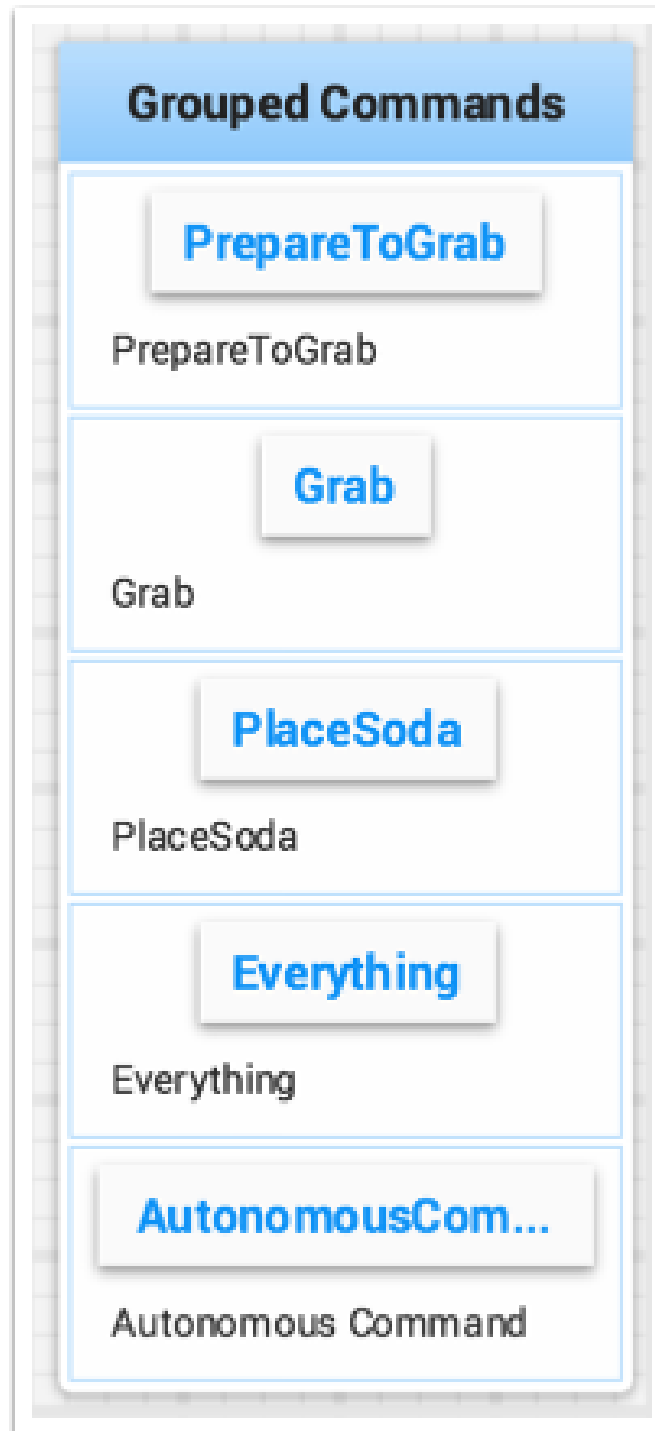
C++

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

PYTHON

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

Shuffleboard will display the command name and a button to execute the command. In this way individual commands and command groups can easily be tested without needing special test code in a robot program. In the image below there are a number of commands contained in a Shuffleboard list. Pressing the button once runs the command and pressing it again stops the command. To use this feature the robot must be enabled in teleop mode.



Testing and Tuning PID Loops

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:

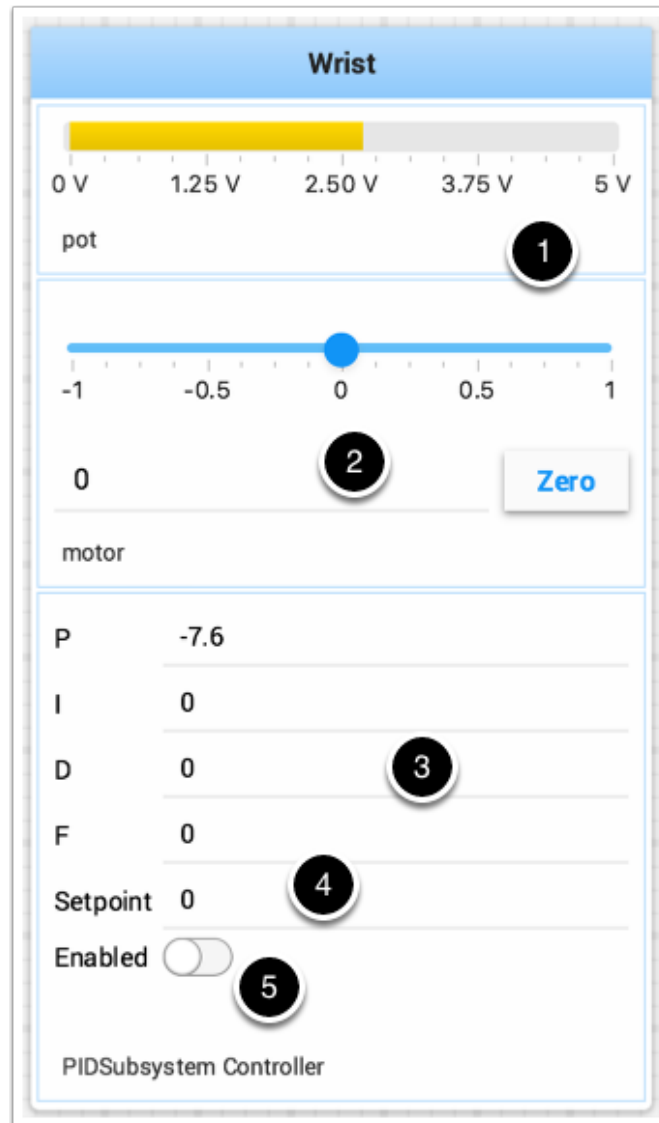
1. Reading sensor values to determine how far the robot or mechanism from the desired setpoint. The setpoint is the sensor value that corresponds to the expected goal. For example, a robot arm with a wrist joint should be able to move to a specified angle very quickly and stop at that angle as indicated by a sensor. A potentiometer is a sensor that can measure rotational angle. By connecting it to an analog input, the program can get a voltage measurement that is directly proportional to the angle.
2. Compute an error (the difference between the sensor value and the desired value). The sign of the error value indicates which side of the setpoint the wrist is on. For example negative values might indicate that the measured wrist angle is larger than the desired wrist angle. The magnitude of the error is how far the measured wrist angle is from the actual wrist angle. If the error is zero, then the measured angle exactly matches the desired angle. The error can be used as an input to the PID algorithm to compute a motor speed.
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 has a `PIDController` class that implements the PID algorithm and accepts constants that correspond to the K_p , K_i , and K_d values. The PID algorithm has three components that contribute to computing the motor speed from the error.

1. P (proportional) - this is a term that when multiplied by a constant (K_p) will generate a motor speed that will help move the motor in the correct direction and speed.
2. I (integral) - this term is the sum of successive errors. The longer the error exists the larger the integral contribution will be. It is simply a sum of all the errors over time. If the wrist isn't quite getting to the setpoint because of a large load it is trying to move, the integral term will continue to increase (sum of the errors) until it contributes enough to the motor speed to get it to move to the setpoint. The sum of the errors is multiplied by a constant (K_i) to scale the integral term for the system.
3. D (differential) - this value is the rate of change of the errors. It is used to slow down the motor speed if it's moving too fast. It's computed by taking the difference between the current error value and the previous error value. It is also multiplied by a constant (K_d) to scale it to match the rest of the system.

Tuning the PID Controller

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).



This is the test mode picture of a wrist subsystem that has a potentiometer as the sensor (pot) and a motor controller connected to the motor. It has a number of areas that correspond to the PIDSubsystem.

1. The analog input voltage value from the potentiometer. This is the sensor input value.
2. A slider that moves the wrist motor in either direction with 0 as stopped. The positive and negative values correspond to moving up or down.
3. The PID constants as described above (F is a feedforward value that is used for speed PID loops)
4. The setpoint value that corresponds the to the pot value when the wrist has reached the desired value
5. Enables the PID controller - No longer working, see below.

Try various PID gains to get the desired motor performance. You can look at the video linked to at the beginning of this article or other sources on the internet to get the desired performance.

Önemli: 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.

Enable Functionality in the New `PIDController`

The following example demonstrates how to create a button on your dashboard that will enable/disable the `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()))

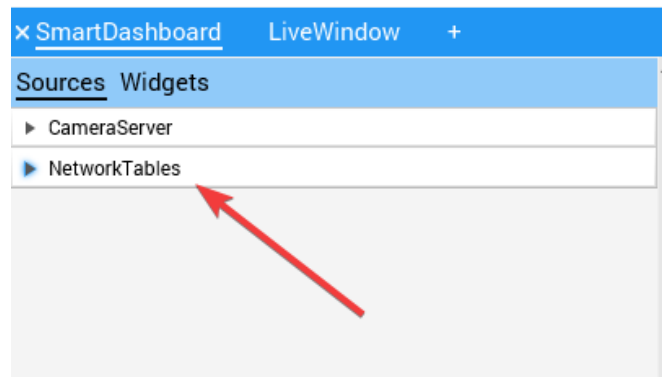
# Timed Robot Example
if (shooterEnable.getBoolean()):
    # Calculates the output of the PID algorithm based on the sensor reading
    # and sends it to a motor
    motor.set(pid.calculate(encoder.getDistance(), setpoint))

```

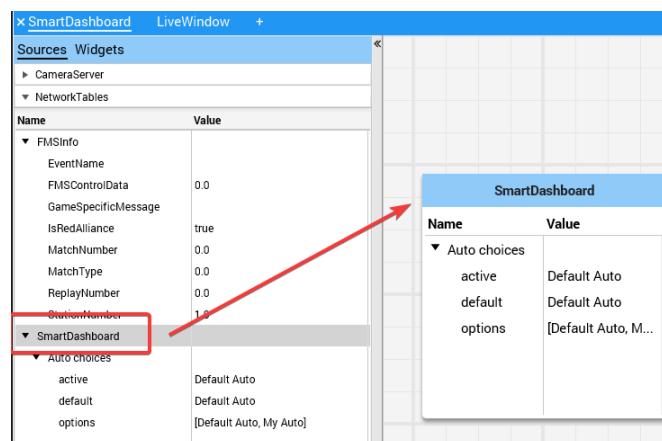
Viewing Hierarchies of Data

Dragging a key with other keys below it (deeper in the hierarchy) displays the hierarchy in a tree, similar to the NetworkTables sources on the left.

Select the data source:



Click and drag the NetworkTables key into the preferred tab.



11.2.4 Shuffleboard - Custom Widgets

Built-in Plugins

Shuffleboard provides a number of built-in plugins that handle common tasks for FRC® use, such as camera streams, all widgets, and *NetworkTables* connections.

Base Plugin

The base plugin defines all the data types, widgets, and layouts necessary for FRC use. It does *not* define any of the source types, or any special data types or widgets for those source types. Those are handled by the *NetworkTables Plugin* and the *CameraServer Plugin*. This separation of concerns makes it easier for teams to create plugins for custom source types or protocols (eg HTTP, ZeroMQ) for the FRC data types without needing a NetworkTables client.

CameraServer Plugin

The camera server plugin provides sources and widgets for viewing camerastreams from the CameraServer WPILib class.

This plugin depends on the *NetworkTables Plugin* in order to discover the available camera streams.

Stream discovery

CameraServer sources are automatically discovered by looking at the /CameraPublisher NetworkTable.

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

For example, a camera named “Camera” with a server at `roborio-0000-frc.local` would have this table layout:

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

This setup will automatically discover all camera streams hosted on a roboRIO by the CameraServer class in WPILib. Any non-WPILib projects that want to have camera streams appear in shuffleboard will have to set the streams entry for the camera server.

NetworkTables Plugin

The NetworkTables plugin provides data sources backed by ntcore. Since the LiveWindow, SmartDashboard, and Shuffleboard classes in WPILib use NetworkTables to send the data to the driver station, this plugin will need to be loaded in order to use those classes.

This plugin handles the connection and reconnection to NetworkTables automatically, users of shuffleboard and writers of custom plugins will not have to worry about the intricacies of the NetworkTables protocol.

Creating a Plugin

Overview

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

- NetworkTables Plugin: To connect to data published over NetworkTables
- Base Plugin: To display custom FRC® data types in custom widgets
- CameraServer Plugin: To view streams from the CameraServer

Tüyo: An example custom Shuffleboard plugin which creates a custom data type and a simple widget for displaying it can be found [here](#).

Create a Custom Plugin

In order to define a plugin, the plugin class must be a subclass of `edu.wpi.first.shuffleboard.api.Plugin` or one of its subclasses. An example of a plugin class would be as following.

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 {
}

```

Additional explanations on how these attributes are used, including version numbers can be found [here](#).

Note the `@Description` annotation is needed to tell the plugin loader the properties of the custom plugin class. Plugin classes are permitted to have a default constructor but it cannot take any arguments.

Building plugin

The easiest way to build plugins is to utilize 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"`

Plugins are allowed to have dependencies on other plugins and libraries, however, they must be included correctly in the maven or gradle build file. When a plugin depends on other plugins, it is good practice to define those dependencies so the plugin does not load when the dependencies do not load as well. This can be done using the `@Requires` annotation as shown below:

```
@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 {
}
```

The `minVersion` specifies the minimum allowable version of the plugin that can be loaded. For example, if the `minVersion` is 1.4.5, and the plugin with the version 1.4.7 is loaded, it will be allowed to do so. However, if the plugin with the version 1.2.4 is loaded, it will not be allowed to since it is less than the `minVersion`.

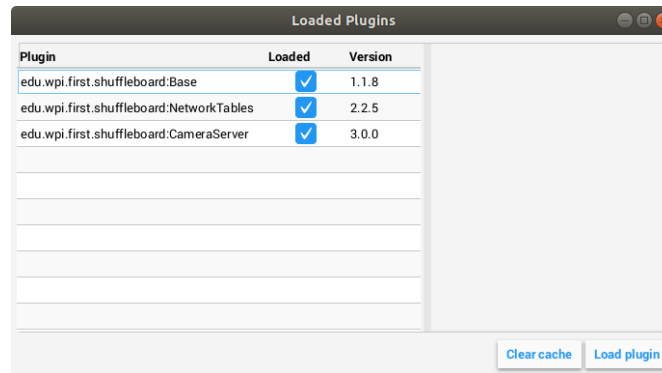
Deploying Plugin To Shuffleboard

In order to load a plugin in Shuffleboard, you will need to generate a jar file of the plugin and put it in the `~/Shuffleboard/plugins` folder. This can be done automatically by running from the shuffleboard root gradlew `:example-plugins:PLUGIN-NAME:installPlugin`

After deploying, Shuffleboard will cache the path of the plugin so it can be automatically loaded the next time Shuffleboard loads. It may be necessary to click on `Clear Cache` under the plugins menu to remove a plugin or reload a plugin into Shuffleboard.

Manually Adding Plugin

The other way to add a plugin to Shuffleboard is to compile it to a jar file and add it from Shuffleboard. The jar file is located in `example-plugins\PLUGIN-NAME\build\libs` after running gradlew `build` in the shuffleboard root. Open Shuffleboard, click on the file tab in the top left, and choose Plugins from the drop down menu.



From the plugins window, choose the “Load plugin” button in the bottom right, and select your jar file.

Creating Custom Data Types

Widgets allow us to control and visualize different types of data. This data could be integers and doubles or even Java Objects. In order to display these types of data using widgets, it is helpful to create a container class for them. It is not necessary to create your own Data Class if the widget will handle single fielded data types such as doubles, arrays, or strings.

Creating The Data Class

In this example, we will create a custom data type for a 2D Point and its x and y coordinates. In order to create a custom data type class, it must extend the abstract class `ComplexData`. Your custom data class must also implement the `asMap()` method that returns the represented data as a simple map as noted below with the `@Override` annotation:

```
import edu.wpi.first.shuffleboard.api.data.ComplexData;

import java.util.Map;

public class MyPoint2D extends ComplexData<MyPoint2D> {

    private final double x;
    private final double y;

    //Constructor should take all the different fields needed and assign them their
    //corresponding instance variables.
    public MyPoint2D(double x, double y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public Map<String, Object> asMap() {
        return Map.of("x", x, "y", y);
    }
}
```

It is also good practice to override the default `equals` and `hashCode` methods to ensure that different objects are considered equivalent when their fields are the same. The `asMap()` method should return the data represented in a simple `Map` object as it will be mapped to the

NetworkTables entry it corresponds to. In this case, we can represent the point as its X and Y coordinates and return a Map containing them.

```
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);
    }
}
```

Other methods can be added to retrieve or edit fields and instance variables, however, it is good practice to make these classes immutable to prevent changing the source data objects. Instead, you can make a new copy object instead of manipulating the existing object. For example, if we wanted to change the y coordinate of our point, we can define the following method:

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

This creates a new MyPoint2D object and returns it with the new y-coordinate. Same can be done for changing the x coordinate.

Creating a Data Type

There are two different data types that can be made: Simple data types that have only one field (ie. a single number or string), and Complex data types that have multiple data fields (ie. multiple strings, multiple numbers).

In order to define a simple data type, the class must extend the SimpleDataType<DataType> class with the data type needed and implement the getDefaultValue() method. In this example, we will use a double as our simple data type.

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

    private static final String NAME = "Double";

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

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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

```

The class constructor is set to private to ensure that only a single instance of the data type will exist.

In order to define a complex data type, the class must extend the `ComplexDataType` class and override the `fromMap()` and `getDefaultValue()` methods. We will use our `MyPoint2D` class as an example to see what a complex data type class would look like.

```

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);
    }
}

```

The following code above works as noted:

The `fromMap()` method creates a new `MyPoint2D` using the values in the `NetworkTables` entry it is bound to. The `getOrDefault` method will return 0.0 if it cannot get the entry values. The `getDefaultValue` will return a new `MyPoint2D` object if no source is present.

Exporting Data Type To Plugin

In order to have the data type be recognized by Shuffleboard, the plugin must export them by overriding the `getDataTypes` method. For example,

```

public class MyPlugin extends Plugin {

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

```

(sonraki sayfaya devam)

```
}  
  
}
```

Creating A Widget

Widgets allow us to view, change, and interact with data published through different data sources. The CameraServer, NetworkTables, and Base plugins provide the widgets to control basic data types (including FRC-specific data types). However, custom widgets allow us to control our custom data types we made in the previous sections or Java Objects.

The basic `Widget` interface inherits from the `Component` and `Sourced` interfaces. `Component` is the most basic building block of components that be displayed in Shuffleboard. `Sourced` is an interface for things that can handle and interface with data sources to display or modify data. Widgets that don't support data bindings but simply have child nodes would not use the `Sourced` interface but simply the `Component` interface. Both are basic building blocks towards making widgets and allows us to modify and display data.

A good widget allows the end-user to customize the widget to suit their needs. An example could be to allow the user to control the range of the number slider, that is, its maximum and minimum or the orientation of the slider itself. The view of the widget or how it looks is defined using FXML. FXML is an XML based language that is useful for defining the static layout of the widget (Panels, Labels and Controls).

More about FXML can be found [here](#).

Defining a Widget's FXML

In this example, we will create two sliders to help us control the X and Y coordinates of our `Point2D` data type we created in previous sections. It is helpful to place the FXML file in the same package as the Java class.

In order to create an empty, blank window for our widget, we need to create a `Pane`. A `Pane` is a parent node that contains other child nodes, in this case, 2 sliders. There are many different types of `Pane`, they are as noted:

- **Stack Pane**
 - Stack Panes allow elements to be overlaid. Also, StackPanes by default center child nodes.
- **Grid Pane**
 - Grid Panes are extremely useful defining child elements using a coordinate system by creating a flexible grid of rows and columns on the pane.
- **Flow Pane**
 - Flow Panes wrap all child nodes at a boundary set. Child nodes can flow vertically (wrapped at the height boundary for the pane) or horizontally (wrapped at the width boundary of the pane).
- **Anchor Pane**
 - Anchor Panes allow child elements to be placed in the top, bottom, left side, right side, or center of the pane.

Layout panes are also extremely useful for placing child nodes in one horizontal row using a `HBox` or one vertical column using a `VBox`.

The basic syntax for defining a Pane using FXML would be as the following:

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

The `fx:controller` attribute contains the name of the widget class. An instance of this class is created when the FXML file is loaded. For this to work, the controller class must have a no-argument constructor.

Creating A Widget Class

Now that we have a Pane, we can now add child elements to that pane. In this example, we can add two slider objects. Remember to add an `fx:id` to each element so they can be referenced in our Java class we will make later on. We will use a `VBox` to position our slider on top of each other.

```
<?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>
```

Now that we have finished creating our FXML file, we can now create a widget class. The widget class should include a `@Description` annotation that states the supported data types of the widget and the name of the widget. If a `@Description` annotation is not present, the plugin class must implement the `get()` method to return its widgets.

It also must include a `@ParametrizedController` annotation that points to the FXML file containing the layout of the widget. If the class that only supports one data source it must extend the `SimpleAnnotatedWidget` class. If the class supports multiple data sources, it must extend the `ComplexAnnotatedWidget` class. For more information, see *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)
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
@ParametrizedController("Point2DWidget.fxml")
public final class Point2DWidget extends SimpleAnnotatedWidget<MyPoint2D> {
}

```

If you are not using a custom data type, you can reference any Java data type (ie. Double.class), or if the widget does not need data binding you can pass NoneType.class.

Now that we have created our class we can create fields for the widgets we declared in our FXML file using the @FXML annotation. For our two sliders, an example would be:

```
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;
}

```

In order to display our pane on our custom widget we need to override the getView() method and return our 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;
    }
}

```

Binding Elements and Adding Listeners

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.

An example, in this case, would be changing the X and Y coordinate of our 2D point by changing the values of `xSlider` and `ySlider` respectively.

A good practice is to set bindings in the `initialize()` method tagged with the `@FXML` annotation which is required to call the method from FXML if the method is not public.

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

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

    @FXML
    private StackPane root;

    @FXML
    private Slider xSlider;

    @FXML
    private Slider ySlider;

    @FXML
    private void initialize() {
        xSlider.valueProperty().bind(dataOrDefault.map(MyPoint2D::getX));
        ySlider.valueProperty().bind(dataOrDefault.map(MyPoint2D::getY));
    }

    @Override
    public Pane getView() {
        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.

Using a listener is another way to change values when the slider or data source has changed. For example a listener for our slider would be:

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

In this case, the `setData()` method sets the value in the data source of the widget to the `newValue`.

Exploring Custom Components

Widgets are not automatically discovered when loading plugins; the defining plugin must explicitly export it for it to be usable. This approach is taken to allow multiple plugins to be defined in the same JAR.

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

Set Default Widget For Data type

In order to set your widget as default for your custom data type, you can override the `getDefaultComponents()` in your plugin class that stores a `Map` for all default widgets as noted below:

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

Custom Themes

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 comes with three themes by default: Material Light, Material Dark, and Midnight. These are color variations on the same material design stylesheet. In addition, they inherit from a `base.css` stylesheet that defines styles for the custom components defined in shuffleboard or libraries that it uses; the base material design stylesheet only applies to the UI components built into JavaFX.

There are two ways to define a custom theme: place the stylesheets in a directory with the name of the theme in `~/Shuffleboard/themes`; for example, a theoretical “Yellow” theme could be placed in

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

All the stylesheets in the directory will be treated as part of the theme.

Loading Themes via Plugins

Custom themes can also be defined by plugins. This makes them easier to share and bundle with custom widgets, but are slightly more difficult to define. The theme object will need a reference to a class defined in the plugin so that the plugin loader can determine where the stylesheets are located. If a class is passed that is *not* present in the JAR that the plugin is in, the theme will not be able to be used.

```
@Description(group = "com.example", name = "My Plugin", version = "1.2.3", summary = "
↳ ")
class MyPlugin extends Plugin {

    private static final Theme myTheme = new Theme(MyPlugin.class, "My Theme Name", "/
↳ path/to/stylesheet", "/path/to/stylesheet", ...);

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

Modifying or Extending Shuffleboard's Default Themes

Shuffleboard's Material Light and Material Dark themes provide a lot of the framework for light and dark themes, respectively, as well as many styles specific to shuffleboard, Controls-FX, and Medusa UI components to fit with the material-style design.

Themes that want to modify these themes need to add import statements for these stylesheets:

```
@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 */
```

Note that base.css internally imports material.css, and light.css, dark.css, and midnight.css all import base.css, so importing light.css will implicitly import both base.css and material.css as well.

Source Code for the CSS Files

- `_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>

Material Design Color Swatches

The material design CSS uses color swatch variables for almost everything. These variables can be set from custom CSS files, reducing the amount of custom code needed.

The `-swatch-<100|200|300|400|500>` variables define progressively darker shades of the same primary color. The light theme uses the default shades of blue set in `material.css`, but the dark theme overrides these with shades of red. `-swatch-<|light|dark>-gray` defines three levels of gray to use for various background or text colors.

Overriding the Swatch Colors

Replacing blue with red (light)

```
@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%);
}
```

Replacing red with blue (dark)

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

.root {
  -swatch-100: #BBDEFB;
  -swatch-200: #90CAF9;
  -swatch-300: #64B5F6;
  -swatch-400: #42A5F5;
  -swatch-500: #2196F3;
}
```

Widget Types

While `Widget` is pretty straightforward as far as the interface is concerned, there are several intermediate implementations to make it easier to define the widget.

Class	Description
<code>AbstractWidget</code>	Implements <code>getProperties()</code> , <code>getSources()</code> , and <code>titleProperty()</code>
<code>SingleTypeWidget<T></code>	Adds properties for widgets that only support a single data type
<code>AnnotatedWidget</code>	Adds default implementations for <code>getName()</code> and <code>getDataTypes()</code> for widgets with a <code>@Description</code> annotation
<code>SingleSourceWidget</code>	For widgets with only a single source (by default, widgets support multiple sources)
<code>SimpleAnnotatedWidget<T></code>	Combines <code>SingleTypeWidget<T></code> , <code>AnnotatedWidget</code> , and <code>SingleSourceWidget</code>

There are also two annotations to help define widgets:

Name	Description
<code>@ParametrizedController</code>	Allows widgets to be FXML controllers for JavaFX views defined via FXML
<code>@Description</code>	Lets the name and supported data types be defined in a single line

AbstractWidget

This class implements `getProperties()`, `getSources()`, `addSource()`, and `titleProperty()`. It also defines a method `exportProperties(Property<?>...)` method so subclasses can easily add custom widget properties, or properties for the JavaFX components in the widget. Most of the [widgets in the base plugin](#) use this.

SingleTypeWidget

A type of widget that only supports a single data type. This interface is parametrized and has methods for setting or getting the data, as well as a method for getting the (single) data type of the widget.

AnnotatedWidget

This interface implements `getDataTypes()` and `getName()` by looking at the `@Description` annotation on the implementing class. This *requires* the annotation to be present, or the widget will not be able to be loaded and used.

```
// No @Description annotation!
public class WrongImplementation implements AnnotatedWidget {
    // ...
}
```

```
@Description(name = ..., dataTypes = ...)
public class CorrectImplementation implements AnnotatedWidget {
    // ...
}
```

SingleSourceWidget

A type of widget that only uses a single source.

SimpleAnnotatedWidget

A combination of `SingleTypeWidget<T>`, `AnnotatedWidget`, and `SingleSourceWidget`. Most widgets in the base plugin extend from this class. This also has a protected field called `dataOrDefault` that lets subclasses use a default data value if the widget doesn't have a source, or if the source is providing null.

@ParametrizedController

This annotation can be placed on a widget class to let shuffleboard know that it's an FXML controller for a JavaFX view defined via FXML. The annotation takes a single parameter that defines where the FXML file *in relation to the class on which it is placed*. For example, a widget in the directory `src/main/java/com/acme` that is an FXML controller for a FXML file in `src/main/resources/com/acme` can use the annotation as either

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

or as

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

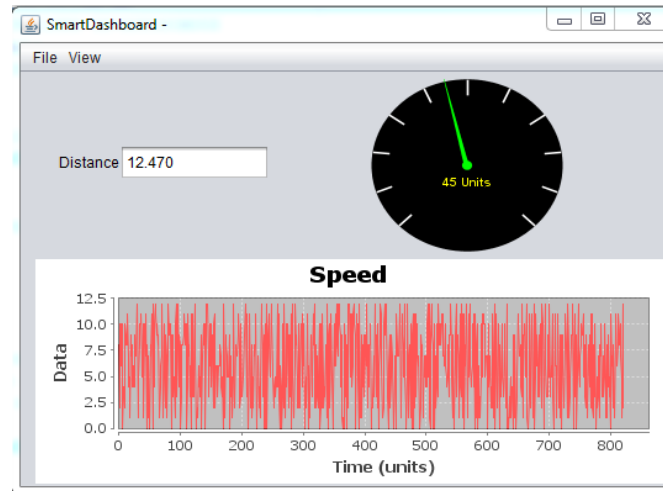
@Description

This allows widgets to have their name and supported data types defined by a single annotation, when used alongside [AnnotatedWidget](#).

11.3 SmartDashboard

SmartDashboard is a simple and efficient dashboard that uses relatively few computer resources. It does not have the fancy look or some of the features Shuffleboard has, but it displays network tables data with a variety of widgets without bogging down the driver station computer.

11.3.1 SmartDashboard Introduction

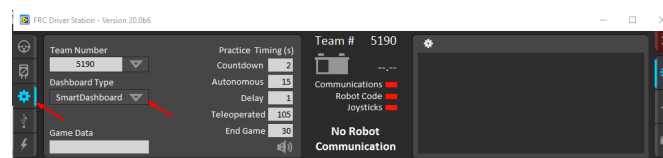


The SmartDashboard is a Java program that will display robot data in real time. The SmartDashboard helps you with these things:

- 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

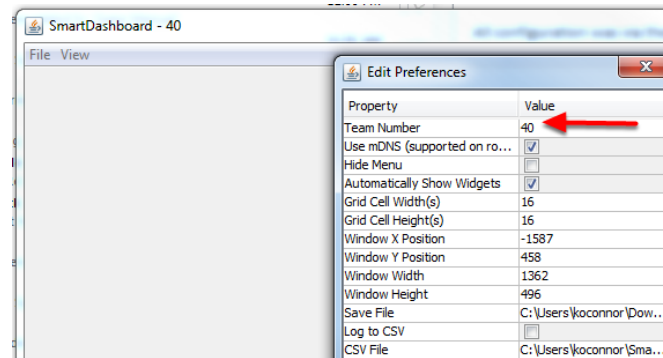
The displayed data is automatically formatted in real-time as the data is sent from the robot, but you can change the format or the display widget types and then save the new screen layouts to be used again later. And with all these options, it is still extremely simple to use. To display some data on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



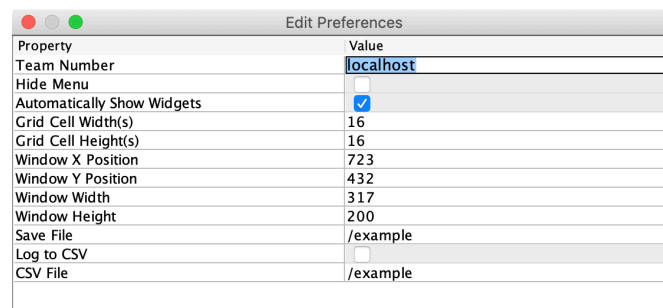
The first time you launch the SmartDashboard you should be prompted for your team number. To change the team number after this: click **File > Preferences** to open the Preferences dialog. Double-click the box to the right of **Team Number** and enter your FRC® Team Number, then click outside the box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

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

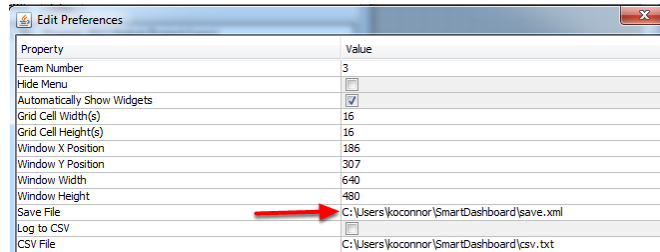
Setting a Custom NetworkTables Server Location

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!

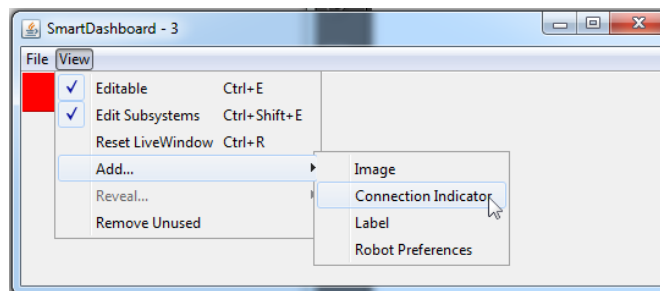


Locating the Save File



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 Displaying Expressions from a Robot Program

Not: Often debugging or monitoring the status of a robot involves writing a number of values to the console and watching them stream by. With SmartDashboard you can put values to a GUI that is automatically constructed based on your program. As values are updated, the corresponding GUI element changes value - there is no need to try to catch numbers streaming by on the screen.

Writing Values to SmartDashboard

JAVA

```
protected void execute() {
    SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge());
    SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition());
    SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle());
    SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder());
    SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder());
    SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle());
    SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage());
    SmartDashboard.putNumber("RPM", shooter.getRPM());
}
```

C++

```
void Command::Execute() {
    frc::SmartDashboard::PutBoolean("Bridge Limit", BridgeTipper.AtBridge());
    frc::SmartDashboard::PutNumber("Bridge Angle", BridgeTipper.GetPosition());
    frc::SmartDashboard::PutNumber("Swerve Angle", Drivetrain.GetSwerveAngle());
    frc::SmartDashboard::PutNumber("Left Drive Encoder", Drivetrain.GetLeftEncoder());
    frc::SmartDashboard::PutNumber("Right Drive Encoder", Drivetrain.
    GetRightEncoder());
    frc::SmartDashboard::PutNumber("Turret Pot", Turret.GetCurrentAngle());
    frc::SmartDashboard::PutNumber("Turret Pot Voltage", Turret.GetAverageVoltage());
    frc::SmartDashboard::PutNumber("RPM", Shooter.GetRPM());
}
```

PYTHON

```
from wpilib import SmartDashboard

SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge())
SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition())
SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle())
SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder())
SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder())
SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle())
SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage())
SmartDashboard.putNumber("RPM", shooter.getRPM())
```

You can write Boolean, Numeric, or String values to the SmartDashboard by simply calling the correct method for the type and including the name and the value of the data, no additional code is required. Any time in your program that you write another value with the same name, it appears in the same UI element on the screen on the driver station or development computer. As you can imagine this is a great way of debugging and getting status of your robot as it is operating.

Creating Widgets on SmartDashboard

Widgets are populated on the SmartDashboard automatically, no user intervention is required. Note that the widgets are only populated when the value is first written, you may need to enable your robot in a particular mode or trigger a particular code routine for an item to appear. To alter the appearance of the widget, see the next two sections *Changing the Display Properties of a Value* and *Changing the Display Widget Type for a Value*.

Stale Data

SmartDashboard uses *NetworkTables* for communicating values between the robot and the driver station laptop. NetworkTables acts as a distributed table of name and value pairs. If a name/value pair is added to either the client (laptop) or server (robot) it is replicated to the other. If a name/value pair is deleted from, say, the robot but the SmartDashboard or OutlineViewer are still running, then when the robot is restarted, the old values will still appear in the SmartDashboard and OutlineViewer because they never stopped running and continue to have those values in their tables. When the robot restarts, those old values will be replicated to the robot.

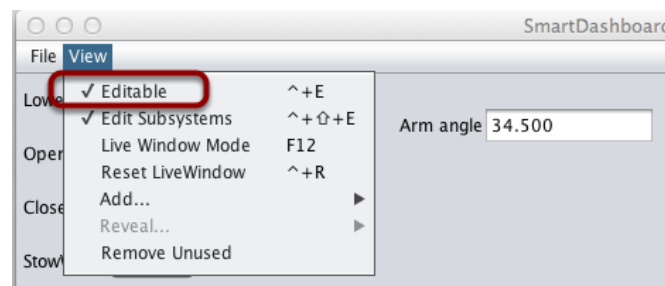
To ensure that the SmartDashboard and OutlineViewer are showing current values, it is necessary to restart the NetworkTables clients and robot at the same time. That way, old values that one is holding won't get replicated to the others.

This usually isn't a problem if the program isn't constantly changing, but if the program is in development and the set of keys being added to NetworkTables is constantly changing, then it might be necessary to do the restart of everything to accurately see what is current.

11.3.3 Changing the display properties of a value

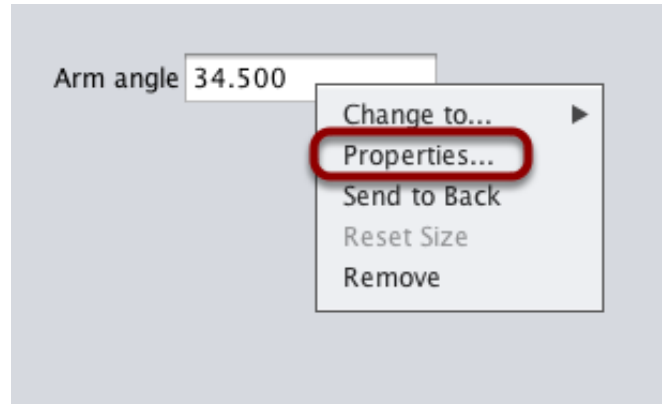
Each value displayed with SmartDashboard has a set of properties that effect the way it's displayed.

Setting the SmartDashboard display into editing mode



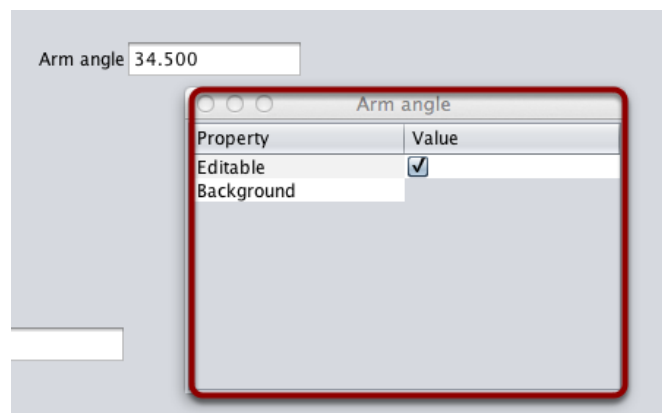
The SmartDashboard has two modes it can operate in, display mode and edit mode. In edit mode you can move around widgets on the screen and edit their properties. To put the SmartDashboard into edit mode, click the "View" menu, then select "Editable" to turn on edit mode.

Getting the properties editor of a widget



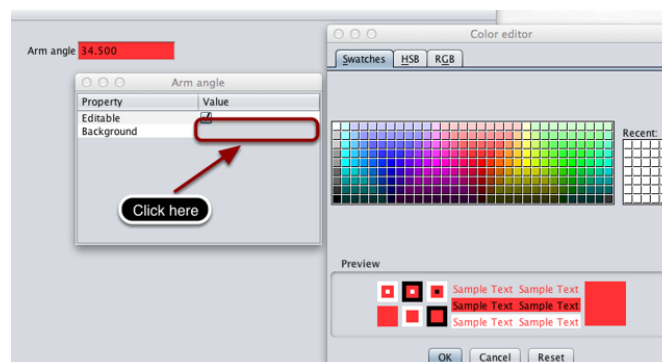
Once in edit mode, you can display and edit the properties for a widget. Right-click on the widget and select “Properties...”.

Editing the properties on a field



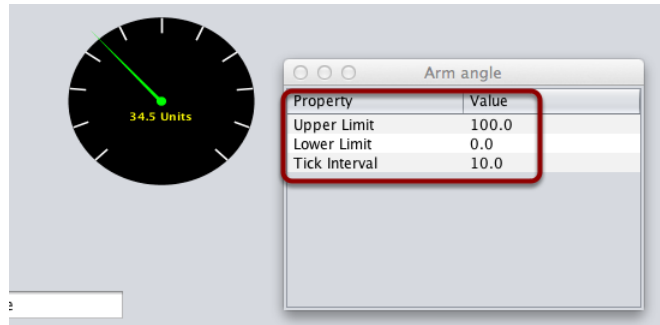
A dialog box will be shown in response to the “Properties...” menu item on the widgets right-click context menu.

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

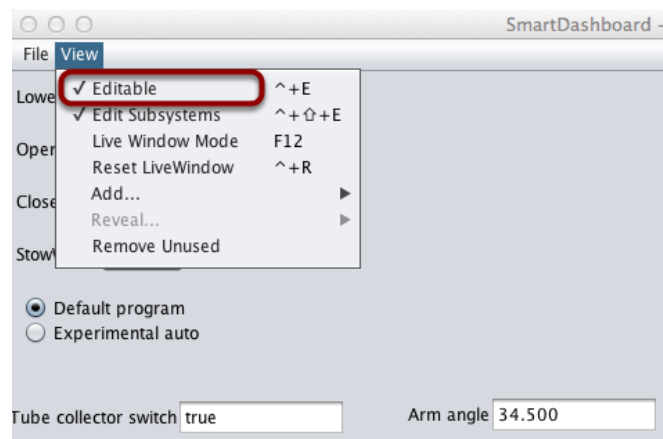


Different widget types have different sets of editable properties to change the appearance. In this example, the upper and lower limits of the dial and the tick interval are changeable parameters.

11.3.4 Changing the Display Widget Type for a Value

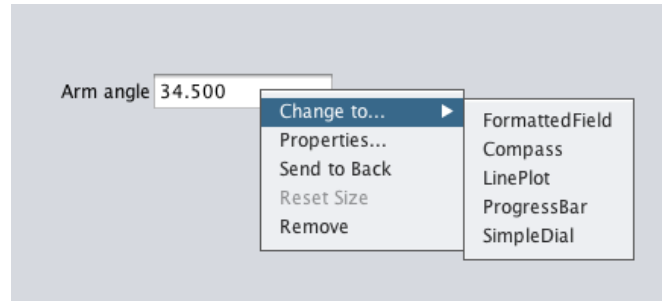
One can change the type of widget that displays values with the SmartDashboard. The allowable widgets depend on the type of the value being displayed.

Setting Edit Mode



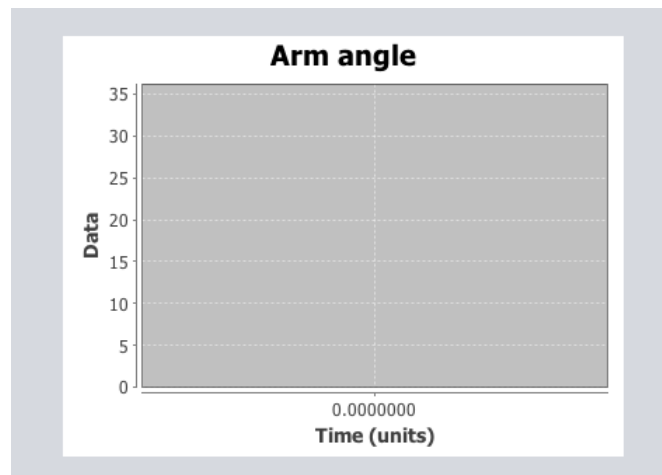
Make sure that the SmartDashboard is in edit mode. This is done by selecting Editable from the View menu.

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 Choosing an Autonomous Program

Often teams have more than one autonomous program, either for competitive reasons or for testing new software. Programs often vary by adding things like time delays, different strategies, etc. The methods to choose the strategy to run usually involves switches, joystick buttons, knobs or other hardware based inputs.

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

Not: 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;
    }
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    }
}

```

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

Not: The code snippets shown below are part of the HatchbotTraditional example project (Java, C++, Python):

Creating the SendableChooser Object

In RobotContainer, create a variable to hold a reference to a SendableChooser object. Two or more commands can be created and stored in new variables. Using the SendableChooser, one can choose between them. In this example, SimpleAuto and ComplexAuto are shown as options.

Java

```
// A simple auto routine that drives forward a specified distance, and then stops.
private final Command m_simpleAuto =
    new DriveDistance(
        AutoConstants.kAutoDriveDistanceInches, AutoConstants.kAutoDriveSpeed, m_
↪robotDrive);

// A complex auto routine that drives forward, drops a hatch, and then drives
↪backward.
private final Command m_complexAuto = new ComplexAuto(m_robotDrive, m_
↪hatchSubsystem);

// A chooser for autonomous commands
SendableChooser<Command> m_chooser = new SendableChooser<>();
```

C++ (using raw pointers)

```
// The autonomous routines
DriveDistance m_simpleAuto{AutoConstants::kAutoDriveDistanceInches,
    AutoConstants::kAutoDriveSpeed, &m_drive};
ComplexAuto m_complexAuto{&m_drive, &m_hatch};

// The chooser for the autonomous routines
frc::SendableChooser<frc2::Command*> m_chooser;
```

C++ (using CommandPtr)

```
// The autonomous routines
frc2::CommandPtr m_simpleAuto = autos::SimpleAuto(&m_drive);
frc2::CommandPtr m_complexAuto = autos::ComplexAuto(&m_drive, &m_hatch);

// The chooser for the autonomous routines
frc::SendableChooser<frc2::Command*> m_chooser;
```

Python

```
# A simple auto routine that drives forward a specified distance, and then
↳ stops.
self.simpleAuto = DriveDistance(
    constants.kAutoDriveDistanceInches, constants.kAutoDriveSpeed, self.drive
)

# A complex auto routine that drives forward, drops a hatch, and then drives
↳ backward.
self.complexAuto = ComplexAuto(self.drive, self.hatch)

# Chooser
self.chooser = wpilib.SendableChooser()
```

Setting up SendableChooser

Imagine that you have two autonomous programs to choose between and they are encapsulated in commands SimpleAuto and ComplexAuto. To choose between them:

In RobotContainer, create a SendableChooser object and add instances of the two commands to it. There can be any number of commands, and the one added as a default (setDefaultOption), becomes the one that is initially selected. Notice that each command is included in an setDefaultOption() or addOption() method call on the SendableChooser instance.

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)
```

Starting an Autonomous Command

In `Robot.java`, when the autonomous period starts, the `SendableChooser` object is polled to get the selected command and that command must be scheduled.

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()

```

Running the Scheduler during Autonomous

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.

Not: Running the scheduler can occur in the autonomousPeriodic() function or robotPeriodic(), both will function similarly in autonomous mode.

Java

```

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

```

C++ (Source)

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

Canceling the Autonomous Command

In `Robot.java`, when the teleop period begins, the autonomous command will be canceled.

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 }
```

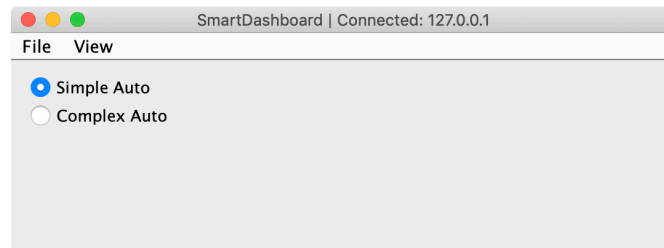
C++ (Source)

```
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 Display

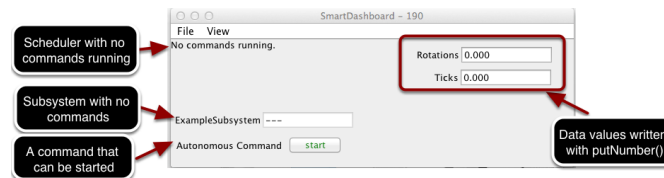


When the SmartDashboard is run, the choices from the SendableChooser are automatically displayed. You can simply pick an option before the autonomous period begins and the corresponding command will run.

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++

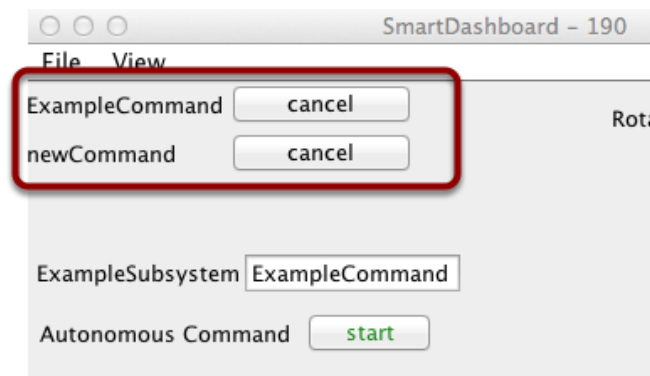
```
frc::SmartDashboard::PutData(frc2::CommandScheduler::GetInstance());
```

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);
```

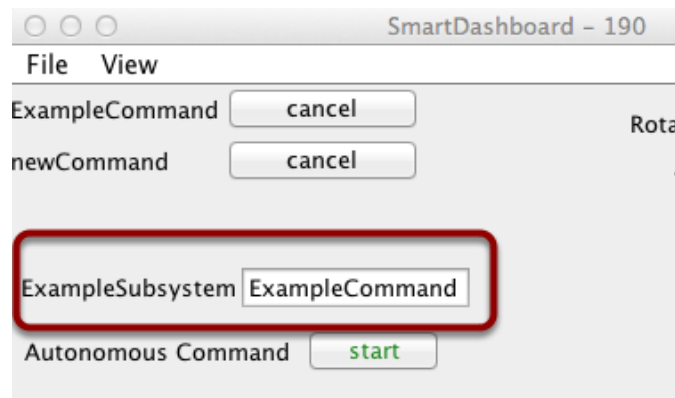
C++

```
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);
```

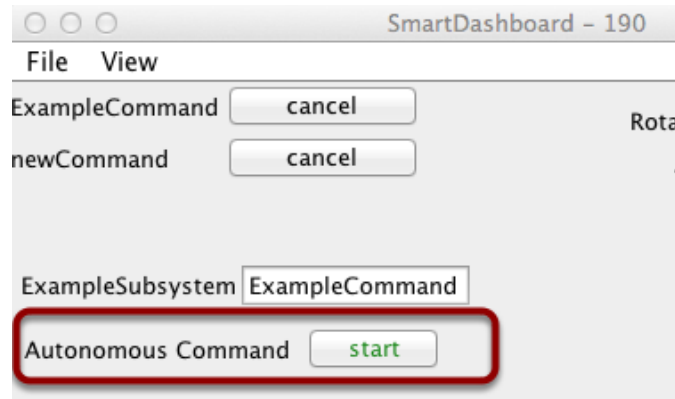
C++

```
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.

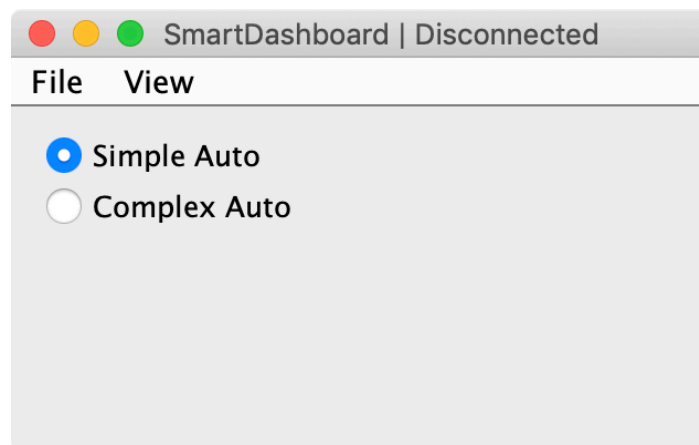


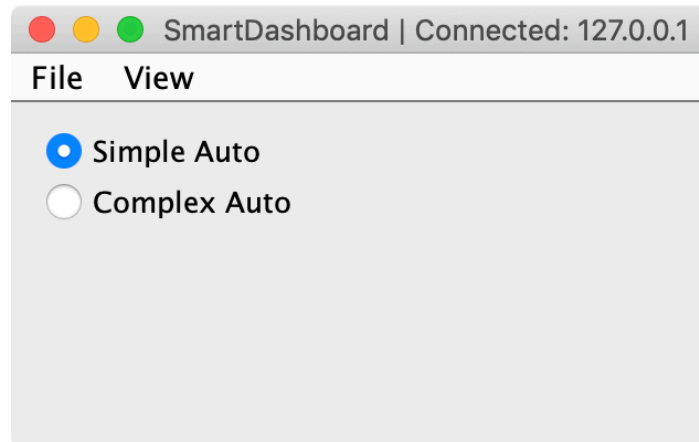
In this example you can see a button labeled Autonomous Command. Pressing this button will run the associated command and is an excellent way of testing commands one at a time without having to add throw-away test code to your robot program. Adding buttons for each command makes it simple to test the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

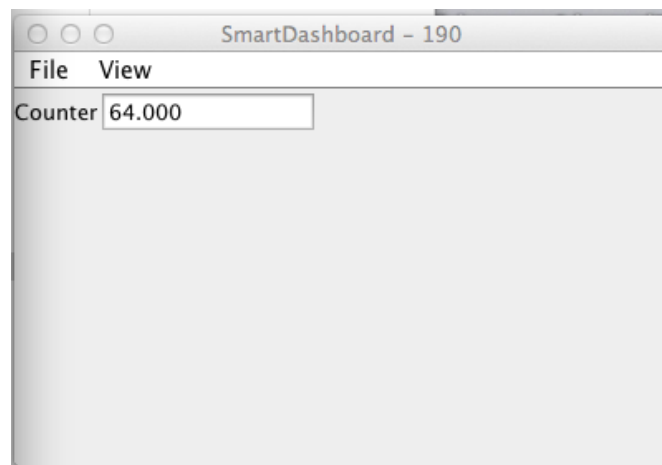
```
from wpilib import SmartDashboard

self.counter = 0.0

def teleopPeriodic(self) -> None:
    SmartDashboard.putNumber("Counter", self.counter += 1)
```

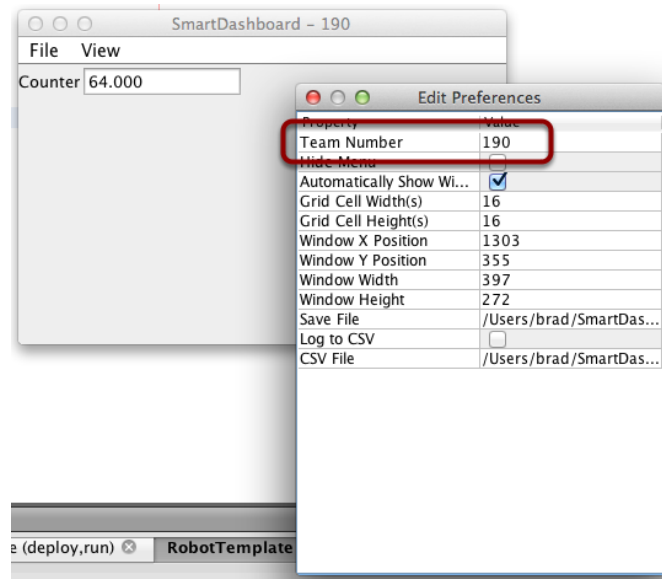
This is a minimal robot program that writes a value to the SmartDashboard. It simply increments a counter 50 times per second to verify that the connection is working. However, to minimize bandwidth usage, NetworkTables by default will throttle the updates to 10 times per second.

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

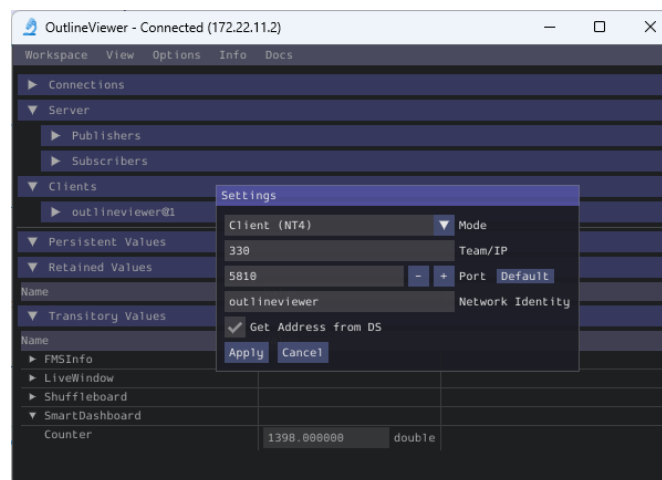


If the display of the value is not appearing, verify that the team number is correctly set as shown in this picture. The preferences dialog can be viewed by selecting File, then Preferences.

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 uses NetworkTables to send data between the robot and the Dashboard (Driver Station) computer. NetworkTables sends data as name, value pairs, like a distributed hashtable between the robot and the computer. When a value is changed in one place, its value is automatically updated in the other place. This mechanism and a standard set of name (keys) is how data is displayed on the SmartDashboard.

There is a hierarchical structure in the name space creating a set of tables and subtables. SmartDashboard data is in the SmartDashboard subtable and LiveWindow data is in the LiveWindow subtable as shown below.

For informational purposes, the names and values can be displayed using the OutlineViewer application that is installed in the same location as the SmartDashboard. It will display all the NetworkTables keys and values as they are updated.

SmartDashboard Data Values

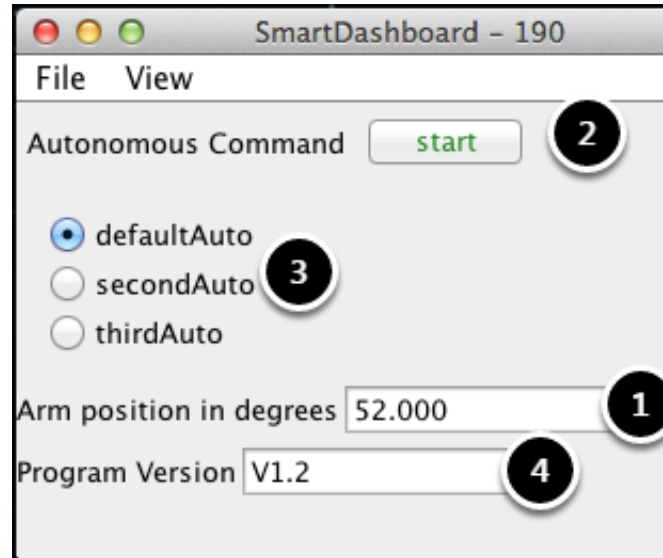
/SmartDashboard/Arm position in degrees	52.0	1
/SmartDashboard/Autonomous Command/~TYPE~	Command	2
/SmartDashboard/Autonomous Command/isParented	false	
/SmartDashboard/Autonomous Command/name	AutonomousCommand	
/SmartDashboard/Autonomous Command/running	false	
/SmartDashboard/Chooser/~TYPE~	String Chooser	3
/SmartDashboard/Chooser/default	defaultAuto	
/SmartDashboard/Chooser/options	[defaultAuto, secondAuto, th	
/SmartDashboard/Program Version	V1.2	4

SmartDashboard values are created with key names that begin with SmartDashboard/. The above values viewed with OutlineViewer correspond to data put to the SmartDashboard with the following statements:

```
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");
```

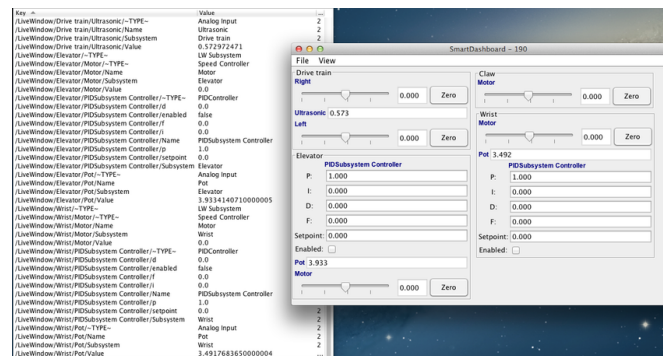
The Arm position is created with the putNumber() call. The AutonomousCommand is written with a putData("Autonomous Command", command) that is not shown in the above code fragment. The chooser is created as a SendableChooser object and the string value, Program Version is created with the putString()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

View of SmartDashboard



The code from the previous step generates the table values as shown and the SmartDashboard display as shown here. The numbers correspond to the NetworkTables variables shown in the previous step.

LiveWindow Data Values



LiveWindow data is automatically grouped by subsystem. The data is viewable in the SmartDashboard when the robot is in Test mode (set on the Driver Station). If you are not writing a command based program, you can still cause sensors and actuators to be grouped for easy viewing by specifying the subsystem name. In the above display you can see the key names and the resultant output in Test mode on the SmartDashboard. All the strings start with / LiveWindow then the Subsystem name, then a group of values that are used to display each element. The code that generates this LiveWindow display is shown below:

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

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

(sonraki sayfaya devam)

(önceki sayfadan devam)

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

drivetrainUltrasonic = new AnalogInput(3);
drivetrainUltrasonic.setName("Drive train", "Ultrasonic");

elevatorMotor = new PWMVictorSPX(6);
elevatorMotor.setName("Elevator", "Motor");

elevatorPot = new AnalogInput(4);
elevatorPot.setName("Elevator", "Pot");

wristPot = new AnalogInput(2);
wristPot.setName("Wrist", "Pot");

wristMotor = new PWMVictorSPX(3);
wristMotor.setName("Wrist", "Motor");

clawMotor = new PWMVictorSPX(5);
clawMotor.setName("Claw", "Motor");
```

Values that correspond to actuators are not only displayed, but can be set using sliders created in the SmartDashboard in Test mode.

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

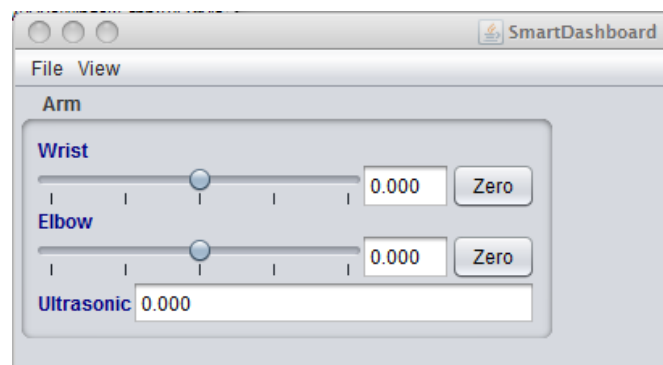
```
from wpilib import Jaguar, Ultrasonic, Victor
from commands2 import SubsystemBase

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.

Önemli: 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 rightDrive{1};
frc::BuiltInAccelerometer accel{};
frc::PWMVictorSPX arm{3};

void Robot::RobotInit() {
    wpi::SendableRegistry::SetName(&arm, "SomeSubsystem", "Arm");
    wpi::SendableRegistry::SetName(&accel, "SomeSubsystem", "Accelerometer");
}
```

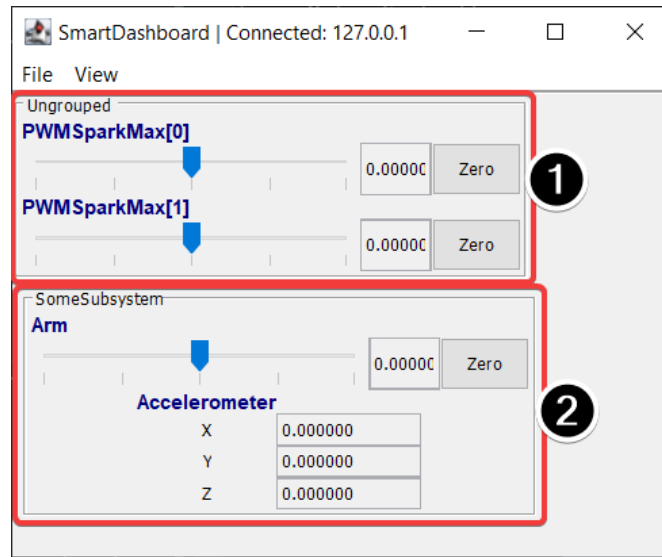
PYTHON

```
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

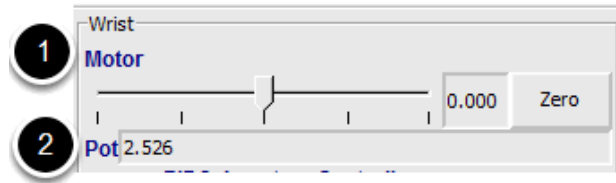


This is the output in the SmartDashboard display when the robot is placed into test mode. In the display shown above the objects listed as Ungrouped were implicitly created by WPILib when the corresponding objects were created. These objects are contained in a subsystem group called “Ungrouped” **(1)** 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.

PID Tuning with SmartDashboard

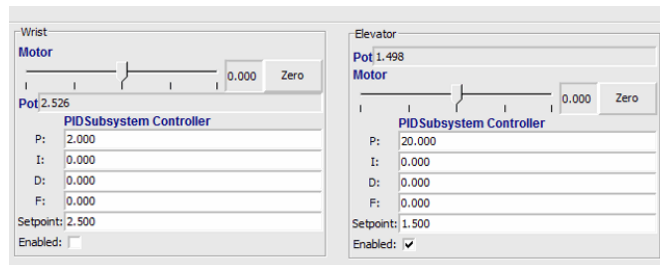
The PID (Proportional, Integral, Differential) is an algorithm for determining the motor speed based on sensor feedback to reach a setpoint as quickly as possible. For example, a robot with an elevator that moves to a predetermined position should move there as fast as possible then stop without excessive overshoot leading to oscillation. Getting the PID controller to behave this way is called “tuning”. The idea is to compute an error value that is the difference between the current value of the mechanism feedback element and the desired (setpoint) value. In the case of the arm, there might be a potentiometer connected to an analog channel that provides a voltage that is proportional to the position of the arm. The desired value is the voltage that is predetermined for the position the arm should move to, and the current value is the voltage for the actual position of the arm.

Finding the setpoint values with LiveWindow



Create a PID Subsystem for each mechanism with feedback. The PID Subsystems contain the actuator (motor) and the feedback sensor (potentiometer in this case). You can use Test mode to display the subsystem sensors and actuators. Using the slider manually adjust the actuator to each desired position. Note the sensor values (2) for each of the desired positions. These will become the setpoints for the PID controller.

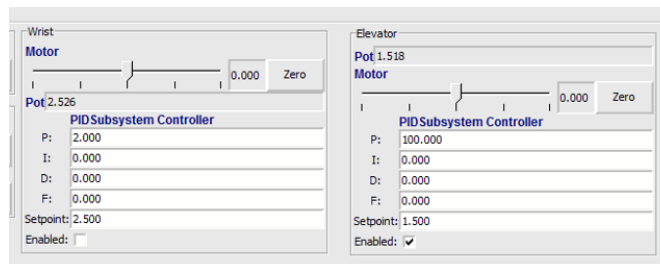
Viewing the PIDController in LiveWindow



In Test mode, the PID Subsystems display their P, I, and D parameters that are set in the code. The P, I, and D values are the weights applied to the computed error (P), sum of errors over time (I), and the rate of change of errors (D). Each of those terms is multiplied by the weights and added together to form the motor value. Choosing the optimal P, I, and D values can be difficult and requires some amount of experimentation. The Test mode on the robot allows the values to be modified, and the mechanism response observed.

Önemli: 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.

Tuning the PIDController



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.

Once you have determined P, I, and D values they can be inserted into the program. You'll find them either in the properties for the PIDSubsystem in RobotBuilder or in the constructor for the PID Subsystem in your code.

The F (feedforward) term is used for controlling velocity with a PID controller.

More information can be found at [WPILib'de PID Kontrolü](#).

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.

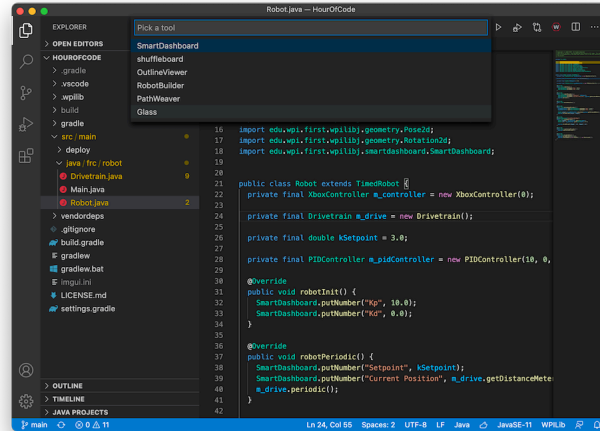
Not: Glass will not be available within the list of dashboards in the NI Driver Station.

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*.

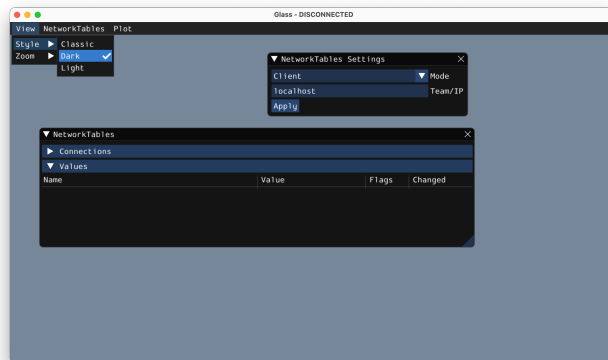


Not: You can also launch Glass directly by navigating to `~/wpilib/YYYY/tools` and running `Glass.py` (Linux and macOS) or by using the shortcut inside the WPILib Tools desktop folder (Windows).

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.

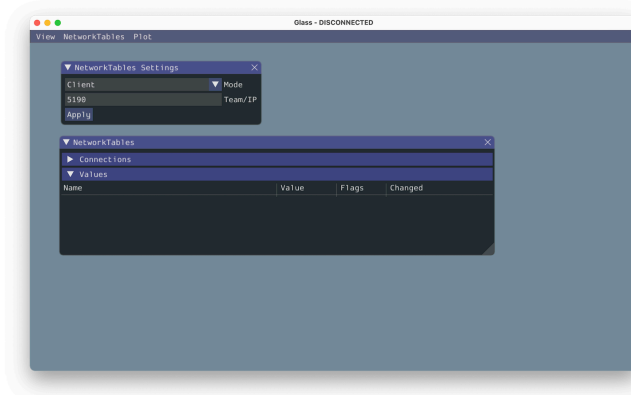
The `glass.ini` configuration file can simply be deleted to restore Glass to a “clean slate”.

11.4.2 Establishing NetworkTables Connections

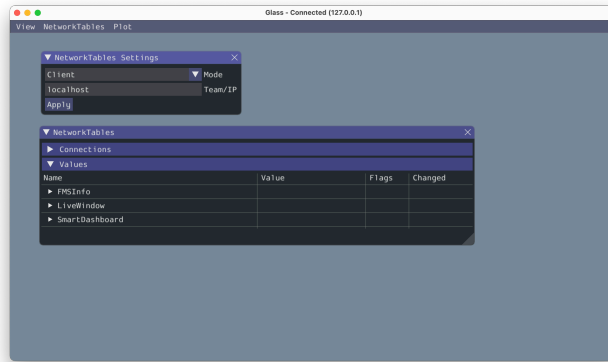
Glass uses the *NetworkTables* protocol to establish a connection with your robot program. It is also used to transmit and receive data to and from the robot.

Connecting to a Robot

When Glass is first launched, you will see two widgets – *NetworkTables Settings* and *NetworkTables*. To connect to a robot, select *Client* under *Mode* in the *NetworkTables Settings* widget, enter your team number and click on *Apply*.



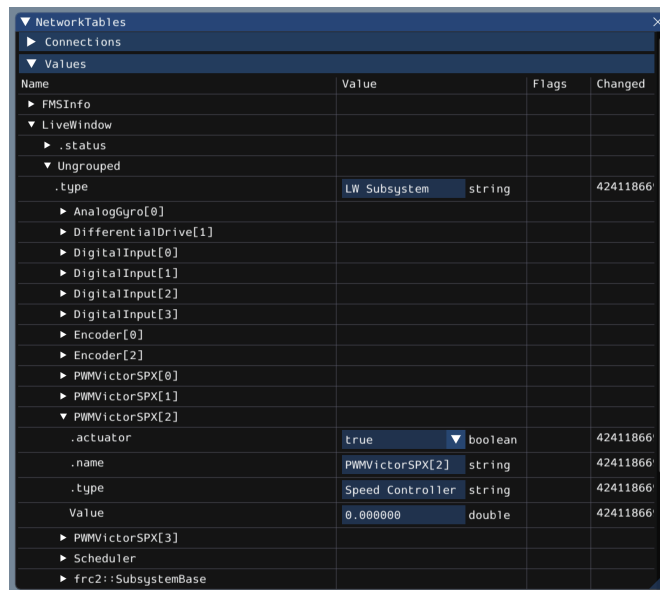
You can also connect to a robot that is running in simulation on your computer (including Romi robots) by typing in `localhost` into the *Team/IP* box.



Önemli: The NetworkTables connection status is always visible on the title bar of the Glass application.

Viewing NetworkTables Entries

The *NetworkTables* widget can be used to view all entries that are being sent over NetworkTables. These entries are hierarchically arranged by main table, sub-table, and so on.



Furthermore, you can view all connected NetworkTables clients under the *Connections* pane of the widget.

11.4.3 Glass Widgets

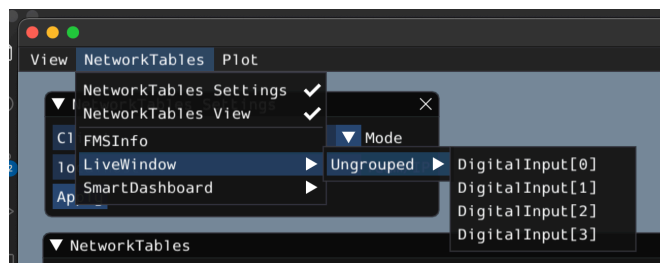
Specialized widgets are available for certain types that exist in robot code. These include objects that are manually sent over NetworkTables such as `SendableChooser` instances, or hardware that is automatically sent over *LiveWindow*.

Not: 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.

Not: 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);
```

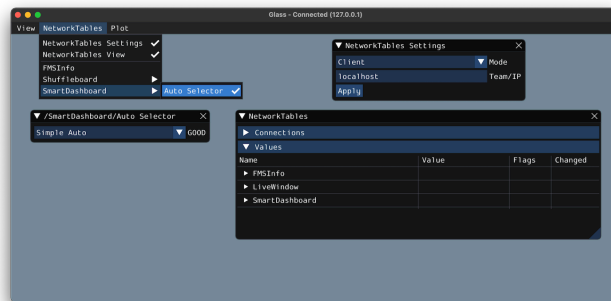
PYTHON

```
from wpilib import SmartDashboard

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

Not: 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++

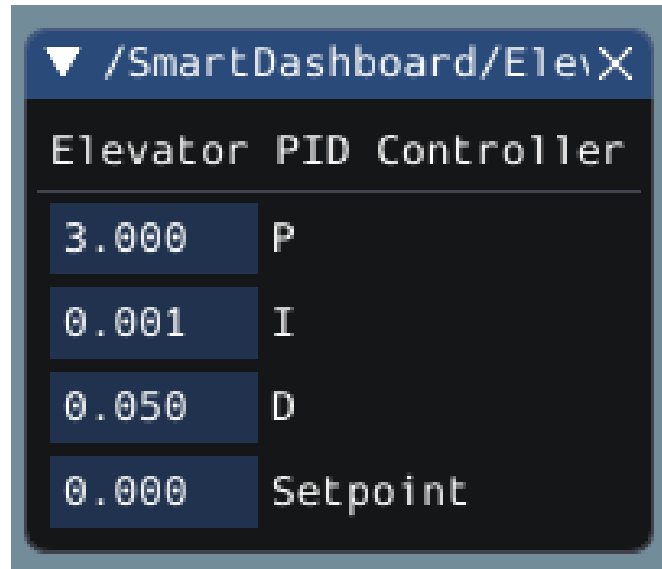
```
frc::SmartDashboard::PutData("Elevator PID Controller", &m_elevatorPIDController);
```

PYTHON

```
from wpilib import SmartDashboard

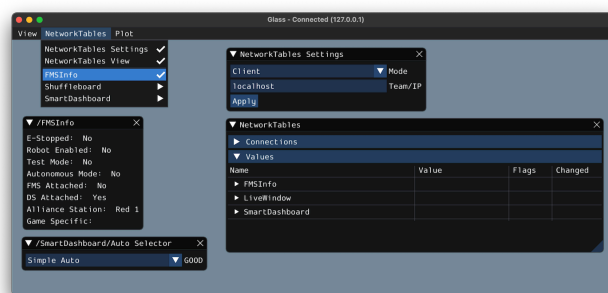
SmartDashboard.putData("Elevator PID Controller", elevatorPIDController)
```

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++

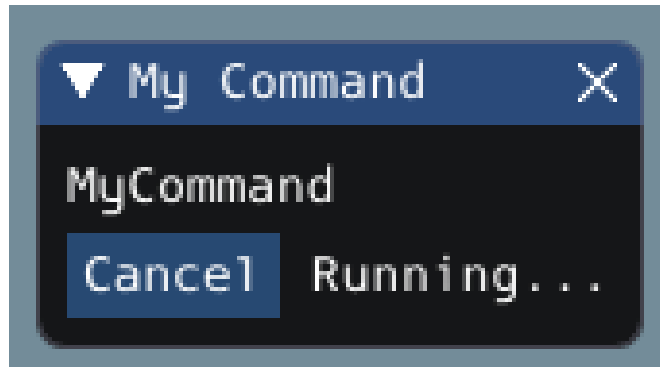
```
#include <frc/smartdashboard/SmartDashboard.h>
...
MyCommand command{...};
frc::SmartDashboard::PutData("My Command", &command);
```

PYTHON

```
from wpilib import SmartDashboard

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

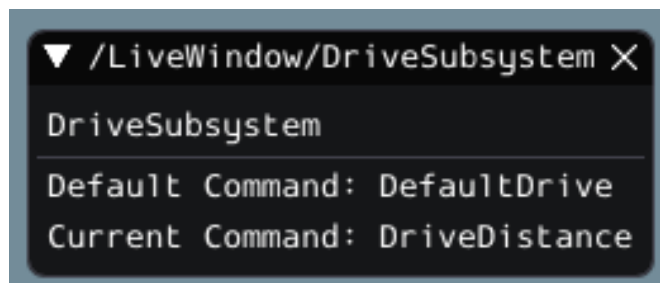
Not: The `MyCommand` instance can also be sent using a lower-level NetworkTables API or using the *Shuffleboard API*. In this case, the `SmartDashboard` API was used, meaning that the *Command Selector* widget will appear under the `SmartDashboard` table name.



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.

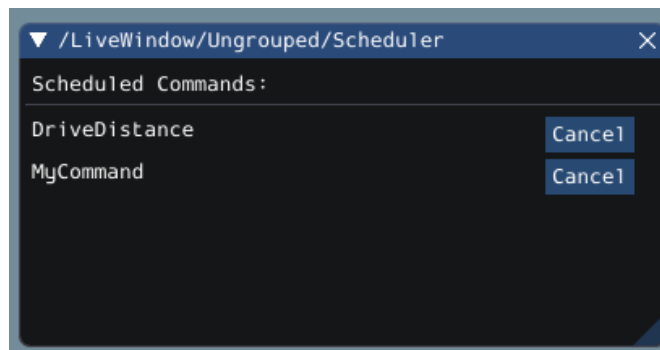
Subsystem Widget

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())
```

Not: 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)
        ),
    )

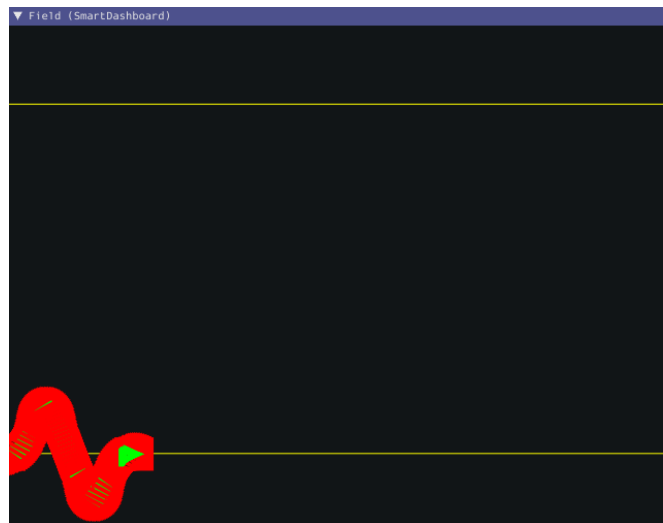
    # Create Field2d for robot and trajectory visualizations.
    self.field = wpilib.Field2d()

    # 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

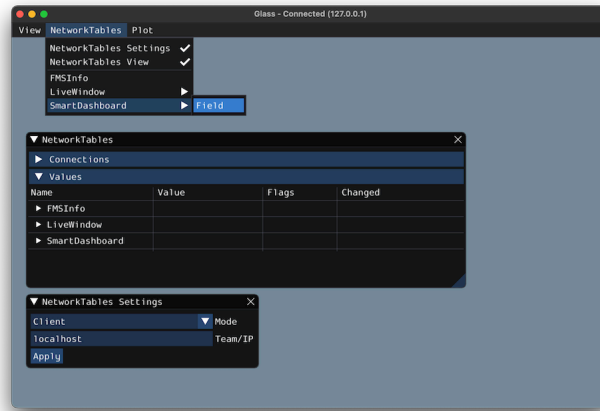
The sent trajectory can be viewed with *Glass* through the dropdown *NetworkTables* -> *SmartDashboard* -> *Field2d*.



Not: 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

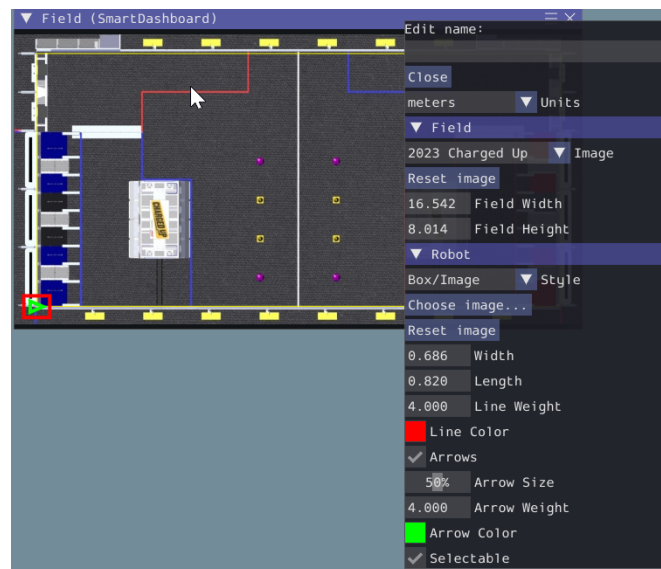
After sending the `Field2d` instance over `NetworkTables`, the *Field2d* widget can be added to Glass by selecting *NetworkTables* in the menu bar, choosing the table name that the instance was sent over, and then clicking on the *Field* button.



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.

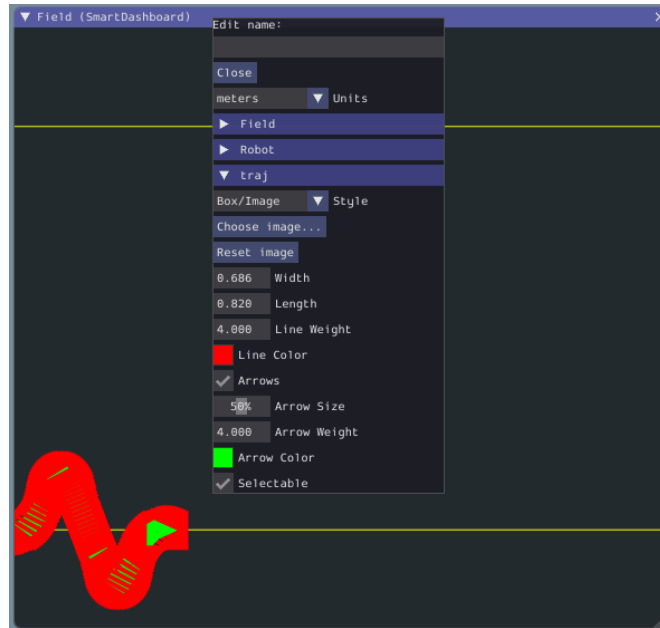
You can choose from an existing field layout using the *Image* drop-down. Or you can select a custom file by setting the *Image* to Custom and selecting *Choose image....* You can choose to either select an image file or a PathWeaver JSON file as long as the image file is in the same directory. Choosing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

Not: 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*.

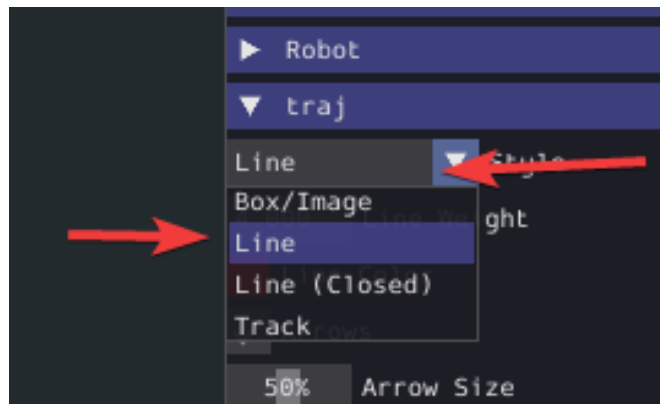


Modifying Pose Style

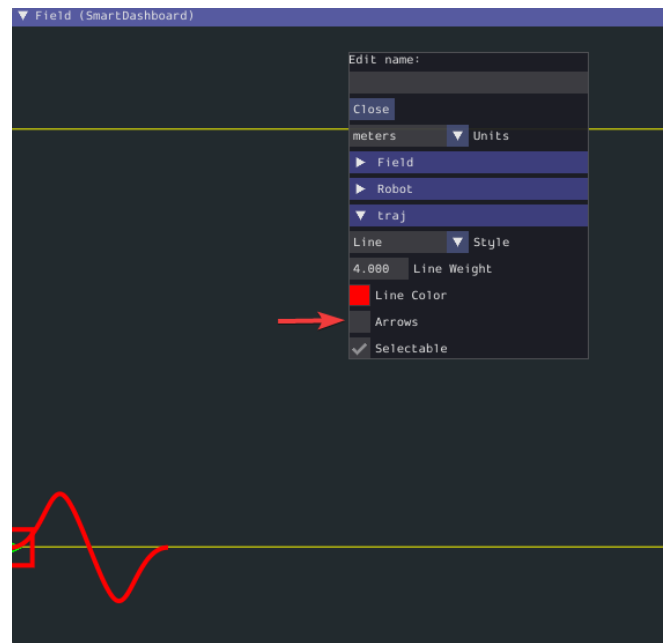
Poses can be customized in a plethora of ways by right clicking on the Field2d menu bar. Examples of customization are: line width, line weight, style, arrow width, arrow weight, color, etc.



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.

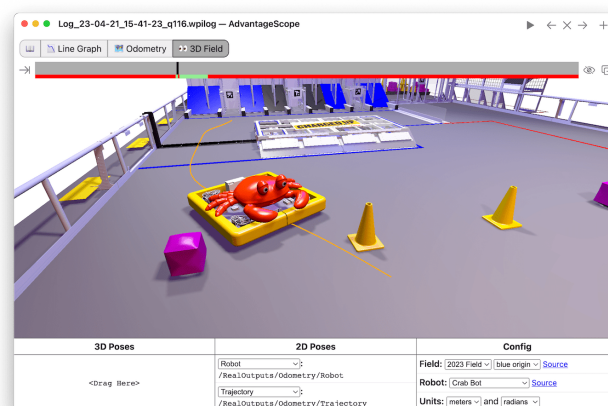


Now, uncheck the *Arrows* checkbox. This will cause our trajectory to look like a nice and fluid line!



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 be 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

```
48 // MechanismLigament2d objects represent each "section"/"stage" of the mechanism,
↳ and are based
49 // off the root node or another ligament object
50 m_elevator = root.append(new MechanismLigament2d("elevator",
↳ kElevatorMinimumLength, 90));
51 m_wrist =
52 m_elevator.append(
53 new MechanismLigament2d("wrist", 0.5, 90, 6, new Color8Bit(Color.
↳ kPurple)));
```

C++

```
63 // MechanismLigament2d objects represent each "section"/"stage" of the
64 // mechanism, and are based off the root node or another ligament object
65 frc::MechanismLigament2d* m_elevator =
66 m_root->Append<frc::MechanismLigament2d>("elevator", 1, 90_deg);
67 frc::MechanismLigament2d* m_wrist =
68 m_elevator->Append<frc::MechanismLigament2d>(
69 "wrist", 0.5, 90_deg, 6, frc::Color8Bit{frc::Color::kPurple});
```

PYTHON

```
37 # MechanismLigament2d objects represent each "section"/"stage" of the
↳ mechanism, and are based
38 # off the root node or another ligament object
39 self.elevator = self.root.appendLigament(
40 "elevator", self.kElevatorMinimumLength, 90
41 )
42 self.wrist = self.elevator.appendLigament(
43 "wrist", 0.5, 90, 6, wpilib.Color8Bit(wpilib.Color.kPurple)
44 )
```

Then, publish the `Mechanism2d` object to NetworkTables:

JAVA

```
55 // post the mechanism to the dashboard
56 SmartDashboard.putData("Mech2d", mech);
```

C++

```
36 // publish to dashboard
37 frc::SmartDashboard::PutData("Mech2d", &m_mech);
```

PYTHON

```
46 # post the mechanism to the dashboard
47 wpilib.SmartDashboard.putData("Mech2d", self.mech)
```

Not: 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 Mechanism-Ligament2d 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

```
59 @Override
60 public void robotPeriodic() {
61     // update the dashboard mechanism's state
62     m_elevator.setLength(kElevatorMinimumLength + m_elevatorEncoder.getDistance());
63     m_wrist.setAngle(m_wristPot.get());
64 }
```

C++

```
40 void RobotPeriodic() override {
41     // update the dashboard mechanism's state
42     m_elevator->SetLength(kElevatorMinimumLength +
43                         m_elevatorEncoder.GetDistance());
44     m_wrist->SetAngle(units::degree_t{m_wristPotentiometer.Get()});
45 }
```

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())

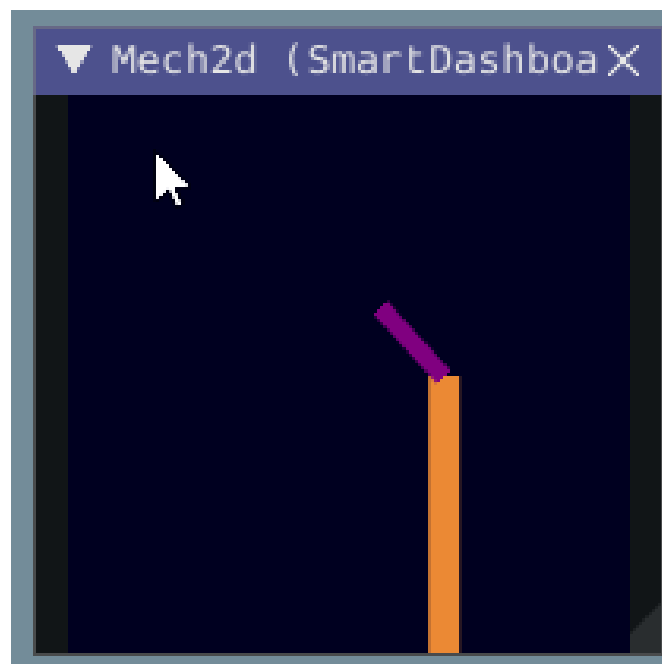
```

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.

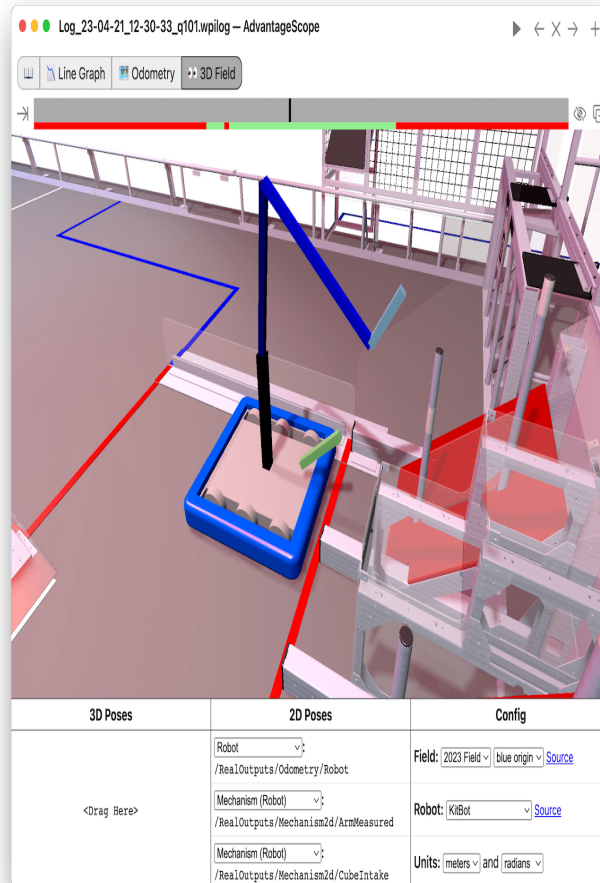


Once the widget appears as shown below, you can resize and place it on the Glass workspace as you desire. Right-clicking the top of the widget will allow you to customize the name of the widget. As the wrist potentiometer and elevator encoder changes, the mechanism will update in the widget.



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

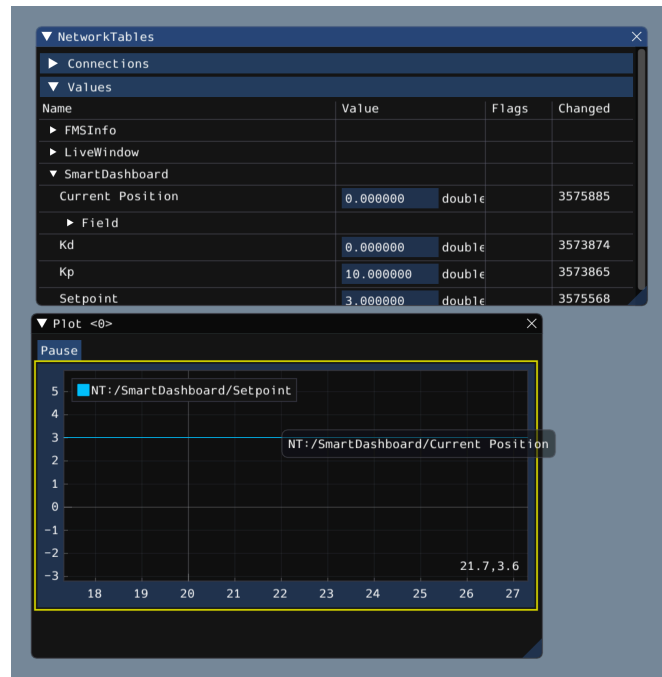
As mentioned above, the Mechanism2d visualization can be combined with *Physics Simulation* to help you program mechanisms before your robot is built. The *ArmSimulation* ([Java](#) / [C++](#) / [Python](#)) and *ElevatorSimulation* ([Java](#) / [C++](#) / [Python](#)) examples combine physics simulation 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.

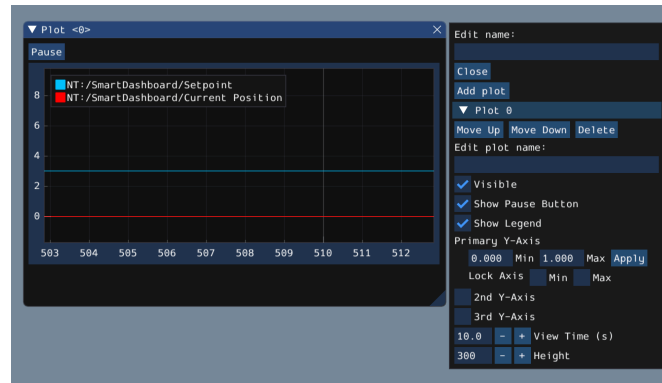
Creating a Plot

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:

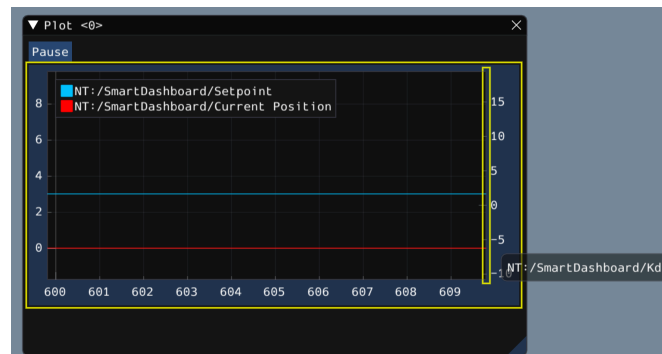


Manipulating Plots

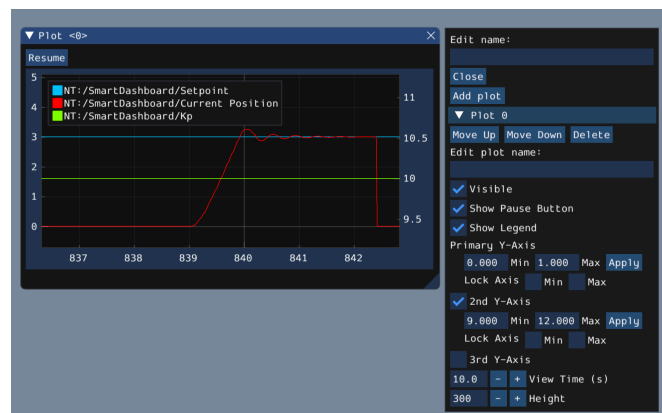
You can click and drag on the plot to move around and scroll on top of the plot to zoom the y axes in and out. Double clicking on the graph will autoscale it so that the zoom and axis limits fit all of the data it is plotting. Furthermore, right-clicking on the plot will present you with a plethora of 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

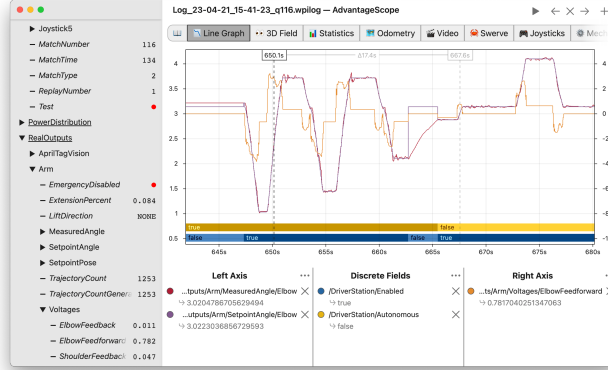
AdvantageScope is a data visualization tool for *NetworkTables*, *WPILib data logs*, and *Driver Station logs*. It is a programmer's tool (rather than a competition dashboard) and can be used to debug real or simulated robot code from a log file or live over the network.

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.

Not: 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



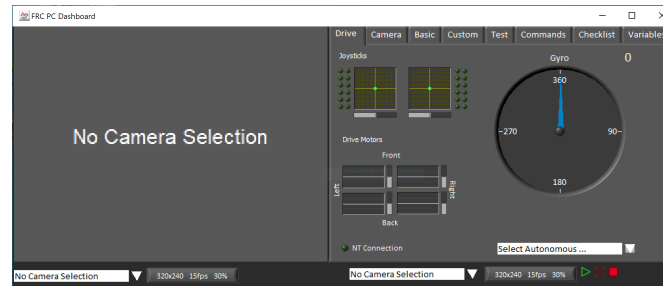
11.6 LabVIEW Dashboard

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 Kontrol Paneli

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 Kontrol Paneli



Gösterge Panosu iki ana bölüme ayrılmıştır. Sol bölme, bir kamera görüntüsünü görüntülemek içindir. Sağ bölme şunları içerir:

- Kumanda kolu ve sürücü motor değerleri için göstergeler (LabVIEW robot kodu ile kullanıldığında varsayılan olarak bağlanır), bir cayro göstergesi, bir Otonom seçim metin kutusu, bir bağlantı göstergesi ve kamera için bazı kontroller ve göstergeler içeren sürücü sekmesi
- Bazı varsayılan kontrolleri ve göstergeleri içeren temel sekme

- Sol bölmedeki izleyiciye benzer ikincil bir kamera görüntüleyici içeren kamera sekmesi
- Kontrol panelini LabVIEW kullanarak özelleştirmek için özel sekme
- LabVIEW çerçevesinde Test Modu ile kullanım için Test sekmesi
- Yeni LabVIEW C&C Framework ile kullanım için Komutlar sekmesi
- Eşleşmelerden önce ve / veya eşleşmeler arasında tamamlamak üzere görev listeleri oluşturmak için kullanılabilen kontrol listesi sekmesi
- Variables tab that displays the raw NetworkTables variables in a tree view format

LabVIEW Dashboard ayrıca sağ altta bulunan Kayıt / Oynatma işlevini de içerir. Bu özellik ile ilgili daha fazla ayrıntı aşağıda 'Record/Playback - Kayıt / Oynatma' altında verilmiştir.

Kamera Görüntüsü ve Kontrolleri

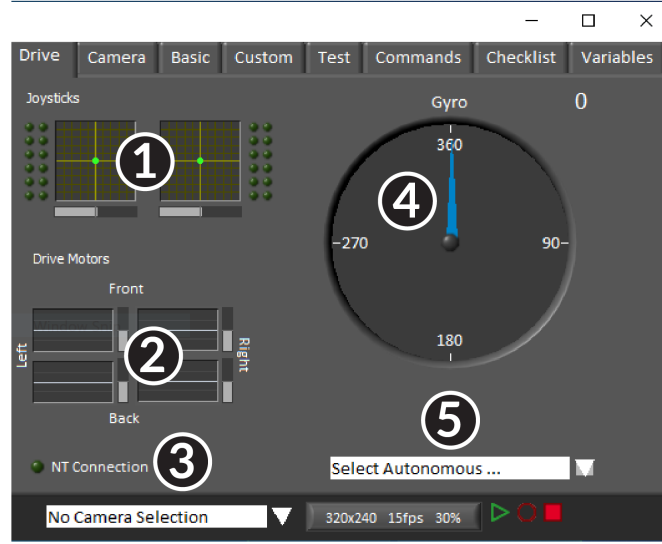


The left pane is used to display a video feed from a camera located on the robot. There are also some controls and indicators related to the camera below the tab area:

1. Kamera Görüntü Ekranı
2. Mode Selector-Mod Seçici - Bu açılır menü, kullanılacak kamera ekranı türünü seçmenize olanak tanır. Seçenekler, Kamera Kapalı, USB Kamera Yazılımı (yazılım sıkıştırma), USB Kamera HW (donanım sıkıştırması) ve IP Kamera (Eksen kamera) şeklindedir. Bilgisayarınız USB üzerinden roboRIO'ya bağlandığında IP Kamera ayarının çalışmayacağını unutmayın.
3. Camera Settings-Kamera Ayarları - Bu kontrol, kontrol panelindeki görüntü akışının çözünürlüğünü, kare hızını ve sıkıştırmasını değiştirmenize olanak tanır, yapılandırmayı açmak için kontrole tıklayın.
4. Bant Genişliği Göstergesi - Görüntü akışının yaklaşık bant genişliği kullanımını gösterir. Gösterge, "safe-güvenli" bant genişliği kullanımı için yeşil, takımların dikkatli kullanması gerektiğinde sarı, akış bant genişliği yarışma alanında çalışacak seviyelerin ötesinde ise kırmızı renkte görüntülenecektir.
5. Framerate-Kare Hızı - Görüntü akışının yaklaşık alınan kare hızını gösterir.

Tüyo: Bant genişliği göstergesi, açık olan tüm kamera akışları için birleşik bant genişliğini gösterir.

Sürüş



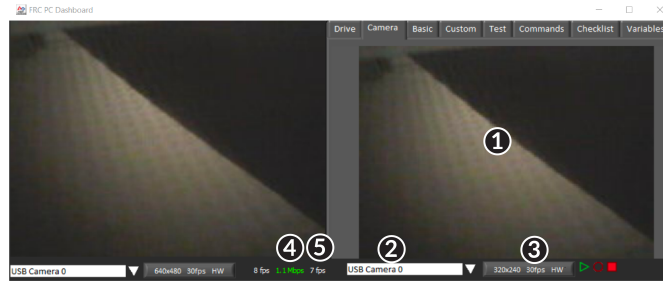
Orta bölme, LabVIEW çerçevesiyle kullanıldığında kumanda çubukları ve sürücü komutları hakkında geri bildirim sağlayan bir bölüm ve NetworkTables durumunu ve otonom seçiciyi görüntüleyen bir bölüm içerir:

1. LabVIEW çerçevesini kullanırken 2 adede kadar kumanda kolu için X, Y ve Gaz Kelebeği bilgilerini ve düğme değerlerini görüntüler
2. LabVIEW çerçevesi kullanılırken motor kontrolörlerine gönderilen değerleri görüntüler
3. Displays a connection indicator for the NetworkTables data from the robot
4. Bir Gyro değeri görüntüler
5. Otonom modları seçmek için kullanılacak bir metin kutusu görüntüler. Her dilin kod şablonlarında, birden çok özerk program arasından seçim yapmak için bu kutuyu kullanma örnekleri vardır.

Bu göstergeler (Gyro dışında) LabVIEW çerçevesi kullanılırken varsayılan olarak uygun değerlere bağlanır. Bunları C ++ / Java koduyla kullanma hakkında bilgi için bkz: *doc: using-the-labview-dashboard-with-c ++ - java-code*.

Kamera

Tüyo: Sol bölme yalnızca tek bir kamera çıkışı görüntüleyebilir, bu nedenle gerekirse ikinci bir kamera çıkışı görüntülemek için sağ bölmedeki kamera sekmesini kullanın.

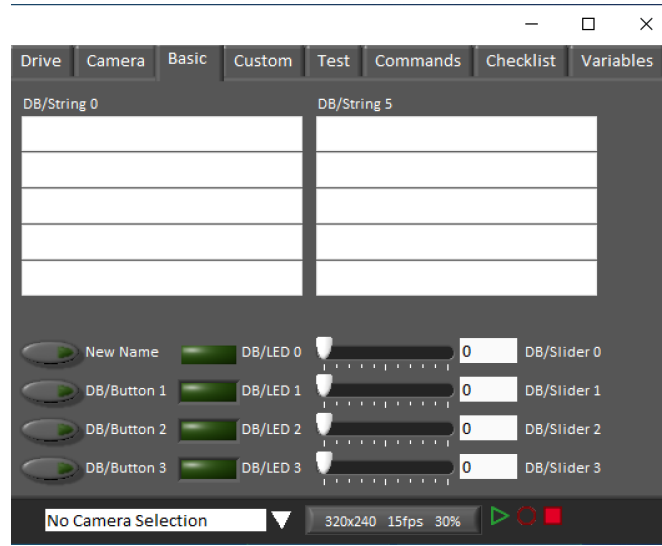


The camera tab is used to display a video feed from a camera located on the robot. There are also some controls and indicators related to the camera below the tab area:

1. Kamera Görüntü Ekranı
2. Mode Selector-Mod Seçici - Bu açılır menü, kullanılacak kamera ekranı türünü seçmenize olanak tanır. Seçenekler, Kamera Kapalı, USB Kamera Yazılımı (yazılım sıkıştırma), USB Kamera HW (donanım sıkıştırması) ve IP Kamera (Eksen kamera) şeklindedir. Bilgisayarınız USB üzerinden roboRIO'ya bağlandığında IP Kamera ayarının çalışmayacağını unutmayın.
3. Camera Settings-Kamera Ayarları - Bu kontrol, kontrol panelindeki görüntü akışının çözünürlüğünü, kare hızını ve sıkıştırmasını değiştirmenize olanak tanır, yapılandırmayı açmak için kontrole tıklayın.
4. Bant Genişliği Göstergesi - Görüntü akışının yaklaşık bant genişliği kullanımını gösterir. Gösterge, "safe-güvenli" bant genişliği kullanımı için yeşil, takımların dikkatli kullanması gerektiğinde sarı, akış bant genişliği yarışma alanında çalışacak seviyelerin ötesinde ise kırmızı renkte görüntülenecektir.
5. Framerate-Kare Hızı - Görüntü akışının yaklaşık alınan kare hızını gösterir.

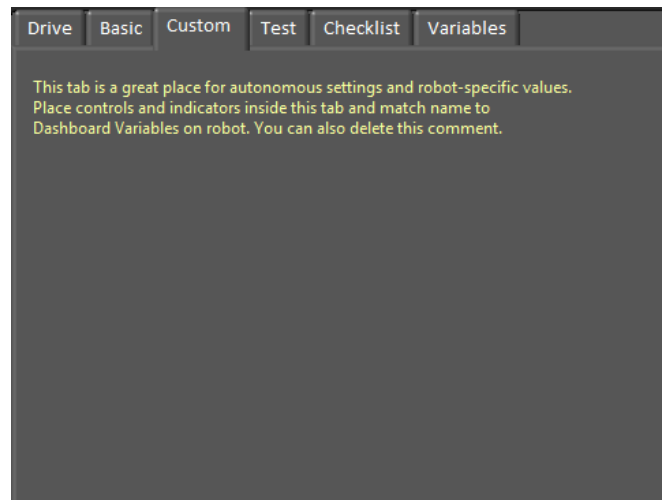
Tüyo: Bant genişliği göstergesi, açık olan tüm kamera akışları için birleşik bant genişliğini gösterir.

Temel



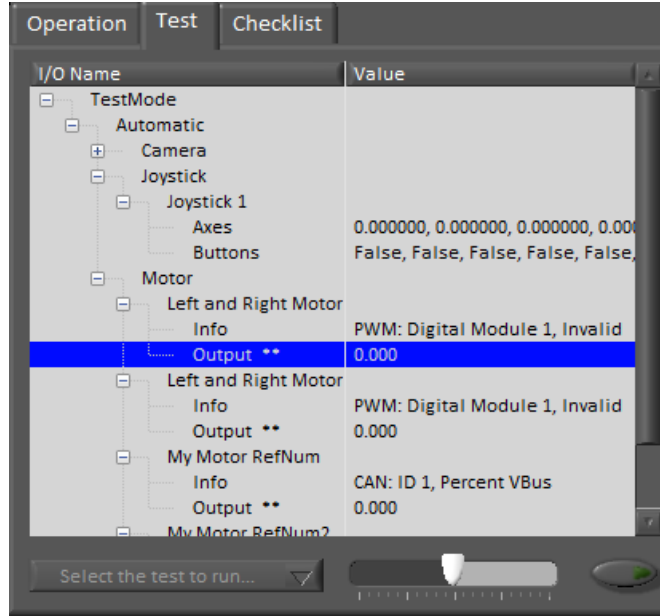
Temel sekmesi, robotu kontrol etmek veya robottan bilgileri görüntülemek için kullanılabilen önceden doldurulmuş çeşitli çift yönlü kontroller / göstergeler içerir. Her bir öğeyle ilişkili SmartDashboard anahtar adları, aynı adlandırma modelini izleyen ve soldaki DB / String 0'dan DB / String 4'e ve DB / String 5'ten DB / String'e kadar olan Dizeler haricinde göstergenin yanında etiketlenir. Sağda 9. LabVIEW çerçevesi, Teleop'taki Düğmeler ve Kaydırıcılardan bir okuma örneği içerir. Ayrıca, Başlangıçta etiketlerin özelleştirilmesine ilişkin bir örnek içerir. Bu sekmeyi C ++ Java koduyla kullanma hakkında daha fazla ayrıntı için, bakınız: doc: *using-the-labview-dashboard-with-c ++ -java-code*.

Custom-Özel



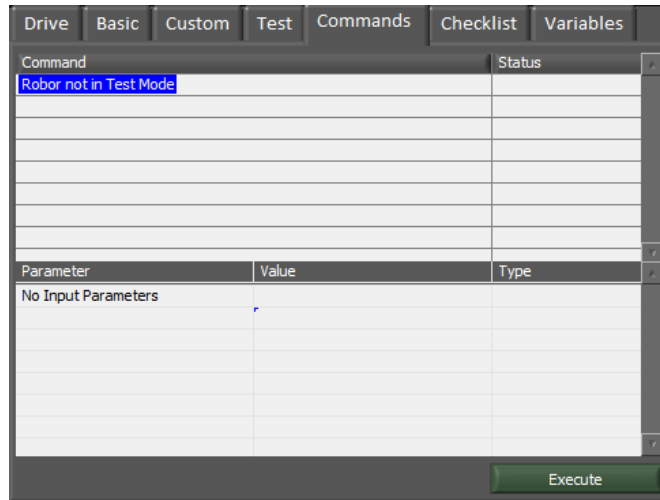
Özel sekmesi, mevcut herhangi bir işlevi kaldırmadan LabVIEW kullanarak kontrol paneline ek kontroller / göstergeler eklemenizi sağlar. Bu sekmeyi özelleştirmek için LabVIEW'de bir Gösterge Tablosu projesi oluşturmanız gerekir.

Test-Ölçek



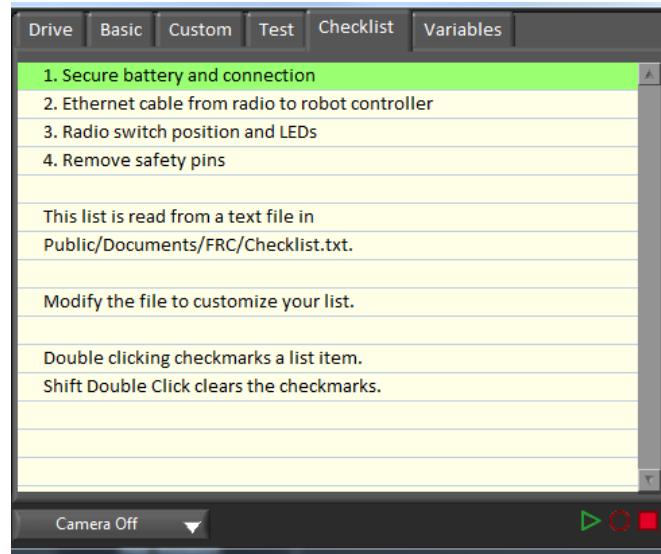
Test sekmesi, LabVIEW kullanan ekipler için Test moduyla kullanım içindir (Java ve C++ ekipleri Test Modunu kullanırken SmartDashboard veya Shuffleboard kullanmalıdır). Kitaplıklardaki birçok öge için Giriş / Çıkış bilgileri burada otomatik olarak doldurulacaktır. Yanlarında ** bulunan tüm öğeler, kontrol paneli tarafından kontrol edilebilen çıktılardır. Bir çıkışı kontrol etmek için, üzerine tıklayarak seçin, değeri ayarlamak için kaydırıcıyı sürükleyin ve ardından çıkışı etkinleştirmek için yeşil düğmeye basılı tutun. Yeşil düğme serbest bırakılır bırakılmaz, çıktı devre dışı bırakılacaktır. Bu sekme, robot üzerinde testleri çalıştırmak ve izlemek için de kullanılabilir. LabVIEW çerçevesinde örnek bir test sağlanmıştır. Açılır kutudan bu testin seçilmesi, kaydırıcı yerine testin durumunu gösterecek ve kontrolleri etkinleştirecektir.

Commands-Komutlar



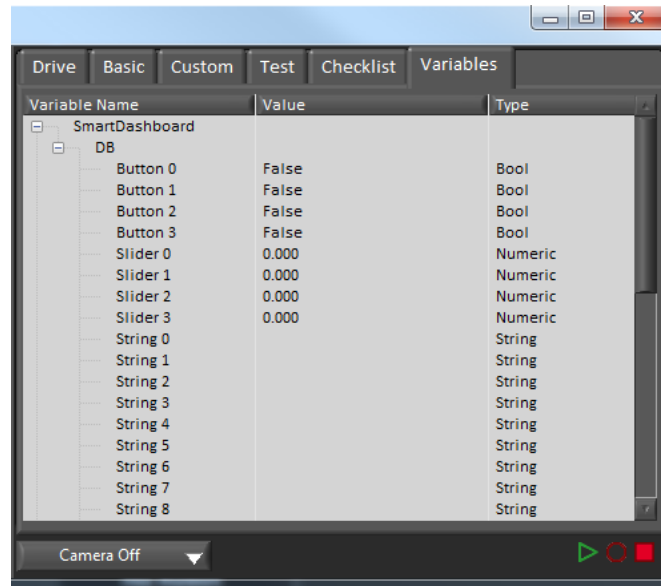
Komutlar sekmesi, hangi komutların çalıştığını görmek ve test amacıyla komutları manuel olarak çalıştırmak için Robot Test modundayken kullanılabilir.

Checklist-Kontrol listesi



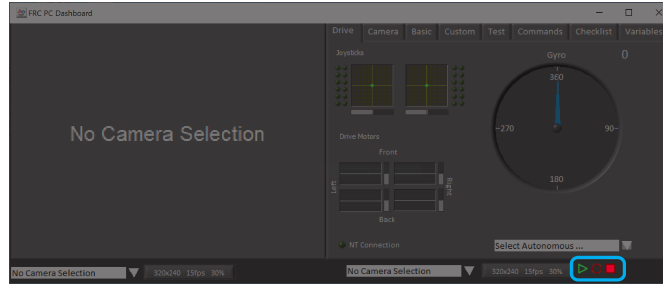
Kontrol listesi sekmesi, takımlar tarafından maçlardan önce veya maçlar arasında gerçekleştirilecek bir görev listesi oluşturmak için kullanılabilir. Kontrol Listesi sekmesini kullanma talimatları, varsayılan kontrol listesi dosyasında önceden doldurulmuştur.

Variables-Değişkenler



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-Kayıt/Oynatma



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-Kayıt



Kayda başlamak için kırmızı dairesel Kayıt düğmesini tıklayın. Sağ bölmenin arka planı kayıt yaptığınızı belirtmek için kırmızıya dönecektir. Kaydı durdurmak için kırmızı kare Durdur düğmesine basın.

Playback-Geri çalma



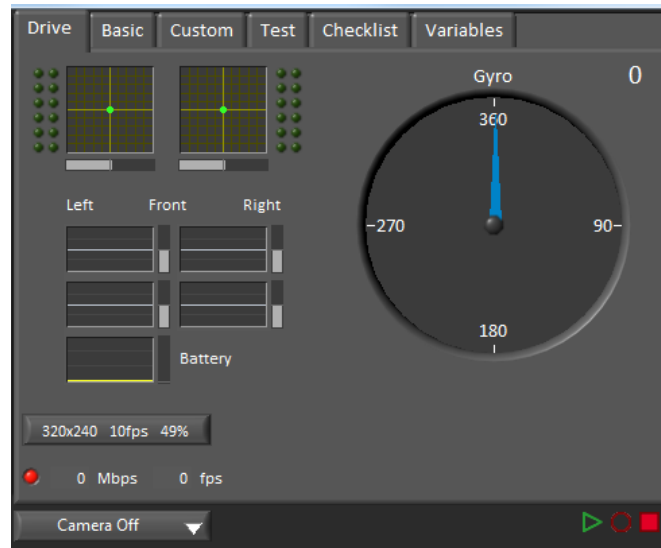
Bir kaydı oynatmak için yeşil üçgen Oynat düğmesine tıklayın. Sağ bölmenin arka planı yeşil renkte yanıp sönmeye başlayacak ve kamera bölmesinin altında kayıttan yürütme kontrolleri görünecektir.

1. Dosya Seçici - Açılır menü, oynatılacak bir günlük dosyası seçmenize olanak tanır. Günlük dosyaları, tarih ve saat kullanılarak adlandırılır ve açılır menü ayrıca dosyanın uzunluğunu gösterir. Bir günlük dosyası seçmek, o dosyayı hemen oynatmaya başlayacaktır.
2. Oynat / Duraklat düğmesi - Bu düğme, günlük dosyasının oynatılmasını duraklatmanıza ve devam ettirmenize olanak tanır.
3. Oynatma Hızı - Bu açılır menü, oynatma hızını 1/10 hızdan 10x hıza ayarlamanıza olanak tanır, varsayılan gerçek zamanlıdır (1x)
4. Zaman Kontrol Kaydırıcısı - Bu kaydırıcı, istenen konuma tıklayarak veya kaydırıcıyı sürükleyerek günlük dosyasında hızlı ileri veya geri sarmanıza olanak tanır.
5. Ayarlar - Bir günlük dosyası seçiliyken, bu açılır menü bir dosyayı yeniden adlandırmanıza veya silmenize veya günlükleri içeren klasörü Windows Gezgini'nde (Typically C:\Users\Public\Documents\FRC\Log Files\Dashboard) açmanıza olanak tanır.

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 to 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++

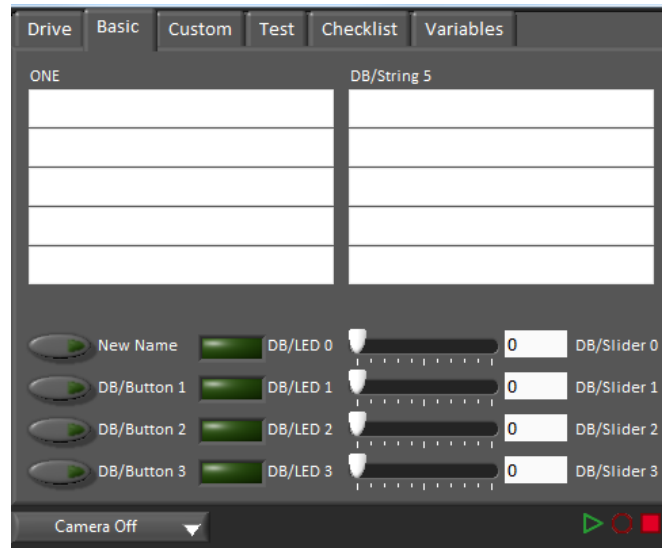
```
frc::SmartDashboard::PutNumberArray("Gyro", {drivetrain.GetLeftFront(), drivetrain.  
↳GetRightFront(), drivetrain.GetLeftBack(), drivetrain.GetRightBack()});
```

PYTHON

```
from wpilib import SmartDashboard

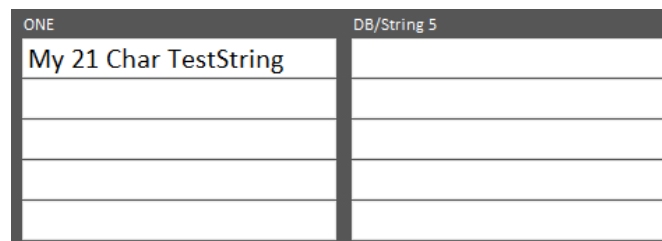
SmartDashboard.putNumberArray("RobotDrive Motors", [self.drivetrain.getLeftFront(),  
↳self.drivetrain.getRightFront(), self.drivetrain.getLeftBack(), self.drivetrain.  
↳getRightBack()])
```

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);
```

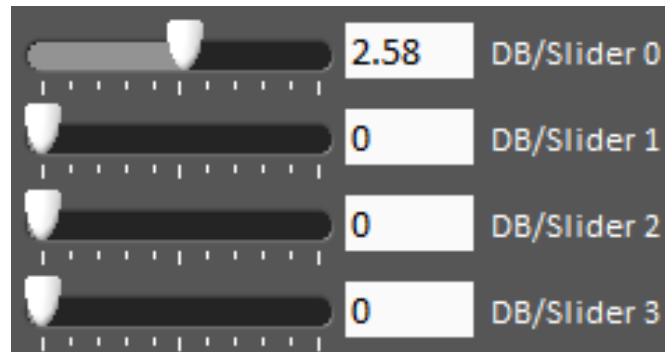
C++

```
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

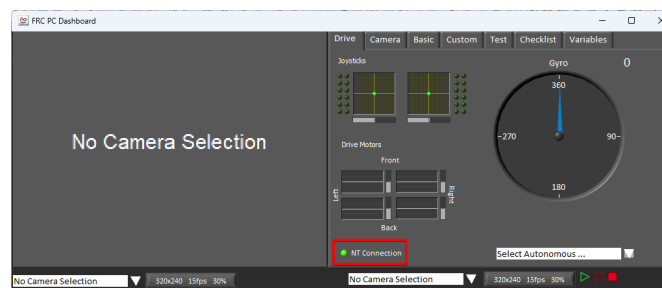
```
from wpilib import SmartDashboard

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

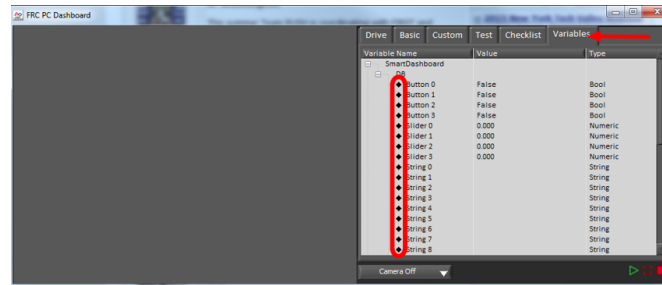
11.6.3 Troubleshooting Dashboard Connectivity

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

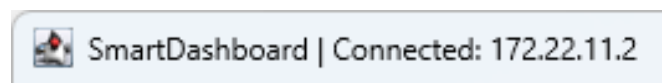


The LabVIEW Dashboard has a NetworkTables Connection indicator on the front panel.

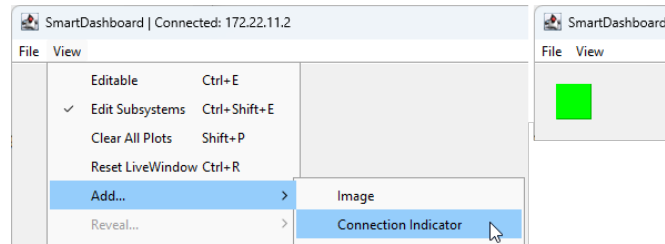


On the Variables tab of the Dashboard, the variables are shown with a black diamond when they are not synced with the robot. Once the Dashboard connects to the robot and these variables are synced, the diamond will disappear.

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

NetworkTables: not connected

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 - Connected (127.0.0.1)

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

172.22.11.2 — AdvantageScope

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

12.1 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.

Not: 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

Not: 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 `DataLog-Reader` 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()`).

Not: 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.

Not: 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

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

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

C++

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

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

PYTHON

```
from wpilib import DataLogManager

DataLogManager.start()
```

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

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

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

Logging Joystick Data

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

JAVA

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

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

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

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

C++

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

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

// Record both DS control and joystick data
DriverStation::StartDataLog(DataLogManager::GetLog());

// (alternatively) Record only DS control data
DriverStation::StartDataLog(DataLogManager::GetLog(), false);
```

PYTHON

```
from wpilib import DataLogManager, DriverStation

# Starts recording to data log
DataLogManager.start()

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

# (alternatively) Record only DS control data
DriverStation.startDataLog(DataLogManager.getLog(), False)
```

12.3.3 Custom Data Logging using DataLog

The DataLog class (Java, C++, Python) and its associated LogEntry classes (e.g. BooleanLogEntry, DoubleLogEntry, etc) provides low-level access for writing data logs.

Not: Unlike NetworkTables, there is no change checking performed. **Every** call to a LogEntry.append() function will result in a record being written to the data log. Checking for changes and only appending to the log when necessary is the responsibility of the caller.

The LogEntry classes can be used in conjunction with DataLogManager to record values only to a data log and not to NetworkTables:

JAVA

```
import edu.wpi.first.util.datalog.BooleanLogEntry;
import edu.wpi.first.util.datalog.DataLog;
import edu.wpi.first.util.datalog.DoubleLogEntry;
import edu.wpi.first.util.datalog.StringLogEntry;
import edu.wpi.first.wpilibj.DataLogManager;
```

```
BooleanLogEntry myBooleanLog;
DoubleLogEntry myDoubleLog;
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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

```
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

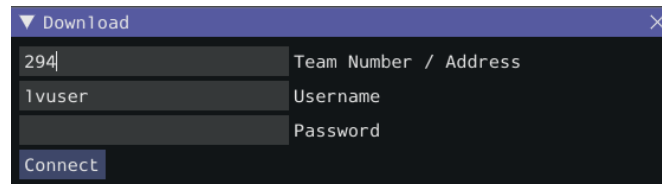
The DataLogTool desktop application integrates a SFTP client for downloading data log files from a network device (e.g. roboRIO or coprocessor) to the local computer.

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

Not: 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.

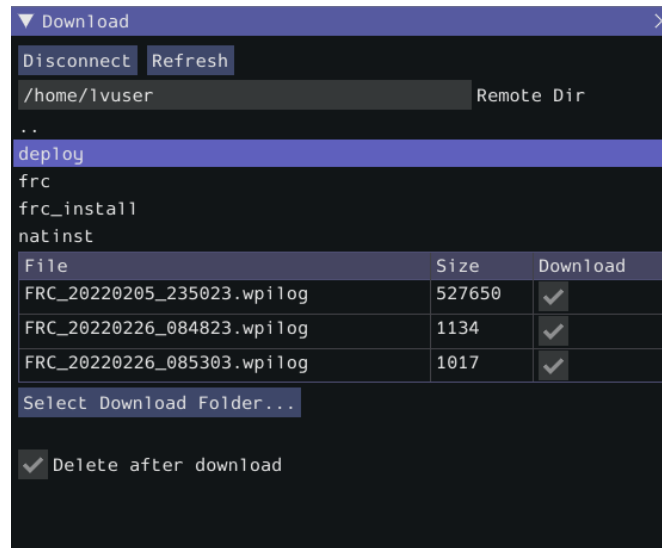
The tool also supports connecting to network devices other than the roboRIO, such as coprocessors, as long as the device supports SFTP password-based authentication.

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.

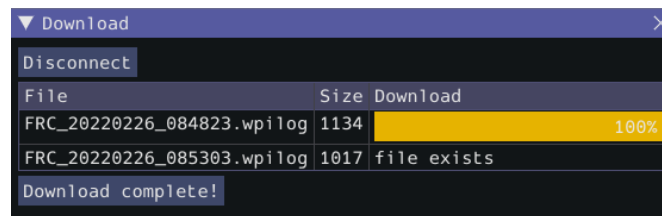
Not: On the roboRIO, log files are typically saved to either /home/lvuser/logs or /u/logs (USB stick location).



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

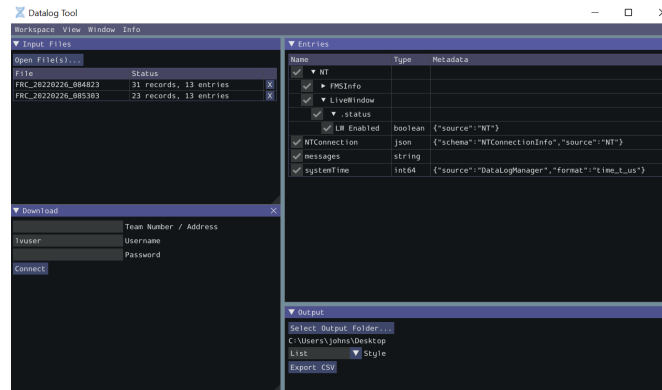
If you want to delete the files from the remote device after they are downloaded, check the *Delete after download* checkbox.

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.

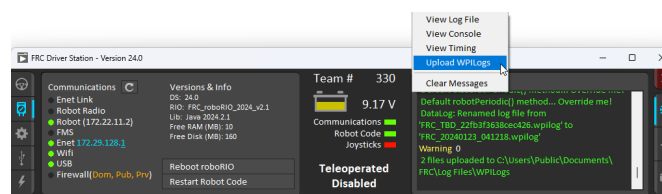


The conversion process is started by opening data log files in the “Input Files” window. Files are opened by clicking *Open File(s)...* Summary status on each file (e.g. number of records and entries) is displayed. Clicking X in the table row closes the file.

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(
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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

Tüyo: 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.

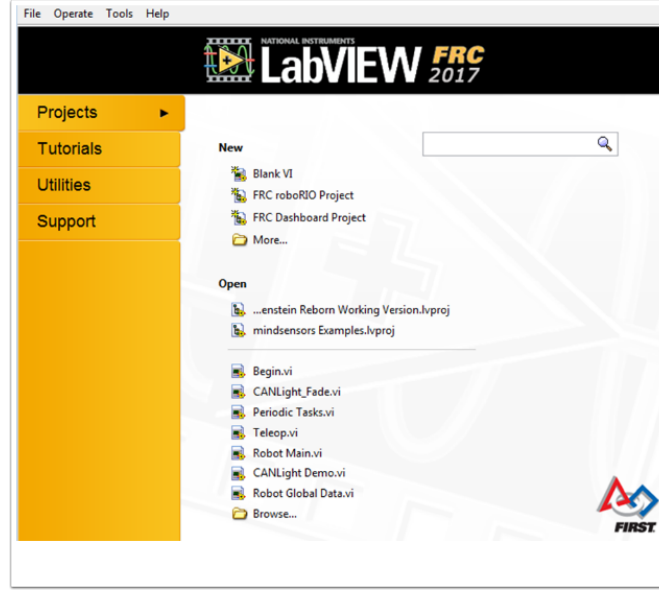
13.1 Robot Programları Oluşturma

13.1.1 Tank Sürüşü Rehberi

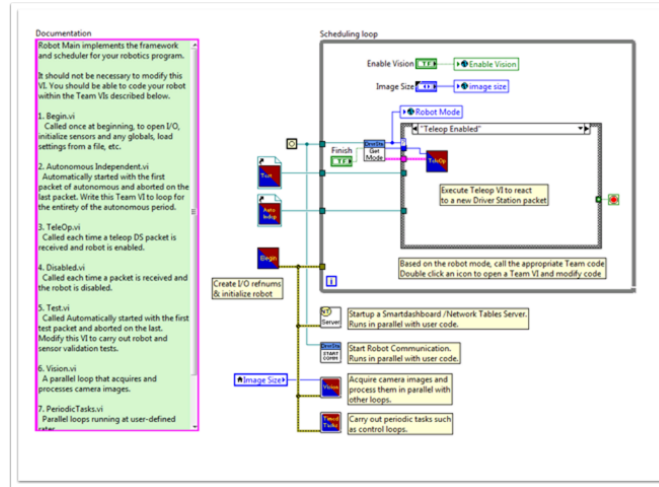
Soru: Robotumun tank sürücüyü kullanarak iki kumanda kolu ile sürmesini nasıl sağlayabilirim?

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. LabVIEW'i açın ve `FRC roboRIO Project` e çift tıklayın.

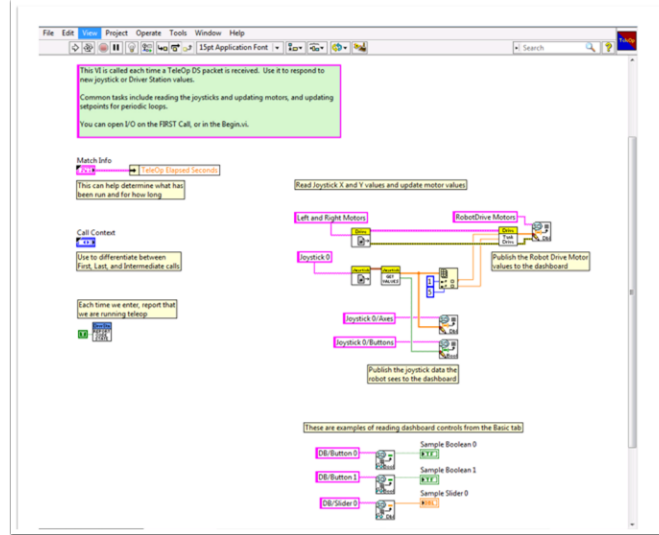


2. Projenize bir isim verin, takım numaranızı ekleyin ve Arcade Drive Robot roboRIO'yu seçin. Başka bir seçenek seçebilirsiniz, ancak bu eğitimde bu proje için tank sürücüsünün nasıl ayarlanacağı anlatılacaktır.
3. Project Explorer penceresinde, Robot Main.vi'yi açın.
4. Blok şemasını görmek için: kbd: **Ctrl + E** ye basın. Aşağıdaki resme benzemelidir:



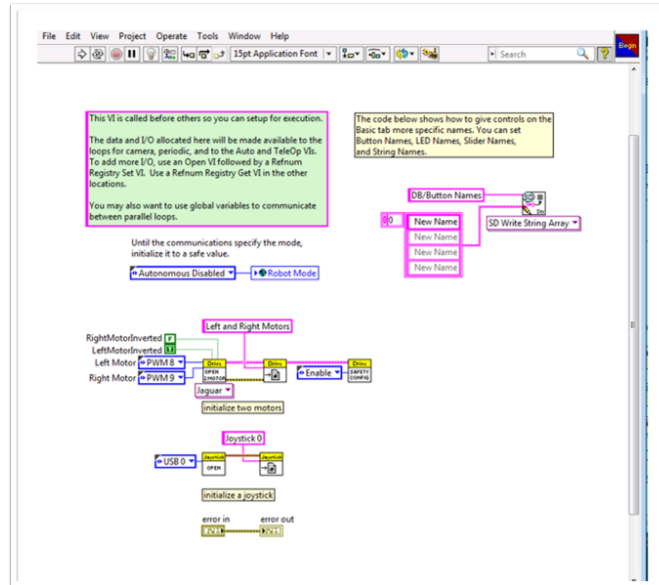
5. Teleop Etkin vaka yapısının içindeki "Teleop" vi ögesine çift tıklayın. Blok şemasına bakın. Burada iki değişiklik yapmak isteyeceksiniz:
- Arcade Drive'ı tank drive.vi ile değiştirin. Bu, blok diyagram >> WPI Robotics Library >> Robot Drive>> üzerine sağ tıklayıp Tank Drive VI'ya tıklayarak bulunabilir.
 - Get Values.vi 'den sonraki Index Array fonksiyonunu bulun. İki sayısal sabit oluşturmanız ve her birini dizin girişlerinden birine bağlamanız gerekecektir. FRC® Sürücü İstasyonundaki USB Aygıtları sekmesine bakarak her bir dizinin değerinin ne olması gerektiğini belirleyebilirsiniz. Hangi numaraya (indeks) bağlı olduklarını belirlemek için iki kumanda çubuğunu hareket ettirin. Her bir kumanda kolu için Y eksenini dizinini kullanmak isteyebilirsiniz. Bunun nedeni, motorların ileri gitmesini istediğinizde joystick'i yukarı, geri gittiğinizde aşağı doğru itmenin sezgisel olmasıdır. Her biri için X eksenini

seçerseniz, robot motorlarının hareket etmesini sağlamak için kumanda çubuğunu sola veya sağa (x eksen yönleri) hareket ettirmeniz gerekir. Kurulumumda, sol motorların Y eksen kontrolü için dizin 1'i ve sağ motorlar Y eksen kontrolü olarak dizin 5'i seçtim. LabVIEW'deki ayarlamaları aşağıdaki resimde görebilirsiniz:



6. Daha sonra "Robot Main.vi"nize geri dönmek ve "Begin.vi." üzerine çift tıklamak isteyeceksiniz.
7. Bu VI'da onaylanması gereken ilk şey, sol ve sağ motorlarınızın LabVIEW'deki PDP'nizdeki (Güç Dağıtım Paneli) ile aynı PWM hatlarına bağlı olmasıdır.
8. Bu VI'da teyit edilecek ikinci şey, "Open 2 Motor.vi"nin doğru motor kontrolörünün seçilmiş olmasıdır (Talon, Jaguar, Victor, vb.).

Örneğin, Jaguar motor denetleyicileri kullanıyorum ve motorlarım PWM 8 ve 9'a bağlı. Aşağıdaki resim, yapmam gereken değişiklikleri gösteriyor:



9. Ayarlamalar yaptığınız tüm Vis'i kaydedin ve artık tank sürücülü bir robotu sürebilirsiniz!

13.1.2 Command ve Control Eğitimi

Giriş

Command and Control, robota özgü alt sistemlerden oluşan bir koleksiyon için robot kodunu komutlar ve denetleyiciler halinde düzenleyen 2016 sezonu için eklenen yeni bir LabVIEW şablonudur. Her bir alt sistem, mekanizma için uygun hızda çalışan bağımsız bir kontrol döngüsüne veya state machine ve istenen işlemleri ve ayar noktalarını güncelleyen yüksek seviyeli komutlara sahiptir. Bu, otonom kodun eşzamanlı komut dizileri oluşturmasını çok kolaylaştırır. Bu arada TeleOp, aynı komutları tamamlanmayı beklemeye gerek kalmadan kullanabildiği ve sürücü ekibinin girdisine göre yeni komutların kolay iptaline ve başlatılmasına izin verdiği için avantaj sağlar. Her alt sistemde, zaman içindeki sensör ve kontrol değerlerini gösteren bir panel ve hata ayıklamaya yardımcı olmak için komut izleme bulunur.

Command ve Control nedir ?

Command and Control recognizes that FRC® robots tend to be built up of relatively independent mechanisms such as Drive, Shooter, Arm, etc. Each of these is referred to as a subsystem and needs code that will coordinate the various sensors and actuators of the subsystem in order to complete requested commands, or actions, such as “Close Gripper” or “Lower Arm”. One of the key principles of this framework is that subsystems will each have an independent controller loop that is solely responsible for updating motors and other actuators. Code outside of the subsystem controller can issue commands which may change the robot’s output, but should not directly change any outputs. The difference is very subtle but this means that outputs can only possibly be updated from one location in the project. This speeds up debugging a robot behaving unexpectedly by giving you the ability to look through a list of commands sent to the subsystem rather than searching your project for where an output may have been modified. It also becomes easier to add an additional sensor, change gearing, or disable a mechanism without needing to modify code outside of the controller.

Öncelikle Otonom ve TeleOp’tan oluşan oyun kodunun tipik olarak ayar noktalarını güncelleme ve belirli mekanizmaların durumuna tepki vermesi gerekecektir. Autonomous için, robotun çalışmasını bir dizi işlem olarak tanımlamak çok yaygındır - buraya sürün, alın, oraya taşıyın, ateş edin, vb. Komutlar, hızlı bir şekilde karmaşık rutinler oluşturmak için ek mantıkla sırayla bağlanabilir. TeleOp için, aynı komutlar eşzamanlı olarak yürütülebilir, robotun her zaman en son sürücü girdilerini işlemesine izin verir ve uygun şekilde uygulanırsa yeni komutlar kesintiye uğrayarak, sürücü ekibinin saha koşullarına hızlı bir şekilde yanıt verirken aynı zamanda otomatikleştirilmiş komutlardan ve komutlardan dizilerinden yararlanmasını sağlar..

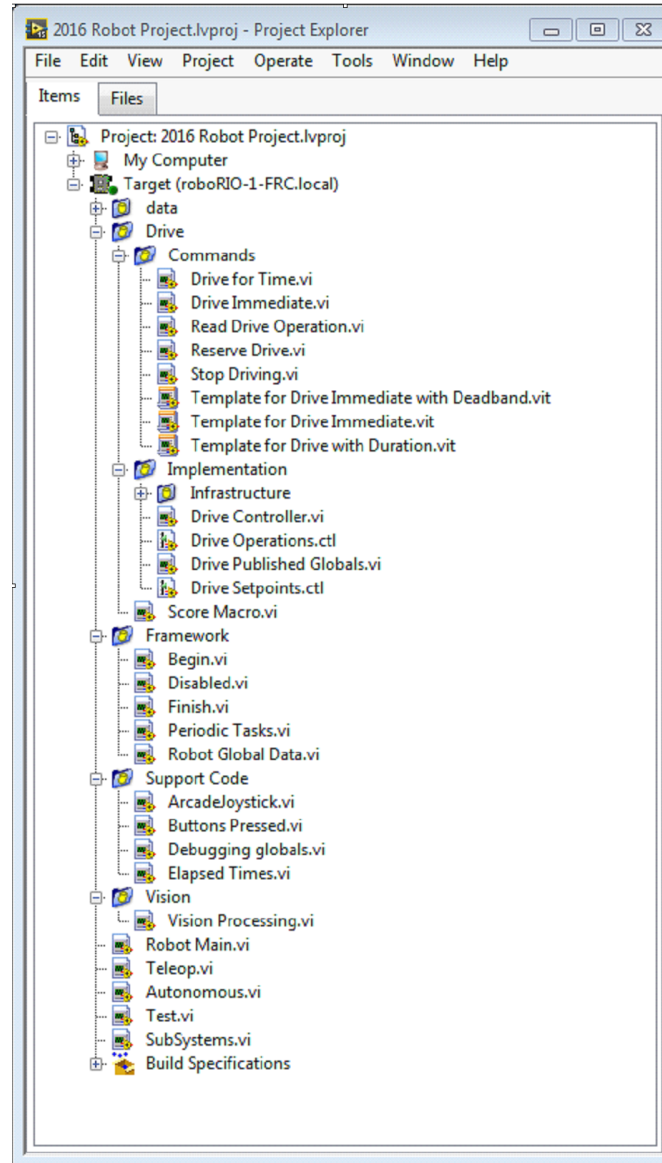
Neden Command ve Control kullanmalıyım?

Command ve Control, mevcut LabVIEW proje şablonlarına işlevsellik ekleyerek kodun daha karmaşık robotlar ve robot koduyla daha iyi ölçeklenmesini sağlar. Alt sistemler, uygulamanın ayrıntılarını soyutlamak için kullanılır ve oyun kodu, yüksek seviyeli komut VI dizilerinden oluşturulur. Komutların kendileri, ayar noktalarını güncelleyebilen, mühendislik birimleri ve mekanizma birimleri arasında sayısal ölçeklendirme / haritalama gerçekleştirebilen ve senkronizasyon seçenekleri sunan VI’lardır. Robotta dişli oranının değiştirilmesi gibi fiziksel değişiklikler yapılırsa, bu değişikliği kod tabanının tamamında yansıtmak için yalnızca birkaç Vis komutunda değişiklikler yapılabilir.

I/O kapsülleme, daha öngörülebilir işlem ve kaynak çakışmaları meydana geldiğinde daha hızlı hata ayıklama sağlar. Her bir komut bir VI olduğu için, komutlar arasında tek adım atabilir veya her bir alt sisteme gönderilen tüm komutların bir listesini görüntülemek için yerleşik Trace işlevini kullanabilirsiniz. Çerçeve, eşzamansız bildirim ve tutarlı veri yayılımı kullanır, bu da bir diziyi programlamayı kolaylaştırır çalıştırılacak doğru komutu belirlemek için basit bir mantık ekleyin.

Part 1: Proje Gezgini

Proje Gezgini, robot sisteminiz için kullanacağınız tüm Vis ve dosyalar için organizasyon sağlar. Aşağıda, sistemimizin genişletilmesine yardımcı olmak için Proje Gezgini'ndeki ana bileşenlerin bir açıklaması bulunmaktadır. En sık kullanılan öğeler kalın olarak işaretlenmiştir.



My Computer

Projenin yüklendiği bilgisayardaki işlemi tanımlayan öğeler. Bir robot projesi için, bu bir simülasyon hedefi olarak kullanılır ve simülasyon dosyalarıyla doldurulur.

Sim Destek Dosyaları

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

Robot Simulation Readme.html

Documents the *PWM* channels and robot info you will need in order to write robot code that matches the wiring of the simulated robot.

Dependencies-Bağımlılıklar

Simüle edilmiş robotun kodu tarafından kullanılan dosyaları gösterir. Simüle edilmiş robot hedefi için kod belirlediğinizde bu doldurulacaktır.

Derleme Özellikleri

Bu, simüle edilmiş robot hedefi için kodun nasıl oluşturulacağını ve dağıtılacağını tanımlayan dosyaları içerecektir.

Target (roboRIO-TEAM-FRC.local)

(Adres) konumunda bulunan roboRIO'da operasyonu tanımlayan öğeler.

Drive - Sürüş

Robot sürücü tabanı için alt sistem uygulaması ve komutları. Bu, WPILib RobotDrive VI'lar için özel bir yedek görevi görür.

Framework-Çerçeve

Çok sık kullanılmayan bir alt sistemin parçası olmayan robot kodu için kullanılan VI'lar.

Begin

Robot kodu ilk başladığında bir kez çağrılır. Bu, belirli bir alt sisteme ait olmayan başlatma kodu için kullanışlıdır.

Disabled

Devre dışı bırakılan her paket için bir kez çağrılır ve robotun hareket etmesini istemediğinizde sensörlerde hata ayıklamak için kullanılabilir.

Finish

Geliştirme sırasında bu, robot kodu bittiğinde çağrılabilir. Durdurulduğunda veya güç kapatıldığında çağrılmaz.

Periodic Tasks

Hata ayıklama veya izleme için ad hoc periyodik döngüler için iyi bir yer

Robot Global Data

Bir alt sisteme ait olmayan robot bilgilerini paylaşmak için kullanışlıdır.

Destek Kodu

Hata ayıklama ve kod geliştirme yardımcıları.

Vision

Kamera ve görüntü işleme için alt sistem ve komutlar.

Robot Main.vi

Kod geliştirirken çalıştıracağınız en üst seviye VI.

Autonomous.vi

Otonom periyodu boyunca çalışan VI.

Teleop.vi

Her TeleOp paketi için çağrılan VI.

Test.vi

Driver Station test modundayken çalışan VI.

SubSystems.vi

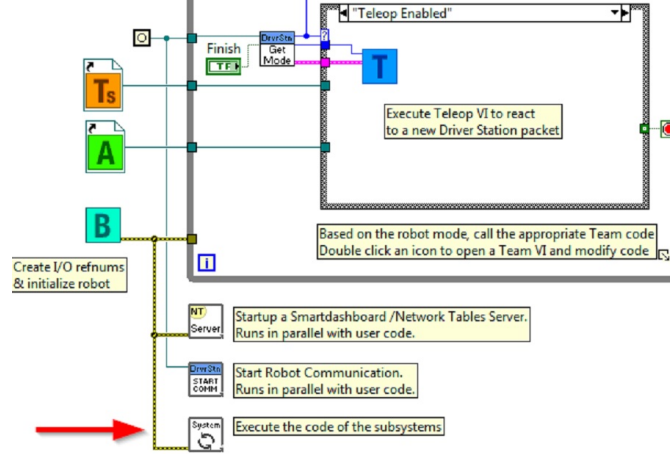
Tüm alt sistemleri içeren ve başlatan VI.

Dependencies-Bağımlılıklar

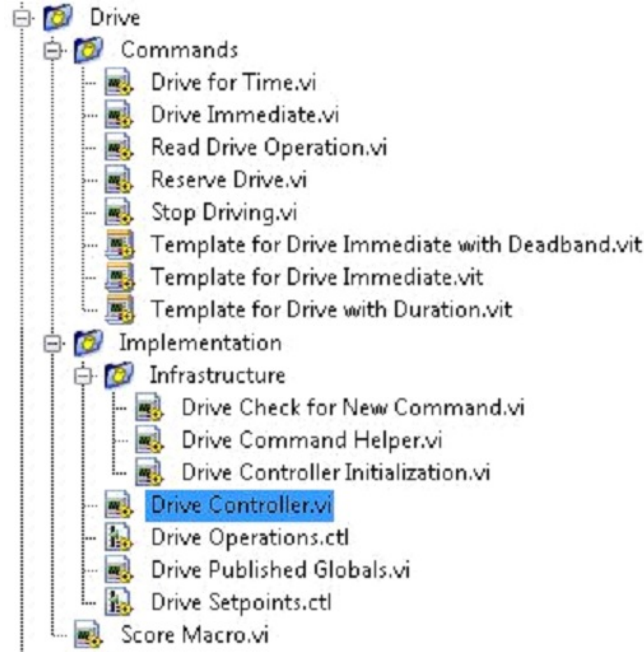
Robot kodu tarafından kullanılan dosyaları gösterir.

Derleme Özellikleri

Kod doğru çalıştıktan sonra kodu bir başlangıç uygulaması olarak oluşturmak ve çalıştırmak için kullanılır.



Drive Subsystem Project Explorer



Commands:

Bu klasör, denetleyiciden bir işlem gerçekleştirmesini isteyen komut VI'ları içerir. Ayrıca, ek sürücü komutları oluşturmak için şablonlar içerir.

Not: Yeni bir komut oluşturduktan sonra, denetleyicinin yeni işlemi tanımlamak için kullandığı alanları eklemek veya güncellemek için Drive Setpoints.cti dosyasını düzenlemeniz gerekebilir. Ayrıca, her değer için bir durum eklemek için Drive Controller.vi'ye gitmeniz ve

durum yapısını değiştirmeniz gerekir.

Uygulama

Bunlar, alt sistemi oluşturmak için kullanılan VI'lar ve Kontrollerdir.

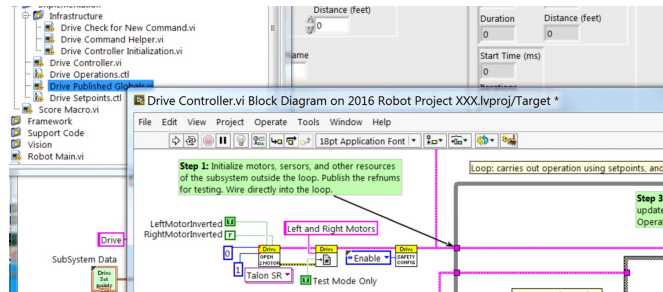
Altyapı VI ları

- Drive Check for New Command: Denetleyici döngüsünün her yinelenmesi olarak adlandırılır. Yeni komutları kontrol eder, zamanlama verilerini günceller ve tamamlandığında bir bekleme komutunu bildirir.
- Drive Command Helper.vi: Komutlar, yeni bir komutun verildiğini kontrolöre bildirmek için bu VI'ya çağırır.
- Drive Controller Initialization.vi: Bildiriciyi tahsis eder ve zamanlamayı, varsayılan komutu ve diğer bilgileri tek bir veri kablosunda birleştirir.
- Drive Controller.vi: Bu VI, control/state machine döngüsünü içerir. Panel ayrıca hata ayıklama için yararlı ekranlar içerebilir.
- Drive Operation.ctl: Bu typedef, denetleyicinin çalışma modlarını tanımlar. Birçok komut bir işlemi paylaşabilir.
- Drive Setpoint.ctl: Drive alt sisteminin tüm çalışma modları tarafından kullanılan veri alanlarını içerir.
- Drive Published Globals.vi: Sürücü alt sistemi hakkında genel bilgileri yayınlamak için yararlı bir yer.

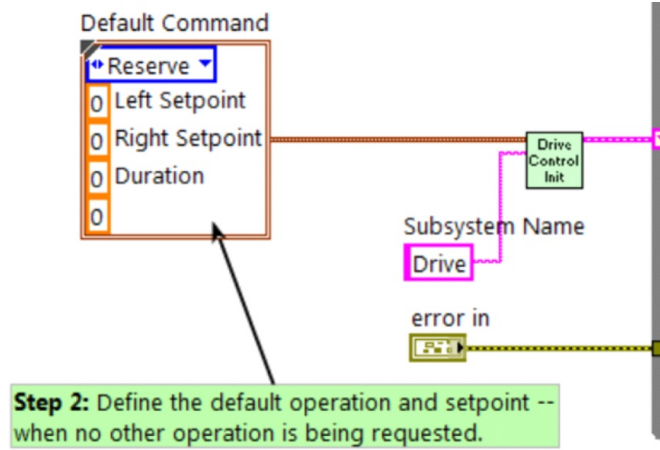
Bölüm 2: Drive Subsystem-Sürücü Alt Sistemini Başlatma

Denetleyicinin blok diyagramında, nasıl düzenleneceğini bilmek isteyeceğiniz önemli alanları gösteren yeşil yorumlar vardır.

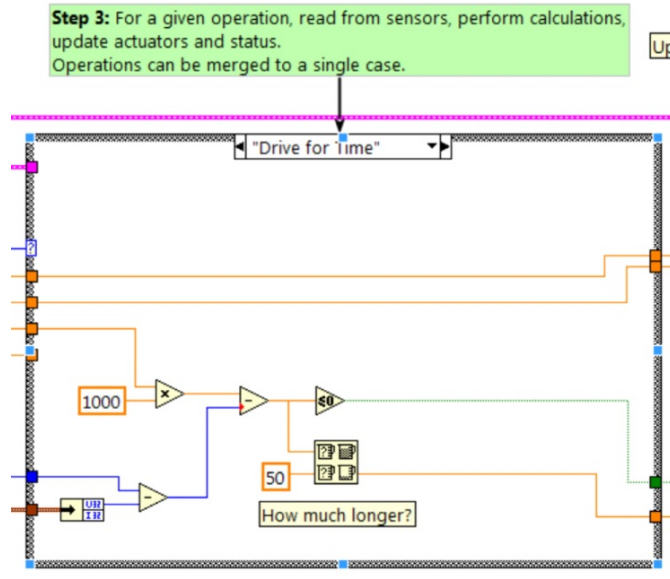
Kontrol döngüsünün solundaki alan, alt sistem başladığında bir kez çalıştırılacaktır. Burası, genellikle tüm I/O ve durum verilerini tahsis edeceğiniz ve başlatacağınız yerdir. I/O referanslarını yayınlatabilirsiniz veya onları yalnızca Test Modu'na kaydederek diğer kodların bir komut kullanmadan motorları güncelleyememesi için onları gizli tutabilirsiniz.



Not: Her bir alt sistem için kaynakları Begin.vi yerine ilgili Controller.vi'de başlatmak, I/O kapsüllemeyi geliştirir, olası kaynak çakışmalarını azaltır ve hata ayıklamayı basitleştirir.

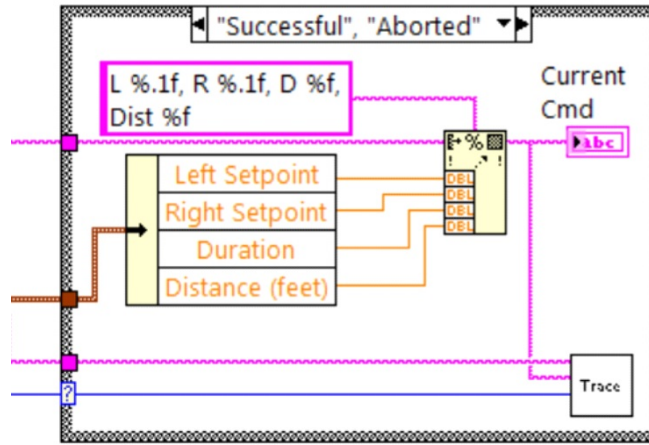


Başlatma işleminin bir kısmı, başka hiçbir işlem işlenmediğinde varsayılan işlemi ve ayar noktası değerlerini seçmektir.



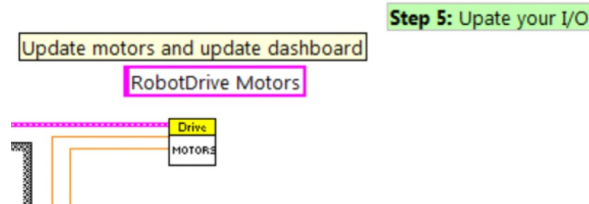
Kontrol döngüsünün içinde, işlemlerin gerçekte uygulandığı bir durum ifadesi bulunur. Ayar noktası değerleri, yinleme gecikmesi, yinleme sayısı ve sensörlerin tümü, alt sistemin nasıl çalıştığını etkileyebilir. Bu case yapısı, alt sistemin her çalışma durumu için bir değere sahiptir.

Step 4: Format a string to describe this cmd and how the previous cmd finished



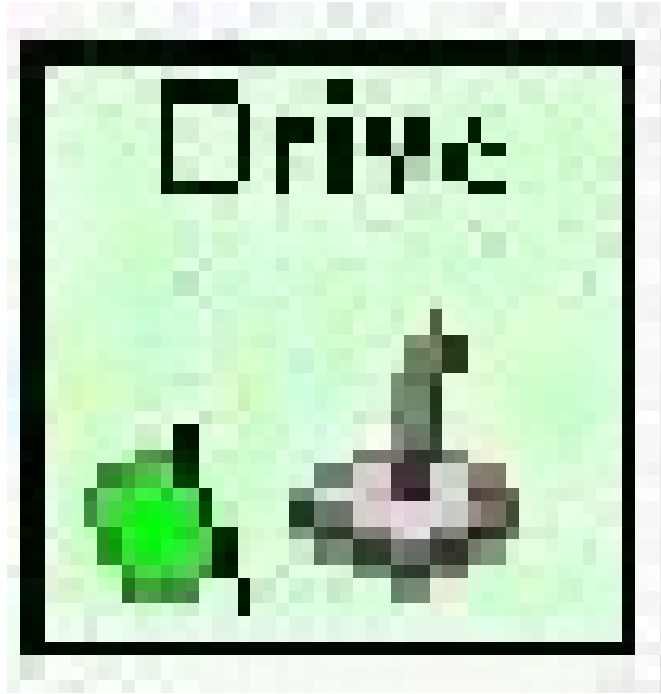
Denetleyici döngüsünün her yinelenmesi isteğe bağlı olarak Trace VI'yı güncelleyecektir. Çerçeve zaten alt sistem adını, çalışmasını ve açıklamasını içerir ve izleme bilgilerine ek ayar noktası değerlerini biçimlendirmeyi yararlı bulabilirsiniz. Trace VI'yı açın ve robot kodu her bir alt sisteme gönderilen mevcut ayar noktaları ve komutlarda çalışırken Enable'e tıklayın.

Kontrolörün birincil amacı, alt sistem için aktüatörleri güncellemektir. Bu, case yapısı içinde meydana gelebilir, ancak çoğu zaman, değerlerin her zaman doğru değerle ve kodda yalnızca bir konumda güncellendiğinden emin olmak için bunu yapının aşağı yönünde yapmak yararlıdır.



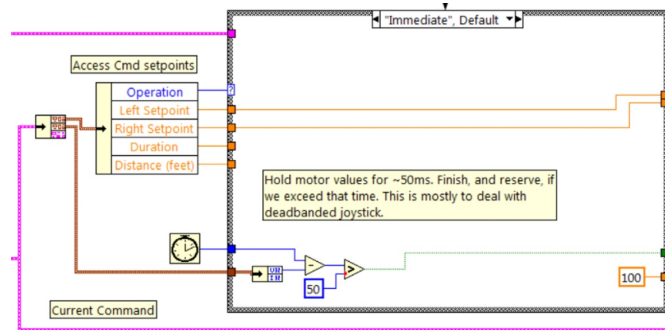
Bölüm 3: Drive Subsystem Taşınan Komutlar

Her yeni alt sistem için gönderilen 3 örnek komut vardır:

Drive Immediate.vi

Motorlar için istenen sol ve sağ hızları alır ve motorları hemen bu ayar noktalarına ayarlar.

Immediate durumu, motorları komutla tanımlanan ayar noktasına günceller. Yeni bir komut gelene kadar veya bir zaman aşımı değerine kadar motorların bu değeri korumasını istediğiniz için komut bitmiş sayılmaz. Zaman aşımı, bir komutun ölü bant içerdiği her zaman yararlıdır. Ölü banttan daha küçükse küçük değerler istenmez ve komut zaman aşımına uğramadıkça hırıltı veya sürünmeye neden olur.

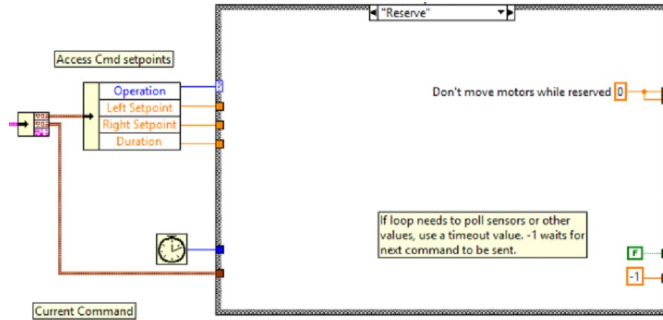


Stop Driving.vi



Robotu sabit hale getirerek tahrik motorlarını sıfırlayın.

Reserve komutu motorları kapatır ve yeni bir komut bekler. Reserve, adlandırılmış bir komut dizisi ile kullanıldığında, robotu hareket ettirmiyor olsa bile sürücü alt sisteminin bir dizinin parçası olduğunu tanımlar. Bu, aynı anda çalışan komutlar arasında alt sistem kaynağında arabuluculuk yapmaya yardımcı olur.



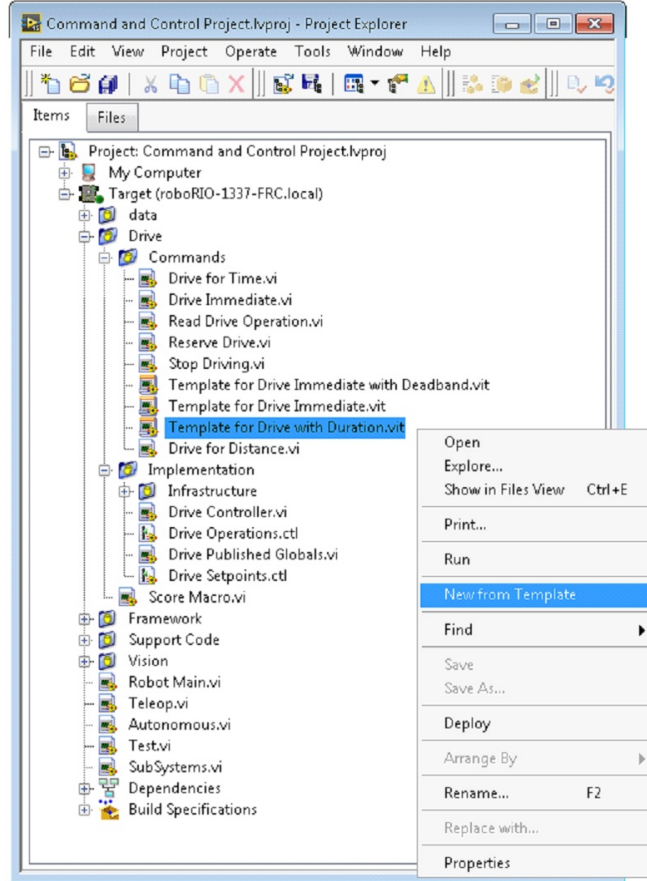
Part 4: Yeni Komutlar Yaratmak

Komut ve Kontrol çerçevesi, kullanıcıların bir alt sistem için kolayca yeni komutlar oluşturmaya olanak tanır. Yeni bir komut oluşturmak için alt sistem klasörünü açın / Commands project explorer penceresinde, yeni komutunuzun başlangıç noktası olarak kullanmak için VI Şablonlarından birini seçin, sağ tıklayın ve New From Template öğesini seçin.

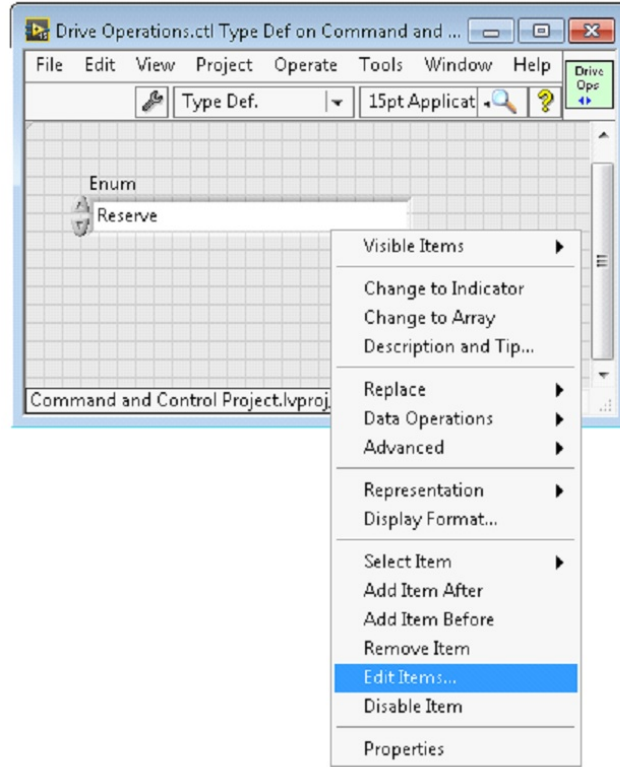
- **Immediate-Hemen:** Bu VI, alt sistemi yeni ayar noktası hakkında bilgilendirir.
- **Immediate with deadband :** Bu VI, giriş değerini ölü bantla karşılaştırır ve isteğe bağlı olarak alt sistemi yeni ayar noktası hakkında bilgilendirir. Bu, joystick devamlı değerleri kullanıldığında çok kullanışlıdır.

- **With duration:** This VI notifies the subsystem to perform this command for the given duration, and then return to the default state. Synchronization determines whether this VI Starts the operation and returns immediately, or waits for the operation to complete. The first option is commonly used for TeleOp, and the second for Autonomous sequencing.

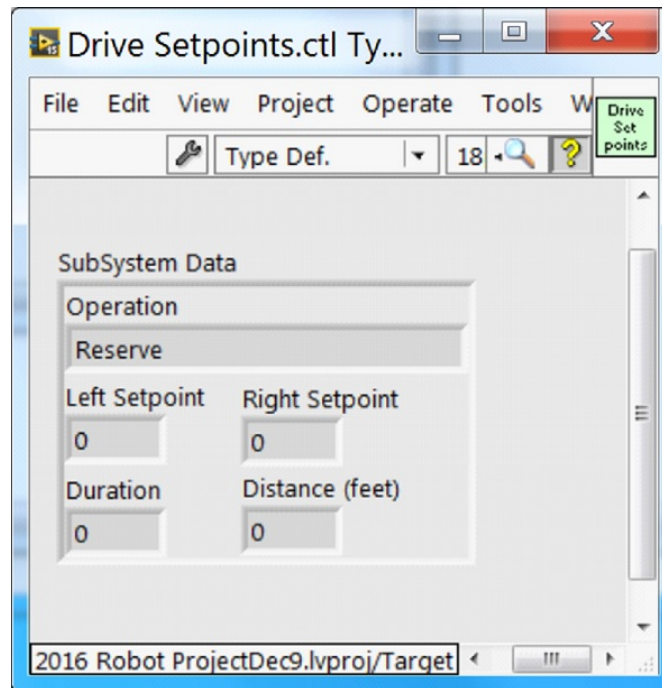
Bu örnekte yeni bir komutu ekleyeceğiz: “Mesafe için Sür”.



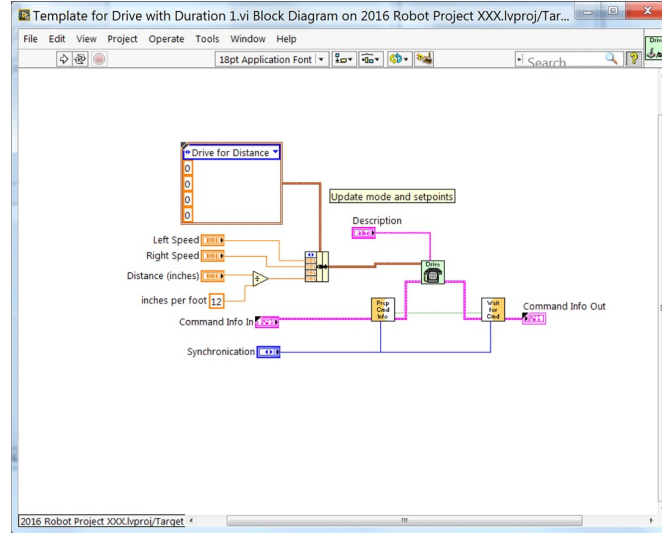
First, save the new VI with a descriptive name such as “Drive for Distance”. Next, determine whether the new command needs a new value added the Drive Operations enum typedef. The initial project code already has an enum value of Drive for Distance, but the following image shows how you would add one if needed.



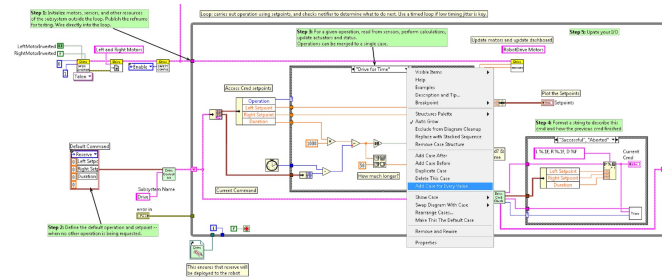
Bir komutun yürütülmesi için ek bilgiye ihtiyaç duyarsa, bunu ayar setpoint kontrolüne ekleyin. Varsayılan olarak, Sürücü alt sistemi, yürütülecek işlemle birlikte Sol Ayar Noktası-Setpoint, Sağ Ayar Noktası-Setpoint ve Duration-süre için alanlara sahiptir. Drive for Distance komutu, Süreyi-Duration mesafe olarak yeniden kullanabilirdi, ancak devam edelim ve Drive Setpoints.ctl'ye Distance (feet) adı verilen sayısal bir kontrol ekleyelim.



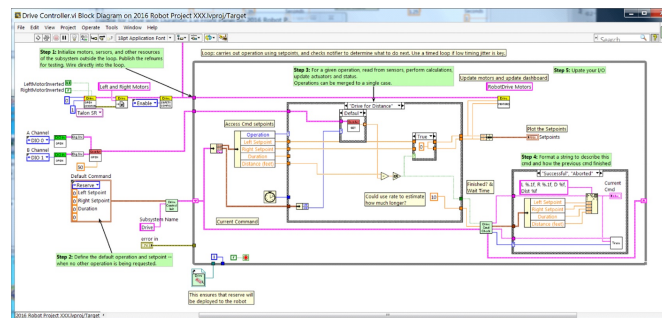
Bundan sonra, komutumuzu belirtmek için gereken tüm alanlara sahip olduğumuzda, yeni oluşturulan Drive for Distance.vi ı değiştirebiliriz. Aşağıda gösterildiği gibi, enum açılır menüsünden Drive for Distance ögesini seçin ve mesafeyi, hızları vb. belirtmek için bir VI parametresi ekleyin. Birimler uyuşmuyorsa, VI komutu birimler arasında haritalamak için harika bir yerdirdir.



Next, add code to the Drive Controller to define what happens when the Drive for Distance command executes. Right click on the Case Structure and Duplicate or Add Case for Every Value. This will create a new “Drive for Distance” case.

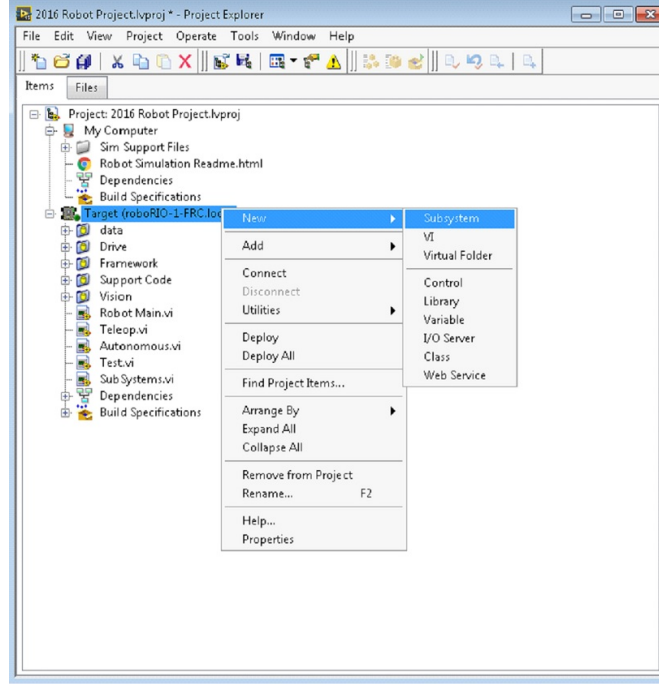


In order to access new setpoint fields, grow the “Access Cmd setpoints” unbundle node. Open your encoder(s) on the outside, to the left of the loop. In the new diagram of the case structure, we added a call to reset the encoder on the first loop iteration and read it otherwise. There is also some simple code that compares encoder values and updates the motor power. If new controls are added to the setpoints cluster, you should also consider adding them to the Trace. The necessary changes are shown in the image below.

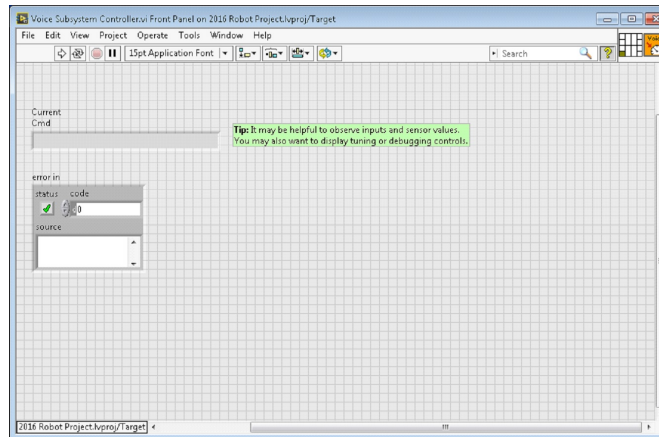


Bölüm 5: Bir Alt Sistem Oluşturma

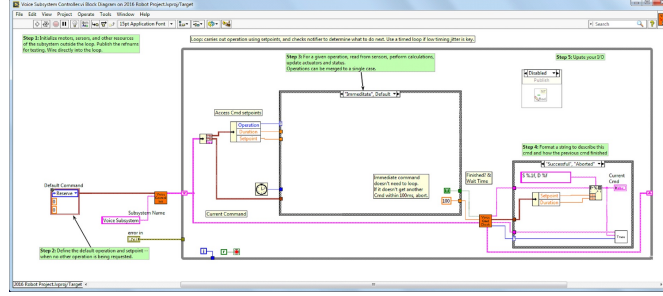
Yeni bir alt sistem oluşturmak için roboRIO hedefine sağ tıklayın ve New» Subsystem Sistem ögesini seçin. Açılan iletişim kutusunda, alt sistemin adını girin, işletim modlarını listeleyn ve simgenin rengini belirtin.



OK'e tıkladığınızda, alt sistem klasörü oluşturulacak ve proje disk klasörüne ve ağacına eklenecektir. Bir alt sistemi oluşturan VI'ların ve kontrollerin temel bir uygulamasını içerecektir. Yeni denetleyiciye bir çağrı, Alt Sistemler VI'ya eklenecektir. Controller VI açılacak, I/O eklemeniz ve state-machine veya kontrol kodunu uygulamanız için hazır olacaktır. Oluşturulan VI simgeleri, iletişim kutusunda sağlanan rengi ve adı kullanacaktır. Oluşturulan kod, ayar noktası alanları ve işlemleri için typedef'leri kullanacaktır.



Aşağıda yeni oluşturulan alt sistemin blok şeması bulunmaktadır. Bu kod, alt sistemi oluşturduğunuzda otomatik olarak üretilecektir.



13.2 LabVIEW Kaynakları

13.2.1 LabVIEW Kaynakları

Not: LabVIEW’de programlama ve özellikle FRC ® programlama hakkında daha fazla bilgi edinmek için LabVIEW’deki robotlar için aşağıdaki kaynaklara göz atın.

LabVIEW Temelleri

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 Eğitimleri

NI ayrıca, **temelden gelişmişe kadar değişen birçok FRC'ye özel eğitim ve sunuma**<https://forums.ni.com/t5/FIRST-Robotics-Competition/Archived-2015-FRC-LabVIEW-Additional-Resources/ta-p/3528790?profile.language=en> ev sahipliği yapıyor. Derinlemesine tek bir kaynak için, sayfanın altına yakın bir yerde bulunan FRC Basic and Advanced Training Classes’a bakın.

Yüklü Eğitimler ve Örnekler

Ayrıca, LabVIEW kurulumunuzun bir parçası olarak sağlanan her türlü görev ve bileşen için eğitimler ve örnekler de vardır. Öğreticilere erişmek için, LabVIEW Açılış ekranından (program ilk başlatıldığında görünen ekran) sol taraftaki **Tutorials** sekmesine tıklayın. Öğreticilerin hepsinin tek bir belgede olduğunu unutmayın, bu nedenle açıldıktan sonra açılış ekranına geri dönmekten diğer eğitimlere göz atmakta özgürsünüz.

Örneklerle erişmek için ya **Support** sekmesine, ardından **Find FRC Examples** a tıklayın veya bir program üzerinde çalışırken **Help** menüsünü açın, **Find Examples** u seçin ve **FRC Robotics** klasörünü açın .

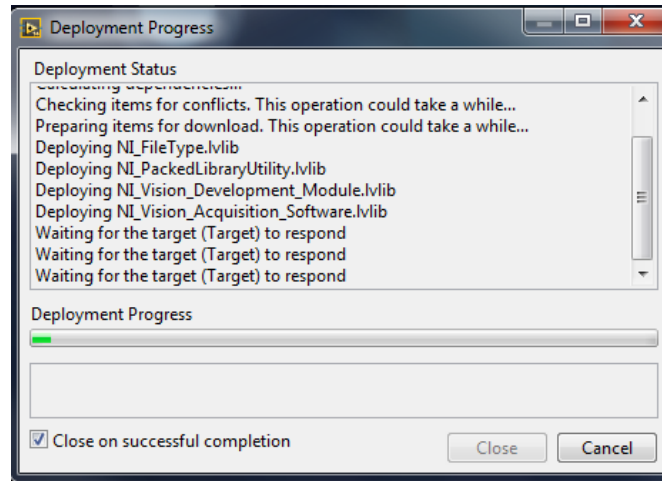
Third Party Resources

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

13.2.2 Waiting for Target to Respond - Bozuk döngülerden kurtarma

Not: 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.

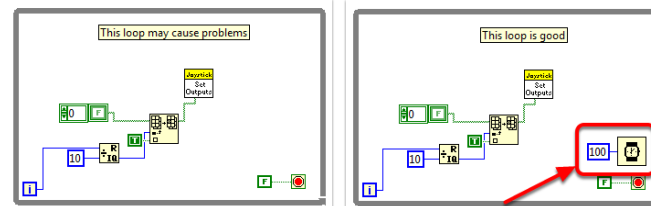
Belirti



Bu sorunun birincil belirtisi, yukarıda gösterildiği gibi “Waiting for the target (Target) to respond” adımıyla askıda kalan yeni robot kodunu indirme girişimleridir. Bu semptomun başka olası nedenleri de olduğunu unutmayın (bir C++Java programından LabVIEW programına geçmek gibi), ancak burada açıklanan adımlar bunların çoğunu veya tümünü çözecektir.

İndirme iletişim kutusunu kapatmak için Cancel a tıklayın.

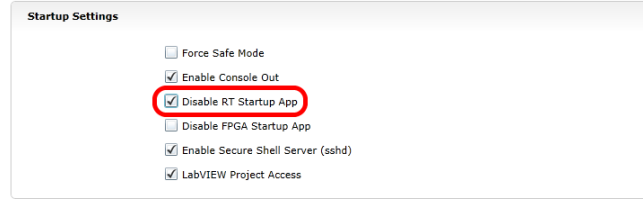
Sorun



Bu sorunun yaygın bir kaynağı, LabVIEW kodunuzdaki kısıtlanmamış döngülerdir. Sınırlanmamış döngü, herhangi bir gecikme ögesi içermeyen bir döngüdür (soldaki gibi). Nereden

bakmaya başlayacağınızdan emin değilseniz, Disabled.VI, Periodic Tasks.VI ve Vision Processing.VI, bu tür döngü için ortak konumlardır. Kodla ilgili sorunu çözmek için, sağ döngüde bulunan Timing paletinden Wait(ms) VI gibi bir gecikme ögesi ekleyin.

Set No App - Uygulama Yok Ayarla

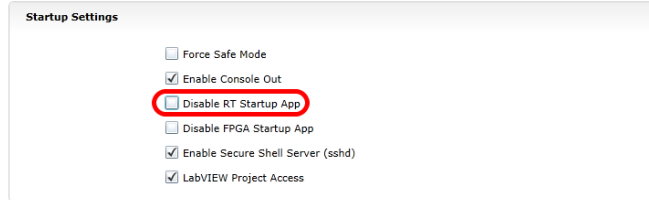


RoboRIO web sunucusunu kullanma (daha fazla ayrıntı için [roboRIO Web Dashboard Startup Settings](#) deki makaleye bakın). “RT Başlangıç Uygulamasını Devre Dışı Bırak” kutusunu **Check** işaretleyin .

Reboot -Yeniden başlat

Cihazdaki Reset düğmesini kullanarak veya web sayfasının sağ üst köşesindeki Restart seçeneğine tıklayarak roboRIO’yu yeniden başlatın.

Uygulama Yok’u Temizle



RoboRIO web sunucusunu kullanma (daha fazla ayrıntı için: [roboRIO Web Dashboard Startup Settings](#) ’deki makaleye bakın). “RT Başlangıç Uygulamasını Devre Dışı Bırak” kutusunun **Uncheck** işaretini kaldırın.

LabVIEW Kodunu Yükle

LabVIEW kodunu yükleyin (Run düğmesini veya Run as Startup’ı kullanarak). RoboRIO’yu yeniden başlatmadan önce LabVIEW kodunu Run as Startup olarak ayarladığınızdan emin olun, aksi takdirde yukarıdaki talimatları tekrar izlemeniz gerekecektir.

13.2.3 İki Kamera Modu Arasında Nasıl Geçiş Yapılır

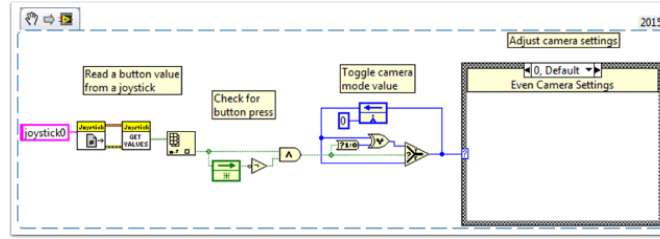
Bu kod, iki farklı kamera modu arasında geçiş yapmak için bir butonun nasıl kullanılacağını gösterir. Kod dört aşamadan oluşur.

İlk aşamada joystick üzerindeki bir butonun değeri okunur.

Daha sonra, mevcut okuma bir **Feedback Node** ve bazı Boolean aritmetiği kullanılarak önceki okuma ile karşılaştırılır. Bunlar birlikte, kamera modunun, buton basılı tutulurken birden çok kez ileri ve geri gitmek yerine, yalnızca butona ilk basıldığında değiştirilmesini sağlar.

Bundan sonra, ikinci aşamanın sonucunu mevcut kamera modu değeri üzerinden maskeleyerek kamera modu açılır. Buna bit masking -bit maskeleme denir ve bunu **XOR** işlevi ile yaparak, kod ikinci aşama doğruya döndüğünde kamera modunu değiştirir ve başka türlü bir şey yapmaz.

Son olarak, her kamera modu için kodu en sondaki şart yapısına ekleyebilirsiniz. Kod her çalıştırıldığında, bu bölüm mevcut kamera modunun kodunu çalıştıracaktır.



13.2.4 LabVIEW Örnekleri ve Rehberleri

Popüler Öğreticiler

Otonom Zamanlı Hareket Eğitimi

- Robotunuzu farklı zaman aralıkların da otonom olarak hareket ettirin
- Otonom Hareket hakkında daha fazla bilgi edinin

Temel Motor Kontrol Eğitimi

- RoboRIO motor donanımınızı ve yazılımınızı kurun
- FRC ® 'i kurmayı öğrenin Kontrol Sistemi ve FRC Robot Projesi
- Motor Kontrolü hakkında daha fazlasını görün

Görüntü İşleme Eğitimi

- Temel Görüntü İşleme tekniklerini ve NI Vision Assistant'ı nasıl kullanacağınızı öğrenin
- Kameralar ve Görüntü İşleme hakkında daha fazla bilgi edinin

PID Kontrol Eğitimi

- PID Kontrolü nedir ve bunu nasıl uygulayabilirim?

Command and Control <<https://forums.ni.com/t5/FIRST-Robotics-Competition/Command-and-Control-Tutorial/ta-p/3534946?profile.language=en>>`__

- Command ve Control nedir ?
- Nasıl uygulayım?

Sürücü İstasyonu Eğitimi

- FRC Driver Station'ı tanıyın

Test Modu Rehberi<https://forums.ni.com/t5/FIRST-Robotics-Competition/FRC-2015-Test-Mode-Tutorial/ta-p/3535797?profile.language=en> _

- Test Modunu kurmayı ve kullanmayı öğrenin

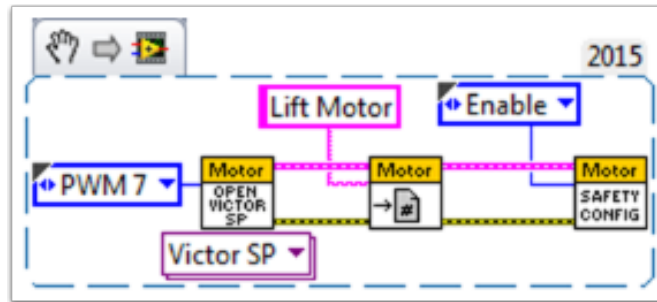
Daha fazla örnek ve bilgi mi arıyorsunuz? Daha fazla belgede arama yapın veya kendi sorunuzu, örnek kodunuzu veya rehberinizi `buraya tıklayarak gönderin! <<https://forums.ni.com/t5/FIRST-Robotics-Competition/tkb-p/3019?profile.language=en>>` __ Yayınlarınızı bir etiketle işaretlemeyi unutmayın!

13.2.5 Bir Projeye Bağımsız Motor Ekleme

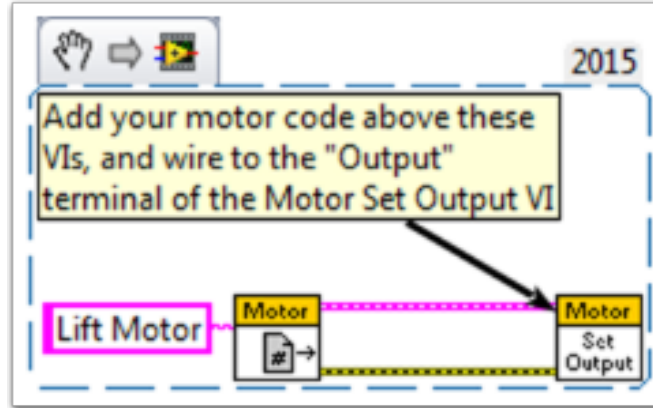
Tekerlekleri kontrol eden sürücünüzün tamamı ayarlandıktan sonra, kol gibi tekerleklerden tamamen bağımsız bir şeyi kontrol etmek için ek bir motor eklemeniz gerekebilir. Bu motor tankınızın, atari salonunuzun veya mecanum tekerleklerinin bir parçası olmayacağından, kesinlikle bağımsız olarak kontrol edilmesini isteyeceksiniz.

Bu VI Parçacıkları, halihazırda çok motorlu bir sürücü içeren bir projede tek bir motorun nasıl kurulacağını gösterir. HAND> OK> LABVIEW sembolünü görürseniz, resmi blok diyagramınıza sürükleyin ve işte: kod! Tamam, işte nasıl yapıyorsun.

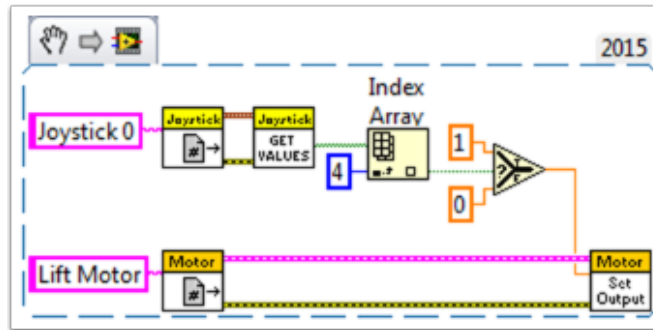
FIRST, create a motor reference in the **Begin.vi**, using the **Motor Control Open VI** and **Motor Control Refnum Registry Set VI**. These can be found by right-clicking in the block diagram and going to **WPI Robotics Library>>RobotDrive>>Motor Control**. Choose your *PWM* line and name your motor. 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.)



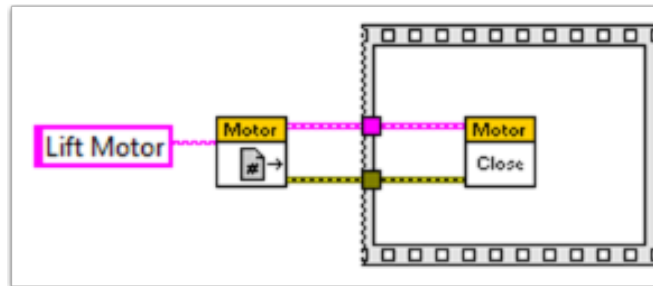
Şimdi, **** Motor Control Refnum Registry Get VI **** 'yı kullanarak **** Teleop.vi **** de motorunuza (adın tam olması gerekir) referans verin ve **** Motor Kontrol Seti Çıkışı VI *** ile ne yapacağını söyleyin *****. Bunlar yukarıdaki VI'larla aynı yerdedir.



Örneğin, sonraki parça Kaldırma Motoruna, Kumanda Kolu 0'da düğme 4'e basıldığında ileri hareket etmesini ve aksi takdirde hareketsiz kalmasını söyler. Benim için düğme 4, Xbox tarzı denetleyicimin ("Joystick 0") sol tamponudur. Çok daha derinlemesine kumanda kolu düğmesi seçenekleri için, bakın: ref: `Motorları veya Solenoidleri Kontrol Etmek İçin Kumanda Kolu Düğmelerini Kullanma <docs / software / labview / kaynaklar / nasıl-kullanılır-joystick-düğmeleri-kontrol-motorları-veya-solenoidler: Motorları veya Solenoidleri Kontrol Etmek İçin Kumanda Kolu Düğmelerini Kullanma>`.



Son olarak, **** Motor Control Refnum Registry Get VI **** ve **** Motor Control Close VI **** kullanarak **** Finish.vi **** 'deki (tıpkı sürücü ve joystick ile yaptığımız gibi) referansları kapatmamız gerekir. . Bu resim Close VI'yı tek başına düz bir sekans yapısında gösterirken, tüm Close VI'ların aynı çerçevede olmasını gerçekten istiyoruz. Bu iki VI'yı diğer Get VI ve Close VI'ların altına yerleştirebilirsiniz (joystick ve sürücü için).

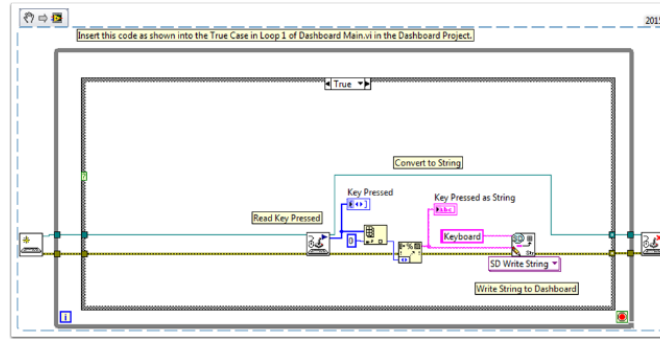


Umarım bu, şimdiye kadarki en iyi robotu programlamanıza yardımcı olur! İyi şanslar!

13.2.6 RoboRIO ile Klavye Navigasyonu

Bu örnek, bir joystick veya başka bir kontrolcü aygıt yerine klavye yönlendirmesi kullanarak robotu kontrol etmek için bazı öneriler sunar. Bu durumda, bir tank drive konfigürasyonunda iki sürücü motorunu kontrol etmek için A, W, S ve D tuşlarını kullanırız.

İlk VI Snippet'i, Dashboard Main VI'ya dahil edilmesi gereken koddur. Bu kodu Loop 1'in True durumuna ekleyebilirsiniz. Kod, loop-döngü başlamadan önce klavyeye bir bağlantı açar ve her yinelemede basılan tuşu okur. Bu bilgiler bir stringe dönüştürülür ve daha sonra robot projesinde Teleop VI'ya aktarılır. Loop 1'in çalışması durduğunda, klavye bağlantısı kapanır.

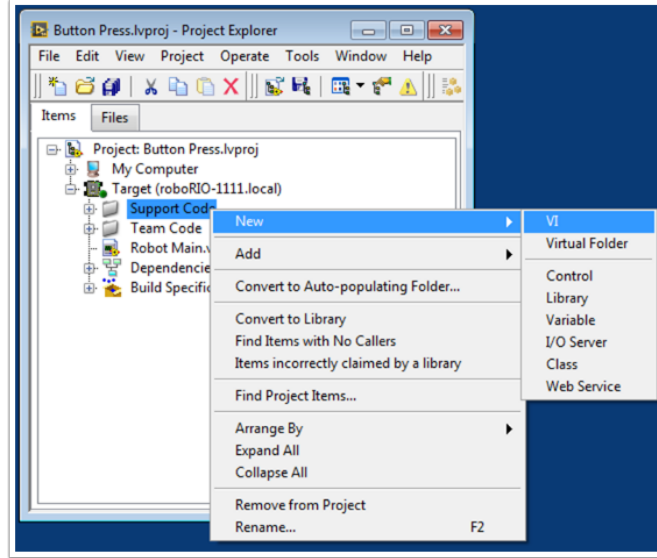


İkinci VI Snippet'i, Teleop VI'ya dahil edilmesi gereken koddur. Bu, Dashboard'dan hangi tuşa basıldığını gösteren string değerini okur. Bir Case Structure-Koşul Yapısı daha sonra, anahtara bağlı olarak sol ve sağ motorlara hangi değerlerin yazılması gerektiğini belirler. Bu durumda, W ileri, A sol, D sağ ve S ters. Bu örnekteki her durum, motorları yarı hızda çalıştırır. Bunu kodunuzda aynı şekilde tutabilir, değerleri değiştirebilir veya sürücünün hızı ayarlamasına izin vermek için ek kod ekleyebilirsiniz, böylece gerektiğinde hızlı veya yavaş sürebilirsiniz. Motor değerleri seçildikten sonra sürücü motorlarına yazılır ve motor değerleri dashboardda yayınlanır.

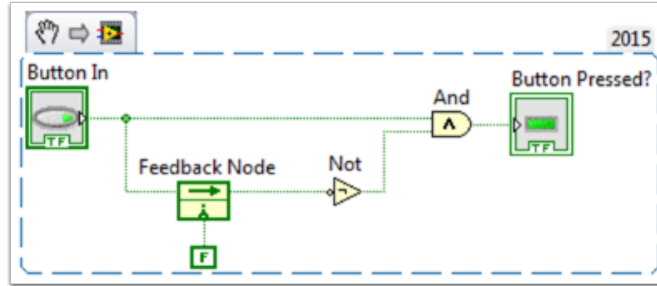
13.2.7 One-Shot Buton yapmak

Joystick Get Values işlevini kullanırken, bir joystick düğmesine basmak, düğme bırakılana kadar düğmenin TRUE okumasına neden olur. Bu, büyük olasılıkla her basışta birden çok DOĞRU değeri okuyacağınız anlamına gelir. Ya düğmeye her basıldığında yalnızca bir DOĞRU değeri okumak isterseniz? Bu genellikle "One-shot Button" olarak adlandırılır. Aşağıdaki eğitim, bunu yapmak için Teleop.vi'nize bırakabileceğiniz bir subVI'yı nasıl oluşturacağınızı gösterecektir.

Öncelikle, projenizin Support Code klasöründe yeni bir VI oluşturun.



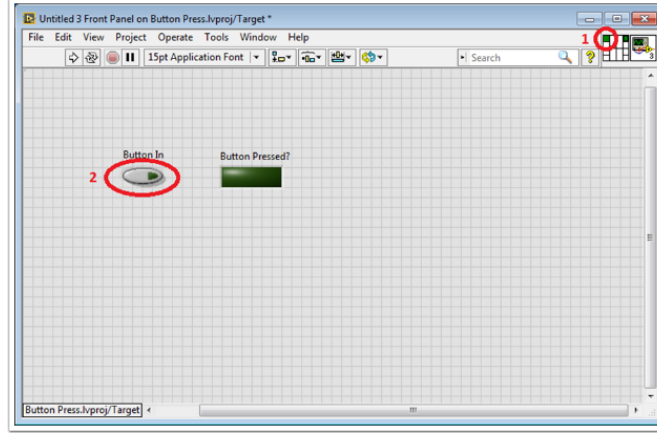
Şimdi yeni VI'nın blok diyagramında, aşağıdaki kod parçasığını bırakın.



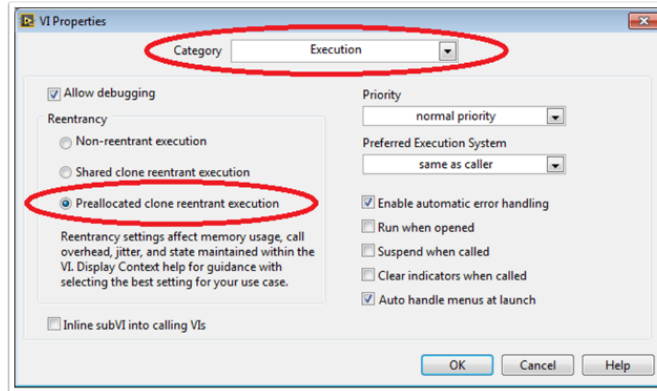
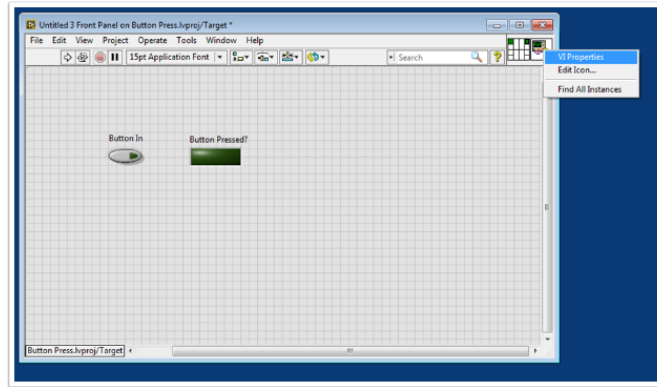
Bu kod, Feedback Node adı verilen bir işlevi kullanır. Düğmenin mevcut değerini feedback node düğümünün sol tarafına bağladık. Feedback node düğümünün okundan çıkan tel, düğmenin önceki değerini temsil eder. Geri bildirim düğümünüzdeki ok burada gösterildiği gibi ters yöne gidiyorsa, yönü tersine çevirme seçeneğini bulmak için sağ tıklayın.

Bir düğmeye basıldığında, düğmenin değeri FALSE'dan TRUE'ya gider. Bu VI'nın çıktısının yalnızca düğmenin mevcut değeri TRUE ve düğmenin önceki değeri FALSE olduğunda TRUE olmasını istiyoruz.

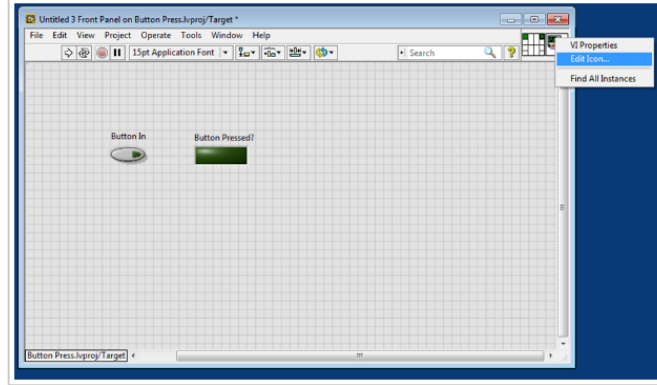
Daha sonra boolean kontrolünü ve göstergesini VI'nın giriş ve çıkışlarına bağlamamız gerekir. Bunu yapmak için, önce bağlayıcı bölmesindeki bloğa tıklayın, ardından ikisini bağlamak için düğmeye tıklayın (aşağıdaki şemaya bakın). Gösterge için bunu tekrarlayın.



Daha sonra, bu VI'nın özelliklerini TeleOp.vi'mizde bu VI'nın katlarını kullanabilmemiz için değiştirmemiz gerekiyor. VI Simgesine sağ tıklayın ve VI Özellikleri'ne gidin. Ardından "Execution" kategorisini seçin ve "Preallocated clone reentrant execution" yı seçin.

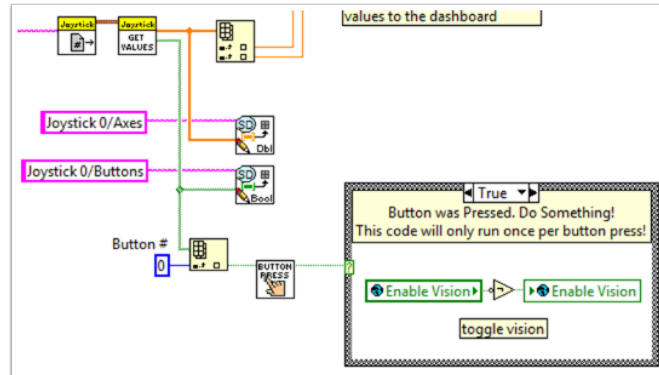


Son olarak, VI Simgesini VI'nın işlevini daha açıklayıcı olacak şekilde değiştirmeliyiz. Simgeye sağ tıklayın ve Edit Icon'a gidin. Create a new Icon-Yeni bir Simge oluşturun.



Son olarak, VI'yı açıklayıcı bir adla kaydedin. Artık bu VI'yı Support Files klasöründen Tele-Op.vi'nize sürükleyip bırakabilirsiniz. İşte tamamlanan VI'nın bir kopyası: Button_Press.vi

İşte bu VI'yı nasıl kullanabileceğinize dair bir örnek.



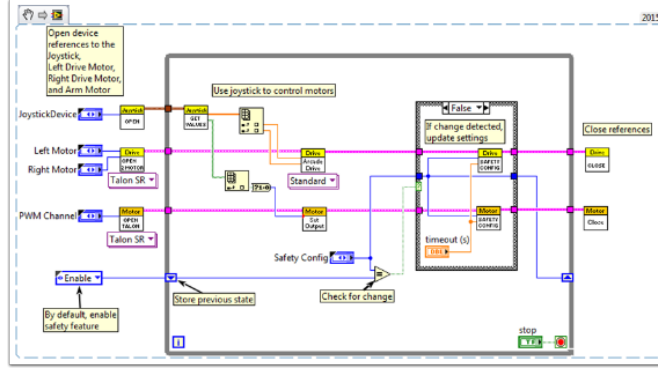
13.2.8 Robot Kodunuza Güvenlik Özellikleri Ekleme

Karmaşık projelerde yaygın olarak görülen bir sorun, tüm kodunuzun beklediğiniz anda çalıştığından emin olmaktır. RoboRIO’da yüksek önceliğe sahip görevler, uzun yürütme süreleri veya sık çağırımlar işlem gücü aldığında sorunlar ortaya çıkabilir. Bu, işlemcinin meşgul olması nedeniyle gerçekleştirilemeyen görevler için “starvation-açlıktan ölme” olarak bilinen duruma yol açar. Çoğu durumda bu, kumanda kollarından ve diğer cihazlardan girişinize tepki süresini basitçe yavaşlatır. Ancak bu, robotunuzun tahrik motorlarının siz onları durdurmaya çalıştıktan sonra uzun süre açık kalmasına da neden olabilir. Bundan dolayı herhangi bir robotik felaketten kaçınmak için, görev girdisi tıkanmasını kontrol eden ve potansiyel olarak zararlı işlemleri otomatik olarak kapatan güvenlik özellikleri uygulayabilirsiniz.

Motorlar için güvenlik kontrollerinin kolay uygulanmasını sağlayan yerleşik fonksiyonlar vardır. Bu işlevler şunlardır:

- Robot Sürüş Güvenlik Yapılandırması
- Motor Sürücü Güvenlik Yapılandırması
- Röle Güvenliği Yapılandırması
- *PWM* Safety Configuration
- Solenoid Güvenlik Yapılandırması
- Robot Sürücü Gecikmesi ve Güncelleme Güvenliği

Tüm Güvenlik Yapılandırması işlevlerinde, programlamanız çalışırken güvenlik kontrollerini etkinleştirebilir ve devre dışı bırakabilir ve hangi zaman aşımının uygun olduğunu düşündüğünüzü yapılandırabilirsiniz. İşlevler, güvenliği etkinleştirilmiş tüm aygıtların bir önbellegini tutar ve bunlardan herhangi birinin zaman sınırlarını aşıp aşmadığını kontrol eder. Varsa, önbellekteki tüm cihazlar devre dışı bırakılır ve robot hemen durur veya röle / PWM / solenoid çıkışları kapatılır. Aşağıdaki kod, motorların kapatılmadan önce hiçbir giriş almayacağı maksimum bir zaman sınırı ayarlamak için Sürücü Güvenlik Yapılandırması işlevlerinin nasıl kullanılacağını gösterir.



Emniyet kapatmasını test etmek için, döngüye zaman aşımınızdan daha uzun bir Bekleme işlevi eklemeyi deneyin!

Güvenlik kontrollerinin uygulanmasıyla ilgili son işlev-Robot Drive Delay and Update Safety-roboRIO'yu zaman sınırını aşmadan Otonom Moda geçirmenizi sağlar. Sürücü Çıkışı işlevlerine ağır yüklü çağrılar yapmadan mevcut motor çıkışını koruyacak ve ayrıca motorların aniden durmaması için güvenlik kontrollerinin düzenli olarak güncellenmesini sağlayacaktır.

Genel olarak, robotunuzun kasıtsız olarak tehlikeli bir durumda bırakılmadığından emin olmak için projenizde bir tür güvenlik kontrolünün uygulanması şiddetle tavsiye edilir!

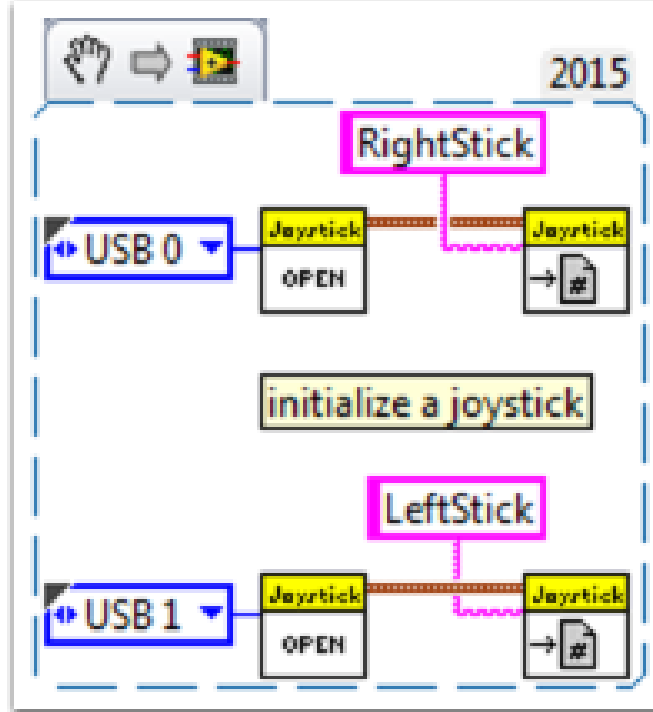
13.2.9 Motorları veya Solenoidleri Kontrol Etmek İçin Kumanda Kolu Düğmelerini Kullanma

Tahrik sistemlerimizi çalışır hale getirdikçe, motorlar ve solenoidler gibi yardımcı cihazlarımızı bağlamaya geçiyoruz. Bununla birlikte, bu cihazları kontrol etmek için genellikle joystick düğmelerini kullanacağız. Buna başlamak için, joystick düğmeleriyle cihazları kontrol etmenin birkaç yolunu inceleyeceğiz.

Bunun gibi bir belgeden VI Snippet'i tıklayıp doğrudan LabVIEW kodunuza sürükleyebileceğinizi biliyor muydunuz? Bunu bu belgedeki parçacıklarla deneyin.

Kurulum:

Yapılandırma ne olursa olsun, "Begin.vi" ye bir, iki veya daha fazla (gerçekten heyecanlıysanız) joystick eklemeniz gerekir. İlk örnek 2 oyun çubuğu kullanır ve diğerleri yalnızca birini kullanır. Aşağıdaki snippet gibi başka yerlerde de kullanabilmemiz için her birine benzersiz bir ad verin. Masamın sol ve sağ tarafında oldukları için onlara "LeftStick" ve "RightStick" adını verdim. Oyun çubuklarınız zaten yapılandırılmışsa, harika! Bu adımı atlayabilirsiniz.

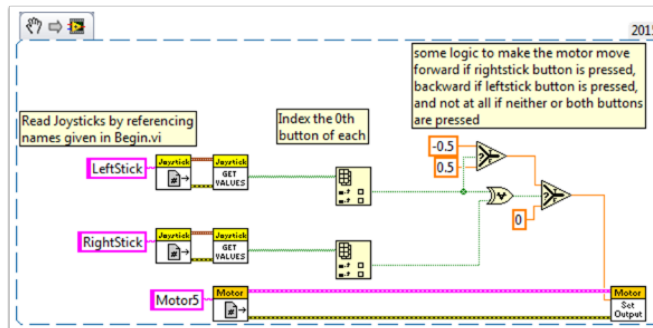


Bu belgedeki kodun geri kalanı “Teleop.VI” içine yerleştirilecektir. Burada, kumanda kolu düğmelerimizi motorlarımızın veya solenoidlerimizin farklı yönlerini kontrol etmek için programlayacağız.

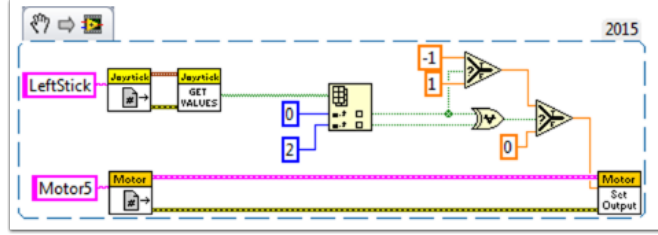
Senaryo 1

“Bir düğmeye bastığımda motorun bir yöne, farklı bir düğmeye bastığımda diğer yöne hareket etmesini istiyorum.”

Bu kod, aynı motoru kontrol etmek için iki farklı kumanda kolundaki 0 düğmesini kullanır. LeftStick üzerindeki 0 butonuna basılırsa, motor geriye doğru hareket eder ve RightStick üzerindeki 0 butonuna basılırsa, motor ileri doğru hareket eder. Her iki düğmeye de basılırsa veya hiçbir düğmeye basılmazsa, motor hareket etmez. Burada motor referansına “Motor5” adını verdim, ancak motorunuza “Begin.vi” de istediğiniz adı verebilirsiniz.



Kontrol için aynı joystick’ten birden fazla düğme kullanmak isteyebilirsiniz. Bunun bir örneği için, Senaryo 2’deki VI snippetine veya aşağıdaki VI snippetine bakın.



Burada 0 ve 2 numaralı kumanda kolu düğmelerini kullandım, ancak ihtiyacınız olan düğmeleri kullanmaktan çekinmeyin.

Senaryo 2

** Farklı kumanda kolu düğmelerinin çeşitli hızlarda hareket etmesini istiyorum. **

Bu örnek, bastığınız düğmelere bağlı olarak bir motora farklı şeyler yaptırmanız gerektiğinde yardımcı olabilir. Örneğin, oyun çubuğumun bir tetikleyici (düğme 0) ve üstte 4 düğme (1-4 arası düğmeler) olduğunu varsayalım. Bu durumda, aşağıdaki düğmeler aşağıdaki işlevlere sahip olmalıdır:

- button 1 - yarım hızda geri git
- button 2 - yarım hızda ileri git
- button 3 - 1/4 hızda geri git
- button 4 - 1/4 hızda ileri git
- trigger- tam hız ileri! (tam hızda ileri)

Daha sonra boolean dizisini “JoystickGetValues.vi” den alır ve onu bir “Boolean Array to Number” düğümüne (Numeric Palette-Conversion Palette) bağlarız. Bu, boolean dizisini kullanabileceğimiz bir sayıya dönüştürür. Bu rakamı bir case structure’a bağlayın.

Her durum, dizideki değerlerin ikili gösterimine karşılık gelir. Bu örnekte, her durum tek düğmeli bir kombinasyona karşılık gelir. Altı durum ekledik: 0 (tüm düğmeler kapalı), 1 (düğme 0 açık), 2 (düğme 1 açık), 4 (düğme 2 açık), 8 (düğme 3 açık) ve 16 (düğme 4 açık). 3 değerini atladığımızı dikkat edin. 3 aynı anda basılan 0 ve 1 düğmelerine karşılık gelir. Bunu gereksinimlerimizde tanımlamadık, bu yüzden varsayılan durumun ele almasına izin vereceğiz.

LabVIEW 2014 Case Structure Yardım belgesini buradan incelemek faydalı olabilir:

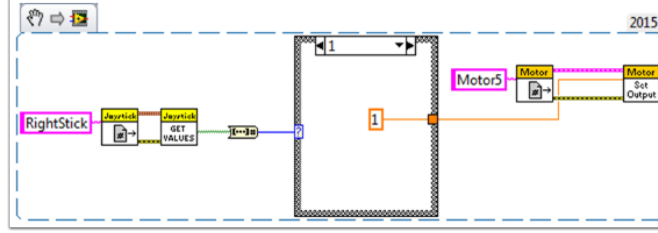
https://zone.ni.com/reference/en-XX/help/371361L-01/glang/case_structure/

Ayrıca, burada case yapılarıyla ilgili 3 Topluluk rehberi vardır:

<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>

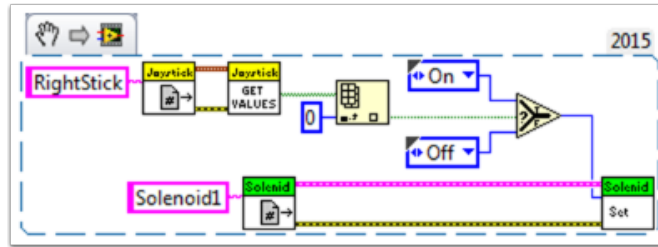


Gereksinimlerimiz basit olduğundan, her durumda sadece tek bir sabite ihtiyacımız var. Durum 1 için (tam ileri) 1, 2. durum için (yarım geri) a -0.5 kullanıyoruz, vb. 1 ile -1 arasındaki herhangi bir sabit değeri kullanabiliriz. 0 durumunu varsayılan olarak bıraktım, bu nedenle birden fazla düğmeye basıldığında (tanımlanmamış herhangi bir duruma ulaşıldığında) motor duracaktır. Elbette bu durumları istediğiniz gibi özelleştirmekte özgürsünüz.

Senaryo 3

“Joystick düğmelerimle bir solenoidi kontrol etmek istiyorum.”

Şimdiye kadar, joystick'in düğmeleri bir dizi boolean'den nasıl çıkarıldığını biliyoruz. İlgilendiğimiz düğmeyi elde etmek için bu diziyi indekslememiz ve bu booleanı bir seçili düğme bağlamamız gerekir. “Solenoid Set.vi” giriş olarak bir Enum gerektirdiğinden, numaralandırmanın en kolay yolu “Solenoid Set.vi” nin “Value” girişine sağ tıklamak ve “Create Constant” u seçmektir. Bu sabiti çoğaltın ve bir kopyasını True terminale ve birini de seçilen düğmünün False terminaline bağlayın. Ardından, seçme düğmünün çıkışını solenoid VI'nın “Value” girişine bağlayın.



Mutlu Robot Çalışmaları!

13.2.10 FRC için LabVIEW'de Local ve Global Değişkenler

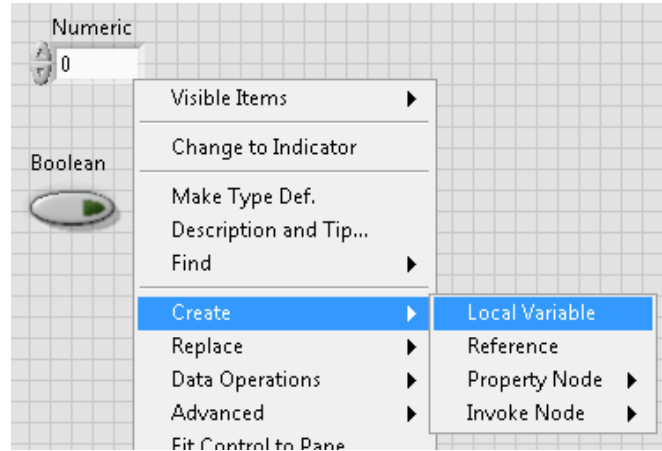
Bu örnek, yerel ve genel değişkenlere, FRC için varsayılan LabVIEW'de nasıl kullanıldıklarına bir giriş olarak hizmet eder ® Robot Projesi ve bunları projenizde nasıl kullanmak isteyebileceğiniz.

Localk ve global değişkenler, aynı VI (yerel değişkenler) veya farklı VI'lar (global değişkenler) içindeki konumlar arasında veri aktarmak için kullanılabilir ve LabVIEW'in meşhur olduğu geleneksel [Veri Akışı Paradigması](#) 'i bozar. Bu nedenle, herhangi bir nedenle değeri doğrudan bir node dan başka bir node a bağlayamadığınızda yararlı olabilirler.

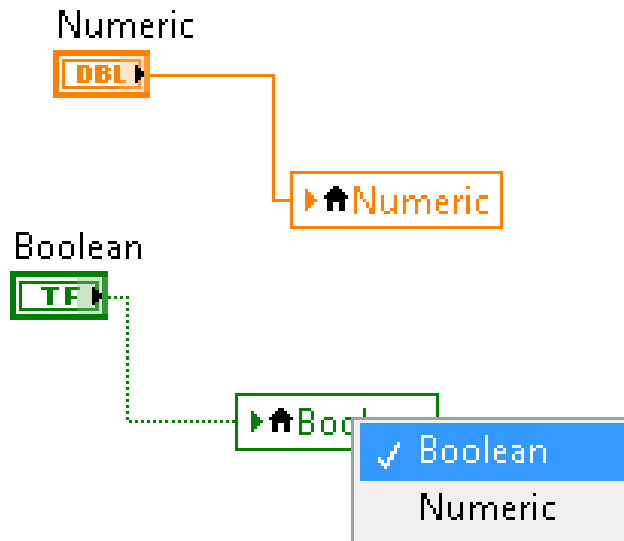
Not: Olası nedenlerden biri, verileri ardışık döngü yinelemeleri arasında geçirmeniz gerekmesidir; Miro_T, [bu yazıda](#) ele alındı. LabVIEW'deki **feedback node** https://zone.ni.com/reference/en-XX/help/371361L-01/lvconcepts/block_diagram_feedback/ ' __ shift register'a eşdeğer olarak kullanılabileceği, ancak bu başka bir konu olabilir!

Local ve Global Değişkenlere Giriş

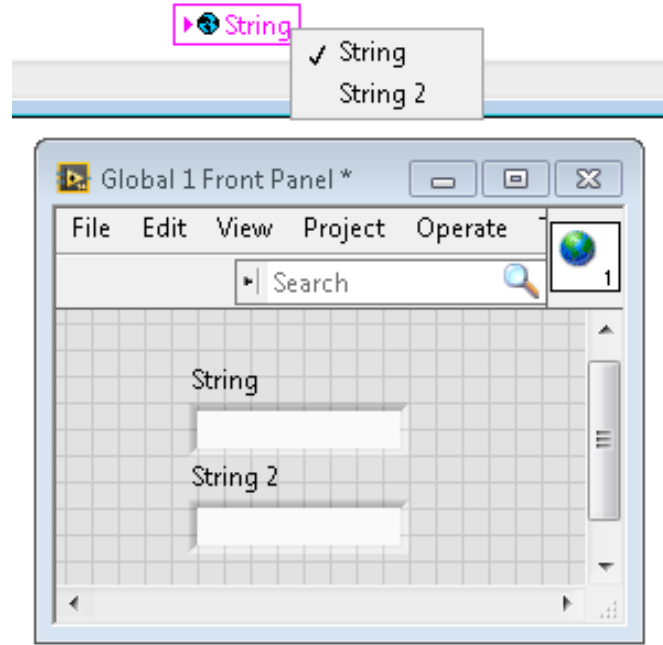
Yerel değişkenler aynı VI içerisinde kullanılabilir. Ön Panelinizde bir kontrole veya göstergeye sağ tıklayarak yerel bir değişken oluşturun:



Blok diyagram üzerindeki Structures paletinden de bir yerel değişken oluşturabilirsiniz. Bir VI'da birden fazla yerel değişkeniniz olduğunda, hangi değişkeni seçmek için sol tıklayabilirsiniz:



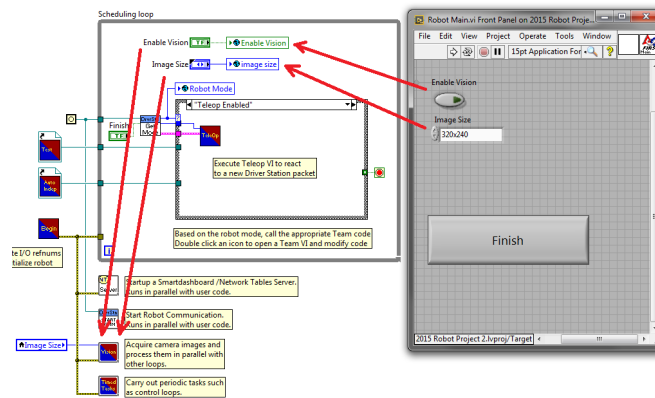
Global değişkenler biraz farklı oluşturulur. Structures paletinden blok diyagrama bir tane ekleyin ve üzerine çift tıkladığınızda ayrı bir ön panel açtığına dikkat edin. Bu ön panelin bir blok diyagramı yoktur, ancak ön panele istediğiniz kadar varlık ekleyip *.Vi dosyası olarak kaydedebilirsiniz:



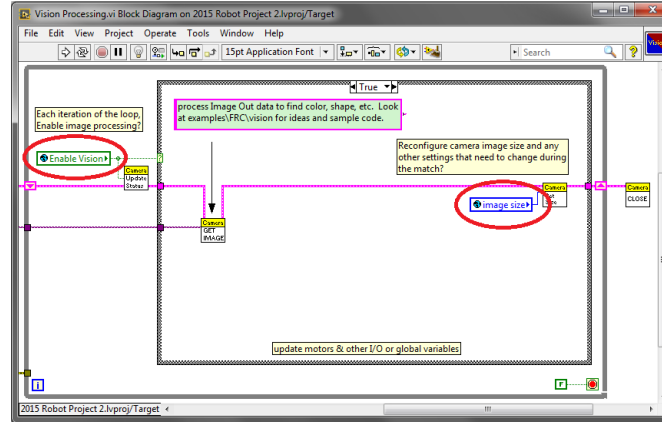
Not: Local ve Global değişkenleri kullanırken yarış koşullarından kaçınmaya çok dikkat edin! Esasen, en son hangi konuma yazıldığını bilmenin bir yolu olmadan aynı değişkene birden çok konumda yanlışlıkla yazmadığınızdan emin olun. Daha kapsamlı bir açıklama için [şu yardım belgesine](#) bakın:

FRC Robot Project için Default LabVIEW Nasıl Kullanılırlar?

“Vizyonu Etkinleştir” ve “Görüntü Boyutu” için global değişkenler, Robot Ana VI’nın her yinelenmesi sırasında yazılır...



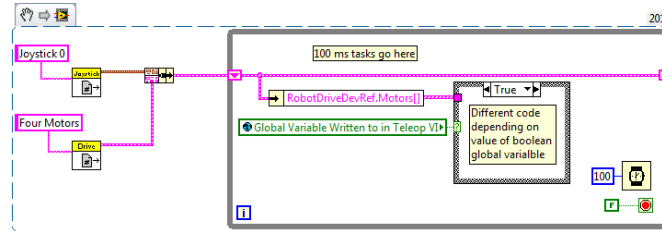
... Ve sonra Vision Processing VI’nın her yinelenmesinde okuyun:



Bu, kullanıcının, LabVIEW Development Environment dan Robot Main VI'ya konuşlandırırken, görüntü işlemeyi enable/disable etmesine ve Robot Main'nin Ön Panelinden görüntü boyutunu değiştirmesine olanak tanır.

Bunları Projenizde Nasıl Kullanabilirsiniz?

Periodic Tasks VI için blok şemasına bakın. Belki de Boolean gibi, Teleop VI'da global bir değişkene yazılabilen ve ardından Periodic Tasks VI'dan okunabilen bir değer vardır. Ardından, boolean genel değişkenine bağlı olarak, Periodic Tasks VI'da hangi kod veya değerleri kullanacağınıza karar verebilirsiniz:



13.2.11 Kompresörü LabVIEW'de Kullanma

Bu kod parçası roboRIO projenizi Pnömatik Kontrol Modülünü (PCM) kullanmak için nasıl ayarlayacağınızı gösterir. PCM, tankta belirli basınçlar ölçüldüğünde kompresörü otomatik olarak başlatır ve durdurur. RoboRIO programınızda aşağıdaki VI'ları eklemeniz gerekecektir.

Daha fazla bilgi için aşağıdaki bağlantılara bakın:

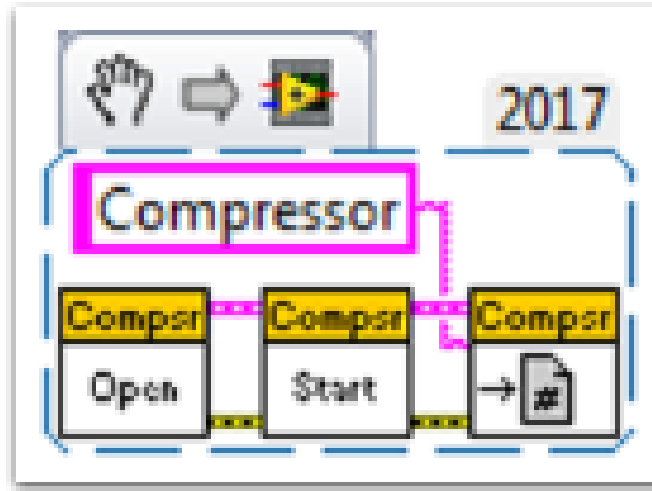
FRC Pneumatics Manual

PCM User's Guide

`RoboRIO için Adım Adım Pnömatik<<http://team358.org/files/pneumatic/Pneumatics-StepByStep-roboRIO.pdf>>`

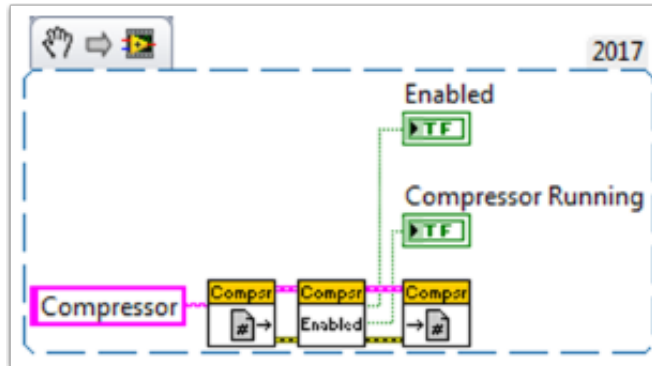
VI'yı başlatın

Bu pasajı Begin.vi'ye yerleştirin.



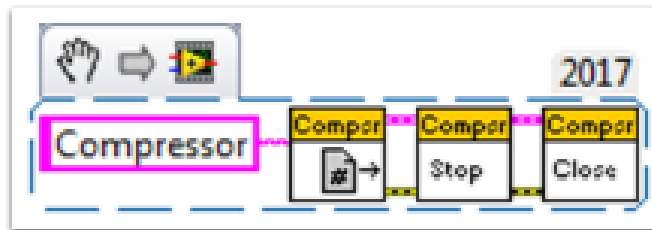
Teleop VI

Bu bölümü Teleop.vi'ye yerleştirin. Bu bölüm yalnızca çıktıları diğer işlemler için kullanıyorsanız gereklidir.



VI'yı sonlandır

Bu parçacığı, Finish.vi'nin Close Refs, verileri kaydetme vb. bölümüne yerleştirin.



14.1 pyproject.toml usage

Not: 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"
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
# "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
```

Not: 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

Not: 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

Uyarı: 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.

Ayrıca bakınız:

[Konsol Çıktısını Görüntüleme](#)

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
```

Not: 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
```


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.

Not: 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 *Üçüncü Taraf CAN Cihazları*, *3. Taraf Kütüphaneleri*, and *Third Party Example Projects* for more information.

15.1.2 PWM Motor Controllers in Depth

İpucu: 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

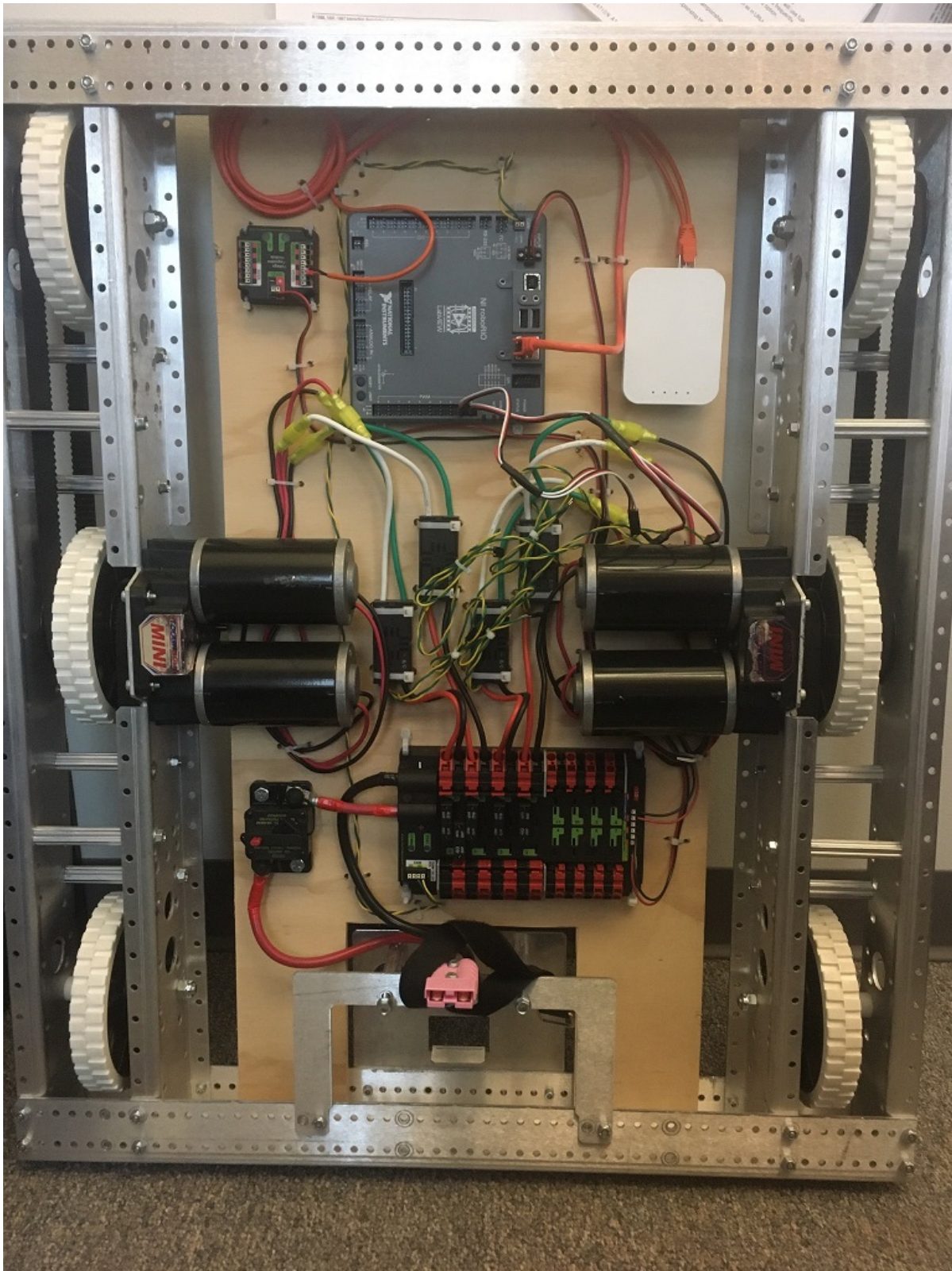
So if WPILib handles all this scaling, why would you ever need to calibrate your motor controller? The values WPILib uses for scaling are approximate based on measurement of a number of samples of each controller type. Due to a variety of factors, the timing of an individual motor controller may vary 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.

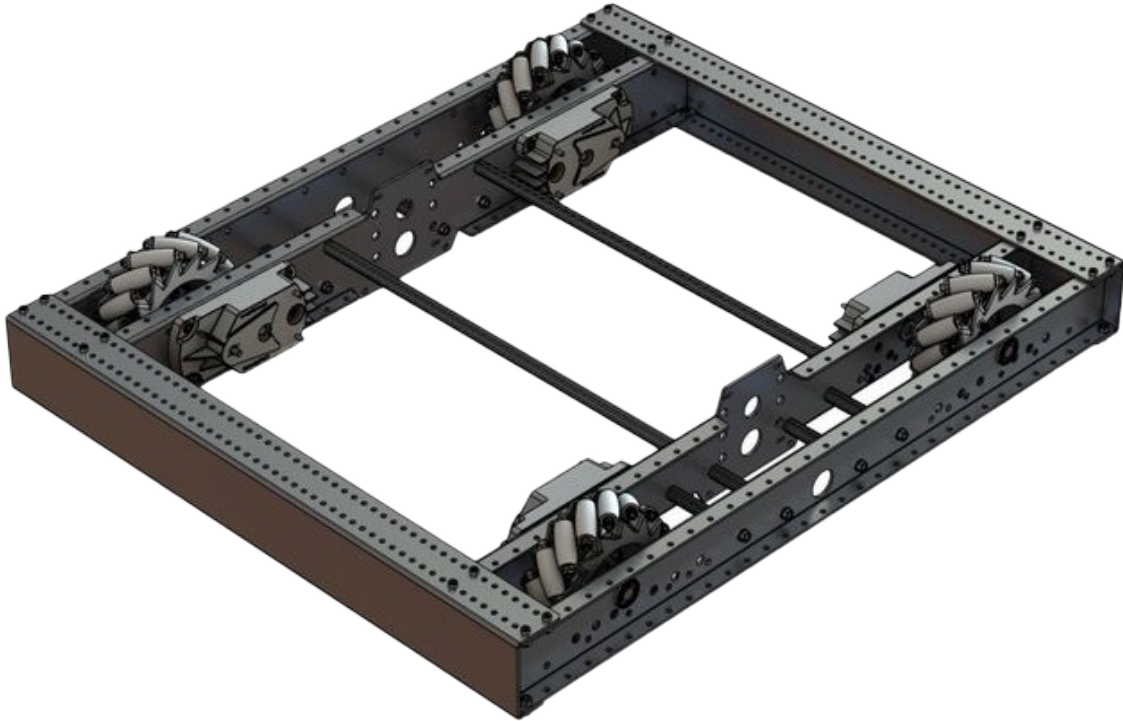
Standard drivetrains

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 PID-Controller), 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.

Not: See the [Coordinate System](#) section for more detail about the axis conventions and coordinate systems.

Using the DifferentialDrive class to control Differential Drive robots

Not: 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 documentation 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 `Subsystem`):

```

// 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

Not: 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)
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
right = wpilib.MotorControllerGroup(frontLeft, rearLeft)

self.drive = wpilib.drive.DifferentialDrive(left, right)
```

Drive Modes

Not: 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);
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    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
    )

```

(sonraki sayfaya devam)

```

    )

    self.stick = wpilib.Joystick(self.kJoystickChannel)

```

Mecanum Drive Modes

Not: 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 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.

Not: 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.

Not: 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++):

Not: 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 proportional 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 [Donanım Bileşenine Genel Bakış](#).

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

Not: This section covers using sensors in software. For a guide to sensor hardware, see [Sensöre Genel Bakış - Donanım](#).

Not: 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

Not: This section covers accelerometers in software. For a hardware guide to accelerometers, see *İvmeölçerler - Donanım*.

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

Not: 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

```
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++

```
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

Not: This section covers gyros in software. For a hardware guide to gyros, see [Gyroscopes - Donanim](#).

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.

Not: 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.

Uyarı: 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.

Uyarı: 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.

Not: 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.

Not: 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

Not: 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).

Uyarı: 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)
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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)

```

Not: 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);
}

```

(sonraki sayfaya devam)

```

@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);
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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.

Not: 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

Not: This section covers ultrasonics in software. For a hardware guide to ultrasonics, see [Ultrasonik - Donanım](#).

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);
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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:

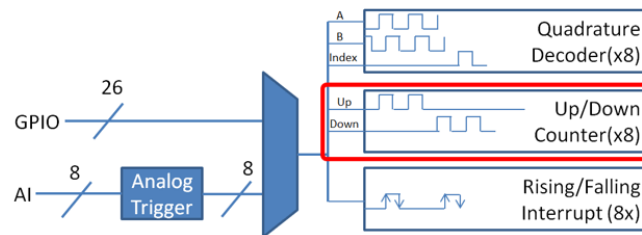
Java

```
// 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.

Not: 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.

Not: 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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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

Not: 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.

Not: 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

Not: 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

Not: 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

Not: 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

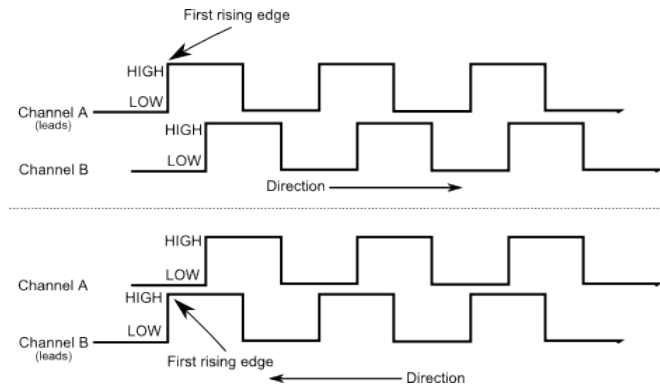
Not: This section covers encoders in software. For a hardware guide to encoders, see [Kodlayıcılar-Donanım](#).

Encoders are devices used to measure motion (usually, the rotation of a shaft).

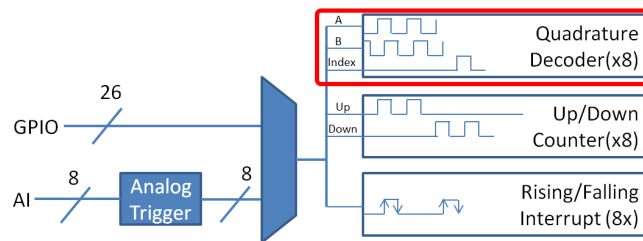
Önemli: The classes in this document are only used for encoders that are plugged directly into the roboRIO! Please reference the appropriate vendors' documentation for using encoders plugged into motor controllers.

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

Not: 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.

Not: 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

Not: 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

Not: 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

Not: 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

Not: 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

Not: 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

Not: 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*

Not: 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); }};
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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:

Not: 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);
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
        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

Not: This section covers analog inputs in software. For a hardware guide to analog inputs, see [Analog Girişler - Donanım](#).

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

Not: 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:

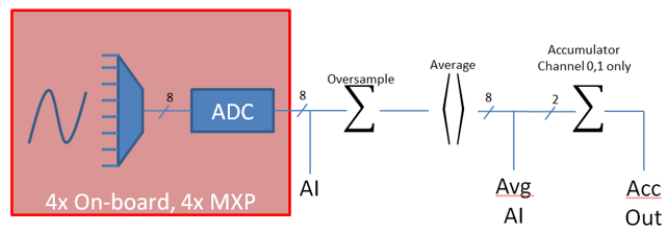
JAVA

```
// 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);
```

Not: 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

Not: 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 necessary 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

Not: This section covers analog potentiometers in software. For a hardware guide to analog potentiometers, see [Analog Potansiyometreler - Donanim](#).

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

Not: 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

Not: 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 [ultraso-nics](#).

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

Not: This section covers digital inputs in software. For a hardware guide to digital inputs, see [Dijital Girişler - Donanım](#).

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 [Dijital Girişler - Donanım](#)). 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

Not: 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

Not: 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 {
```

(sonraki sayfaya devam)


```

    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);
        }
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    } 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.

Uyarı: 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.

Not: 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++

```

11 class Robot : public frc::TimedRobot {
12     private:
13         static constexpr int kLength = 60;
14
15         // PWM port 9
16         // Must be a PWM header, not MXP or DIO
17         frc::AddressableLED m_led{9};
18         std::array<frc::AddressableLED::LEDData, kLength>
19             m_ledBuffer; // Reuse the buffer
20         // Store what the last hue of the first pixel is
21         int firstPixelHue = 0;
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

7 void Robot::RobotInit() {
8     // Default to a length of 60, start empty output
9     // Length is expensive to set, so only set it once, then just update data
10    m_led.SetLength(kLength);
11    m_led.SetData(m_ledBuffer);
12    m_led.Start();
13 }

```

Not: 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);

```

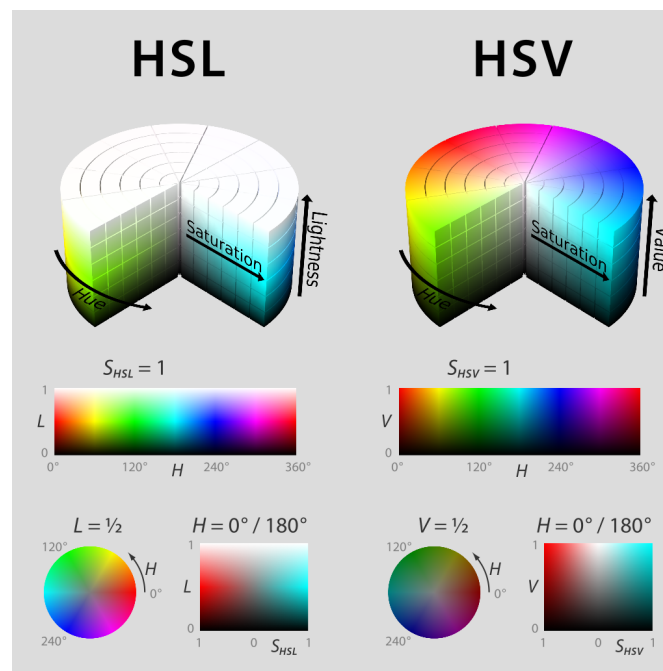
C++

```
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.

Not: 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()))
48         ↪ % 180;
49         // Set the value
50         m_ledBuffer.setHSV(i, hue, 255, 128);
51     }
52     // Increase by to make the rainbow "move"
53     m_rainbowFirstPixelHue += 3;
54     // Check bounds
55     m_rainbowFirstPixelHue %= 180;
56 }
```

C++

```
22 void Robot::Rainbow() {
23     // For every pixel
24     for (int i = 0; i < kLength; i++) {
25         // Calculate the hue - hue is easier for rainbows because the color
26         // shape is a circle so only one value needs to precess
27         const auto pixelHue = (firstPixelHue + (i * 180 / kLength)) % 180;
28         // Set the value
29         m_ledBuffer[i].SetHSV(pixelHue, 255, 128);
30     }
31     // Increase by to make the rainbow "move"
32     firstPixelHue += 3;
33     // Check bounds
34     firstPixelHue %= 180;
35 }
```

Now that we have our rainbow method created, we have to actually call the method and set the data of the LED.

Java

```
34 @Override
35 public void robotPeriodic() {
36     // Fill the buffer with a rainbow
37     rainbow();
38     // Set the LEDs
39     m_led.setData(m_ledBuffer);
40 }
```

C++

```
15 void Robot::RobotPeriodic() {
16     // Fill the buffer with a rainbow
17     Rainbow();
18     // Set the LEDs
19     m_led.SetData(m_ledBuffer);
20 }
```

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.

Not: Another name for a motor controller is a speed controller.

İpucu: 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.

Uyarı: 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)

16.1 CAN cihazlarını kullanmak

CAN has many advantages over other methods of connection between the robot controller and peripheral devices.

- CAN bağlantıları, cihazdan cihaza papatya dizimlidir, bu da genellikle her cihazı RIO'nun kendisine bağlamaktan çok daha kısa kablolarla sonuçlanır.
- 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 çift yönlüdür, bu nedenle CAN motor kontrolörleri verileri RIO'ya geri gönderebilir, bu da yine PWM kontrolörlerinin sunabileceğinden daha kapsamlı bir özellik seti olduğunu gösterir.

For instructions on wiring CAN devices, see the relevant section of the *robot wiring guide*.

CAN cihazları genellikle kendi WPILib sınıflarına sahiptir. Aşağıdaki bölümler, bu sınıflardan birkaçının kullanımını açıklayacaktır.

16.2 Pnömatik Kontrol Modülü



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.

Uyarı: 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++

```

28 // Get the voltage going into the PDP, in Volts.
29 // The PDP returns the voltage in increments of 0.05 Volts.
30 double voltage = m_pdp.GetVoltage();
31 frc::SmartDashboard::PutNumber("Voltage", voltage);

```

PYTHON

```

34 # Get the voltage going into the PDP, in Volts.
35 # The PDP returns the voltage in increments of 0.05 Volts.
36 voltage = self.pdp.getVoltage()
37 wpilib.SmartDashboard.putNumber("Voltage", voltage)

```

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**

```

37 // Retrieves the temperature of the PDP, in degrees Celsius.
38 double temperatureCelsius = m_pdp.getTemperature();
39 SmartDashboard.putNumber("Temperature", temperatureCelsius);

```

C++

```

33 // Retrieves the temperature of the PDP, in degrees Celsius.
34 double temperatureCelsius = m_pdp.GetTemperature();
35 frc::SmartDashboard::PutNumber("Temperature", temperatureCelsius);

```

PYTHON

```

39 # Retrieves the temperature of the PDP, in degrees Celsius.
40 temperatureCelsius = self.pdp.getTemperature()
41 wpilib.SmartDashboard.putNumber("Temperature", temperatureCelsius)

```

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

```

41 // Get the total current of all channels.
42 double totalCurrent = m_pdp.getTotalCurrent();
43 SmartDashboard.putNumber("Total Current", totalCurrent);
44
45 // Get the total power of all channels.
46 // Power is the bus voltage multiplied by the current with the units Watts.
47 double totalPower = m_pdp.getTotalPower();
48 SmartDashboard.putNumber("Total Power", totalPower);
49
50 // Get the total energy of all channels.
51 // Energy is the power summed over time with units Joules.
52 double totalEnergy = m_pdp.getTotalEnergy();
53 SmartDashboard.putNumber("Total Energy", totalEnergy);

```

C++

```

37 // Get the total current of all channels.
38 double totalCurrent = m_pdp.GetTotalCurrent();
39 frc::SmartDashboard::PutNumber("Total Current", totalCurrent);
40
41 // Get the total power of all channels.
42 // Power is the bus voltage multiplied by the current with the units Watts.
43 double totalPower = m_pdp.GetTotalPower();
44 frc::SmartDashboard::PutNumber("Total Power", totalPower);
45
46 // Get the total energy of all channels.
47 // Energy is the power summed over time with units Joules.
48 double totalEnergy = m_pdp.GetTotalEnergy();
49 frc::SmartDashboard::PutNumber("Total Energy", totalEnergy);

```

PYTHON

```

43 # Get the total current of all channels.
44 totalCurrent = self.pdp.getTotalCurrent()
45 wpilib.SmartDashboard.putNumber("Total Current", totalCurrent)
46
47 # Get the total power of all channels.
48 # Power is the bus voltage multiplied by the current with the units Watts.
49 totalPower = self.pdp.getTotalPower()
50 wpilib.SmartDashboard.putNumber("Total Power", totalPower)
51
52 # Get the total energy of all channels.
53 # Energy is the power summed over time with units Joules.
54 totalEnergy = self.pdp.getTotalEnergy()
55 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 Üçüncü Taraf CAN Cihazları

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](#)

Çeşitli satıcılardan ortak üçüncü taraf CAN cihazlarının bir listesi ve ilgili harici belgelere bağlantılar aşağıda verilmiştir:

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 Motor Sürücüleri

- **** Talon FX (Falcon 500 Motorlu ile) ****
 - [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 Sensörleri

- **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 Robotik

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 Motor Sürücüleri

- **SPARK MAX**
 - API Documentation ([Java](#), [C++](#))
 - [Technical Manual](#)
- **SPARK Flex**
 - API Documentation ([Java](#), [C++](#))
 - [Technical Manual](#)

16.5.3 Fusion ile Oynamak

Playing With Fusion (PWF), Venom entegre motor / kontrol cihazının yanı sıra bir Uçuş Süresi mesafe sensörü sunar:

PWF Motor Sürücüleri

- **Venom**
 - API Belgeleri ([Java <https://www.playingwithfusion.com/frc/2020/javadoc/com/playingwithfusion/psummary.html>](https://www.playingwithfusion.com/frc/2020/javadoc/com/playingwithfusion/psummary.html) __, [C++ <https://www.playingwithfusion.com/frc/2020/cppdoc/html/annotated.htm>](https://www.playingwithfusion.com/frc/2020/cppdoc/html/annotated.htm) __)
 - [Teknik Kılavuz <https://www.playingwithfusion.com/include/getfile.php?fileid=7086>](https://www.playingwithfusion.com/include/getfile.php?fileid=7086)
—

PWF Sensörleri

- **Time of Flight Sensor**
 - API Belgeleri ([Java <https://www.playingwithfusion.com/frc/2020/javadoc/com/playingwithfusion/psummary.html>](https://www.playingwithfusion.com/frc/2020/javadoc/com/playingwithfusion/psummary.html) __, [C++ <https://www.playingwithfusion.com/frc/2020/cppdoc/html/annotated.htm>](https://www.playingwithfusion.com/frc/2020/cppdoc/html/annotated.htm) __)
 - [Teknik Kılavuz <https://www.playingwithfusion.com/include/getfile.php?fileid=7091>](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 Cihaz Özellikleri

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 adresleme

FRC CAN düğümleri, ID'yi 5 bileşene bölen, önceden tanımlanmış bir şemaya dayalı olarak tanımlar:

Cihaz Türü

Bu, bahsedilen aygıtın türünü açıklayan 5 bitlik bir değerdir. Aşağıda yeni tanımlanmış cihaz türlerinin bir tablosu bulunmaktadır. Rezerve bölümünden yeni tür bir cihaz edinmek istiyorsanız, lütfen FIRST'ten talepte bulununuz.

Cihaz Türleri	
Yayın Mesajları - Broadcast	0
Robot Kontrolörü	1
Motor Sürücüsü	2
Röle Kontrolörü	3
Gyro Sensörü	4
İvmeölçer	5
Ultrasonik Sensör	6
Dişli Sensörü	7
Güç Dağıtım Modülü - Power Distribution Module	8
Pnömatik Kontrolörü	9
Çeşitli	10
IO Breakout	11
Rezerve	12-30
Fabrika Yazılımı güncellemesi	31

Üretici firma

Bu, CAN cihazının üreticisini belirten 8 bitlik bir değerdir. Şu anda atanan değerler aşağıdaki tabloda bulunabilir. Reserved havuzundan bir üretici kimliğinin atanmasını istiyorsanız, lütfen FIRST'a bir istek gönderin.

Üretici firma	
Yayın -Broadcast	0
NI	1
Luminary Micro	2
DEKA	3
CTR Electronics	4
REV Robotics	5
Grapple	6
MindSensors	7
Team Use	8
Kauai Labs	9
Copperforge	10
Playing With Fusion	11
Studica	12
The Thrifty Bot	13
Redux Robotics	14
AndyMark	15
Vivid Hosting	16
Rezerve	17-255

API / Mesaj Tanımlayıcı

The API or Message Identifier is a 10-bit value that identifies a particular command or message type. These identifiers are unique for each Manufacturer + Device Type combination (so an API identifier that may be a "Voltage Set" for a Luminary Micro Motor Controller may be a "Status Get" for a CTR Electronics Motor Controller or Current Get for a CTR Power Distribution Module).

Mesaj tanımlayıcı 2 alt alana ayrılmıştır: 6 bit API Sınıfı ve 4 bit API Dizini-Index.

16.6.2 Protected Frames - Korumalı Çerçeveler

Aktüatör kontrol yeteneğini uygulayan FRC CAN Düğümleri (motor kontrolörleri, röleler, pnömatik kontrolörleri vb.), Robotun etkinleştirildiğini ve komutların ana robot kontrolöründen (yani roboRIO) geldiğini doğrulamak için bir yol uygulamalıdır.

16.6.3 Yayın Mesajları - Broadcast

Yayın mesajları, cihaz türü ve üretici alanları 0 olarak ayarlanarak tüm düğümlere gönderilen mesajlardır. Yayın mesajları için API Sınıfı 0'dır. Şu anda tanımlanmış yayın mesajları aşağıdaki tabloda gösterilmektedir:

Description - Açıklama	
Disable - Devre dışı bırak	0
System Halt - Sistem durdu	1
System Reset	2
Device Assign - Cihaz Atama	3
Device Query - Cihaz Sorgusu	4
Heartbeat - Kalp atışı	5
Sync	6
Update - Güncelleme	7
Firmware Sürümü	8
Enumerate - Numaralandır	9
System Resume - Sistem Devam Ettirme	10

Devices should disable immediately when receiving the Disable message (arbID 0). Implementation of other broadcast messages is optional.

16.6.4 FRC CAN Düğümleri için Gereksinimler

CAN Düğümlerinin FRC Sisteminde kullanım için kabul edilmesi için, aşağıdakileri yapmaları gerekir:

- Öngörülen FRC formatıyla eşleşen Arbitration ID Kimliklerini kullanarak iletişim kurun:
 - Geçerli, kullanılan bir CAN Cihaz Tipi (Tablo 1'e göre - CAN Cihaz Tipleri)
 - Geçerli, yayınlanmış bir Üretici Kimliği (Tablo 2'ye göre - CAN Üretici Kodları)
 - API Sınıf (ları) ve Dizin (leri) Cihaz üreticisi tarafından atanır ve belgelendirilir.
 - Cihaz tipinin birden fazla biriminin aynı ağda birlikte var olması amaçlanıyorsa, kullanıcı tarafından seçilebilen bir cihaz numarası.
- Yayın Mesajları bölümünde ayrıntılı olarak açıklanan minimum Yayın mesajı gereksinimlerini destekleyin.
- 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.

Description - Açıklama	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.

17.1 Git Sürüm Kontrolüne Giriş

Önemli: A more in-depth guide on Git is available on the [Git website](#).

Git, Linus Torvalds tarafından oluşturulan ve Linux çekirdeğini oluşturmak ve sürdürmekle de bilinen bir Dağıtılmış Sürüm Kontrol Sistemidir (VCS). Sürüm Kontrolü, geliştiriciler için kod değişikliklerini izlemek için bir sistemdir. Git Sürüm Kontrolünün avantajları şunlardır:

- Separation of testing environments into *branches*
- Geçmişi silmeden belirli bir *işlemeye* gitme yeteneği
- *Taahhütleri* çeşitli şekillerde, bunları birleştirmek de dahil olmak üzere yönetme yeteneği
- Çeşitli diğer özellikler, bkz. [here](#)

17.1.1 Önkoşullar

Önemli: Bu eğitici Windows işletim sistemini kullanır

Git'i aşağıdaki bağlantılardan indirip kurmanız gerekir:

- *Windows* <<https://git-scm.com/download/win>> _
- *macOS* <<https://git-scm.com/download/mac>> _
- *Linux* <<https://git-scm.com/download/linux>> _

Not: <https://www.google.com/search?q=adding+git+to+path> yolunuza Git eklemeniz gerekebilir

17.1.2 Git Kelime Bilgisi

Git revolves around several core data structures and commands:

- **Repository:** Kök dizindeki “.git ” klasörü dahil kodunuzun veri yapısı
- **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:** uzak depoyu yerel değişikliklerinizle güncelleyin
- **Pull:** Yerel deponuzu uzaktan değişikliklerle güncelleyin
- **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 Depo

Git deposu, bir projenin yapısını, geçmişini ve dosyalarını içeren bir veri yapısıdır.

Git depoları genellikle şunlardan oluşur:

- Bir .git klasörü. Bu klasör arşivle ilgili çeşitli bilgileri içerir.
- Bir .gitignore dosyası. Bu dosya, kaydettiğinizde dahil edilmesini *istemediğiniz* dosyaları veya dizinleri içerir.
- Dosyalar ve Klasörler. Bu, deponun ana içeriğidir.

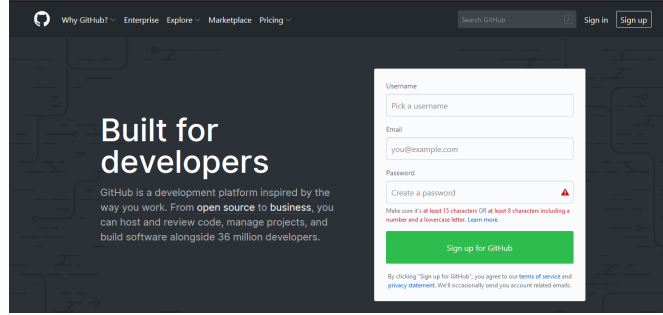
Depoyu oluşturma

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.

Not: There are various providers that can host repositories. [Gitlab](#) and [Bitbucket](#) are a few alternatives to Github.

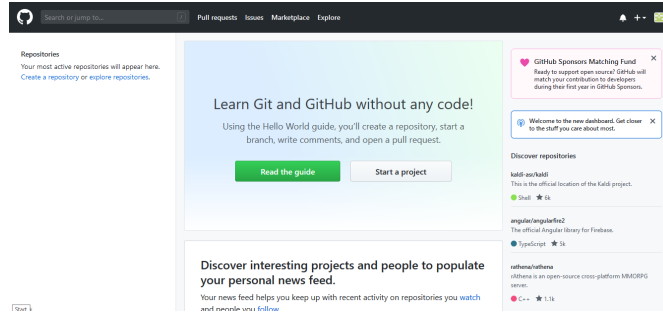
GitHub Hesabı Oluşturma

Go ahead and create a GitHub account by visiting the [website](#) and following the on-screen prompts.

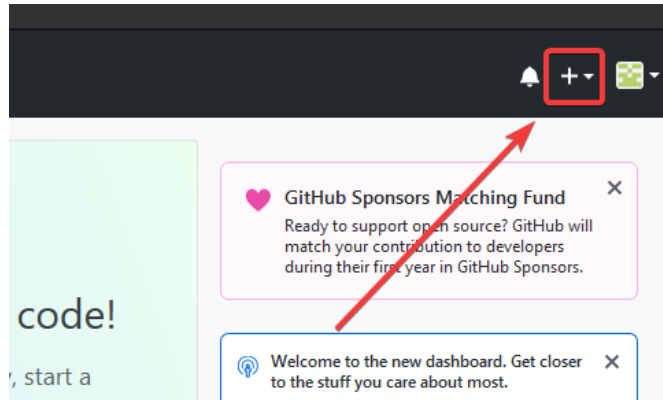


Yerel Yaratılış

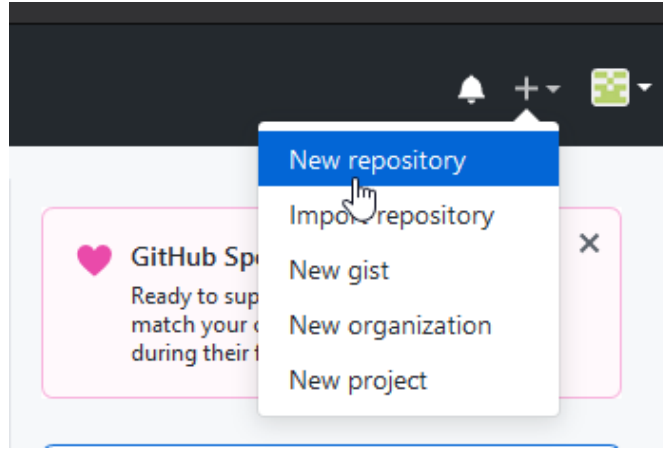
Hesabınızı oluşturup doğruladıktan sonra ana sayfayı ziyaret etmek isteyeceksiniz. Gösterilen resme benzer görünecek.



Sağ üstteki artı simgesini tıklayın.



Sonra “Yeni Depo” tıklayın



Uygun bilgileri doldurun ve ardından “Depo oluřtur” u tıklayın

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner: ExampleUser9007 / Repository name: ExampleRepo ✓

Great repository names are short and memorable. Need inspiration? How about [reimagined-palm-tree?](#)

Description (optional)

☒ Public
Anyone can see this repository. You choose who can commit.

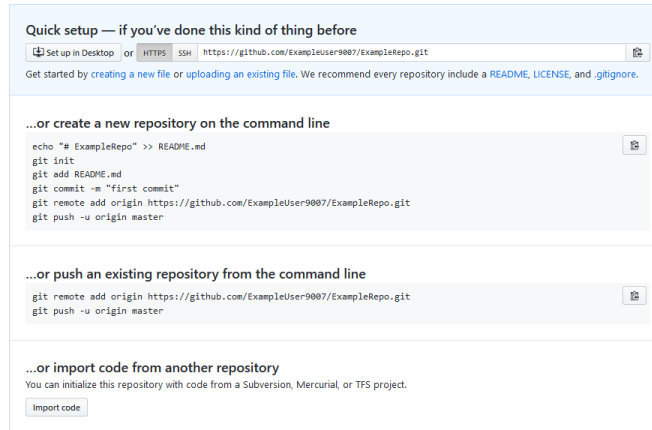
☐ Private
You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None ⓘ

Create repository

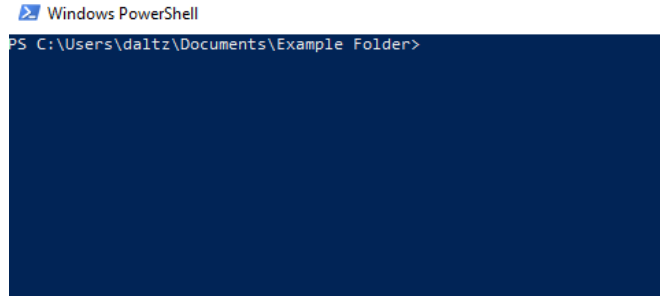
Buna benzer bir ekran görmelisiniz



Not: The keyboard shortcut `Ctrl+~` can be used to open a terminal in Visual Studio Code for

Windows.

Şimdi bir PowerShell penceresi açıp proje dizininize gitmek isteyeceksiniz. PowerShell ile ilgili mükemmel bir öğretici *burada* <<https://programminghistorian.org/en/lessons/intro-to-powershell>> __ bulunabilir. Lütfen alternatif işletim sistemlerinde bir terminalin nasıl açılacağını öğrenmek için arama motorunuza danışın.



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.

Not: Dosya yolunu değiştirin Replace the filepath "C:\Users\ExampleUser9007\Documents\Example Folder" repo'yu oluşturmak istediğiniz klasörle ve uzak URL'yi değiştirin. Önceki adımlarda oluşturduğunuz deponun URL'si ile ``https://github.com/ExampleUser9007/ExampleRepo.git``.

Empty Directory

```
> cd "C:\Users\ExampleUser9007\Documents\Example Folder"
> git init
Initialized empty Git repository in C:/Users/ExampleUser9007/Documents/Example Folder/.git/
↪.git/
> echo "# ExampleRepo" >> README.md
> git add README.md
> git commit -m "First commit"
[main (root-commit) fafafa] First commit
 1 file changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README.md
> git remote add origin https://github.com/ExampleUser9007/ExampleRepo.git
> git push -u origin main
```

Existing Project

```
> cd "C:\Users\ExampleUser9007\Documents\Example Folder"
> git init
Initialized empty Git repository in C:/Users/ExampleUser9007/Documents/Example Folder/
↪.git/
> git add .
> git commit -m "First commit"
[main (root-commit) fafafa] First commit
 1 file changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README.md
> git remote add origin https://github.com/ExampleUser9007/ExampleRepo.git
> git push -u origin main
```

17.1.4 Kaydetme

Depolar, öncelikle kayıtlardan oluşur. İşlemler, kaydedilmiş durumlar veya kodun *sürümleri*dir.

Önceki örnekte, README.md adlı bir dosya oluşturduk. Bu dosyayı favori metin düzenleyicinizde açın ve birkaç satırı düzenleyin. Dosyayı biraz düzelttikten sonra kaydedip kapatın. PowerShell'e gidin ve aşağıdaki komutları yazın.

```
> 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
```

Not: Writing good commit messages is a key part of a maintainable project. A guide on writing commit messages can be found [here](#).

Git Çekme

Not: Kullanıcı otomatik olarak mevcut çalışma dalıyla birleşmek istemediğinde `git fetch` kullanılabilir

Bu komut, uzak depodan geçmişini veya taahhütleri alır. Uzaktan kumanda, sahip olmadığınız işleri içerdiğinde, otomatik olarak birleştirmeye çalışacaktır. Bakınız: *ref:docs / yazılım / temel-programlama / git-başlarken: Birleştirme*.

Çalıştır: `git pull`

Git Ekle

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 Kaydetme

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.

Çalıştır: `git commit -m "mesajı buraya yazın"`

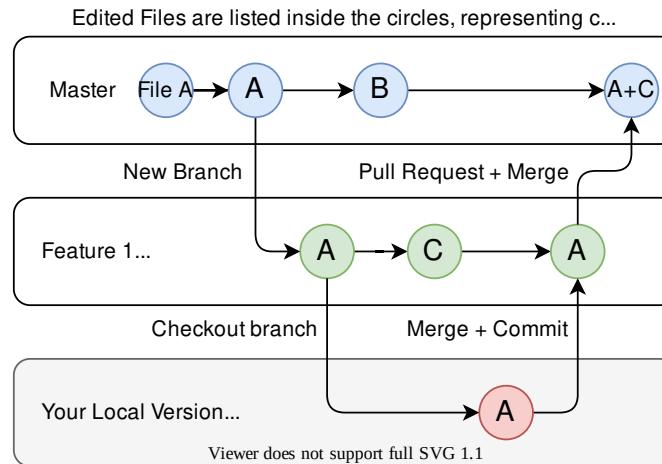
Git Push

Yerel değişikliklerinizi uzak bilgisayara (Bulut) yükleyin (itin)

Çalıştır: `git push`

17.1.5 Şubeler

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.



Yukarıdaki örnekte, FeatureB, FeatureA ile birleştirilmiştir. Bu, birleştirme denen şeydir. Değişiklikleri bir daldan diğerine “birleştiriyorsunuz”.

Şube Oluşturma

Çalıştır: `git şube şube adı` burada şube adı, oluşturulacak dalın adıdır. Yeni şube geçmiş, mevcut aktif şubeden oluşturulacaktır.

Şube Girmek

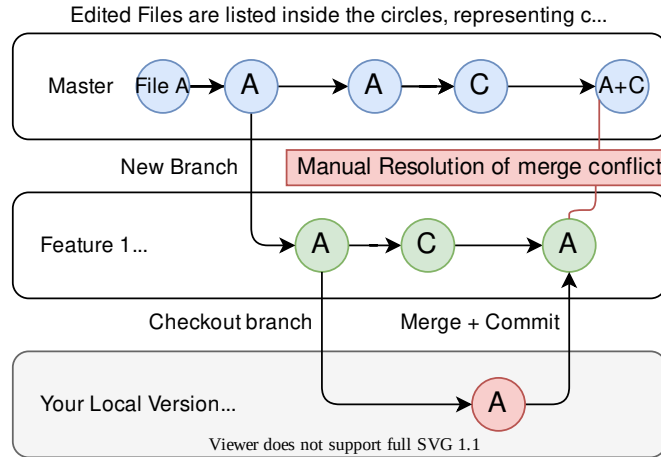
Bir şube oluşturulduktan sonra şubeye girmeniz gerekir.

Çalıştır: `git checkout şube adı` burada şube adı, daha önce oluşturulan daldır.

17.1.6 Birleştirme

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 Sıfırlama

Bazen geçmişin sıfırlanması veya bir işlemenin geri alınması gerekir. Bu, birden çok şekilde yapılabilir.

Kaydetmeyi Geri Döndürme

Not: Git, hangi şubeyi veya menşei seçmesi gerektiğini bilmediği için bir birleştirmeyi geri alamazsınız.

Geçmişi geri döndürmek için bir commit çalıştırmasına yol açan `git revert commit-id`. Commit ID'leri `git log` komutu kullanılarak gösterilebilir.

Kafayı Sıfırlama

Uyarı: Kafayı zorla sıfırlamak tehlikeli bir komuttur. Hedefin dışındaki tüm geçmişi kalıcı olarak siler. Uyarıldın!

Çalıştır: `git reset --hard commit-id`.

17.1.8 Çatallar

Çatallar dallara benzer şekilde tedavi edilebilir. Yukarı akışı (orijinal depo) orijinle (çatalı depo) birleştirebilirsiniz.

Bir Depoyu Klonlamak

Bir deponun zaten bir uzaktan kumandada yaratılmış ve depolanmış olması durumunda, onu kullanarak klonlayabilirsiniz.

```
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](#).

Çatal Güncelleme

1. Yukarı akışı ekleyin: `git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git`
2. Eklendiğini onaylayın: `git remote -v`
3. Yukarı akıştan değişiklikleri çekin: `git yukarı akış getir`
4. Değişiklikleri başlıkta birleştirin: `git merge upstream / upstream-branch-name`

17.1.9 Gitignore

Önemli: Ekiplerin, robot projelerinde bulunan “.gitignore ” dosyasını *değiştirmemeleri* son derece önemlidir. Bu, çevrimdışı dağıtımın çalışmamasına neden olabilir.

Bir .gitignore dosyası, genellikle git add ile otomatik olarak işlenmeyecek dosyaların listesi olarak kullanılır. Bu dosyada listelenen herhangi bir dosya veya dizin *işlenmeyecektir*. Ayrıca git status <<https://git-scm.com/docs/git-status>> _ ile görünmeyecekler.

Additional Information can be found [here](#).

Bir Klasörü Gizleme

Sonunda eğik çizgi ile gizlenecek klasörü içeren yeni bir satır eklemeniz yeterlidir

EX: `` dışlanacak dizin /``

Bir Dosyayı Gizlemek

Deponun köküne göre ön bekleyen dizinler de dahil olmak üzere, gizlenecek dosyanın adını içeren yeni bir satır ekleyin.

EX: dizin / dosyadan-gizlenecek.txt

EX: dosya-to-hide2.txt

17.1.10 ek bilgi

Resmi git <<https://git-scm.com/docs/gittutorial>> __ web sitesinde çok daha ayrıntılı bir eğitim bulunabilir.

Yaygın hataları düzeltmek için bir kılavuz, git [uçuş kuralları](#) havuzunda bulunabilir.

17.2 C ++ Ünite Kitablığı

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 Ünite Kitaplığını Kullanma

Birim kitaplığı, yalnızca başlık içeren bir kitaplıktır. Kullanmak istediğiniz birimler için kaynak dosyalarınıza ilgili başlığı eklemelisiniz. İşte mevcut birimlerin listesi.

```
#include <units/acceleration.h>
#include <units/angle.h>
#include <units/angular_acceleration.h>
#include <units/angular_velocity.h>
#include <units/area.h>
#include <units/capacitance.h>
#include <units/charge.h>
#include <units/concentration.h>
#include <units/conductance.h>
#include <units/current.h>
#include <units/curvature.h>
#include <units/data.h>
#include <units/data_transfer_rate.h>
#include <units/density.h>
#include <units/dimensionless.h>
#include <units/energy.h>
#include <units/force.h>
#include <units/frequency.h>
#include <units/illuminance.h>
#include <units/impedance.h>
#include <units/inductance.h>
#include <units/length.h>
#include <units/luminous_flux.h>
#include <units/luminous_intensity.h>
#include <units/magnetic_field_strength.h>
#include <units/magnetic_flux.h>
#include <units/mass.h>
#include <units/moment_of_inertia.h>
#include <units/power.h>
#include <units/pressure.h>
#include <units/radiation.h>
#include <units/solid_angle.h>
#include <units/substance.h>
#include <units/temperature.h>
#include <units/time.h>
#include <units/torque.h>
#include <units/velocity.h>
#include <units/voltage.h>
#include <units/volume.h>
```

units/math.h başlığı, units::math::abs() gibi birime duyarlı işlevler sağlar.

Ünite Tipleri ve Taşıyıcı Sınıf Tipleri

C ++ ünite kitaplığı iki tür tür tanımına dayanır: birim türleri ve kapsayıcı türleri.

Ünite Tipleri

Birim türleri, herhangi bir gerçek depolanmış değer olmaksızın soyut bir birim kavramına karşılık gelir. Birim türleri, birimler kitaplığının temel “building block-yapı taşı”dır , az sayıda “basic-temel” birim türünden (örneğin `` metre `` , `` saniye `` vb.) yapıcı bir şekilde (`` compound_unit-birleşik_birim`` şablonu kullanılarak) tanımlanır.

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.

Container-Taşıyıcı Sınıf Türleri

Container-Taşıyıcı Sınıf türleri, bir birime göre boyutlandırılmış gerçek bir miktara karşılık gelir - yani, bunlar gerçekte sayısal değeri tutan şeydir. Container-Taşıyıcı Sınıf türleri, `unit_t` şablonuna sahip birim türlerinden oluşturulur. Çoğu birim türünün son eki `_t` ile aynı adı taşıyan karşılık gelen bir kap türü vardır - örneğin, birim türü `units::meter`, konteyner türü `units::meter_t` ye karşılık gelir.

Bir birimin belirli bir miktarı kullanıldığında (değişken veya yöntem parametresi olarak), bu, Container-Taşıyıcı Sınıf türünün bir örneği olacaktır. Varsayılan olarak, Container-Taşıyıcı Sınıf türleri gerçek değeri `` double``-çift`` olarak depolar - ileri düzey kullanıcılar bunu manuel olarak `unit_t` şablonunu çağırarak değiştirebilir.

Birim ve Container-Taşıyıcı Sınıf türler türlerinin tam listesi *documentation* <<https://github.com/nholthaus/units#namespaces>> _ belgelerinde bulunabilir.

Ünite Örnekleri Oluşturma

Belirli bir ünitenin örneğini oluşturmak için, container türünün bir örneğini oluşturuyoruz:

```
// The variable speed has a value of 5 meters per second.
units::meter_per_second_t speed{5.0};
```

Alternatif olarak, ünite kitaplığı, daha yaygın kap türlerinden bazıları için tanımlanmış `type literals` sahiptir. Bunlar, bir birimi daha kısa ve öz bir şekilde tanımlamak için tür çıkarımı ile birlikte kullanılabilir:

```
// The variable speed has a value of 5 meters per second.
auto speed = 5_mps;
```

Türler birbirleri arasında dönüştürülebildiği sürece, birimler başka bir container türünün değeri kullanılarak da başlatılabilir. Örneğin, bir `foot_t` değerinden bir `meter_t` değeri oluşturulabilir.

```
auto feet = 6_ft;
units::meter_t meters{feet};
```

Aslında, dönüştürülebilir birim türlerini temsil eden tüm kap türleri *implicitly convertible*. Bu nedenle, aşağıdakiler tamamen doğrudur:

```
units::meter_t distance = 6_ft;
```

Kısacası, kodumuzun herhangi bir yerinde * herhangi bir -any other* uzunluk birimi * yerine * herhangi-any * uzunluk birimini kullanabiliriz; birimler kitaplığı bizim için otomatik olarak doğru dönüşümü gerçekleştirecektir.

Hesap Yapma Ünitesi

Container-Taşıyıcı türleri, işlemin * boyutsal olarak * sağlam olması gerektiği ek koşuluyla, temel alınan veri türlerinin tüm sıradan aritmetik işlemlerini destekler. Bu nedenle, ekleme her zaman iki uyumlu container türünde yapılmalıdır:

```
// 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;
```

Çarpma, herhangi bir kap türü çiftinde gerçekleştirilebilir ve bir bileşik birimin kap türünü verir:

Not: Bir hesaplama bir bileşik birim türü verdiğinde, bu tür yalnızca sonuç türü açıkça belirtilmişse işlem noktasında geçerlilik açısından kontrol edilecektir. ``auto-otomatik`` kullanılırsa, bu kontrol yapılmayacaktır. Örneğin, mesafeyi zamana böldüğümüzde, sonucun gerçekten bir hız olmasını sağlamak isteyebiliriz (yani, ``units::meter_per_second_t``). İade türünün ``auto-otomatik`` olarak bildirilmesi durumunda bu kontrol yapılmayacaktır.

```
// 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
```

Fonksiyonlar

Bazı standart işlevler (``kelepçe-clamp`` gibi), aritmetik işlemlerin gerçekleştirilebileceği herhangi bir türü kabul edecek şekilde şablonlanmıştır. container-konteyner türleri olarak depolanan miktarlar bu işlevlerle sorunsuz çalışacaktır.

Bununla birlikte, diğer standart işlevleri yalnızca sıradan sayısal türlerde çalışır (örneğin, double). Birim kitaplığının ``units::math`` ad alanı, birimleri kabul eden bu işlevlerin birçoğu için sarmalayıcılar içerir. Bu tür işlevlere örnek olarak ``sqrt``, ``pow``, vb. verilebilir.

```
auto area = 36_sq_m;
units::meter_t sideLength = units::math::sqrt(area);
```

Ünite/Birim Sargısının Çıkarılması

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 Kodundaki Ünite Kitaplığı Örneği

WPILib'in yeni özelliklerindeki yöntemler için çeşitli argümanlar (ör *kinematics*) Birimler kitaplığını kullanır. İşte bir örnek *yörüngeyi örnekleme*.

```
// 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);
```

Bazı WPILib sınıfları, birden çok birim türü seçeneğiyle doğal olarak çalışabilen nesneleri temsil eder - örneğin, bir hareket profili doğrusal mesafede (ör. Metre) veya açısal mesafede (ör. Radyan) çalışabilir. Bu tür sınıflar için, birim türü şablon parametresi olarak gereklidir:

```
// 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};
```

Daha ayrıntılı belgeler için lütfen birim kitaplığı için resmi `GitHub sayfasını <<https://github.com/nholthaus/units>>` _ ziyaret edin.

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 at the [Units javadoc](#) to browse the available units and their types.
- [Measure](#), which is used to tag a value with a unit.

Not: 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 calculate 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);
```

Uyari: 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](#)

<code>mut_plus(double, Unit)</code>	Increments the internal value by an amount in another unit. The internal unit will stay the same
<code>mut_plus(Measure)</code>	Increments the internal value by another measurement. The internal unit will stay the same
<code>mut_minus(double, Unit)</code>	Decrements the internal value by an amount in another unit. The internal unit will stay the same
<code>mut_minus(Measure)</code>	Decrements the internal value by another measurement. The internal unit will stay the same
<code>mut_times(double)</code>	Multiplies the internal value by a scalar
<code>mut_divide(double)</code>	Divides the internal value by a scalar
<code>mut_replace(double, Unit)</code>	Overrides the internal state and sets it to equal the given value and unit
<code>mut_replace(Measure)</code>	Overrides the internal state to make it identical to the given measurement
<code>mut_setMagnitude(double)</code>	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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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(
            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;
    }
}

```

Uyarı: 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)));
}
```

Not: 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);
    }
}
```

(sonraki sayfaya devam)

```

}

// 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);

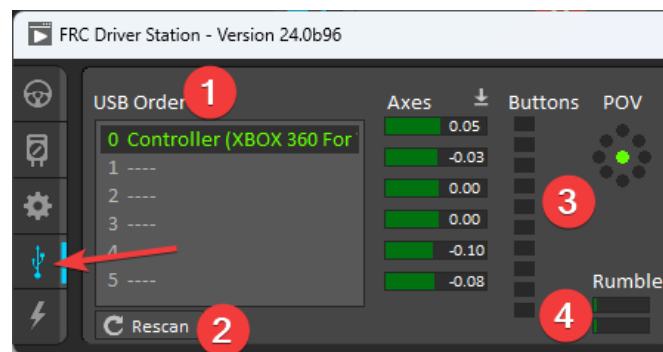
```

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: *Komutları Trigger-Tetikleyicilere Bağlama*.

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.

Not: 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.

Önemli: 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.

Önemli: 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
```

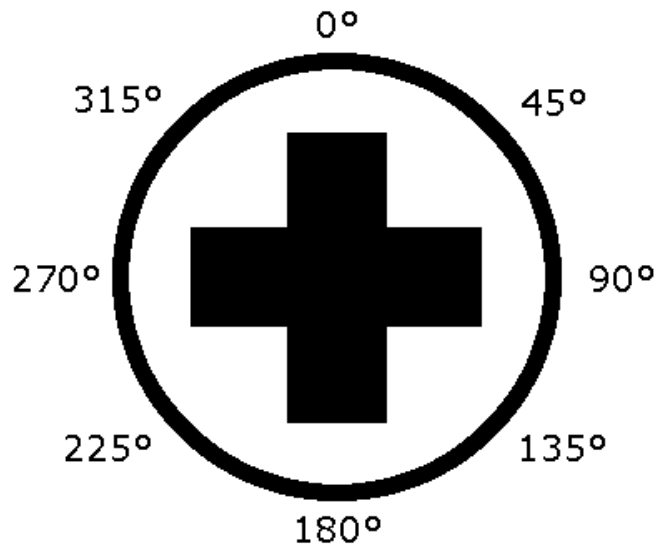
PYTHON

```
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)`.

Önemli: 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

Not: Usage such as the following is for code not using the command-based framework. For button usage in the command-based framework, see [Komutları Trigger-Tetikleyicilere Bağlama](#).

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
```

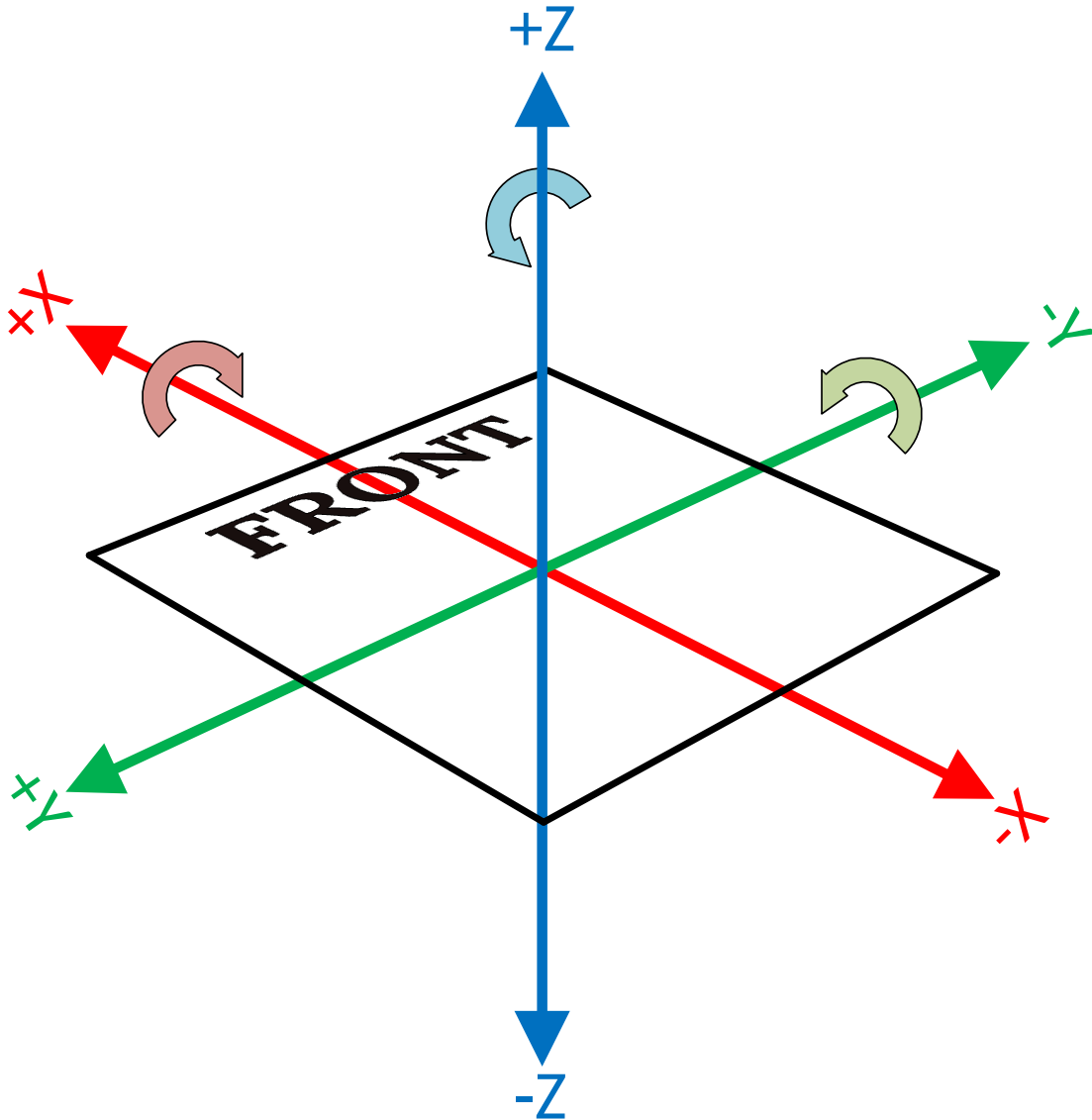
17.5 Coordinate System

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.

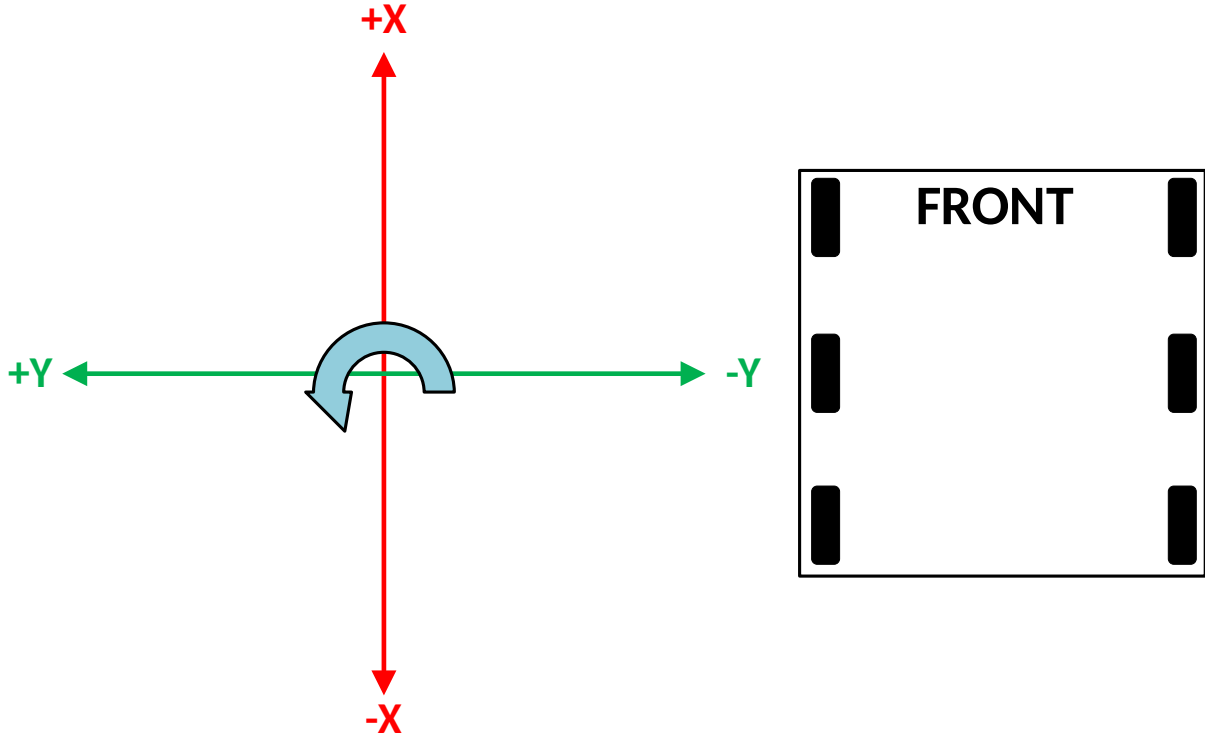
17.5.1 WPILib coordinate system

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.



Şekil 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.



Şekil 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.

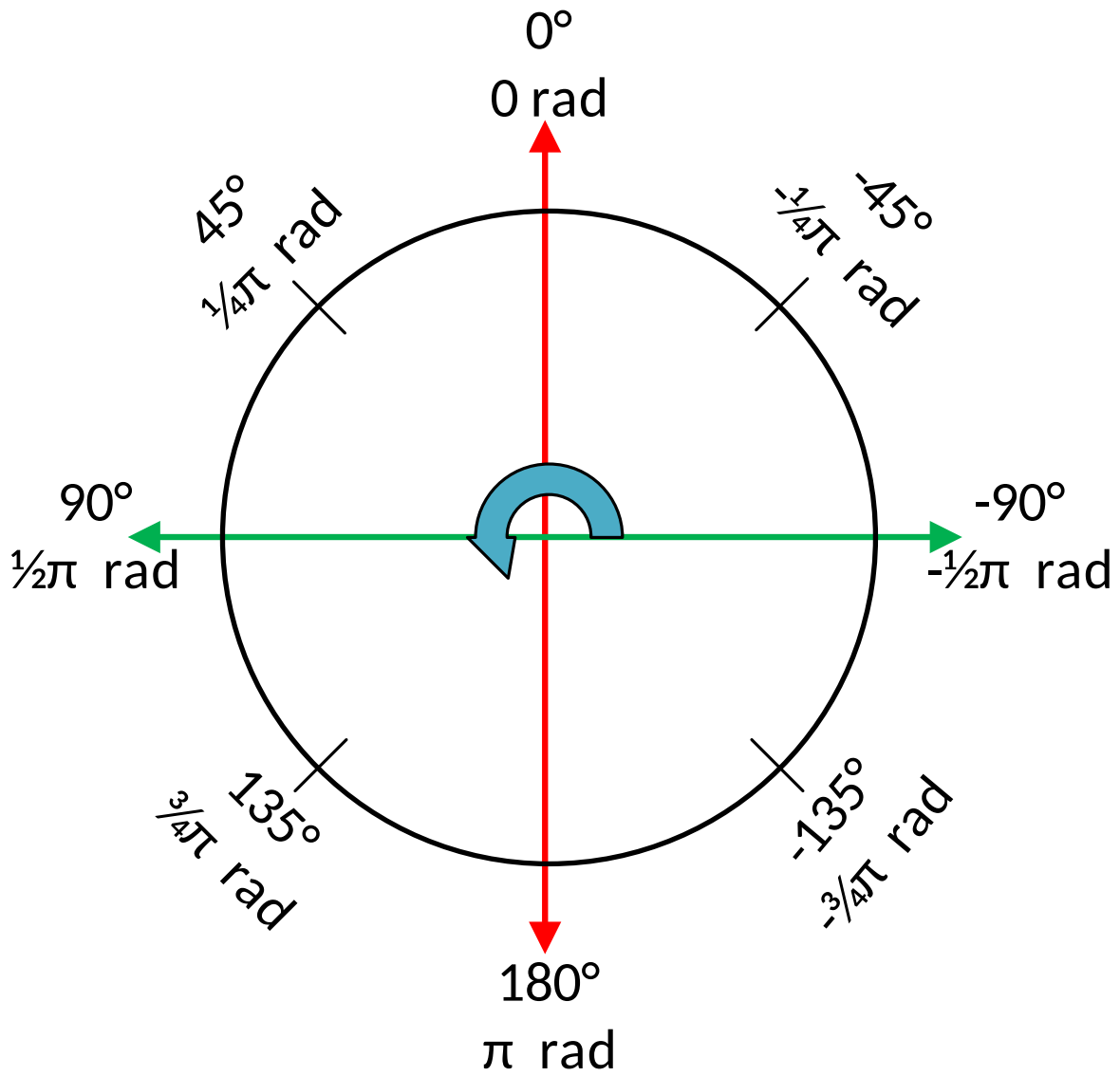
The figure above shows the unit circle with common angles labeled in degrees ($^\circ$) and radians (rad). Notice that rotation to the right is negative, and the range for the whole unit circle is -180° to 180° ($-\pi$ radians to π radians).

Not: 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.

Uyarı: 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.

Uyarı: 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



Şekil 3: Unit circle with common angles

180° they read 181°, not -179°. Some sensors have configuration settings where you can choose their wrapping behavior and range, while others need to be handled in your code. Careful attention should be paid to make sure sensor readings are consistent and your control loop handles wrapping in the same way as your sensor.

17.5.3 Joystick and controller coordinate system

Joysticks, including the sticks on controllers, don't use the same NWU coordinate system. They use the NED (North-East-Down) convention, where the positive X axis points ahead, the positive Y axis points right, and the positive Z axis points down. When viewed with each positive axis pointing toward you, counter-clockwise (CCW) is a positive value and clockwise (CW) is a negative value.

It's important to note that joystick input values are rotations around an axis, not translations. In practical terms, this means:

- pushing forward on the joystick (toward the positive X axis) is a CW rotation around the Y axis, so you get a negative Y value.
- pushing to the right (toward the positive Y axis) is a CCW rotation around the X axis, so you get a positive X value.
- twisting the joystick CW (toward the positive Y axis) is a CCW rotation around the Z axis, so you get a positive Z value.

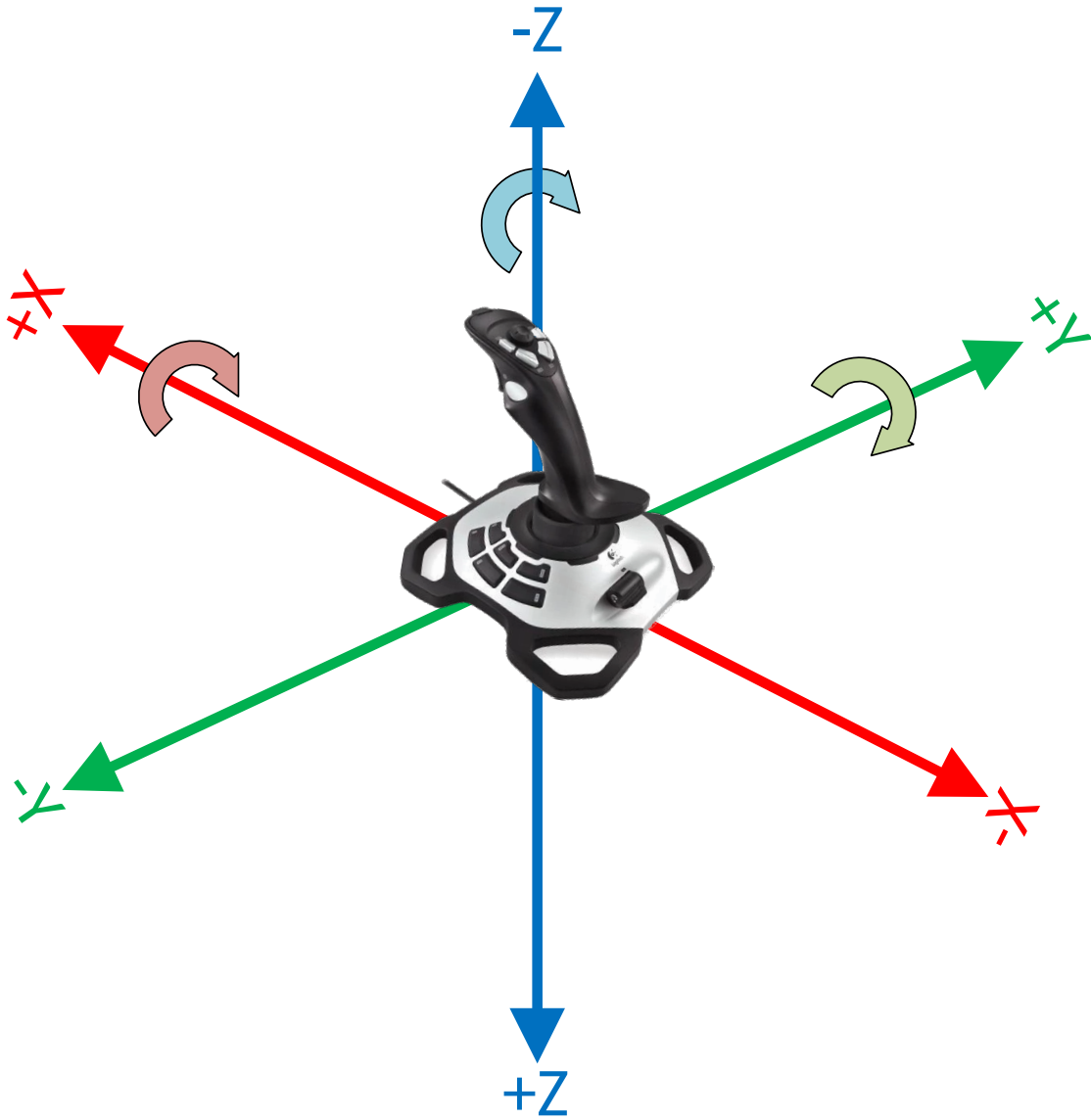
17.5.4 Using Joystick and controller input to drive a robot

You may have noticed, the coordinate system used by WPILib for the robot is not the same as the coordinate system used for joysticks and controllers. Care needs to be taken to understand the difference, and properly pass driver input to the drive subsystem.

Differential drivetrain example

Differential drivetrains are non-holonomic, which means the robot drivetrain cannot move side-to-side (strafe). This type of drivetrain can move forward and backward along the X axis, and rotate around the Z axis. Consider a common arcade drive scheme using a single joystick where the driver pushes the joystick forward/backward for forward/backward robot movement, and push the joystick left/right to rotate the robot left/right.

The code snippet below uses the `DifferentialDrive` and `Joystick` classes to drive the robot with the arcade scheme described above. `DifferentialDrive` uses the robot coordinate system defined above, and `Joystick` uses the joystick coordinate system.



Şekil 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

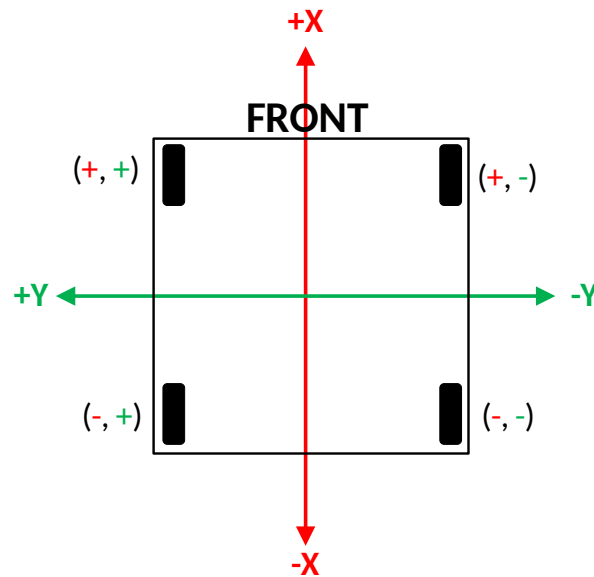
```
# Drive using the X, Y, and Z axes of the joystick.  
speeds = ChassisSpeeds(-self.stick.getY(), -self.stick.getX(), -self.stick.getZ())
```

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.



Şekil 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')

Uyarı: 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

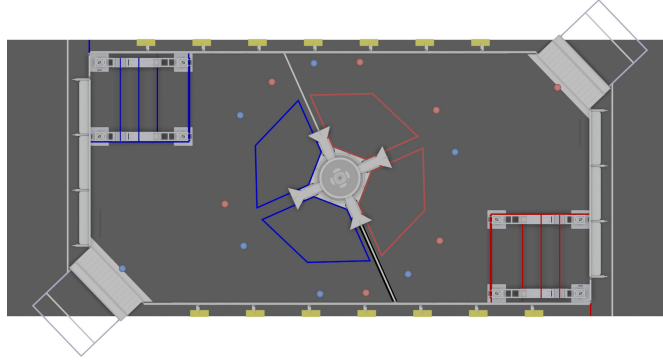
The field coordinate system (or global coordinate system) is an absolute coordinate system where a point on the field is designated as the origin. Two common uses of the field coordinate system will be explored in this document:

- 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.

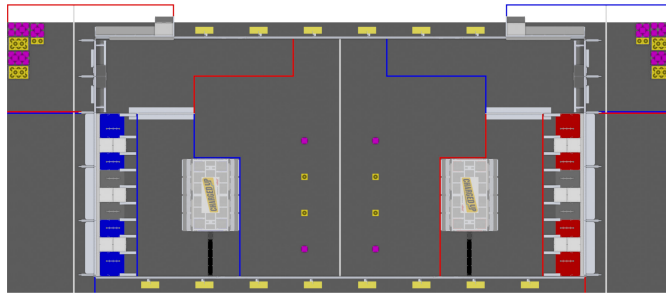


Şekil 6: Rotated field from RAPID REACT in 2022¹

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>



Şekil 7: Mirrored field from CHARGED UP in 2023²

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.

Uyarı: 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.

Not: At competition events, the FMS will automatically report your Team Station and alliance color. When you are not connected to an FMS, you can choose your Team Station and alliance color on the Driver Station *Operation Tab*.

Always blue origin

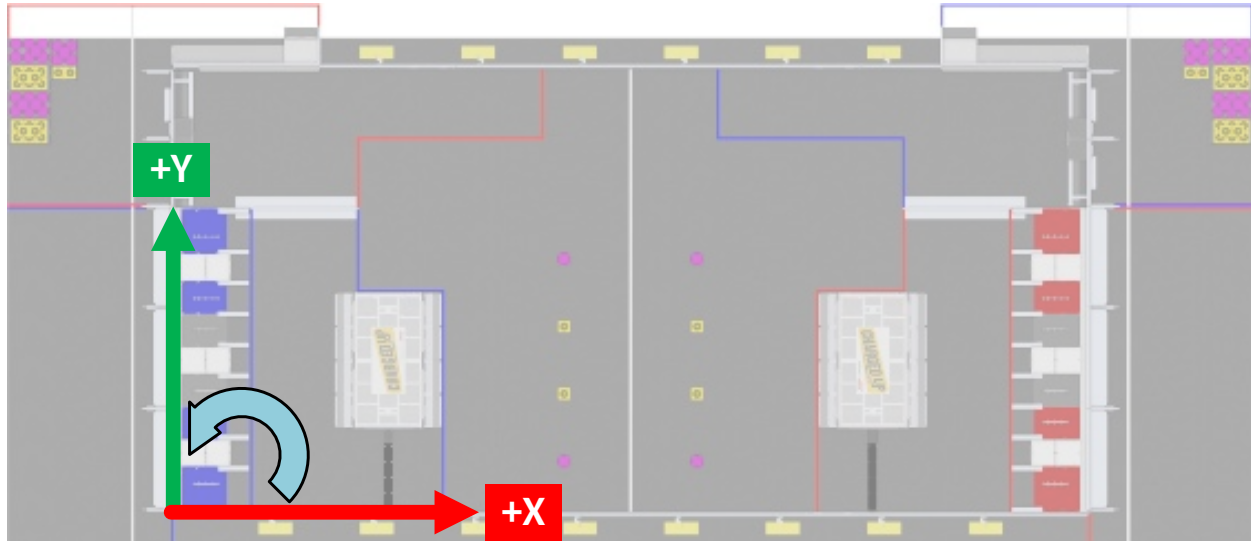
You may choose to define the origin of the field on the blue side, and keep it there regardless of your alliance color. With this solution, positive x-axis points away from the blue alliance wall.

Some advantages to this approach are:

- Pose estimation with AprilTags is simplified. AprilTags throughout the field are unique. If you keep the coordinate system the same regardless of alliance, there is no need for special logic to deal with the location of AprilTags on the field relative to your alliance.
- Many of the tools and libraries used in FRC follow this convention. Some of the tools include: PathPlanner, Choreo, and the ShuffleBoard and Glass Field2d widget.

In order to use this approach for field oriented driving, driver input needs to consider the alliance color. When your alliance is red and the driver is standing behind the red alliance wall, they will want the robot to move downfield toward the blue alliance wall. However, when your alliance is blue, the driver will want the robot to go downfield toward the red alliance wall.

A simple way to deal with field oriented driving is to check the alliance color reported by the *DriverStation* class, and invert the driver's controls based on the alliance. As noted above, your alliance color can change so it needs to be checked on every robot iteration.



Şekil 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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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

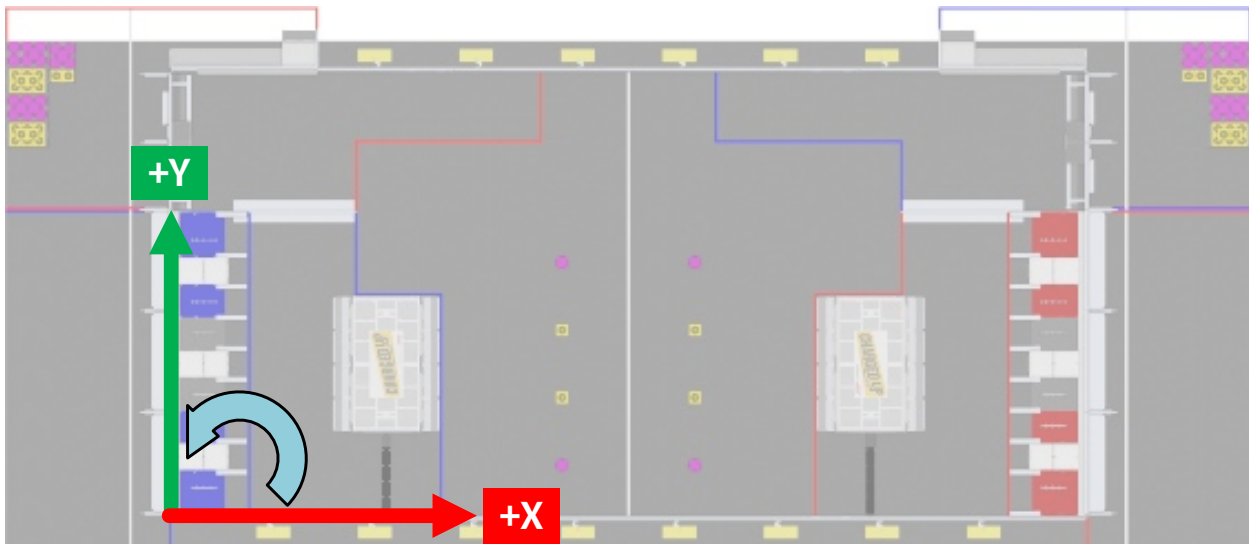
# Create field relative ChassisSpeeds for controlling Swerve
chassis_speeds = wpilib.ChassisSpeeds.FromFieldRelativeSpeeds(
    xSpeed * invert, ySpeed * invert, zRotation, self.imu.GetAngle()
)

# 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.

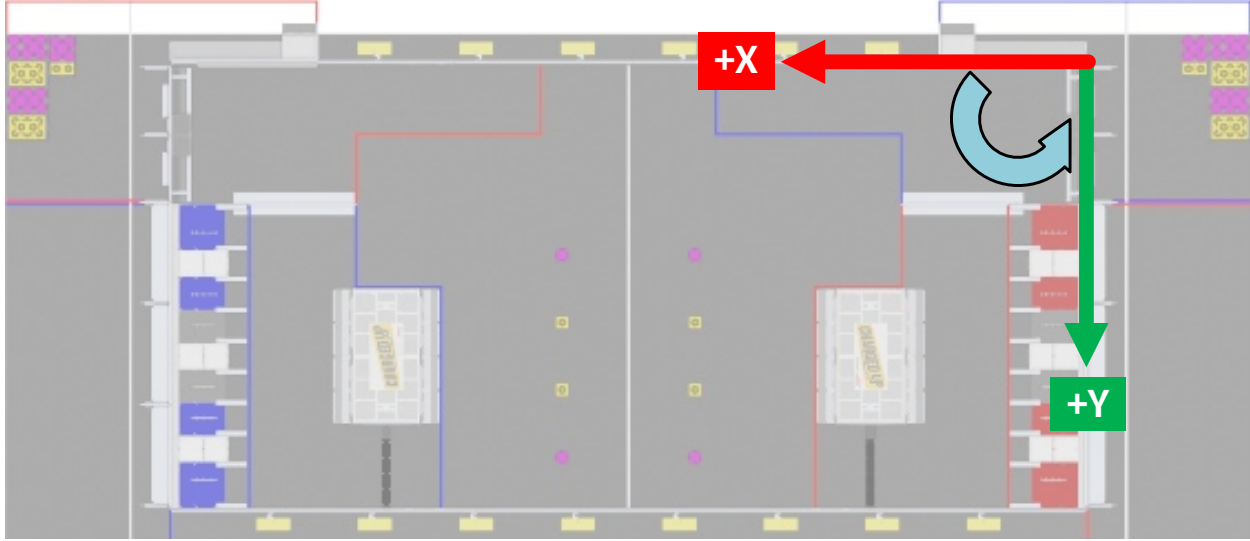
When you are on the blue alliance, your origin looks like this:



Şekil 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.



Şekil 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't
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

→ 't already exist
    wpilib.Preferences.initDouble(
        Constants.kArmPositionKey, self.armSetpointDegrees
    )
    wpilib.Preferences.initDouble(Constants.kArmPKey, self.armKp)

```

Preferences are stored using a name, the key. It's helpful to store the key in a constant, like `kArmPositionKey` and `kArmPKey` in the code above to avoid typing it multiple times and avoid typos. We also declare variables, `kArmKp` and `armPositionDeg` to hold the data retrieved from preferences.

In `robotInit`, each key is checked to see if it already exists in the Preferences database. The `containsKey` method takes one parameter, the key to check if data for that key already exists in the preferences database. If it doesn't exist, a default value is written. The `setDouble` method takes two parameters, the key to write and the data to write. There are similar methods for other data types like booleans, ints, and strings.

If using the Command Framework, this type of code could be placed in the constructor of a Subsystem or Command.

17.6.2 Reading Preferences

Java

```

public void loadPreferences() {
    // Read Preferences for Arm setpoint and kP on entering Teleop
    m_armSetpointDegrees = Preferences.getDouble(Constants.kArmPositionKey, m_
→ armSetpointDegrees);
    if (m_armKp != Preferences.getDouble(Constants.kArmPKey, m_armKp)) {
        m_armKp = Preferences.getDouble(Constants.kArmPKey, m_armKp);
        m_controller.setP(m_armKp);
    }
}

```

C++

```

void Arm::LoadPreferences() {
    // Read Preferences for Arm setpoint and kP on entering Teleop
    m_armSetpoint = units::degree_t{
        frc::Preferences::GetDouble(kArmPositionKey, m_armSetpoint.value());
    };
    if (m_armKp != frc::Preferences::GetDouble(kArmPKey, m_armKp)) {
        m_armKp = frc::Preferences::GetDouble(kArmPKey, m_armKp);
        m_controller.SetP(m_armKp);
    }
}

```

Python

```
def loadPreferences(self):
    # Read Preferences for Arm setpoint and kP on entering Teleop
    self.armSetpointDegrees = wpilib.Preferences.getDouble(
        Constants.kArmPositionKey, self.armSetpointDegrees
    )
    if self.armKp != wpilib.Preferences.getDouble(Constants.kArmPKey, self.armKp):
        self.armKp = wpilib.Preferences.getDouble(Constants.kArmPKey, self.armKp)
        self.controller.setP(self.armKp)
```

Reading a preference is easy. The `getDouble` method takes two parameters, the key to read, and a default value to use in case the preference doesn't exist. There are similar methods for other data types like booleans, ints, and strings.

Depending on the data that is stored in preferences, you can use it when you read it, such as the proportional constant above. Or you can store it in a variable and use it later, such as the setpoint, which is used in `telopPeriodic` below.

Java

```
@Override
public void teleopPeriodic() {
    if (m_joystick.getTrigger()) {
        // Here, we run PID control like normal.
        m_arm.reachSetpoint();
    } else {
        // Otherwise, we disable the motor.
        m_arm.stop();
    }
}
```

```
/** Run the control loop to reach and maintain the setpoint from the preferences. */
public void reachSetpoint() {
    var pidOutput =
        m_controller.calculate(
            m_encoder.getDistance(), Units.degreesToRadians(m_armSetpointDegrees));
    m_motor.setVoltage(pidOutput);
}
```

C++

```
void Robot::TeleopPeriodic() {
    if (m_joystick.GetTrigger()) {
        // Here, we run PID control like normal.
        m_arm.ReachSetpoint();
    } else {
        // Otherwise, we disable the motor.
        m_arm.Stop();
    }
}
```



```
void Arm::ReachSetpoint() {  
    // Here, we run PID control like normal, with a setpoint read from  
    // preferences in degrees.  
    double pidOutput = m_controller.Calculate(  
        m_encoder.GetDistance(), (units::radian_t{m_armSetpoint}.value()));  
    m_motor.SetVoltage(units::volt_t{pidOutput});  
}
```

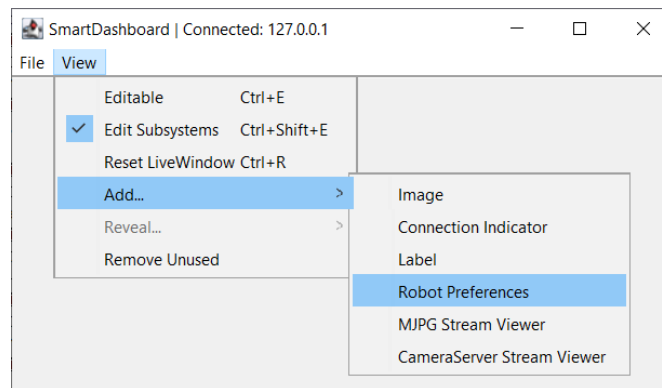
Python

```
def teleopPeriodic(self):  
    if self.joystick.getTrigger():  
        # Here, we run PID control like normal.  
        self.arm.reachSetpoint()  
    else:  
        # Otherwise, we disable the motor.  
        self.arm.stop()
```

```
def reachSetpoint(self):  
    pidOutput = self.controller.calculate(  
        self.encoder.getDistance(),  
        units.degreesToRadians(self.armSetpointDegrees),  
    )  
    self.motor.setVoltage(pidOutput)
```

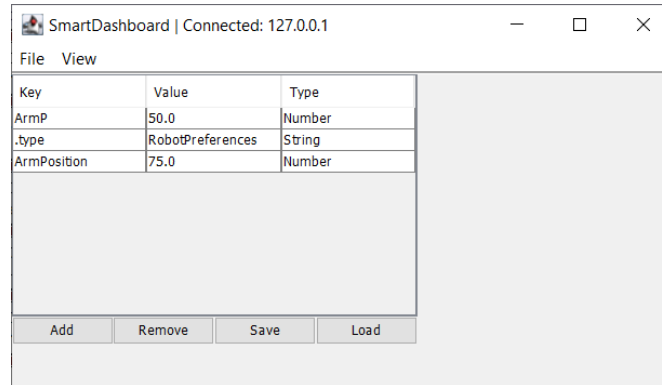
17.6.3 Using Preferences in SmartDashboard

Displaying Preferences in SmartDashboard



In the SmartDashboard, the Preferences display can be added to the display by selecting **View** then **Add...** then **Robot Preferences**. This reveals the contents of the preferences file stored in the roboRIO flash memory.

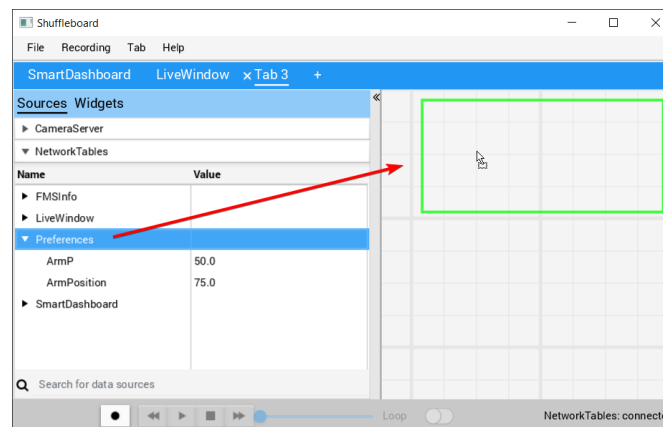
Editing Preferences in SmartDashboard



The values are shown here with the default values from the code. If the values need to be adjusted they can be edited here and saved.

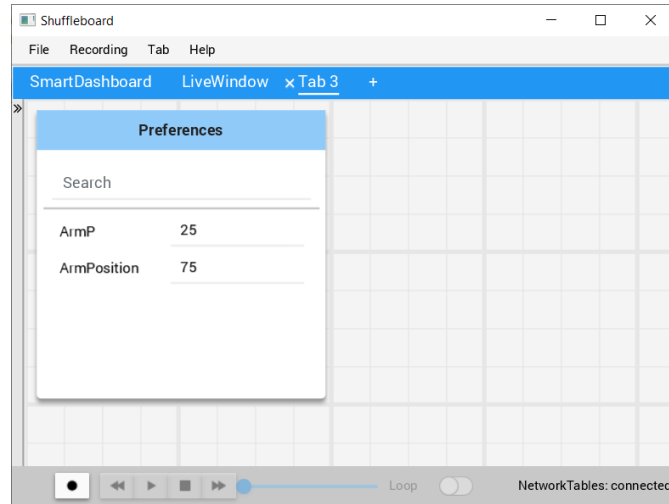
17.6.4 Using Preferences in Shuffleboard

Displaying Preferences in Shuffleboard



In Shuffleboard, the Preferences display can be added to the display by dragging the preferences field from the sources window. This reveals the contents of the preferences file stored in the roboRIO flash memory.

Editing Preferences in Shuffleboard

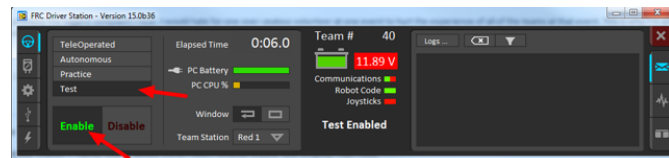


The values are shown here with the default values from the code. If the values need to be adjusted they can be edited here.

17.7 Using Test Mode

Test mode is designed to enable programmers to have a place to put code to verify that all systems on the robot are functioning. In each of the robot program templates there is a place to add test code to the robot.

17.7.1 Enabling Test Mode



Test mode on the robot can be enabled from the Driver Station just like autonomous or teleop. To enable test mode in the Driver Station, select the “Test” button and enable the robot. The test mode code will then run.

17.7.2 Adding Test mode code to your robot code

When in test mode, the `testInit` method is run once, and the `testPeriodic` method is run once per tick, in addition to `robotPeriodic`, similar to teleop and autonomous control modes.

Adding test mode can be as painless as calling your already written Teleop methods from Test. Or you can write special code to try out a new feature that is only run in Test mode, before integrating it into your teleop or autonomous code. You could even write code to move all motors and check all sensors to help the pit crew!

17.7.3 LiveWindow in Test Mode

Önemli: 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

Beklenmeyen bir hata oluştu.

Robotunuz beklenmedik bir hata verdiğinde, bu mesaj bazı konsol çıktılarında (Driver Station ya da RioLog) görünecektir. Büyük ihtimalle robotunuzun aniden durduğunu; ya da bir ihtimal hiç hareket etmediğini de fark edeceksiniz. Bu beklenmeyen hatalar *olağandışı durum* olarak adlandırılır.

Olağandışı bir durum oluşması, kodunuzda düzeltilmesi gereken bir ya da birden fazla hatanın olduğu anlamına gelir.

Bu makale, bu hataların bulunması ve düzeltilmesinde yer alan araç ve teknikleri incelemektedir.

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:

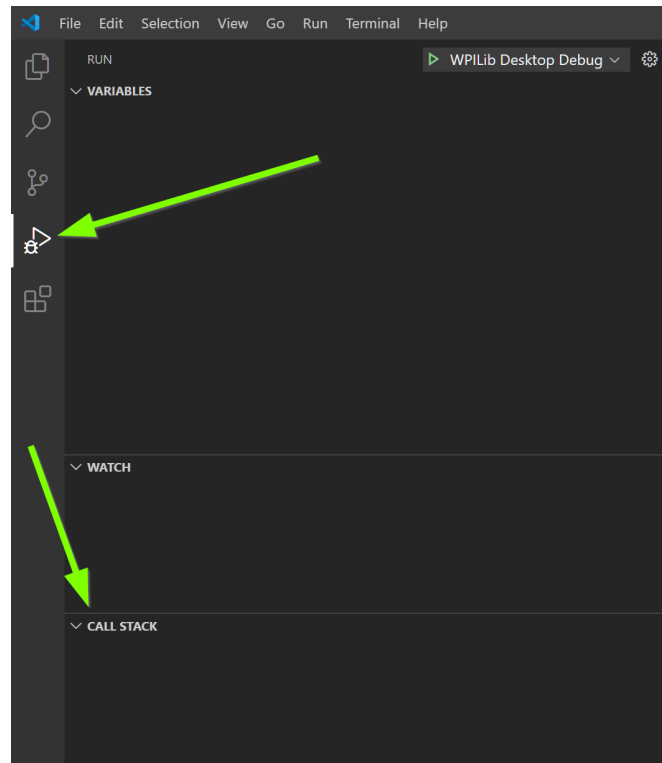
```
Error at frc.robot.Robot.buggyMethod(TooManyBugs.java:1138): Unhandled exception: ↪
↪java.lang.NullPointerException
    at frc.robot.Robot.buggyMethod(TooManyBugs.java:1138)
    at frc.robot.Robot.barInit(Bar.java:21)
    at frc.robot.Robot.fooInit(Foo.java:34)
    at frc.robot.Robot.robotInit(Robot.java:24)
    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.

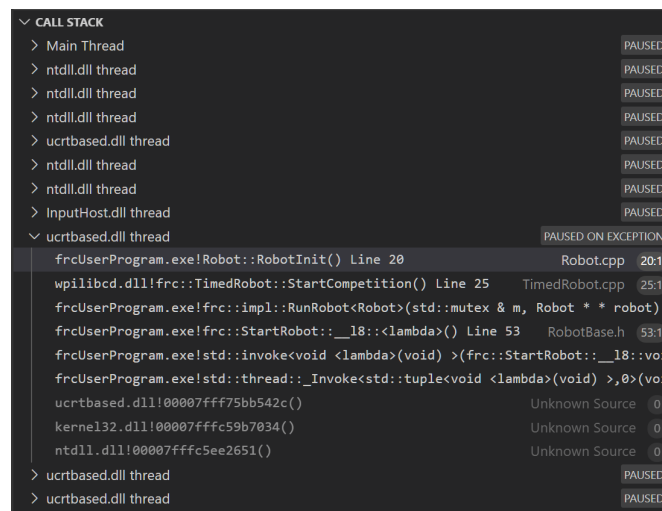
C++

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.

Not: 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.

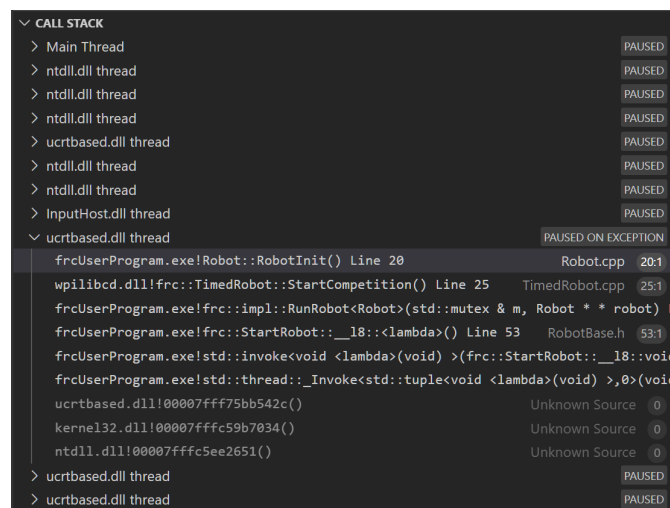
Alternatively, you can step through lines of code with the single step debugger, and stop when you hit line 23. Inspecting the `armMotorCtrl` object at that point would show that it is null.

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

```
18 int armLengthRatio;  
19 int elbowToWrist_in = 39;  
20 int shoulderToElbow_in = 0; //TODO  
21  
22 @Override  
23 public void robotInit() {  
24     armLengthRatio = elbowToWrist_in / shoulderToElbow_in;  
25 }
```

C++

```
17 class Robot : public frc::TimedRobot {  
18     public:  
19         void RobotInit() override {  
20             armLengthRatio = elbowToWrist_in / shoulderToElbow_in;  
21         }  
22  
23     private:  
24         int armLengthRatio;  
25         int elbowToWrist_in = 39;  
26         int shoulderToElbow_in = 0; //TODO  
27  
28 };
```

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.

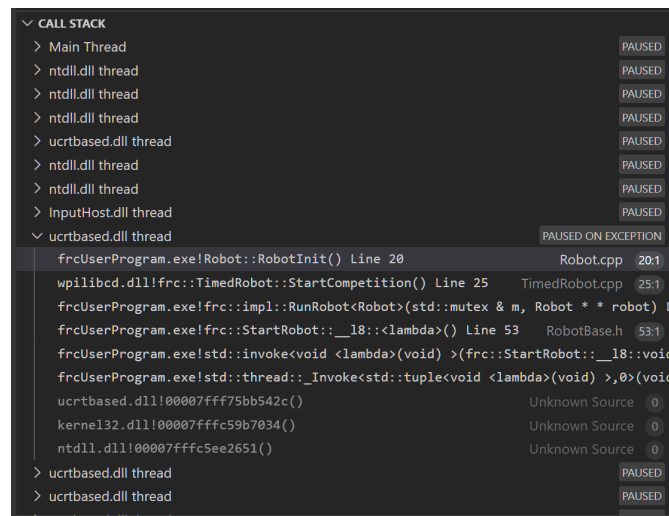
Alternatively, by running the single-step debugger and stopping on line 24, you could inspect the value of all variables to discover `shoulderToElbow_in` has a value of 0.

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

```
18 int armLengthRatio;  
19 int elbowToWrist_in = 39;  
20 int shoulderToElbow_in = 3;  
21  
22 @Override  
23 public void robotInit() {  
24  
25     armLengthRatio = elbowToWrist_in / shoulderToElbow_in;  
26  
27 }
```

C++

```
17 class Robot : public frc::TimedRobot {  
18     public:  
19     void RobotInit() override {  
20         armLengthRatio = elbowToWrist_in / shoulderToElbow_in;  
21     }  
22  
23     private:  
24         int armLengthRatio;  
25         int elbowToWrist_in = 39;  
26         int shoulderToElbow_in = 3  
27  
28 };
```

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

```

19 PWMSparkMax leftFrontMotor;
20 PWMSparkMax leftRearMotor;
21
22 @Override
23 public void robotInit() {
24     leftFrontMotor = new PWMSparkMax(0);
25     leftRearMotor = new PWMSparkMax(0);
26 }

```

C++

```

17 class Robot : public frc::TimedRobot {
18     public:
19         void RobotInit() override {
20             m_frontLeftMotor.Set(0.5);
21             m_rearLeftMotor.Set(0.25);
22         }
23
24     private:
25         frc::PWMVictorSPX m_frontLeftMotor{0};
26         frc::PWMVictorSPX m_rearLeftMotor{0};
27
28 };

```

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>

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

↳ (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]

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
at int frc::StartRobot<Robot>() + 0x3d4 [0x13c9c]
at __libc_start_main + 0x114 [0xb57ec580]
```

Error at RunRobot: Error: The robot program quit unexpectedly. This is usually due to [a code error](#).

The above stacktrace can help determine where the error occurred.

See <https://wpilib.org/stacktrace> for more information.

```
at void frc::impl::RunRobot<Robot>(wpi::priority_mutex&, Robot**) + 0x1c8
↳ [0x13838]
at int frc::StartRobot<Robot>() + 0x3d4 [0x13c9c]
at __libc_start_main + 0x114 [0xb57ec580]
```

terminate called after throwing an instance of 'frc::RuntimeError'

what(): PWM or DIO 0 previously allocated.

Location of the previous allocation:

```
at frc::PWM::PWM(int, bool) + 0x50 [0xb6f01b68]
at frc::PWMMotorController::PWMMotorController(std::basic_string_view<char,
↳ std::char_traits<char> >, int) + 0x70 [0xb6ef7d50]
at frc::PWMVictorSPX::PWMVictorSPX(int) + 0x3c [0xb6e9af1c]
at void frc::impl::RunRobot<Robot>(wpi::priority_mutex&, Robot**) + 0xa8
↳ [0x13718]
at int frc::StartRobot<Robot>() + 0x3d4 [0x13c9c]
at __libc_start_main + 0x114 [0xb57ec580]
```

Location of the current allocation:: Channel 0

The key thing to notice here is the string, PWM or DIO 0 previously allocated.. That string is your primary clue that something in code has incorrectly “doubled up” on pin 0 usage.

The message example above was generated on a roboRIO. If you are running in simulation, it might look different.

Fixing HAL Resource Already Allocated Issues

HAL: Resource already allocated are some of the most straightforward errors to fix. Just spend a bit of time looking at the electrical wiring on the robot, and compare that to what’s in code.

In the example, the left motor controllers are plugged into *PWM* ports 0 and 1. Therefore, corrected code would look like this:

JAVA

```
19 PWMSparkMax leftFrontMotor;
20 PWMSparkMax leftRearMotor;
21
22 @Override
23 public void robotInit() {
24
25     leftFrontMotor = new PWMSparkMax(0);
26     leftRearMotor = new PWMSparkMax(1);
27
28 }
```


C++

```
:lineno-start: 17

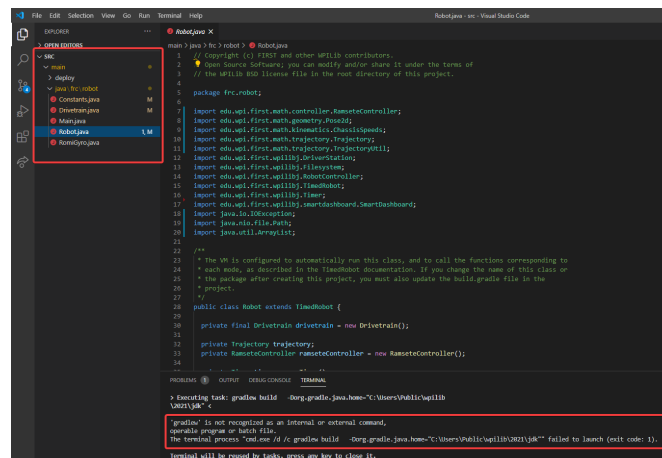
class Robot : public frc::TimedRobot {
public:
    void RobotInit() override {
        m_frontLeftMotor.Set(0.5);
        m_rearLeftMotor.Set(0.25);
    }

private:
    frc::PWMVictorSPX m_frontLeftMotor{0};
    frc::PWMVictorSPX m_rearLeftMotor{1};

};
```

gradlew is not recognized...

gradlew is not recognized as an internal or external command is a common error that can occur when the project or directory that you are currently in does not contain a gradlew file. This usually occurs when you open the wrong directory.



In the above screenshot, you can see that the left-hand sidebar does not contain many files. At a minimum, VS Code needs a couple of files to properly build and deploy your project.

- gradlew
- build.gradle
- gradlew.bat

If you do not see any one of the above files in your project directory, then you have two possible causes.

- A corrupt or bad project.
- You are in the wrong directory.

Fixing gradlew is not recognized...

gradlew is not recognized... is a fairly easy problem to fix. First identify the problem source:

Are you in the wrong directory? - Verify that the project directory is the correct directory and open this.

Is your project missing essential files? - This issue is more complex to solve. The recommended solution is to *recreate your project* and manually copy necessary code in.

17.9 Treating Functions as Data

Regardless of programming language, one of the first things anyone learns to do when programming a computer is to write a function (also known as a “method” or a “subroutine”). Functions are a fundamental part of organized code - writing functions lets us avoid duplicating the same piece of code over and over again. Instead of writing duplicated sections of code, we call a single function that contains the code we want to execute from multiple places (provided we named the function well, the function name is also easier to read than the code itself!). If the section of code needs some additional information about its surrounding context to run, we pass those to the function as “parameters”, and if it needs to yield something back to the rest of the code once it finishes, we call that a “return value” (together, the parameters and return value are called the function’s “signature”);

Sometimes, we need to pass functions from one part of the code to another part of the code. This might seem like a strange concept, if we’re used to thinking of functions as part of a class definition rather than objects in their own right. But at a basic level, functions are just data - in the same way we can store an integer or a double as a variable and pass it around our program, we can do the same thing with a function. A variable whose value is a function is called a “functional interface” in Java, and a “function pointer” or “functor” in C++.

17.9.1 Why Would We Want to Treat Functions as Data?

Typically, code that calls a function is coupled to (depends on) the definition of the function. While this occurs all the time, it becomes problematic when the code *calling* the function (for example, WPILib) is developed independently and without direct knowledge of the code that *defines* the function (for example, code from an FRC team). Sometimes we solve this challenge through the use of class interfaces, which define collections of data and functions that are meant to be used together. However, often we really only have a dependency on a *single function*, rather than on an *entire class*.

For example, WPILib offers several ways for users to execute certain code whenever a joystick button is pressed - one of the easiest and cleanest ways to do this is to allow the user to *pass a function* to one of the WPILib joystick methods. This way, the user only has to write the code that deals with the interesting and team-specific things (e.g., “move my robot arm”) and not the boring, error-prone, and universal thing (“properly read button inputs from a standard joystick”).

For another example, the *Command-based framework* is built on Command objects that refer to methods defined on various Subsystem classes. Many of the included Command types (such as InstantCommand and RunCommand) work with *any* function - not just functions associated with a single Subsystem. To support building commands generically, we need to support passing

functions from a Subsystem (which interacts with the hardware) to a Command (which interacts with the scheduler).

In these cases, we want to be able to pass a single function as a piece of data, as if it were a variable - it doesn't make sense to ask the user to provide an entire class, when we really just want them to give us a single appropriately-shaped function.

It's important that *passing* a function is not the same as *calling* a function. When we call a function, we execute the code inside of it and either receive a return value, cause some side-effects elsewhere in the code, or both. When we *pass* a function, nothing in particular happens *immediately*. Instead, by passing the function we are allowing some *other* code to call the function *in the future*. Seeing the name of a function in code does not always mean that the code in the function is being run!

Inside of code that passes a function, we will see some syntax that either refers to the name of an existing function in a special way, or else defines a new function to be passed inside of the call expression. The specific syntax needed (and the rules around it) depends on which programming language we are using.

17.9.2 Treating Functions as Data in Java

Java represents functions-as-data as instances of [functional interfaces](#). A “functional interface” is a special kind of class that has only a single method - since Java was originally designed strictly for object-oriented programming, it has no way of representing a single function detached from a class. Instead, it defines a particular group of classes that *only* represent single functions. Each type of function signature has its own functional interface, which is an interface with a single function definition of that signature.

This might sound complicated, but in the context of WPILib we don't really need to worry much about using the functional interfaces themselves - the code that does that is internal to WPILib. Instead, all we need to know is how to pass a function that we've written to a method that takes a functional interface as a parameter. For a simple example, consider the signature of `Commands.runOnce` (which creates an `InstantCommand` that, when scheduled, runs the given function once and then terminates):

Not: 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):

Not: 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.

Not: *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.
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

20 // You can use getTeamOrDefault(team) instead of getTeamNumber if you
21 // want to store a team number in this file.
22 team = project.frc.getTeamNumber()
23 debug = project.frc.getDebugOrDefault(false)
24
25 artifacts {
26     // First part is artifact name, 2nd is artifact type
27     // getTargetTypeClass is a shortcut to get the class type using a
28     ↪ string
29     frcJava(getArtifactTypeClass('FRCJavaArtifact')) {
30     }
31
32     // Static files artifact
33     frcStaticFileDeploy(getArtifactTypeClass('FileTreeArtifact')) {
34         files = project.fileTree('src/main/deploy')
35         directory = '/home/lvuser/deploy'
36     }
37 }
38 }
39 }
40 }

```

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](#).

Uyarı: 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.

Uyarı: 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() {
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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.

Uyarı: 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`.

Buradaki belgelere ek olarak, FRC | reg | Kontrol Sistemini ve yazılımı anlamaya yardımcı olacak ekipler.

18.1 Diğer Belgeler

Bu siteye ek olarak ekiplerin dokümantasyonu kontrol edebileceği birkaç yer daha var:

- *NI FRC Topluluk Belgeleri Bölümü* <<https://forums.ni.com/t5/FIRST-Robotics-Competition/bd-p/1014?profile.language=en&view=documents>> __
- [FIRST Inspires Teknik Kaynaklar Sayfası](#)
- [CTRE Software & Resources Page](#)
- [REV Robotics Documentation](#)

18.2 Forumlar

Sıkışmış mı? Belgelerde yanıtlanmayan bir sorunuz mu var? Resmi Destek şu forumlarda sağlanır:

- [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>> __ (kablolama, donanım ve Sürücü İstasyonu soruları)
- [FIRST Inspires Programming Forum](#) (C ++, Java için programlama soruları, veya LabVIEW)

18.3 CTRE Destek

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 Robotics Support

Support for REV Robotics components (SPARK MAX, Sensors, Pneumatic Hub, Power Distribution Hub, Radio Power Module) is provided via phone at [844-255-2267](tel: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 Hata Raporlama

Found a bug? Let us know by reporting it in the Issues section of the appropriate WPILibSuite project on GitHub: <https://github.com/wpilibsuite>

accelerometer

A common sensor used to measure acceleration in one or more axis.

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.

autonom

The first phase of each match is called Autonomous (auto) and consists of the robot's running pre-programmed instructions.

back-EMF

In electric motors, the force generated by the interaction of spinning magnets in a coil of wire which opposes spinning motion.

boolean

A form of data with only two possible values (true or false), intended to represent the two truth values of logic and Boolean algebra.

call stack

A specially-organized region of memory which helps the program keep track of what function it is in. As each function calls another, the call point is recorded and added to the top of the structure, forming a "stack" of references. Additionally, local variables will also be stored in this stack. See [call stack](#) on Wikipedia for more info.

CAD

Computer-Aided Design - software used to design an accurate model of an object. For FRC this is often used to design the robot to get accurate measurements and aid construction.

CAM

Computer-Aided Manufacturing - the use of software to control machine tools in the manufacturing of work pieces.

CAN

Controller Area Network - message-based protocol designed to allow microcontrollers

and devices to communicate with each other's applications without a host computer.

CD

Chief Delphi - [FRC team 47](#) inspired a popular community driven [forum](#) that today serves as an unofficial discussion hub for all things FRC.

central limit theorem

A core concept in probability which states that when many independent variables are added up, the result tends to look like a "normal" (or Gaussian) distribution, regardless of whether the independent variables themselves are normally distributed. See [Central Limit Theorem](#) on Wikipedia for more info.

CIM

CCL Industrial Motor, Limited - [Chiaphua Components Limited](#) is the company that made the commonly used, relatively powerful, brushed motor.

Classical Mechanics

The branch of physics which studies and describes the motion of relatively large, relatively slow objects. See [Classical Mechanics](#) on Wikipedia for more info.

COTS

Commercial off the shelf - a standard (i.e. not custom order) part commonly available from a vendor to all teams for purchase.

composition

A formal software term for building (or "composing") software entities out of smaller component entities. See [object composition](#) on Wikipedia for more info.

CRTP

Continuously Recurring Template Pattern - A software idiom in which a class X' derives from a class template instantiation using X' itself as a template argument. See [CRTP](#) on Wikipedia for more info.

CSA

Control Systems Advisor - FRC volunteer position that assists teams with Robot Control System-related issues.

CTRE

[Cross the Road Electronics LLC](#) - is an engineering design, software development and electronics manufacturer based outside of Detroit in Macomb, MI. They primarily focus on high-performing, high-quality electronics communication, motor control, and control system products for FIRST teams and the EV industry. CTR Electronics was founded in 2006 by two FRC mentors who met through their robotics team: Mike Copioli and Omar Zrien and is staffed largely by FRC alumni, and active volunteers & mentors.

C ++

One of the four officially supported programming languages.

declarative programming

A style of software which focuses on describing *what* a program should do, rather than *how* it gets done. See [declarative programming](#) on Wikipedia for more info.

dependency injection

A software design pattern where each class receives all objects it depends upon. Sometimes these are passed through the constructor, but not always. See [dependency injection](#) on Wikipedia for more info.

deprecated

Software that has been replaced and will no longer receive new features. Deprecated software will be maintained for at least 1 year, but may be removed after that. For example,

if a method is deprecated prior to the 2022 season, it will be usable in the 2022 season, but may be removed prior to the 2023 season. Teams are encouraged to not use deprecated methods in new code. WPILib always deprecates features at least one year prior to removing them from the codebase.

design pattern

A particular, intentionally-chosen style of organizing code. A design pattern intentionally excludes using certain features of a programming language to constrain developers into solutions that are well-suited to a particular problem-space. See [design pattern](#) on Wikipedia for more info.

DHCP

Dynamic Host Configuration Protocol - the protocol that allows a central device to assign unique IP addresses to all other devices.

encapsulation

A software design pattern which uses a class to hide the implementation details of other classes. See [encapsulation](#) on Wikipedia for more info.

entry

In [NetworkTables](#), a combined [publisher](#) and [subscriber](#). The subscriber is always active, but the publisher is not created until a publish operation is performed (e.g. a value is “set”, aka published, on the entry). This may be more convenient than maintaining a separate publisher and subscriber.

enumeration

A list of all elements of a set, typically used to refer to a set of pre-defined values.

EPA

[Expected Points Added](#) - builds upon the Elo rating system, but transforms ratings to point units and makes several modifications.

event-driven programming

A style of programming where certain parts of code generate “events” as a result of some input (sensors, user interaction, etc). Then, other parts of code listen for and respond to “handle” these events. See [event-based](#) on Wikipedia for more info.

FIRST

For Inspiration and Recognition of Science and Technology - a global nonprofit organization that prepares young people for the future through a suite of life-changing youth robotics programs that build skills, confidence, and resilience.

FLL

FIRST Lego League - Introduces science, technology, engineering, and math (STEM) to children ages 4-16 through fun, exciting hands-on learning.

floating point

A method for approximating real numbers in computer-based arithmetic, using a fixed precision integer scaled by an integer exponent. Typically computer systems support both “single” precision (32-bit storage) and “double” precision (64-bit storage) floating point values, as defined by IEEE 754.

FMS

Field Management System - the electronics core responsible for sensing and controlling the FIRST Robotics Competition field.

FPGA

Field-programmable gate array - a specialized integrated circuit consisting of many digital logic elements, which can be configured to act in different patterns. This allows

its behavior to be changed after manufacturing. In the context of FRC, National Instruments provides a specific configuration for the RIO's FPGA which allows it to process the electrical inputs and outputs at a very high rate. See [FPGA](#) on Wikipedia for more info.

FRC

FIRST Robotics Competition - Combining the excitement of sport with the rigors of science and technology. The ultimate Sport for the Mind inspiring High-school students.

FTA

FIRST Technical Advisor - FRC volunteer position that is responsible for ensuring FIRST Robotics Competition events run smoothly, safely, and in accordance with FIRST requirements, and ensuring a high-quality experience for all event participants and teams.

FTC

FIRST Tech Challenge - Grades 7-12 are challenged to design, build, program, and operate robots to compete in a head-to-head challenge in an alliance format.

GDC

Game Design Committee - designs the game for each year and develops the rules and field setup for each competition.

GP

Gracious Professionalism - part of the ethos of FIRST. It's a way of doing things that encourages high-quality work, emphasizes the value of others, and respects individuals and the community.

GradleRIO

The mechanism that powers the deployment of robot code to the roboRIO.

gyroscope

A device that measures rate of rotation. It can add up the rotation measurements to determine [heading](#) of the robot. ("gyro", for short)

rota

Robotun işaret ettiği yön, genellikle bir açı olarak derece cinsinden ifade edilir.

imperative programming

A style of programming that focuses on *what* the code should be doing, step by step, every loop. See [imperative programming](#) on Wikipedia for more info.

IMU

Inertial Measurement Unit - a sensor that combines both an [accelerometer](#) and a [gyroscope](#) into a single sensor.

I2C

Inter-Integrated Circuit - a synchronous, multi-master/multi-slave (controller/target), single-ended, serial communication bus.

Java

One of the four officially supported programming languages.

JSON

JavaScript Object Notation - A standardized way of organizing data into named values. The organized data can be easily [serialized](#). While the original usage was in Javascript, it can be used and interested by most modern programming languages. See [JSON](#) on Wikipedia for more info.

KOP

Kit of Parts - the collection of items listed on the Kickoff Kit checklists, distributed to the

team via FIRST Choice, or paid for completely (except shipping) with a Product Donation Voucher (PDV).

KOP chassis

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.

simülasyon

Takımların yanlarında gerçek bir robot bulunmadan kodlarını test edebilmeleri için bir yol.

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 *topics*.

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.

uzaktan kontrol

Maçın ikinci aşamasına Uzaktan Kontrol Periyodu (Teleoperated Period / Teleop) adı verilir ve bu bölümde sürücüler robotlarını uzaktan kontrol ederler.

topic

In [NetworkTables](#), a named data channel.

torque

A force applied at a distance from some axis of rotation

trajectory

A trajectory is a smooth curve, with velocities and accelerations at each point along the curve, connecting two endpoints on the field.

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.

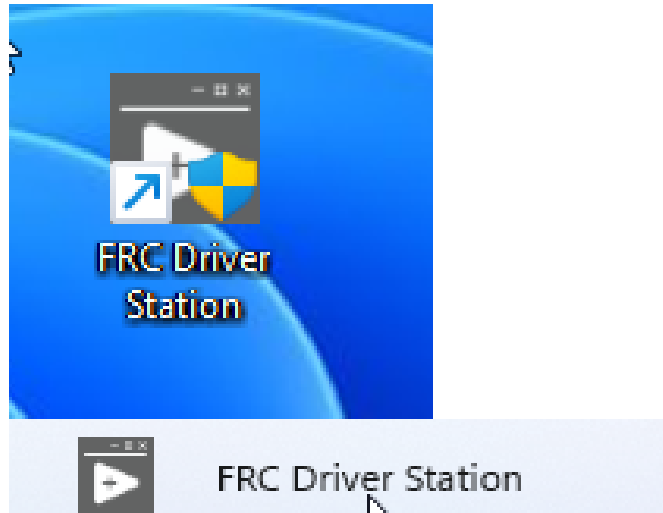
WFA

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.

20.1 FRC Driver Station Powered by NI LabVIEW

Bu makale, FRC|reg| Sürücü İstasyonu NI LabVIEW tarafından desteklenmektedir.
Driver Station yazılımının yüklenmesi hakkında bilgi için bkz [this document](#)

20.1.1 FRC Driver Station - Sürücü İstasyonunu Başlatma



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

Not: By default the FRC Driver Station launches the *LabVIEW Dashboard*. It can also be configured on *Setup Tab* to launch the other Dashboards: *SmartDashboard* and *Shuffleboard*. WPILib must be *installed* to use SmartDashboard and Shuffleboard.

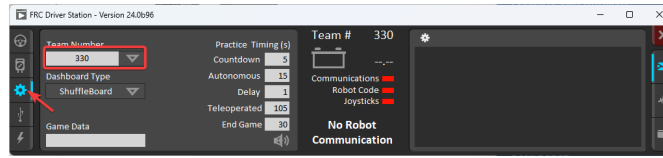
20.1.2 Driver Station Tuş Kısayolları

- 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.

Not: Boşluk çubuğu, Driver Station penceresinin seçili olup olmadığına bakılmaksızın robotu E-Stop yapar

Uyarı: 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.

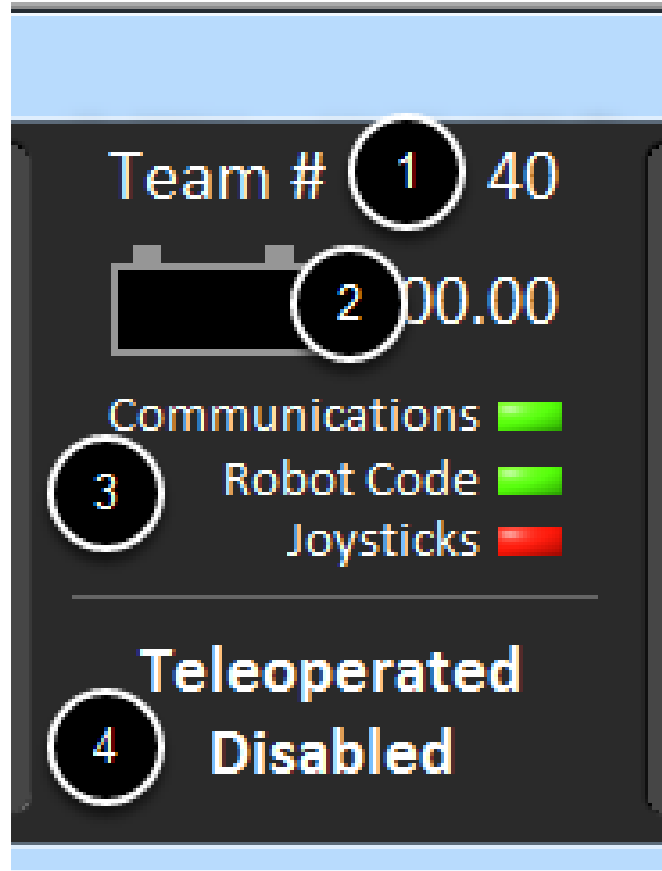
20.1.3 Sürücü İstasyonunu Kurmak



Robotunuza bağlanmak için DS, takım numaranıza ayarlanmalıdır. Bunu yapmak için Kurulum sekmesine tıklayın ve ardından takım numarası kutusuna takım numaranızı girin. Ayarın etkili olması için return tuşuna basın veya kutunun dışını tıklayın.

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

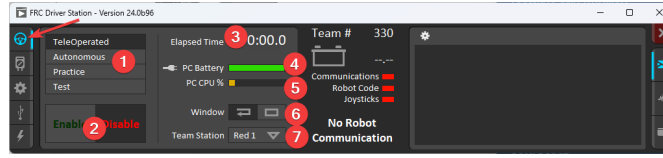
20.1.4 Status Paneli



Driver Station Status Bölmesi, ekranın ortasında yer alır ve seçilen sekmeden bağımsız olarak her zaman görülebilir. DS ve robotun durumu hakkında bir dizi kritik bilgi görüntüler:

1. Team # - DS'nin şu anda yapılandırıldığı Takım numarası. Bu, FRC takım numaranızla eşleşmelidir. Takım numarasını değiştirmek için Setup Sekmesine bakın.
2. Battery Voltage - DS bağlıysa ve roboRIO ile iletişim halindeyse, bu mevcut akü voltajını bir sayı olarak ve pil simgesinde zaman içinde küçük bir voltaj çizelgesiyle görüntüler. RoboRIO karartması tetiklendiğinde sayısal göstergenin arka planı kırmızıya dönecektir. Daha fazla bilgi için [RoboRIO Kesintisi ve Mevcut Çekişi Anlama](#) a bakın.
3. Major Status Indicators - Bu üç gösterge, DS için ana durum öğelerini görüntüler. "Communications", DS'nin şu anda roboRIO üzerindeki FRC Network Communications Task ile iletişim kurup kurmadığını gösterir (TCP ve UDP iletişimi için ikiye bölünmüştür). "Robot Code" göstergesi, takım Robot Kodunun şu anda çalışıp çalışmadığını gösterir (robot kodundaki Drive Station görevinin pil voltajını güncelleyip güncellemediğine göre belirlenir), "Joysticks" göstergesi en az bir kumanda çubuğunun takılı olup olmadığını gösterir ve DS tarafından tanındı.
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 Tabı

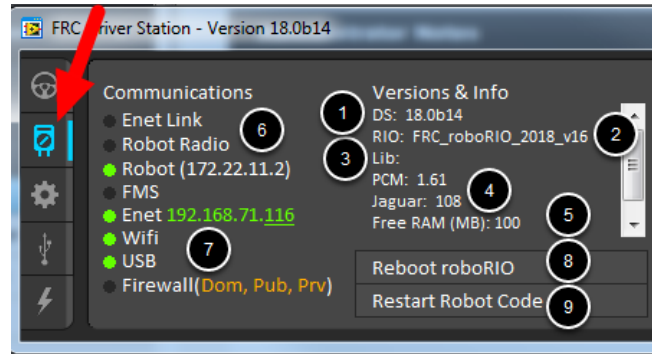


Operations Sekmesi, robotun modunu kontrol etmek ve robot çalışırken ek anahtar durum göstergeleri sağlamak için kullanılır.

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 - FMS'ye bağlı olmadığında, takım istasyonunu robota aktarım yapacak şekilde ayarlar.

Not: 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.

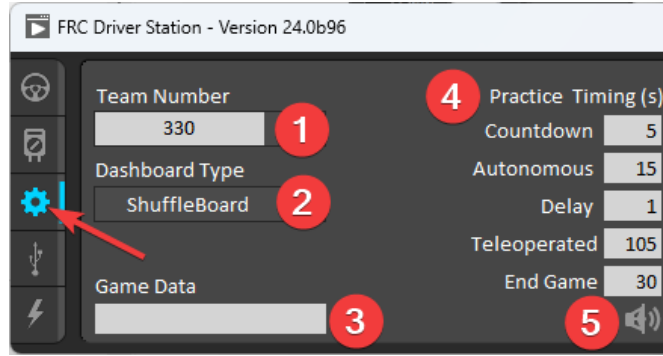
20.1.6 Diagnostics-Arıza tespit Tabı



Diagnostics Sekmesi, takımların robotlarıyla ilgili sorunları teşhis etmek için kullanabilecekleri ek durum göstergeleri içerir:

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 - Bu göstergelerin üst yarısı, çeşitli bileşenlere bağlantı durumunu gösterir.
 - “Enet Link”, bilgisayarın ethernet bağlantı noktasına bağlı bir şey olduğunu gösterir.
 - “Robot Radio”, 10.XX.YY.1’de robot kablosuz ağ köprüsüne ping durumunu gösterir.
 - “Robot”, mDNS kullanarak roboRIO’ya ping durumunu gösterir (statik 10.TE.AM.2 adresinin geri dönüşü ile).
 - “FMS”, DS’nin FMS’den paket alıp almadığını gösterir (bu bir ping göstergesi DEĞİLDİR).
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”, tespit edilen Ethernet adaptörünün IP adresini gösterir
 - “WiFi”, bir kablosuz adaptörün etkin olarak algılanıp algılanmadığını gösterir
 - “USB”, bir roboRIO USB bağlantısının tespit edilip edilmediğini gösterir
 - “Firewall” indicates if any firewalls are detected as enabled. Enabled firewalls will show in orange (Dom = Domain, Pub = Public, Prv = Private)
8. *Reboot roboRIO* - This button attempts to perform a remote reboot of the roboRIO (after clicking through a confirmation dialog).
9. *Restart Robot Code* - This button attempts to restart the code running on the robot (but not restart the OS).

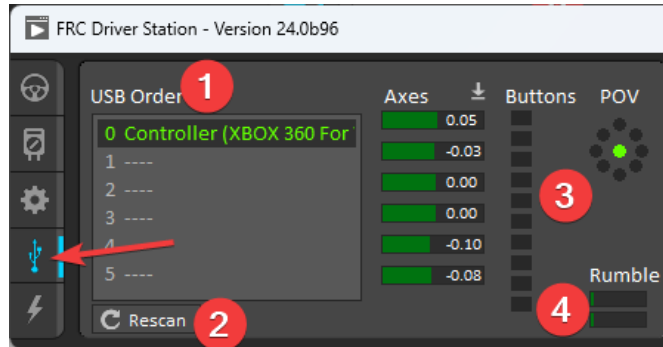
20.1.7 Setup Tabı



Setup Sekmesi, ekiplerin Driver Station'ın çalışmasını kontrol etmek için kullanabileceği bir dizi düğme içerir:

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* - Bu kutular, uygulama modu sırasının her bir kısmının zamanlamasını kontrol eder. Robot uygulama modunda etkinleştirildiğinde, DS otomatik olarak yukarıdan aşağıya gösterilen modlarda ilerler.
5. *Audio Kontrolü* - Bu düğme, practice Modu kullanıldığında ses tonlarının çıkıp çıkmayacağını kontrol eder.

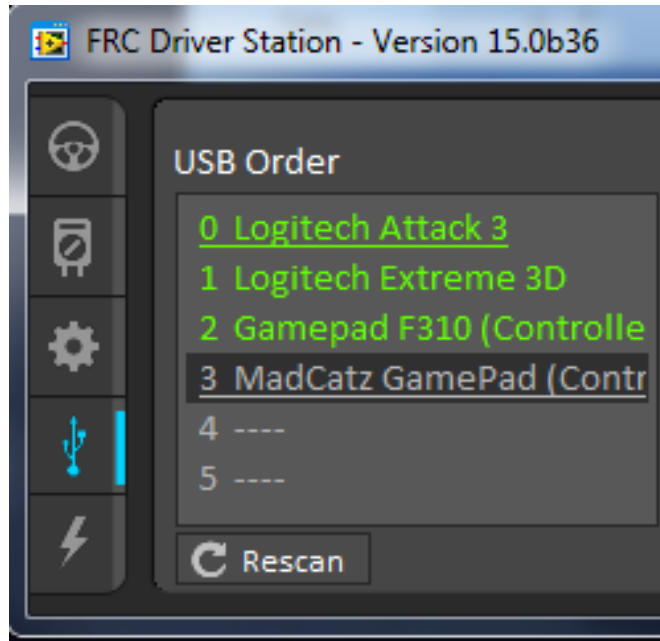
20.1.8 USB Devices Sekmesi



USB Devices sekmesi, DS'ye bağlı USB Aygıtları hakkındaki bilgileri içerir.

1. USB Setup List - Bu, DS'ye bağlı tüm uyumlu USB cihazlarının bir listesini içerir. Bir cihazda bir düğmeye basmak, adı yeşil renkte vurgular ve cihaz adının önüne 2 * lar koyar
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 - Bu göstergeler, eksenlerin, düğmelerin ve joystick'in POV'unun mevcut durumunu gösterir.
4. Rumble - XInput cihazları (X-Box denetleyicileri gibi) için Rumble denetimi görünecektir. Bu, cihazın gürültü işlevselliğini test etmek için kullanılabilir. Üst çubuk "Sağ Rumble" ve alttaki çubuk "Sol Rumble" dır. Bar boyunca herhangi bir yere tıklamak ve basılı tutmak sarsıntıyı orantılı olarak etkinleştirecektir (sol sarsıntı yok = 0, sağ tam sarsıntı = 1). Bu yalnızca bir kontroldür ve robot kodunda ayarlanan Rumble değerini göstermez.

Cihazları Yeniden Düzenleme ve Kilitleme



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.

Not: 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

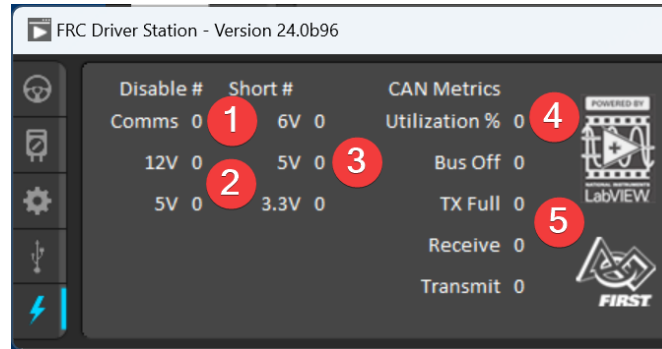
behavior is not determinate (the devices may swap slots). If you unplug one or more of the set of devices, the positions of the others may move; they should return to the proper locked slots when all devices are reconnected.

Örnek: Yukarıdaki görüntü 4 cihazı göstermektedir:

- Kilitli “Logitech Attack 3” joystick. Bu cihaz, başka bir yere sürüklenmedikçe veya kilidi açılmadıkça bu konumda kalacak
- Kilitlenmemiş bir “Logitech Extreme 3D” joystick
- Logitech F310 oyun kumandası olan kilidi açılmış bir “Gamepad F310 (Denetleyici)”
- MadCatz Xbox 360 Kumandası olan kilitli, ancak bağlantısı kesilmiş bir “MadCatz Game-Pad (Denetleyici)”

Bu örnekte, Logitech Extreme 3D oyun çubuğunun çıkarılması, F310 Gamepad’ın 1. yuvaya çıkmasına neden olacaktır. MadCatz Gamepad takmak (Yuva 1 ve 2’deki cihazlar çıkarılsa ve bu yuvalar boş olsa bile) Yuva 3’ü işgal edecektir.

20.1.9 CAN/Power Tabı

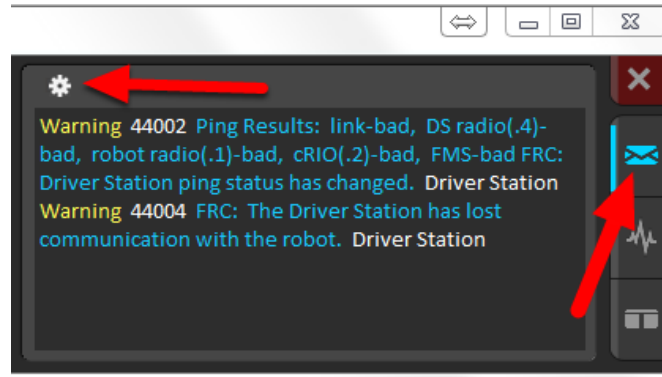


DS’nin sol tarafındaki son sekme CAN/Robot Power sekmesidir. Bu sekme roboRIO’nun güç durumu ve CAN veriyolunun durumu hakkında bilgiler içerir:

1. Comms Faults - DS’nin bağlandığından beri meydana gelen Haberleşme arızalarının sayısını gösterir
2. 12V Faults - DS’nin bağlanmasından bu yana meydana gelen giriş gücü arızalarının (Brownouts) sayısını gösterir.
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 veriyolunun kullanım yüzdesini gösterir
5. CAN faults - DS bağlandığından beri 4 tip CAN arızasının her birinin sayısını gösterir.

Bir arıza tespit edilirse, bu sekmenin göstergesi (yukarıdaki resimde mavi ile gösterilmiştir) kırmızıya dönecektir.

20.1.10 Messages Tabı



The Messages tab displays diagnostic messages from the DS, WPILib, User Code, and/or the roboRIO.

To access settings for the Messages tab, click the Gear icon. This will display a menu that will allow you to clear the box, launch a larger Console window for viewing messages, launch the DS Log Viewer, launch a viewer for program timing, or download log files from the robot.

20.1.11 Charts Sekmesi



Charts sekmesi, takımların robot sorunlarını teşhis etmesine yardımcı olmak için robot durumunun gelişmiş göstergelerini çizer ve görüntüler:

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. Alt grafikte pil voltajı sarı renkte (soldaki eksene karşı), roboRIO CPU kırmızı renkte (sağdaki eksene karşı), DS Requested modu grafiğin altında sürekli bir çizgi olarak ve onun üzerinde robot modunu kesintili bir çizgi olarak gösterir.
3. Bu anahtar, alttaki grafikte DS Requested ve Robot Reported modları için kullanılan renkleri gösterir.
4. Chart scale - These controls change the time scale of the DS Charts.
5. This button launches the [DS Log File Viewer](#).

DS Requested , Drive Station'ın robota giriş için komut verdiği moddur. Robot Reported modu, her dil için kodlama çerçevelerinde bulunan raporlama yöntemlerine göre gerçekte kodun

çalıştırıldığı moddur.

20.1.12 Her İki Sekme

The last tab on the right side is the Both tab which displays Messages and Charts side by side.

20.2 En iyi Driver Station pratikleri

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](#).

Driver Station'ü FIRST Robotik Yarışması (FRC) sahasındaki takımınız için bir engel olmadığından emin olmak ister misiniz? Sağlam bir Driver Station dizüstü bilgisayarı oluşturmak ve yapılandırmak, son yapım günü ile yarışmanız arasındaki süre için kolay bir projedir. Binlerce maçta birçok takım tarafından öğrenilen dersleri bulmak için okumaya devam edin.

20.2.1 Yarışmaya Gitmeden Önce

1. Yalnızca Driver Station olarak kullanılacak bir dizüstü bilgisayar ayırın. Birçok takım bunu yapar. Özel bir makine, konfigürasyonu tek bir amaç için yönetmenize olanak tanır - sahada rekabet etmeye hazır olmak. Adanmış, FRC tarafından sağlanan Driver Station yazılımı ve kurulu veya çalışan ilişkili Dashboard dışında başka hiçbir yazılım olmadığı anlamına gelir.
2. Driver Station için iş sınıfı bir dizüstü bilgisayar kullanın. Neden? Çünkü Best Buy'daki 300 \$ 'lık Black Friday özel etkinliğinden çok daha dayanıklı olası gerekir. Bu bilgisayar yarışmada zor şartlarda çalışacaktır. İşletme sınıfı dizüstü bilgisayarlar daha kaliteli aygıt sürücülerine sahiptir ve sürücüler tüketici dizüstü bilgisayarlarından daha uzun süre korunur. Bu, yatırımınızın daha uzun ömürlü olmasını sağlar. Lenovo ThinkPad T serisi ve Dell Latitude, yarışmalarda sıkça göreceğiniz iki popüler işletme sınıfı markadır. EBay'de her gün binlerce satış var. Yeni çaylak kitlerinde sağlanan dizüstü bilgisayar, giriş seviyesi için iyi bir makinedir. Takımlar, görüş ve gösterge tablolarıyla daha fazlasını yaptıkça genellikle ondan daha büyük ekranlara geçiş yapar.
3. Yeni yerine kullanılmış dizüstü bilgisayarları düşünün. FRC| reg| Driver Station ve kontrol paneli yazılımı çok az sistem kaynağı kullanır, bu nedenle yeni bir dizüstü bilgisayar satın almanıza gerek kalmaz - bunun yerine 4-5 yıllık ucuz bir kullanılmış olanı satın alın. Hatta bölgenizdeki kullanılmış bir bilgisayar mağazasından bağışlanmış bir tane bile alabilirsiniz.

Not: 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. Dizüstü bilgisayarda önerilen özellikler
 - a. RAM - 8GB of RAM or greater
 - b. Minimum 1440x1050 çözünürlükle 13 "veya daha büyük bir ekran boyutu.
 - c. Portlar

- i. Yerleşik bir Ethernet bağlantı noktası çok tercih edilir. Tam boyutlu bir bağlantı noktası olduğundan emin olun. Menteşeli Ethernet bağlantı noktaları, tekrar kullanıma dayanmaz.
- ii. Ethernet bağlantınızı yapmak için bir Ethernet bağlantı noktası koruyucusu kullanın. Bu, dizüstü bilgisayardaki bağlantı noktasının ömrünü uzatır. Menteşeli Ethernet bağlantı noktasına sahip tüketici sınıfı bir dizüstü bilgisayarınız varsa, bu özellikle önemlidir.
- iii. Dizüstü bilgisayarınızdaki Ethernet bağlantı noktası tehlikeli ise ya dizüstü bilgisayarınızı değiştirin (önerilir) ya da tanınmış bir markadan bir USB Ethernet dongle satın alın. Çoğu ekip, öncelikle ucuz donanım ve kötü sürücüler nedeniyle USB Ethernet'in yerleşik Ethernet'ten daha az güvenilir olduğunu fark eder. KOP'ta çaylaklara verilen dongle'lar iyi çalışma konusunda bir üne sahiptir.
- iv. Minimum 2 USB bağlantı noktası
- d. Klavye. Sahadaki yalnızca dokunmatik bilgisayarlarda hızlı bir şekilde sorun gidermek zordur.
- 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. Yarışmadan bir hafta önce tüm Windows güncellemelerini yükleyin. Bu, güncellemelerin Driver Station işlevlerini etkilememesini sağlamak için size zaman tanır. Bunu yapmak için Windows Update ayarları sayfasını açın ve güncel olduğunuzu görün. Değilse bekleyen güncellemeleri yükleyin. Güncel olduğunuzdan emin olmak için yeniden başlatın ve tekrar kontrol edin.
- 6. Güncellemelerin yarışma saatleri sırasında yüklenmesini önlemek için Windows Güncellemeleri için "Etkin Saatleri" değiştirin. Başlat -> Ayarlar -> Güncelleme ve Güvenlik -> Windows Güncelleme'ye gidin, ardından Etkin saatleri değiştir'i seçin. Bir yarışmaya seyahat ediyorsanız, saat dilimi farklılıklarını dikkate alın. Bu, sahada güncelleme yüklenmesi nedeniyle sürücü istasyonunuzun yeniden başlatılmamasını veya başarısız olmasını sağlamaya yardımcı olacaktır.
- 7. Remove any 3rd party antivirus or antimalware software. Instead, use Windows Defender on Windows 10 or 11. Since you're only connecting to the internet for Windows and FRC software updating, the risk is low. Only install software on your driver station that's needed for driving. Your goal here is to eliminate variables that might interfere with proper operation. Remove any unneeded preinstalled software ("bloatware") that came with the machine. Don't use the laptop as your Steam machine for gaming back at the hotel the night before the event. Many teams go as far as having a separate programming laptop.
- 8. Avoid managed Windows 10 or 11 installations from the school's IT department. These deployments are built for the school environment and often come with unwanted software that interferes with your robot's operation.
- 9. Dizüstü bilgisayar pili / gücü
 - a. Turn off Put the computer to sleep in your power plan for both battery and powered operation.
 - b. USB Seçmeli Askıya Almayı kapatın:

- i. Tepsideki pil / şarj simgesine sağ tıklayın, ardından Güç Seçenekleri'ni seçin.
 - ii. Güç planınızın plan ayarlarını düzenleyin.
 - iii. Gelişmiş güç ayarlarını değiştir bağlantısını tıklayın.
 - iv. Gelişmiş ayarlarda aşağı kaydırın ve hem Pil hem de Takılı için USB seçmeli askıya alma ayarını devre dışı bırakın.
 - c. Yukarıdaki değişiklikleri yaptıktan sonra dizüstü bilgisayar pilinin en az bir saat şarj edilebildiğinden emin olun. Bu, robotun ve sürücü ekibinin kuyruktan geçmesi ve şebeke gücü olmadan ittifak istasyonuna ulaşması için bolca zaman sağlar.
10. RoboRIO'ya bağlanmak için güvenilir bir USB ve Ethernet kablosu getirin.
 11. Joystick / oyun kumandası denetleyicilerinizin yere düşmesini ve / veya USB bağlantı noktalarına kaymasını önlemek için tutma / gerilim azaltma ekleyin. Bu, kesintili denetleyici bağlantılarıyla ilgili sorunları önlemeye yardımcı olur.
 12. Sürmek için kullandığınız Windows kullanıcı hesabı, Yönetici grubunun bir üyesi olmalıdır.

20.2.2 Yarışmada

1. Turn off Windows firewall using [these instructions](#).
2. Özel donanım Wi-Fi anahtarını kullanarak veya Adaptör Ayarları kontrol panelinden devre dışı bırakarak Wi-Fi adaptörünü kapatın.
3. Pitteyken sürücü istasyonunu şarj edin.
4. Oturum açma parolalarını kaldırın veya sürücü ekibindeki herkesin parolayı bildiğinden emin olun. Sürücülerin dizüstü bilgisayarın şifresini bilmeden sahaya ne sıklıkla geldiklerine şaşıracaksınız.
5. LabView kodunuzun kalıcı olarak dağıtıldığından ve LabView Eğitim Kılavuzundaki talimatları kullanarak "run as startup-başlangıç olarak çalıştır" olarak ayarlandığından emin olun. Robotu her açtığınızda kodu dağıtmanız gerekiyorsa, yanlış yapıyorsunuz demektir.
6. Web taramasını FRC ile ilgili web siteleriyle sınırlayın. Bu, yarışma sırasında kötü amaçlı yazılım alma olasılığını en aza indirir.
7. Yazılım güncellemeleri yapmak için internet erişimini kullanmayı planlamayın. Mekanda muhtemelen hiç olmayacak ve otel Wi-Fi kalitesi büyük ölçüde farklılık gösteriyor. Güncellemelere ihtiyacınız varsa, pitteki bir Kontrol Sistemi Danışmanı ile iletişime geçin.

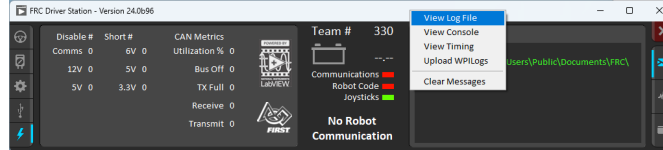
20.2.3 Her Maçtan Önce

1. Sizin maçınızdan önceki maçın bitiminden önce dizüstü bilgisayarınızın açık olduğundan ve oturum açtığınızdan emin olun.
2. Maç sırasında ihtiyaç duyulmayan programları - örneğin, Visual Studio Code veya LabView - yarışırken ederken kapatın.
3. Dizüstü bilgisayar şarj cihazınızı sahaya getirin. Her oyuncu istasyonunda size güç sağlanır.

4. Dizüstü bilgisayarınızı cırt cırtlı bantla oynatıcı istasyonu rafına sabitleyin. İttifak ortağınızın ne zaman otonom bir programlama sorunu yaşayacağını ve duvarı patlatacağını asla bilemezsiniz.
5. Kumanda çubuklarının ve denetleyicilerin doğru USB bağlantı noktalarına atandığından emin olun
 - a. FRC Driver Station yazılımındaki USB sekmesinde, gerektiğinde oyun çubukları atamak için sürükleyip bırakın.
 - b. Joystickler / denetleyiciler yeşil görünmüyorsa yeniden tarama düğmesini (F1) kullanın
 - c. Yarışma sırasında, oyun çubuğu veya kontrol cihazlarının fişi çekilir ve ardından tekrar takılırsa veya yarışma sırasında gri renge dönerse, yeniden tarama düğmesini (F1) kullanın.
6. Ensure your *Dashboard is connected to the robot* after your driver station connects to the robot.

20.3 Driver Station Log File Viewer

In an effort to provide information to aid in debugging, the FRC® Driver Station creates log files of important diagnostic data while running. These logs can be reviewed later using the FRC Driver Station Log Viewer. The Log Viewer can be found via the shortcut installed in the Start menu, in the FRC Driver Station folder in Program Files, or via the Gear icon in the Driver Station.



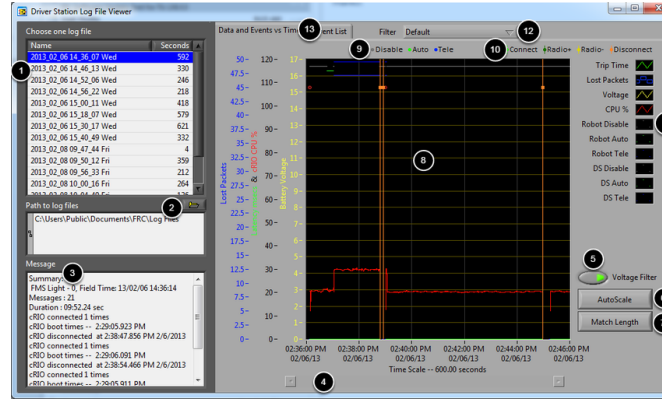
Not: Several alternative tools exist that provide similar functionality to the FRC Driver Station Log Viewer. *AdvantageScope* is an option included in WPILib, and *DSLOG Reader* is a third-party option. Note that WPILib offers no support for third-party projects.

20.3.1 Etkinlik Kayıtları- Logs

The Driver Station logs all messages sent to the Messages box on the Diagnostics tab (not the User Messages box on the Operation tab) into a new Event Log file. When viewing Log Files with the Driver Station Log File Viewer, the Event Log and Driver Station Log files are overlaid in a single display.

Log files are stored in C:\Users\Public\Documents\FRC\Log Files. Each log has date and timestamp in the file name and has two files with extension .dslog and .dsevents.

20.3.2 Log Viewer UI

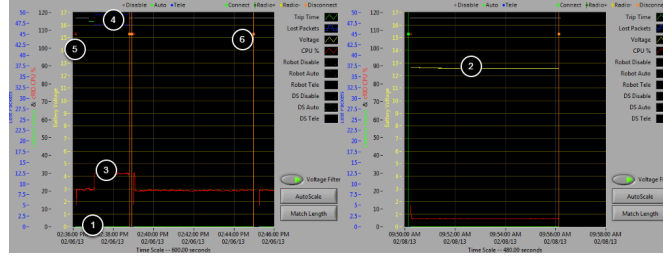


Log Viewer, Driver Station günlük dosyalarının analizine yardımcı olacak bir dizi kontrol ve ekran içerir:

1. File Selection Box - Bu pencere, seçili klasördeki tüm kullanılabilir günlük dosyalarını görüntüler. Listedeki bir günlük dosyasını seçmek için tıklayın.
2. Path to Log Files - Bu kutu, görüntüleyicinin günlük dosyaları için aradığı geçerli klasörü görüntüler. Bu, varsayılan olarak Driver Station'ın günlük dosyalarını içinde sakladığı klasördür. Farklı bir konuma göz atmak için klasör simgesini tıklayın.
3. Mesaj Kutusu-Message Box - Bu kutu, Olay Günlüğündeki tüm mesajların bir özetini görüntüler. Grafikteki bir olayın üzerine gelindiğinde, bu kutu o olayın bilgilerini görüntülemek için değişir.
4. Scroll Bar - Grafik yakınlaştırıldığında, bu kaydırma çubuğu grafiğin yatay kaydırılmasına izin verir.
5. Voltage Filter - Bu kontrol Voltaj Filtresini açar ve kapatır (varsayılan olarak açıktır). Voltaj Filtresi, Pil Voltajı alınmadığında CPU%, robot modu ve trip time gibi verileri filtreler (DS'nin roboRIO ile iletişim halinde olmadığını gösterir).
6. AutoScale - Bu düğme, günlükteki tüm verileri göstermek için grafiği uzaklaştırır.
7. Match Length-Maç Uzunluğu - Bu düğme, grafiği yaklaşık olarak bir FRC eşleşmesinin uzunluğuna ölçeklendirir (2 dakika ve 30 saniye gösterilir). Maçın başlangıcını otomatik olarak bulmaz, Otonom modun başlangıcını bulmak için kaydırma çubuğunu kullanarak kaydırmanız gerekecektir.
8. Grafik-Graph - Bu ekran, DS Günlük dosyasından (voltaj, yolculuk süresi, roboRIO CPU%, Kayıp Paketler ve robot modu) alınan grafik verilerini ve ayrıca üst üste binen olay verilerini (grafikte noktalar olarak gösterilen, seçili olayların dikey çizgiler olarak gösterildiği tüm grafik). Grafikteki olay işaretçilerinin üzerine gelmek, ekranın sol alt kısmındaki Messages penceresinde olayla ilgili bilgileri görüntüler.
9. Robot Mode Key - Ekranın üst kısmında görüntülenen Robot Modu için anahtar
10. Önemli olay belirteçleri - Grafikte dikey çizgiler olarak gösterilen, önemli olayların belirteçleridir
11. Graph tuşu - Grafik verileri için önemlidir
12. Filtre Kontrolü-Filter Control - Filtre modunu seçmek için açılır menü (filtre modları aşağıda açıklanmıştır)

13. Sekme Kontrolü-Tab Control - Grafik (Veriler ve Olaylara karşı Zaman) ve Olay Listesi ekranları arasında geçiş yapma kontrolü.

20.3.3 Using the Graph Display



Grafik Ekranı aşağıdaki bilgileri içerir:

1. Gezi Süresi - Trip Time ms (yeşil çizgi) ve saniye başına Kayıp Paket - Lost Packets (mavi dikey çubuklar olarak görüntülenir) cinsinden gösterilir. Bu örnek resimlerde Trip Time, grafiğin altında düz yeşil bir çizgidir ve kayıp paket yoktur
2. Sarı çizgi olarak gösterilen Pil voltajı grafiği.
3. Kırmızı çizgi olarak roboRIO CPU% grafiği
4. Robot modu ve DS modu grafiği. Ekranın üst grubu, Driver Station tarafından komut verilen modu gösterir. Altta set, robot kodu tarafından bildirilen modu gösterir. Bu örnekte robot, devre dışı ve otonom modlar sırasında modunu rapor etmiyor, ancak Teleop sırasında rapor edilmiş.
5. Olayın meydana geldiği zamanı gösteren olay işaretçileri grafikte görüntülenecektir. Hatalar kırmızı renkte görüntülenecektir; uyarılar sarı renkte görüntülenecektir. Bir olay işaretleyicisinin üzerine gelmek, ekranın sol alt kısmındaki Messages kutusunda olayla ilgili bilgileri görüntüler.
6. Önemli olaylar, grafik ekranı boyunca dikey çizgiler olarak gösterilir.

Grafiğin bir bölümünü yakınlaştırmak için, istenen görüntüleme alanını tıklayıp sürükleyin. Yalnızca zaman eksenini yakınlaştıracabilirsiniz, dikey olarak yakınlaştıramazsınız.

20.3.4 Event List

DS Time	Event Message Text
2:36:07.288 PM	WARNING <Code> 44007 occurred at FRC_NetworkCommunications <secondsSinceReboot> 421.365 Warning <Code> 44001 occurred at No Change to Network Configuration: "Local Area Connection"<noNIC FRC: Time since robot boot. Driver Station <time>2/6/2013 2:36:07 PM<unique#>3 ERROR <Code> 44009 occurred at Driver Station <time>2/6/2013 2:36:06 PM<unique#>2 FRC: A joystick was disconnected while the robot was enabled. Warning <Code> 44006 occurred at Driver Station <time>2/6/2013 2:36:06 PM<unique#>1 FRC: Custom I/O is not enabled or is not connected to the driver station.
2:36:07.328 PM	FMS Connected: FMS Light - 0, Field Time: 13/02/06 14:36:14
2:36:10.441 PM	WARNING <Code> 44008 occurred at FRC_NetworkCommunications <radioLostEvents> 173.563 <radioSee FRC: Robot radio detection times.
2:37:01.461 PM	Watchdog Expiration: System 1, User 0
2:38:47.856 PM	Warning <Code> 44004 occurred at Driver Station <time>2/6/2013 2:38:47 PM<unique#>4 FRC: The Driver Station has lost communication with the robot.
2:38:49.356 PM	Warning <Code> 44002 occurred at Ping Results: link-GOOD, DS radio(4)-GOOD, robot radio(1)-GOOD, <time>2/6/2013 2:38:49 PM<unique#>5 FRC: Driver Station ping status has changed.
2:38:53.460 PM	WARNING <Code> 44007 occurred at FRC_NetworkCommunications <secondsSinceReboot> 587.369 FRC: Time since robot boot.
2:38:54.466 PM	Warning <Code> 44004 occurred at Driver Station <time>2/6/2013 2:38:53 PM<unique#>6 FRC: The Driver Station has lost communication with the robot.
2:38:55.468 PM	Warning <Code> 44002 occurred at Ping Results: link-GOOD, DS radio(4)-GOOD, robot radio(1)-GOOD, <time>2/6/2013 2:38:55 PM<unique#>7 FRC: Driver Station ping status has changed.
2:38:59.278 PM	WARNING <Code> 44008 occurred at FRC_NetworkCommunications <radioLostEvents> 339.065 <radioSee FRC: Robot radio detection times. WARNING <Code> 44007 occurred at FRC_NetworkCommunications <secondsSinceReboot> 593.367

Event List sekmesi, Driver Station tarafından kaydedilen olayların (uyarılar ve hatalar) bir listesini görüntüler. Görüntülenen olaylar ve ayrıntılar, o anda etkin olan filtre tarafından belirlenir (resimlerde "All Events, All Info" filtresi etkin gösterilir).

20.3.5 Filtreler

Şu anda Günlük Görüntüleyicide-Log Viewer üç filtre bulunmaktadır:

1. Default: Bu filtre, Driver Station tarafından üretilen birçok hata ve uyarıyı filtreler. Bu filtre, Robot'ta kod tarafından atılan hataları tanımlamak için kullanışlıdır.
2. All Events and Time: Bu filtre tüm olayları ve meydana geldikleri zamanı gösterir
3. All Events, All Info: Bu filtre tüm olayları ve tüm kayıtlı bilgileri gösterir. Şu anda, bu filtre ile "All Events and Time" arasındaki temel fark, bu seçeneğin belirli bir mesajın ilk oluşumu için "benzersiz" göstericiyi göstermesidir.

20.3.6 Maçlardan Günlükleri Tanımlama

3:19:30.893 PM FMS Connected: Practice - 1, Field Time: 13/02/06 15:19:37

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 Günlük Görüntüleyici-Log Viewer ile Yaygın Bağlantı Hatalarını Tanımlama

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.

Not: Bu bölümde gösterilen tüm günlük dosyalarının Match Length - Uzunluğu Eşleştir düğmesi kullanılarak uzunluk eşleşecek şekilde ölçeklendiğini ve ardından otonom modun başlangıcına kaydırıldığını unutmayın. Ayrıca, günlüklerin çoğu pil voltajı bilgilerini içermez, günlük kaydı için kullanılan platform pil voltajını bildirmek için uygun şekilde kablolanmamıştır.

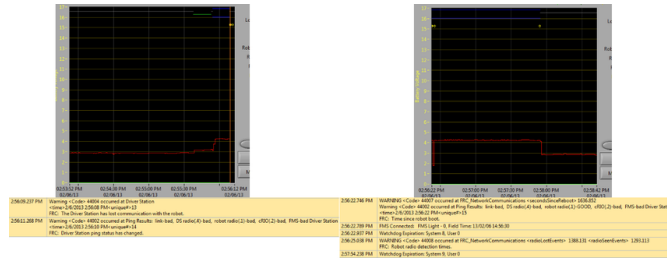
Tüyo: Günlük Görüntüleyicide bulunan bazı hata mesajları aşağıda gösterilmektedir ve daha fazlası: doc: *Driver Station Hataları / Uyarıları* makalesinde detaylandırılmıştır.

“Normal” Log-Kayıtlar



Bu, normal bir maç günlüğü örneğidir. İlk kutuda yer alan hatalar ve uyarılar, DS'nin ilk başlatıldığı zamandır ve göz ardı edilebilir. Bu, bu olayların “FMS Connected:” olayından önce meydana geldiği gözlemlenerek doğrulanır. Gösterilen son olay da göz ardı edilebilir, aynı zamanda robotun DS'ye ilk bağlanmasından (FMS'ye bağlandıktan 3 saniye sonra gerçekleşir) ve maç başlamadan yaklaşık 30 saniye önce gerçekleşir.

FMS'den bağlantı kesik



DS, FMS'den ve dolayısıyla robottan ayrıldığında, maç sırasında günlüğü parçalara ayırabilir. Bu problemin temel göstergeleri, 1. günlüğün son olayı olup, FMS'ye bağlantının artık "bad" olduğunu ve 2. günlükten gelen ikinci olaydır; bu, yeni bir FMS bağlı mesajdır ve ardından DS'nin hemen Teleop Enabled'e geçiş yapması gelir. . Bu tür bir arızanın en yaygın nedeni, mandalı olmayan bir ethernet kablosu veya DS bilgisayarında hasarlı bir ethernet bağlantı noktasıdır.

roboRIO Reboot-Yeniden başlatmak



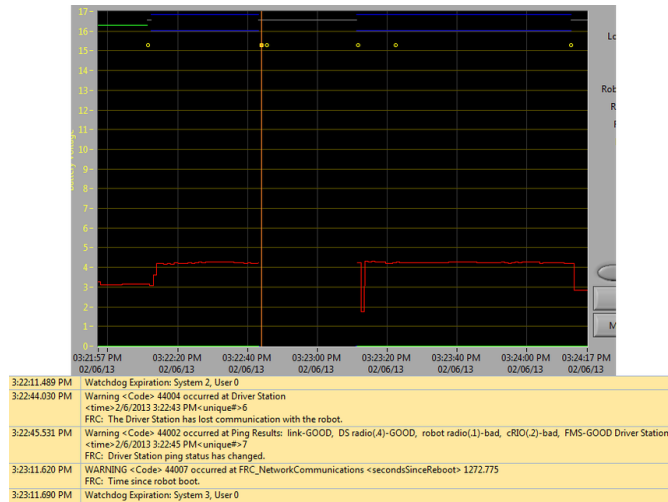
“Time since robot boot” mesajı, roboRIO’nun yeniden başlatılmasının neden olduğu bir bağlantı hatasındaki birincil göstergedir. Bu günlükte DS, ilk olayda belirtildiği gibi 3:01:36’da roboRIO ile bağlantısını kaybeder. İkinci olay, bağlantı başarısız olduktan sonra başlatılan ping’in roboRIO dışındaki tüm cihazlarda başarılı olduğunu gösterir. 3:01:47’de roboRIO pinglere tekrar yanıt vermeye başlar, 3:01:52’de ek bir ping başarısız olur. 3:02:02’de Driver Station roboRIO’ya bağlanır ve roboRIO bunun 3.682 saniyedir açık olduğunu bildirir. Bu, roboRIO’nun yeniden başlatıldığının açık bir göstergesidir. Kod yüklenmeye devam ediyor ve 3:02:24’te kod, kamera ile iletişimde bir hata olduğunu bildiriyor. Kodun başlaması tamamlanmadan hemen önce hiçbir robot kodunun çalışmadığını belirten bir uyarı da rapor edilmiştir.

Robot üzerindeki Ethernet kablo problemleri



Robot üzerindeki ethernet kablosuyla ilgili bir sorun, öncelikle roboRIO'nun kötüye giden pingi ve roboRIO yeniden bağlandığında Telsiz Kayboldu ve Telsiz Görüldü olayları ile gösterilir. RoboRIO yeniden bağlandığında "Time since robot boot-Robot açılışından itibaren geçen süre" mesajı, roboRIO'nun yeniden başlatılmadığını da gösterir. Bu örnekte, robot Ethernet kablosunun bağlantısı 3:31:38'de kesildi. Ping durumu, radyonun hala bağlı olduğunu gösterir. Robot 3:32:08 de yeniden bağlandığında "Time since robot boot-Robot açılışından itibaren geçen süre" 1809 saniyedir ve bu roboRIO'nun açıkça yeniden başlamadığını gösterir. 3:32:12'de robot, radyoyu 24.505 saniye önce kaybettiğini ve 0.000 saniye önce geri geldiğini belirtir. Bu noktalar grafikte dikey çizgiler olarak, radyo kayıpları için sarı ve radyo görülenler için yeşil olarak işaretlenmiştir. Bağlantının kesilmesi ve bağlantı yoluyla gösterildiği gibi zamanların gerçek olaylardan biraz farklı olduğunu, ancak ne olduğu hakkında ek bilgi sağlamaya yardımcı olduğunu unutmayın.

Radio yeniden başlatma-reboot



Robot telsizinin yeniden başlatılması, tipik olarak telsizle ~40-45 saniye süreyle bağlantının kesilmesiyle karakterize edilir. Bu örnekte, radyo kısa bir süre 3:22:44 de güç kaybetti ve bu

da yeniden başlatılmasına neden oldu. 3:22:45 olay, telsize gelen ping'in başarısız olduğunu gösterir. 3:23:11'de DS, roboRIO ile iletişimi yeniden kazanır ve roboRIO 1272.775 saniyedir açık olduğunu belirtir. Telsizdeki ağ cihazının çok hızlı bir şekilde geri geldiğini unutmayın, bu nedenle anlık bir güç kaybı, "telsizin kaybolması"/"telsizin görülmesi" olay çiftiyle sonuçlanmayabilir. Daha uzun bir bozulma, radyo olaylarının DS tarafından günlüğe kaydedilmesine neden olabilir. Bu durumda, telsizin yeniden başlatılmasına işaret eden ayırt edici faktör, DS'den gelen radyonun ping durumudur. Radyo sıfırlanırsa, telsize ulaşamaz. Sorun robotta bir kablolama veya bağlantı sorunuysa, radyo pingi "Good" kalmalıdır.

20.4 Driver Station Hataları / Uyarıları

In an effort to provide both Teams and Volunteers (*FTA / CSA / etc.*) more information to use when diagnosing robot problems, a number of Warning and Error messages have been added to the Driver Station. These messages are displayed in the DS diagnostics tab when they occur and are also included in the DS Log Files that can be viewed with the Log File Viewer. This document discusses the messages produced by the DS (messages produced by WPILib can also appear in this box and the DS Logs).

20.4.1 Joystick'in bağlantısız olması

```
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.
```

Bu hata, bir Joystick bağlantısız olduğunda tetiklenir. Mesaj metninin aksine bu hata, robot etkinleştirilmemiş veya hatta DS'ye bağlı olmasa bile oluşacaktır. Joystick'ler doğru şekilde bağlanmış ve çalışıyor olsa bile, Driver Station her başlatıldığında bu mesajın tek bir örneğinin oluştuğunu göreceksiniz.

Not: Joystick Unplugged warnings can be silenced by calling `DriverStation.silenceJoystickConnectionWarning(true)` (Java, C++)

20.4.2 İletişimi Kaybetmek

```
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.
```

Bu Uyarı mesajı, Driver Station robotla iletişimi kaybettiğinde oluşur (İletişim göstergesi yeşilden kırmızıya döner). Bu mesajın tek bir örneği, iletişim kurulmadan önce DS başladığında oluşur.

20.4.3 Ping Durumu

```
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 Robot Önyüklemesinden Bu Yana Geçen Süre

```
WARNING<Code>44007 occurred at FRC_NetworkCommunications
**<secondsSinceReboot> 3.585**
FRC: Time since robot boot.
```

Bu mesaj, DS roboRIO ile her iletişim kurmaya başladığında oluşur. Mesaj roboRIO'nun çalışma süresini saniye cinsinden belirtir ve bir roboRIO Yeniden Başlatılması nedeniyle iletişim kaybının olup olmadığını belirlemek için kullanılabilir.

20.4.5 Radyo-Modem Algılama Süreleri

```
WARNING<Code>44008 occurred at FRC_NetworkCommunications
<radioLostEvents> 19.004<radioSeenEvents> 0.000
FRC: Robot radio detection times

WARNING<Code>44008 occurred at FRC_NetworkCommunications
<radioLostEvents> 2.501,422.008<radioSeenEvents> 0.000,147.005
FRC: Robot radio detection times.
```

Bu mesaj, DS roboRIO ile iletişim kurmaya başladığında yazdırılabilir ve modem en son kaybolup görülmesinden bu yana geçen süreyi saniye cinsinden gösterir. Yukarıdaki ilk örnek görüntüde mesaj, roboRIO'nun telsiz bağlantısının mesaj yazdırılmadan 19 saniye önce kesildiğini ve mesaj yazdırıldığında telsizin tekrar görüldüğünü gösterir. RoboRIO'nun başlatılmasından bu yana birden fazla radioLost veya radioSeen olayı meydana gelirse, virgülle ayrılmış her türden en fazla 2 olay dahil edilecektir.

20.4.6 Robot Kodu Yok

```
Warning<Code>44003 occurred at Driver Station
<time>2/8/2013 9:50:13 AM<unique#>8
FRC: No robot code is currently running.
```

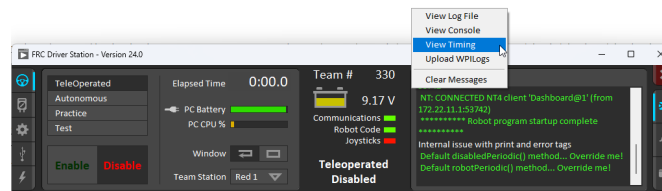
Bu mesaj, DS roboRIO ile iletişim kurmaya başladığında, ancak çalışan robot kodu algılandığında oluşur. RoboRIO önyüklenirken Driver Station açıksa ve çalışıyorsa bu mesajın tek bir örneği oluşacaktır, çünkü DS robot kodu yüklemeyi bitirmeden önce roboRIO ile iletişime başlayacaktır.

20.5 Driver Station Timing Viewer

The 2024 Driver Station has a new window to help diagnose robot control issues.

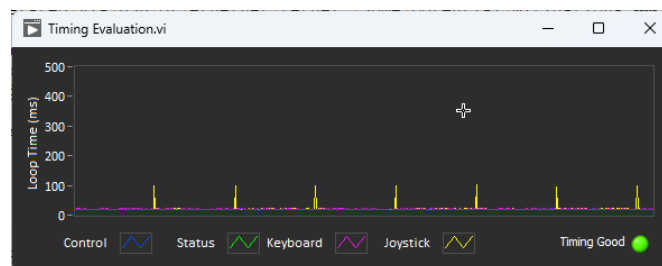
20.5.1 Opening the Timing Viewer

To start the Driver Station Timing Viewer, select the gear icon and then select *View Timing*.

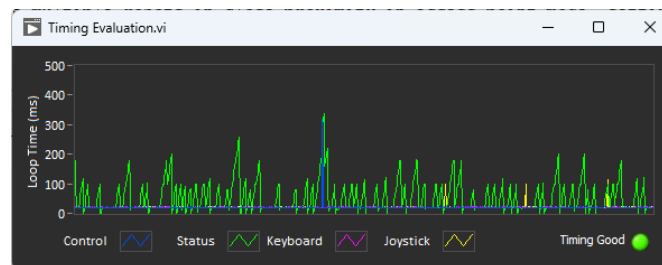


20.5.2 Viewing Timing

The Timing viewer shows the timing of the driver station loops measuring the joysticks, 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 than a second.



20.6 FMS Offseason için Radyo Programlama

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 Önkoşullar

Install the FRC® Radio Configuration Utility software per the instructions in *Programming your radio*

Yazılımı kullanmaya başlamadan önce:

1. Yapılandırma yardımcı programının köprü ile düzgün bir şekilde iletişim kurmasını engelleyebileceğinden bilgisayarınızdaki WiFi bağlantılarını devre dışı bırakın.
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.

Programlanmış Configuration

Radio Configuration Utility Programı, çalıştırıldığında radyoya bir dizi yapılandırma ayarı programlar. Bu ayarlar tüm modlarda (etkinliklerde dahil) telsiz için geçerlidir. Bunlar şunları içerir:

- 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
- Kablolu bağlantı noktalarını köprüleyin, böylece birbirlerinin yerine kullanılabilirler
- 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.

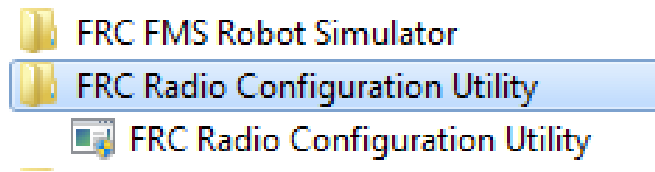
Tüyo: Yapılandırıldığında telsiz durum ışıklarının davranışı hakkında ayrıntılar için bkz: ref: Durum Işığı Referansı <docs/hardware/hardware-basics/status-lights-ref:OpenMesh Radio>.

Radio Configuration yardımcı programının takım sürümü ile programlandığında, kullanıcı hesapları **sadece DAP'ler için** fabrika ayarı varsayılanları olarak bırakılır (veya ayarlanır)

- Kullanıcı adı : root
- Şifre : root

Not: Yapılandırmayı manuel olarak değiştirmeniz önerilmez

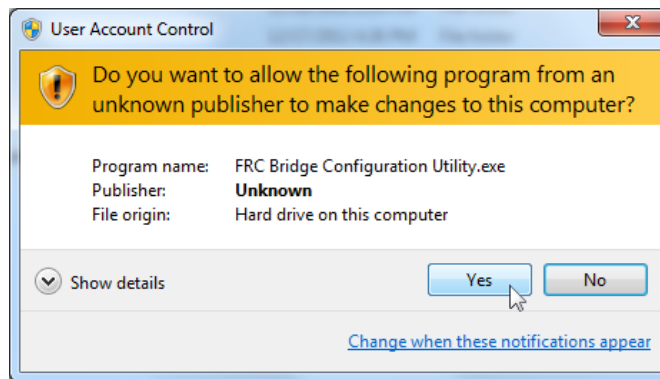
20.6.2 Yazılımı başlatın



Programı başlatmak için Start menüsünü veya masaüstü kısayolunu kullanın.

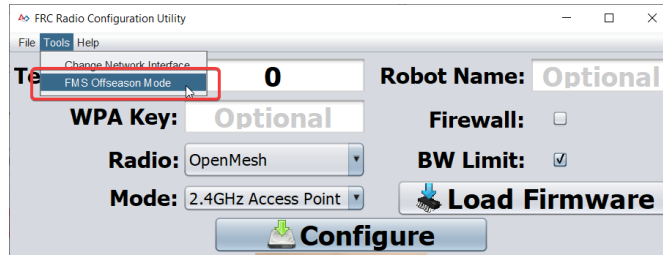
Not: If you need to locate the program, it is installed to C:\Program Files (x86)\FRC Radio Configuration Utility. For 32-bit machines the path is C:\Program Files\FRC Radio Configuration Utility

20.6.3 İstenirse programın değişiklik yapmasına izin verin



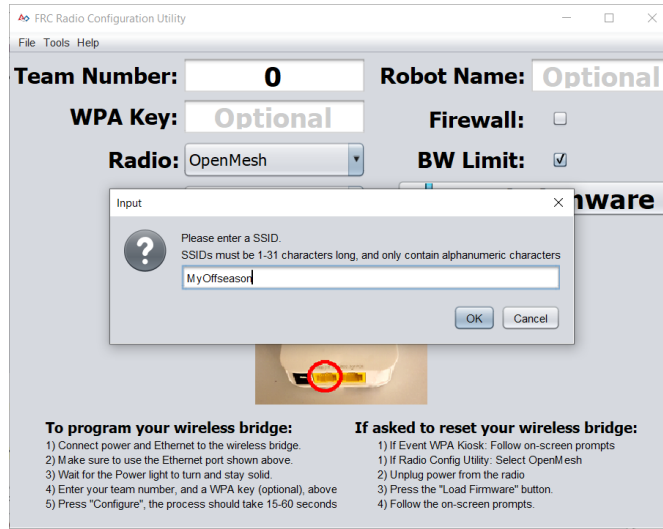
A prompt may appear about allowing the configuration utility to make changes to the computer. Click Yes if the prompt appears.

20.6.4 Enter FMS Offseason Mode



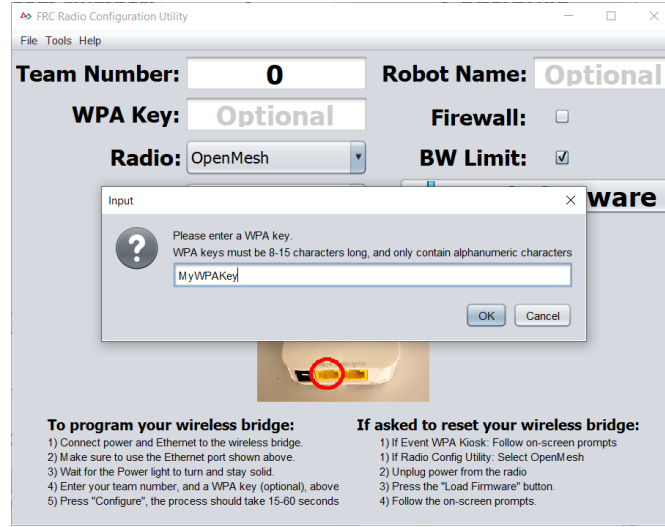
FMS-Lite Mode'a girmek için Tools -> FMS-Lite Mode tıklayınız.

20.6.5 SSID girin



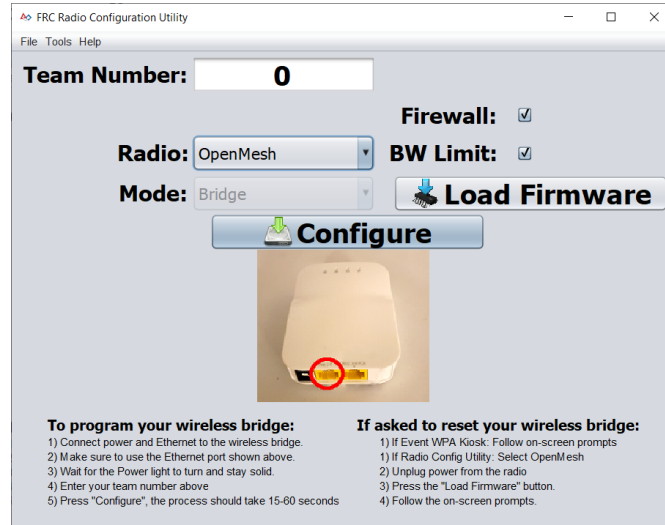
Kablosuz ağınızın SSID'sini (adını) kutuya girin ve Tamam'ı tıklayın.

20.6.6 WPA Key girin



Ağınız için WPA anahtarını kutuya girin ve Tamam'ı tıklayın. Güvenli olmayan bir ağ kullanıyorsanız kutuyu boş bırakın.

20.6.7 Radyoları Programlama



Kiosk artık girilen ağa bağlanmak için herhangi bir sayıda radyoyu programlamaya hazırdır. Her bir radyoyu programlamak için, radyoyu Kiosk'a bağlayın, kutuda Team Number ayarlayın ve Configure'e tıklayın.

Kiosk, OpenMesh, D-Link Rev A veya D-Link Rev B telsizlerini "Radio" açılır menüsünden uygun seçeneği seçerek sezon dışı bir FMS ağında çalışacak şekilde programlayacaktır.

Not: Bu modda D-Link telsizlerinde bant genişliği sınırlamaları ve QoS yapılandırılmayacaktır.

20.6.8 SSID veya Anahtarın Değiştirilmesi

Yanlış bir şey girerseniz veya SSID veya WPA anahtarını değiştirmeniz gerekirse, kiosku FMS-Lite Modundan çıkarmak için Tools menüsüne gidin ve FMS-Lite Moduna tıklayın. Kiosk'u FMS-Lite Moduna geri getirmek için tekrar tıkladığınızda, SSID ve Anahtar için yeniden sorulacaktır.

20.6.9 Troubleshooting

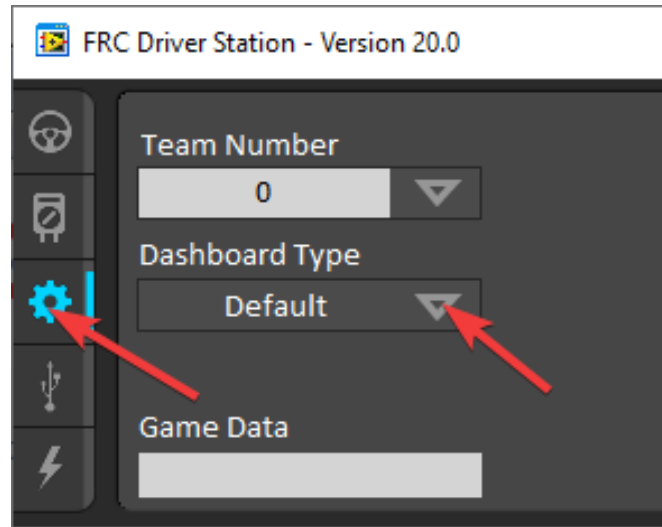
See the troubleshooting steps in *Programming your radio*

20.7 Driver Station'ı Özel Panoyu Başlatacak Şekilde Manuel Olarak Ayarlama

Not: WPILib varsayılan konuma yüklenmemişse (örneğin, dosyalar bir PC'ye manuel olarak kopyalandığında), tercih edilen kontrol paneli düzgün başlatılamayabilir. DS'nin başladığında özel bir kontrol paneli başlatmasını sağlamak için varsayılan kontrol panelinin ayarlarını manuel olarak değiştirmeniz gerekir.

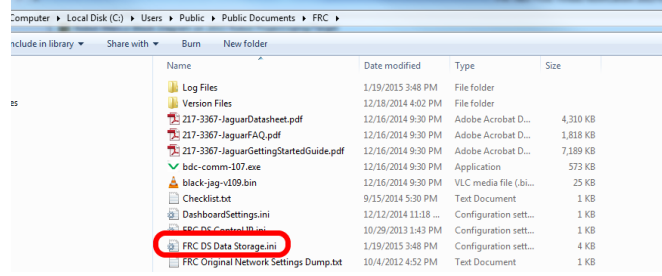
Uyarı: Çoğu kurulum için buna gerek yoktur, önce diliniz için uygun: ref: `Dashboard Type` ayarı <docs / software / driverstation / driver-station: Setup Tab> 'seçeneğini kullanmayı deneyin.

20.7.1 Driver Station'ı Varsayılanı Ayarlayın



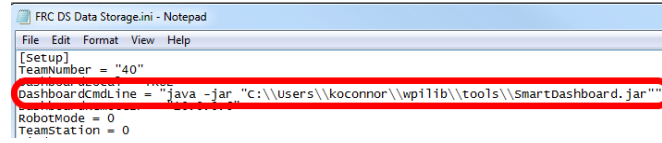
Driver Station yazılımını açın, Setup sekmesine tıklayın ve Dashboard ayarını Default olarak ayarlayın. ** Ardından Driver Station'ı kapatın! **

20.7.2 DS Veri Depolama dosyasını açın



C:\Users\Public\Documents\FRC ye göz atın ve açmak için FRC DS Data Storage üzerine çift tıklayın.

20.7.3 DashboardCmdLine -dashboard komut satırı



``DashboardCmdLine`` ile başlayan satırı bulun. Sürücü istasyonu başladığında başlatılması için kontrol panelini gösterecek şekilde değiştirin

LabVIEW Özel Kontrol Paneli

= 'den sonraki dizeyi şununla değiştirin: "C:\\PATH\\TO\\DASHBOARD.exe" burada belirtilen yol, gösterge tablosu exe dosyasına giden yoldur. `` FRC DS Data Storage`` dosyasını kaydedin.

Java Dashboard- Java Kontrol Paneli

" = " 'Den sonraki dizeyi `` java -jar "C: \ PATH \ TO \ DASHBOARD.jar" "ile değiştirin; burada belirtilen yol, gösterge panosu ` jar` 'dosyasına giden yoldur. ` FRC DS Data Storage `` dosyasını kaydedin.

Tüyo: Shuffleboard ve Smartdashboard için Java 11 gerekir.

WPILib yükleyiciden Dashboard

= 'den sonraki dizeyi wscript ile değiştirin C: \ Users \ Public \ wpilib \ YYYY \ tools \ DASHBOARD.vbs `` i burada `` YYYY`` i yıl ve `` `` DASHBOARD.vbs `` , `` Shuffleboard.vbs `` ıveya `` Smartdashboard.vbs`` ıdır. `` FRC DS Data Storage`` ıdosyasını kaydedin.

20.7.4 Driver Station'ı başlatın

Driver Station artık her açıldığında kontrol panelini başlatmalıdır.

21.1 RobotBuilder - Giriş

21.1.1 RobotBuilder'a Genel Bakış

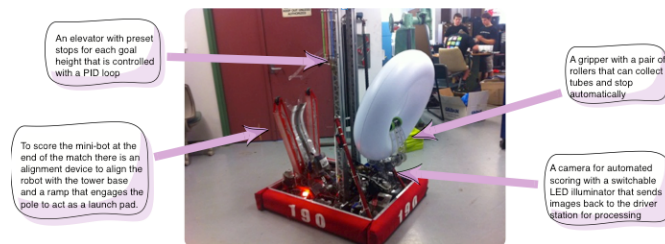
RobotBuilder, robot geliştirme sürecine yardımcı olmak için tasarlanmış bir uygulamadır. RobotBuilder size şu konularda yardımcı olabilir:

- Standart kod üretiliyor.
- Robotunuzu düzenleyin ve temel alt sistemlerinin neler olduğunu bulun.
- Tüm sensörleriniz ve aktüatörleriniz için yeterli kanalınız olup olmadığını kontrol edin.
- Kablolama şemaları oluşturun.
- Operatör arayüzünüzü kolayca değiştirin.
- Daha...

RobotBuilder ile bir program oluşturmak, herhangi bir robot için aynı olan birkaç adımı izleyerek çok basit bir işlemdir. Bu ders takip edebileceğiniz adımları açıklamaktadır. Belgenin sonraki bölümlerinde bu adımların her biri hakkında daha fazla ayrıntı bulabilirsiniz.

Not: RobotBuilder, yeni Command Framework kullanarak kod üretir. Yeni çerçeve hakkında daha fazla ayrıntı için bkz [Command Based Programming](#).

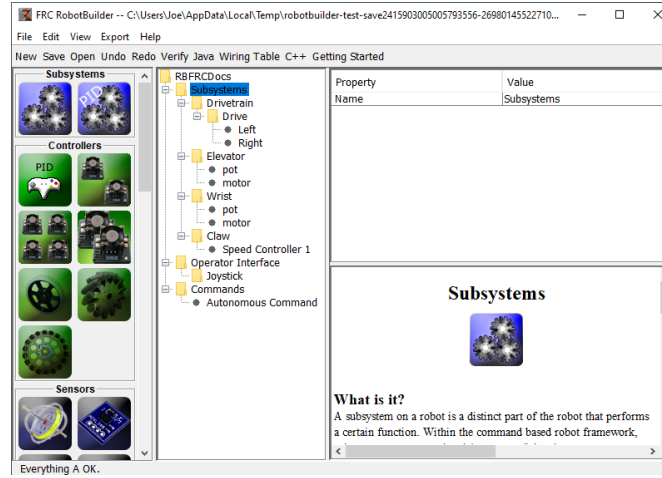
Robotu Subsystems-Alt Sistemlere Ayırın



Robotunuz doğal olarak tahrik trenleri, kollar, atıcılar, toplayıcılar, manipülatörler, bilek eklemleri vb. Gibi bir dizi küçük sistemden oluşur. Robotunuzun tasarımına bakmalı ve onu daha küçük, ayrı çalıştırılan alt sistemlere bölmelisiniz. Bu özel örnekte bir asansör, bir minibot hizalama cihazı, bir kavrayıcı ve bir kamera sistemi bulunmaktadır. Ek olarak sürücü tabanı da içerebilir. Robotun bu parçalarının her biri ayrı ayrı kontrol edilir ve alt sistemler için iyi adaylar oluşturur.

Daha fazla bilgi için bkz : doc: “Bir Alt Sistem Oluşturma <robotbuilder-creating-subsystem>”.

Her Alt Sistemi Projeye Ekleme



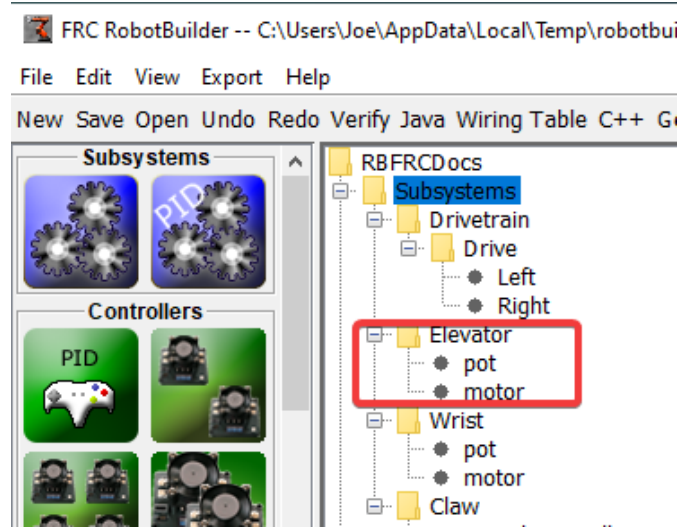
Her alt sistem, RobotBuilder'daki “Subsystems-Alt Sistemler” klasörüne eklenecek ve anlamlı bir ad verilecektir. Alt sistemlerin her biri için, alt sistemler hakkında daha fazla bilgi belirtmek için doldurulan birkaç öznelik vardır. Ek olarak, oluşturmak isteyebileceğiniz iki tür alt sistem vardır:

1. PIDS alt sistemleri - genellikle bir alt sistemin çalışmasını bir PID denetleyicisi ile kontrol etmek istenir. Bu, programınızdaki alt sistem elemanını, örneğin kol açısı gibi, istenen bir konuma daha hızlı ve sonra ulaşırken durmasını sağlayan koddur. PIDSubsystems yerleşik PID Denetleyici koduna sahiptir ve genellikle kendiniz eklemekten daha kullanışlıdır. PIDS alt sistemleri, cihazın hedef konuma ne zaman ulaştığını belirleyen bir sensöre ve ayar noktasına sürülen bir aktüatöre (motor kontrolörü) sahiptir.
2. Normal alt sistem - bu alt sistemlerin entegre bir PID denetleyicisi yoktur ve geri bildirim için PID kontrolü olmayan alt sistemler için veya varsayılan yerleşik PID denetleyiciyle ele alınabilecek olandan daha karmaşık denetim gerektiren alt sistemler için kullanılır.

Bu belgelerin çoğuna baktıkça, alt sistem türleri arasındaki farklar daha belirgin hale gelecektir.

Daha fazla bilgi için bkz : doc: “Creating a Subsystem <robotbuilder-creating-subsystem>” ve : ref: *Write Code for a Subsystem <docs/software/wpilib-tools/robotbuilder/writing-code/robotbuilder-writing-subsystem-code:Writing the Code for a Subsystem>*.

Alt Sistemlerin her birine Bileşen Ekleme



Her bir alt sistem, işlemlerini gerçekleştirmek için kullandığı bir dizi aktüatör, sensör ve denetleyiciden oluşur. Bu sensörler ve aktüatörler, ilişkili oldukları alt sisteme eklenir. Sensörlerin ve aktüatörlerin her biri, RobotBuilder paletinden gelir ve uygun alt sisteme sürüklenir. Her biri için, genellikle bağlantı noktası numaraları ve bileşene özgü diğer parametreler gibi ayarlanması gereken başka özellikler vardır.

Bu örnekte, Elevator alt sistemine sürüklenen bir motor ve bir potansiyometre (motor ve pot) kullanan bir Elevator alt sistemi vardır.

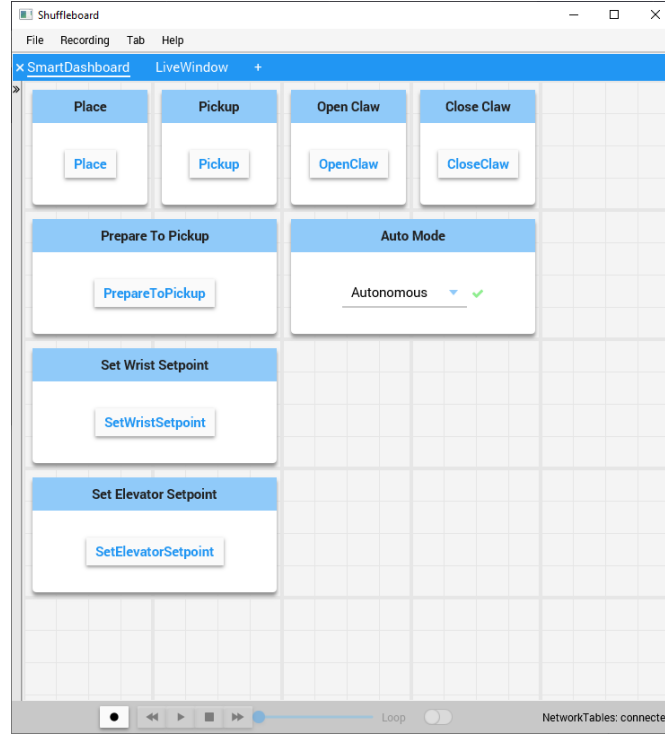
Alt Sistem Hedeflerini Açıklayan Komutlar Ekleme

Komutlar, robotun gerçekleştireceği farklı hedeflerdir. Bu komutlar, komut "Commands" klasörünün altına sürüklenerek eklenir. Bir komut oluştururken 7 seçenek vardır (resmin solundaki palette gösterilir):

- Normal komutlar - bunlar en esnek komutlardır, hedefe ulaşmak için gerekli eylemleri gerçekleştirmek için tüm kodu yazmanız gerekir.
- Zamanlanmış komutlar - bu komutlar, bir zaman aşımından sonra sona eren bir komutun basitleştirilmiş bir sürümüdür.
- Instant commands - these commands are a simplified version of a command that runs for one iteration and then ends
- Komut grupları - bu komutlar, hem sıralı hem de paralel olarak çalışan diğer komutların birleşimidir. Bir dizi temel komut uygulandıktan sonra daha karmaşık eylemler oluşturmak için bunları kullanın.
- Ayar noktası komutları - ayar noktası komutları bir PID Alt Sistemini sabit bir ayar noktasına veya istenen konuma taşır.
- PID komutları - bu komutlar, normal bir alt sistemle kullanılmak üzere yerleşik bir PID denetleyicisine sahiptir.
- Koşullu komutlar - bu komutlar, başlatma anında çalıştırılacak iki komuttan birini seçer.

Daha fazla bilgi için bakınız : doc: ` Bir Komut Oluşturma <robotbuilder-creating-command>`
ve : ref: ` Komut Kodunu Yazma <docs/software/wpilib-tools/robotbuilder/writing-code/robotbuilder-writing-command-code:Writing the Code for a Command>` .

Her Komutu Test Etme

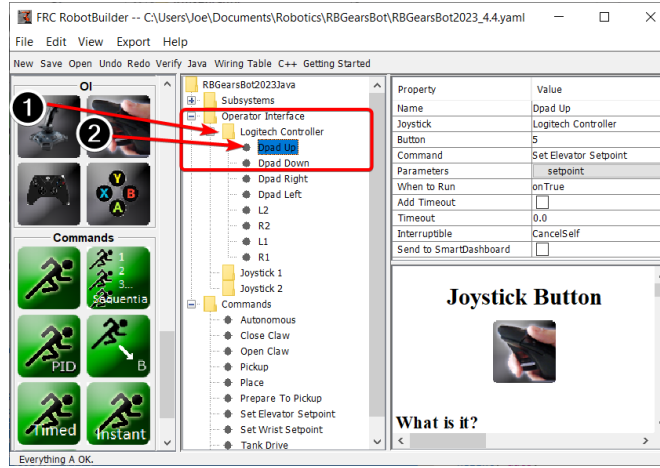


Her komut Shuffleboard veya SmartDashboard'dan çalıştırılabilir. Bu, komutları operatör arayüzüne veya bir komut grubuna eklemekten önce test etmek için kullanışlıdır. "Button on SmartDashboard" özelliğini işaretli bıraktığınız sürece, SmartDashboard'da bir düğme oluşturulur. Düğmeye bastığınızda, komut çalışacak ve istediğiniz eylemi gerçekleştirip gerçekleştirmediyini kontrol edebilirsiniz.

Düğmeler oluşturularak her komut ayrı ayrı test edilebilir. Tüm komutlar ayrı ayrı çalışıyorsa, robotun bir bütün olarak çalışacağından oldukça emin olabilirsiniz.

Daha fazla bilgi için bkz [Testing with Smartdashboard](#).

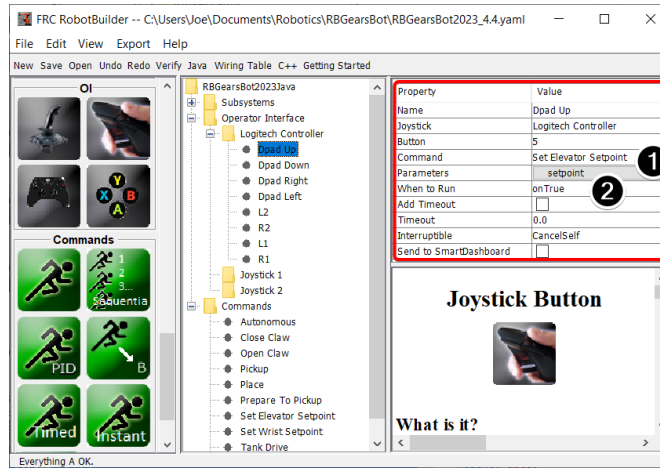
Operatör Arayüzü Bileşenlerinin Ekleneşi



Operatör arayüzü, kumanda kollarından, oyun kumandalarından ve diğer HID giriş cihazlarından oluşur. RobotBuilder'daki programınıza operatör arayüzü bileşenleri (kumanda kolları, kumanda kolu düğmeleri) ekleyebilirsiniz. Tüm bileşenleri başlatacak ve komutlara bağlanmalarına izin verecek kodu otomatik olarak üretecektir.

Operatör arayüz bileşenleri paletten RobotBuilder programındaki "Operator Interface" klasörüne sürüklenir. Önce (1) programa Joystick'i ekleyin, ardından ilgili joysticklerin (2) altına düğmeler koyun ve bunlara ShootButton gibi anlamlı adlar verin.

Komutların Operatör Arayüzüne Bağlanması

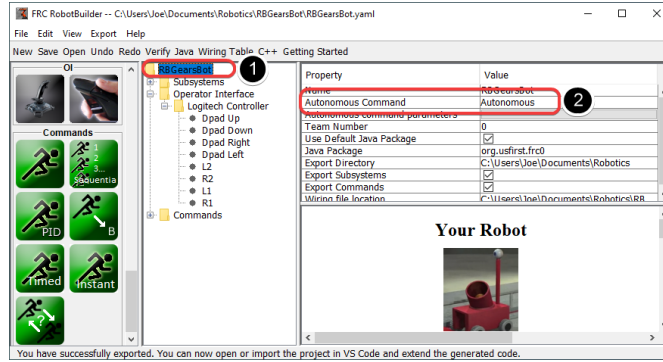


Komutlar düğmelerle ilişkilendirilebilir, böylece bir düğmeye basıldığında komut zamanlanır. Bu, çoğunlukla, robot programınızın uzaktan çalıştırılan bölümünün çoğunu işlemelidir.

Bu, RobotBuilder programındaki JoystickButton nesnesine (1) komutu ekleyerek, ardından (2) komutun programlandığı koşulu ayarlayarak yapılır.

Daha fazla bilgi için bkz : [doc:Operatör Arayüzünü Komuta Bağlama <robotbuilder-operator-interface-to-command>](#).

Otonom Komutlar Geliştirme

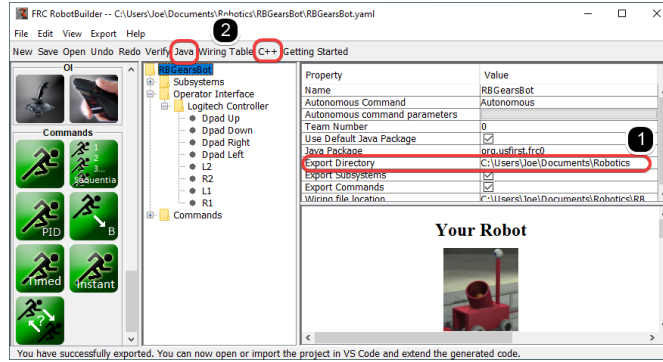


Komutlar, otonom programlar geliştirmeyi kolaylaştırır. Robot otonom döneme girdiğinde hangi komutun çalıştırılması gerektiğini belirtirsiniz ve otomatik olarak programlanacaktır. Komutları yukarıda tartışıldığı gibi test ettiyseniz, bu sadece hangi komutun çalıştırılması gerektiğini seçme meselesi olmalıdır.

RobotBuilder projesinin (1) kökündeki robotu seçin, ardından çalıştırılacak komutu seçmek için Otonom Komut özelliğini (2) düzenleyin. Bu kadar basit!

Daha fazla bilgi için bkz [Setting the Autonomous Commands](#).

Kod Oluşturuluyor



Yukarıda belirtilen sürecin herhangi bir noktasında, RobotBuilder'ın oluşturduğunuz projeyi temsil edecek bir C++ veya Java programı oluşturmasını sağlayabilirsiniz. Bu, proje özelliklerinde (1) projenin konumu belirtilerek ve ardından kodu oluşturmak için uygun araç çubuğu düğmesine tıklanarak yapılır (2).

Daha fazla bilgi için bkz [Generating RobotBuilder Code](#).

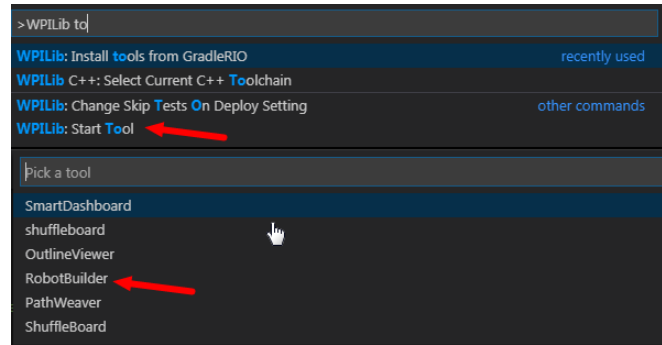
21.1.2 RobotBuilder'ı Başlatma

Not: RobotBuilder bir Java programıdır ve bu nedenle Java tarafından desteklenen herhangi bir platformda çalışabilmelidir. RobotBuilder'ı macOS, Windows ve Linux'un çeşitli sürümlerinde başarıyla çalıştırıyoruz.

RobotBuilder'ı Edinme

RobotBuilder, WPILib Çevrimdışı Yükleyicinin bir parçası olarak indirilir. Daha fazla bilgi için bkz [Windows/macOS/Linux installation guides](#)

Seçenek 1 - Visual Studio Kodundan Başlama

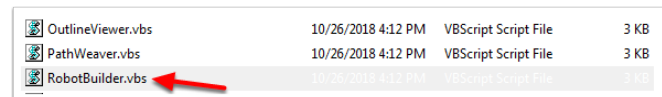


WPILib Komut Paletini başlatmak için : kbd: *Ctrl + Shift + P* 'ye basın ve "WPILib" yazın veya sağ üstteki WPILib logosunu tıklayın. Seçin : guilabel: `Start Tool, ardından : guilabel: `Robot Builder` seçin.

Seçenek 2 - Kısayollar

Shortcuts are installed to the Windows Start Menu and the 2024 WPILib Tools folder on the desktop.

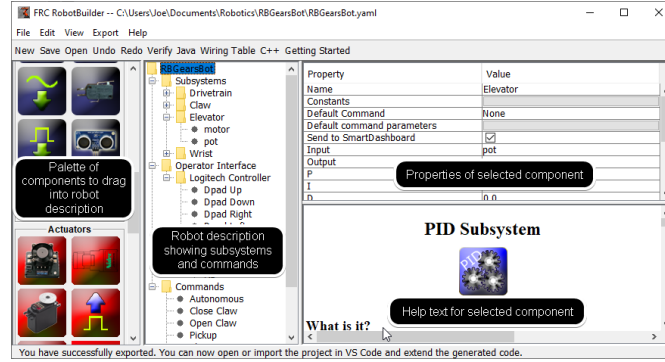
Seçenek 3 - Komut Dosyasından Çalıştırmak



Yükleme işlemi, araçları `` ~/wpilib/ YYYY/tools `` konumuna yükler (burada YYYY yılı ve ~, Windows'ta C:\Users \Public şeklindedir).

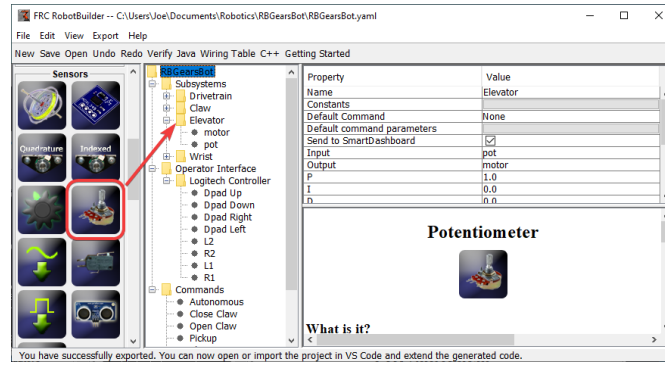
Bu klasörün içinde, her aracı başlatmak için kullanabileceğiniz .vbs (Windows) ve .py (macOS / Linux) dosyalarını bulacaksınız. Bu komut dosyaları, araçları doğru JDK kullanarak başlatmaya yardımcı olur ve araçları başlatmak için kullanmanız gerekenlerdir.

21.1.3 RobotBuilder Kullanıcı Arayüzü



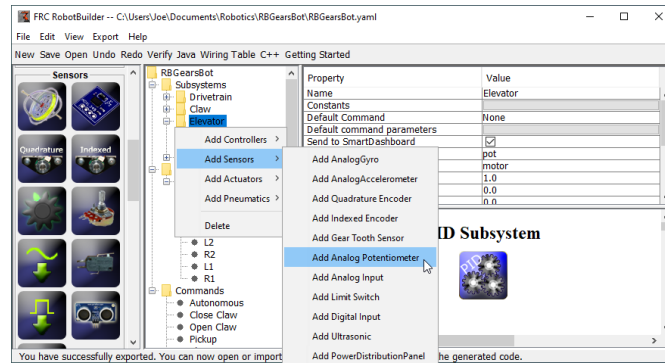
RobotBuilder, robot programlarının hızlı gelişimi için tasarlanmış bir kullanıcı arayüzüne sahiptir. Hemen hemen tüm işlemler sürükleyip bırak veya açılır listelerden seçenekler belirleyerek gerçekleştirilir.

Öğeleri Paletten Robot Açıklamasına Sürüklemek



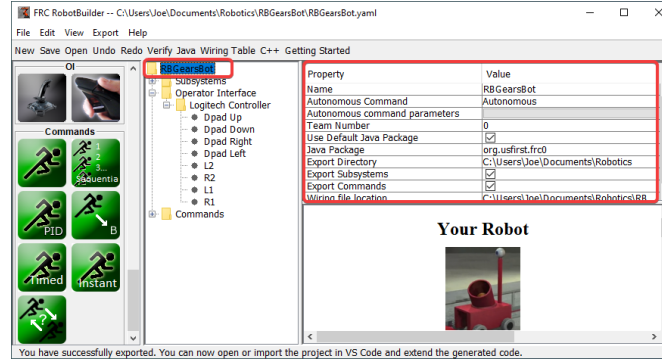
Palet öğesi üzerinde sürüklemeyi başlatarak ve öğenin yerleştirilmesini istediğiniz kapta sonlandırarak öğeleri paletten robot açıklamasına sürükleyebilirsiniz. Bu örnekte, Asansör alt sistemine bir potansiyometre düşürülüyor.

Sağ Tıklama Bağlam Menüsünü Kullanarak Bileşen Ekleme



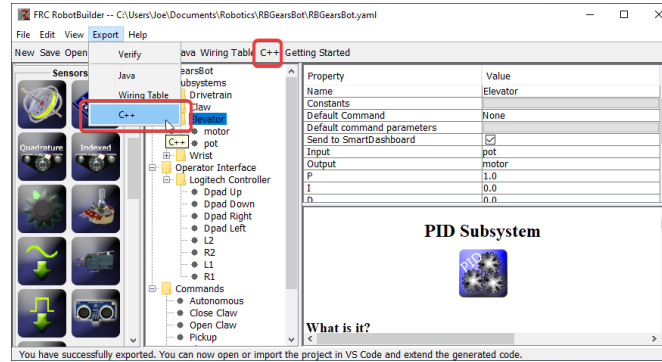
Robot açıklamasına öğe eklemenin bir kısayol yöntemi, konteyner nesnesine (Kaldırıcı) sağ tıklamak ve eklenmesi gereken öğeyi (Potansiyometre) seçmektir. Bu, sürükle ve bırak işlemiyle aynıdır ancak bazı insanlar için daha kolay olabilir.

Robot Tanımı Öğelerinin Özelliklerini Düzenleme



Seçilen bir öğenin özellikleri, özellikler görüntüleyicide görünecektir. Özellikler, sağ taraftaki sütundaki değer seçilerek düzenlenebilir.

Menü Sisteminin Kullanılması



RobotBuilder için işlemler, menü sistemi veya araç çubuğundaki eşdeğer öğe (varsa) aracılığıyla seçilebilir.

21.1.4 Robot Projesini Ayarlamak

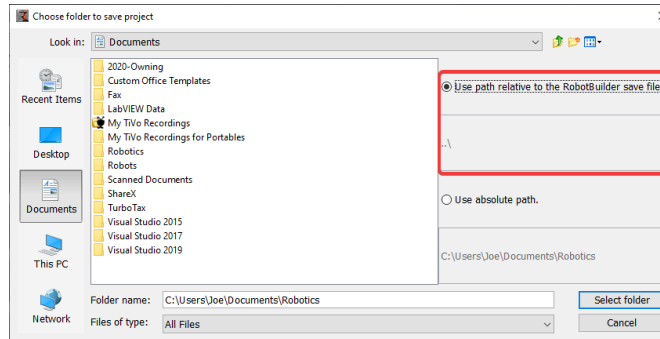
RobotBuilder programı, oluşturulan programın ve diğer oluşturulan dosyaların düzgün çalışması için ayarlanması gereken bazı varsayılan özelliklere sahiptir. Bu kurulum bilgileri, robot açıklaması özelliklerinde (ilk satır) saklanır.

Robot Project Properties

Robotu tanımlayan özellikler şunlardır:

- **Name** - Oluşturulan robot projesinin adı
- **Autonomous Command** - program otonom moda getirildiğinde varsayılan olarak çalışacak komut
- **Autonomous Command Parameters** - Otonom Komut için Parametreler
- **Team Number** - Kodu dağıtırken robotu bulmak için kullanılacak proje takım numarası.
- **Use Default Java Package** - İşaretlenirse, RobotBuilder varsayılan paketi (frc.robot) kullanacaktır. Aksi takdirde, kullanılacak özel bir paket adı belirtebilirsiniz.
- **Java Package** - Proje kodunu oluştururken kullanılan, oluşturulan Java paketinin adı
- **Export Directory** - Java veya C ++ 'ya Aktar seçildiğinde projenin oluşturulduğu klasör
- **Export Subsystems** - RobotBuilder'ın Alt Sistem sınıflarını projenizden dışa aktarması gerekecekse seçilecektir
- **Export Commands** - Eğer seçiliyse RobotBuilder'ın projenizden Komut sınıflarını dışa aktarması gerekip gerekmediği kontrol eder
- **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.

RobotBuilder Projesi ile Kaynak Kontrolünü Kullanma

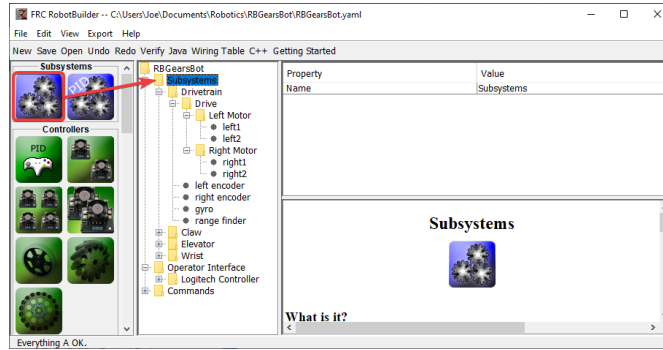


Kaynak kontrolü kullanılırken proje tipik olarak birkaç bilgisayarda kullanılacaktır ve proje dizinine giden yol bir kullanıcının bilgisayarından diğerine farklı olabilir. RobotBuilder proje dosyası mutlak bir yol kullanılarak depolanırsa, genellikle kullanıcı adını içerir ve birden çok bilgisayarda kullanılamaz. Bunun çalışması için, "relative path" u seçin ve yolu proje dosyasından bir dizin uzaklığı olarak belirtin. Yukarıdaki örnekte, proje dosyası, dosya hiyerarşisindeki proje dosyalarının hemen üzerindeki klasörde saklanır. Bu durumda, kullanıcı adı yolun bir parçası değildir ve tüm bilgisayarlarınızda taşınabilir olacaktır.

21.1.5 Subsystem-Alt Sistem Oluřturma

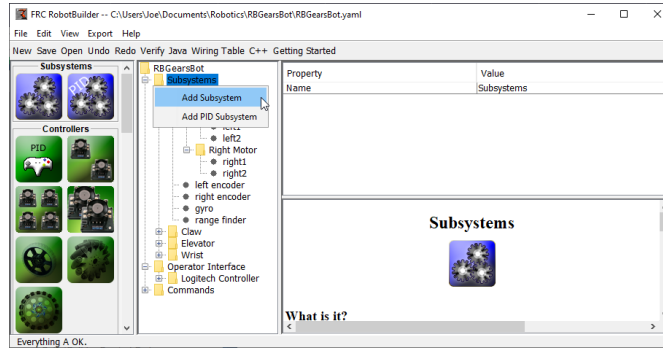
Subsystems are classes that encapsulate (or contain) all the data and code that make a subsystem on your robot operate. The first step in creating a robot program with the RobotBuilder is to identify and create all the subsystems on the robot. Examples of subsystems are grippers, ball collectors, the 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.

Paleti Kullanarak Alt Sistem Oluřturma



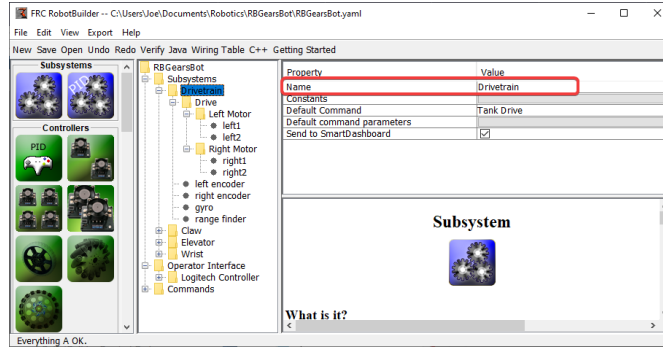
Bir alt sistem sınıfı oluřturmak iin alt sistem simgesini paletten robot aıklamasındaki Alt Sistemler klasrne srkleyin.

Baėlam Mensn Kullanarak Alt Sistem Oluřturma



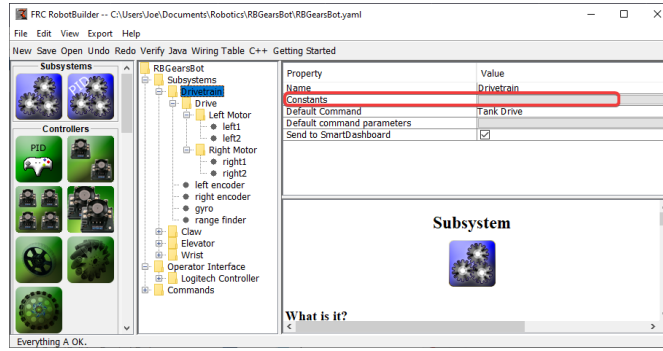
Bu klasre bir alt sistem eklemek iin robot aıklamasındaki Subsystem-Alt Sistem klasrne saė tıklayın.

Subsystem-Alt Sistemi adlandırın



Alt sistemi ya sürükleyerek ya da yukarıda açıklandığı gibi bağlam menüsünü kullanarak oluşturduktan sonra, alt sisteme vermek istediğiniz adı yazmanız yeterlidir. Ad, boşluklarla ayrılmış birden çok kelime olabilir, RobotBuilder sizin için uygun bir Java veya C++ sınıf adı oluşturmak için kelimeleri birleştirecektir.

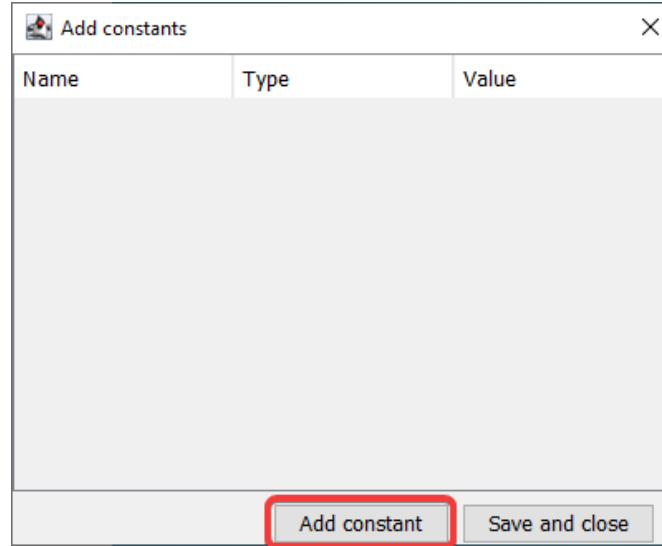
Sabitler Ekleme



Sabitler, kodunuzdaki sihirli sayıların miktarını azaltmak için çok kullanışlıdır. Alt sistemlerde, bir asansörün belirli yükseklikleri için sensör değerleri veya robotun hareket hızı gibi belirli değerleri takip etmek için kullanılabilirler.

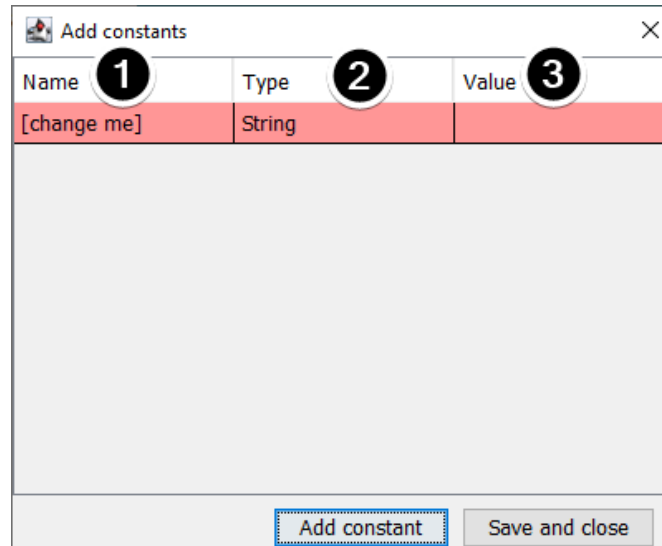
Varsayılan olarak, bir alt sistemde sabitler olmayacaktır. Bazılarını oluşturmak üzere bir ilişim kutusu açmak için “Constants-Sabitler” in yanındaki düğmeye basın.

Sabitler Oluřturma



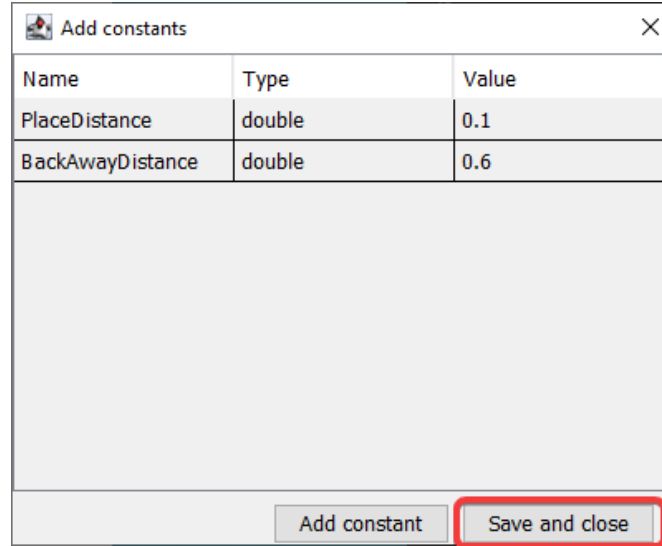
Constants-Sabitler tablosu ilk bařta boř olacaktır. Sabit eklemek iin “Add constant-Sabit ekle” ye basın.

Sabit Ekle



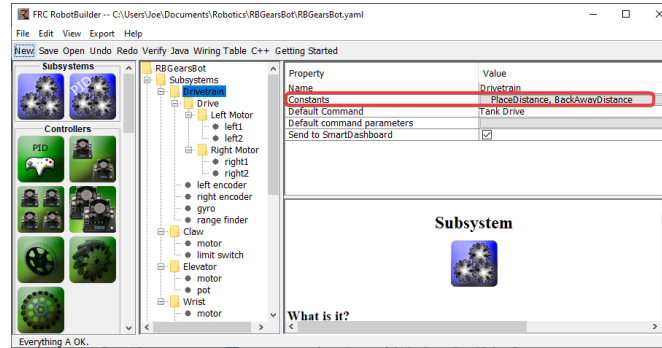
1. Sabitin adı. Bunu açıklayıcı bir řeye deęiřtirin. Bu aktarma organı rneęinde, bazı iyi sabitler “PlaceDistance” ve “BackAwayDistance” olabilir.
2. Sabitin tr. Bu byk olasılıkla bir double olacaktır, ancak řunlardan birini seebilirsiniz: String, double, int, long, boolean veya byte.
3. Sabitin deęeri.

Sabitleri Kaydetme



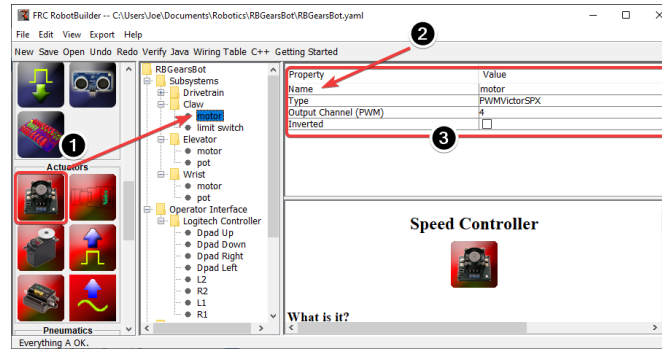
Sabitleri ekledikten ve değerlerini ayarladıktan sonra, sabitleri kaydetmek ve iletişim kutusunu kapatmak için sadece “Save and close-Kaydet ve kapat” a basın. Kaydetmek istemiyorsanız, pencerenin üst kısmındaki çıkış düğmesine basın.

Kaydettikten sonra



Sabitleri kaydettikten sonra, isimler alt sistem özelliklerinde “Constants -Sabitler” butonunda görünecektir.

Aktüatörleri / Sensörleri Alt Sisteme Sürükleme



Bir alt sisteme bileşen eklemenin üç adımı vardır:

1. Aktüatörleri veya sensörleri gerektiği şekilde paletten alt sisteme sürükleyin.
2. Aktüatöre veya sensöre anlamlı bir ad verin
3. Alt sistemdeki her öge için modül numaraları ve kanal numaraları gibi özellikleri düzenleyin.

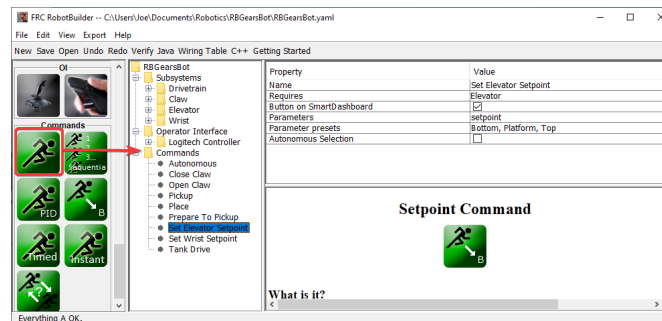
RobotBuilder, robot üzerindeki her modül için artan kanal numaralarını otomatik olarak kullanacaktır. Robotu henüz kablolamadıysanız, RobotBuilder'ın her sensör veya aktüatör için benzersiz kanal numaraları atamasına ve robotu üretici kablolama tablosuna göre kablolamasına izin verebilirsiniz.

Bu sadece RobotBuilder'da alt sistemi oluşturur ve daha sonra alt sistem için iskelet kodu oluşturur. Robotunuzu gerçekten çalıştırmasını sağlamak için lütfen bakınız : ref: "Bir Alt Sistem için Kod Yazma <docs/software/wpilib-tools/robotbuilder/writing-code/robotbuilder-writing-subsystem-code:Writing the Code for a Subsystem>".

21.1.6 Bir Command-Komut Oluşturmak

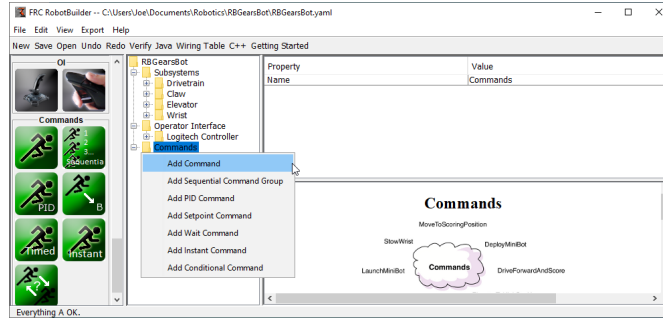
Komutlar, alt sistemleriniz için davranışlar veya eylemler sağlayan, oluşturduğunuz sınıflardır. Alt sistem sınıfı, asansörü hareket ettirmek için MoveElevator gibi alt sistemin çalışmasını veya asansörün PID ayar noktasını ayarlamak için ElevatorToSetPoint'i ayarlamalıdır. Komutlar alt sistem işlemini başlatır ve ne zaman bittiğini takip eder.

Komutu Commands Klasörüne sürükleyin



Basit komutlar paletten robot açıklamasına sürüklenebilir. Komut, Commands klasörü altında oluşturulacaktır.

Context menüsünü Kullanarak Komut Oluşturma



Robot açıklamasındaki Command klasöründe sağ tıklanan context menüsünü kullanarak da komutlar oluşturabilirsiniz.

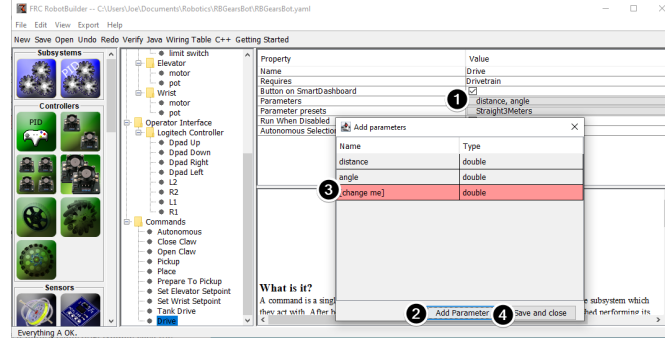
Komutu Yapılandırma

Property	Value
Name	Set Elevator Setpoint
Requires	Elevator
Button on SmartDashboard	<input checked="" type="checkbox"/>
Parameters	setpoint
Parameter presets	Bottom, Platform, Top
Autonomous Selection	<input type="checkbox"/>

1. Komuta, komutun ne yapacağını açıklayan anlamlı bir ad verin. Sözcükler arasında boşluklar olabilmesine rağmen komutlar kod içindeymiş gibi adlandırılmalıdır.
2. Bu komutun gerektirdiği alt sistemi ayarlayın. Bu komut zamanlandığında, o anda çalışan ve bu komutu da gerektiren herhangi bir komutu otomatik olarak durduracaktır. Pençeyi açma komutu o anda çalışıyorsa (pençe alt sistemini gerektiriyorsa) ve pençe kapama komutu zamanlanmışsa, açmayı hemen durdurur ve kapatmaya başlar.
3. RobotBuilder'a komut için SmartDashboard'da düğmeler oluşturması gerekıp gerekmediğini söyleyin. Her parametre ön ayarı için bir düğme oluşturulacaktır.
4. Bu komutun alacağı parametreleri ayarlayın. Parametrelili tek bir komut, parametre almayan iki veya daha fazla komutla aynı şeyi yapabilir. Örneğin, "Drive Forward", "Drive Backward" ve "Drive Distance" komutları, yön ve mesafe için değerler alan tek bir komutta birleştirilebilir.
5. Parametreler için ön ayarları ayarlayın. Bunlar, bir kumanda kolu düğmesine bağlamak veya bir alt sistem için varsayılan komutu ayarlamak gibi komutu kullanırken RobotBuilder'ın başka bir yerinde kullanılabilir.
6. *Run When Disabled-Devre Dışı Bırakıldığında Çalıştır*. Robot devre dışı bırakıldığında komutun çalışmasına izin verir. Ancak, devre dışıyken komut verilen hiçbir aktüatör harekete geçmeyecektir.
7. *Autonomous Selection-Otonom Seçim*. Komutun, otonom için seçilebilmesi için Gönderilebilir Seçici'ye eklenip eklenmeyeceği.

Setpoint komutları tek bir parametre ile gelir (double türünde 'setpoint'); ayar noktası komutları için parametreler eklenemez, düzenlenemez veya silinemez.

Parametreleri Ekleme ve Düzenleme

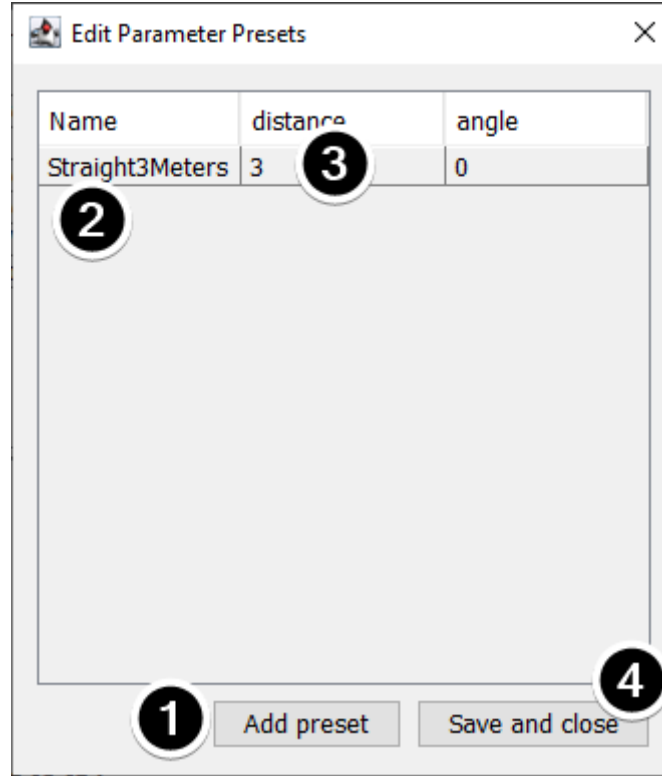


Parametre eklemek veya düzenlemek için:

1. Özellik tablosunun *Value* sütunundaki butona tıklayın
2. Bir parametre eklemek için *Add Parameter* düğmesine basın
3. Yeni eklenen bir parametre. Ad varsayılan olarak *[change me]* ve tür varsayılan olarak *Double* şeklindedir. Varsayılan ad geçersizdir, bu nedenle dışa aktarmadan önce değiştirmeniz gerekecektir. Adı değiştirmeye başlamak için: `guiabel:Name` hücrelerine çift tıklayın. Türü seçmek için *Type* hücrelerine çift tıklayın.
4. *Save and close* düğmesi tüm değişiklikleri kaydedecek ve pencereyi kapatacaktır.

Satırlar basitçe sürüklenerek yeniden sıralanabilir ve seçilip sil veya geri tuşuna basılarak silinebilir.

Parametre Ön Ayarları Ekleme ve Düzenleme

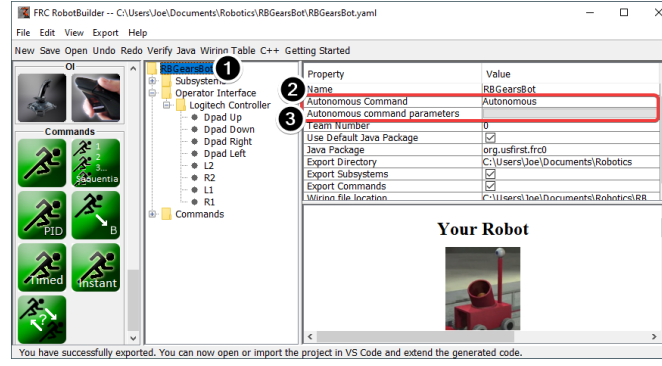


1. Yeni bir ön ayar eklemek için *Add parameter set* ye tıklayın.
2. Ön ayarın adını açıklayıcı bir adla değiştirin. Bu örnekteki ön ayarlar, gripper alt sistemini açmak ve kapatmak içindir.
3. Ön ayar için parametre (ler) in değerini değiştirin. Ya bir değer yazabilir (ör. "3.14") veya komutun gerektirdiği alt sistemde tanımlanan sabitler arasından seçim yapabilirsiniz. Sabitin türünün parametre ile aynı türde olması gerektiğine dikkat edin - örneğin, double tür bir parametreye bir int-türü sabit atayamazsınız.
4. Değişiklikleri kaydetmek ve iletişim kutusundan çıkmak için *Save and close* a tıklayın; kaydetmeden çıkmak için, pencerenin üst çubuğundaki çıkış düğmesine basın.

21.1.7 Otonom Komutları Ayarlama

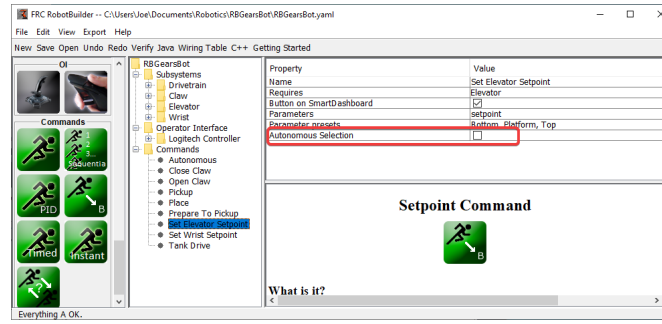
Bir komut, basitçe robotun gerçekleştirdiği bir veya daha fazla eylem (davranışlar) olduğundan, bir robotun otonom çalışmasını bir komut olarak tanımlamak mantıklıdır. Tek bir komut olabileceği gibi, bir komut grubu (birlikte gerçekleşen bir komut grubu) olması daha olasıdır.

RobotBuilder generates code for a *Sendable Chooser* which allows the autonomous command to run to be chosen from the dashboard.



Gösterge tablosunda başka bir komut seçilmezse çalışan varsayılan otonom komutu belirlemek için:

- Robot programı açıklamasında robotu seçin
- Otonom komut alanını, robot otonom moda getirildiğinde çalışması gereken komutla doldurun. Bu bir açılır alandır ve size tanımlanmış herhangi bir komutu seçme seçeneği verecektir.
- Varsa, komutun alacağı parametreleri ayarlayın.



Gönderilebilir Seçici'ye seçenek olarak eklenecek komutları seçmek için, Otonom Seçim onay kutusunu seçin.

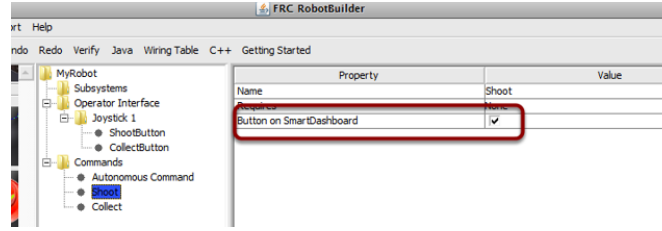
Robot otonom moda alındığında, seçilen Otonom komut programlanacaktır.

21.1.8 Bir Komutu Test Etmek İçin Shuffleboard Kullanma

Komutlar, Shuffleboard/SmartDashboard'a komutu tetiklemek için bir düğme eklenerek kolayca test edilir. Bu şekilde, robot programının geri kalanıyla hiçbir entegrasyon gerekli değildir ve komutlar bağımsız olarak kolayca test edilebilir. Bu, komutları doğrulamanın en kolay yoludur, çünkü programınızdaki tek bir kod satırı ile, Shuffleboard'da komutu çalıştıracak bir düğme oluşturulabilir. Bu düğmeler, gelecekte alt sistemleri ve komut işlemlerini doğrulamak için yerinde bırakılabilir.

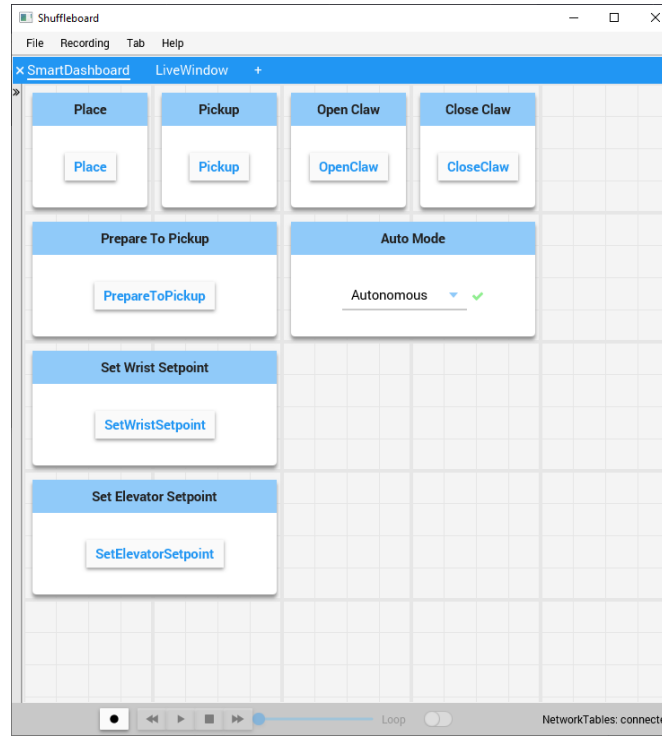
Bu, her biri komut yazan birden çok programcılığı barındırma avantajına sahiptir. Kod ana robot projesinde kontrol edildiğinde, komutlar ayrı ayrı test edilebilir.

Shuffleboard'da Düğme Oluşturma



Düğme, komutun bir örneğini robot programından kontrol paneline koyarak SmartDashboard'da oluşturulur. Bu o kadar yaygın bir işlemdir ki, RobotBuilder'a bir onay kutusu olarak eklenmiştir. Komutlarınızı yazarken, kutunun işaretli olduğundan ve düğmelerin sizin için otomatik olarak oluşturulduğundan emin olun.

Düğmelerin Çalıştırılması



Düğmeler otomatik olarak oluşturulacak ve kontrol paneli ekranında görünecektir. Shuffleboard'daki düğmeleri yeniden düzenleyebilirsiniz. Bu örnekte, her biri test için ilişkili bir düğme içeren birkaç komut vardır. Komutlar düğmesine basmak komutu çalıştıracaktır. Bir kez basıldığında, tekrar basmak komutu keserek Interrupted() yönteminin çağrılmasına neden olur.

Manuel Olarak Komut Ekleme

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));
```

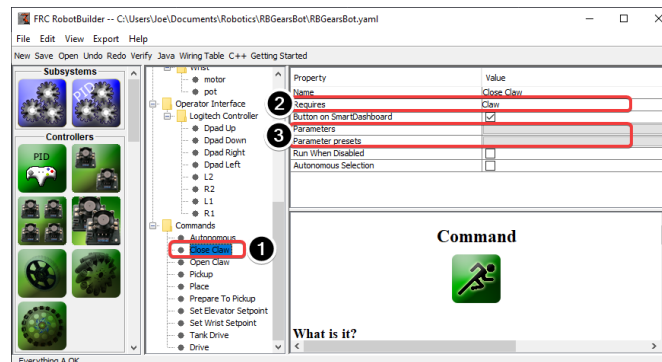
Komutlar, kodu kendiniz yazarak Shuffleboard'a manuel olarak eklenebilir. Bu, Shuffleboard'daki düğmeyle ilişkilendirilmesi gereken adla birlikte komutun örneklerini PutData yöntemine ileterek yapılır. Bu örnekler, düğmeye her basıldığında planlanır. Sonuç, RobotBuilder'ın oluşturduğu kodla tamamen aynıdır, ancak RobotBuilder'daki onay kutusuna tıklamak tüm kodu elle yazmaktan çok daha kolaydır.

21.1.9 Operatör Arayüzünü Bir Komuta Bağlama

Komutlar, robotunuzun davranışlarını ele alır. Komut, kaldırma ve asansör gibi bazı çalışma modlarına bir alt sistem başlatır ve bir ayar noktasına veya zaman aşımına ulaşana kadar çalışmaya devam eder. Komut daha sonra alt sistemin bitmesini beklemeyi işler. Bu şekilde komutlar, daha karmaşık davranışlar geliştirmek için sırayla çalışabilir.

RobotBuilder ayrıca, operatör arayüzündeki bir düğmeye her basıldığında çalıştırılacak bir komut programlamak için kod oluşturacaktır. Ayrıca, belirli bir tetikleme koşulu gerçekleştiğinde bir komutu çalıştırmak için kod yazabilirsiniz.

Düğmeye Basılarak Komut Çalıştırın

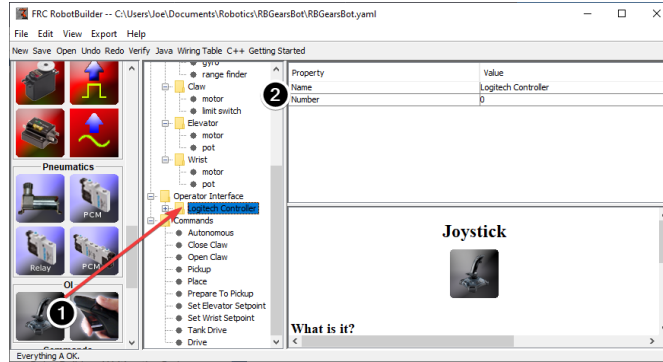


Bu örnekte, bir logitech gamepad (düğme 6) üzerinde dpad sağ yön düğmesine her basıldığında çalışacak şekilde "Close Claw-Pençeyi Kapat" komutunu programlamak istiyoruz.

1. Çalıştırma komutu "Close Claw-Pençeyi Kapat" olarak adlandırılır ve işlevi robotun pençesini kapatmaktır

2. Komutun Claw subsystem gerektirdiğine dikkat edin. Bu, aynı zamanda pençe kullanılan başka bir işlem olsa bile bu komutun çalışmaya başlamasını sağlayacaktır. Bu durumda önceki komut kesintiye uğrayacaktır.
3. Parametreler, bir komutun birden çok şey yapmasını mümkün kılar; hazır ayarlar, komuta ilettiğiniz değerleri tanımlamanıza ve bunları yeniden kullanmanıza izin verir

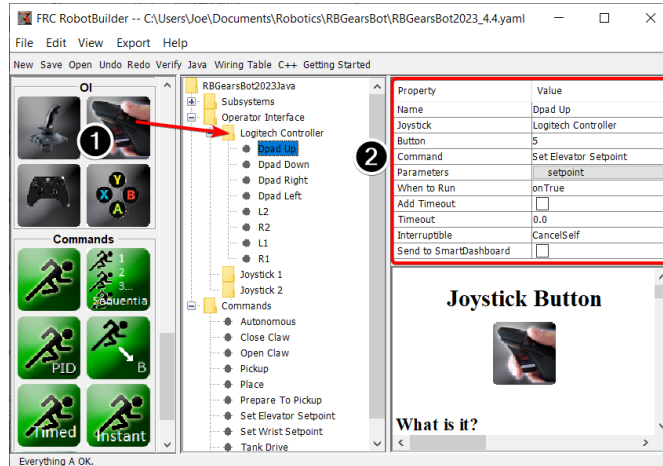
Joystick’i Robot Programına Ekleme



Joystick’i robot programına ekleyin

1. Kumanda çubuğunu robot programındaki Operatör Arayüzü klasörüne sürükleyin
2. Joystick’i, joystick kullanımını yansıtacak şekilde adlandırın ve USB port numarasını ayarlayın

Bir Düğmeyi “Move Elevator-Kaldırıcı hareketi” Komutuna Bağlama



Programa basılması gereken butonu ekleyin

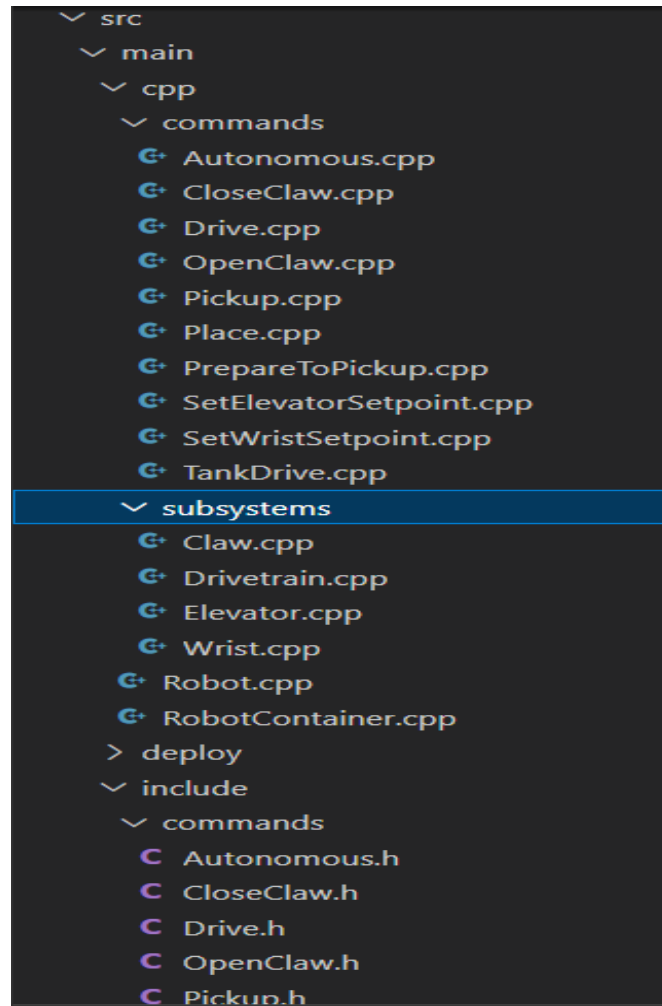
1. Joystick düğmesini, joystick’in altına gelecek şekilde Joystick’e (Logitech Controller) sürükleyin.
2. Set the properties for the button: the button number, the command to run when the button is pressed, parameters the command takes, and the *When to run* property to *onTrue* to indicate that the command should run whenever the joystick button is pressed.

Not: Joystick düğmeleri, bir Joystick'e (under-altına) sürüklenmelidir. Düğmeleri eklemekten önce Operatör Arayüzü klasöründe bir kumanda çubuğunuz olmalıdır.

21.1.10 RobotBuilder tarafından Oluşturulan Kod

RobotBuilder Tarafından Oluşturulan Bir Projenin Düzeni





RobotBuilder tarafından oluşturulan bir proje, bir paket (Java'da) veya Komutlar için bir klasör (C++'da) ve Alt Sistemler için bir diğerinden oluşur. Her komut veya alt sistem nesnesi bu kaplar altında saklanır. Projenin en üst seviyesinde robot ana programını bulacaksınız (RobotContainer.java/C++).

Komut Tabanlı bir robotun organizasyonu hakkında daha fazla bilgi için, bakınız [Komut Tabanlı Bir Robot Projesi Yapılandırma](#)

Otomatik Oluşturulan Kod

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);
```

Robot açıklaması değiştirildiğinde ve kod yeniden dışa aktarıldığında, RobotBuilder dosyada yaptığımız değişiklikleri koruyacak ve böylece kodunuzu koruyacak şekilde tasarlanmıştır. Bu, RobotBuilder'ı tam bir yaşam döngüsü aracı yapar. RobotBuilder hangi kodun değiştirilmesinin uygun olduğunu bilmek için, bazı özel yorumlarla sınırlandırılarak potansiyel olarak yeniden yazılması gereken bölümler oluşturur. Bu yorumlar yukarıdaki örnekte gösterilmektedir. Bu yorum bloklarına herhangi bir kod eklemeyin, projenin RobotBuilder'dan bir sonraki dışa aktarılışında yeniden yazılacaktır.

Bu bloklardan birinin içindeki kodun değiştirilmesi gerekiyorsa, yorumlar kaldırılabilir, ancak bu daha sonra başka güncellemelerin yapılmasını önleyecektir. Yukarıdaki örnekte, //BEGIN ve //END açıklamaları kaldırıldıysa, daha sonra RobotBuilder'a başka bir gerekli alt sistem eklendi, sonraki dışa aktarmada oluşturulmayacaktır.

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.

Ana Robot Programı**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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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  */
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,
42          ↪ and put our
43          // autonomous chooser on the dashboard.
44          m_robotContainer = RobotContainer.getInstance();
45          HAL.report(tResourceType.kResourceType_Framework, tInstances.kFramework_
46          ↪RobotBuilder);
47      }
48
49      /**
50       * This function is called every robot packet, no matter the mode. Use this for
51       ↪ items like
52       * diagnostics that you want ran during disabled, autonomous, teleoperated and
53       ↪ test.
54       *
55       * <p>This runs after the mode specific periodic functions, but before
56       * LiveWindow and SmartDashboard integrated updating.
57       */
58      @Override
59      public void robotPeriodic() {
60          // Runs the Scheduler. This is responsible for polling buttons, adding newly-
61          ↪ scheduled
62          // commands, running already-scheduled commands, removing finished or
63          ↪ interrupted commands,
64          // and running subsystem periodic() methods. This must be called from the
65          ↪ robot's periodic
66          // block in order for anything in the Command-based framework to work.
67          CommandScheduler.getInstance().run(); // (2)
68      }
69
70      /**
71       * This function is called once each time the robot enters Disabled mode.
72       */
73      @Override
74      public void disabledInit() {
75      }
76
77      @Override
78      public void disabledPeriodic() {

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

73     }
74
75     /**
76     * This autonomous runs the autonomous command selected by your {@link RobotContainer} class.
77     */
78     @Override
79     public void autonomousInit() {
80         m_autonomousCommand = m_robotContainer.getAutonomousCommand(); // (3)
81
82         // schedule the autonomous command (example)
83         if (m_autonomousCommand != null) {
84             m_autonomousCommand.schedule();
85         }
86     }
87
88     /**
89     * This function is called periodically during autonomous.
90     */
91     @Override
92     public void autonomousPeriodic() {
93     }
94
95     @Override
96     public void teleopInit() {
97         // This makes sure that the autonomous stops running when
98         // teleop starts running. If you want the autonomous to
99         // continue until interrupted by another command, remove
100        // this line or comment it out.
101        if (m_autonomousCommand != null) {
102            m_autonomousCommand.cancel();
103        }
104    }
105
106    /**
107    * This function is called periodically during operator control.
108    */
109    @Override
110    public void teleopPeriodic() {
111    }
112
113    @Override
114    public void testInit() {
115        // Cancels all running commands at the start of test mode.
116        CommandScheduler.getInstance().cancelAll();
117    }
118
119    /**
120    * This function is called periodically during test mode.
121    */
122    @Override
123    public void testPeriodic() {
124    }
125
126 }

```


C++ (Header)

```
11 // ROBOTBUILDER TYPE: Robot.
12 #pragma once
13
14 #include <frc/TimedRobot.h>
15 #include <frc2/command/Command.h>
16
17 #include "RobotContainer.h"
18
19 class Robot : public frc::TimedRobot { // {1}
20 public:
21     void RobotInit() override;
22     void RobotPeriodic() override;
23     void DisabledInit() override;
24     void DisabledPeriodic() override;
25     void AutonomousInit() override;
26     void AutonomousPeriodic() override;
27     void TeleopInit() override;
28     void TeleopPeriodic() override;
29     void TestPeriodic() override;
30
31 private:
32     // Have it null by default so that if testing teleop it
33     // doesn't have undefined behavior and potentially crash.
34     frc2::Command* m_autonomousCommand = nullptr;
35
36     RobotContainer* m_container = RobotContainer::GetInstance();
37 };
```

C++ (Source)

```
11 // ROBOTBUILDER TYPE: Robot.
12
13 #include "Robot.h"
14
15 #include <frc/smartdashboard/SmartDashboard.h>
16 #include <frc2/command/CommandScheduler.h>
17
18 void Robot::RobotInit() {}
19
20 /**
21  * This function is called every robot packet, no matter the mode. Use
22  * this for items like diagnostics that you want to run during disabled,
23  * autonomous, teleoperated and test.
24  *
25  * <p> This runs after the mode specific periodic functions, but before
26  * LiveWindow and SmartDashboard integrated updating.
27  */
28 void Robot::RobotPeriodic() { frc2::CommandScheduler::GetInstance().Run(); } // (2)
29
30 /**
31  * This function is called once each time the robot enters Disabled mode. You
32  * can use it to reset any subsystem information you want to clear when the
33  * robot is disabled.
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

34  */
35  void Robot::DisabledInit() {}
36
37  void Robot::DisabledPeriodic() {}
38
39  /**
40   * This autonomous runs the autonomous command selected by your {@link
41   * RobotContainer} class.
42   */
43  void Robot::AutonomousInit() {
44      m_autonomousCommand = m_container->GetAutonomousCommand(); // {3}
45
46      if (m_autonomousCommand != nullptr) {
47          m_autonomousCommand->Schedule();
48      }
49  }
50
51  void Robot::AutonomousPeriodic() {}
52
53  void Robot::TeleopInit() {
54      // This makes sure that the autonomous stops running when
55      // teleop starts running. If you want the autonomous to
56      // continue until interrupted by another command, remove
57      // this line or comment it out.
58      if (m_autonomousCommand != nullptr) {
59          m_autonomousCommand->Cancel();
60          m_autonomousCommand = nullptr;
61      }
62  }
63
64  /**
65   * This function is called periodically during operator control.
66   */
67  void Robot::TeleopPeriodic() {}
68
69  /**
70   * This function is called periodically during test mode.
71   */
72  void Robot::TestPeriodic() {}
73
74  #ifndef RUNNING_FRC_TESTS
75  int main() { return frc::StartRobot<Robot>(); }
76  #endif

```

Bu, RobotBuilder tarafından oluşturulan ana programdır. Bu programın birkaç bölümü vardır (vurgulanan bölümler):

1. Bu sınıf TimedRobot'u geliştirir. TimedRobot her 20 ms'de bir autonomousPeriodic() ve teleopPeriodic() yöntemlerinizi çağıracaktır.
2. Her 20 ms'de bir çağrılan robotPeriodic yönteminde, bir planlama geçişi yapın.
3. Sağlanan otonom komut, autonomousInit() yönteminde otonom başlangıcında planlanır ve teleopInit() içindeki otonom dönemin sonunda iptal edilir.

RobotContainer

Java

```

11 // ROBOTBUILDER TYPE: RobotContainer.
12
13 package frc.robot;
14
15 import frc.robot.commands.*;
16 import frc.robot.subsystems.*;
17 import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
18 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
19 import edu.wpi.first.wpilibj2.command.Command.InterruptionBehavior;
20
21 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =IMPORTS
22 import edu.wpi.first.wpilibj2.command.Command;
23 import edu.wpi.first.wpilibj2.command.InstantCommand;
24 import edu.wpi.first.wpilibj.Joystick;
25 import edu.wpi.first.wpilibj2.command.button.JoystickButton;
26 import frc.robot.subsystems.*;
27
28 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =IMPORTS
29
30
31 /**
32  * This class is where the bulk of the robot should be declared. Since Command-based
33  * is a
34  * "declarative" paradigm, very little robot logic should actually be handled in the
35  * {@link Robot}
36  * periodic methods (other than the scheduler calls). Instead, the structure of the
37  * robot
38  * (including subsystems, commands, and button mappings) should be declared here.
39  */
40 public class RobotContainer {
41
42     private static RobotContainer m_robotContainer = new RobotContainer();
43
44     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
45     // The robot's subsystems
46     public final Wrist m_wrist = new Wrist(); // (1)
47     public final Elevator m_elevator = new Elevator();
48     public final Claw m_claw = new Claw();
49     public final Drivetrain m_drivetrain = new Drivetrain();
50
51     // Joysticks
52     private final Joystick joystick2 = new Joystick(2); // (3)
53     private final Joystick joystick1 = new Joystick(1);
54     private final Joystick logitechController = new Joystick(0);
55
56     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
57
58     // A chooser for autonomous commands
59     SendableChooser<Command> m_chooser = new SendableChooser<>();
60
61     /**
62      * The container for the robot. Contains subsystems, OI devices, and commands.

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

106     SmartDashboard.putData("Auto Mode", m_chooser);
107 }
108
109 public static RobotContainer getInstance() {
110     return m_robotContainer;
111 }
112
113 /**
114  * Use this method to define your button->command mappings. Buttons can be created
115  * by instantiating a {@link GenericHID} or one of its subclasses ({@link
116  * edu.wpi.first.wpilibj.Joystick} or {@link XboxController}), and then passing it
117  * to a
118  * {@link edu.wpi.first.wpilibj2.command.button.JoystickButton}.
119  */
120 private void configureButtonBindings() {
121     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =BUTTONS
122     // Create some buttons
123     final JoystickButton r1 = new JoystickButton(logitechController, 12); // (4)
124     r1.onTrue(new Autonomous().withInterruptBehavior(InterruptionBehavior.kCancelSelf));
125
126     final JoystickButton l1 = new JoystickButton(logitechController, 11);
127     l1.onTrue(new Place().withInterruptBehavior(InterruptionBehavior.kCancelSelf));
128
129     final JoystickButton r2 = new JoystickButton(logitechController, 10);
130     r2.onTrue(new Pickup().withInterruptBehavior(InterruptionBehavior.kCancelSelf));
131
132     final JoystickButton l2 = new JoystickButton(logitechController, 9);
133     l2.onTrue(new PrepareToPickup().withInterruptBehavior(InterruptionBehavior.
134     kCancelSelf));
135
136     final JoystickButton dpadLeft = new JoystickButton(logitechController, 8);
137     dpadLeft.onTrue(new OpenClaw(1.0, m_claw).withInterruptBehavior(InterruptionBehavior.
138     kCancelSelf));
139
140     final JoystickButton dpadRight = new JoystickButton(logitechController, 6);
141     dpadRight.onTrue(new CloseClaw( m_claw ).withInterruptBehavior(InterruptionBehavior.
142     kCancelSelf));
143
144     final JoystickButton dpadDown = new JoystickButton(logitechController, 7);
145     dpadDown.onTrue(new SetElevatorSetpoint(0, m_elevator).
146     withInterruptBehavior(InterruptionBehavior.kCancelSelf));
147
148     final JoystickButton dpadUp = new JoystickButton(logitechController, 5);
149     dpadUp.onTrue(new SetElevatorSetpoint(0.3, m_elevator).
150     withInterruptBehavior(InterruptionBehavior.kCancelSelf));
151
152     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =FUNCTIONS
153 }
154
155 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =FUNCTIONS
156 public Joystick getLogitechController() {
157     return logitechController;
158 }

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

155
156 public Joystick getJoystick1() {
157     return joystick1;
158 }
159
160 public Joystick getJoystick2() {
161     return joystick2;
162 }
163
164
165 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =FUNCTIONS
166
167 /**
168  * Use this to pass the autonomous command to the main {@link Robot} class.
169  *
170  * @return the command to run in autonomous
171  */
172 public Command getAutonomousCommand() {
173     // The selected command will be run in autonomous
174     return m_chooser.getSelected();
175 }
176
177
178 }

```

C++ (Header)

```

11 // ROBOTBUILDER TYPE: RobotContainer.
12
13 #pragma once
14
15 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
16 #include <frc/smartdashboard/SendableChooser.h>
17 #include <frc2/command/Command.h>
18
19 #include "subsystems/Claw.h"
20 #include "subsystems/Drivetrain.h"
21 #include "subsystems/Elevator.h"
22 #include "subsystems/Wrist.h"
23
24 #include "commands/Autonomous.h"
25 #include "commands/CloseClaw.h"
26 #include "commands/Drive.h"
27 #include "commands/OpenClaw.h"
28 #include "commands/Pickup.h"
29 #include "commands/Place.h"
30 #include "commands/PrepareToPickup.h"
31 #include "commands/SetElevatorSetpoint.h"
32 #include "commands/SetWristSetpoint.h"
33 #include "commands/TankDrive.h"
34 #include <frc/Joystick.h>
35 #include <frc2/command/button/JoystickButton.h>
36
37 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

38
39 class RobotContainer {
40
41 public:
42
43     frc2::Command* GetAutonomousCommand();
44     static RobotContainer* GetInstance();
45
46     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =PROTOTYPES
47     // The robot's subsystems
48     Drivetrain m_drivetrain; // (1)
49     Claw m_claw;
50     Elevator m_elevator;
51     Wrist m_wrist;
52
53
54     frc::Joystick* getJoystick2();
55     frc::Joystick* getJoystick1();
56     frc::Joystick* getLogitechController();
57
58     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =PROTOTYPES
59
60 private:
61
62     RobotContainer();
63
64     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
65     // Joysticks
66     frc::Joystick m_logitechController{0}; // (3)
67     frc::Joystick m_joystick1{1};
68     frc::Joystick m_joystick2{2};
69
70     frc::SendableChooser<frc2::Command*> m_chooser;
71
72     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
73
74     Autonomous m_autonomousCommand;
75     static RobotContainer* m_robotContainer;
76
77     void ConfigureButtonBindings();
78 };

```

C++ (Source)

```

11 // ROBOTBUILDER TYPE: RobotContainer.
12
13 #include "RobotContainer.h"
14 #include <frc2/command/ParallelRaceGroup.h>
15 #include <frc/smartdashboard/SmartDashboard.h>
16
17
18
19 RobotContainer* RobotContainer::m_robotContainer = NULL;
20

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

21 RobotContainer::RobotContainer() : m_autonomousCommand(
22     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTOR
23 ){
24
25
26
27     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTOR
28
29     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =SMARTDASHBOARD
30     // Smartdashboard Subsystems
31     frc::SmartDashboard::PutData(&m_drivetrain);
32     frc::SmartDashboard::PutData(&m_claw);
33     frc::SmartDashboard::PutData(&m_elevator);
34     frc::SmartDashboard::PutData(&m_wrist);
35
36
37     // SmartDashboard Buttons
38     frc::SmartDashboard::PutData("Drive: Straight3Meters", new Drive(3, 0, &m_
39     drivetrain)); // (6)
40     frc::SmartDashboard::PutData("Drive: Place", new Drive(Drivetrain::PlaceDistance,
41     Drivetrain::BackAwayDistance, &m_drivetrain));
42     frc::SmartDashboard::PutData("Set Wrist Setpoint: Horizontal", new
43     SetWristSetpoint(0, &m_wrist));
44     frc::SmartDashboard::PutData("Set Wrist Setpoint: Raise Wrist", new
45     SetWristSetpoint(-45, &m_wrist));
46     frc::SmartDashboard::PutData("Set Elevator Setpoint: Bottom", new
47     SetElevatorSetpoint(0, &m_elevator));
48     frc::SmartDashboard::PutData("Set Elevator Setpoint: Platform", new
49     SetElevatorSetpoint(0.2, &m_elevator));
50     frc::SmartDashboard::PutData("Set Elevator Setpoint: Top", new
51     SetElevatorSetpoint(0.3, &m_elevator));
52     frc::SmartDashboard::PutData("Prepare To Pickup", new PrepareToPickup());
53     frc::SmartDashboard::PutData("Place", new Place());
54     frc::SmartDashboard::PutData("Pickup", new Pickup());
55     frc::SmartDashboard::PutData("Open Claw: OpenTime", new OpenClaw(1.0_s, &m_claw));
56     frc::SmartDashboard::PutData("Close Claw", new CloseClaw(&m_claw));
57
58     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =SMARTDASHBOARD
59
60     ConfigureButtonBindings();
61
62     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DEFAULT-COMMANDS
63     m_drivetrain.SetDefaultCommand(TankDrive([this] {return getJoystick1()->GetY();},
64     [this] {return getJoystick2()->GetY();}, &m_drivetrain)); // (5)
65
66     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DEFAULT-COMMANDS
67
68     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =AUTONOMOUS
69
70     m_chooser.AddOption("Set Elevator Setpoint: Bottom", new SetElevatorSetpoint(0, &
71     m_elevator));
72     m_chooser.AddOption("Set Elevator Setpoint: Platform", new SetElevatorSetpoint(0.
73     2, &m_elevator));
74     m_chooser.AddOption("Set Elevator Setpoint: Top", new SetElevatorSetpoint(0.3, &m_
75     elevator));

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

66     m_chooser.SetDefaultOption("Autonomous", new Autonomous()); // (2)
67
68     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =AUTONOMOUS
69
70     frc2::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).
94         ↪WithInterruptBehavior(frc2::Command::InterruptBehavior::kCancelSelf));
95
96     m_dpadDown.OnTrue(SetElevatorSetpoint(0, &m_elevator).
97         ↪WithInterruptBehavior(frc2::Command::InterruptBehavior::kCancelSelf));
98
99     m_dpadRight.OnTrue(CloseClaw( &m_claw ).
100         ↪WithInterruptBehavior(frc2::Command::InterruptBehavior::kCancelSelf));
101
102     m_dpadLeft.OnTrue(OpenClaw(1.0_s, &m_claw).
103         ↪WithInterruptBehavior(frc2::Command::InterruptBehavior::kCancelSelf));
104
105     m_l2.OnTrue(PrepareToPickup().
106         ↪WithInterruptBehavior(frc2::Command::InterruptBehavior::kCancelSelf));
107
108     m_r2.OnTrue(Pickup().
109         ↪WithInterruptBehavior(frc2::Command::InterruptBehavior::kCancelSelf));
110
111     m_l1.OnTrue(Place().
112         ↪WithInterruptBehavior(frc2::Command::InterruptBehavior::kCancelSelf));
113
114     m_r1.OnTrue(Autonomous().
115         ↪WithInterruptBehavior(frc2::Command::InterruptBehavior::kCancelSelf));
116
117     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =BUTTONS
118 }
119
120 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =FUNCTIONS

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

114 frc::Joystick* RobotContainer::getLogitechController() {
115     return &m_logitechController;
116 }
117 frc::Joystick* RobotContainer::getJoystick1() {
118     return &m_joystick1;
119 }
120 frc::Joystick* RobotContainer::getJoystick2() {
121     return &m_joystick2;
122 }
123
124 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =FUNCTIONS
125
126
127 frc2::Command* RobotContainer::GetAutonomousCommand() {
128     // The selected command will be run in autonomous
129     return m_chooser.GetSelected();
130 }
131

```

Bu, alt sistemlerin ve operatör arayüzünün tanımlandığı RobotBuilder tarafından oluşturulan RobotContainer'dır. Bu programın birkaç bölümü vardır (vurgulanan bölümler):

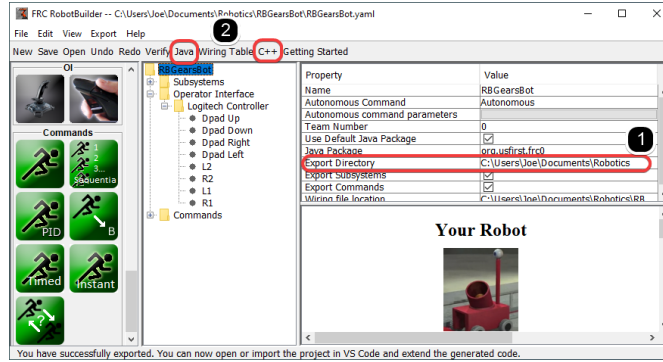
1. Alt sistemlerin her biri burada açıklanmıştır. Bunları gerektiren herhangi bir komuta parametre olarak geçirilebilirler.
2. RobotBuilder robot özelliklerinde sağlanan otonom bir komut varsa, bu, kontrol panelinde seçilmek üzere Gönderilebilir Seçici'ye eklenir.
3. Tüm operatör arayüzü bileşenlerinin kodu burada oluşturulur.
4. Ek olarak, OI düğmelerini çalıştırılması gereken komutlara bağlayan kod da burada oluşturulur.
5. Commands to be run on a subsystem when no other commands are running are defined here.
6. Bir gösterge panosu aracılığıyla çalıştırılacak komutlar burada tanımlanır.

21.2 RobotBuilder - Kodu Yazmak

21.2.1 Bir Proje için Kod Oluşturma

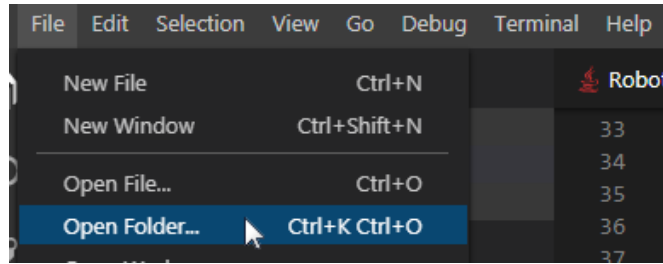
Robot framework'ünüzü RobotBuilder'da kurduktan sonra, kodu dışa aktarmanız ve Visual Studio Code'a yüklemeniz gerekir. Bu makale bunu yapma sürecini açıklamaktadır.

Proje için Kod Oluşturun



Dışa Aktarma Klasörünün istediğiniz yere (1) yönelttiğini doğrulayın ve ardından VS Code projesi oluşturmak veya kodu güncellemek için Java veya C++'a (2) tıklayın.

Projeyi Visual Studio Code'da Açın

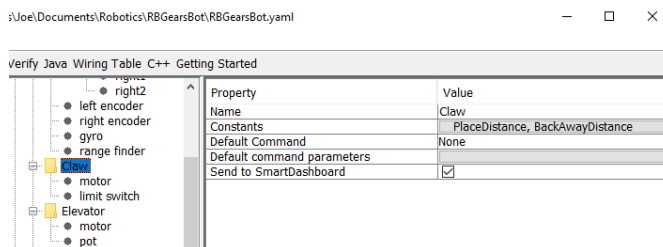


VS Kodunu açın ve **Dosya -> Klasörü Aç** seçeneğini seçin. Dışa Aktarma konumunuza gidin ve **Klasör Seç** seçeneğine tıklayın.

21.2.2 Subsystem-Alt Sistem için Kod Yazma

Gerçek bir çalışan alt sistem oluşturmak için kod eklemek çok basittir. Geri bildirim kullanmayan basit alt sistemler için son derece basit olduğu ortaya çıkıyor. Bu bölümde bir *Claw* alt sistemi örneğine bakacağız. *Claw* alt sistemi ayrıca bir nesnenin kavramada olup olmadığını belirlemek için bir limit anahtarına sahiptir.

Claw Subsystem-Pençe Alt Sisteminin RobotBuilder Temsili



Bir robot kolunun ucundaki pençe, tek bir VictorSPX Motor Kontrol Cihazı tarafından çalıştırılan bir alt sistemdir. Motorun yapmasını istediğimiz üç şey var, açmaya başlaması, kapanmaya

başlaması ve hareket etmeyi bırakması. Bu, alt sistemin sorumluluğundadır. Açma ve kapama zamanlaması, bu eğitimin ilerleyen bölümlerinde bir komutla ele alınacaktır. Pençenin bir nesneyi tutup tutmadığını anlamak için bir yöntem de tanımlayacağız.

Subsystem-Alt Sistem Yetenekleri Ekleme

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
25 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =IMPORTS
26
27
28 /**
29  *
30  */
31 public class Claw extends SubsystemBase {
32     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTANTS
33     public static final double PlaceDistance = 0.1;
34     public static final double BackAwayDistance = 0.6;
35
36     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTANTS
37
38     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
39     private PWMVictorSPX motor;
40     private DigitalInput limitswitch;
41
42     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
43
44     /**
45      *
46      */
47     public Claw() {
48         // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTORS
49         motor = new PWMVictorSPX(4);
50         addChild("motor",motor);
51         motor.setInverted(false);
52
53         limitswitch = new DigitalInput(4);
54         addChild("limit switch", limitswitch);
55
56
57
58         // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTORS

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

59     }
60
61     @Override
62     public void periodic() {
63         // This method will be called once per scheduler run
64     }
65
66     @Override
67     public void simulationPeriodic() {
68         // This method will be called once per scheduler run when in simulation
69     }
70
71     }
72
73     public void open() {
74         motor.set(1.0);
75     }
76
77     public void close() {
78         motor.set(-1.0);
79     }
80
81     public void stop() {
82         motor.set(0.0);
83     }
84
85     public boolean isGripping() {
86         return limitswitch.get();
87     }
88
89 }

```

C++

```

11 // ROBOTBUILDER TYPE: Subsystem.
12
13 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
14 #include "subsystems/Claw.h"
15 #include <frc/smartdashboard/SmartDashboard.h>
16
17 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
18
19 Claw::Claw(){
20     SetName("Claw");
21     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
22     SetSubsystem("Claw");
23
24     AddChild("limit switch", &m_limitswitch);
25
26
27     AddChild("motor", &m_motor);
28     m_motor.SetInverted(false);
29
30     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

31 }
32
33 void Claw::Periodic() {
34     // Put code here to be run every loop
35 }
36
37
38 void Claw::SimulationPeriodic() {
39     // This method will be called once per scheduler run when in simulation
40 }
41
42
43 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CMDPIDGETTERS
44 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CMDPIDGETTERS
45
46
47
48 void Claw::Open() {
49     m_motor.Set(1.0);
50 }
51
52 void Claw::Close() {
53     m_motor.Set(-1.0);
54 }
55
56 void Claw::Stop() {
57     m_motor.Set(0.0);
58 }
59
60 bool Claw::IsGripping() {
61     return m_limitswitch.Get();
62 }

```

claw.java veya claw.cpp ye pençe hareketini açacak, kapatacak ve durduracak ve pençe sınır anahtarını alacak yöntemler ekleyin. Bunlar pençeyi gerçekten çalıştıran komutlar tarafından kullanılacak.

Not: Bu belgedeki değişiklikleri görmeyi kolaylaştırmak için yorumlar bu dosyadan kaldırılmıştır.

motor ve limitswitch adlı üye değişkeninin RobotBuilder tarafından yaratıldığına dikkat edin, böylece alt sistem genelinde kullanılabilir. Sürüklenen palet öğelerinizin her biri, RobotBuilder'da verilen ada sahip bir üye değişkenine sahip olacaktır.

Üstbilgi Dosyasına Yöntem Bildirimleri Ekleme (Yalnızca C++)

C++

```

11 // ROBOTBUILDER TYPE: Subsystem.
12 #pragma once
13
14 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
15 #include <frc2/command/SubsystemBase.h>
16 #include <frc/DigitalInput.h>
17 #include <frc/motorcontrol/PWMVictorSPX.h>
18
19 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
20
21 /**
22  *
23  *
24  * @author ExampleAuthor
25  */
26 class Claw: public frc2::SubsystemBase {
27 private:
28     // It's desirable that everything possible is private except
29     // for methods that implement subsystem capabilities
30     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
31     frc::DigitalInput m_limitswitch{4};
32     frc::PWMVictorSPX m_motor{4};
33
34     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
35 public:
36     Claw();
37
38     void Periodic() override;
39     void SimulationPeriodic() override;
40     void Open();
41     void Close();
42     void Stop();
43     bool IsGripping();
44     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CMDPIDGETTERS
45
46     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CMDPIDGETTERS
47     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTANTS
48     static constexpr const double PlaceDistance = 0.1;
49     static constexpr const double BackAwayDistance = 0.6;
50
51     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTANTS
52
53 };
54

```

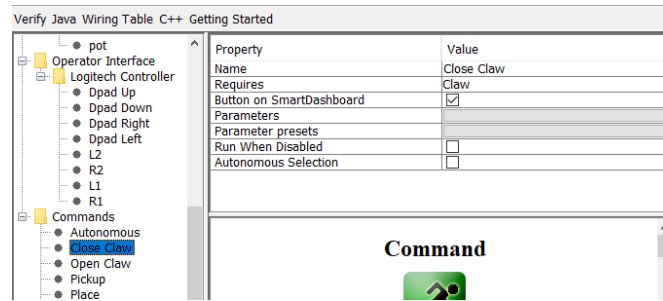
Yöntemleri sınıf uygulama dosyasına Claw.cpp eklemeye ek olarak, yöntemlerin bildirimlerinin Claw.h başlık dosyasına eklenmesi gerekir. Eklenmesi gereken beyanlar burada gösterilmektedir.

Açma ve kapama davranışını pençe alt sistemine eklemek için şunları yapmanız gerekir <../introduction/robotbuilder-creating-command> komutlarını tanımlayın.

21.2.3 Bir Komutun Kodunu Yazmak

Subsystem sınıfları, robotunuzdaki mekanizmaları hareket ettirir, ancak doğru zamanda durmasını ve daha karmaşık işlemlerle sıralanmasını sağlamak için Komutlar yazarsınız. Daha önce *writing the code for a subsystem* pençe açılmasını, kapanmasını başlatmak veya hareket etmeyi durdurmak için bir robot üzerindeki *Claw* alt sisteminin kodunu geliştirdik. Şimdi pençe motorunu doğru zamanda çalıştıracak bir komutun kodunu yazarak pençenin açılıp kapanmasını sağlayacağız. Pençe örneğimiz, motoru 1 saniye süreyle çalıştırdığımız veya kapatmak için limit anahtarı açılıncaya kadar çalıştırdığımız çok basit bir mekanizmadır.

RobotBuilder'da Pençe Komutunu Kapat



Bu, RobotBuilder'daki *CloseClaw* komutunun tanımıdır. *Claw* alt sistemini gerektirdiğine dikkat edin. Bu, sonraki adımda açıklanmaktadır.

CloseClaw Sınıfı oluşturuldu

JAVA

```

11 // ROBOTBUILDER TYPE: Command.
12
13 package frc.robot.commands;
14 import edu.wpi.first.wpilibj2.command.CommandBase;
15
16 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =IMPORTS
17 import frc.robot.subsystems.Claw;
18
19 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =IMPORTS
20
21 /**
22  *
23  */
24 public class CloseClaw extends CommandBase {
25
26     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLE_DECLARATIONS
27     private final Claw m_claw;
28
29     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLE_DECLARATIONS
30
31     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTORS
32
33

```

(sonraki sayfaya devam)


```
34 public CloseClaw(Claw subsystem) {
35
36
37 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTORS
38 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLE_SETTING
39
40 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLE_SETTING
41 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =REQUIRES
42
43     m_claw = subsystem;
44     addRequirements(m_claw);
45
46 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =REQUIRES
47 }
48
49 // Called when the command is initially scheduled.
50 @Override
51 public void initialize() {
52     m_claw.close(); // (1)
53 }
54
55 // Called every time the scheduler runs while the command is scheduled.
56 @Override
57 public void execute() {
58 }
59
60 // Called once the command ends or is interrupted.
61 @Override
62 public void end(boolean interrupted) {
63     m_claw.stop(); // (3)
64 }
65
66 // Returns true when the command should end.
67 @Override
68 public boolean isFinished() {
69     return m_claw.isGripping(); // (2)
70 }
71
72 @Override
73 public boolean runsWhenDisabled() {
74     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DISABLED
75     return false;
76
77 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DISABLED
78 }
79 }
```

C++

```

11 // ROBOTBUILDER TYPE: Command.
12
13 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTOR
14
15 #include "commands/CloseClaw.h"
16
17 CloseClaw::CloseClaw(Claw* m_claw)
18 :m_claw(m_claw){
19
20     // Use AddRequirements() here to declare subsystem dependencies
21     // eg. AddRequirements(m_Subsystem);
22     SetName("CloseClaw");
23     AddRequirements({m_claw});
24
25 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTOR
26
27 }
28
29 // Called just before this Command runs the first time
30 void CloseClaw::Initialize() {
31     m_claw->Close(); // (1)
32 }
33
34 // Called repeatedly when this Command is scheduled to run
35 void CloseClaw::Execute() {
36
37 }
38
39 // Make this return true when this Command no longer needs to run execute()
40 bool CloseClaw::IsFinished() {
41     return m_claw->IsGripping(); // (2)
42 }
43
44 // Called once after isFinished returns true
45 void CloseClaw::End(bool interrupted) {
46     m_claw->Stop(); // (3)
47 }
48
49 bool CloseClaw::RunsWhenDisabled() const {
50     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DISABLED
51     return false;
52
53     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DISABLED
54 }

```

RobotBuilder, *CloseClaw* komutu için sınıf dosyalarını oluşturacaktır. Komut, pençe davranışını, yani zaman içindeki işlemi temsil eder. Bu çok basit pençe mekanizmasını çalıştırmak için motorun yakın yönde çalışması gerekir. *Claw* alt sistemi, motoru doğru yönde çalıştırma ve durdurma yöntemlerine sahiptir. Komutların sorumluluğu, motoru doğru zamanda çalıştırmaktır. Kutularda gösterilen kod satırları, bu davranışı eklemek için eklenir.

1. *CloseClaw* Initialize yönteminde *Claw* alt sistemine eklenen *Close()* yöntemini çağırarak pençe motorunu kapanma yönünde hareket ettirin.
2. This command is finished when the limit switch in the *Claw* subsystem is tripped.
3. *End()* yöntemi, komut bittiğinde çağrılır ve temizlenecek bir yerdir. Bu durumda motor,

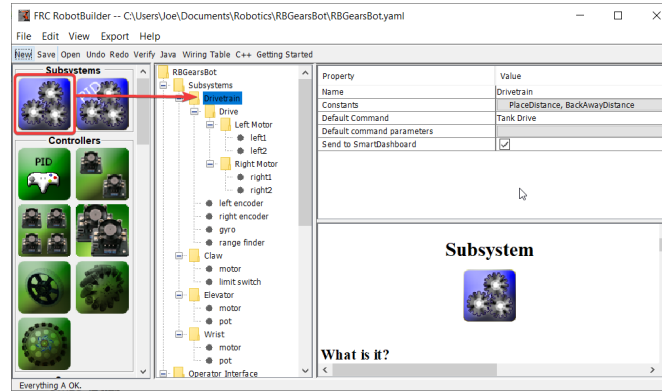
süre dolduğu için durdurulur.

21.2.4 Robotu Tank Sürüş methodu ve Joysticklerle Sürmek

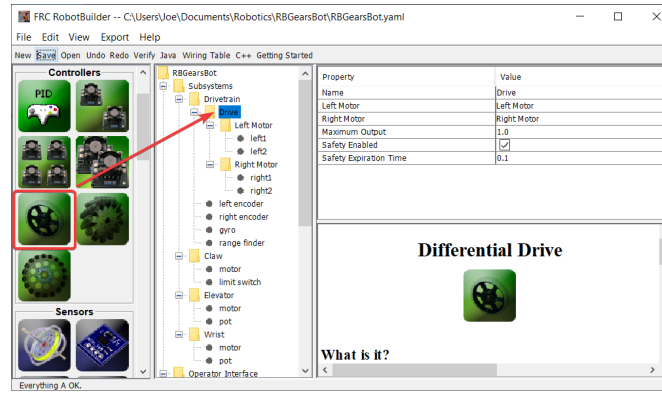
Yaygın bir kullanım durumu, bir alt sistemin parçası olan bazı aktüatörleri çalıştırması gereken bir kumanda çubuğuna sahip olmaktır. Sorun, joystick'in RobotContainer sınıfında oluşturulması ve kontrol edilecek motorların alt sistemde olmasıdır. Buradaki fikir, programlandığında, joystick'ten girdi okuyan ve motorları çalıştıran alt sistemde oluşturulan bir yöntemi çağıran bir komut oluşturmaktır.

Bu örnekte, bir çift kumanda kolu kullanılarak tank sürücüsünde çalıştırılan bir sürücü tabanı alt sistemi gösterilmektedir.

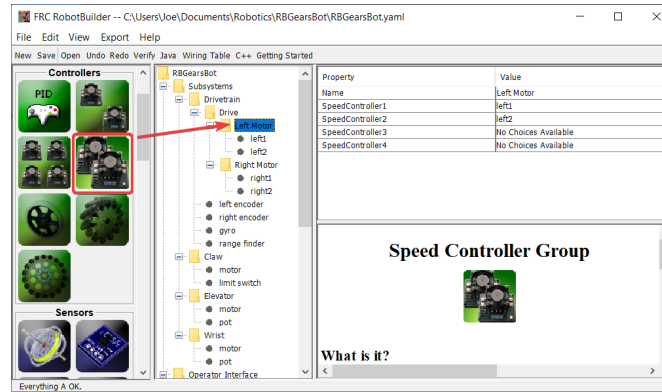
Drive Train Subsystem Oluşturun



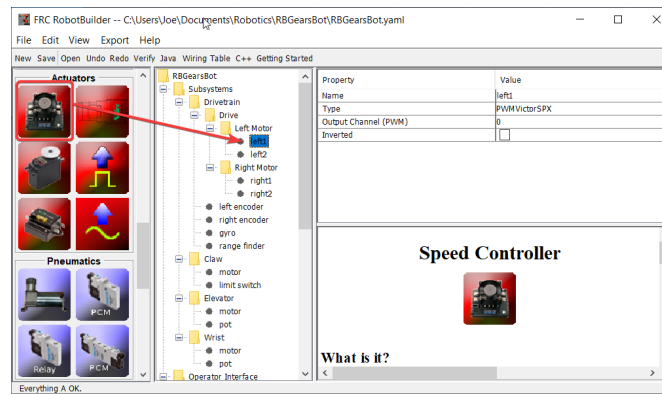
Drive Train adlı bir alt sistem oluşturun. Sorumluluğu, robot tabanı için sürüşü idare etmek olacaktır.



Aktarma Organının içinde iki motorlu sürücü için bir Diferansiyel Sürücü nesnesi oluşturun. Diferansiyel Sürücü sınıfının bir parçası olarak bir sol motor ve sağ motor vardır.

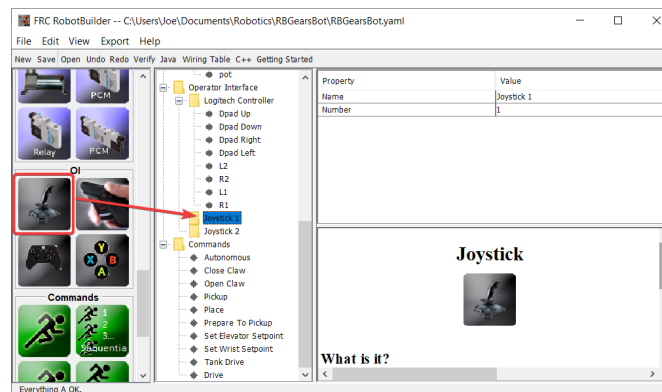


Since we want to use more than two motors to drive the robot, inside the Differential Drive, create two Motor Controller Groups. These will group multiple motor controllers so they can be used with Differential Drive.



Finally, create two Motor Controllers in each Motor Controller Group.

Joystickleri Operatör Arayüzüne Ekleyin



Operatör Arayüzüne iki kumanda çubuğu ekleyin, biri sol çubuk ve diğeri sağ çubuktur. İki kumanda kolundaki y eksenini, robotları sola ve sağa sürmek için kullanılır.

Not: Sonraki adıma geçmeden önce programınızı C ++ veya Java'ya aktardığınızdan emin olun.

Motorları Alt Sisteme Yazmak İçin Bir Yöntem Oluşturun

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.AnalogGyro;
22 import edu.wpi.first.wpilibj.AnalogInput;
23 import edu.wpi.first.wpilibj.CounterBase.EncodingType;
24 import edu.wpi.first.wpilibj.Encoder;
25 import edu.wpi.first.wpilibj.drive.DifferentialDrive;
26 import edu.wpi.first.wpilibj.motorcontrol.MotorController;
27 import edu.wpi.first.wpilibj.motorcontrol.MotorControllerGroup;
28 import edu.wpi.first.wpilibj.motorcontrol.PWMVictorSPX;
29
30 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =IMPORTS
31
32
33 /**
34  *
35  */
36 public class Drivetrain extends SubsystemBase {
37     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTANTS
38     public static final double PlaceDistance = 0.1;
39     public static final double BackAwayDistance = 0.6;
40
41     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTANTS
42
43     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
44     private PWMVictorSPX left1;
45     private PWMVictorSPX left2;
46     private MotorControllerGroup leftMotor;
47     private PWMVictorSPX right1;
48     private PWMVictorSPX right2;
49     private MotorControllerGroup rightMotor;
50     private DifferentialDrive drive;
51     private Encoder leftencoder;
52     private Encoder rightencoder;
53     private AnalogGyro gyro;
54     private AnalogInput rangefinder;
55
56     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
57
58     /**
59      *
60      */
61     public Drivetrain() {
62         // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTORS
63         left1 = new PWMVictorSPX(0);

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

64  addChild("left1",left1);
65  left1.setInverted(false);
66
67  left2 = new PWMVictorSPX(1);
68  addChild("left2",left2);
69  left2.setInverted(false);
70
71  leftMotor = new MotorControllerGroup(left1, left2 );
72  addChild("Left Motor",leftMotor);
73
74
75  right1 = new PWMVictorSPX(5);
76  addChild("right1",right1);
77  right1.setInverted(false);
78
79  right2 = new PWMVictorSPX(6);
80  addChild("right2",right2);
81  right2.setInverted(false);
82
83  rightMotor = new MotorControllerGroup(right1, right2 );
84  addChild("Right Motor",rightMotor);
85
86
87  drive = new DifferentialDrive(leftMotor, rightMotor);
88  addChild("Drive",drive);
89  drive.setSafetyEnabled(true);
90  drive.setExpiration(0.1);
91  drive.setMaxOutput(1.0);
92
93
94  leftencoder = new Encoder(0, 1, false, EncodingType.k4X);
95  addChild("left encoder",leftencoder);
96  leftencoder.setDistancePerPulse(1.0);
97
98  rightencoder = new Encoder(2, 3, false, EncodingType.k4X);
99  addChild("right encoder",rightencoder);
100 rightencoder.setDistancePerPulse(1.0);
101
102 gyro = new AnalogGyro(0);
103 addChild("gyro",gyro);
104 gyro.setSensitivity(0.007);
105
106 rangefinder = new AnalogInput(1);
107 addChild("range finder", rangefinder);
108
109
110
111 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTORS
112 }
113
114 @Override
115 public void periodic() {
116     // This method will be called once per scheduler run
117
118 }
119

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

120  @Override
121  public void simulationPeriodic() {
122      // This method will be called once per scheduler run when in simulation
123
124  }
125
126  // Put methods for controlling this subsystem
127  // here. Call these from Commands.
128
129  public void drive(double left, double right) {
130      drive.tankDrive(left, right);
131  }
132  }

```

C++ (Header)

```

11  // ROBOTBUILDER TYPE: Subsystem.
12  #pragma once
13
14  // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
15  #include <frc2/command/SubsystemBase.h>
16  #include <frc/AnalogGyro.h>
17  #include <frc/AnalogInput.h>
18  #include <frc/Encoder.h>
19  #include <frc/drive/DifferentialDrive.h>
20  #include <frc/motorcontrol/MotorControllerGroup.h>
21  #include <frc/motorcontrol/PWMVictorSPX.h>
22
23  // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
24
25  /**
26   *
27   *
28   * @author ExampleAuthor
29   */
30  class Drivetrain: public frc2::SubsystemBase {
31  private:
32      // It's desirable that everything possible is private except
33      // for methods that implement subsystem capabilities
34      // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
35  frc::AnalogInput m_rangefinder{1};
36  frc::AnalogGyro m_gyro{0};
37  frc::Encoder m_rightencoder{2, 3, false, frc::Encoder::k4X};
38  frc::Encoder m_leftencoder{0, 1, false, frc::Encoder::k4X};
39  frc::DifferentialDrive m_drive{m_leftMotor, m_rightMotor};
40  frc::MotorControllerGroup m_rightMotor{m_right1, m_right2 };
41  frc::PWMVictorSPX m_right2{6};
42  frc::PWMVictorSPX m_right1{5};
43  frc::MotorControllerGroup m_leftMotor{m_left1, m_left2 };
44  frc::PWMVictorSPX m_left2{1};
45  frc::PWMVictorSPX m_left1{0};
46
47      // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
48  public:

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

49 Drivetrain();
50
51 void Periodic() override;
52 void SimulationPeriodic() override;
53 void Drive(double left, double right);
54 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CMDPIDGETTERS
55
56 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CMDPIDGETTERS
57 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTANTS
58 static constexpr const double PlaceDistance = 0.1;
59 static constexpr const double BackAwayDistance = 0.6;
60
61 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTANTS
62
63
64 };

```

C++ (Source)

```

11 // ROBOTBUILDER TYPE: Subsystem.
12
13 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
14 #include "subsystems/Drivetrain.h"
15 #include <frc/smartdashboard/SmartDashboard.h>
16
17 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
18
19 Drivetrain::Drivetrain(){
20     SetName("Drivetrain");
21     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DECLARATIONS
22     SetSubsystem("Drivetrain");
23
24     AddChild("range finder", &m_rangefinder);
25
26
27     AddChild("gyro", &m_gyro);
28     m_gyro.SetSensitivity(0.007);
29
30     AddChild("right encoder", &m_rightencoder);
31     m_rightencoder.SetDistancePerPulse(1.0);
32
33     AddChild("left encoder", &m_leftencoder);
34     m_leftencoder.SetDistancePerPulse(1.0);
35
36     AddChild("Drive", &m_drive);
37     m_drive.SetSafetyEnabled(true);
38     m_drive.SetExpiration(0.1_s);
39     m_drive.SetMaxOutput(1.0);
40
41
42     AddChild("Right Motor", &m_rightMotor);
43
44
45     AddChild("right2", &m_right2);

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

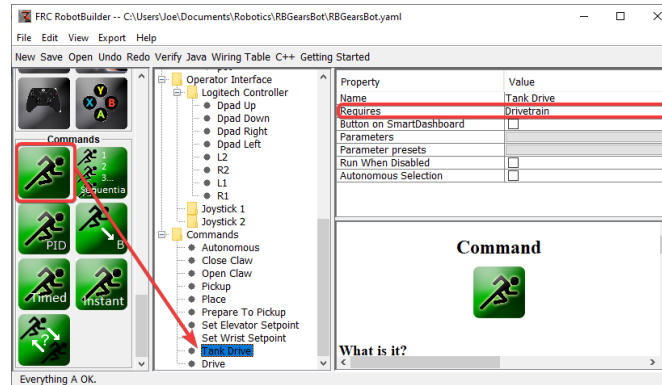
46 m_right2.SetInverted(false);
47
48 AddChild("right1", &m_right1);
49 m_right1.SetInverted(false);
50
51 AddChild("Left Motor", &m_leftMotor);
52
53
54 AddChild("left2", &m_left2);
55 m_left2.SetInverted(false);
56
57 AddChild("left1", &m_left1);
58 m_left1.SetInverted(false);
59
60 // END AUTOGENERATED 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 }

```

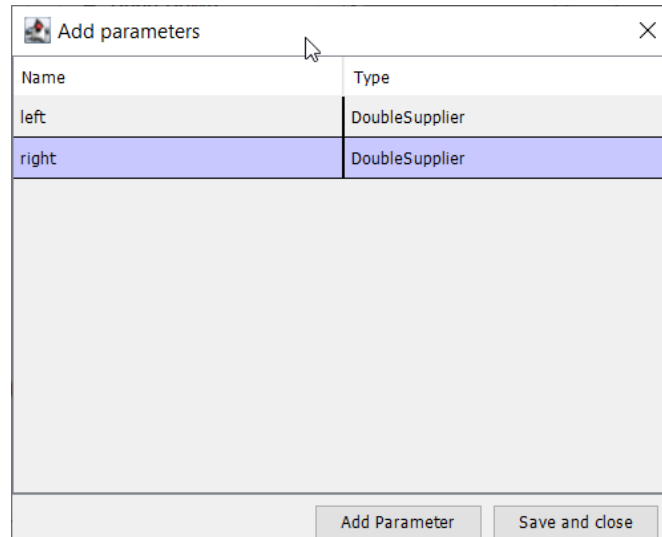
Create a method that takes the joystick inputs, in this case the left and right driver joystick. The values are passed to the DifferentialDrive object that in turn does tank steering using the joystick values. Also create a method called stop() that stops the robot from driving, this might come in handy later.

Not: Anlaşılır olması için bu örnekte bazı RobotBuilder çıktıları kaldırılmıştır

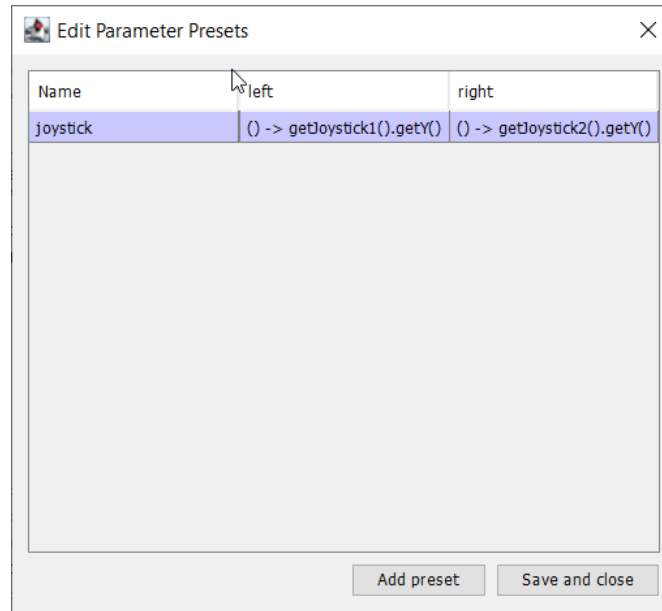
Kumanda Kolu Değerlerini Okuyun ve Subsystem Methodlarını Çağırın



Bu durumda Tank Drive adlı bir komut oluşturun. Amacı, joystick değerlerini okumak ve Drive Base alt sistemine göndermek olacaktır. Bu komutun Drive Train alt sistemini Gerektirdiğine dikkat edin. Bu, Drive Train'i başka bir şey kullanmaya çalıştığında çalışmasının durmasına neden olacaktır.



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();}`

Not: Sonraki adıma geçmeden önce programınızı C ++ veya Java'ya aktardığınızdan emin olun.

Sürüş yapmak için Kodu Ekleyin

java

```
11 // ROBOTBUILDER TYPE: Command.
12
13 package frc.robot.commands;
14 import edu.wpi.first.wpilibj.Joystick;
15 import edu.wpi.first.wpilibj2.command.CommandBase;
16 import frc.robot.RobotContainer;
17 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =IMPORTS
18 import frc.robot.subsystems.Drivetrain;
19
20 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =IMPORTS
21
22 /**
23  *
24  */
25 public class TankDrive extends CommandBase {
26
27     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLE_DECLARATIONS
28     private final Drivetrain m_drivetrain;
29
30     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLE_DECLARATIONS
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

31
32 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTORS
33
34
35 public TankDrive(Drivetrain subsystem) {
36
37
38 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTORS
39 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLE_SETTING
40
41 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLE_SETTING
42 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =REQUIRES
43
44     m_drivetrain = subsystem;
45     addRequirements(m_drivetrain);
46
47 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =REQUIRES
48 }
49
50 // Called when the command is initially scheduled.
51 @Override
52 public void initialize() {
53 }
54
55 // Called every time the scheduler runs while the command is scheduled.
56 @Override
57 public void execute() {
58     m_drivetrain.drive(m_left.getAsDouble(), m_right.getAsDouble());
59 }
60
61 // Called once the command ends or is interrupted.
62 @Override
63 public void end(boolean interrupted) {
64     m_drivetrain.drive(0.0, 0.0);
65 }
66
67 // Returns true when the command should end.
68 @Override
69 public boolean isFinished() {
70     return false;
71 }
72
73 @Override
74 public boolean runsWhenDisabled() {
75     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DISABLED
76     return false;
77
78 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DISABLED
79 }
80 }

```

C++ (Header)

```

11 // ROBOTBUILDER TYPE: Command.
12
13 #pragma once
14
15 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
16
17 #include <frc2/command/CommandHelper.h>
18 #include <frc2/command/CommandBase.h>
19
20 #include "subsystems/Drivetrain.h"
21
22 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =INCLUDES
23 #include "RobotContainer.h"
24 #include <frc/Joystick.h>
25
26 /**
27  *
28  *
29  * @author ExampleAuthor
30  */
31 class TankDrive: public frc2::CommandHelper<frc2::CommandBase, TankDrive> {
32 public:
33 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTOR
34 explicit TankDrive(Drivetrain* m_drivetrain);
35
36 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTOR
37
38 void Initialize() override;
39 void Execute() override;
40 bool IsFinished() override;
41 void End(bool interrupted) override;
42 bool RunsWhenDisabled() const override;
43
44 private:
45 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLES
46
47
48 Drivetrain* m_drivetrain;
49 frc::Joystick* m_leftJoystick;
50 frc::Joystick* m_rightJoystick;
51
52 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =VARIABLES
53 };
54

```

C++ (Source)

```

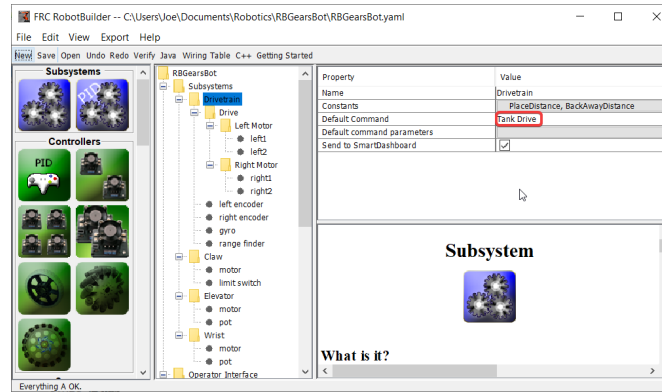
11 // ROBOTBUILDER TYPE: Command.
12
13 // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTOR
14
15 #include "commands/TankDrive.h"
16
17 TankDrive::TankDrive(Drivetrain* m_drivetrain)
18 :m_drivetrain(m_drivetrain){
19
20     // Use AddRequirements() here to declare subsystem dependencies
21     // eg. AddRequirements(m_Subsystem);
22     SetName("TankDrive");
23     AddRequirements({m_drivetrain});
24
25 // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =CONSTRUCTOR
26 }
27
28 // Called just before this Command runs the first time
29 void TankDrive::Initialize() {
30
31 }
32
33 // Called repeatedly when this Command is scheduled to run
34 void TankDrive::Execute() {
35     m_drivetrain->Drive(m_left(),m_right());
36 }
37
38 // Make this return true when this Command no longer needs to run execute()
39 bool TankDrive::IsFinished() {
40     return false;
41 }
42
43 // Called once after isFinished returns true
44 void TankDrive::End(bool interrupted) {
45     m_drivetrain->Drive(0,0);
46 }
47
48 bool TankDrive::RunsWhenDisabled() const {
49     // BEGIN AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DISABLED
50     return false;
51
52     // END AUTOGENERATED CODE, SOURCE =ROBOTBUILDER ID =DISABLED
53 }

```

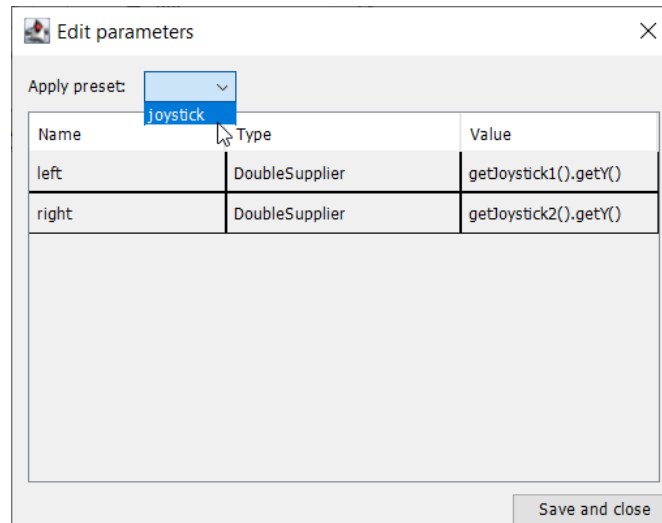
Add code to the execute method to do the actual driving. All that is needed is pass the for the left and right parameters to the Drive Train subsystem. The subsystem just uses them for the tank steering method on its DifferentialDrive object. And we get tank steering.

Ayrıca end() methodu da doldurduk, böylece bu komut kesildiğinde veya durdurulduğunda, motorlar bir güvenlik önlemi olarak durdurulacak.

Varsayılan Komutu Yap



Son adım, Tank Tahrik komutunun Aktarma Organları alt sistemi için “Default Command” olmasını sağlamaktır. Bu, Aktarma Sistemini başka bir komut kullanmadığında kumanda kol-larının kontrol altında olacağı anlamına gelir. Muhtemelen arzu edilen davranış budur. Oto-nom kod çalışırken, aynı zamanda aktarma organını da gerektirecek ve Tank Sürüş komutunu kesecektir. Otonom kod bittiğinde, DriveWithJoysticks komutu otomatik olarak yeniden baş-layacak (çünkü bu varsayılan komuttur) ve operatörler kontrolü geri alacak. Teleop otomatik sürüş yapan herhangi bir kod yazarsanız, bu komutlar da Tank Sürüşü komutunu kesecek ve tam kontrole sahip olacak şekilde DriveTrain’i “require-gerektirmelidir”.



The final step is to choose the joystick parameter preset previously set up.

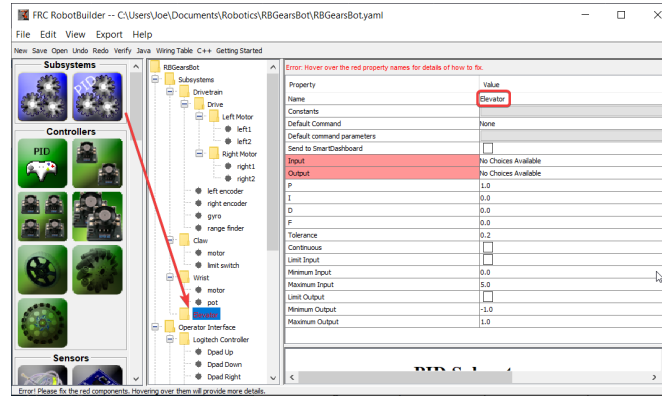
Not: Devam etmeden önce programınızı C++ veya Java’ya aktardığınızdan emin olun.

21.3 RobotBuilder - Gelişmiş

21.3.1 Aktüatörleri Kontrol Etmek İçin PIDSubsystem Kullanımı

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.

Bir PIDSubsystem oluşturun

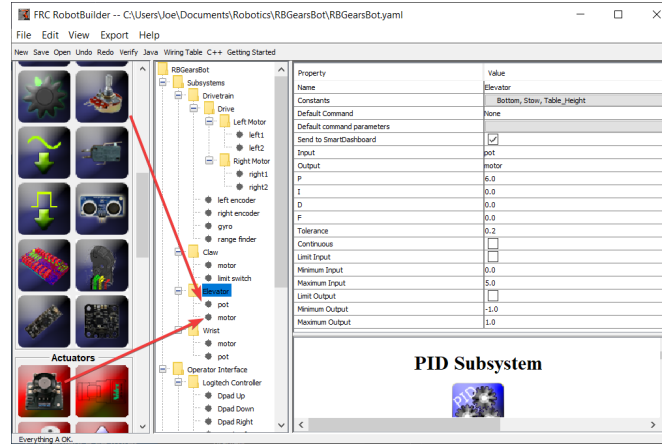


Bir mekanizmanın konumunu veya hızını kontrol etmek için geri bildirim kullanan bir alt sistem oluşturmak çok kolaydır.

1. Paletten bir PIDS alt sistemini robot açıklamasındaki Alt Sistemler klasörüne sürükleyin
2. PIDSubsystem alt sistem için daha anlamlı bir adla yeniden adlandırın, bu durumda Elevator

Robot açıklamasının bazı bölümlerinin kırmızıya döndüğüne dikkat edin. Bu, bu bileşenlerin (PIDSubsystem) tamamlanmadığını ve doldurulması gerektiğini gösterir. Eksik veya yanlış olan özellikler kırmızı ile gösterilir.

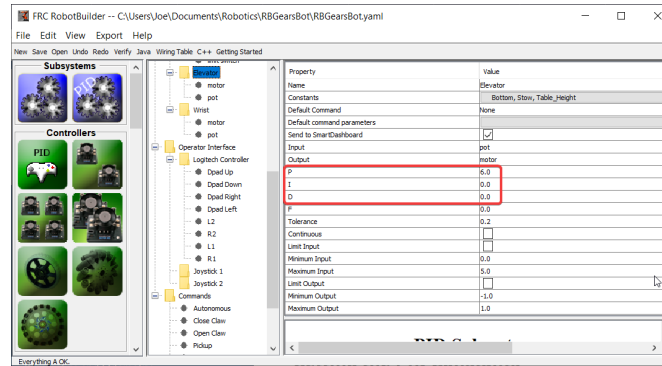
PIDSubsystem' e Sensör ve Aktüatör Ekleme



PIDSubsystem için eksik bileşenleri ekleyin

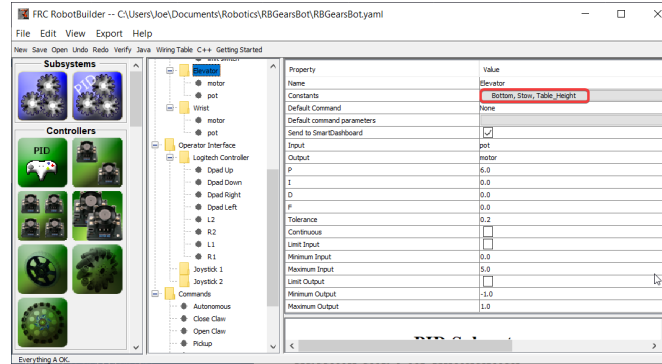
1. Aktüatörü (bir motor kontrolörü) belirli bir alt sisteme sürükleyin - bu durumda Elevator
2. Geri bildirim için kullanılacak sensörü alt sisteme sürükleyin; bu durumda sensör, asansör yüksekliği geri bildirimi verebilecek bir potansiyometredir.

PID Parametrelerini Doldurun

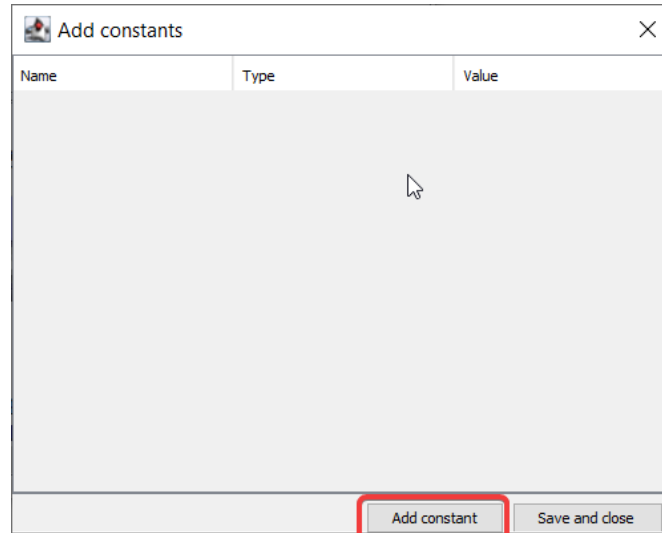


Bileşenin istenen hassasiyetini ve kararlılığını elde etmek için P, I ve D değerlerinin doldurulması gerekir. Asansörümüz söz konusu olduğunda, I ve D terimleri için orantılı bir sabit olan 6.0 ve 0 kullanırız.

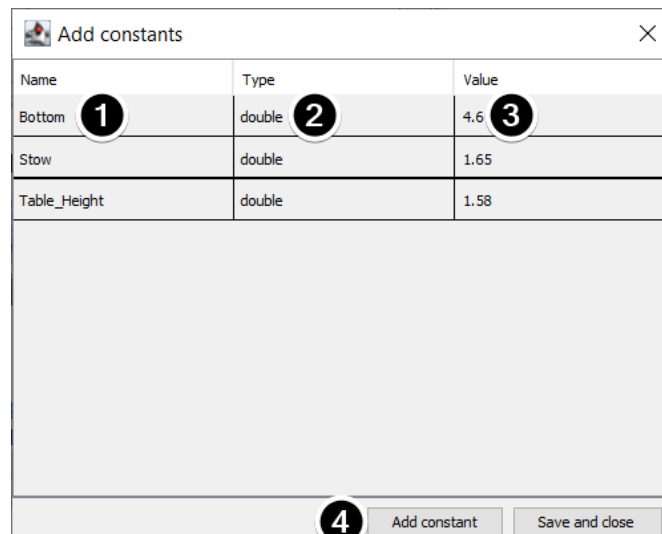
Setpoint Constants Oluşturun



Elevator ayar noktalarını yönetmeyi kolaylaştırmak için, ayar noktalarını yönetmek için sabitler oluşturacağız. Sabitler iletişim kutusunu açmak için sabitler kutusunu tıklayın.



add constant butonuna tıklayın



1. Sabit için bir ad girin, bu durumda: Bottom-Alt

2. Açılır menüden sabit için bir tür seçin, bu durumda: double-çift
3. Sabit-constant için bir değer seçin, bu durumda: 4.65
4. Sabitleri eklemeye devam etmek için *add constant* e tıklayın
5. After entering all constants, Click *Save and close*

21.3.2 PIDSubsystem için Kod Yazma

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.

Elevator PID alt sisteminin RobotBuilder’da oluşturulduğundan emin olun. Tamamen ayarlandıktan sonra, Dışa Aktar menüsünü veya Java / C ++ araç çubuğu menüsünü kullanarak proje için Java / C ++ kodu oluşturun.

RobotBuilder generates the PIDSubsystem methods such that no additional code is needed for basic operation.

PID Constants-Sabitlerini Ayarlama

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  
}
```

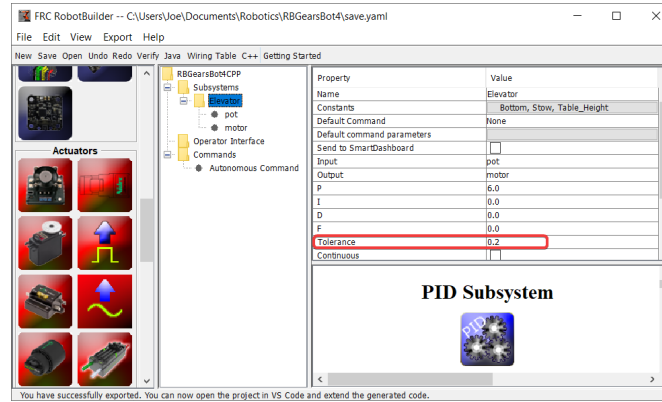
The useOutput method writes the calculated PID output directly to the motor.

Elevator PIDSubsystem oluşturmak için gereken tek şey bu.

21.3.3 Ayar Noktası Komutu

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.

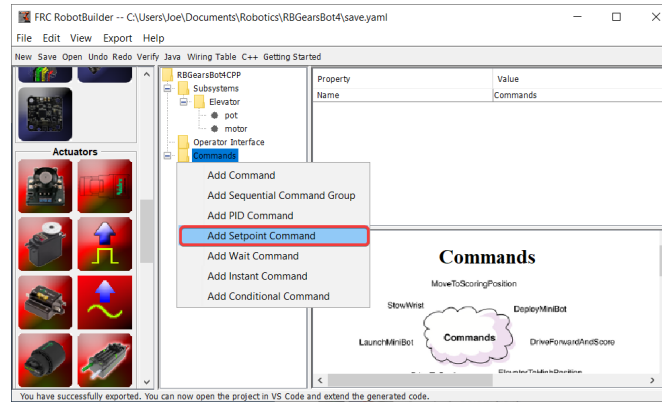
Bir PIDS alt sistemi ile başlayın



Bir robotta, açığı ölçen potansiyometreli bir bilek eklemi olduğunu varsayalım. Bilek eklemi hareket ettiren motoru ve açığı ölçen potansiyometreyi içeren bir PIDS alt sistemi First *create a PIDSubsystem* oluşturun. PIDS alt sisteminde tüm PID sabitleri doldurulmuş ve düzgün çalışıyor olmalıdır.

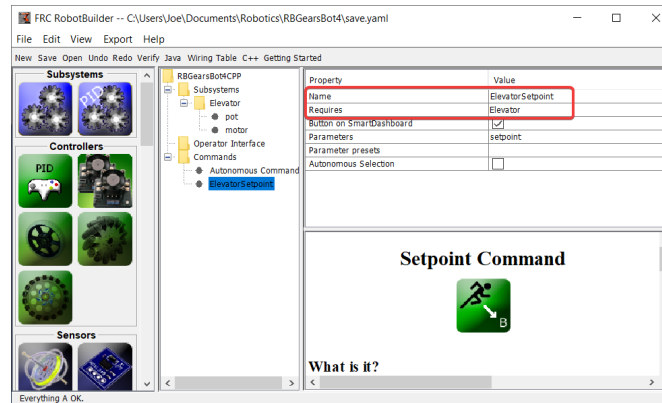
**** Tolerans **** parametresini ayarlamak önemlidir. Bu, mevcut değerin ayar noktasından ne kadar uzakta olabileceğini ve hedefte değerlendirilebileceğini kontrol eder. Bu, Ayar Noktası Komutunun bir sonraki komuta geçmek için kullandığı kriterdir.

Ayar Noktası Komutunu Oluşturma

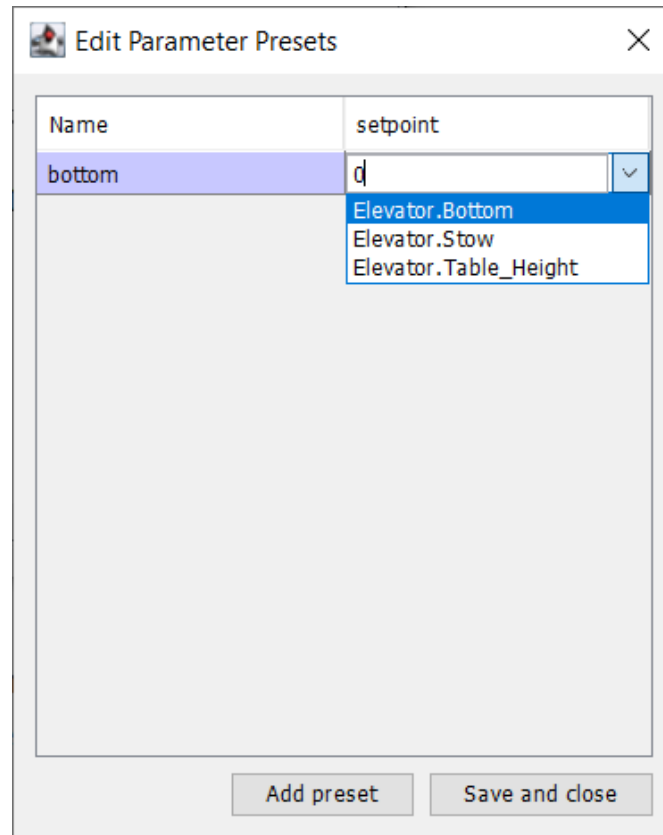


Paletteki Komutlar klasörüne sağ tıklayın ve “Ayar Noktası Ekle komutu” nu seçin.

Ayar Noktası Komut Parametreleri



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.

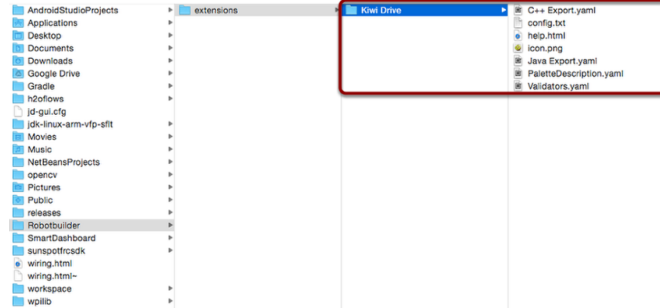
Bu komut her zamanlandığında, alt sistemi otomatik olarak belirtilen ayar noktasına yönlendirecektir. PIDS alt sisteminde belirtilen tolerans dahilinde ayar noktasına ulaşıldığında, komut sona erer ve bir sonraki komut başlar. PIDS alt sisteminde bir tolerans belirtmek önemlidir veya bu komut tolerans sağlanamadığı için asla bitmeyebilir.

Not: PID Kontrolü hakkında daha fazla bilgi için lütfen bakınız: <ref:Advanced Controls Introduction <docs/software/advanced-controls/introduction/index:Advanced Controls Introduction>>.

21.3.4 Özel Bileşenler Ekleme

RobotBuilder, motorlar, kontrolörler ve sensörler için yalnızca WPILib kullanan robot programları oluşturmak için çok iyi çalışıyor. Ancak özel sınıfları kullanan ekipler için, RobotBuilder'ın bu sınıflar için herhangi bir desteği yoktur, bu nedenle bunları RobotBuilder'da kullanmak için birkaç adım atılması gerekir.

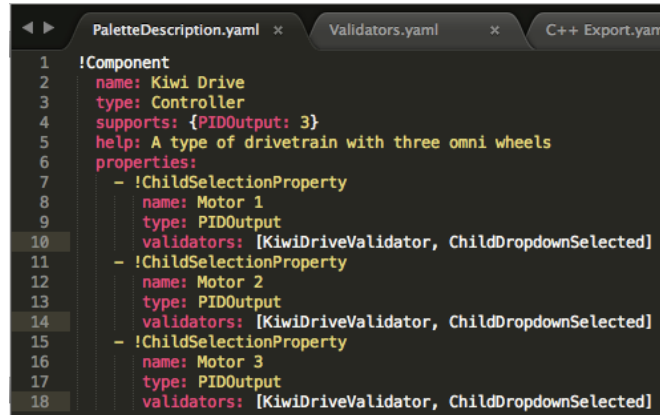
Özel Bileşen Yapısı



Özel bileşenlerin tümü yıl olarak burada bulunur. ~/wpilib/YYYY/Robotbuilder/extensions where ~ is C:\Users\Public on Windows and YYYY is the FRC® year.

Özel bir bileşen için gereken yedi dosya ve bir klasör vardır. Klasör, bileşeni ve nasıl dışa aktarılacağını açıklayan dosyaları içerir. Bileşenle aynı ada sahip olmalıdır (örneğin, bir Kiwi sürücü denetleyicisi için "Kiwi Drive", altı motorlu sürücü denetleyicisi için "Robot Drive 6", vb.). Dosyalar, burada gösterilenlerle aynı adlara ve uzantılara sahip olmalıdır. Diğer dosyalar bu yedi ile birlikte klasörde olabilir, ancak yedi tanesi RobotBuilder'ın özel bileşeni tanıması için mevcut olmalıdır.

PaletteDescription.yaml



Satır satır:

- !Component-Bileşen: Yeni bir bileşenin başlangıcını bildirir
- name: Bileşenin adı. Palette/ağaçta görünecek olan budur - bu aynı zamanda içeren klasörün adıdır aynı olmalıdır.
- type-tür: bileşenin türü (bunlar daha sonra ayrıntılı olarak açıklanacaktır)

- supports-destekler: destekleyebileceği her bileşen türünün miktarının bir haritası. RobotBuilder'daki motor kontrolörlerinin tümü PIDOutput'lardır, bu nedenle bir kivi sürücüsü üç PIDOutput'u destekleyebilir. Bir bileşen herhangi bir şeyi desteklemiyorsa (sensörler veya motor kontrol cihazları gibi), bu satırı dışarıda bırakın
- help-yardım: bu bileşenlerden birinin üzerine geldiğinde faydalı bir mesaj veren kısa bir dize
- properties: bu bileşenin özelliklerinin listesi. Bu kivi sürücü örneğinde, her motor için bir tane olmak üzere çok benzer üç özellik vardır. Bir ChildSelectionProperty, kullanıcının, düzenlenen bileşenin alt bileşenlerinden belirli türden bir bileşeni seçmesine olanak tanır (bu nedenle, burada, kiwi sürücüsüne eklenen bir PIDOutput - yani bir motor denetleyicisi - isteyen bir açılır menü gösterilir)

RobotBuilder'ın desteklediği bileşen türleri (bunlar büyük / küçük harfe duyarlıdır):

- Komut
- Alt sistem
- PIDOutput (motor kontrolleri)
- PIDSource (PIDSource uygulayan sensör, örneğin analog potansiyometre, kodlayıcı)
- Sensör (PIDSource uygulamayan sensör, örneğin limit anahtarı)
- Kontrolör (robot sürücü, PID kontrolör vb.)
- Aktüatör (motor olmayan bir çıkış, örneğin solenoid, servo)
- Joystick - Oyun kolu
- Joystick Düğmesi

Properties-Özellikleri

Özel bir bileşenle ilgili özellikler:

- StringProperty: bir bileşen bir dizeye ihtiyaç duyduğunda kullanılır, ör. bileşenin adı
- BooleanProperty: bir bileşen bir boole değerine ihtiyaç duyduğunda kullanılır, ör. Smart-Dashboard'a bir düğme koymak
- DoubleProperty: bir bileşen bir sayı değerine ihtiyaç duyduğunda kullanılır, ör. PID sabitleriSeçimlerÖzellik
- ChildSelectionProperty: bir alt bileşen seçmeniz gerektiğinde kullanılır, ör. bir RobotDrive'daki motor kontrolörleri
- TypeSelectionProperty: programın herhangi bir yerinden verilen türdeki herhangi bir bileşeni seçmeniz gerektiğinde kullanılır, ör. PID komutu için giriş ve çıkış

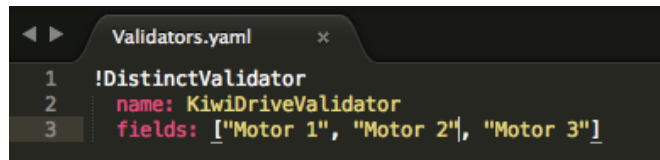
Her mülkün alanları aşağıda açıklanmıştır:

```

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



```

1 !DistinctValidator
2   name: KiwiDriveValidator
3   fields: ["Motor 1", "Motor 2", "Motor 3"]

```

PaletteDescription.yaml içindeki motor özelliklerinin her birinin doğrulayıcılar girişinde “KiwiDriveValidator” olduğunu fark etmiş olabilirsiniz. Yerleşik bir doğrulayıcı olmadığı için Validators.yaml’de tanımlanması gerekiyordu. Bu örnek doğrulayıcı çok basittir - sadece adlandırılmış alanların her birinin diğerlerinden farklı bir değere sahip olmasını sağlar.

Yerleşik Validator-doğrulamayı ve Validator Türleri

```

Validators:
- !DistinctValidator
  name: RobotDrive2
  fields: ["Left Motor", "Right Motor"]
- !DistinctValidator
  name: RobotDrive3
  fields: ["Left Front Motor", "Left Rear Motor", "Right Front Motor", "Right Rear Motor"]
- !ExistsValidator
  name: ChildDropDownSelected
  ignore: [null, "null", "", 0, 1, 2, 3, "No Choices Available", "None"]
  error: "You must select a component of the valid type beneath this item. If no options exist, drag one under this component."
- !ExistsValidator
  name: TypeDropDownSelected
  ignore: [null, "null", "", 0, 1, 2, 3, "No Choices Available", "None"]
  error: "You must select a component of the valid type. If no options exist, create a new component of the right type."
- !UniqueValidator
  name: AnalogInput
  fields: [Channel (Analog)]
- !UniqueValidator
  name: DigitalChannel
  fields: [Channel (Digital)]
- !UniqueValidator
  name: PWMOutput
  fields: [Channel (PWM)]
- !UniqueValidator
  name: CANID
  fields: [CAN ID]
- !UniqueValidator
  name: Joystick
  fields: [Number]
- !UniqueValidator
  name: RelayOutput
  fields: [Channel (Relay)]
- !UniqueValidator
  name: Solenoid
  fields: [Channel (Solenoid), POH (Solenoid)]
- !UniqueValidator
  name: POCCompID
  fields: [POH ID]
- !ListValidator
  name: List

```

Yerleşik doğrulamalar çok kullanışlıdır (özellikle bağlantı noktası / kanal kullanımı için UniqueValidators), ancak bazen önceki adımda olduğu gibi özel bir doğrulamaya ihtiyaç duyulur

- DistinctValidator: Verilen alanların her birinin değerlerinin benzersiz olduğundan emin olur
- ExistsValidator: Bu doğrulamayı kullanarak cihaz için bir değerin ayarlandığından emin olur
- UniqueValidator: Özelliğin değerinin verilen alanlar için genel olarak benzersiz olduğundan emin olur
- ListValidator: Bir liste özelliğindeki tüm değerlerin geçerli olduğundan emin olur

C++ Export.yaml

```

1 Kiwi Drive
2 Defaults: "CustomComponent,None"
3 className: "KiwiDrive"
4 Construction: "#variable($Name).reset(new ${ClassName}($variable(Motor_1), $variable(Motor_2), $variable(Motor_3)))"

```

Dosyanın satır satır dökümü:

- Kiwi Drive: dışa aktarılan bileşenin adı. Bu, PaletteDescription.yaml'de ayarlanan ad ve bu dosyayı içeren klasörün adıdır.
- Varsayılanlar: Bu bileşenin ihtiyaç duyduğu dahil etme, sınıfın adı, bir yapı şablonu ve daha fazlası için bazı varsayılan değerler sağlar. CustomComponent varsayılanı, bileşeni kullanan her oluşturulan dosyaya Custom/\${ClassName}.h için bir include ekler (örneğin, RobotDrive.h, #include "Custom/KiwiDrive.h" içerir. dosyanın üst kısmı)
- className: eklediğiniz özel sınıfın adı.
- Construction : bileşenin nasıl inşa edilmesi gerektiğine dair bir talimat. Değişkenler değerleri ile değiştirilecek ("\${ClassName}", "KiwiDrive" ile değiştirilecektir), ardından makrolar değerlendirilecektir (örneğin, `#variable(\$Name)` , drivebaseKiwiDrive ile değiştirilebilir `).

Bu örnek, constructor-yapıcıyla bir KiwiDrive sınıfı beklemektedir.

```
KiwiDrive(SpeedController, SpeedController, SpeedController)
```

Ekibiniz Java kullanıyorsa, bu dosya boş olabilir.

Java Export.yaml

```
Java Export.yaml
1 Kiwi_Drive:
2 Defaults: "CustomComponent,None"
3 ClassName: "KiwiDrive"
4 Construction: "Variable($Name) = new ${ClassName}($Variable($Motor_1), $Variable($Motor_2), $Variable($Motor_3));"
```

C ++ dışa aktarma dosyasına çok benzer; tek fark İnşaat hattı olmalıdır. Bu örnek, yapıcıyla bir KiwiDrive sınıfı beklemektedir.

```
KiwiDrive(SpeedController, SpeedController, SpeedController)
```

Ekibiniz C ++ kullanıyorsa, bu dosya boş olabilir.

Makroları ve Değişkenleri Kullanma

Makrolar, RobotBuilder’ın değişkenleri oluşturulan koda eklenecek metne dönüştürmek için kullandığı basit işlevlerdir. Her zaman “#” simgesiyle başlarlar ve işlevlere benzer bir sözdizimine sahiptirler: `` <macro_name> (arg0, arg1, arg2, ...) `` Muhtemelen kullanmanız gereken tek makro `` #variable (bileşen_adı) `` dır.

#variable bir string alır, genellikle bir yerde tanımlanmış bir değişkendir (yani “Name”, RobotBuilder’daki bileşene verilen addır, “Kol Motoru” gibi) ve onu içinde tanımlanan bir değişkenin adına dönüştürür. üretilen kod. Örneğin, #variable (“Arm Motor”), ArmMotor dizesiyle sonuçlanır

Değişkenlere, değişken adının önüne bir dolar işareti (“\$”) yerleştirilerek başvurulur; bu, değişkeni dosyadaki diğer metinden kolayca ayırt etmek için isteğe bağlı olarak güzel parantez içine yerleştirilebilir. Dosya ayrıştırıldığında dolar işareti, değişken adı ve küme parantezleri değişkenin değeriyle değiştirilir (örneğin, \${ClassName}, KiwiDrive “ ile değiştirilir).

Değişkenler, bileşen özellikleridir (örn. Kiwi sürücü örneğinde “Motor 1”, “Motor 2”, “Motor 3”) veya aşağıdakilerden biridir:

1. Short_Name: RobotBuilder’daki editör panelinde bileşene verilen ad
2. Ad: bileşenin tam adı. Bileşen bir alt sistemdeyse, bu, alt sistemin adına eklenen kısa ad olacaktır.
3. Export-Dışa Aktar: Varsa, bu bileşenin içinde oluşturulacağı dosyanın adı. Bu, aktüatörler, kontrolörler ve sensörler gibi bileşenler için “RobotMap” olmalıdır; veya oyun kumandaları veya diğer özel OI bileşenleri gibi şeyler için “OI”. “CustomComponent” varsayılanının RobotMap’e aktarılacağını unutmayın.
4. Import : Bu bileşenin kullanılabilmesi için dahil edilmesi veya içe aktarılması gereken dosyalar.
5. Declaration: Bu bileşen türünde bir değişkenin nasıl beyan edileceğine dair Construction’a benzer bir talimat. Bu, varsayılan “None” olarak alınır.
6. Construction: bu bileşenin yeni bir örneğinin nasıl oluşturulacağına ilişkin talimat
7. LiveWindow: bu bileşenin LiveWindow’a nasıl ekleneceğine dair bir talimat
8. Extra: kodlama türünü ayarlaması gereken kodlayıcılar gibi bu bileşenin doğru davranması için herhangi bir ekstra işlev veya yöntem çağırısı için talimatlar.

9. Prototip (yalnızca C ++): Bileşenin bildirildiği dosyada oluşturulacak bir işlevin prototipi, tipik olarak OI sınıfında bir alıcı
10. Function: Bileşenin bildirildiği dosyada oluşturulacak bir işlev, genellikle OI sınıfındaki bir alıcı
11. PID: Eğer varsa, bileşenin PID çıktısının nasıl alınacağına dair bir talimat (örneğin, #variable(\$Short_Name) ->PIDGet())
12. ClassName: Bileşenin temsil ettiği sınıfın adı (örneğin, KiwiDrive veya Joystick)

Adında boşluklar olan değişkenleriniz varsa ("Motor 1", "Sağ Ön Motor" vb.), Dışa aktarma dosyalarında bunları kullanırken boşlukların alt çizgi ile değiştirilmesi gerekir.

help.html

```
1 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
2
3 <head>
4 <link rel="stylesheet" href="styles.css" type="text/css" media="screen" />
5 </head>
6
7 <body>
8 <h1>Kiwi Drive</h1>
9 <center></center>
10 <h2>What is it?</h2>
11 <p>
12 Kiwi drive is a type of omni-directional drivetrain with three omni wheels,
13 usually at 120° angles to each other.
14 </p>
15 <h2>Properties</h2>
16 <dl>
17 <dt>Motor 1</dt>
18 <dd>The first motor</dd>
19 <dt>Motor 2</dt>
20 <dd>The second motor</dd>
21 <dt>Motor 3</dt>
22 <dd>The third motor</dd>
23 </dl>
24 <h2>See Also</h2>
25 <ul>
26 <li>
27 <a href="http://en.wikipedia.org/wiki/Kiwi_drive">Kiwi drive on Wikipedia</a>
28 </li>
29 </ul>
30 </body>
31 </html>
```

Bileşen hakkında bilgi veren bir HTML dosyası. Bunun olabildiğince ayrıntılı olması daha iyidir, ancak programcı (lar) bileşene yeterince aşina ise veya o kadar basitse, ayrıntılı bir açıklamada çok az nokta varsa kesinlikle gerekli değildir.

config.txt

```
1 section=Controllers
```

Bileşen hakkında çeşitli bilgileri tutan bir yapılandırma dosyası. Şu anda, bu yalnızca bileşeni yerleştirmek için palet bölümüne sahiptir.

Paletin bölümleri (bunlar büyük / küçük harfe duyarlıdır):

- Alt sistemler
- Kontrolörler

- Sensörler
- Aktüatörler
- Pnömatik
- OI
- Komutlar

icon.png

Palette ve yardım sayfasında görünen simge. Bu 64x64 .png dosyası olmalıdır.

Görsel dağınıklığı önlemek için içinde bulunduğu bölümün renk şemasını ve genel stilini kullanmalıdır, ancak bu tamamen isteğe bağlıdır. Simgelerin ve arka planların Photoshop .psd dosyaları [src / main / icons / icons](#) ve simgelerin ve arka planların png dosyaları [src / main / resources / icons](#) içinde yer alır.

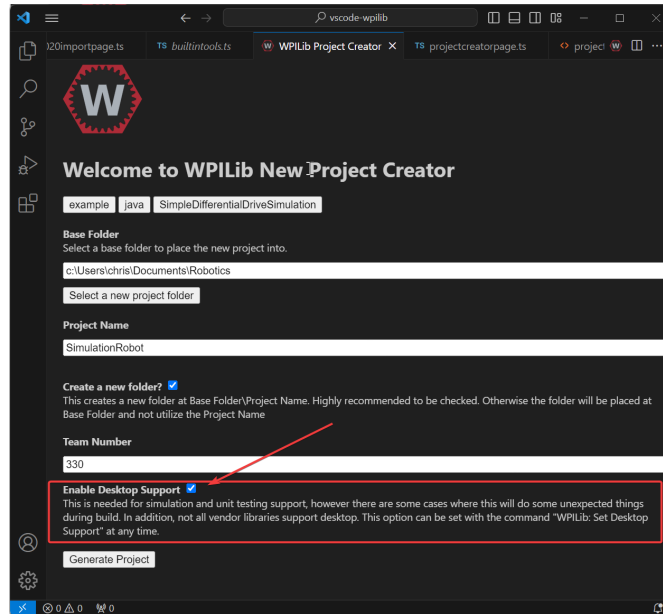
Robot Simülasyonu

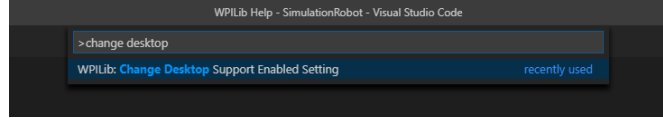
22.1 Robot Simülasyonuna Giriş

Genellikle bir takım, gerçek bir robotu olmadan kodunu test etmek isteyebilir. WPILib, ekiple-re basit derecelendirme komutlarını kullanarak çeşitli robot özelliklerini simüle etme yeteneği sağlar.

Java/C++

Masaüstü Simülatörünün kullanılması Masaüstü Desteğinin etkinleştirilmesini gerektirir. Bu, robot projenizi oluştururken “Enable Desktop Support Checkbox” işaretleyerek veya Visual Studio Code komut paletinden “WPILib: Change Desktop Support Enabled Setting” i çalıştırarak yapılabilir.





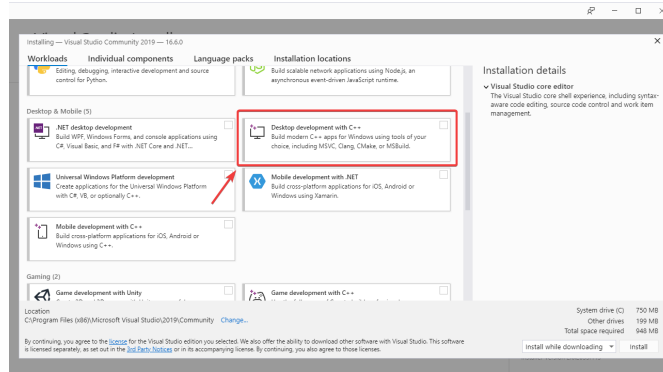
Masaüstü desteği, robot projenizin kök dizininde bulunan `build.gradle` dosyanızı manuel olarak düzenleyerek de etkinleştirilebilir. Basitçe `includeDesktopSupport = false` u `includeDesktopSupport = true` olarak değiştirin

Önemli: Masaüstü/simülasyon desteğinin etkinleştirilmesinin istenmeyen sonuçlara yol açabileceğine dikkat etmek önemlidir. Tüm üreticiler bu seçeneği desteklemez ve kitaplıklarını kullanan kod, simülasyonu çalıştırmaya çalışırken bile çökebilir!

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`.

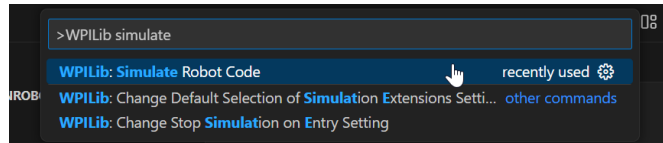
Not: C++ robot simulation requires that a native compiler to be installed. For Windows, this would be [Visual Studio 2022 version 17.9 or later](#) (**not** VS Code), macOS requires [Xcode 14 or later](#), and Linux (Ubuntu) requires the `build-essential` package.

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



Running Robot Simulation

Temel robot simülasyonu VS Code kullanılarak çalıştırılabilir. Bu, VS Code'un komut paleti kullanılarak herhangi bir komut kullanılmadan yapılabilir.



Visual Studio Code'daki konsol çıktınız aşağıdaki gibi görünmelidir. Ancak, takımlar muhtemelen simülasyonu çalıştırmak yerine kodlarını gerçekten *test etmek* isteyeceklerdir. Bu, aşağıdakiler kullanılarak yapılabilir [WPILib'in Simülasyon GUI](#).

```
***** Robot program starting *****
Default disabledInit() method... Override me!
Default disabledPeriodic() method... Override me!
Default robotPeriodic() method... Override me!
```

Önemli: Simulation can also be run outside of VS Code using `./gradlew simulateJava` for Java or `./gradlew simulateNative` for C++.

Not: 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

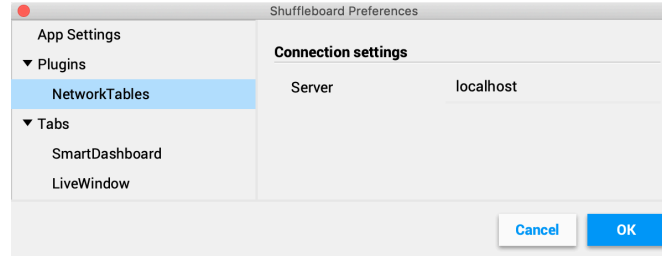
```
python3 -m robotpy sim
```

22.1.1 Robot Gösterge Panellerini Çalıştırma

Shuffleboard, SmartDashboard, Glass, and AdvantageScope can be used with WPILib simulation when they are configured to connect to the local computer (i.e. `localhost`).

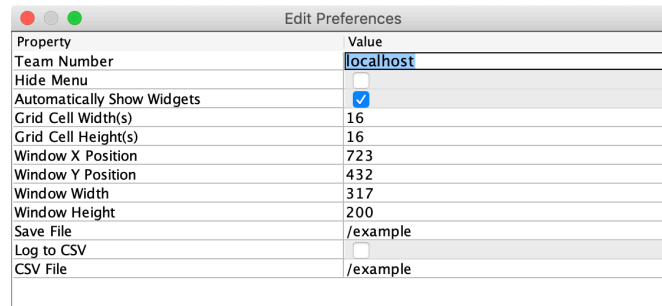
Shuffleboard-Karıştırma panosu

Shuffleboard is automatically configured to look for a `NetworkTables` instance from the robot-RIO but **not from other sources**. To connect to a simulation, open Shuffleboard preferences from the *File* menu and select *NetworkTables* under *Plugins* on the left navigation bar. In the *Server* field, type in the IP address or hostname of the `NetworkTables` host. For a standard simulation configuration, use `localhost`.



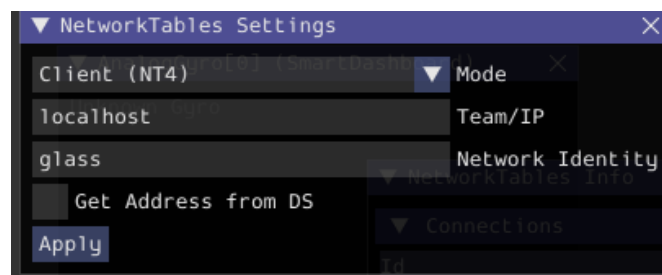
SmartDashboard

SmartDashboard is automatically configured to look for a NetworkTables instance from the roboRIO, but **not from other sources**. To connect to a simulation, open SmartDashboard preferences under the *File* menu and in the *Team Number* field, enter the IP address or hostname of the NetworkTables host. For a standard simulation configuration, use `localhost`.



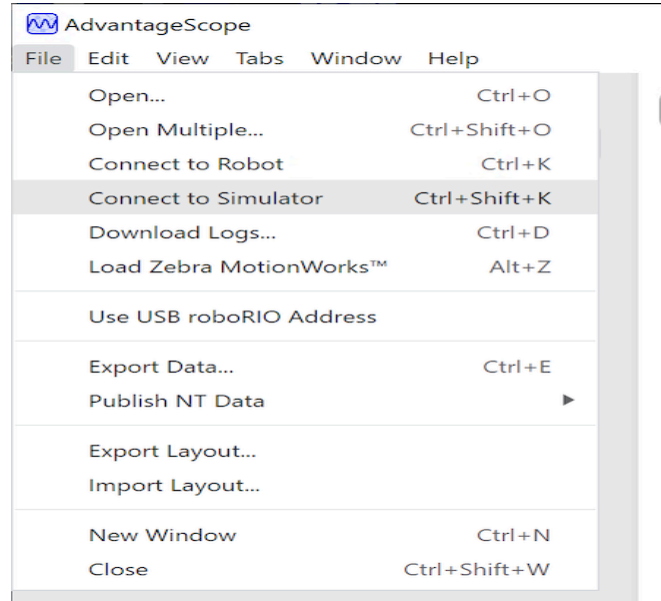
Glass

Glass is automatically configured to look for a NetworkTables instance from the roboRIO, but **not from other sources**. To connect to a simulation, open *NetworkTables Settings* under the *NetworkTables* menu and in the *Team/IP* field, enter the IP address or hostname of the NetworkTables host. For a standard simulation configuration, use `localhost`.



AdvantageScope

No configuration is required to connect to a NetworkTables instance running on the local computer. To connect to a simulation, click *Connect to Simulator* under the *File* menu or press Ctrl+Shift+K.

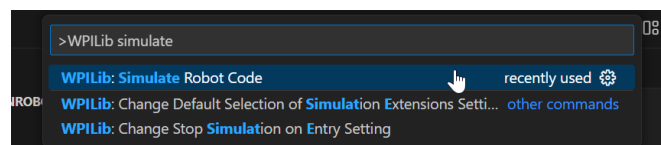


22.2 Simulation Specific User Interface Elements

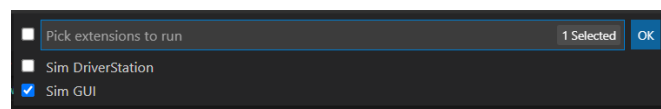
WPILib, bir grafik kullanıcı arabirimi (GUI) bileşeni sunmak için genişletilmiş robot simülasyonuna sahiptir. Bu, ekiplerin robotlarının girdilerini ve çıktılarını kolayca görselleştirmesine olanak tanır.

Not: The Simulation GUI is very similar in many ways to *Glass*. Some of the following pages will link to Glass sections that describe elements common to both GUIs.

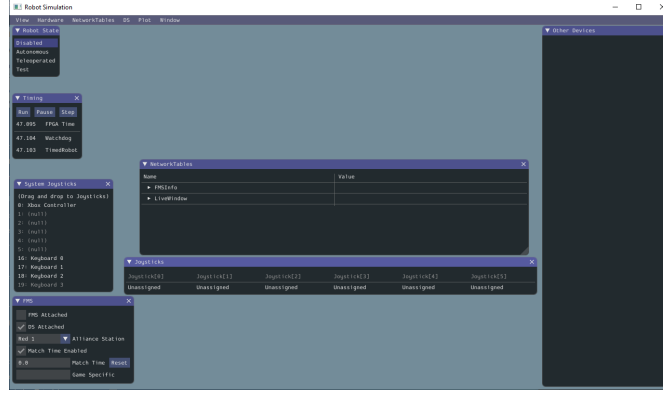
22.2.1 GUI'yi çalıştırma



GUI'yi **Run Simulation** komut paleti seçeneğiyle başlatabilirsiniz.

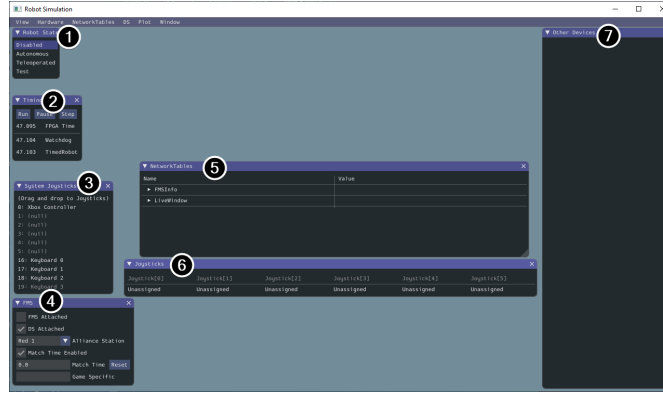


And the Sim GUI option should popup in a new dialog and will be selected by default. Press *Ok*. This will now launch the Simulation GUI!



22.2.2 GUI'yi kullanma

Düzeni Öğrenmek



Aşağıdaki öğeler simülasyon GUI'sinde varsayılan olarak gösterilir:

1. **Robot State** - Bu, robotun mevcut durumu veya "modudur". Normal Driver Station'da yaptığınız gibi modu değiştirmek için etiketlere tıklayabilirsiniz.
2. **Timing-Zamanlama** - Robotun zamanlayıcılarının değerlerini gösterir ve zamanlamanın değiştirilmesine izin verir.
3. **System Joysticks** - Bu, şu anda sisteminize bağlı kumanda çubuklarının bir listesidir.
4. **FMS** - This is used for simulating many of the common *FMS* systems.
5. **NetworkTables** - Bu, NetworkTables'da yayınlanan verileri gösterir.
6. **Joysticks** - Bu, robot kodunun doğrudan çekebileceği joysticklerdir.
7. **Other Devices-Diğer Cihazlar** - Bu, Parça Kitine dahil olan ADXRS450 cayro veya simülasyonu destekleyen üçüncü taraf cihazlar gibi diğer kategorilerin hiçbirine girmeyen cihazları içerir.

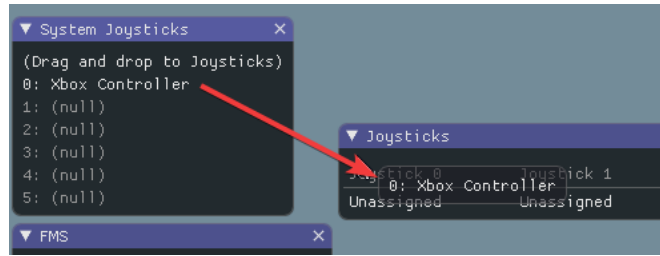
Aşağıdaki öğeler Donanım menüsünden eklenebilir ancak varsayılan olarak gösterilmez.

1. ****Addressable LEDs-Adreslenebilir LED'ler **** - Bu, AddressableLED Sınıfı tarafından kontrol edilen LED'leri gösterir.
2. **Analog Inputs** - Bu, Analog tabanlı jiroskoplar gibi normalde roboRIO'da **ANALOG IN** konektörünü kullanan tüm cihazları içerir.

3. **DIO** - (Dijital Giriş Çıkışı) Bu, roboRIO'da **DIO** konektörünü kullanan tüm cihazları içerir.
4. **Encoders** - Bu, Encoder sınıfını genişleten veya kullanan tüm somutlaştırılmış cihazları gösterecektir.
5. **PDPs-PDP'ler** - Bu, Power Distribution Panel nesnesini gösterir.
6. **PWM Outputs** - This is a list of instantiated *PWM* devices. This will appear as many devices as you instantiate in robot code, as well as their outputs.
7. **Relays** - Bu, tüm röle cihazlarını içerir. Buna VEX Spike röleleri dahildir.
8. **Solenoids** - Bu, "connected" solenoidlerin bir listesidir. Bir solenoid nesne oluşturduğunuzda ve çıkışları ittiğinizde, bunlar burada gösterilir.

Kumanda Çubuklarına System Kumanda Çubuğu Ekleme

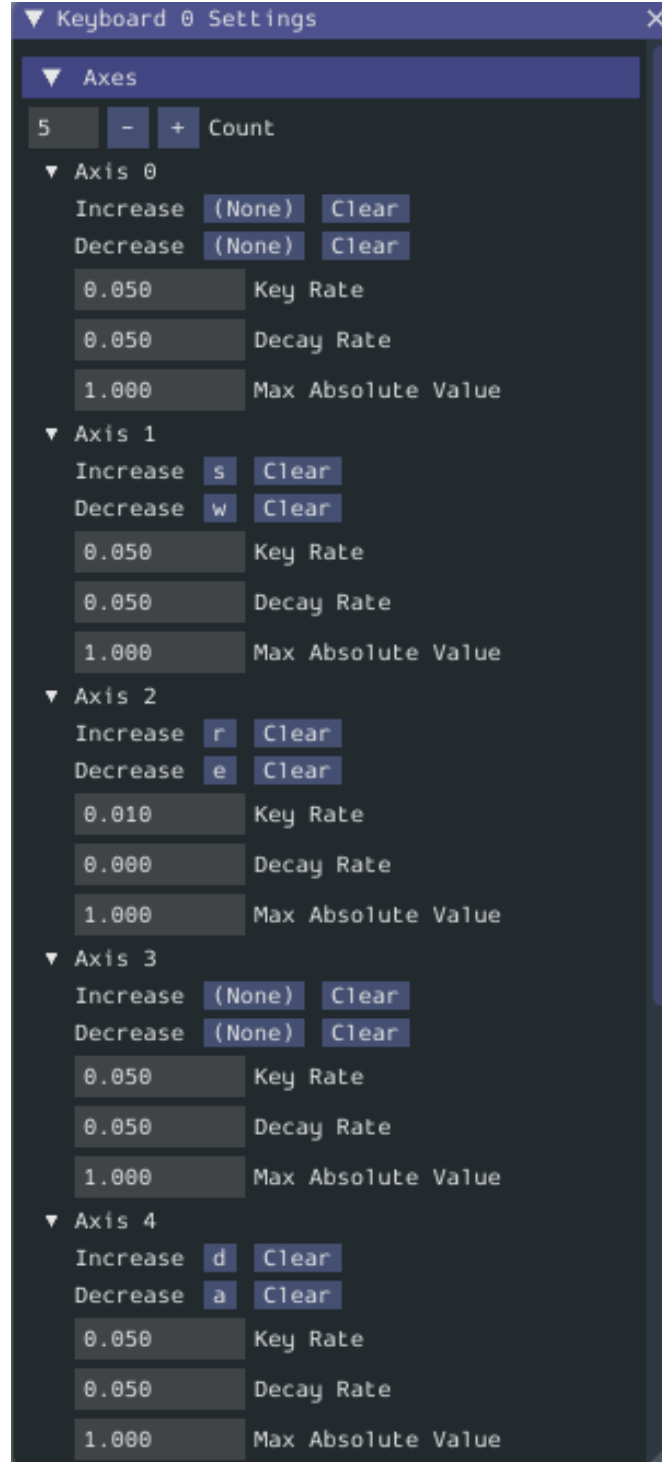
Sistem kumanda çubukları listesinden bir kumanda çubuğu eklemek için, "System Joysticks" menüsü altında gösterilen bir kumanda çubuğuna tıklayıp "Joysticks" menüsüne sürükleyin.



Not: FRC ® Driver Station, bağlı oyun kumandalarına özel eşleme yapar ve WPILib simülatörü bunları varsayılan olarak "eşleme" yapmaz. Bu davranışı, "Joysticks" menüsünün altındaki "Map gamepad-Harita oyun kumandası" geçiş düğmesine basarak etkinleştirebilirsiniz.

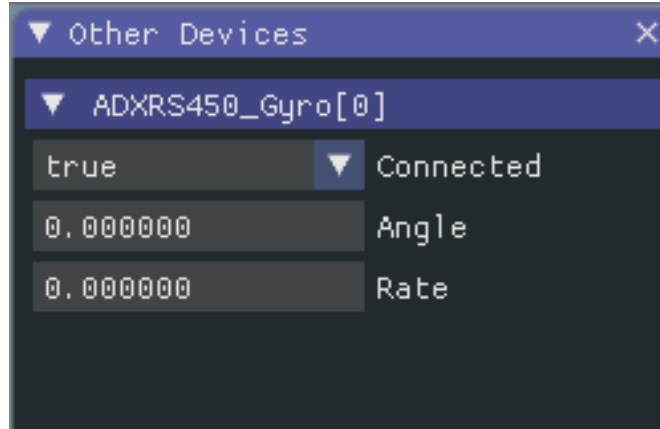
Klavyeyi Joystick olarak kullanma

You add a keyboard to the list of system joysticks by clicking and dragging one of the keyboard items (e.g. Keyboard 0) just like a joystick above. To edit the settings of the keyboard go to the *DS* item in the menu bar then choose *Keyboard 0 Settings*. This allows you to control which keyboard buttons control which axis. This is a common example of how to make the keyboard similar to a split sticks arcade drive on an Xbox controller (uses axis 1 & 4):



ADXRS450 Girişlerini Değiştirme

ADXRS450 nesnesini kullanmak, cayo tabanlı çıktıları test etmenin harika bir yoludur. Bu, “Other Devices” menüsünde görünecektir. Daha sonra “Connected”, “Angle” ve “Rate” gibi çeşitli seçenekleri gösteren bir açılır menü ortaya çıkar. Bu değerlerin tümü, değiştirebileceğiniz ve robotunuzun anında kodlayıp kullandığı değerlerdir.



22.2.3 Robot Kodundan Simülasyonu Belirleme

Üretici kitaplıklarının robot simülasyonunu çalıştırırken derlenmediği durumlarda, içeriklerini bir boolean döndüren `RobotBase.isReal()` ile kapsayabilirsiniz.

JAVA

```
TalonSRX motorLeft;
TalonSRX motorRight;

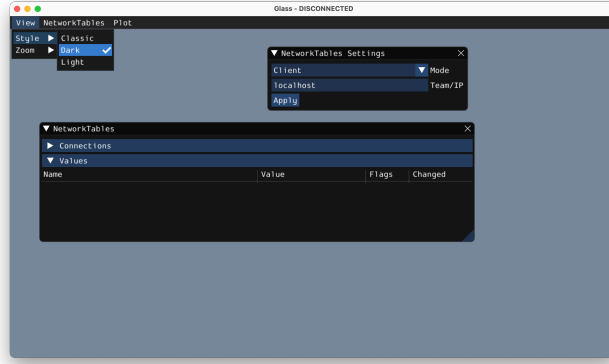
public Robot() {
    if (RobotBase.isReal()) {
        motorLeft = new TalonSRX(0);
        motorRight = new TalonSRX(1);
    }
}
```

Not: C++ 'da değer türlerini yeniden atamak, taşıma veya kopyalama ataması gerektirir; Hem SIM'i desteklemeyen hem de taşıma veya kopyalama atama operatörü olmayan üretici sınıfları, bir değer türü yerine bir işaretçi kullanılmadıkça koşullu tahsis ile çalışamaz.

22.2.4 Görünüm Ayarlarını Değiştirme

View menü öğesi şunları içerir *Zoom* ve *Style* özelleştirilebilir ayarları. *Zoom* seçeneği, uygulamadaki metnin boyutunu belirler, oysa *Style* seçeneği, *Classic*, *Light* ve *Dark* modlar arasında seçim yapmanızı sağlar .

Dark stil ayarının bir örneği aşağıdadır:



22.2.5 Uygulama Verilerini Temizleme

Application data for the Simulation GUI, including widget sizes and positions as well as other custom information for widgets is stored in a `imgui.ini` file. This file is stored in the root of the project directory that the simulation is run from.

The `imgui.ini` configuration file can simply be deleted to restore the Simulation GUI to a “clean slate”.

22.3 WPILib ile Fizik Simülasyonu

Çünkü *durum-uzay gösterimi* , `:systems` 'in dynamics kompakt bir şekilde temsil etmemize olanak tanır, robotlarda fiziksel sistemleri simüle etmek için onu kullanabiliriz. Bu simülatörlerin amacı, mevcut simülasyon dışı kullanıcı kodunu değiştirmeden robot mekanizmalarının hareketini simüle etmektir. Bu tür simülatörlerin temel akışı aşağıdaki gibidir:

- Normal kullanıcı kodunda:
 - PID veya benzer kontrol algoritmaları, kodlayıcı (veya diğer sensör) okumalarından voltaj komutları üretir
 - Motor çıkışları ayarlandı
- Simülasyon periyodik kodunda:
 - Simülasyonun `state` 'i, `:term:inputs` genellikle bir PID döngüsünden ayarlanmış motorlardan gelen voltajlar kullanılarak güncellenir.
 - Simüle edilmiş kodlayıcı (veya diğer sensör) okumaları, sonraki zaman adımında kullanıcı kodunun kullanılması için ayarlanır

22.3.1 WPILib'in Simülasyon Sınıfları

WPILib'de aşağıdaki fizik simülasyon sınıfları mevcuttur:

- Doğrusal dinamikli sistemleri modellemek için LinearSystemSim,
- FlywheelSim - Volan Sim
- DiferansiyelDrivetrainSim
- ElevatorSim, which models gravity in the direction of elevator motion
- SingleJointedArmSim, which models gravity proportional to the arm angle
- Akü voltajı düşüşünü çeken akımlara göre tahmin eden BatterySim

Tüm simülasyon sınıfları (diferansiyel sürücü simülatörü haricinde) LinearSystemSim sınıfından miras alır. Varsayılan olarak, dinamikler doğrusal sistem dinamikleridir $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$. Alt sınıflar UpdateX(x, u, dt) gibi metotları, yerçekimi modelleme, özel ve doğrusal olmayan dinamikler sağlamak için yöntemini geçersiz kılar.

Not: 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 Kullanıcı Kodunda Kullanım

Aşağıdakiler WPILib'den edinilebilir elevatorsimulation [example project](#).

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.

Not: Asansör ve kol simülatörleri, simüle edilmiş konumun verilen minimum veya maksimum yükseklik veya açıları aşmasını önleyecektir. Sonsuz dönme veya harekete sahip bir mekanizmayı simüle etmek istiyorsanız, LinearSystemSim daha iyi bir seçenek olabilir.

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,

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

55     Constants.kMaxElevatorHeightMeters,
56     true,
57     0,
58     VecBuilder.fill(0.01));
59 private final EncoderSim m_encoderSim = new EncoderSim(m_encoder);

```

C++

```

51 // Simulation classes help us simulate what's going on, including gravity.
52 frc::sim::ElevatorSim m_elevatorSim{m_elevatorGearbox,
53                                     Constants::kElevatorGearing,
54                                     Constants::kCarriageMass,
55                                     Constants::kElevatorDrumRadius,
56                                     Constants::kMinElevatorHeight,
57                                     Constants::kMaxElevatorHeight,
58                                     true,
59                                     0_m,
60                                     {0.01}};
61 frc::sim::EncoderSim m_encoderSim{m_encoder};

```

Daha sonra, teleopPeriodic/TeleopPeriodic (Java/C++), asansörümüzü yerden 30 inç yükseklikte bir ayar noktasına götürmek için basit bir PID kontrol döngüsü kullanır.

Java

```

31 @Override
32 public void teleopPeriodic() {
33     if (m_joystick.getTrigger()) {
34         // Here, we set the constant setpoint of 0.75 meters.
35         m_elevator.reachGoal(Constants.kSetpointMeters);
36     } else {
37         // Otherwise, we update the setpoint to 0.
38         m_elevator.reachGoal(0.0);
39     }
40 }

```

```

99 public void reachGoal(double goal) {
100     m_controller.setGoal(goal);
101
102     // With the setpoint value we run PID control like normal
103     double pidOutput = m_controller.calculate(m_encoder.getDistance());
104     double feedforwardOutput = m_feedforward.calculate(m_controller.getSetpoint().
105     ↪ velocity);
106     m_motor.setVoltage(pidOutput + feedforwardOutput);

```

C++

```

20 void Robot::TeleopPeriodic() {
21     if (m_joystick.GetTrigger()) {
22         // Here, we set the constant setpoint of 0.75 meters.
23         m_elevator.ReachGoal(Constants::kSetpoint);
24     } else {
25         // Otherwise, we update the setpoint to 0.
26         m_elevator.ReachGoal(0.0_m);
27     }
28 }

42 void Elevator::ReachGoal(units::meter_t goal) {
43     m_controller.SetGoal(goal);
44     // With the setpoint value we run PID control like normal
45     double pidOutput =
46         m_controller.Calculate(units::meter_t{m_encoder.GetDistance()});
47     units::volt_t feedforwardOutput =
48         m_feedforward.Calculate(m_controller.GetSetpoint().velocity);
49     m_motor.SetVoltage(units::volt_t{pidOutput} + feedforwardOutput);
50 }

```

Daha sonra, `simulationPeriodic/SimulationPeriodic` (Java/C++), asansörün simüle edilmiş konumunu güncellemek için motora uygulanan gerilimi kullanır. Sadece simüle edilmiş robotlar için periyodik olarak çalıştığı için `SimulationPeriodic` kullanıyoruz. Bu, simülasyon kodumuzun gerçek bir robot üzerinde çalıştırılmayacağı anlamına gelir.

Not: 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`.

Son olarak, simüle edilmiş kodlayıcının mesafe okuması, simüle edilmiş asansörün konumu kullanılarak ayarlanır ve robotun pil voltajı, asansör tarafından çekilen tahmini akımı kullanılarak ayarlanır.

Java

```

79 public void simulationPeriodic() {
80     // In this method, we update our simulation of what our elevator is doing
81     // First, we set our "inputs" (voltages)
82     m_elevatorSim.setInput(m_motorSim.getSpeed() * RobotController.
83     ↪ getBatteryVoltage());
84
85     // Next, we update it. The standard loop time is 20ms.
86     m_elevatorSim.update(0.020);
87
88     // Finally, we set our simulated encoder's readings and simulated battery voltage
89     m_encoderSim.setDistance(m_elevatorSim.getPositionMeters());
90     // SimBattery estimates loaded battery voltages
91     RoboRioSim.setVInVoltage(
92     ↪ BatterySim.calculateDefaultBatteryLoadedVoltage(m_elevatorSim.
93     ↪ getCurrentDrawAmps()));
94 }

```

C++

```
20 void Elevator::SimulationPeriodic() {
21     // In this method, we update our simulation of what our elevator is doing
22     // First, we set our "inputs" (voltages)
23     m_elevatorSim.SetInput(frc::Vectord<1>{
24         m_motorSim.GetSpeed() * frc::RobotController::GetInputVoltage()});
25
26     // Next, we update it. The standard loop time is 20ms.
27     m_elevatorSim.Update(20_ms);
28
29     // Finally, we set our simulated encoder's readings and simulated battery
30     // voltage
31     m_encoderSim.SetDistance(m_elevatorSim.GetPosition().value());
32     // SimBattery estimates loaded battery voltages
33     frc::sim::RoboRioSim::SetVInVoltage(
34         frc::sim::BatterySim::Calculate({m_elevatorSim.GetCurrentDraw()}));
35 }
```

22.4 Device Simulation

WPILib, SimDevice API biçiminde simülasyon cihazı verilerini yönetmenin bir yolunu sağlar.

22.4.1 Simulating Core WPILib Device Classes

Core WPILib device classes (i.e Encoder, Ultrasonic, etc.) have simulation classes named EncoderSim, UltrasonicSim, and so on. These classes allow interactions with the device data that wouldn't be possible or valid outside of simulation. Constructing them outside of simulation likely won't interfere with your code, but calling their functions and the like is undefined behavior - in the best case they will do nothing, worse cases might crash your code! Place functional simulation code in simulation-only functions (such as `simulationPeriodic()`) or wrap them with `RobotBase.isReal()` / `RobotBase::IsReal()` checks (which are `constexpr` in C++).

Not: This example will use the EncoderSim class as an example. Use of other simulation classes will be almost identical.

Simülasyon Cihazı Nesneleri Oluşturma

Simulation device object can be constructed in two ways:

- a constructor that accepts the regular hardware object.
- a constructor or factory method that accepts the port/index/channel number that the device is connected to. These would be the same number that was used to construct the regular hardware object. This is especially useful for *unit testing*.

JAVA

```
// create a real encoder object on DIO 2,3
Encoder encoder = new Encoder(2, 3);
// create a sim controller for the encoder
EncoderSim simEncoder = new EncoderSim(encoder);
```

C++

```
// create a real encoder object on DIO 2,3
frc::Encoder encoder{2, 3};
// create a sim controller for the encoder
frc::sim::EncoderSim simEncoder{encoder};
```

Reading and Writing Device Data

Each simulation class has getter (getXxx()/GetXxx()) and setter (setXxx(value)/SetXxx(value)) functions for each field Xxx. The getter functions will return the same as the getter of the regular device class.

JAVA

```
simEncoder.setCount(100);
encoder.getCount(); // 100
simEncoder.getCount(); // 100
```

C++

```
simEncoder.SetCount(100);
encoder.GetCount(); // 100
simEncoder.GetCount(); // 100
```

Registering Callbacks

In addition to the getters and setters, each field also has a registerXxxCallback() function that registers a callback to be run whenever the field value changes and returns a CallbackStore object. The callbacks accept a string parameter of the name of the field and a HALValue object containing the new value. Before retrieving values from a HALValue, check the type of value contained. Possible types are HALValue.kBoolean/HAL_BOOL, HALValue.kDouble/HAL_DOUBLE, HALValue.kEnum/HAL_ENUM, HALValue.kInt/HAL_INT, HALValue.kLong/HAL_LONG.

In Java, call close() on the CallbackStore object to cancel the callback. Keep a reference to the object so it doesn't get garbage-collected - otherwise the callback will be canceled by GC. To provide arbitrary data to the callback, capture it in the lambda or use a method reference.

In C++, save the CallbackStore object in the right scope - the callback will be canceled when the object goes out of scope and is destroyed. Arbitrary data can be passed to the callbacks via the param parameter.

Uyarı: Attempting to retrieve a value of a type from a HALValue containing a different type is undefined behavior.

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 Simulating Other Devices - The SimDeviceSim Class

Not: Vendors might implement their connection to the SimDevice API slightly different than described here. They might also provide a simulation class specific for their device class. See your vendor's documentation for more information as to what they support and how.

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*.

The SimDeviceSim object is created using a string key identical to the key the vendor used to construct the underlying SimDevice in their device class. This key is the one that the device shows up with in the *Other Devices* tab, and is typically of the form Prefix:Device Name[index]. If the key contains ports/index/channel numbers, they can be passed as separate arguments to the SimDeviceSim constructor. The key contains a prefix that is hidden by default in the SimGUI, it can be shown by selecting the *Show prefix* option. Not including this prefix in the key passed to SimDeviceSim will not match the device!

JAVA

```
SimDeviceSim device = new SimDeviceSim(deviceKey, index);
```

C++

```
frc::sim::SimDeviceSim device{deviceKey, index};
```

Once we have the `SimDeviceSim`, we can get `SimValue` objects representing the device's fields. Type-specific `SimDouble`, `SimInt`, `SimLong`, `SimBoolean`, and `SimEnum` subclasses also exist, and should be used instead of the type-unsafe `SimValue` class. These are constructed from the `SimDeviceSim` using a string key identical to the one the vendor used to define the field. This key is the one the field appears as in the `SimGUI`. Attempting to retrieve a `SimValue` object outside of simulation or when either the device or field keys are unmatched will return null - this can cause `NullPointerException` in Java or undefined behavior in C++.

JAVA

```
SimDouble field = device.getDouble(fieldKey);
field.get();
field.set(value);
```

C++

```
hal::SimDouble field = device.GetDouble(fieldKey);
field.Get();
field.Set(value);
```

22.5 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.

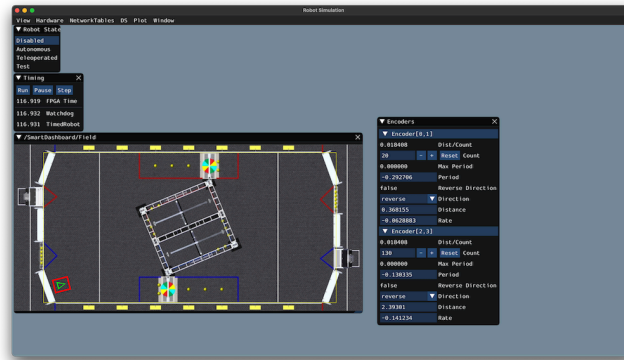
Both of these examples are also available in the VS Code *New Project* window.

22.5.1 Drivetrain Simulation Overview

Not: 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.

Not: 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.

Not: 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};
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
// 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.

Not: 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 object like 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.

Not: 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.

Not: 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.

    // 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.

Önemli: 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.

Not: 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.

Not: 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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
// 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.

Not: 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.

Not: 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.  
    );
```

Not: You can use the `KitbotMotor`, `KitbotGearing`, and `KitbotWheelSize` enum (Java) / struct (C++) to get commonly used configurations of the Kit of Parts Chassis.

Önemli: 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.

Not: 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.setInput(m_leftMotor.get() * RobotController.getInputVoltage(),
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

        m_rightMotor.get() * RobotController.getInputVoltage());

    // Advance the model by 20 ms. Note that if you are running this
    // subsystem in a separate thread or have changed the nominal timestep
    // of TimedRobot, this value needs to match it.
    m_driveSim.update(0.02);

    // Update all of our sensors.
    m_leftEncoderSim.setDistance(m_driveSim.getLeftPositionMeters());
    m_leftEncoderSim.setRate(m_driveSim.getLeftVelocityMetersPerSecond());
    m_rightEncoderSim.setDistance(m_driveSim.getRightPositionMeters());
    m_rightEncoderSim.setRate(m_driveSim.getRightVelocityMetersPerSecond());
    m_gyroSim.setAngle(-m_driveSim.getHeading().getDegrees());
}

```

C++

```

frc::PWMSparkMax m_leftMotor{0};
frc::PWMSparkMax m_rightMotor{1};

Drivetrain() {
    ...
    m_leftEncoder.SetDistancePerPulse(2 * std::numbers::pi * kWheelRadius /
↪ kEncoderResolution);
    m_rightEncoder.SetDistancePerPulse(2 * std::numbers::pi * kWheelRadius /
↪ kEncoderResolution);
}

void SimulationPeriodic() {
    // Set the inputs to the system. Note that we need to convert
    // the [-1, 1] PWM signal to voltage by multiplying it by the
    // robot controller voltage.
    m_driveSim.SetInputs(
        m_leftMotor.get() * units::volt_t(frc::RobotController::GetInputVoltage()),
        m_rightMotor.get() * units::volt_t(frc::RobotController::GetInputVoltage()));

    // Advance the model by 20 ms. Note that if you are running this
    // subsystem in a separate thread or have changed the nominal timestep
    // of TimedRobot, this value needs to match it.
    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());
}

```

Önemli: 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.

Önemli: 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.

Not: 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);
}
```

Not: 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());
}
```

Önemli: 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.

Not: 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 Birim Test Etme

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 Test Tablosu Kodu Yazma

Not: This example can be easily adapted to the command-based paradigm by having Intake inherit from SubsystemBase.

Our subsystem will be an Infinite Recharge intake mechanism containing a piston and a motor: the piston deploys/retracts the intake, and the motor will pull the Power Cells inside. We don’t want the motor to run if the intake mechanism isn’t deployed because it won’t do anything.

To provide a “clean slate” for each test, we need to have a function to destroy the object and free all hardware allocations. In Java, this is done by implementing the `AutoCloseable` interface and its `.close()` method, destroying each member object by calling the member’s `.close()` method - an object without a `.close()` method probably doesn’t need to be closed. In C++, the default destructor will be called automatically when the object goes out of scope and will call destructors of member objects.

Not: Vendors might not support resource closing identically to the way shown here. See your vendor’s documentation for more information as to what they support and how.

Java

```

import edu.wpi.first.wpilibj.DoubleSolenoid;
import edu.wpi.first.wpilibj.PneumaticsModuleType;
import edu.wpi.first.wpilibj.examples.unittest.Constants.IntakeConstants;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;

public class Intake implements AutoCloseable {
    private final PWMSparkMax m_motor;
    private final DoubleSolenoid m_piston;

    public Intake() {
        m_motor = new PWMSparkMax(IntakeConstants.kMotorPort);
        m_piston =
            new DoubleSolenoid(
                PneumaticsModuleType.CTREPCM,
                IntakeConstants.kPistonFwdChannel,
                IntakeConstants.kPistonRevChannel);
    }

    public void deploy() {
        m_piston.set(DoubleSolenoid.Value.kForward);
    }

    public void retract() {
        m_piston.set(DoubleSolenoid.Value.kReverse);
        m_motor.set(0); // turn off the motor
    }

    public void activate(double speed) {
        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)

```
#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 Writing Tests

Önemli: Tests are placed inside the test source set: `/src/test/java/` and `/src/test/cpp/` for Java and C++ tests, respectively. Files outside that source root do not have access to the test framework - this will fail compilation due to unresolved references.

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.

Each test method should contain at least one *assertion* (`assert*()` in Java or `EXPECT_*()` in C++). These assertions verify a condition at runtime and fail the test if the condition isn't met. If there is more than one assertion in a test method, the first failed assertion will crash the test - execution won't reach the later assertions.

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 static org.junit.jupiter.api.Assertions.*`. In Google Test (C++), assertions are macros from the `<gtest/gtest.h>` header.

Not: Comparison of floating-point values isn't accurate, so comparing them should be done with an acceptable error parameter (DELTA).

Java

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import edu.wpi.first.hal.HAL;
import edu.wpi.first.wpilibj.DoubleSolenoid;
import edu.wpi.first.wpilibj.PneumaticsModuleType;
import edu.wpi.first.wpilibj.examples.unittest.Constants.IntakeConstants;
import edu.wpi.first.wpilibj.simulation.DoubleSolenoidSim;
import edu.wpi.first.wpilibj.simulation.PWMSim;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class IntakeTest {
    static final double DELTA = 1e-2; // acceptable deviation range
    Intake m_intake;
    PWMSim m_simMotor;
    DoubleSolenoidSim m_simPiston;

    @BeforeEach // this method will run before each test
```

(sonraki sayfaya devam)


```

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
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());
}

```

For more advanced usage of JUnit and Google Test, see the framework docs.

22.6.3 Çalışma Testleri

Not: Tests will always be run in simulation on your desktop. For prerequisites and more info, see [the simulation introduction](#).

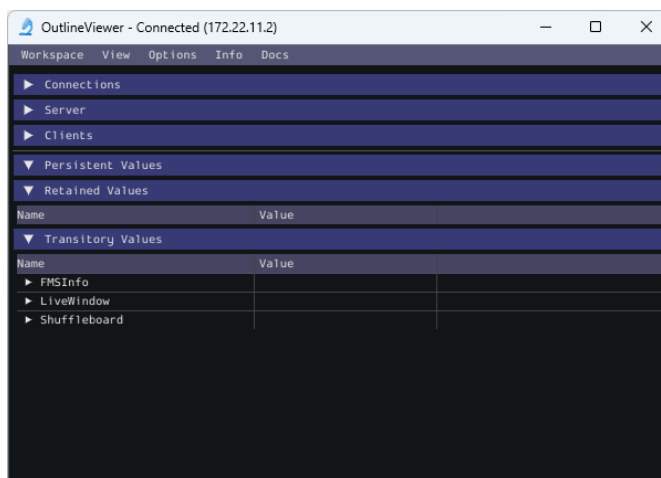
For Java tests to run, make sure that your `build.gradle` file contains the following block:

```
74 test {  
75     useJUnitPlatform()  
76     systemProperty 'junit.jupiter.extensions.autodetection.enabled', 'true'  
77 }
```

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.

By default, Gradle runs the tests whenever robot code is built, including deploys. This will increase deploy time, and failing tests will cause the build and deploy to fail. To prevent this from happening, you can use *Change Skip Tests On Deploy Setting* from the Command Palette to configure whether to run tests when deploying.

OutlineViewer

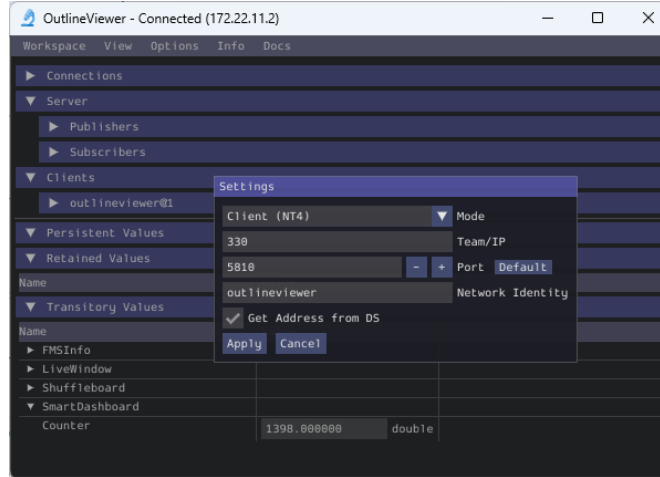


OutlineViewer is a utility used to view, modify and add to the contents of the NetworkTables for debugging purposes. It displays all key value pairs currently in the NetworkTables and can be used to modify the value of existing keys or add new keys to the table. OutlineViewer is included in the Java 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 select *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](#).

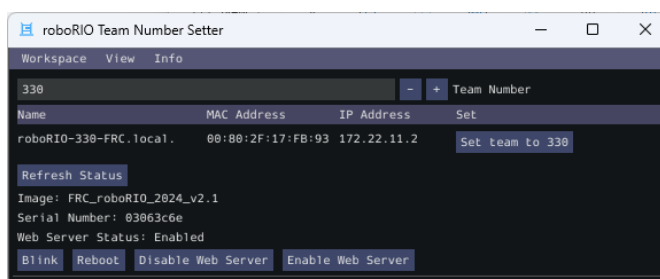
Not: You can use `localhost` instead of a team number to point OutlineViewer at a simulated robot, Romi or XRP.



Ağ Tablolarına ek anahtar/değer çiftleri eklemek için, bir konuma sağ tıklayın ve ilgili veri tipini seçin.

Not: LabVIEW ekipleri, OutlineViewer ile aynı işlevi gerçekleştirmek için LabVIEW Kontrol Panelinin Değişkenler sekmesini kullanabilir.

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 debugging 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.

25.1 Vision (Görüntü) 'a Giriş

25.1.1 Vision nedir?

FRC'de vizyon ® ekiplerin hem otonom hem de teleoperasyon dönemlerinde gol atmasına ve sürüş yapmasına yardımcı olmak için robota bağlı bir kamera kullanır.

Görme Yöntemleri

FRC'de çoğu ekibin görmek için kullandığı iki ana yöntem vardır.

Yayın Akışı

Bu yöntem, sürücünün ve manipulatörün robotun bakış açısından görsel bilgileri alabilmesi için kameranın Sürücü İstasyonuna aktarılmasını içerir. Bu yöntem basittir ve uygulanması çok az zaman alır, bu da görme işleme özelliklerine ihtiyacınız yoksa iyi bir seçenektir.

- *Streaming using the roboRIO*

İşleme

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.

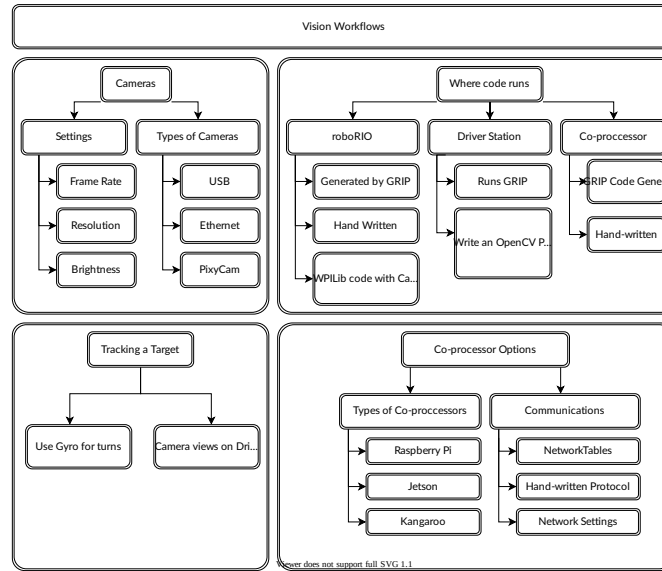
- Raspberry Pi ile Görüntü İşleme *Vision Processing with Raspberry Pi*.
- :ref:`roboRIO <docs/software/vision-processing/roborio/using-the-cameraserver-on-the-roborio:Advanced Camera Server Program> ile Görüntü İşleme`

Görsel işleme için bir yardımcı işlemci kullanmanın artıları ve eksileri hakkında ek bilgi için sonraki sayfaya bakın docs / software / vision-processing / Introduction / Strategies-for-vision-programlama: Strategies for Vision Programming.

25.1.2 Görüntü Programlama Stratejileri

Bilgisayarla görmeyi kullanmak, robotunuzun sahadaki unsurlara duyarlı olmasını ve çok daha otonom olmasını sağlamanın harika bir yoludur. Genellikle FRC [reg] oyunlar, topları veya diğer oyun parçalarını kendi kendine hedefe atmak veya sahadaki yerlere gitmek için bonus puanlar vardır. Bilgisayarla görme, bu sorunların çoğunu çözmenin harika bir yoludur. Ve eğer meydan okumayı gerçekleştirebilecek özerk bir kodunuz varsa, o zaman insan sürücülere yardımcı olmak için teleop döneminde de kullanılabilir.

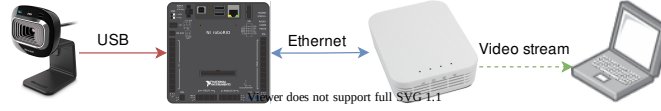
Görüntü işleme için bileşenleri ve görsel denetim programının nerede çalışması gerektiğini seçmek için bir çok seçenek vardır. WPILib ve ilgili araçlar bir dizi seçeneği destekler ve ekiplere ne yapacaklarına karar vermeleri için büyük bir esneklik sağlar. Bu makale, mevcut birçok seçenek ve değiş tokuş hakkında size fikir vermeye çalışacaktır.



OpenCV Computer Vision Kitaplığı

**** OpenCV ****, akademi ve endüstri genelinde yaygın olarak kullanılan açık kaynaklı bir bilgisayar görüntü kitaplığıdır. GPU hızlandırmalı işleme sağlayan donanım üreticilerinden destek alır, C ++, Java ve Python dahil olmak üzere bir dizi dil için bağlantıları vardır. Ayrıca birçok web sitesi, kitap, video ve eğitim kursları ile iyi bir şekilde belgelenmiştir, bu nedenle nasıl kullanılacağını öğrenmenize yardımcı olacak birçok kaynak vardır. WPILib'in C ++ ve Java sürümleri OpenCV kitaplıklarını içerir, kitaplıkta video yakalama, işleme ve görüntüleme desteği ve vizyon algoritmalarınızı oluşturmanıza yardımcı olacak araçlar vardır. OpenCV hakkında daha fazla bilgi için bkz. <https://opencv.org>.

RoboRIO'da Görüntü Kodu

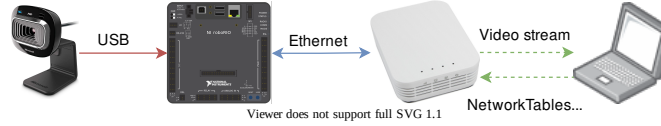


Vision code can be embedded into the main robot program on the roboRIO. Building and running the vision code is straightforward because it is built and deployed along with the robot program. The vision code can be written in C++, Java, or Python. The disadvantage of this approach is that having vision 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.

CameraServer sınıfı tarafından sağlanan işlevleri kullanarak, video akışı Shuffleboard gibi gösterge tablolarına gönderilebilir, böylece operatörler kameranın gördüklerini görebilir. Ek olarak, OpenCV komutları bilgi görüntülere eklenebilir, böylece hedefler veya diğer ilginç nesneler gösterge tablosu görünümünde tanımlanabilir.

DS Bilgisayarda Görüntü Kodu

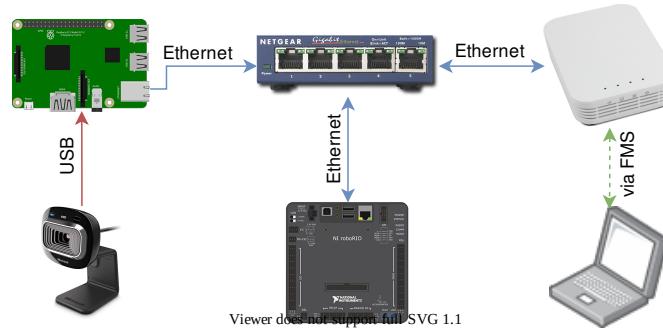


When vision code is running on the DS computer, the video is streamed back to the Driver Station laptop for processing. Even the older Classmate laptops are substantially faster at vision processing than the roboRIO. You can write your own vision program using a language of your choosing. Python makes a good choice since there is a native NetworkTables implementation and the OpenCV bindings are very good.

Görüntüler işlendikten sonra, hedef konum, mesafe veya ihtiyacınız olan herhangi bir şey gibi anahtar değerler, NetworkTables ile robota geri gönderilebilir. Görüntülerin dizüstü bilgisayara gönderilmesi gerektiğinden gecikme eklendiğinden, bu yaklaşım genellikle daha yüksek gecikmeye sahiptir. Bant genişliği sınırlamaları ayrıca işleme için kullanılan görüntülerin maksimum çözünürlüğünü ve FPS'sini de sınırlar.

The video stream can be displayed on Shuffleboard or SmartDashboard.

Yardımcı İşlemcide Görüntü Kodu



Raspberry Pi gibi yardımcı işlemciler, görüntü kodunu desteklemek için idealdir (bkz: ref: docs / software / vision-processing / wpilibpi / using-the-raspberry-pi-for-frc: Raspberry Pi for FRC). Avantajı, tam hızda çalışabilmeleri ve robot programına müdahale etmemeleridir. Bu durumda, kamera muhtemelen yardımcı işlemciye veya (Ethernet kameraları olması durumunda) robot üzerindeki bir Ethernet anahtarına bağlıdır. Program herhangi bir dilde yazılabilir; Python, OpenCV ve NetworkTables'a olan basit bağlantıları nedeniyle iyi bir seçimdir. Bazı ekipler, en yüksek hız ve en yüksek çözünürlük için Nvidia Jetson gibi yüksek performanslı görüntü yardımcı işlemcileri kullandı, ancak bu yaklaşım genellikle gelişmiş Linux ve programlama bilgisi gerektiriyor.

Bu yaklaşım, biraz daha fazla programlama uzmanlığı ve az miktarda ek ağırlık gerektirir, ancak aksi takdirde, yardımcı işlemciler roboRIO'dan çok daha hızlı olduğundan ve görüntü işleme ile gerçekleştirilebildiğinden, diğer iki yaklaşıma kıyasla her iki yerin en iyisini minimum gecikme veya bant genişliği kullanımı ile getirir.

Veriler, yardımcı işlemciye görüntü programından robota NetworkTables veya bir ağ veya seri bağlantı üzerinden özel bir protokol kullanılarak gönderilebilir.

Kamera Seçenekleri

WPILib tarafından desteklenen bir dizi kamera seçeneği vardır. Kameraların çalışmayı etkileyen bir dizi parametresi vardır; örneğin, kare hızı ve görüntü çözünürlüğü alınan görüntülerin kalitesini etkiler, ancak çok yüksek etkili işleme süresi ayarlandığında ve sürücü istasyonuna gönderilirse, sahadaki mevcut bant genişliğini aşabilir.

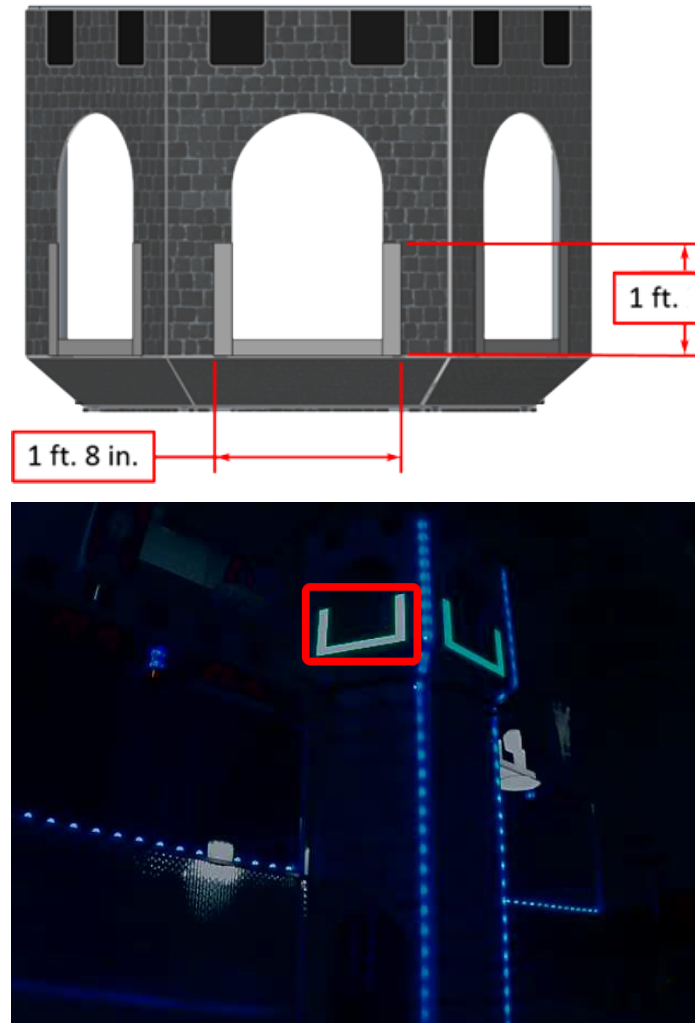
C++ ve Java'daki CameraServer sınıfı, robota bağlı kameralarla arayüz oluşturmak için kullanılır. Bir Kaynak nesnesi aracılığıyla yerel işleme için çerçeveleri alır ve akışı orada görüntülemek veya işlemek için sürücü istasyonunuza gönderir.

25.1.3 Hedef Bilgisi ve Geri Yansımaya

Birçok FRC® oyunlarda, görüş işlemeye yardımcı olmak için alan öğelerine tutturulmuş retroreflektif bant bulunur. Bu belge 2016 FRC oyunundaki Vizyon Hedeflerini ve hedefleri oluşturan malzemenin görsel özelliklerini açıklamaktadır.

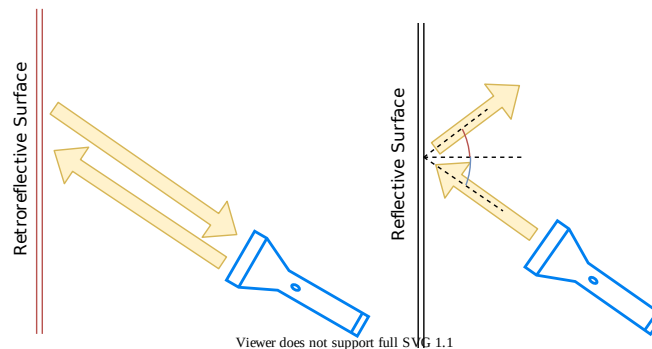
Not: Tüm saha bileşenlerinin resmi boyutları ve çizimleri için lütfen Resmi Saha Çizimlerine-Official Field Drawings bakın.

Hedefler



Her bir 2016 vizyon hedefi, 2" genişliğinde retroreflektif malzemeden (3M 8830 Silver Marking Film) yapılmış 1 '8" genişliğinde, 1' yüksekliğinde U şeklindeki oluşur. Hedefler, her yüksek hedefin hemen dibine bitişik konumlandırılmıştır. Gerekli olduğu gibi aydınlatıldığında, retroreflektif bant parlak ve/veya renge doymuş bir işaret üretir.

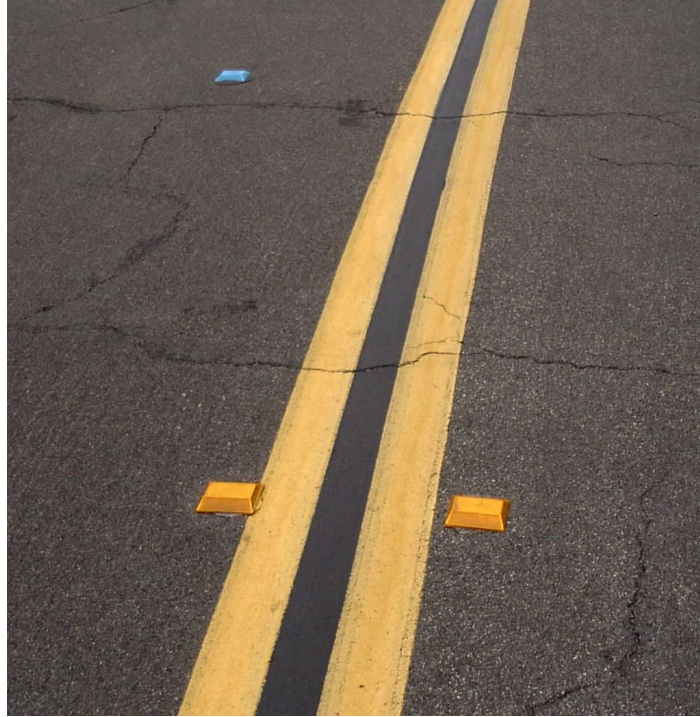
Retroreflektivite ve Reflektiflik



Yüksek derecede yansıtıcı malzemeler genellikle, ışığın ek bir açıyla “yansıtacağı” şekilde aynalanır. Yukarıda solda gösterildiği gibi mavi ve kırmızı açılarının toplamı 180 derecedir. Buna eşdeğer bir açıklama, ışığın yüzeye dik olarak çizilen yeşil çizgiyi normal yüzey etrafında yansıtmasıdır. Yüzeye doğrultulan bir ışığın, yalnızca mavi açı ~ 90 derece olduğunda ışık kaynağına döneceğine dikkat edin.

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.

Retroreflection örnekleri





Bu malzeme, genellikle yol işaretlerinin, bisikletlerin ve yayaların gece görünürlüğünü artırmak için kullanıldığından, nispeten tanıdık olmalıdır.

Başlangıçta retro-reflection karanlık güvenliği için kullanışlı bir özellik gibi görünmeyebilir, ancak yukarıda gösterildiği gibi ışık ve göz birbirine yakın olduğunda, yansıyan ışık göze geri döner ve malzeme büyük mesafelerde bile parlak bir şekilde parlar. Sürücünün gözleri ile araç farları arasındaki küçük açı nedeniyle, geri yansıtıcı malzemeler gece sürüşü sırasında uzaktaki nesnelerin görünürlüğünü büyük ölçüde artırabilir.

Gösteri

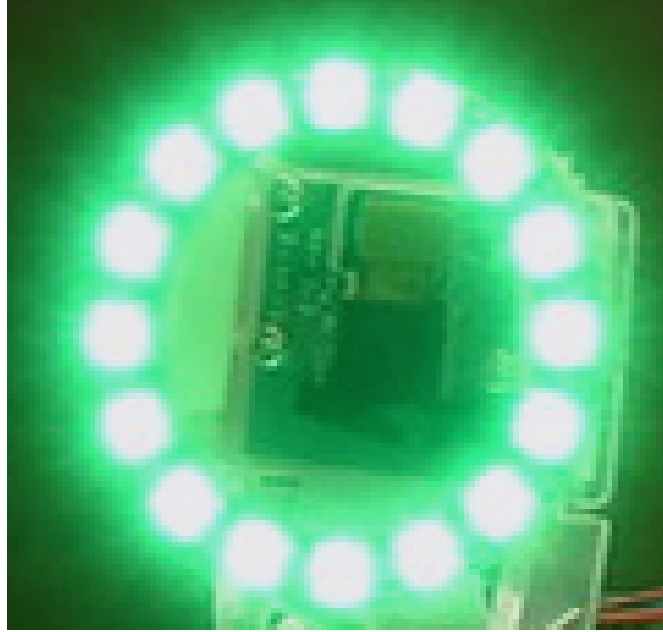
Retro-reflective malzeme özelliklerini daha fazla keşfetmek için:

1. Malzemenin bir parçasını duvara veya dikey bir yüzeye yerleştirin
2. 10-20 fit uzakta durun ve malzemeye küçük bir el feneri tutun.
3. Göbek seviyenizde tutulan ışıkla başlayın ve gözünüzün arasına gelene kadar yavaşça kaldırın. Işık gözlerinize yaklaştıkça, geri dönen ışığın yoğunluğu hızla artacaktır.
4. Odadaki diğer yerlere gidip tekrarlayarak açığı değiştirin. Parlak yansıma geniş bir bakış açısı aralığında gerçekleşmelidir, ancak ışık kaynağından göze olan açı önemlidir ve oldukça küçük olmalıdır.

Farklı ışık kaynakları ile deneyler yapın. Malzeme, beyaz boyadan yüzlerce kat daha yansıtıcıdır; bu nedenle loş ışık kaynakları iyi çalışacaktır. Örneğin, kırmızı bir bisiklet güvenlik ışığı, ışık kaynağının renginin yansıyan ışığın rengini belirlediğini gösterecektir. Mümkünse, her biri kendi ışık kaynağına sahip birkaç ekip üyesini farklı konumlara yerleştirin. Bu, efektlerin büyük ölçüde bağımsız olduğunu ve malzemenin aynı anda çeşitli ekip üyelerine farklı renklerde görünebileceğini gösterecektir. Bu aynı zamanda malzemenin çevresel aydınlatmaya büyük ölçüde bağlı olduğunu da gösterir. İzleyiciye dönen ışık neredeyse tamamen kontrol ettikleri bir ışık kaynağı tarafından veya doğrudan arkalarındaki bir ışık tarafından belirlenir.

El fenerini kullanarak, halihazırda çevrenizde bulunan diğer reflektörlü eşyaları tanımlayın: giysiler, sırt çantaları, ayakkabılar vb.

Aydınlatma



Retro-reflective bandın, kendisine bir ışık kaynağı yönlendirilmedikçe parlamayacağını ve ışık kaynağının kamera merceğinin veya gözlemcinin gözlerinin çok yakınından geçmesi gerektiğini gördük. Bunu başarmanın birkaç yolu olsa da, araştırılması gereken çok yararlı bir ışık kaynağı türü, yukarıda gösterilen halka flaş veya halka ışıktır. Işık kaynağını doğrudan kamera merceğinin üzerine veya çevresine yerleştirir ve çok eşit bir aydınlatma sağlar. Parlak çıktıları ve küçük boyutları nedeniyle, LED'ler özellikle bu tür bir cihazı oluşturmak için kullanışlıdır.

Yukarıda gösterildiği gibi, ucuz dairesel LED düzenlemeleri çeşitli renk ve boyutlarda mevcuttur ve kameralara takılması kolaydır ve hatta bazıları bir Raspberry Pi'den kapatılabilir. Dağınık eşit aydınlatma için tasarlanmamış olsalar da, retro-yansıtıcı bandın parlamasına neden olmak için oldukça iyi çalışırlar. FIRST Choice aracılığıyla küçük bir yeşil LED halka mevcuttur. Diğer benzer LED halkaları, SuperBrightLED'ler gibi tedarikçilerden temin edilebilir.

Örnek Görüntüler

Örnek görüntüler, her dil için kod örnekleriyle birlikte bulunur (LabVIEW ile paketlenmiştir ve C++ / Java örnekleriyle aynı konumda ayrı bir ZIP içinde).

25.1.4 Hedeflerin Belirlenmesi ve İşlenmesi

Bir görüntü yakalandıktan sonraki adım, görüntüdeki Vision Target(s)-Görme Hedeflerini belirlemektir. Bu belge, 2016 hedeflerini belirlemek için tek bir yaklaşımdan geçecektir. Bu bölümde kullanılan görüntülerin kamera ile kasıtlı olarak düşük pozlama ayarlı olarak çekildiğini ve yanan hedefler haricinde çok karanlık görüntüler oluşturduğunu unutmayın, ayrıntılar için Kamera Ayarları bölümüne bakın.

Gerçek görüntü

Aşağıda gösterilen görüntü, burada açıklanan örnek için başlangıç görüntüsüdür. Görüntü, FIRST® konumunda bulunan yeşil halka ışığı kullanılarak alınmıştır Seçim, farklı boyutta ek bir halka ışıyla birleştirildi. Görüntü kodu örnekleriyle birlikte ek örnek görüntüler sağlanır.



HSL / HSV nedir?

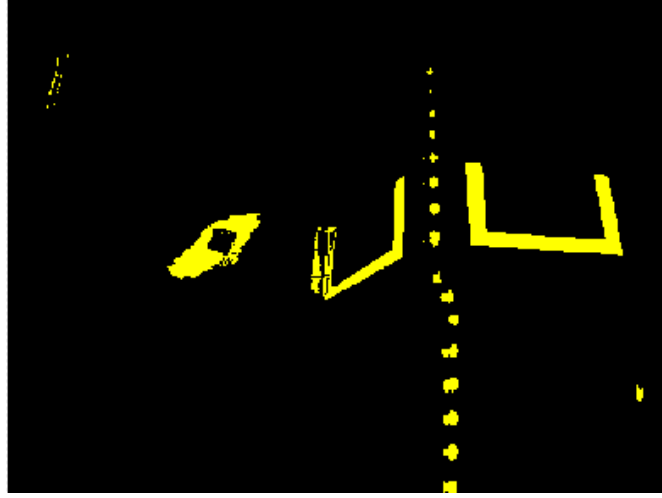
Ton veya rengin tonu genellikle sanatçının renk çarkında görülür ve gökkuşağı Kırmızı, Turuncu, Sarı, Yeşil, Mavi, Çivit Mavisi ve Menekşe renklerini içerir. Renk tonu, tekerlek üzerindeki bir radyal açı kullanılarak belirlenir, ancak görüntülemeye daire tipik olarak, sıfırdan kırmızıdan başlayarak, gökkuşağı boyunca dönerek ve üst uçta tekrar kırmızıya sararak yalnızca 256 birim içerir. Bir rengin doygunluğu, renk miktarını veya ton renginin gri gölgeye oranını belirtir. Daha yüksek oran, daha renkli, daha az gri demektir. Sıfır doygunluğun tonu yoktur ve tamamen gridir. Parlaklık veya Değer, tonun harmanlandığı gri tonunu gösterir. Siyah 0 ve beyaz 255'tir.

Örnek kod, hedefin rengini belirtmek için HSV renk uzayını kullanır. Birincil neden, Hedeflerin parlaklığını, görüntünün geri kalanına göre, Değer (HSV) veya Parlaklık (HSL) bileşenini kullanarak bir filtreleme kriteri olarak kullanmaya kolayca izin vermesidir. HSV renk sistemini kullanmanın bir başka nedeni, örnekte kullanılan eşikleme işleminin HSV renk uzayında yapıldığında roboRIO üzerinde daha verimli çalışmasıdır.

Maskeleme

Bu ilk adımda, aşağıda sarı renkte gösterilen bir ikili maske oluşturmak için piksel değerleri sabit renk veya parlaklık değerleriyle karşılaştırılır. Bu tek adım, hedefin geriye dönük yansıtıcı bandının parçası olmayan piksellerin çoğunu ortadan kaldırır. Renk tabanlı maskeleme, rengin nispeten doygun, parlak ve tutarlı olması koşuluyla iyi çalışır. Renk eşitsizlikleri genellikle HSL (Ton, Doygunluk ve Parlaklık) veya HSV (Ton, Doygunluk ve Değer) renk alanı kullanılarak belirtildiğinde RGB (Kırmızı, Yeşil ve Mavi) alanından daha doğrudur. Bu, özellikle renk aralığı bir veya daha fazla boyutta oldukça geniş olduğunda geçerlidir.

Hedefe ek olarak, görüntünün diğer parlak kısımlarının da (üstten ışık ve kule aydınlatması) maskeleme adımıyla yakalandığına dikkat edin.



Parçacık Analizi

Maskeleme işleminden sonra, alanı, sınırlayıcı dikdörtgeni ve parçacıklar için eşdeğer dikdörtgeni incelemek için bir parçacık raporu işlemi kullanılır. Bunlar, en dikdörtgen olan şekilleri seçmeye yardımcı olmak için birkaç puanlanmış terimi hesaplamak için kullanılır. Aşağıda açıklanan her test bir puan (0-100) oluşturur ve bu daha sonra parçacığın hedef olup olmadığına karar vermek için önceden tanımlanmış puan limitleriyle karşılaştırılır.

Kapsama alanı

Alan puanı, parçacığın çevresine çizilen sınırlayıcı kutunun alanıyla karşılaştırılarak parçacık alanı karşılaştırılarak hesaplanır. Arkadan yansıtıcı şeritlerin alanı 80 inç karedir (~ 516 : matematik: " cm^2 "). Hedefi içeren dikdörtgenin alanı 240 inç karedir ($\sim 0,15$: matematik: " m^2 "). Bu, alan ve sınırlayıcı kutu alanı arasındaki ideal oranın $1/3$ olduğu anlamına gelir. $1/3$ 'e yakın alan oranları 100'e yakın bir puan üretecektir, oran $1/3$ 'ten farklılaştıkça puan 0'a yaklaşacaktır.

En Boy Oranı

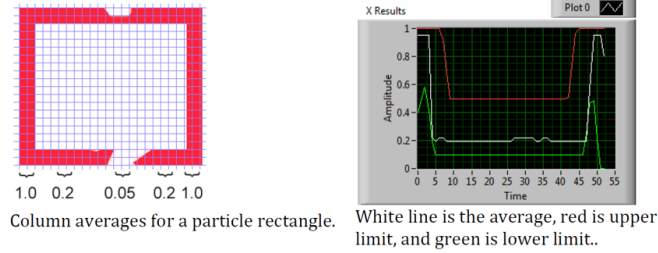
En boy oranı puanı (Partikül Genişliği / Partikül Yüksekliği) temel alınarak belirlenir. Parçacığın genişliği ve yüksekliği “eşdeğer dikdörtgen” adı verilen bir şey kullanılarak belirlenir. Eşdeğer dikdörtgen, kenar uzunlukları olan dikdörtgendir : x ve y burada: $2x+2y$, parçacık çevresine ve: $x \cdot y$, parçacık alanına eşittir. Eşdeğer dikdörtgen, en boy oranı hesaplamasında kullanılır, çünkü dikdörtgenin eğrilmesinden sınırlayıcı kutuyu kullanmaktan daha az etkilenir. En boy oranı için sınırlayıcı kutu dikdörtgeni kullanıldığında, dikdörtgen eğriltildikçe yükseklik artar ve genişlik azalır.

Hedef 1,6 oran için 20 “(508 mm) genişliğinde ve 12” (304,8 mm) yüksekliğindedir. Tespit edilen en boy oranı bu ideal oran ile karşılaştırılır. En boy oranı puanı, oran hedef oranla eşleştiğinde ve oran aşağı veya yukarı değiştiğinde doğrusal olarak düştüğünde 100’e geri dönecek şekilde normalleştirilir.

An

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 Profilleri



Kenar puanı, parçacığın hem X hem de Y yönlerinde uygun profille eşleşip eşleşmediğini açıklar. Gösterildiği gibi, orijinal görüntüden çıkarılan sınırlayıcı kutu boyunca satır ve sütun ortalamaları kullanılarak ve bunu bir profil maskesi ile karşılaştırarak hesaplanır. Puan, üst ve alt sınır değerleri arasındaki satır veya sütun ortalamaları içindeki değerlerin sayısına bağlı olarak 0 ile 100 arasında değişir.

Ölçümler

Bir parçacık hedef olarak değerlendirilebilecek kadar iyi puan alırsa, konum ve mesafe gibi bazı gerçek dünya ölçümlerini hesaplamak mantıklıdır. Örnek kod bu temel ölçümleri içerir, bu yüzden onu daha iyi anlamak için ilgili matematiğe bakalım.

Durum

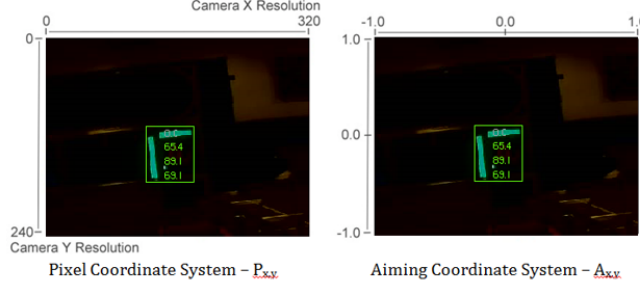
Hedef konum, hem parçacık hem de sınırlayıcı kutu tarafından iyi tanımlanmıştır, ancak tüm koordinatlar, ekranın sol üst köşesinde 0,0 ve kamera çözünürlüğü tarafından belirlenen sağ ve alt kenarlar olmak üzere piksel cinsindendir. Bu, piksel matematiği için kullanışlı bir sistemdir, ancak bir robotu sürmek için neredeyse hiç kullanışlı değildir; Öyleyse daha yararlı olabilecek bir şeyle değiştirelim.

Bir noktayı piksel sisteminden nişan alma sistemine dönüştürmek için aşağıda gösterilen formülü kullanabiliriz.

The resulting coordinates are close to what you may want, but the Y axis is inverted. This could be corrected by multiplying the point by [1,-1] (Note: this is not done in the sample code). This coordinate system is useful because it has a centered origin and the scale is similar to joystick outputs and Drive inputs.

$$A_{x,y} = \left(P_{x,y} - \frac{\text{resolution}_{x,y}}{2} \right) / \frac{\text{resolution}_{x,y}}{2}$$

$$A_{x,y} = (P_{x,y} - \frac{\text{resolution}_{x,y}}{2}) / \frac{\text{resolution}_{x,y}}{2}$$



Görüş alanı

Hedeften mesafenizi, sapmanızı ve eğiminizi belirlemek için bilinen sabitleri ve hedefin koordinat düzlemindeki konumunu kullanabilirsiniz. Ancak, bunları hesaplamak için FOV'nuzu (görüş alanınızı) belirlemelisiniz. Dikey görüş alanını ampirik olarak belirlemek için, kameranızı düz bir yüzeyden belirli bir mesafe uzağa ayarlayın ve en üstteki ve en alttaki piksel sırası arasındaki mesafeyi ölçün.

$$\frac{1}{2}FOV_{vertical} = \tan \left(\frac{\frac{1}{2}distance_y}{distance_z} \right)$$

Yatay FOV'u aynı yöntemi kullanarak, ancak ilk ve son piksel sütunu arasındaki mesafeyi kullanarak bulabilirsiniz.

Pitch and Yaw

FOV'lerinizi ve hedef koordinat sistemindeki hedefinizin konumunu öğrendikten sonra, robotunuza göre hedefin eğimini ve sapmasını bulmak basittir.

$$pitch = \frac{A_y}{2} FOV_{vertical}$$

$$yaw = \frac{A_x}{2} FOV_{horizontal}$$

Mesafe

Hedefiniz robotunuzdan önemli ölçüde farklı bir yükseklikte ise, kameranız ile kamera arasındaki mesafeyi hesaplamak için hedefin ve kameranızın fiziksel yüksekliği ve kameranızın monte edildiği açı gibi bilinen sabitleri kullanabilirsiniz. hedef.

$$distance = \frac{height_{target} - height_{camera}}{\tan(angle_{camera} + pitch)}$$

Diğer bir seçenek, alan-mesafe için bir arama tablosu oluşturmak veya alan ve mesafenin ters varyasyon sabitini tahmin etmektir. Ancak bu yöntem daha az doğrudur.

Not: Yukarıdaki açı ve mesafe tahmin yöntemlerinden en iyi sonuçları elde etmek için, kalibrasyon matrisini kullanarak hedefin piksellerini yeniden projekte ederek doğruluğu etkileyebilecek tüm bozulmalardan kurtulmak için OpenCV kullanarak kameranızı kalibre edebilirsiniz.

25.1.5 Videoyu Okuyun ve İşleyin: CameraServer Sınıfı

Kavramlar

Genellikle FRC | reg | içinde kullanılan kameralar (Axis kamera gibi ticari USB ve Ethernet kameraları) nispeten sınırlı çalışma modları sunar. Genel olarak, tek bir çözünürlük ve kare hızında yalnızca tek bir görüntü çıkışı (tipik olarak JPG gibi bir RGB sıkıştırılmış biçimde) sağlarlar. Bir seferde yalnızca bir uygulama kameraya erişebileceğinden, USB kameralar özellikle sınırlıdır.

CameraServer birden fazla kamerayı destekler. Bir kameranın bağlantısı kesildiğinde otomatik olarak yeniden bağlanma gibi ayrıntıları yönetir ve ayrıca kameradaki görüntüleri birden çok "clients-istemciye" sunar (örneğin, hem robot kodunuz hem de kontrol paneliniz kameralara aynı anda bağlanabilir).

Kamera Adları

CameraServer'daki her kamera benzersiz bir şekilde adlandırılmalıdır. Bu aynı zamanda Pano'daki kamera için görünen addır. CameraServer'ın ``startAutomaticCapture()`` ve ``addAxisCamera()`` işlevlerinin bazı varyantları, kamerayı otomatik olarak adlandırır (örneğin, "USB Kamera 0" veya "Eksen Kamera") veya kameraya daha açıklayıcı bir ad verebilirsiniz (örneğin "Giriş Kamı"). Tek şart, her kameranın benzersiz bir isme sahip olmasıdır.

USB Kamera Notları

CPU kullanımı

CameraServer, yalnızca gerektiğinde sıkıştırma ve açma işlemlerini gerçekleştirerek ve hiçbir istemci bağlı değilken akışı otomatik olarak devre dışı bırakarak CPU kullanımını en aza indirecek şekilde tasarlanmıştır.

CPU kullanımını en aza indirmek için, gösterge paneli çözünürlüğü kamera ile aynı çözünürlüğe ayarlanmalıdır; bu, CameraServer'ın görüntüyü açıp yeniden sıkıştırmamasına izin verir; bunun yerine, kameradan alınan JPEG görüntüsünü doğrudan kontrol paneline iletebilir. Kontrol panelindeki çözünürlüğün değiştirilmesinin kamera çözünürlüğünü * değiştirmediğini * unutmamak önemlidir; kamera çözünürlüğünü değiştirmek, kamera nesnesi üzerinde ``setResolution()`` çağrısı yapılarak yapılabilir.

USB Bant Genişliği

RoboRIO, USB arayüzleri üzerinden bir seferde yalnızca bu kadar çok veriyi iletebilir ve alabilir. Kamera görüntüleri çok fazla veri gerektirebilir ve bu nedenle bu sınıra uymak nispeten kolaydır. Bir USB bant genişliği hatasının en yaygın nedeni, JPEG olmayan bir video modu seçmek veya özellikle birden fazla kamera bağlıyken çok yüksek bir çözünürlük çalıştırmaktır.

Mimari

CameraServer iki katmandan oluşur; yüksek düzeyli WPILib **CameraServer class** ve düşük düzey **cscore library**.

CameraServer Sınıfı

CameraServer sınıfı (WPILib'in bir parçası), robot kodunuza kamera eklemek için yüksek düzeyde bir arayüz sağlar. Ayrıca, LabVIEW Dashboard ve Shuffleboard gibi Driver Station panolarının kameraları listeleyebilmesi ve akışlarının nerede konumlandığını belirleyebilmesi için, kameralar ve kamera sunucuları hakkındaki bilgileri NetworkTables'da yayınlamaktan sorumludur. Oluşturulan tüm kameraların ve sunucuların bir veritabanını korumak için tek bir model kullanır.

CameraServer'daki bazı temel işlevler şunlardır:

- **startAutomaticCapture()** : Bir USB kamera ekleyin (örn. Microsoft LifeCam) ve pano-
dan görüntülenebilmesi için onun için bir sunucu başlatır.

- `addAxisCamera ()` : Bir Axis kamerası ekleyin. Axis kamerasından görüntüleri robot kodunuzda işlemiyor olsanız bile, Axis kamerasının Dashboard'un açılır kamera listesinde görünmesi için bu işlevi kullanmak isteyebilirsiniz. Ayrıca bir sunucu başlatır, böylece sürücü istasyonunuz roboRIO'ya USB aracılığıyla bağlandığında Axis akışı hala görüntülenebilir (iki robot radyo Ethernet portuna hem Axis kamera hem de roboRIO bağlıysa rekabette yararlıdır).
- `getVideo ()`: Bir kameraya OpenCV erişimi sağlayın. Bu, roboRIO'da (robot kodunuzda) görüntü işleme için kameradan görüntüler almanızı sağlar.
- `putVideo ()`: OpenCV görüntülerini besleyebileceğiniz bir sunucu başlatın. Bu, özel işlenmiş ve / veya açıklamalı görüntüleri kontrol paneline iletmenize olanak tanır.

cscore Kütüphanesi

Cscore kütüphanesi, aşağıdakilere daha düşük düzeyde uygulama sağlar:

- USB ve HTTP (örneğin Axis) kameralardan görüntüler alın
- Kamera ayarlarını değiştirin (ör. Kontrast ve parlaklık)
- Kamera video modlarını değiştirin (piksel formatı, çözünürlük ve kare hızı)
- Bir web sunucusu olarak hareket edin ve görüntüleri standart bir MJPEG akışı olarak sunun
- Görüntü işleme için görüntüleri OpenCV Mat nesnelere / nesnelere dönüştürün

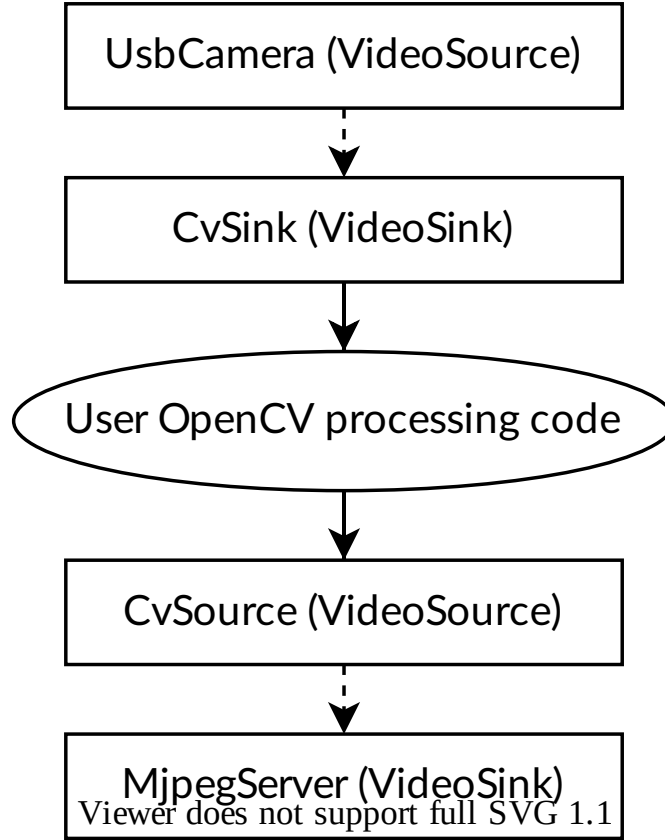
Kaynaklar ve Havuzlar

Cscore kütüphanesinin temel mimarisi, işlevsellik kaynaklar ve havuzlar arasında bölünmüş olarak MJPGStreamer ile benzerdir. Aynı anda oluşturulan ve çalışan birden çok kaynak ve birden çok havuz olabilir.

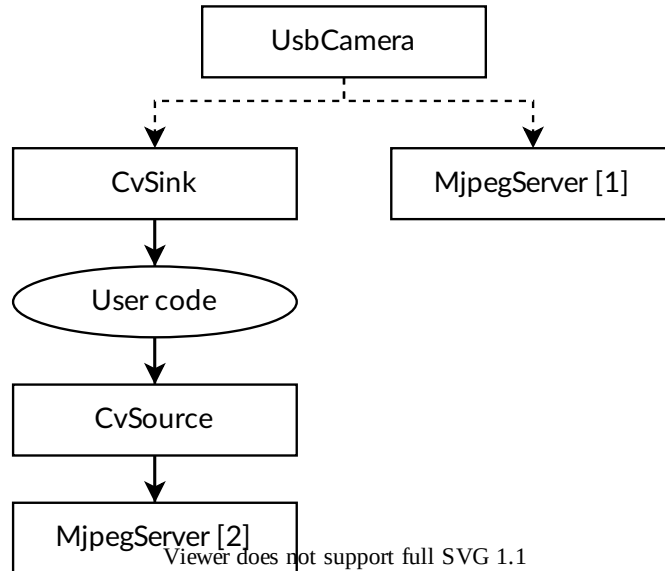
Görüntü üreten bir nesne bir kaynaktır ve görüntüleri kabul eden / tüketen bir nesne bir havuzdur. Üretme / tüketme, kütüphane perspektifindedir. Dolayısıyla kameralar birer kaynaktır (görüntü üretirler). MJPEG web sunucusu, programın içinden görüntüleri kabul ettiği için bir havuzdur (bu görüntüleri bir web tarayıcısına veya kontrol paneline iletiyor olsa bile). Kaynaklar birden fazla havuza bağlanabilir, ancak havuzlar bir ve yalnızca bir kaynağa bağlanabilir. Bir havuz bir kaynağa bağlandığında, cscore kitaplığı her bir görüntüyü kaynaktan havuza aktarmaya özen gösterir.

- **** Kaynaklar ****, ayrı kareleri (bir USB kamera tarafından sağlananlar gibi) alır ve yeni bir çerçeve mevcut olduğunda bir olayı tetikler. Belli bir kaynağı dinleyen havuzlar yoksa, işlemci ve G / Ç kaynaklarını kaydetmek için kitaplık duraklayabilir veya bir kaynaktan bağlantıyı kesebilir. Kütüphane, yalnızca olayların tetiklenmesini duraklatıp yeniden başlatarak kamera bağlantı kesmelerini / yeniden bağlantılarını özerk bir şekilde yönetir (örneğin, bir bağlantı kesilmesi yeni çerçevelerin olmaması, bir hata olmamasıyla sonuçlanır).
- ****Sinks-Havuzlar **** belirli bir kaynağın etkinliğini dinler, en son görüntüyü alır ve uygun formatta hedefine iletir. Kaynaklara benzer şekilde, belirli bir havuz etkin değilse (örneğin, HTTP sunucusu üzerinden yapılandırılmış bir MJPEG'e hiçbir istemci bağlı değilse), kitaplık, işlemci kaynaklarını korumak için işlemenin bazı kısımlarını devre dışı bırakabilir.

Kullanıcı kodu (bir FRC robot programında kullanılanlar gibi), OpenCV kaynağı ve havuz nesneleri aracılığıyla bir kaynak (bir kamera gibi işlenmiş çerçeveler sağlar) veya bir havuz (işlenmek üzere bir çerçeve alır) olarak işlev görebilir. Bu nedenle, bir kameradan görüntüleri alan ve işlenmiş görüntüleri sunan bir görüntü işleme hattı aşağıdaki grafiğe benzer:



Kaynakların bağlı birden fazla havuzu olabileceğinden, boru hattı dallara ayrılabilir. Örneğin, orijinal kamera görüntüsü, UsbCamera kaynağını CvSink'e ek olarak ikinci bir MjpegServer havuzuna bağlayarak da sunulabilir, bu da aşağıdaki grafikte sonuçlanır:



Kamera tarafından yeni bir görüntü yakalandığında, hem CvSink hem de MjpegServer [1] onu alır.

Yukarıdaki grafik, aşağıdaki CameraServer snippet'inin oluşturduğu şeydir:

JAVA

```
import edu.wpi.first.cameraserver.CameraServer;
import edu.wpi.first.cscore.CvSink;
import edu.wpi.first.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 uygulaması, aşağıdakileri cscore düzeyinde etkin bir şekilde yapar (açıklama amacıyla). CameraServer, tüm cscore nesneleri için benzersiz adlar oluşturma ve bağlantı noktası numaralarını otomatik olarak seçme gibi birçok ayrıntıyla ilgilenir. CameraServer ayrıca, kapsam dışına çıktıklarında imha edilmemeleri için oluşturulan nesnelerin tek bir kaydını tutar.

JAVA

```
import edu.wpi.first.cscore.CvSink;
import edu.wpi.first.cscore.CvSource;
import edu.wpi.first.cscore.MjpegServer;
import edu.wpi.first.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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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::kJPEG, 640, 480, 30);
cs::MjpegServer mjpegServer2("serve_Blur", 1182);
mjpegServer2.SetSource(outputStream);
```

Referans Sayma

Tüm cscore nesneleri dahili olarak referans olarak sayılır. Bir havuzun bir kaynağa bağlanması, kaynağın referans sayısını artırır, bu nedenle yalnızca havuzun kapsamda tutulması kesinlikle gereklidir. CameraServer sınıfı, CameraServer işlevleriyle oluşturulan tüm nesnelerin kaydını tutar, bu nedenle bu şekilde oluşturulan kaynaklar ve havuzlar hiçbir zaman kapsam dışına çıkmaz (açıkça kaldırılmadıkça).

25.1.6 2017 Vision Örnekleri

LabVIEW

2017 LabVIEW Vision Örneği diğer LabVIEW örnekleriyle birlikte verilmektedir. Açılış ekranından Destek-> FRC Bul | reg | Örneklerden veya başka herhangi bir LabVIEW penceresinden, Yardım-> Örnekler Bul'a tıklayın ve 2017 Vision Örneğini bulmak için Vision klasörünü bulun. Örnek görüntüler örnekle paketlenmiştir.

25.2 WPILibPi ile Vision-Görüntü işleme

25.2.1 Raspberry Pi ile WPILibPi'yi Kullanmanın Videosu

Not: Video, WPILibPi'nin eski adı olan FRCVision'dan bahsediyor.

WPILib ekibinden Peter Johnson 2020'de "WPI Tarafından Sunulan RSN Bahar Konferansı"nda FRC|reg| Raspberry Pi ile görme işlemini açıklıyor..

Sunum bağlantısı [here](#). mevcuttur.

25.2.2 Görme işleme için bir Yardımcı işlemci kullanma

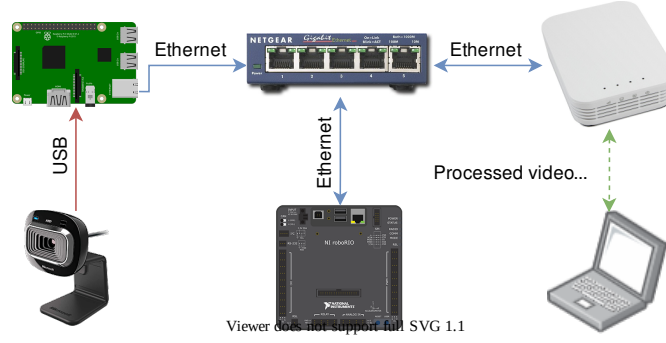
Saha hedeflerini veya oyun parçalarını tanımak için OpenCV gibi kitaplıkları kullanan vision processing-görüntü işleme, genellikle CPU yoğun bir işlem olabilir. Çoğu zaman yük çok önemli değildir ve işlem roboRIO tarafından kolayca halledilebilir. Daha fazla kamera akışının olduğu veya görüntü işlemenin karmaşık olduğu durumlarda, kodu ve kamera bağlantısını farklı bir işlemciye koyarak roboRIO'nun yükünü boşaltmak istenir. FRC ® içinde popüler olan bir dizi işlemci seçeneği vardır. Raspberry PI, Intel tabanlı Kangaroo, en üst düzeyde basitlik için LimeLight veya daha karmaşık görüntü kodları için nVidia Jetson modellerinden biri gibi bir grafik hızlandırıcı gibi.

Strateji

Genel olarak fikir, yardımcı işlemciyi genel olarak aşağıdakileri içeren gerekli yazılımla kurmaktır:

- OpenCV - açık kaynak bilgisayar görme kitaplığı
- *NetworkTables* - to commute the results of the image processing to the roboRIO program
- Kamera sunucusu kitaplığı - kamera bağlantılarını yönetmek ve bir panoda görüntülenebilen akışları yayınlamak için
- Görme programı için kullanılan bilgisayar dili için dil kitaplığı
- Nesne algılamayı gerçekleştiren gerçek görüş programı

The coprocessor is connected to the roboRIO network by plugging it into the extra ethernet port on the network router or, for more connections, adding a small network switch to the robot. The cameras are plugged into the coprocessor, it acquires the images, processes them, and publishes the results, usually target location information, to NetworkTables so it is can be consumed by the robot program for steering and aiming.



Kamera verilerinin kontrol paneline aktarılması

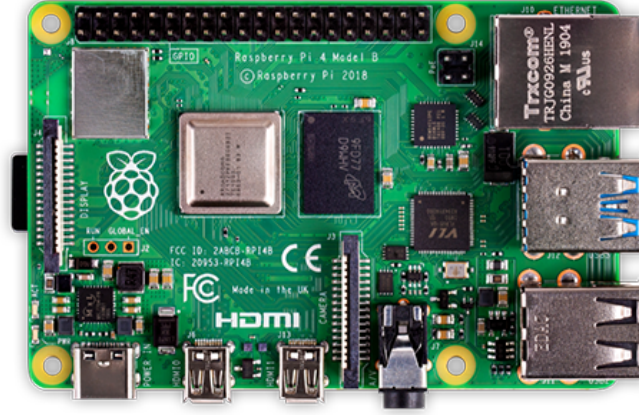
Genellikle, kamera verilerinin robot ağı üzerinden kontrol paneline basitçe aktarılması arzu edilir. Bu durumda, bir veya daha fazla kamera bağlantısı ağa gönderilebilir ve Shuffleboard veya web tarayıcısı gibi bir kontrol panelinde görüntülenebilir. Shuffleboard'u kullanmak, kamera çözünürlüğünü ve bit hızını ayarlamak için kolay kontrollere sahip olmanın yanı sıra kamera akışlarını robottan gönderilen diğer verilerle entegre etme avantajına sahiptir.

Ayrıca görüntüleri işlemek ve görüntüye açıklama eklemek, örneğin görüntü işleme kodunun algıladıklarını gösteren hedef satırlar veya kutular gibi ek açıklama eklemek ve ardından operatörlerin etrafındakilerin net bir resmini görmesini kolaylaştırmak için gösterge panosuna iletmek de mümkündür. robot.

25.2.3 Raspberry Pi'yi FRC için kullanma

En popüler yardımcı işlemci seçeneklerinden biri Raspberry Pi'dir çünkü:

- Düşük maliyet - yaklaşık 35 ABD doları
- Yüksek kullanılabilirlik - Raspberry Pi'si Amazon dahil birçok tedarikçiden bulmak kolaydır
- Çok iyi performans - mevcut Raspberry Pi 3b + aşağıdaki özelliklere sahiptir:
- Teknik Özellikler: - Broadcom BCM2837BO 64 bit ARMv8 QUAD Core A53 64bit İşlemciden güç alan, 1.4GHz'de çalışan Tek Kartlı Bilgisayar - 1GB RAM - BCM43143 Dahili WiFi - Dahili Bluetooth Düşük Enerji (BLE) - 40 pin genişletilmiş GPIO - 4 x USB2 bağlantı noktası - 4 kutuplu Stereo çıkış ve Kompozit video bağlantı noktası - Tam boyutlu HDMI - Raspberry bağlamak için CSI kamera bağlantı noktası - Pi kamera - Raspberry bağlamak için DSI ekran bağlantı noktası - Pi dokunmatik ekran görüntüsü - İşletim sisteminizi yüklemek ve verileri depolamak için MicroSD bağlantı noktası - Yükseltilmiş anahtarlı Mikro USB güç kaynağı (artık 2,5 Ampere kadar desteklemektedir).



Önceden oluşturulmuş Raspberry Pi görüntüsü

Raspberry Pi'yi ekipler için olabildiğince kolaylaştırmak için sağlanan bir Raspberry Pi görüntüsü vardır. Görüntü bir mikro SD karta kopyalanabilir, Pi'ye takılabilir ve önyüklenebilir. Varsayılan olarak şunları destekler:

- A web interface for configuring it for the most common functions
- Ağ arayüzünde yayınlanan rastgele sayıda kamera akışını (varsayılan olarak birdir) destekler
- OpenCV, [NetworkTables](#), Camera Server, and language libraries for C++, Java, and Python custom programs

Tek gereksinim ağa (ve kontrol paneline) bir veya daha fazla kamerayı yayınlamaksa, o zaman programlama gerekmez ve web arayüzü üzerinden tamamen kurulabilir.

Bir sonraki bölümde görüntünün bir flash karta nasıl yükleneceği ve Pi'nin nasıl başlatılacağı anlatılmaktadır.

25.2.4 Pi görüntüsünü çalıştırmak için neye ihtiyacınız var?

Raspberry Pi'yi bir video veya görüntü işlemcisi olarak kullanmaya başlamak için aşağıdakilere ihtiyacınız vardır:

- Raspberry Pi 3 B, Raspberry Pi 3 B+ veya Raspberry Pi 4 B
- Önerilen Hız Sınıfı 10 (10 MB/sn) olan, sağlanan tüm yazılımları saklamak için en az 8 Gb olan bir mikro SD kart
- Pi'yi roboRIO ağınıza bağlamak için ethernet kablosu
- Robotunuzdaki Voltaj Düzenleyici Modülüne (VRM) bağlanmak için bir USB mikro güç kablosu. Daha yüksek güvenilirlik için roboRIO USB bağlantı noktalarından birinden güç sağlamak yerine güç için VRM bağlantısının kullanılması önerilir
- Bir USB dongle (tercih edilir) veya çoğu MicroSD kartla birlikte gelen bir SD'den MicroSD'ye adaptör kullanarak MicroSD karta yazabilen bir dizüstü bilgisayar

Gösterilen, MicroSD kartına görüntü FRC|reg| yazacak ucuz bir USB dongle'dır.



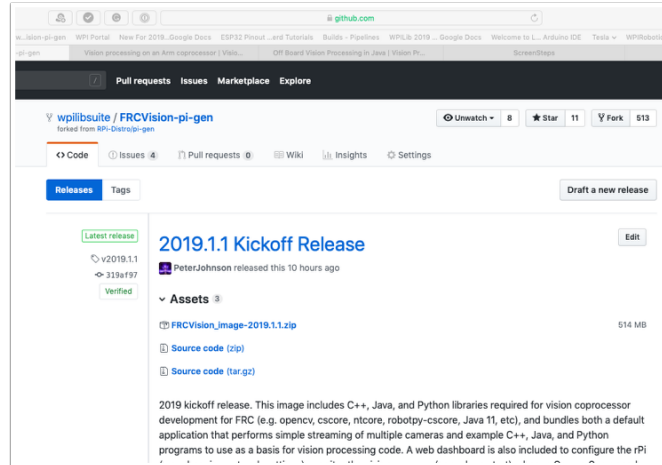
25.2.5 Görüntüyü MicroSD kartınıza yükleme

FRC Raspberry PI görüntüsünü alma

Görüntü, WPILibPi [deposu](#) için GitHub yayın sayfasında saklanır.

Bu sayfadaki talimatlara ek olarak GitHub web sayfasındaki (aşağıda) belgelere bakın.

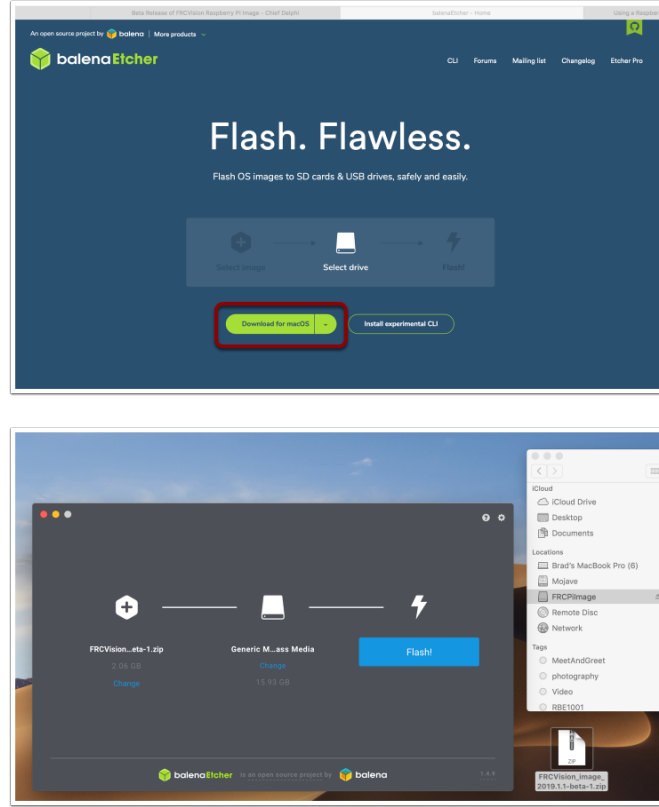
Görsel oldukça büyük olduğundan, indirirken hızlı bir internet bağlantısına sahip olun. Her zaman sürüm listesinin en üstündeki en son sürümü kullanın.



Görüntüyü MicroSD kartınıza kopyalayın

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.

Kaynak olarak zip dosyasını, hedef olarak SD kartınızı seçerek ve “Flash” seçeneğine tıklayarak MicroSD kartını Etcher kullanarak görüntü ile flaşlayın. Oldukça hızlı bir dizüstü bilgisayarda işlemin yaklaşık 3 dakika sürmesini bekleyin.



Raspberry PI Test Etme

1. Mikro SD kartı bir rPi 3'e yerleştirin ve gücü uygulayın.
2. rPi 3 ethernet'i bir LAN veya PC'ye bağlayın. Bir web tarayıcısı açın ve web panosunu açmak için <http://wpilibpi.local/> adresine bağlanın. İlk önyüklemede dosya sistemi yazılabilir olacaktır, ancak sonraki önyüklemeler varsayılan olarak salt okunur olacaktır, bu nedenle değişiklik yapmak için "writable-yazılabilir" düğmesine tıklamak gerekir.

Raspberry PI'da oturum açma

rPi ile çoğu görev, web konsolu arayüzünden yapılabilir. Bazen rPi'de program geliştirme gibi gelişmiş kullanım için oturum açmak gerekir. Oturum açmak için varsayılan Raspberry PI şifresini kullanın:

```
Username: pi
Password: raspberry
```

25.2.6 Raspberry PI

FRC Konsolu

FRC [reg] Raspberry PI görüntüsü, aşağıdakileri kolaylaştıran herhangi bir web tarayıcısında görüntülenebilen bir konsol içerir:

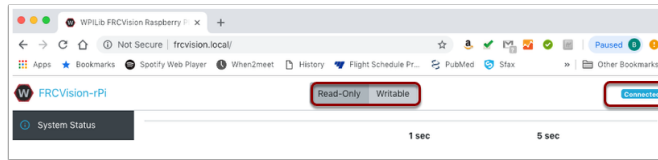
- Raspberry PI durumuna bakın
- Kamerayı çalıştıran arka plan işleminin durumunu görüntüleyin
- Ağ ayarlarını görüntüleyin veya değiştirin
- RPI'ye bağlı her kameraya bakın ve ek kameralar ekleyin
- RPI'ye yeni bir görsel denetim programı yükleyin

RPI'yi Salt Okunur ve Yazılabilir Olarak Ayarlama

RPI normalde Read-Only-Salt Okunur olarak ayarlanır, bu da dosya sisteminin değiştirilemeyeceği anlamına gelir. Bu, rPi'yi kapatmadan güç kesilirse dosya sisteminin bozulmamasını sağlar. Ayarlar değiştirildiğinde (sonraki bölümler), rPI dosya sistemi Salt Okunur olarak ayarlandığında yeni ayarlar kaydedilemez. Dosya sisteminin Salt Okunur'dan Yazılabilir'e değiştirilmesine ve her değişiklik yapıldığında geri alınmasına izin veren düğmeler sağlanmıştır. RPI'da depolanan bilgileri değiştiren diğer düğmelere basılamıyorsa, sistemin Salt Okunur durumunu kontrol edin.

RPI'ye ağ bağlantısının durumu

Konsolun sağ üst köşesinde, rPi'nin şu anda bağlı olup olmadığını gösteren bir etiket vardır. Artık rPi'ye ağ bağlantısı yoksa, Bağlı'dan Bağlantı Kesildi'ye değişecektir.



Sistem durumu

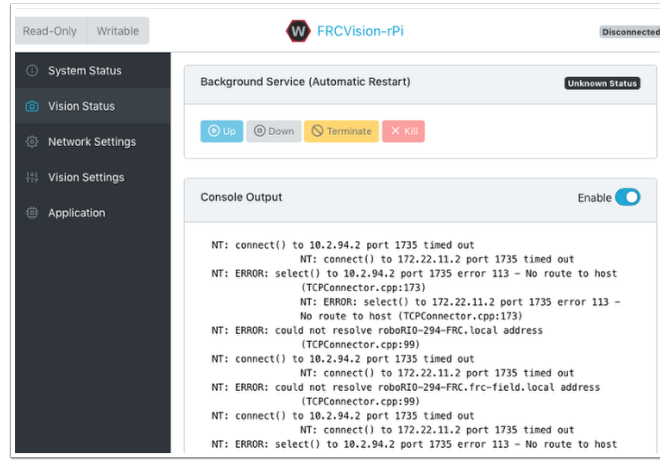
The screenshot shows the FRCVision-rPi web interface with the 'System Status' section expanded. It displays a table of system metrics for '1 sec' and '5 sec' intervals. The metrics include Memory (MB Free), Memory (MB Avail), CPU (% User), CPU (% System), CPU (% Idle), and Network (Kbps). A 'Restart System' button is visible at the bottom.

	1 sec	5 sec
Memory (MB Free)	809	809
Memory (MB Avail)	816	816
CPU (% User)	0	0
CPU (% System)	0	0
CPU (% Idle)	100	99
Network (Kbps)	8	9

Sistem durumu, rPI üzerindeki CPU'nun herhangi bir zamanda ne yaptığını gösterir. 1 saniyelik ortalama ve diğeri 5 saniyelik ortalama olmak üzere iki durum değeri sütunu vardır. Gösterilen:

- 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

Vizyon Durumu



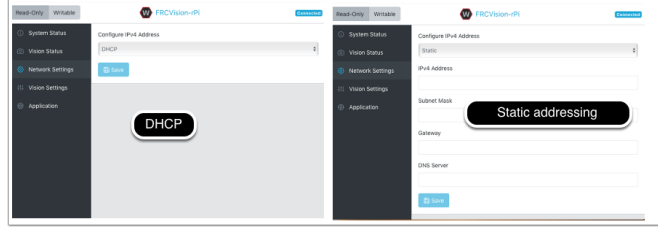
Allows monitoring of the task which is running the camera code in the rPI, either one of the default programs or your own program in Java, C++, or Python. You can also enable and view the console output to see messages coming from the background camera service. In this case there are number of messages about being unable to connect to [NetworkTables](#) (NT: connect()) because in this example the rPI is simply connected to a laptop with no NetworkTables server running (usually the roboRIO.)

Ağ ayarları

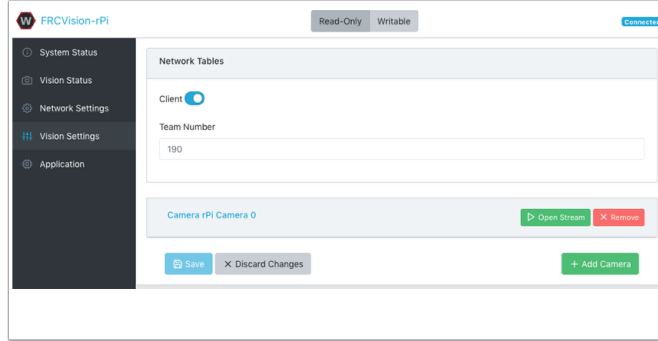
RPI ağ ayarlarının PI'ya bağlanma seçenekleri vardır:

- **DHCP** - the default name resolution usually used by the roboRIO. The default name is wpilibpi.local.
- Statik - sabit bir IP adresi, ağ maskesi ve yönlendirici ayarlarının açıkça doldurulduğu yer
- Statik Geri Dönüslü DHCP - Statik Geri Dönüslü DHCP - PI, DHCP aracılığıyla bir IP adresi almaya çalışır, ancak bir DHCP sunucusu bulamazsa, sağlanan statik IP adresini ve parametreleri kullanır.

Yukarıdaki resim hem DHCP hem de Statik IP Adresleme için ayarları göstermektedir. RPi için mDNS adı, yukarıda seçilen seçeneklerden bağımsız olarak her zaman çalışmalıdır.



Görme Ayarları



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).

Tüm kamera ayarlarını görüntülemek ve değiştirmek için söz konusu kameraya tıklayın. Bu durumda, kamera “Camera rPi Camera 0” olarak adlandırılır ve ada tıklamak, mevcut kamera görünümünü ve ilgili ayarları ortaya çıkarır.

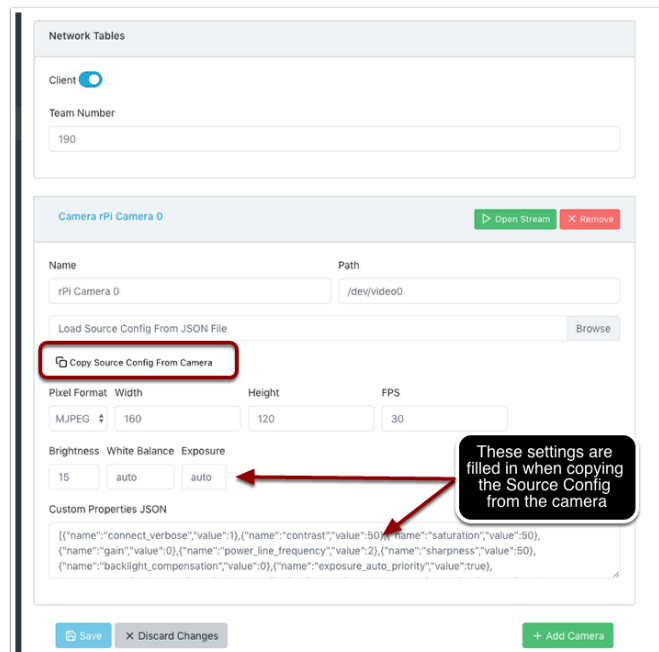
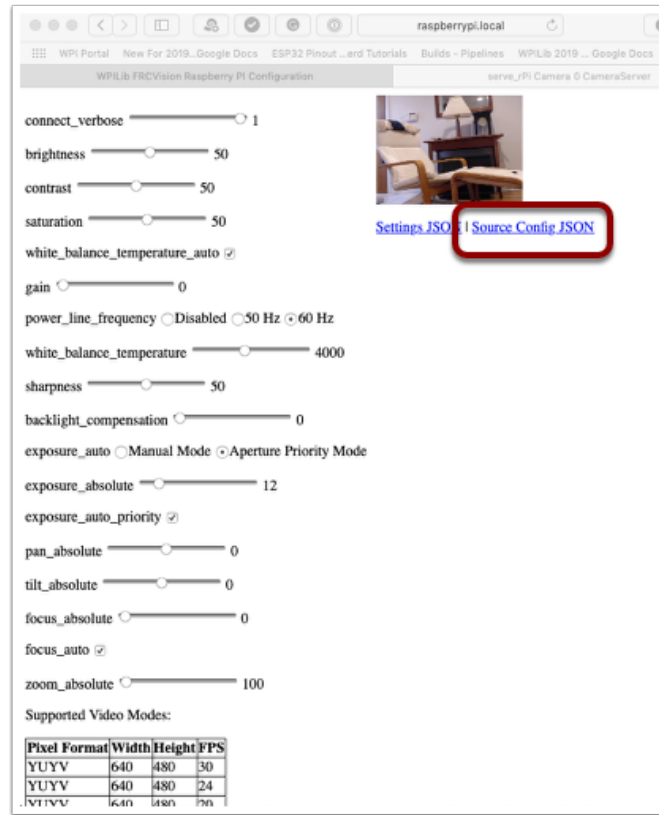
Kamera ayarlarının değiştirilmesi mevcut kamera görünümüne yansıtılır. Sayfanın alt kısmında bu kamera tarafından desteklenen tüm olası kamera modları (Genişlik, Yükseklik ve kare hızlarının kombinasyonları) gösterilmektedir.

Not: Kamera görüntüsü *Open Stream* ekranında görünmüyorsa, sayfanın altındaki desteklenen video modlarını kontrol edin. Ardından ‘Vision Settings’ e geri dönün ve söz konusu kameraya tıklayın ve piksel formatı, genişlik, yükseklik ve FPS’nin desteklenen video modlarında listendiğini doğrulayın.

Yeniden başlatmalarda devam etmek için mevcut ayarları alma

RPi, başlangıçta tüm kamera ayarlarını yükleyecektir. Yukarıdaki ekranda kamera yapılandırmasının düzenlenmesi geçicidir. Değerlerin kalıcı olmasını sağlamak için “Load Source Config From Camera-Kameradan Kaynak Yapılandırmasını Yükle” düğmesine tıklayın ve mevcut ayarlar kamera ayarları alanlarına girilecektir. Ardından sayfanın altındaki “Kaydet” i tıklayın. Not: Ayarları kaydetmek için Yazılabilir dosya sistemini ayarlamalısınız. * Yazılabilir düğmesi sayfanın en üstündedir. *

Kamera ayarlarında (yukarıda) gösterilen yaygın olarak kullanılan bazı kamera ayarları değerleri vardır. Bu değerler Parlaklık, Beyaz Dengesi ve Pozlama, kullanıcı JSON dosyası uygulan-



madan önce kameraya yüklenir. Dolayısıyla, bir kullanıcı JSON dosyası bu ayarları içeriyorsa, metin alanındakilerin üzerine yazacaktır.

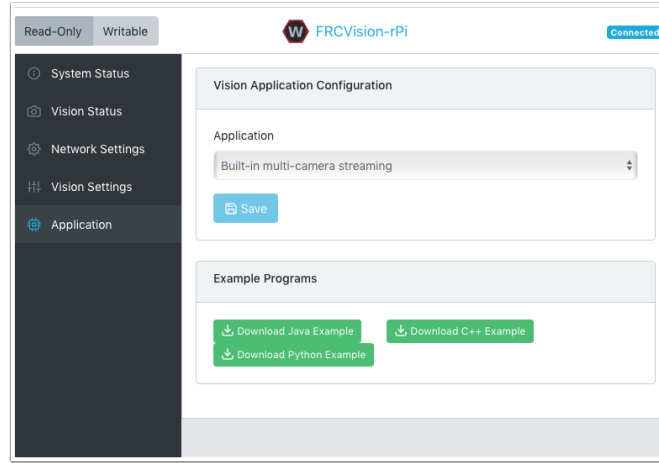
Uygulama

Uygulama sekmesi şu anda rPi’de çalışan uygulamayı gösterir.

Vision iş akışları

Desteklenen dillerin her birinde, C ++, Java veya Python’da OpenCV kullanan örnek bir görme programı vardır. Her örnek program rPi’den video yakalayabilir ve yayınlayabilir. Ek olarak, örneklerde minimum düzeyde OpenCV bulunur. Bunların tümü, sağlanan OpenCV örnek kodunu robot uygulaması için gereken kodla değiştirmek üzere genişletilecek şekilde ayarlanmıştır. RPi Uygulaması sekmesi bir dizi programlama iş akışını destekler:

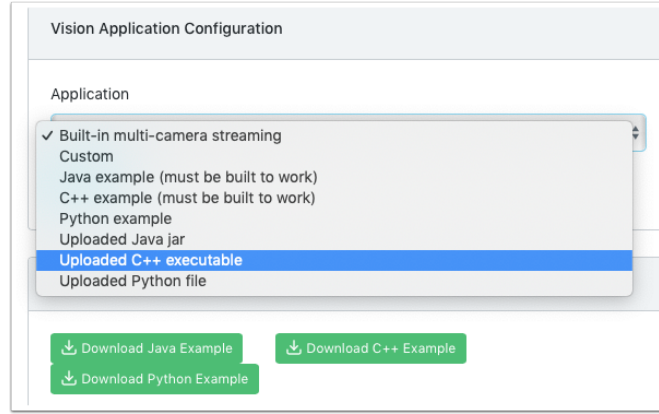
- Sürücü istasyonu bilgisayarında tüketilmek üzere rPi’den bir veya daha fazla kamerayı yayınlayın ve ShuffleBoard kullanılarak görüntülenir
- Dahil edilen araç zincirlerini kullanarak rPi’de örnek programlardan birini (her dil için bir tane: Java, C ++ veya Python) düzenleyin ve oluşturun
- Seçilen dil için örnek bir program indirin ve onu geliştirme bilgisayarınızda düzenleyin ve oluşturun. Ardından bu yerleşik programı rPi’ye geri yükleyin
- Tamamen özel uygulamalar ve komut dosyaları kullanarak her şeyi kendiniz yapın (muhtemelen örneklerden birine göre)



Çalışan uygulama, açılır menüdeki seçeneklerden biri seçilerek değiştirilebilir. Seçenekler şunlardır:

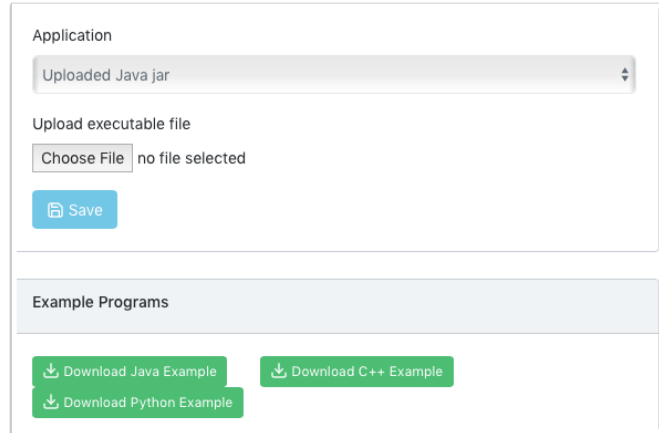
- Built-in multi camera streaming which streams whatever cameras are plugged into the rPi. The camera configuration including number of cameras can be set on the “Vision Settings” tab.
- RPi’ye hiçbir şey yüklemeyen ve geliştiricinin özel bir program ve komut dosyasına sahip olmak istediğini varsayan özel uygulama.
- Kendi uygulamanıza düzenlenebilen önceden yüklenmiş Java, C ++ veya Python örnek programlar.

- Java, C ++ veya Python yüklenmiş program. Java programları, derlenen programla birlikte bir “.jar ” dosyası gerektirir ve C ++ programları, rPI’nin rPI’ye yüklenmesini gerektirir.



Yükleme seçeneklerinden birini seçerken, jar, yürütülebilir veya Python programının seçilebildiği ve rPi’ye yüklenebildiği bir dosya seçici sunulur. Aşağıdaki resimde Yüklenmiş bir Java kavanozu seçilmiştir ve “Choose File-Dosya Seç” düğmesi bir dosya seçecek ve “Save-Kaydet” düğmesine tıklamak seçilen dosyayı yükleyecektir.

Not: rPi’ye yeni bir dosya kaydetmek için, dosya sistemi web sayfasının sol üst köşesindeki “Writable-Yazılabilir” düğmesi kullanılarak yazılabilir olarak ayarlanmalıdır. Yeni dosyayı kaydettikten sonra, yanlışlıkla değişikliklere karşı korunması için dosya sistemini tekrar “Read-Only-Salt Okunur” olarak ayarlayın.



25.2.7 CameraServer’ı kullanma

CameraServer’den Çerçeveleri Yakalama

WPILibPi görüntüsü, kendi görüntü işleme sisteminizi oluşturmak için gerekli tüm kitaplıklarla birlikte gelir. Geçerli kareyi kameradan almak için CameraServer kitaplığını kullanabilirsiniz. CameraServer hakkında bilgi için : ref: docs / software / vision-processing / Introduction / cameraserver-class: Read and Process Video: CameraServer Class.

PY

```
from cscore import CameraServer
import cv2
import numpy as np

CameraServer.enableLogging()

camera = CameraServer.startAutomaticCapture()
camera.setResolution(width, height)

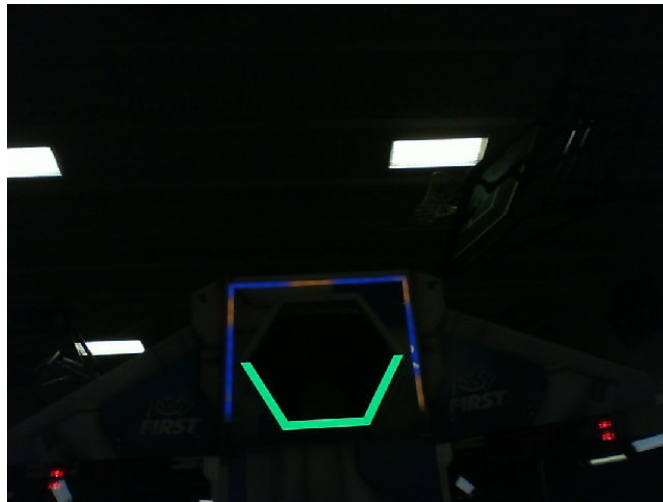
sink = cs.getVideo()

while True:
    time, input_img = cvSink.grabFrame(input_img)

    if time == 0: # There is an error
        continue
```

Not: OpenCV, görüntüde geçmiş nedenlerden dolayı **RGB** değil **BGR** olarak okur. RGB olarak değiştirmek istiyorsanız, `cv2.cvtColor` kullanın.

Aşağıda CameraServer'dan alınabilecek bir görüntü örneği verilmiştir.



CameraServer'a çerçeve gönderme

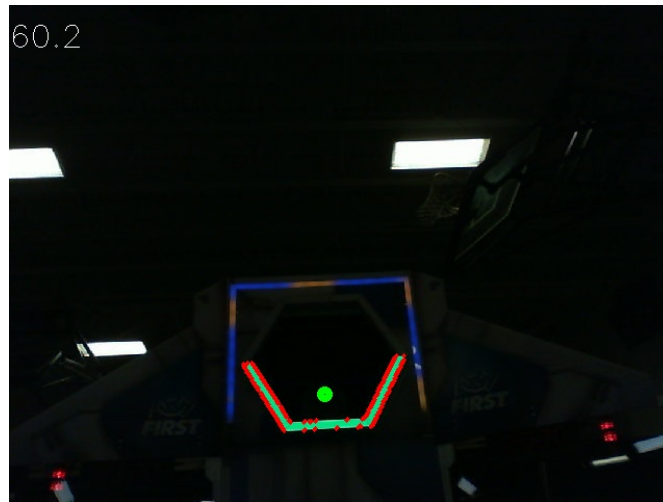
Bazen, hata ayıklama amacıyla veya Shuffleboard gibi bir pano uygulamasında görüntülemek için işlenmiş video karelerini CameraServer örneğine geri göndermek isteyebilirsiniz.

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)
```

Örnek olarak, işleme kodu hedefi kırmızıyla özetleyebilir ve hata ayıklama amacıyla köşeleri sarı olarak gösterebilir.

Aşağıda, CameraServer'a geri gönderilecek ve Driver Station bilgisayarında görüntülenecek tamamen işlenmiş bir görüntü örneği bulunmaktadır.



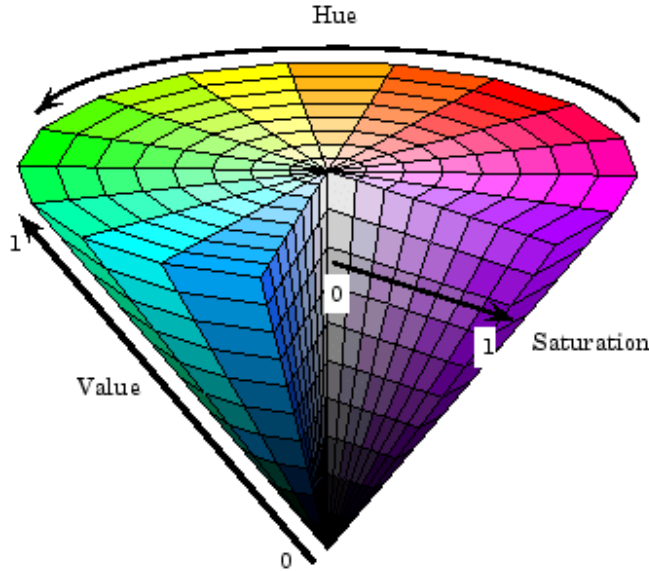
25.2.8 Bir Görüntüyü Eşikleme

Kameranız tarafından çekilmiş gibi renkli bir görüntüyü hedefin “foreground-ön plan” olduğu ikili bir görüntüye dönüştürmek için, her pikselin tonunu, doygunluğunu ve değerini kullanarak görüntüyü eşik yapmamız gerekir.

HSV Modeli

RGB’den farklı olarak HSV, yalnızca piksellerin renklerine göre değil, aynı zamanda renk yoğunluğuna ve parlaklığa göre de filtre uygulamanıza olanak tanır.

- Ton: Pikselin rengini ölçer.
- Doygunluk: Pikselin renk yoğunluğunu ölçer.
- Value: Measures the brightness of the pixel.



Bir BGR görüntü matrisini HSV’ye dönüştürmek için OpenCV’yi kullanabilirsiniz.

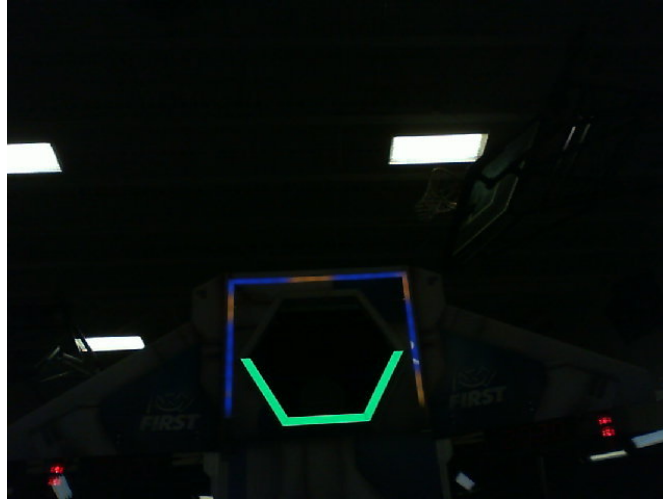
PY

```
hsv_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2HSV)
```

Not: OpenCV’nin ton aralığı, ortak 1 ° ila 360 ° yerine 1 ° ila 180 ° arasındadır. Ortak bir ton değerini OpenCV’ye dönüştürmek için 2’ye bölün.

Thresholding-Eşik

Bu alan görüntüsünü tüm görüntü işleme süreci için örnek olarak kullanacağız.



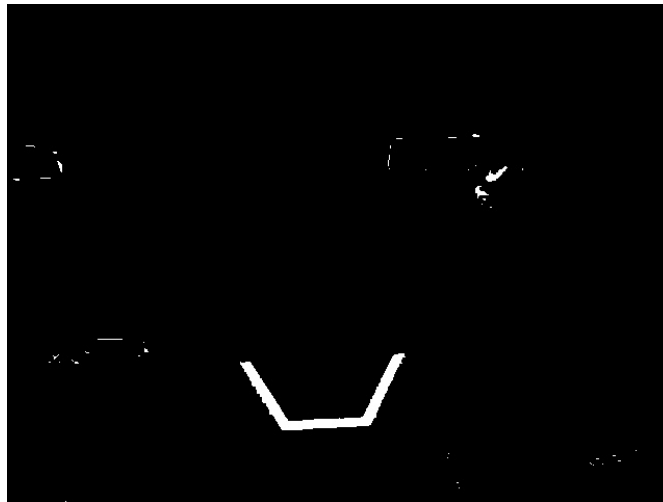
Görüntüyü HSV kullanarak eşleyerek, görüntüyü hedef (ön plan) ve kameranın gördüğü diğer şeyler (arka plan) olarak ayırabilirsiniz. Aşağıdaki kod örneği, bir HSV görüntüsünü HSV değerleriyle eşikleyerek ikili görüntüye dönüştürür.

PY

```
binary_img = cv2.inRange(hsv_img, (min_hue, min_sat, min_val), (max_hue, max_sat, max_val))
```

Not: Ortam aydınlatması mekanlar arasında farklılık gösterebileceğinden, bu değerlerin mekana göre ayarlanması gerekebilir. Anında düzenlemeyi kolaylaştırmak için bu değerlerin NetworkTables aracılığıyla düzenlenmesine izin verilmesi önerilir.

Eşiklemeden sonra, görüntünüz böyle görünmelidir.



Gördüğünüz gibi, eşikleme işlemi% 100 temiz olmayabilir. Gürültüyle başa çıkmak için morfolojik işlemleri kullanabilirsiniz.

25.2.9 Morfolojik İşlemler

Bazen, görüntünüzü eşledikten sonra, ikili görüntünüzde istenmeyen parazit olur. Morfolojik işlemler bu gürültüyü görüntüden çıkarmaya yardımcı olabilir.

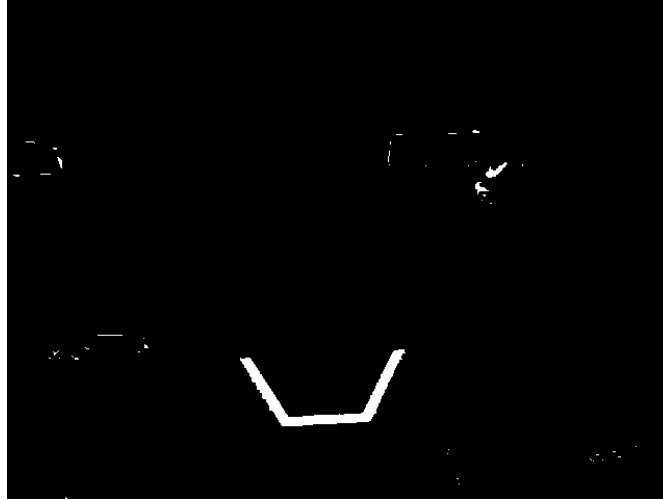
Çekirdek

Çekirdek, başlangıç noktasının ikili görüntünün 1 değerinin her pikselinin üzerine yerleştirildiği basit bir şekildir. OpenCV, çekirdeği NxN matrisiyle sınırlar, burada N tek sayıdır. Çekirdeğin kökeni merkezdir. Ortak bir çekirdek

$$kernel = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Farklı çekirdekler, görüntüyü farklı şekilde etkileyebilir, örneğin yalnızca dikey olarak aşındırma veya genişletme.

Referans için, oluşturduğumuz ikili imajımız:

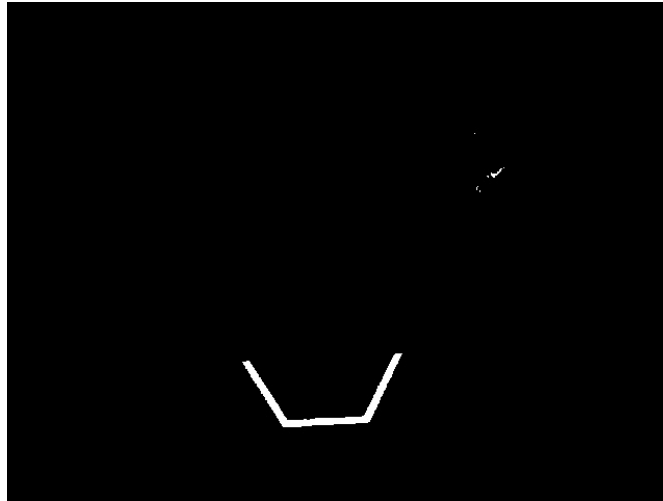


Erozyon

Bilgisayarla görmedeki erozyon, topraktaki erozyona benzer. Ön plandaki nesnelerin sınırlarından uzaklaşır. Bu işlem arka plandaki gürültüyü ortadan kaldırabilir.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.erode(binary_img, kernel, iterations = 1)
```



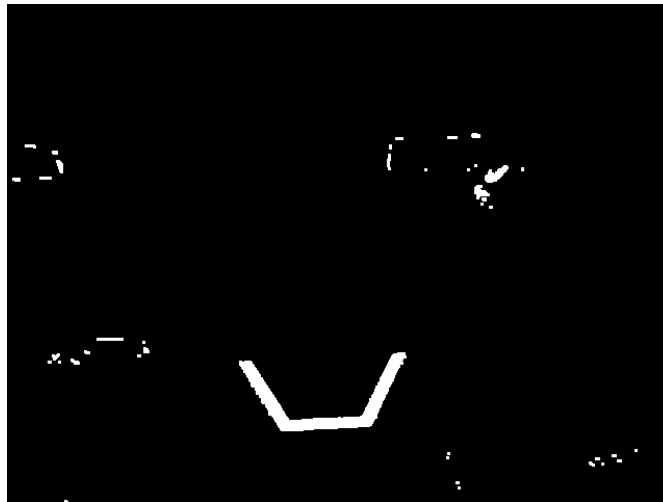
Erozyon sırasında, üst üste binen çekirdeğin pikselleri tamamen ikili görüntünün pikselleri tarafından kapsanmazsa, üzerine bindirildiği piksel silinir.

Dilation-Genişleme

Genleşme erozyonun tam tersidir. Sınırlardan uzaklaşmak yerine onlara katkıda bulunuyor. Bu işlem, daha geniş bir bölgedeki küçük delikleri kaldırabilir.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.dilate(binary_img, kernel, iterations = 1)
```



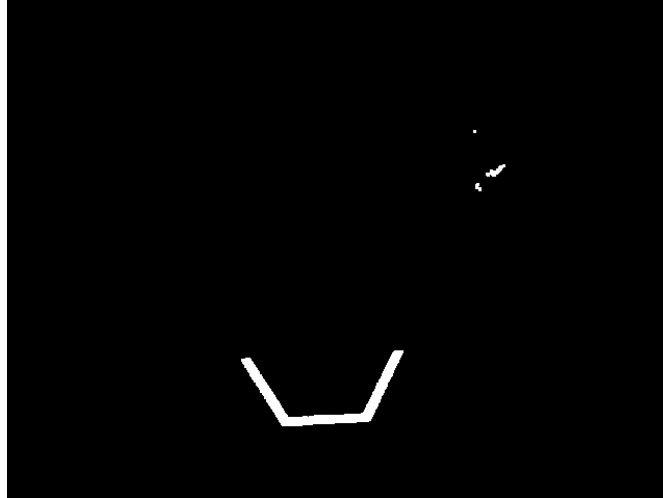
Genişletme sırasında, üst üste binen her çekirdeğin her pikseli genişlemeye dahil edilir.

Opening-Açılış

Açılma erozyon ve ardından genişlemedir. Bu işlem, daha büyük özelliklerin şeklini etkilemeden gürültüyü ortadan kaldırır.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel)
```



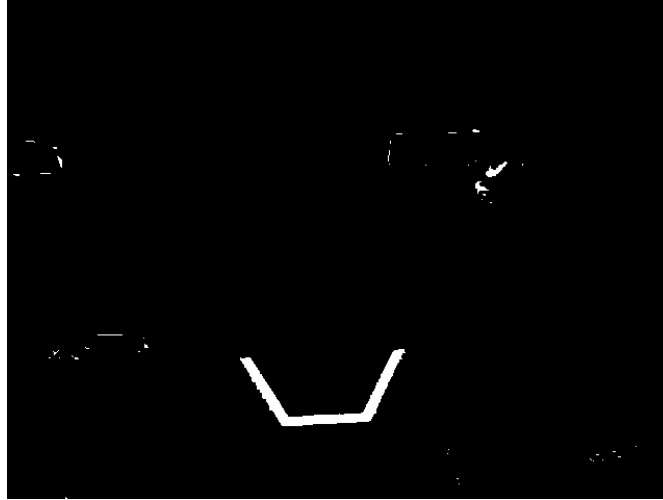
Not: Bu özel durumda, sağ üstteki piksellerden kurtulmak için daha fazla açma yinelemesi yapmak uygundur.

Closing-Kapanış

Kapatma genişlemedir ve ardından erozyon meydana gelir. Bu işlem, daha büyük unsurların şeklini etkilemeden küçük delikleri veya kırılmaları giderir.

PY

```
kernel = np.ones((3, 3), np.uint8)
binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, kernel)
```



25.2.10 Konturlarla Çalışma

Morfolojik işlemlerle gürültüyü eşikledikten ve kaldırdıktan sonra, artık OpenCV'nin findContours yöntemini kullanmaya hazırsınız. Bu yöntem, ikili görüntünüze göre konturlar oluşturmanıza olanak tanır.

Konturları Bulma ve Filtreleme

PY

```
_, contours, _ = cv2.findContours(binary_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

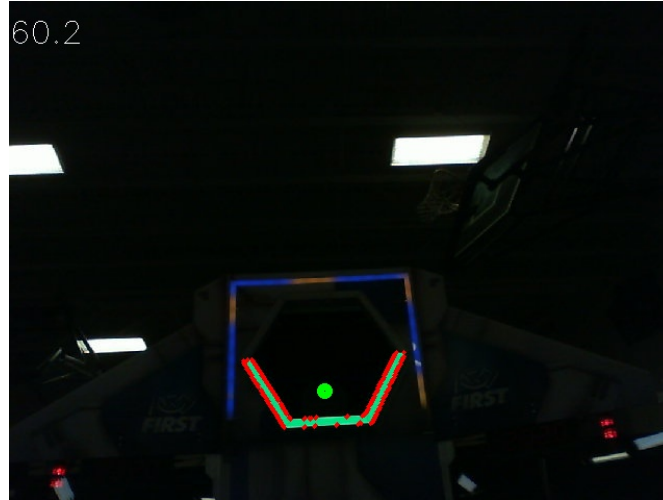
In cases where there is only one vision target, you can just take the largest contour and assume that is the target you are looking for. When there is more than one vision target, you can use size, shape, fullness, and other properties to filter unwanted contours out.

PY

```
if len(contours) > 0:
    largest = contours[0]
    for contour in contours:
        if cv2.contourArea(contour) > cv2.contourArea(largest):
            largest = contour

#
# Contour processing code
#
```

Yeni bulduğunuz konturu çizerseniz, şuna benzemelidir:



Konturlardan Bilgi Çıkarma

Artık istediğiniz kontur(ları) bulduğunuza göre, artık bununla ilgili merkez, köşeler ve dönüş gibi bilgiler almak istiyorsunuz.

Merkez

PY

```
rect = cv2.minAreaRect(contour)
center, _, _ = rect
center_x, center_y = center
```

Kornerler

PY

```
corners = cv2.convexHull(contour)
corners = cv2.approxPolyDP(corners, 0.1 * cv2.arcLength(contour), True)
```

Rotasyon

PY

```
_, _, rotation = cv2.fitEllipse(contour)
```

Bu değerleri nasıl kullanabileceğiniz hakkında daha fazla bilgi için bkz [Ölçümler](#)

NetworkTables’da yayınlama

Bu özellikleri Driver Station ve RoboRIO’ya göndermek için NetworkTables’ı kullanabilirsiniz. Raspberry Pi’de veya RoboRIO’nun kendisinde ek işlemler yapılabilir.

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 Temel Görüntü Örneği

Bu, hedefin konumunu açıklanan hedefleme koordinat sisteminde bildiren temel bir görüş kurulumunun bir örneğidir [here](#) NetworkTables’a ve tespit edilen konturun sınırlayıcı bir dik-dörtgenini görüntülemek için CameraServer’ı kullanır. Bu örnek, CameraServer’a gönderilen görüntülerde işlem kodunun kare hızını gösterecektir.

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()
```

(sonraki sayfaya devam)

```

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)

```

(önceki sayfadan devam)

```

x_list.append((center[0] - width / 2) / (width / 2))
x_list.append((center[1] - width / 2) / (width / 2))

vision_nt.putNumberArray('target_x', x_list)
vision_nt.putNumberArray('target_y', y_list)

processing_time = time.time() - start_time
fps = 1 / processing_time
cv2.putText(output_img, str(round(fps, 1)), (0, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255))
output_stream.putFrame(output_img)

main()

```

25.3 AprilTag Introduction

25.3.1 AprilTags nedir?



AprilTags, birçok farklı uygulama için düşük ek yük, yüksek doğrulukta yerleştirme sağlamak üzere Michigan Üniversitesi'ndeki araştırmacılar tarafından geliştirilmiş bir görsel etiketler sistemidir.

Additional information about the tag system and its creators [can be found on their website](https://www.firstinspires.org/robotics/frc/blog/2022-2023-approved-devices-rules-preview-and-vision-target-update). This document attempts to summarize the content for FIRST robotics related purposes.

FRC uygulaması

FRC şartlarında, AprilTag'ler, robotunuzun sahada nerede olduğunu bilmesine yardımcı olmak için yararlıdır, böylece robotunuzu bir hedef konumuna hizalayabilir.

AprilTag'ler 2011'den beri geliştirilmektedir ve saptama sağlamlığını ve hızını artırmak için yıllar içinde geliştirilmiştir.

2023'ten itibaren FIRST, bir dizi etiket sağlıyor. <<https://www.firstinspires.org/robotics/frc/blog/2022-2023-approved-devices-rules-preview-and-vision-target-update>>`__alana dağılmış, her biri bilinen bir :term`poz`da.

In 2024, [the tag family was updated to the 36h11 family](#).

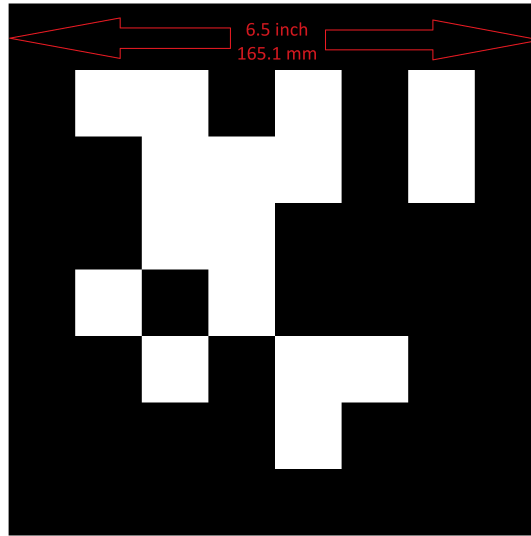
The AprilTag library implementation defines standards on how sets of tags should be designed. Some of the possible tag families [are described here](#).

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. Bazı bit çevirmelerine karşı dikkatli olun (IE, bir bitin renginin yanlış tanımlandığı sorunlar).
2. Etiket olmayan şeylerde bulunması muhtemel “basit” geometrik desenleri içermez. (IE, kareler, çizgiler vb.)
3. Geometrik desenin, hangi yolun yukarı olduğunu her zaman anlayabilmeniz için yeterince asimetrik olduğundan emin olun.

All tags will be printed such that the tag’s main “body” is 6.5 inches in length.



Evde kullanım için, etiket dosyalarının çıktısı alınabilir ve çalışma alanınızın etrafına yerleştirilebilir. İşleme algoritması etiketlerin düz olduğunu varsaydığından, etiketin düz kalmasını sağlamak için bunları sert bir destek malzemesine monte edin.

Yazılım Desteği

The main repository for the source code that detects and decodes AprilTags is [located here](#).

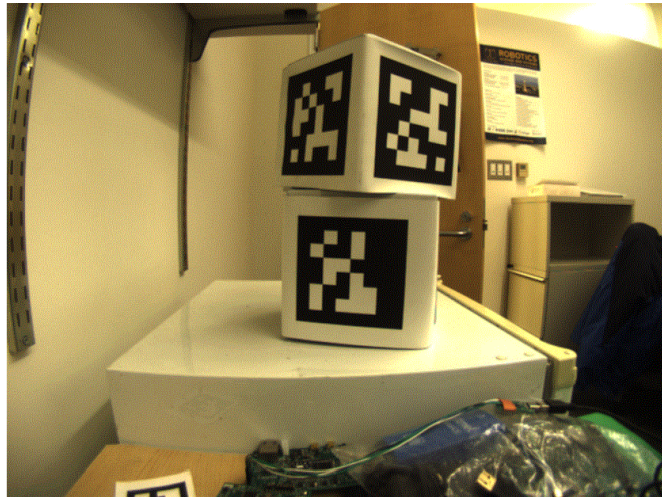
WPILib, FRC için yeni özellikler eklemek üzere depoyu ayırdı. Bunlar içerilir:

1. Raspberry Pi ve roboRIO dahil olmak üzere , yaygın FRC hedefleri için kaynak kodu oluşturmak
2. Java'dan işlevselliğini çağırmaya izin vermek için Java Yerel Arayüz (JNI) desteği ekleme
3. Gradle & Maven yayınlama desteği

İşleme Tekniği

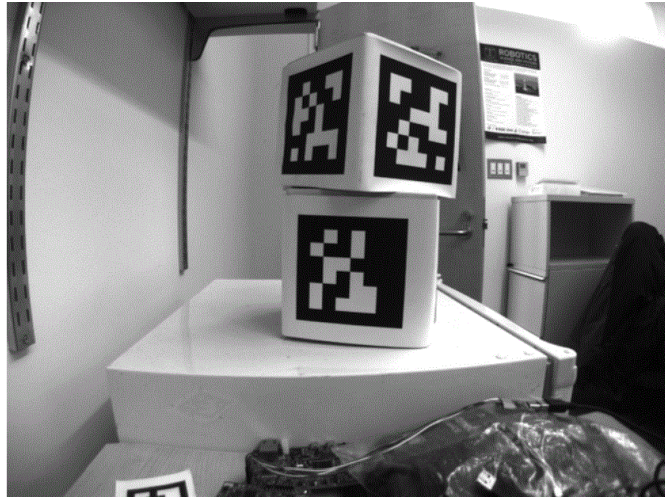
Çoğu FRC ekibinin bir kamera görüntüsünde AprilTag'leri tanımlamak için kendi kodlarını uygulaması gerekmeseyse de, temeldeki kitaplıkların nasıl çalıştığının temellerini bilmek yararlıdır.

Gerçek görüntü



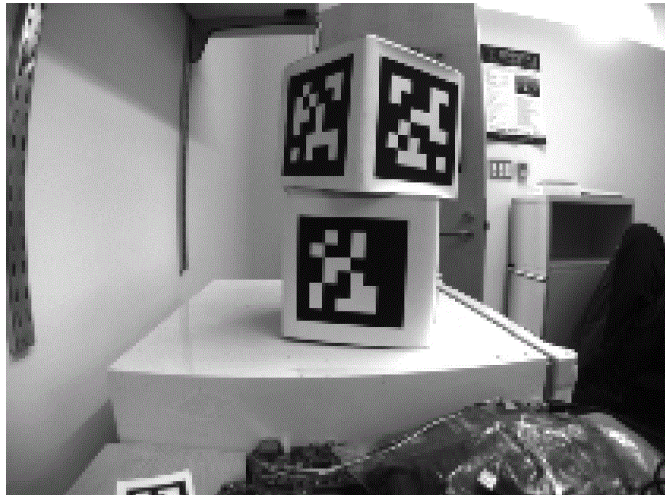
An image from a camera is simply an array of values, corresponding to the color and brightness of each pixel.

Remove Colors



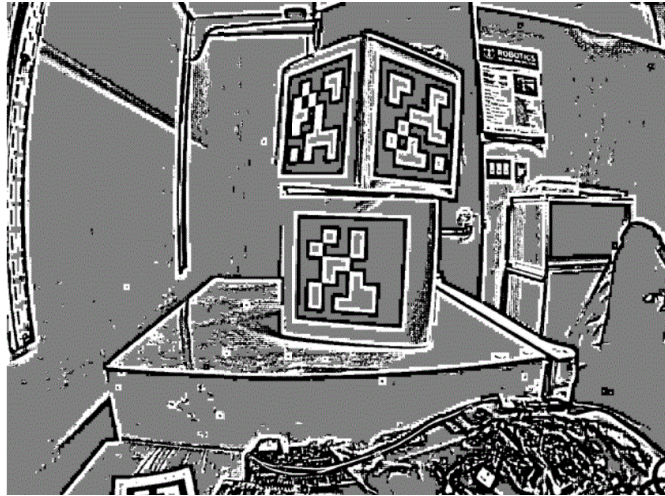
The first step is to convert the image to a grey-scale (brightness-only) image. Color information is not needed to detect the black-and-white tags.

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.

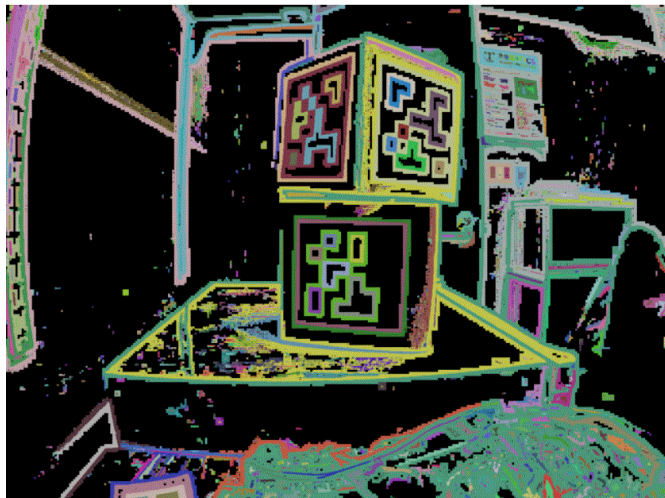
Adaptive Threshold



An adaptive threshold algorithm is run to classify each pixel as “definitely light”, “definitely dark”, or “not sure”.

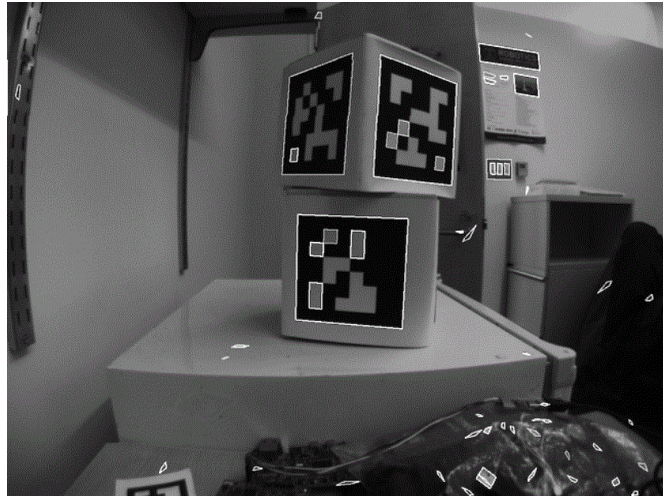
The threshold is calculated by looking at the pixel’s brightness, compared to a small neighborhood of pixels around it.

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 quadrilaterals, 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

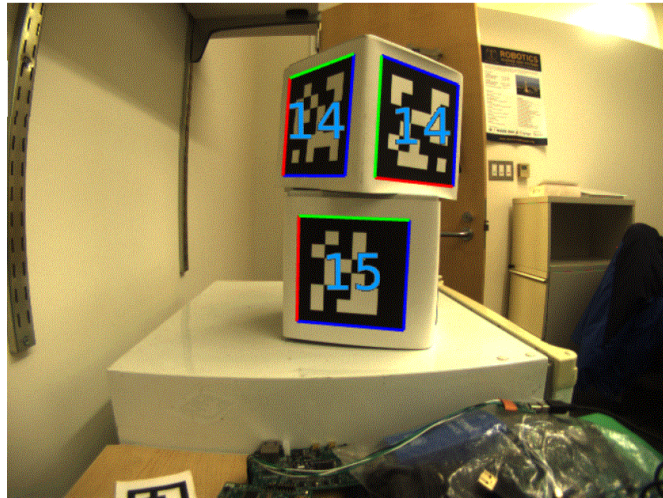


Now that we have one or more regions of pixels which we believe to be a valid AprilTag, we need to identify which tag we are looking at. This is done by “decoding” the pattern of light and dark squares on the inside.

- 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

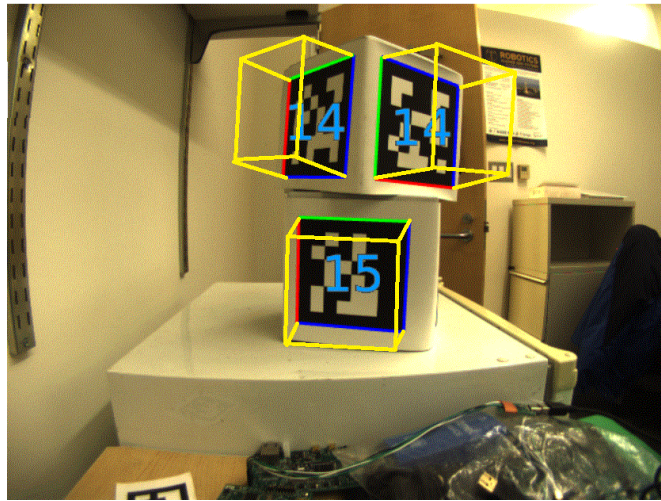
2D Alignment

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 prove 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 lens 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.

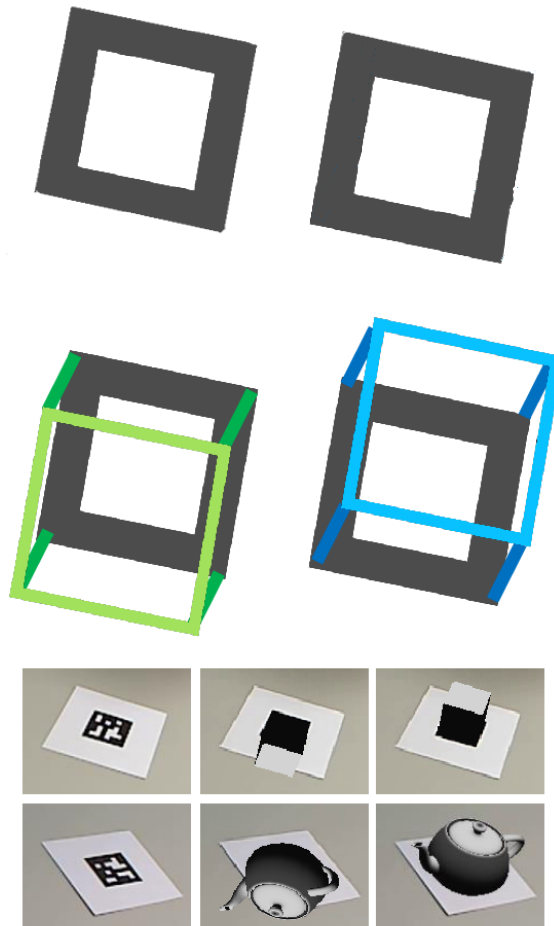
If the camera's position on the robot is known, the robot's position on the field can also be estimated.

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:



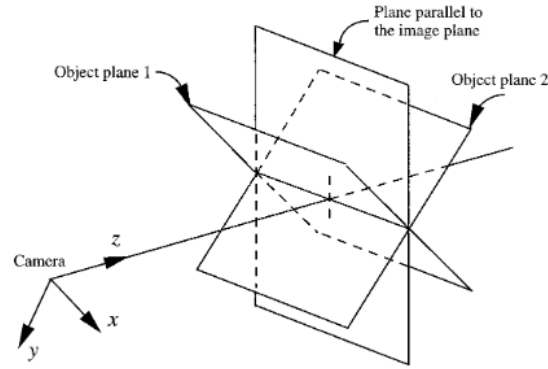


FIG. 4. Two object poses giving the same image under the SOP approximation.

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:

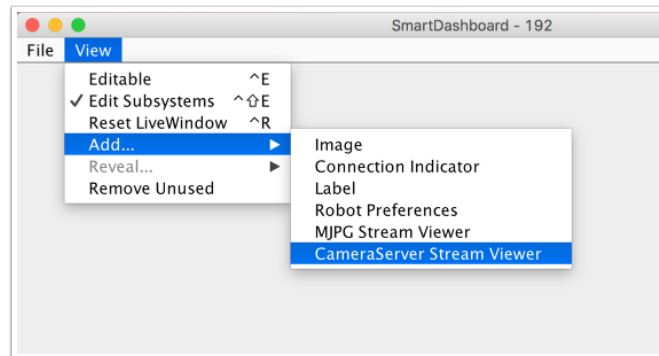
- AprilTags v1
- AprilTags v2
- AprilTags v3
- Pose Ambiguity

25.4 RoboRIO'da Vision (Görüntü işleme)

25.4.1 RoboRIO'da CameraServer'ı kullanma

Basit CameraServer Programı

Aşağıdaki program, roboRIO'ya bağlı Microsoft LifeCam gibi bir USB kameranın otomatik çekimini başlatır. Bu modda kamera, kareleri çekecek ve bunları dashboard'a gönderecektir. Görüntüleri görüntülemek için, dashboard'daki "View-Görünüm" ve ardından "Add-Ekle" menüsünü kullanarak bir CameraServer Stream Viewer widget'ı oluşturun. Görüntüler işlenmemiştir ve sadece kameradan dashboard'a iletilir.



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
12  * send it to the FRC dashboard without doing any vision processing. This is the easiest way to get
13  * camera images 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() {

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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();
    }
};

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

PYTHON

```
1 import wpilib
2 from wpilib.cameraserver import CameraServer
3
4
5 class MyRobot(wpilib.TimedRobot):
6     """
7     Uses the CameraServer class to automatically capture video from a USB webcam and
8     ↪ send it to the
9     ↪ FRC dashboard without doing any vision processing. This is the easiest way to get
10    ↪ camera images
11    ↪ to the dashboard. Just add this to the robotInit() method in your program.
12    """
```

Gelişmiş Camera Server Programı

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

```

7 import edu.wpi.first.cameraserver.CameraServer;
8 import edu.wpi.first.cscore.CvSink;
9 import edu.wpi.first.cscore.CvSource;
10 import edu.wpi.first.cscore.UsbCamera;
11 import edu.wpi.first.wpilibj.TimedRobot;
12 import org.opencv.core.Mat;
13 import org.opencv.core.Point;
14 import org.opencv.core.Scalar;
15 import org.opencv.imgproc.Imgproc;
16
17 /**
18  * This is a demo program showing the use of OpenCV to do vision processing. The
19  * image is acquired
20  * from the USB camera, then a rectangle is put on the image and sent to the
21  * dashboard. OpenCV has
22  * many methods for different types of processing.
23  */
24 public class Robot extends TimedRobot {
25     Thread m_visionThread;
26
27     @Override
28     public void robotInit() {
29         m_visionThread =
30             new Thread(
31                 () -> {
32                     // Get the UsbCamera from CameraServer
33                     UsbCamera camera = CameraServer.startAutomaticCapture();
34                     // Set the resolution
35                     camera.setResolution(640, 480);
36
37                     // Get a CvSink. This will capture Mats from the camera
38                     CvSink cvSink = CameraServer.getVideo();
39                     // Setup a CvSource. This will send images back to the Dashboard
40                     CvSource outputStream = CameraServer.putVideo("Rectangle", 640, 480);
41
42                     // Mats are very memory expensive. Lets reuse this Mat.
43                     Mat mat = new Mat();
44
45                     // This cannot be 'true'. The program will never exit if it is. This
46                     // lets the robot stop this thread when restarting robot code or
47                     // deploying.
48                     while (!Thread.interrupted()) {
49                         // Tell the CvSink to grab a frame from the camera and put it
50                         // in the source mat. If there is an error notify the output.
51                         if (cvSink.grabFrame(mat) == 0) {
52                             // Send the output the error.
53                             outputStream.notifyError(cvSink.getError());
54                             // skip the rest of the current iteration
55                             continue;
56                         }
57                         // Put a rectangle on the image
58                         Imgproc.rectangle(
59                             mat, new Point(100, 100), new Point(400, 400), new Scalar(255,
60                             255, 255), 5);
61                         // Give the output stream a new image to display

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

59         outputStream.putFrame(mat);
60     }
61     });
62     m_visionThread.setDaemon(true);
63     m_visionThread.start();
64 }
65 }

```

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

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

        rectangle(mat, cv::Point(100, 100), cv::Point(400, 400),
                  cv::Scalar(255, 255, 255), 5);
        // Give the output stream a new image to display
        outputStream.PutFrame(mat);
    }
}

void RobotInit() override {
    // We need to run our vision program in a separate thread. If not, our robot
    // program will not run.
    std::thread visionThread(VisionThread);
    visionThread.detach();
}
};

#ifdef 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.

Uyarı: Image processing is a CPU intensive task, and because of the Python Global Interpreter Lock (GIL) **we do NOT recommend using cscore directly in your robot process.** Don't do it. Really.

For more information on the GIL and its effects, you may wish to read the following resources:

- [Python Wiki: Global Interpreter Lock](#)
- [Efficiently Exploiting Multiple Cores with Python](#)

This introduces a number of rules that your image processing code must follow to efficiently and safely run on the RoboRIO:

- Your image processing code must be in its own file. It's easiest to just place it next to your `robot.py`
- Never import the `cscore` package from your robot code, it will just waste memory
- Never import the `wpiLib` or `hal` packages from your image processing file
- The camera code will be killed when the `robot.py` program exits. If you wish to perform cleanup, you should register an `atexit` handler.
- `robotpy-cscore` is not installed on the roboRIO by default, you need to update your `pyproject.toml` file to install it

Uyarı: 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
7     image is acquired
8     from the USB camera, then a rectangle is put on the image and sent to the
9     dashboard. OpenCV has
10    many methods for different types of processing.
11    """
```

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:

```
1 # NOTE: This code runs in its own process, so we cannot access the robot here,
2 #       nor can we create/use/see wpilib objects
3 #
4 # To try this code out locally (if you have robotpy-cscore installed), you
5 # can execute `python3 -m cscore vision.py:main`
6 #
7
8 import cv2
9 import numpy as np
10
11 from cscore import CameraServer as CS
12
13
14 def main():
15     CS.enableLogging()
16
17     # Get the UsbCamera from CameraServer
18     camera = CS.startAutomaticCapture()
19     # Set the resolution
20     camera.setResolution(640, 480)
21
22     # Get a CvSink. This will capture images from the camera
23     cvSink = CS.getVideo()
24     # Setup a CvSource. This will send images back to the Dashboard
25     outputStream = CS.putVideo("Rectangle", 640, 480)
26
27     # Allocating new images is very expensive, always try to preallocate
28     mat = np.zeros(shape=(480, 640, 3), dtype=np.uint8)
29
30     while True:
31         # Tell the CvSink to grab a frame from the camera and put it
32         # in the source image. If there is an error notify the output.
33         time, mat = cvSink.grabFrame(mat)
34         if time == 0:
35             # Send the output the error.
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

36     outputStream.notifyError(cvSink.getError())
37     # skip the rest of the current iteration
38     continue
39
40     # Put a rectangle on the image
41     cv2.rectangle(mat, (100, 100), (400, 400), (255, 255, 255), 5)
42
43     # Give the output stream a new image to display
44     outputStream.putFrame(mat)

```

You need to update `pyproject.toml` contents to include `cscore` in the `robotpy-extras` key (this only shows the portions you need to update):

```

[tool.robotpy]

...

# Add cscore to the robotpy-extras list
robotpy_extras = ["cscore"]

```

Notice that in these examples, the `PutVideo()` method writes the video to a named stream. To view that stream on SmartDashboard or Shuffleboard, select that named stream. In this case that is “Rectangle”.

25.4.2 Birden Fazla Kamera Kullanma

Sürücü Görünümlerini Değiştirme

If you’re interested in just switching what the driver sees, and are using SmartDashboard, the SmartDashboard CameraServer Stream Viewer has an option (“Selected Camera Path”) that reads the given [NetworkTables](#) key and changes the “Camera Choice” to that value (displaying that camera). The robot code then just needs to set the [NetworkTables](#) key to the correct camera name. Assuming “Selected Camera Path” is set to “CameraSelection”, the following code uses the joystick 1 trigger button state to show camera1 and 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

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
public void teleopPeriodic() {
    if (joy1.getTriggerPressed()) {
        System.out.println("Setting camera 2");
        cameraSelection.setString(camera2.getName());
    } else if (joy1.getTriggerReleased()) {
        System.out.println("Setting camera 1");
        cameraSelection.setString(camera1.getName());
    }
}
```

C++

```
cs::UsbCamera camera1;
cs::UsbCamera camera2;
frc::Joystick joy1{0};

nt::NetworkTableEntry cameraSelection;

void RobotInit() override {
    camera1 = frc::CameraServer::StartAutomaticCapture(0);
    camera2 = frc::CameraServer::StartAutomaticCapture(1);

    cameraSelection = nt::NetworkTableInstance::GetDefault().GetTable("")->GetEntry(
        "CameraSelection");
}

void TeleopPeriodic() override {
    if (joy1.GetTriggerPressed()) {
        std::cout << "Setting Camera 2" << std::endl;
        cameraSelection.SetString(camera2.GetName());
    } else if (joy1.GetTriggerReleased()) {
        std::cout << "Setting Camera 1" << std::endl;
        cameraSelection.SetString(camera1.GetName());
    }
}
```

PYTHON

Not: Python requires you to place your image processing code in a separate file from your robot code. You can create `robot.py` and `vision.py` in the same directory.

`robot.py` contents:

```
import wpilib
from ntcore import NetworkTableInstance

class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        self.joy1 = wpilib.Joystick(0)
        self.cameraSelection = NetworkTableInstance.getDefault().getTable("").
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

↪getEntry("CameraSelection")
    wpilib.CameraServer.launch("vision.py:main")

    def teleopPeriodic(self):
        if self.joy1.getTriggerPressed():
            print("Setting camera 2")
            self.cameraSelection.setString("USB Camera 1")
        elif self.joy1.getTriggerReleased():
            print("Setting camera 1")
            self.cameraSelection.setString("USB Camera 0")

```

vision.py contents:

```

from cscore import CameraServer

def main():
    CameraServer.enableLogging()

    camera1 = CameraServer.startAutomaticCapture(0)
    camera2 = CameraServer.startAutomaticCapture(1)

    CameraServer.waitForever()

```

pyproject.toml contents (this only shows the portions you need to update):

```

[tool.robotpy]

...

# Add cscore to the robotpy-extras list
robotpy_extras = ["cscore"]

```

Başka bir gösterge tablosu kullanıyorsanız, kamera sunucusu tarafından kullanılan kamerayı dinamik olarak değiştirebilirsiniz. Bir akış görüntüleyiciyi nominal olarak kamera1'e açarsanız, robot kodu, kumanda kolu tetikleyicisine göre akış içeriğini kamera1 veya kamera2 olarak değiştirir.

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()) {

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
        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
```

Akışları Açık Tutmak

Varsayılan olarak, cscore kitaplığı, kullanılmayan kameraları kapatmada oldukça agresiftir. Bunun anlamı, kameraları değiştirdiğinizde, kullanımda olmayan kamerayla bağlantısının kesilebileceği, bu nedenle geri geçişin kameraya yeniden bağlanırken biraz gecikeceği anlamına gelir. Her iki kamera bağlantısını da açık tutmak için kitaplığa, kullanmıyor olsanız bile akışları açık tutmasını söylemek için `SetConnectionStrategy()` yöntemini kullanın.

Java

```

UsbCamera camera1;
UsbCamera camera2;
VideoSink server;
Joystick joy1 = new Joystick(0);

@Override
public void robotInit() {
    camera1 = CameraServer.startAutomaticCapture(0);
    camera2 = CameraServer.startAutomaticCapture(1);
    server = CameraServer.getServer();

    camera1.setConnectionStrategy(ConnectionStrategy.kKeepOpen);
    camera2.setConnectionStrategy(ConnectionStrategy.kKeepOpen);
}

@Override
public void teleopPeriodic() {
    if (joy1.getTriggerPressed()) {
        System.out.println("Setting camera 2");
        server.setSource(camera2);
    } else if (joy1.getTriggerReleased()) {
        System.out.println("Setting camera 1");
        server.setSource(camera1);
    }
}

```

C++

```

cs::UsbCamera camera1;
cs::UsbCamera camera2;
cs::VideoSink server;
frc::Joystick joy1{0};
bool prevTrigger = false;
void RobotInit() override {
    camera1 = frc::CameraServer::StartAutomaticCapture(0);
    camera2 = frc::CameraServer::StartAutomaticCapture(1);
    server = frc::CameraServer::GetServer();
    camera1.
    ↳SetConnectionStrategy(cs::VideoSource::ConnectionStrategy::kConnectionKeepOpen);
    camera2.
    ↳SetConnectionStrategy(cs::VideoSource::ConnectionStrategy::kConnectionKeepOpen);
}

void TeleopPeriodic() override {
    if (joy1.GetTrigger() && !prevTrigger) {
        std::cout << "Setting Camera 2" << std::endl;
        server.SetSource(camera2);
    } else if (!joy1.GetTrigger() && prevTrigger) {
        std::cout << "Setting Camera 1" << std::endl;
        server.SetSource(camera1);
    }
    prevTrigger = joy1.GetTrigger();
}

```

PYTHON

Not: Python requires you to place your image processing code in a separate file from your robot code. You can create `robot.py` and `vision.py` in the same directory.

`robot.py` contents:

```
import wpilib
from ntcore import NetworkTableInstance

class MyRobot(wpilib.TimedRobot):
    def robotInit(self):
        self.joy1 = wpilib.Joystick(0)
        self.cameraSelection = NetworkTableInstance.getDefault().getTable("").
        ↪getEntry("CameraSelection")
        wpilib.CameraServer.launch("vision.py:main")

    def teleopPeriodic(self):
        if self.joy1.getTriggerPressed():
            print("Setting camera 2")
            self.cameraSelection.setString("USB Camera 1")
        elif self.joy1.getTriggerReleased():
            print("Setting camera 1")
            self.cameraSelection.setString("USB Camera 0")
```

`vision.py` contents:

```
from cscore import CameraServer, VideoSource

def main():
    CameraServer.enableLogging()

    camera1 = CameraServer.startAutomaticCapture(0)
    camera2 = CameraServer.startAutomaticCapture(1)

    camera1.setConnectionStrategy(VideoSource.ConnectionStrategy.kConnectionKeepOpen)
    camera2.setConnectionStrategy(VideoSource.ConnectionStrategy.kConnectionKeepOpen)

    CameraServer.waitForever()
```

`pyproject.toml` contents (this only shows the portions you need to update):

```
[tool.robotpy]

...

# Add cscore to the robotpy-extras list
robotpy_extras = ["cscore"]
```

Not: Her iki kamera da USB ise, daha yüksek çözünürlüklerde USB bant genişliği sınırlamalarıyla karşılaşabilirsiniz, çünkü tüm bu durumlarda roboRIO, her iki kameradan roboRIO'ya aynı anda veri akışı yapacaktır (1 ve 2 seçeneklerinde kısa bir süre için ve sürekli seçenek 3). Kitaplığın bu eşzamanlılıktan kaçınması seçenek 2 durumunda (sadece) teorik olarak mümkündür, ancak bu şu anda uygulanmamaktadır.

Farklı kameralar bant genişliği kullanımını farklı şekilde rapor eder. Kütüphane, sınıra ulaşım ulaşmadığınızı size söyleyecektir; bu hata mesajını alacaksınız:

```
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)
```

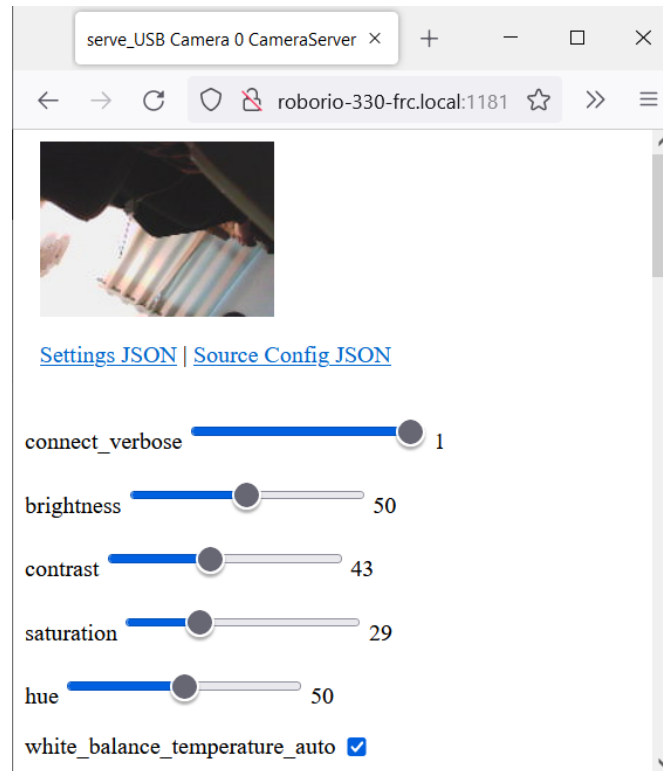
If you're using Option 3 it will give you this error during `RobotInit()`. Thus you should just try your desired resolution and adjusting as necessary until you both don't get that error and don't exceed the radio bandwidth limitations.

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 than *Basit CameraServer Programı*.

Not: 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.

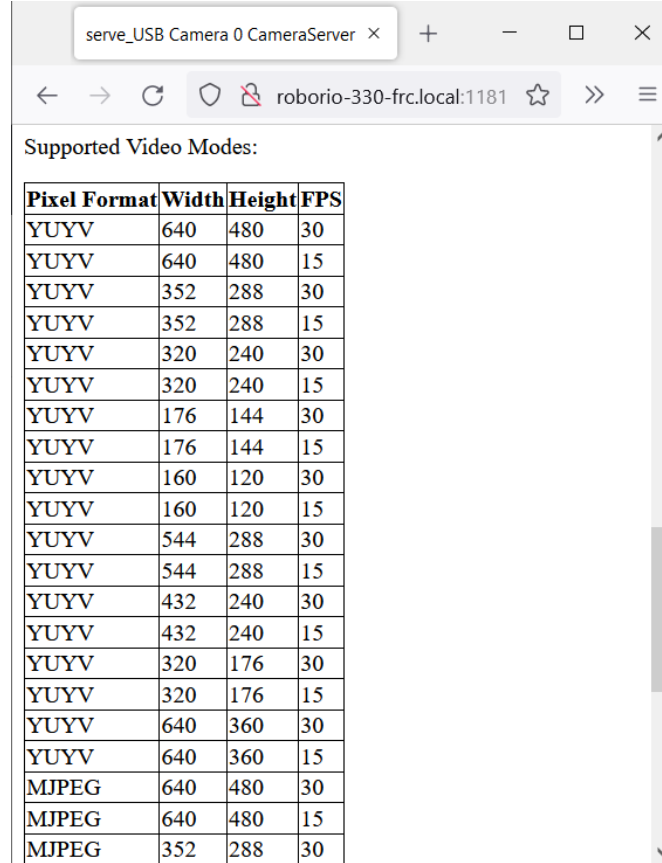
Camera Settings



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



The screenshot shows a web browser window with the title 'serve_USB Camera 0 CameraServer'. The address bar displays 'roborio-330-frc.local:1181'. The main content area is titled 'Supported Video Modes:' and contains a table with the following data:

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.

Komut Tabanlı Programlama

Bu makale dizisi, WPILib komut tabanlı framework'e bir giriş ve referans görevi görür.

Komut tabanlı framework kullanılan örnek projelerin bir koleksiyonu için bakınız: [Komut-Tabanlı Örnekler](#).

26.1 Komut Tabanlı Programlama nedir?

WPILib supports a robot programming methodology called “command-based” programming. In general, “command-based” can refer both the general programming paradigm, and to the set of WPILib library resources included to facilitate it.

“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.

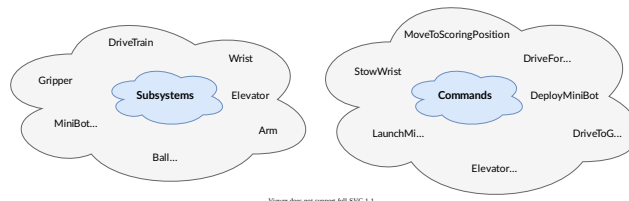
JAVA

```
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 Subsystems ve Commands



Komut tabanlı model, iki temel betimlemeye dayanmaktadır: **komutlar-commands** ve **alt sistemler-subsystems**.

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 [Komutlar](#) 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 Komutlar Nasıl Çalıştırılır ?

Not: Daha fazla bilgi için, bakınız *Komut Zamanlayıcı - 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 Komutlar

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 Bir Komutun Yapısı

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).

Initialization - Başlatma

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.

Execution - Yürütme

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.

Ending - Bitirme

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()`.

Bitiş-end koşullarını belirleme

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 effects.

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++

```
fr2::cmd::Run([&intake] { intake.Activate(); }, {@amp;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.

Önemli: 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`.

Not: 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 noninterruptible = 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 constructed 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)

```

15  frc2::CommandPtr HatchSubsystem::GrabHatchCommand() {
16      // implicitly require `this`
17      return this->RunOnce(
18          [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kForward); });
19  }
20
21  frc2::CommandPtr HatchSubsystem::ReleaseHatchCommand() {
22      // implicitly require `this`
23      return this->RunOnce(
24          [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kReverse); });
25  }

```

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 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,
    // Requires the drive subsystem
    m_robotDrive
)
```

C++

```
frc2::FunctionalCommand(
    // Reset encoders on command start
    [this] { m_drive.ResetEncoders(); },
    // Start driving forward at the start of the command
    [this] { m_drive.ArcadeDrive(ac::kAutoDriveSpeed, 0); },
    // Stop driving at the end of the command
    [this] (bool interrupted) { m_drive.ArcadeDrive(0, 0); },
    // End the command when the robot's driven distance exceeds the desired value
    [this] { return m_drive.GetAverageEncoderDistance() >= kAutoDriveDistanceInches; },
    // Requires the drive subsystem
    {&m_drive}
)
```

To print a string and ending immediately, the library offers the `Commands.print(String)/frc2::cmd::Print(std::string_view)` factory, backed by the `PrintCommand` (Java, C++) subclass of `InstantCommand`.

Waiting

Waiting for a certain condition to happen or adding a delay can be useful to synchronize between different commands in a command composition or between other robot actions.

To wait and end after a specified period of time elapses, the library offers the `Commands.waitSeconds(double)/frc2::cmd::Wait(units::second_t)` factory, backed by the `WaitCommand` (Java, C++) class.

JAVA

```
// Ends 5 seconds after being scheduled  
new WaitCommand(5.0)
```

C++

```
// Ends 5 seconds after being scheduled  
frc2::WaitCommand(5.0_s)
```

To wait until a certain condition becomes true, the library offers the `Commands.waitUntil(BooleanSupplier)/frc2::cmd::WaitUntil(std::function<bool()>)` factory, backed by the `WaitUntilCommand` class (Java, C++).

JAVA

```
// Ends after m_limitSwitch.get() returns true  
new WaitUntilCommand(m_limitSwitch::get)
```

C++

```
// Ends after m_limitSwitch.Get() returns true  
frc2::WaitUntilCommand([&m_limitSwitch] { return m_limitSwitch.Get(); })
```

Control Algorithm Commands

There are commands for various control setups:

- `PIDCommand` uses a PID controller. For more info, see [PIDCommand](#).
- `TrapezoidProfileCommand` tracks a trapezoid motion profile. For more info, see [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.

Not: 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

```

7 import edu.wpi.first.wpilibj.templates.commandbased.subsystems.ExampleSubsystem;
8 import edu.wpi.first.wpilibj2.command.Command;
9
10 /** An example command that uses an example subsystem. */
11 public class ExampleCommand extends Command {
12     @SuppressWarnings({"PMD.UnusedPrivateField", "PMD.SingularField"})
13     private final ExampleSubsystem m_subsystem;
14
15     /**
16      * Creates a new ExampleCommand.
17      *
18      * @param subsystem The subsystem used by this command.
19      */
20     public ExampleCommand(ExampleSubsystem subsystem) {
21         m_subsystem = subsystem;
22         // Use addRequirements() here to declare subsystem dependencies.
23         addRequirements(subsystem);
24     }

```

C++

```

5 #pragma once
6
7 #include <frc2/command/Command.h>
8 #include <frc2/command/CommandHelper.h>
9
10 #include "subsystems/ExampleSubsystem.h"
11
12 /**
13  * An example command that uses an example subsystem.
14  *
15  * <p>Note that this extends CommandHelper, rather extending Command
16  * directly; this is crucially important, or else the decorator functions in
17  * Command will *not* work!
18  */
19 class ExampleCommand
20     : public frc2::CommandHelper<frc2::Command, ExampleCommand> {
21 public:

```

(sonraki sayfaya devam)

```

22  /**
23   * Creates a new ExampleCommand.
24   *
25   * @param subsystem The subsystem used by this command.
26   */
27  explicit ExampleCommand(ExampleSubsystem* subsystem);
28
29  private:
30      ExampleSubsystem* m_subsystem;
31  };

```

26.2.5 Basit Komut Örneği

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

```

5  package edu.wpi.first.wpilibj.examples.hatchbottraditional.commands;
6
7  import edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems.
   ↳ HatchSubsystem;
8  import edu.wpi.first.wpilibj2.command.Command;
9
10 /**
11  * A simple command that grabs a hatch with the {@link HatchSubsystem}.
   ↳ Written explicitly for
12  * pedagogical purposes. Actual code should inline a command this simple.
   ↳ with {@link
13  * edu.wpi.first.wpilibj2.command.InstantCommand}.
14  */
15 public class GrabHatch extends Command {
16     // The subsystem the command runs on
17     private final HatchSubsystem m_hatchSubsystem;
18
19     public GrabHatch(HatchSubsystem subsystem) {
20         m_hatchSubsystem = subsystem;
21         addRequirements(m_hatchSubsystem);
22     }
23
24     @Override
25     public void initialize() {
26         m_hatchSubsystem.grabHatch();
27     }
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.

Daha karmaşık bir vakaya ne dersiniz? Aşağıda aynı örnek projeden bir sürücü komutu verilmiştir:

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)

```

5 #pragma once
6
7 #include <functional>
8
9 #include <frc2/command/Command.h>
10 #include <frc2/command/CommandHelper.h>
11
12 #include "subsystems/DriveSubsystem.h"
13
14 /**
15  * A command to drive the robot with joystick input passed in through

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

16  ↪ lambdas.
17  * Written explicitly for pedagogical purposes - actual code should inline a
18  * command this simple with RunCommand.
19  *
20  * @see RunCommand
21  */
22  class DefaultDrive : public frc2::CommandHelper<frc2::Command, DefaultDrive>
23  ↪ {
24  public:
25  /**
26   * Creates a new DefaultDrive.
27   *
28   * @param subsystem The drive subsystem this command wil run on.
29   * @param forward The control input for driving forwards/backwards
30   * @param rotation The control input for turning
31   */
32  DefaultDrive(DriveSubsystem* subsystem, std::function<double()> forward,
33              std::function<double()> rotation);
34
35  void Execute() override;
36
37  private:
38  DriveSubsystem* m_drive;
39  std::function<double()> m_forward;
40  std::function<double()> m_rotation;
41  };

```

C++ (Source)

```

5  #include "commands/DefaultDrive.h"
6
7  #include <utility>
8
9  DefaultDrive::DefaultDrive(DriveSubsystem* subsystem,
10                             std::function<double()> forward,
11                             std::function<double()> rotation)
12      : m_drive{subsystem},
13        m_forward{std::move(forward)},
14        m_rotation{std::move(rotation)} {
15      AddRequirements(subsystem);
16  }
17
18  void DefaultDrive::Execute() {
19      m_drive->ArcadeDrive(m_forward(), m_rotation());
20  }

```

And then usage:

JAVA

```
59 // Configure default commands
60 // Set the default drive command to split-stick arcade drive
61 m_robotDrive.setDefaultCommand(
62     // A split-stick arcade command, with forward/backward controlled by the left
63     // hand, and turning controlled by the right.
64     new DefaultDrive(
65         m_robotDrive,
66         () -> -m_driverController.getLeftY(),
67         () -> -m_driverController.getRightX()));
```

C++

```
57 // Set up default drive command
58 m_drive.SetDefaultCommand(DefaultDrive(
59     &m_drive, [this] { return -m_driverController.GetLeftY(); },
60     [this] { return -m_driverController.GetRightX(); }));
```

Notice that this command does not override `isFinished()`, and thus will never end; this is the norm for commands that are intended to be used as default commands. Once more, this command is rather simple and calls the subsystem method only from one place, and as such, could be more concisely written using factories:

JAVA

```
51 // Configure default commands
52 // Set the default drive command to split-stick arcade drive
53 m_robotDrive.setDefaultCommand(
54     // A split-stick arcade command, with forward/backward controlled by the left
55     // hand, and turning controlled by the right.
56     Commands.run(
57         () ->
58             m_robotDrive.arcadeDrive(
59                 -m_driverController.getLeftY(), -m_driverController.getRightX()),
60         m_robotDrive));
```

C++

```
52 // Set up default drive command
53 m_drive.SetDefaultCommand(frc2::cmd::Run(
54     [this] {
55         m_drive.ArcadeDrive(-m_driverController.GetLeftY(),
56                             -m_driverController.GetRightX());
57     },
58     {&m_drive}));
```

26.3 Komut Kompozisyonları

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 `runWhenDisabled` 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.

Önemli: 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 command 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

```
20 public class RobotContainer {
21     // The enum used as keys for selecting the command to run.
22     private enum CommandSelector {
23         ONE,
24         TWO,
25         THREE
26     }
27
28     // An example selector method for the selectcommand. Returns the selector that
29     // will select
30     // which command to run. Can base this choice on logical conditions evaluated at
31     // runtime.
32     private CommandSelector select() {
33         return CommandSelector.ONE;
34     }
35
36     // An example selectcommand. Will select from the three commands based on the
37     // value returned
38     // by the selector method at runtime. Note that selectcommand works on Object(),
39     // so the
40     // selector does not have to be an enum; it could be any desired type (string,
41     // integer,
42     // boolean, double...)
43     private final Command m_exampleSelectCommand =
44         new SelectCommand<>{
45             // Maps selector values to commands
46             Map.ofEntries(
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
42         Map.entry(CommandSelector.ONE, new PrintCommand("Command one was
↪selected!")),
43         Map.entry(CommandSelector.TWO, new PrintCommand("Command two was
↪selected!")),
44         Map.entry(CommandSelector.THREE, new PrintCommand("Command three was
↪selected!"))),
45         this::select);
```

C++ (Header)

```
26 // The enum used as keys for selecting the command to run.
27 enum CommandSelector { ONE, TWO, THREE };
28
29 // An example of how command selector may be used with SendableChooser
30 frc::SendableChooser<CommandSelector> m_chooser;
31
32 // The robot's subsystems and commands are defined here...
33
34 // An example selectcommand. Will select from the three commands based on the
35 // value returned by the selector method at runtime. Note that selectcommand
36 // takes a generic type, so the selector does not have to be an enum; it could
37 // be any desired type (string, integer, boolean, double...)
38 frc2::CommandPtr m_exampleSelectCommand = frc2::cmd::Select<CommandSelector>(
39     [this] { return m_chooser.GetSelected(); },
40     // Maps selector values to commands
41     std::pair{ONE, frc2::cmd::Print("Command one was selected!")},
42     std::pair{TWO, frc2::cmd::Print("Command two was selected!")},
43     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 components' 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.

Uyarı: 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.

Not: 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

```

5 package edu.wpi.first.wpilibj.examples.hatchbottraditional.commands;
6
7 import edu.wpi.first.wpilibj.examples.hatchbottraditional.Constants.AutoConstants;
8 import edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems.DriveSubsystem;
9 import edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems.HatchSubsystem;
10 import edu.wpi.first.wpilibj2.command.SequentialCommandGroup;
11
12 /** A complex auto command that drives forward, releases a hatch, and then drives
13     ↪ backward. */
14 public class ComplexAuto extends SequentialCommandGroup {
15     /**
16      * Creates a new ComplexAuto.
17      *
18      * @param drive The drive subsystem this command will run on
19      * @param hatch The hatch subsystem this command will run on
20      */
21     public ComplexAuto(DriveSubsystem drive, HatchSubsystem hatch) {
22         addCommands(
23             // Drive forward the specified distance
24             new DriveDistance(
25                 AutoConstants.kAutoDriveDistanceInches, AutoConstants.kAutoDriveSpeed,
26                 ↪ drive),
27
28             // Release the hatch
29             new ReleaseHatch(hatch),
30
31             // Drive backward the specified distance
32             new DriveDistance(
33                 AutoConstants.kAutoBackupDistanceInches, -AutoConstants.kAutoDriveSpeed,
34                 ↪ drive));
35     }
36 }

```

C++ (Header)

```

5 #pragma once
6
7 #include <frc2/command/CommandHelper.h>
8 #include <frc2/command/SequentialCommandGroup.h>
9
10 #include "Constants.h"
11 #include "commands/DriveDistance.h"
12 #include "commands/ReleaseHatch.h"
13
14 /**
15  * A complex auto command that drives forward, releases a hatch, and then drives
16  * backward.
17  */
18 class ComplexAuto
19     : public frc2::CommandHelper<frc2::SequentialCommandGroup, ComplexAuto> {
20 public:
21     /**
22      * Creates a new ComplexAuto.

```

(sonraki sayfaya devam)


```

23  *
24  * @param drive The drive subsystem this command will run on
25  * @param hatch The hatch subsystem this command will run on
26  */
27  ComplexAuto(DriveSubsystem* drive, HatchSubsystem* hatch);
28  };

```

C++ (Source)

```

5  #include "commands/ComplexAuto.h"
6
7  using namespace AutoConstants;
8
9  ComplexAuto::ComplexAuto(DriveSubsystem* drive, HatchSubsystem* hatch) {
10     AddCommands(
11         // Drive forward the specified distance
12         DriveDistance(kAutoDriveDistanceInches, kAutoDriveSpeed, drive),
13         // Release the hatch
14         ReleaseHatch(hatch),
15         // Drive backward the specified distance
16         DriveDistance(kAutoBackupDistanceInches, -kAutoDriveSpeed, drive));
17     }

```

The advantages and disadvantages of this subclassing approach in comparison to others are discussed in [Subclassing Command Groups](#).

26.4 Subsystems - Alt sistemler

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 Bir Subsystem - Alt Sistem Oluşturma

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

```

7  import edu.wpi.first.wpilibj2.command.Command;
8  import edu.wpi.first.wpilibj2.command.SubsystemBase;
9
10 public class ExampleSubsystem extends SubsystemBase {
11     /** Creates a new ExampleSubsystem. */
12     public ExampleSubsystem() {}
13
14     /**
15      * Example command factory method.
16      *
17      * @return a command
18      */
19     public Command exampleMethodCommand() {
20         // Inline construction of command goes here.
21         // Subsystem::RunOnce implicitly requires `this` subsystem.
22         return runOnce(
23             () -> {
24                 /* one-time action goes here */
25             });
26     }
27
28     /**
29      * An example method querying a boolean state of the subsystem (for example, a
30      * digital sensor).
31      *
32      * @return value of some boolean subsystem state, such as a digital sensor.
33      */
34     public boolean exampleCondition() {
35         // Query some boolean state, such as a digital sensor.
36         return false;
37     }
38
39     @Override
40     public void periodic() {
41         // This method will be called once per scheduler run
42     }
43
44     @Override
45     public void simulationPeriodic() {
46         // This method will be called once per scheduler run during simulation
47     }
48 }

```

C++

```

5  #pragma once
6
7  #include <frc2/command/CommandPtr.h>
8  #include <frc2/command/SubsystemBase.h>
9
10 class ExampleSubsystem : public frc2::SubsystemBase {
11 public:
12     ExampleSubsystem();
13
14     /**
15      * Example command factory method.
16      */
17     frc2::CommandPtr ExampleMethodCommand();
18
19     /**
20      * An example method querying a boolean state of the subsystem (for example, a
21      * digital sensor).
22      *
23      * @return value of some boolean subsystem state, such as a digital sensor.
24      */
25     bool ExampleCondition();
26
27     /**
28      * Will be called periodically whenever the CommandScheduler runs.
29      */
30     void Periodic() override;
31
32     /**
33      * Will be called periodically whenever the CommandScheduler runs during
34      * simulation.
35      */
36     void SimulationPeriodic() override;
37
38 private:
39     // Components (e.g. motor controllers and sensors) should generally be
40     // declared private and exposed only through public methods.
41 };

```

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 Basit Subsystem Örneği

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

```

5 package edu.wpi.first.wpilibj.examples.hatchbottraditional.subsystems;
6
7 import static edu.wpi.first.wpilibj.DoubleSolenoid.Value.kForward;
8 import static edu.wpi.first.wpilibj.DoubleSolenoid.Value.kReverse;
9
10 import edu.wpi.first.util.sendable.SendableBuilder;
11 import edu.wpi.first.wpilibj.DoubleSolenoid;
12 import edu.wpi.first.wpilibj.PneumaticsModuleType;
13 import edu.wpi.first.wpilibj.examples.hatchbottraditional.Constants.HatchConstants;
14 import edu.wpi.first.wpilibj2.command.SubsystemBase;
15
16 /** A hatch mechanism actuated by a single {@link DoubleSolenoid}. */
17 public class HatchSubsystem extends SubsystemBase {
18     private final DoubleSolenoid m_hatchSolenoid =
19         new DoubleSolenoid(
20             PneumaticsModuleType.CTREPCM,
21             HatchConstants.kHatchSolenoidPorts[0],
22             HatchConstants.kHatchSolenoidPorts[1]);
23
24     /** Grabs the hatch. */
25     public void grabHatch() {
26         m_hatchSolenoid.set(kForward);
27     }
28
29     /** Releases the hatch. */
30     public void releaseHatch() {
31         m_hatchSolenoid.set(kReverse);
32     }
33
34     @Override
35     public void initSendable(SendableBuilder builder) {
36         super.initSendable(builder);
37         // Publish the solenoid state to telemetry.
38         builder.addBooleanProperty("extended", () -> m_hatchSolenoid.get() == kForward,
39             null);
40     }
41 }

```

C++ (Header)

```

5  #pragma once
6
7  #include <frc/DoubleSolenoid.h>
8  #include <frc/PneumaticsControlModule.h>
9  #include <frc2/command/SubsystemBase.h>
10
11 #include "Constants.h"
12
13 class HatchSubsystem : public frc2::SubsystemBase {
14 public:
15     HatchSubsystem();
16
17     // Subsystem methods go here.
18
19     /**
20      * Grabs the hatch.
21      */
22     void GrabHatch();
23
24     /**
25      * Releases the hatch.
26      */
27     void ReleaseHatch();
28
29     void InitSendable(wpi::SendableBuilder& builder) override;
30
31 private:
32     // Components (e.g. motor controllers and sensors) should generally be
33     // declared private and exposed only through public methods.
34     frc::DoubleSolenoid m_hatchSolenoid;
35 };

```

C++ (Source)

```

5  #include "subsystems/HatchSubsystem.h"
6
7  #include <wpi/sendable/SendableBuilder.h>
8
9  using namespace HatchConstants;
10
11 HatchSubsystem::HatchSubsystem()
12     : m_hatchSolenoid{frc::PneumaticsModuleType::CTREPCM,
13                     kHatchSolenoidPorts[0], kHatchSolenoidPorts[1]} {}
14
15 void HatchSubsystem::GrabHatch() {
16     m_hatchSolenoid.Set(frc::DoubleSolenoid::kForward);
17 }
18
19 void HatchSubsystem::ReleaseHatch() {
20     m_hatchSolenoid.Set(frc::DoubleSolenoid::kReverse);
21 }
22
23 void HatchSubsystem::InitSendable(wpi::SendableBuilder& builder) {

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

24 SubsystemBase::InitSendable(builder);
25
26 // Publish the solenoid state to telemetry.
27 builder.AddBooleanProperty(
28     "extended",
29     [this] { return m_hatchSolenoid.Get() == frc::DoubleSolenoid::kForward; },
30     nullptr);
31 }

```

Notice that the subsystem hides the presence of the DoubleSolenoid from outside code (it is declared private), and instead publicly exposes two higher-level, descriptive robot actions: grabHatch() and releaseHatch(). It is extremely important that “implementation details” such as the double solenoid be “hidden” in this manner; this ensures that code outside the subsystem will never cause the solenoid to be in an unexpected state. It also allows the user to change the implementation (for instance, a motor could be used instead of a pneumatic) without any of the code outside of the subsystem having to change with it.

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

```

5 package edu.wpi.first.wpilibj.examples.hatchbotinlined.subsystems;
6
7 import static edu.wpi.first.wpilibj.DoubleSolenoid.Value.kForward;
8 import static edu.wpi.first.wpilibj.DoubleSolenoid.Value.kReverse;
9
10 import edu.wpi.first.util.sendable.SendableBuilder;
11 import edu.wpi.first.wpilibj.DoubleSolenoid;
12 import edu.wpi.first.wpilibj.PneumaticsModuleType;
13 import edu.wpi.first.wpilibj.examples.hatchbotinlined.Constants.HatchConstants;
14 import edu.wpi.first.wpilibj2.command.Command;
15 import edu.wpi.first.wpilibj2.command.SubsystemBase;
16
17 /** A hatch mechanism actuated by a single {@link edu.wpi.first.wpilibj.
18     ↳DoubleSolenoid}. */
19 public class HatchSubsystem extends SubsystemBase {
20     private final DoubleSolenoid m_hatchSolenoid =
21         new DoubleSolenoid(
22             PneumaticsModuleType.CTREPCM,
23             HatchConstants.kHatchSolenoidPorts[0],
24             HatchConstants.kHatchSolenoidPorts[1]);
25
26     /** Grabs the hatch. */
27     public Command grabHatchCommand() {
28         // implicitly require `this`
29         return this.runOnce(() -> m_hatchSolenoid.set(kForward));
30     }
31
32     /** Releases the hatch. */
33     public Command releaseHatchCommand() {
34         // implicitly require `this`
35         return this.runOnce(() -> m_hatchSolenoid.set(kReverse));
36     }
37 }

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

35     }
36
37     @Override
38     public void initSendable(SendableBuilder builder) {
39         super.initSendable(builder);
40         // Publish the solenoid state to telemetry.
41         builder.addBooleanProperty("extended", () -> m_hatchSolenoid.get() == kForward,
42         ↪ null);
43     }
44 }

```

C++ (Header)

```

5  #pragma once
6
7  #include <frc/DoubleSolenoid.h>
8  #include <frc/PneumaticsControlModule.h>
9  #include <frc2/command/CommandPtr.h>
10 #include <frc2/command/SubsystemBase.h>
11
12 #include "Constants.h"
13
14 class HatchSubsystem : public frc2::SubsystemBase {
15 public:
16     HatchSubsystem();
17
18     // Subsystem methods go here.
19
20     /**
21      * Grabs the hatch.
22      */
23     frc2::CommandPtr GrabHatchCommand();
24
25     /**
26      * Releases the hatch.
27      */
28     frc2::CommandPtr ReleaseHatchCommand();
29
30     void InitSendable(wpi::SendableBuilder& builder) override;
31
32 private:
33     // Components (e.g. motor controllers and sensors) should generally be
34     // declared private and exposed only through public methods.
35     frc::DoubleSolenoid m_hatchSolenoid;
36 };

```

C++ (Source)

```

5  #include "subsystems/HatchSubsystem.h"
6
7  #include <wpi/sendable/SendableBuilder.h>
8
9  using namespace HatchConstants;
10
11  HatchSubsystem::HatchSubsystem()
12      : m_hatchSolenoid{frc::PneumaticsModuleType::CTREPCM,
13                      kHatchSolenoidPorts[0], kHatchSolenoidPorts[1]} {}
14
15  frc2::CommandPtr HatchSubsystem::GrabHatchCommand() {
16      // implicitly require `this`
17      return this->RunOnce(
18          [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kForward); });
19  }
20
21  frc2::CommandPtr HatchSubsystem::ReleaseHatchCommand() {
22      // implicitly require `this`
23      return this->RunOnce(
24          [this] { m_hatchSolenoid.Set(frc::DoubleSolenoid::kReverse); });
25  }
26
27  void HatchSubsystem::InitSendable(wpi::SendableBuilder& builder) {
28      SubsystemBase::InitSendable(builder);
29
30      // Publish the solenoid state to telemetry.
31      builder.AddBooleanProperty(
32          "extended",
33          [this] { return m_hatchSolenoid.Get() == frc::DoubleSolenoid::kForward; },
34          nullptr);
35  }

```

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

```
117 @Override
118 public void periodic() {
119     // Update the odometry in the periodic block
120     m_odometry.update(
121         Rotation2d.fromDegrees(getHeading()),
122         m_leftEncoder.getDistance(),
123         m_rightEncoder.getDistance());
124     m_fieldSim.setRobotPose(getPose());
125 }
```

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

Not: 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));
```

Not: A command that is assigned as the default command for a subsystem must require that subsystem.

26.5 Komutları Trigger-Tetikleyicilere Bağlama

Otonom dönemin başlangıcında programlanan otonom komutlar ve alt sistemleri halihazırda kullanımda olmadığında otomatik olarak programlanan varsayılan komutlar dışında, bir komutu çalıştırmanın en yaygın yolu onu tetikleyici bir olaya bağlamaktır. Bir insan operatörün bir düğmeye basması gibi. Komut tabanlı paradigma, bunu yapmayı son derece kolaylaştırır.

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

Not: 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.

Not: The Button subclass is deprecated, and usage of its binding methods should be replaced according to the respective deprecation messages in the API docs.

onTrue

This binding schedules a command when a trigger changes from false to true (or, accordingly, when a button changes is initially pressed). The command will be scheduled on the iteration when the state changes, and will not be scheduled again unless the trigger becomes false and then true again (or the button is released and then re-pressed).

JAVA

```
52 // Move the arm to 2 radians above horizontal when the 'A' button is pressed.  
53 m_driverController.a().onTrue(m_robotArm.setArmGoalCommand(2));
```

C++

```
25 // Move the arm to 2 radians above horizontal when the 'A' button is pressed.  
26 m_driverController.A().OnTrue(m_arm.SetArmGoalCommand(2_rad));
```

The onFalse binding is identical, only that it schedules on false instead of on true.

whileTrue

This binding schedules a command when a trigger changes from false to true (or, accordingly, when a button is initially pressed) and cancels it when the trigger becomes false again (or the button is released). The command will *not* be re-scheduled if it finishes while the trigger is still true. For the command to restart if it finishes while the trigger is true, wrap the command in a RepeatCommand, or use a RunCommand instead of an InstantCommand.

JAVA

```
114 // While holding the shoulder button, drive at half speed  
115 new JoystickButton(m_driverController, Button.kRightBumper.value)  
116 .whileTrue(new HalveDriveSpeed(m_robotDrive));
```

C++

```
75 // While holding the shoulder button, drive at half speed  
76 frc2::JoystickButton(&m_driverController,  
77                     frc::XboxController::Button::kRightBumper)  
78 .WhileTrue(HalveDriveSpeed(&m_drive).ToPtr());
```

The whileFalse binding is identical, only that it schedules on false and cancels on true.

toggleOnTrue

This binding toggles a command, scheduling it when a trigger changes from false to true (or a button is initially pressed), and canceling it under the same condition if the command is currently running. Note that while this functionality is supported, toggles are not a highly-recommended option for user control, as they require the driver to keep track of the robot state. The preferred method is to use two buttons; one to turn on and another to turn off. Using a [StartEndCommand](#) or a [ConditionalCommand](#) is a good way to specify the commands that you want to be 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 Tetikleyicileri Oluşturma

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 Komut Tabanlı Bir Robot Projesi Yapılandırma

Kullanıcılar komut tabanlı kütüphaneleri istedikleri gibi kullanmakta özgür olduklarında (ki ileri düzey kullanıcılar bunu yapmaya teşvik edilir), yeni kullanıcılar temel komut tabanlı bir robotun nasıl yapılandırılacağı konusunda yardım isteyebilir.

Komut tabanlı bir robot projesi için standart bir şablon WPILib örnek havuzunda bulunur (Java <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/ten_>, C++ <https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/templates/com_>). Bu bölüm, kullanıcılara bu şablonun yapısı boyunca yol gösterecektir.

Root paket /dizin genellikle 4 sınıf içerir:

“Main”, robotun ana uygulamasıdır (yalnızca Java). Yeni kullanıcılar kesinlikle bu sınıfa dokunmamalıdır. “Robot”, robotun kodunun ana kontrol akışından sorumludur. RobotContainer, robot alt sistemlerini ve komutlarını tutar, ve açıklayıcı robot kurulumunun çoğunun (örneğin düğme bağlamaları) yapıldığı yerdir. “Constants”, robotun tamamında kullanılacak olan ulusal erişilebilir sabitler tutar.

root dizini ayrıca iki alt-paket / alt-dizin içerir: “Subsystems”, tüm kullanıcı tanımlı alt sistem sınıflarını içerir. “Commands”, tüm kullanıcı tanımlı komut sınıflarını içerir.

26.6.1 Robot

Robot (Java, C++ (Header) <https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/_.>, C++ (Source) <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/templates/commandbased/cpp/Robot.cpp>>”) programın kontrol akışından sorumlu olduğundan ve komut tabanlı, en aza indirmek için tasarlanmış bir bildirimsel paradigmadır. kullanıcının açık program kontrol akışına göstermesi gereken dikkat miktarı, komut tabanlı bir projenin Robot sınıfı çoğunlukla boş olmalıdır. Ancak dahil edilmesi gereken birkaç önemli şey var

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   }
```

Java'da, “RobotContainer”'ın bir örneği robotInit() yöntemi sırasında oluşturulmalıdır.-Bu çok önemlidir, çünkü açıklayıcı robot kurulumunun çoğu “RobotContainer” kurucusu tarafından çağırılacaktır.

C++ da, RobotContainer bir değer üyesi olduğuiçin ve “Robot”'un yapımı sırasında yapılacağı için buna gerek yoktur.

Java

```

33  /**
34  * This function is called every 20 ms, no matter the mode. Use this for items like
↪ diagnostics
35  * that you want ran during disabled, autonomous, teleoperated and test.
36  *
37  * <p>This runs after the mode specific periodic functions, but before LiveWindow
↪ and
38  * SmartDashboard integrated updating.
39  */
40  @Override
41  public void robotPeriodic() {
42  // Runs the Scheduler. This is responsible for polling buttons, adding newly-
↪ scheduled
43  // commands, running already-scheduled commands, removing finished or interrupted
↪ commands,
44  // and running subsystem periodic() methods. This must be called from the robot
↪ 's periodic
45  // block in order for anything in the Command-based framework to work.
46  CommandScheduler.getInstance().run();
47  }

```

C++ (Source)

```

11  /**
12  * This function is called every 20 ms, no matter the mode. Use
13  * this for items like diagnostics that you want to run during disabled,
14  * autonomous, teleoperated and test.
15  *
16  * <p> This runs after the mode specific periodic functions, but before
17  * LiveWindow and SmartDashboard integrated updating.
18  */
19  void Robot::RobotPeriodic() {
20  frc2::CommandScheduler::GetInstance().Run();
21  }

```

`` RobotPeriodic () `` yöntemine `` CommandScheduler.getInstance (). Run () `` çağrısının dahil edilmesi önemlidir; bu çağrı olmadan, zamanlayıcı programlanmış herhangi bir komutu yürütmeyecektir. `` TimedRobot `` varsayılan ana döngü frekansı 50Hz ile çalıştığı için bu, periyodik komut ve alt sistem yöntemlerinin çağrılacağı frekanstır. Yeni kullanıcıların bu yöntemi kodlarının herhangi bir yerinden çağrımları önerilmez.

Java

```

56  /** This autonomous runs the autonomous command selected by your {@link
↪ RobotContainer} class. */
57  @Override
58  public void autonomousInit() {
59  m_autonomousCommand = m_robotContainer.getAutonomousCommand();
60
61  // schedule the autonomous command (example)
62  if (m_autonomousCommand != null) {

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

63     m_autonomousCommand.schedule();
64 }
65 }

```

C++ (Source)

```

33 /**
34  * This autonomous runs the autonomous command selected by your {@link
35  * RobotContainer} class.
36  */
37 void Robot::AutonomousInit() {
38     m_autonomousCommand = m_container.GetAutonomousCommand();
39
40     if (m_autonomousCommand) {
41         m_autonomousCommand->Schedule();
42     }
43 }

```

autonomousInit()" yöntemi, ``RobotContainer örneği tarafından döndürülen bir komutu zamanlar. Hangi otonom komutun çalıştırılacağını seçme mantığı, `` RobotContainer " içinde kullanılabilir.

Java

```

71 @Override
72 public void teleopInit() {
73     // This makes sure that the autonomous stops running when
74     // teleop starts running. If you want the autonomous to
75     // continue until interrupted by another command, remove
76     // this line or comment it out.
77     if (m_autonomousCommand != null) {
78         m_autonomousCommand.cancel();
79     }
80 }

```

C++ (Source)

```

46 void Robot::TeleopInit() {
47     // This makes sure that the autonomous stops running when
48     // teleop starts running. If you want the autonomous to
49     // continue until interrupted by another command, remove
50     // this line or comment it out.
51     if (m_autonomousCommand) {
52         m_autonomousCommand->Cancel();
53     }
54 }

```

teleopInit()` yöntemi her çalışan otonom komutu reddeder.Bu genellikle iyi bir uygulamadır.

İleri düzey kullanıcılar, uygun gördükleri şekilde çeşitli başlangıç ve periyodik yöntemlere ek kod eklemekte özgürdürler; fakat, `` Robot.java " ya büyük miktarlarda zorunlu robot kodu

dahil etmenin, komut tabanlı paradigmanın açıklayıcı tasarım felsefesine aykırı olduğunu ve kafa karıştırıcı bir şekilde yapılandırılmış / düzensiz kodla sonuçlanabileceği unutulmamalıdır.

26.6.2 RobotContainer

Bu sınıf (Java <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/examples/RobotContainer.java>>, C++ (Header) <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/templates/RobotContainer.h>>, C++ (Kaynak) <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/templates/RobotContainer.cpp>>), komut tabanlı robotunuz için kurulumun çoğunun gerçekleşeceği yerdir. Bu sınıfta, robotunuzun alt sistemlerini ve komutlarını tanımlayacak, bu komutları tetikleyici olaylara (butonlar gibi) bağlayacak ve otonom rutininizde hangi komutu çalıştıracağınızı belirleyeceksiniz. Bu sınıfın yeni kullanıcıların aşağıdakiler için açıklama isteyebilecekleri birkaç yönü vardır:

Java

```
23 private final ExampleSubsystem m_exampleSubsystem = new ExampleSubsystem();
```

C++ (Header)

```
32 ExampleSubsystem m_subsystem;
```

Dikkat et Alt-sistemler gizli olarak RobotContainer içine atanmıştır. Bu, komuta dayalı yapının önceki enkarnasyonu ile büyük bir zıtlık içindedir, ancak üzerinde mutabık kalınan nesne odaklı uygulamalarla çok daha uyumludur. Eğer alt sistemler ulusal değişkenler olarak atanırsa, kullanıcılara kodun istedikleri yerinden bunlara ulaşmalarına izin verir. Bu, bazı şeyleri kolaylaştırabilirken (örneğin, bu komutlara erişimleri için alt sistemleri komutlara geçirmeye gerek kalmaz), programın kontrol akışını olmadığı için takip etmeyi çok daha zor hale getirir. Kodun hangi bölümlerinin, kodun diğer bölümleri tarafından değiştirilebileceği veya değiştirildiği hemen anlaşılır. Erişim kolaylığı, kullanıcıların kaynak tarafından yönetilen komutların dışında yanlışlıkla alt sistem yöntemlerine çıkan çağrılar yapmasını kolaylaştırdığından, bu aynı zamanda kaynak yönetimi sisteminin işini yapmasına da engel olur.

Java

```
61 return Autos.exampleAuto(m_exampleSubsystem);
```

C++ (Source)

```
34 return autos::ExampleAuto(&m_subsystem);
```

Alt-sistemler gizli üyeler olarak açıklandığından beri, bu komutların komutları çağırması için komutlara açıkça aktarılmaları gerekir ("dependency injection" adı verilen bir kalıp). Bu, burada bir `` ExampleSubsystem `` e bir işaretçi iletilen `` ExampleCommand `` ile yapılır.

Java

```

35  /**
36   * Use this method to define your trigger->command mappings. Triggers can be
    ↪ created via the
37   * {@link Trigger#Trigger(java.util.function.BooleanSupplier)} constructor with an
    ↪ arbitrary
38   * predicate, or via the named factories in {@link
39   * edu.wpi.first.wpilibj2.command.button.CommandGenericHID}'s subclasses for {@link
40   * CommandXboxController Xbox}/{@link edu.wpi.first.wpilibj2.command.button.
    ↪ CommandPS4Controller
41   * PS4} controllers or {@link edu.wpi.first.wpilibj2.command.button.CommandJoystick
    ↪ Flight
42   * joysticks}.
43   */
44  private void configureBindings() {
45      // Schedule `ExampleCommand` when `exampleCondition` changes to `true`
46      new Trigger(m_exampleSubsystem::exampleCondition)
47          .onTrue(new ExampleCommand(m_exampleSubsystem));
48
49      // Schedule `exampleMethodCommand` when the Xbox controller's B button is pressed,
50      // cancelling on release.
51      m_driverController.b().whileTrue(m_exampleSubsystem.exampleMethodCommand());
52  }

```

C++ (Source)

```

19  void RobotContainer::ConfigureBindings() {
20      // Configure your trigger bindings here
21
22      // Schedule `ExampleCommand` when `exampleCondition` changes to `true`
23      frc2::Trigger([this] {
24          return m_subsystem.ExampleCondition();
25      }).OnTrue(ExampleCommand(&m_subsystem).ToPtr());
26
27      // Schedule `ExampleMethodCommand` when the Xbox controller's B button is
28      // pressed, cancelling on release.
29      m_driverController.B().WhileTrue(m_subsystem.ExampleMethodCommand());
30  }

```

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

```

54  /**
55   * Use this to pass the autonomous command to the main {@link Robot} class.
56   *
57   * @return the command to run in autonomous
58   */
59  public Command getAutonomousCommand() {
60      // An example command will be run in autonomous
61      return Autos.exampleAuto(m_exampleSubsystem);
62  }
63  }

```

C++ (Source)

```

32  frc2::CommandPtr RobotContainer::GetAutonomousCommand() {
33      // An example command will be run in autonomous
34      return autos::ExampleAuto(&m_subsystem);
35  }

```

Son olarak, ``getAutonomousCommand()`` yöntemi, kullanıcıların seçtikleri otonom komutlarını ana Robot sınıfına göndermeleri için uygun bir yol sağlar (otonom başladığında onu programlamak için ona erişim gerekir).

26.6.3 Sabitler

Constants sınıfı (Java <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/Constants.java>>, C++ (Header) <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/templates/Constants.h>>) (C++ 'da bu bir sınıf değil, yalnızca birkaç ad alanının tanımlandığı bir başlık dosyasıdır) küresel olarak erişilebilir robot sabitler (hızlar, birim dönüştürme faktörleri, PID kazançları ve sensör / motor portları gibi) saklanabilir. Değişken adlarını daha kısa tutmak için, kullanıcıların bu sabitleri alt sistemlere veya robot modlarına karşılık gelen bireysel iç sınıflara ayırmaları önerilir.

Javada, tüm sabitler `public static final` olarak açıklanmalıdır ki ulusal olarak erişilebilsin ve değiştirilemesin. C++ 'da tüm sabitler `constexpr` olmalıdır.

Bir ``constants`` sınıfının pratikte nasıl görünmesi gerektiğine dair daha fazla açıklayıcı örnekler için, çeşitli komut tabanlı örnek projelerin örneklerine bakın:

- FrisbeeBot (Java <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/Examples/FrisbeeBot/Constants.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/examples/frisbeebot/Constants.h>>)
- GyroDriveCommands (Java <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/Examples/GyroDriveCommands/Constants.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/examples/gyrodrive/Constants.h>>)
- Hatchbot (Java <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/Examples/Hatchbot/Constants.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/examples/hatchbot/Constants.h>>)
- RapidReactCommandBot (Java, C++)

Javada, sabitlerin diğer sınıflardan gerekli iç- sınıfı içe aktararak kullanılması önerilir. Bir `import static` ifadesi bir sınıfın statik ad alanını içinde çalıştığınız sınıfa aktarır, yani her `static` sabiti eğer sınıfın içinde tanımlıysa direkt başvurulabilir. C++ 'da, aynı etki `using namespace`: kullanılarak elde edilebilir.

JAVA

```
import static edu.wpi.first.wpilibj.templates.commandbased.Constants.OIConstants.*;
```

C++

```
using namespace OIConstants;
```

26.6.4 Alt Sistemler

Kullanıcı tanımlı alt sistemler bu paket / dizinine girmelidir.

26.6.5 Komutlar

Kullanıcı tanımlı komutlar bu paket / dizinine girmelidir.

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.isTrue(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:

Not: 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:

Not: 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:

Not: 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);
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
// Try to turn at a rate proportional to the heading error until we're at the
↪ setpoint, then stop
return run(() -> arcadeDrive(0, -controller.calculate(gyro.getHeading(),
↪ targetDegrees)))
    .until(controller::atSetpoint)
    .andThen(runOnce(() -> arcadeDrive(0, 0)));
}
```

C++

```
// TODO
```

This pattern works very well in Java so long as the captured state is “effectively final” - i.e., it is never reassigned. This means that we cannot directly define and capture primitive types (e.g. *int*, *double*, *boolean*) - to circumvent this, we need to wrap any state primitives in a mutable container type (the same way *PIDController* wraps its internal *kP*, *kI*, and *kD* values).

Writing Command Classes

Another possible way to define reusable commands is to write a class that represents the command. This is typically done by subclassing either *Command* or one of the *CommandGroup* classes.

Subclassing Command

Returning to our simple intake command from earlier, we could do this by creating a new subclass of *Command* that implements the necessary *initialize* and *end* methods.

JAVA

```
public class RunIntakeCommand extends Command {
    private Intake m_intake;

    public RunIntakeCommand(Intake intake) {
        this.m_intake = intake;
        addRequirements(intake);
    }

    @Override
    public void initialize() {
        m_intake.set(1.0);
    }

    @Override
    public void end(boolean interrupted) {
        m_intake.set(0.0);
    }

    // execute() defaults to do nothing
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
} // isFinished() defaults to return false
```

C++

```
// TODO
```

This, however, is just as cumbersome as the original repetitive code, if not more verbose. The only two lines that really matter in this entire file are the two calls to `intake.set()`, yet there are over 20 lines of boilerplate code! Not to mention, doing this for a lot of robot actions quickly clutters up a robot project with dozens of small files. Nevertheless, this might feel more “natural,” particularly for programmers who prefer to stick closely to an object-oriented model.

This approach should be used for commands with internal state (not subsystem state!), as the class can have fields to manage said state. It may also be more intuitive to write commands with complex logic as classes, especially for those less experienced with command composition. As the command is detached from any specific subsystem class and the required subsystem objects are injected through the constructor, this approach deals well with commands involving multiple subsystems.

Subclassing Command Groups

If we wish to write composite commands as their own classes, we may write a constructor-only subclass of the most exterior group type. For example, an intake-then-ouptake 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 Commands	Logic
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 Komut Zamanlayıcı - Command Scheduler

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` ayrıca kayıtlı her bir Alt Sistemin `periodic()` yöntemini çalıştırır.

26.8.1 Komut Zamanlayıcıyı Kullanma

CommandScheduler bir *singleton* 'dur, yani yalnızca bir örneğe sahip global olarak erişilebilir bir sınıftır. Buna göre, zamanlayıcıya erişmek için kullanıcıların CommandScheduler.getInstance() komutunu çağırması gerekir.

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).

Bununla birlikte, bir istisna vardır: kullanıcılar, Robot sınıfının robotPeriodic() yönteminin CommandScheduler.getInstance().run() çağrısı *yapmalıdır*. Bu yapılmazsa, zamanlayıcı asla çalışmaz ve komut çerçevesi çalışmaz. Sağlanan komut tabanlı proje şablonunda bu çağrı zaten dahil edilmiştir.

26.8.2 schedule() Metodu

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
215     ↪ doesn't
216     // run when disabled, or the command is already scheduled.
217     if (m_disabled
218         || isScheduled(command)
219         || RobotState.isDisabled() && !command.runsWhenDisabled()) {
220         return;

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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;
235             }
236         }
237         for (Subsystem requirement : requirements) {
238             Command requiring = requiring(requirement);
239             if (requiring != null) {
240                 cancel(requiring);
241             }
242         }
243         initCommand(command, requirements);
244     }
245 }

```

```

181 private void initCommand(Command command, Set<Subsystem> requirements) {
182     m_scheduledCommands.add(command);
183     for (Subsystem requirement : requirements) {
184         m_requirements.put(requirement, command);
185     }
186     command.initialize();
187     for (Consumer<Command> action : m_initActions) {
188         action.accept(command);
189     }
190
191     m_watchdog.addEpoch(command.getName() + ".initialize()");

```

C++ (Source)

```

114 void CommandScheduler::Schedule(Command* command) {
115     if (m_impl->inRunLoop) {
116         m_impl->toSchedule.emplace_back(command);
117         return;
118     }
119
120     RequireUngrouped(command);
121
122     if (m_impl->disabled || m_impl->scheduledCommands.contains(command) ||
123         (frc::RobotState::IsDisabled() && !command->RunsWhenDisabled())) {
124         return;

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

125 }
126
127 const auto& requirements = command->GetRequirements();
128
129 wpi::SmallVector<Command*, 8> intersection;
130
131 bool isDisjoint = true;
132 bool allInterruptible = true;
133 for (auto&& il : m_impl->requirements) {
134     if (requirements.find(il.first) != requirements.end()) {
135         isDisjoint = false;
136         allInterruptible &= (il.second->GetInterruptionBehavior() ==
137                             Command::InterruptionBehavior::kCancelSelf);
138         intersection.emplace_back(il.second);
139     }
140 }
141
142 if (isDisjoint || allInterruptible) {
143     if (allInterruptible) {
144         for (auto&& cmdToCancel : intersection) {
145             Cancel(cmdToCancel);
146         }
147     }
148     m_impl->scheduledCommands.insert(command);
149     for (auto&& requirement : requirements) {
150         m_impl->requirements[requirement] = command;
151     }
152     command->Initialize();
153     for (auto&& action : m_impl->initActions) {
154         action(*command);
155     }
156     m_watchdog.AddEpoch(command->GetName() + ".Initialize()");
157 }
158 }

```

26.8.3 Schedukler-Zamanlayıcı Çalıştırma Sırası

Not: Her bir Command initialize() yöntemi, komut zamanlandığında çağrılır, bu, zamanlayıcı çalıştığında zorunlu değildir (bu komut bir düğmeye bağlı değilse).

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++).

Adım 1: Alt Sistem Periyodik Yöntemlerini Çalıştırın

First, the scheduler runs the `periodic()` method of each registered Subsystem. In simulation, each subsystem's `simulationPeriodic()` method is called as well.

Java

```

278 // Run the periodic method of all registered subsystems.
279 for (Subsystem subsystem : m_subsystems.keySet()) {
280     subsystem.periodic();
281     if (RobotBase.isSimulation()) {
282         subsystem.simulationPeriodic();
283     }
284     m_watchdog.addEpoch(subsystem.getClass().getSimpleName() + ".periodic()");
285 }

```

C++ (Source)

```

183 // Run the periodic method of all registered subsystems.
184 for (auto&& subsystem : m_impl->subsystems) {
185     subsystem.getFirst()->Periodic();
186     if constexpr (frc::RobotBase::IsSimulation()) {
187         subsystem.getFirst()->SimulationPeriodic();
188     }
189     m_watchdog.AddEpoch("Subsystem Periodic()");
190 }

```

Adım 2: Poll Command Scheduling Tetikleyicileri

Not: Tetikleyici bağlamalarının nasıl çalıştığı hakkında daha fazla bilgi için bkz. [Komutları Trigger-Tetikleyicilere Bağlama](#)

İkinci olarak, programlayıcı, bu tetikleyicilere bağlanan yeni komutların programlanıp programlanmayacağını görmek için tüm kayıtlı tetikleyicilerin durumunu sorgular. Bağlı bir komutu zamanlama koşulları karşılanırsa, komut zamanlanır ve `Initialize()` yöntemi çalıştırılır.

Java

```

290 // Poll buttons for new commands to add.
291 loopCache.poll();
292 m_watchdog.addEpoch("buttons.run()");

```

C++ (Source)

```

195 // Poll buttons for new commands to add.
196 loopCache->Poll();
197 m_watchdog.AddEpoch("buttons.Run()");

```

3. Adım: Zamanlanmış-Scheduled Komutları Run/Finish

Üçüncüsü, zamanlayıcı, şu anda zamanlanmış olan her komutun `execute()` yöntemini çağırır ve ardından `isFinished()` yöntemini çağırarak komutun bitip bitmediğini kontrol eder. Komut bittiyse, `end()` yöntemi de çağırılır ve komutun programı kaldırılır ve gerekli alt sistemleri serbest bırakılır.

Bu çağrı dizisinin her komut için yapıldığına dikkat edin - bu nedenle, bir komutun `end()` yöntemi, diğerinin `execute()` yöntemi çağırılmadan önce çağırılmasına sahip olabilir. Komutlar planlandıkları sırayla işlenir.

Java

```

295 // Run scheduled commands, remove finished commands.
296 for (Iterator<Command> iterator = m_scheduledCommands.iterator(); iterator.
↪hasNext(); ) {
297     Command command = iterator.next();
298
299     if (!command.runWhenDisabled() && RobotState.isDisabled()) {
300         command.end(true);
301         for (Consumer<Command> action : m_interruptActions) {
302             action.accept(command);
303         }
304         m_requirements.keySet().removeAll(command.getRequirements());
305         iterator.remove();
306         m_watchdog.addEpoch(command.getName() + ".end(true)");
307         continue;
308     }
309
310     command.execute();
311     for (Consumer<Command> action : m_executeActions) {
312         action.accept(command);
313     }
314     m_watchdog.addEpoch(command.getName() + ".execute()");
315     if (command.isFinished()) {
316         command.end(false);
317         for (Consumer<Command> action : m_finishActions) {
318             action.accept(command);
319         }
320         iterator.remove();
321
322         m_requirements.keySet().removeAll(command.getRequirements());
323         m_watchdog.addEpoch(command.getName() + ".end(false)");
324     }
325 }

```

C++ (Source)

```

201 for (Command* command : m_impl->scheduledCommands) {
202     if (!command->RunsWhenDisabled() && frc::RobotState::IsDisabled()) {
203         Cancel(command);
204         continue;
205     }
206
207     command->Execute();
208     for (auto&& action : m_impl->executeActions) {
209         action(*command);
210     }
211     m_watchdog.AddEpoch(command->GetName() + ".Execute()");
212
213     if (command->IsFinished()) {
214         command->End(false);
215         for (auto&& action : m_impl->finishActions) {
216             action(*command);
217         }
218
219         for (auto&& requirement : command->GetRequirements()) {
220             m_impl->requirements.erase(requirement);
221         }
222
223         m_impl->scheduledCommands.erase(command);
224         m_watchdog.AddEpoch(command->GetName() + ".End(false)");
225     }
226 }

```

4. Adım: Varsayılan Komutları Programlayın - Schedule

Son olarak, kayıtlı herhangi bir Alt Sistemin-Subsystem varsayılan komutu planlanmıştır (eğer varsa). Varsayılan komutun initialize() yönteminin şu anda çağrılacağını unutmayın.

Java

```

340 // Add default commands for un-required registered subsystems.
341 for (Map.Entry<Subsystem, Command> subsystemCommand : m_subsystems.entrySet()) {
342     if (!m_requirements.containsKey(subsystemCommand.getKey())
343         && subsystemCommand.getValue() != null) {
344         schedule(subsystemCommand.getValue());
345     }
346 }

```

C++ (Source)

```

240 // Add default commands for un-required registered subsystems.
241 for (auto&& subsystem : m_impl->subsystems) {
242     auto s = m_impl->requirements.find(subsystem.getFirst());
243     if (s == m_impl->requirements.end() && subsystem.getSecond()) {
244         Schedule({subsystem.getSecond().get()});
245     }
246 }

```

26.8.4 Scheduler - Devre Dışı Bırakma

Programlayıcı, `CommandScheduler.getInstance().Disable()` çağrısı yapılarak devre dışı bırakılabilir. Devre dışı bırakıldığında, planlayıcının `schedule()` ve `run()` komutları hiçbir şey yapmaz.

Programlayıcı, `CommandScheduler.getInstance().enable()` çağrılarak yeniden etkinleştirilebilir.

26.8.5 Komut Olay Yöntemleri

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,
74 ↪ finish
75 CommandScheduler.getInstance()
76     .onCommandInitialize(
77         command ->
78             Shuffleboard.addEventMarker(
79                 "Command initialized", command.getName(), EventImportance.
80 ↪ kNormal));
79 CommandScheduler.getInstance()
80     .onCommandInterrupt(

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

81         command ->
82             Shuffleboard.addEventMarker(
83                 "Command interrupted", command.getName(), EventImportance.
84                 kNormal));
85         CommandScheduler.GetInstance()
86             .onCommandFinish(
87                 command ->
88                     Shuffleboard.addEventMarker(
89                         "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) {
32         frc::Shuffleboard::AddEventMarker(
33             "Command executed", command.GetName(),
34             frc::ShuffleboardEventImportance::kNormal);
35     });
36 frc2::CommandScheduler::GetInstance().OnCommandFinish(
37     [](const frc2::Command& command) {
38         frc::Shuffleboard::AddEventMarker(
39             "Command finished", command.GetName(),
40             frc::ShuffleboardEventImportance::kNormal);
41     });
42 frc2::CommandScheduler::GetInstance().OnCommandInterrupt(
43     [](const frc2::Command& command) {
44         frc::Shuffleboard::AddEventMarker(
45             "Command interrupted", command.GetName(),
46             frc::ShuffleboardEventImportance::kNormal);
47     });

```

26.9 C++ Komutları Üzerine Teknik Bir Tartışma

Not: 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.

Not: The model was further changed in 2023, as described *below*.

26.9.1 Sahiplik Modeli

Eski komut tabanlı framework, ham işaretçilerin kullanımını kullanıyordu, bu da kullanıcıların robot kodlarında `new` -yeni (manuel öbek ayrımlarıyla sonuçlanan) kullanmaları gerektiği anlamına geliyordu. Komutların kime ait olduğuna dair net bir gösterge olmadığından (zamanlayıcı, komut grupları veya kullanıcının kendisi), hafızayı boşaltmakla kimin ilgileneceği belli değildi.

Eski komut tabanlı framework'teki birkaç örnek şu şekilde kod içeriyordu:

```
#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).

Bu göze batan sorun, framework-çerçevenin yeniden yazılmasının nedenlerinden biriydi. Bu yeniden yazma ile birlikte kapsamlı bir sahiplik modeli ve kapsam dışına çıktıklarında hafızayı otomatik olarak boşaltacak akıllı işaretçilerin kullanımıyla birlikte tanıtıldı.

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`

`std::unique_ptr` kullanmak, nesneye kimin sahip olduğunu açıkça belirlememizi sağlar. Bir `std::unique_ptr` kopyalanamadığı için, öbek üzerinde aynı bellek bloğunu işaret eden bir `std::unique_ptr` nin birden fazla örneği olmayacaktır. Örneğin, `SequentialCommandGroup` için bir constructor -yapıcı, bir `std::vector<std::unique_ptr<Command>>&` alır. Bu, `std::unique_ptr<Command>` vektörüne bir rvalue referansı gerektirdiği anlamına gelir. Bunu daha iyi anlamak için bazı örnek kodları adım adım inceleyelim:

```
// 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.
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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.

```

std::shared_ptr ile, açık bir sahiplik modeli yoktur çünkü aynı bellek bloğunu işaret eden bir std::shared_ptr nin birden fazla örneği olabilir. Komutlar std::shared_ptr örneklerinde olsaydı, bir komut grubu veya komut zamanlayıcı, komutun yürütülmesi bittikten sonra sahipliği alamaz ve belleği boşaltamaz çünkü kullanıcı hala kapsamı içinde bir yerdeki bellek bloğuna işaret eden örnek std::shared_ptr ye sahip olabilir.

26.9.2 CRTP kullanımı

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:

Komut Dekoratorleri

Yeni komut tabanlı framework, kullanıcının aşağıdaki gibi bir şey yapmasına olanak tanıyan "komut dekoratörleri" olarak bilinen bir özelliği içerir:

```

auto task = MyCommand().AndThen([] { std::cout << "This printed after my command
↳ ended."; },
requirements);

```

task -görev zamanlandığında, ilk önce MyCommand() çalıştıracak ve bu komutun yürütülmesi bittiğinde, mesajı konsola yazdıracaktır. Bunun dahili olarak elde etmenin yolu, sıralı bir komut grubu kullanmaktır.

Bir önceki bölümden, sıralı bir komut grubu oluşturmak için her komuta benzersiz bir işaretçi vektörüne ihtiyacımız olduğunu hatırlayın. Yazdırma işlevi için benzersiz bir işaretçi oluşturmak oldukça basittir:

```
temp.emplace_back(
    std::make_unique<InstantCommand>(std::move(toRun), requirements));
```

Burada temp , SequentialCommandGroup yapıcısına iletmemiz gereken komut vektörünü depoluyor. Ancak InstantCommand`i eklemekten önce, ``SequentialCommandGroup a, MyCommand() eklememiz gerekiyor. Bunu nasıl yaparız?

```
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.

Soruna Yönelik Çözümler

Buna ilk çözümümüz, Command içinde TransferOwnership() adlı, Command in her alt sınıfının geçersiz kılmak zorunda olduğu sanal bir yöntem oluşturmaktır. Böyle bir geçersiz kılma şuna benzerdi:

```
std::unique_ptr<Command> TransferOwnership() && override {
    return std::make_unique<MyCommand>(std::move(*this));
}
```

Kod türetilmiş alt sınıfta olacağından, *this aslında istenen alt sınıf örneğini işaret eder ve kullanıcı, benzersiz işaretçiyi yapmak için türetilmiş sınıfın tür bilgisine sahiptir.

Birkaç günlük değerlendirmeden sonra, bir CRTP yöntemi önerildi. Burada, CommandHelper adı verilen, aracı olarak türetilmiş bir Command sınıfı mevcut olacaktır. CommandHelper iki şablon argümanına sahip olacaktır, orijinal temel sınıf ve istenen türetilmiş alt sınıf. Bunu anlamak için CommandHelper' in temel uygulamasına bir göz atalım:

```
// In the real implementation, we use SFINAE to check that Base is actually a
// Command or a subclass of Command.
template<typename Base, typename Derived>
class CommandHelper : public Base {
    // Here, we are just inheriting all of the superclass (base class) constructors.
    using Base::Base;

    // Here, we will override the TransferOwnership() method mentioned above.
    std::unique_ptr<Command> TransferOwnership() && override {
        // Previously, we mentioned that we had no information about the derived class
        // to cast to at compile-time, but because of CRTP we do! It's one of our template
        // arguments!
        return std::make_unique<Derived>(std::move(*static_cast<Derived*>(this)));
    }
};
```

Bu nedenle, özel komutlarınızı Command yerine CommandHelper kapsamında yapmak, bu standart levhayı -boilerplate sizin için otomatik olarak uygulayacaktır ve ekiplerden işleri yapmak için oldukça belirsiz gibi görünen bir yöntemi kullanmalarını istemenin arkasındaki sebep budur.

AndThen() örneğimize geri dönersek, şimdi aşağıdakileri yapabiliriz:

```
// 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 Gelişmiş Dekoratorlerin Eksikliği

C++ dekoratorlerinin çoğu, gerçek komutların kendileri yerine `std::function<void()>` alır. Dekoratorlerde `AndThen()`, `BeforeStarting()` vb. gibi gerçek komutları alma fikri düşünüldü, ancak daha sonra çeşitli nedenlerle terk edildi.

Dekoratorleri Şablonlama

Derleme zamanında, bir komut grubuna eklediğimiz komutların türlerini bilmemiz gerektiğinden, şablonları (çoklu komutlar için değişken) kullanmamız gerekecek. Ancak, bu büyük bir uğraş gibi görünmeyebilir. Komut grupları için yapıcılar bunu yine de yapar:

```
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));
}
```

Not: Bu, yukarıda tanımladığımız vektör yapıcısına ek olarak `SequentialCommandGroup` için ikincil bir kurucudur.

Bununla birlikte, şablonlanmış bir fonksiyon yaptığımızda, tanımlı satır içi-inline olarak bildirilmelidir. Bu, bir sorun oluşturan `Command.h` başlığındaki `SequentialCommandGroup` örneğini oluşturmamız gerektiği anlamına gelir. `SequentialCommandGroup.h`, `Command.h` içerir. `Command.h` içerisine `SequentialCommandGroup.h` eklersek, döngüsel bir bağımlılığımız olur. Şimdi nasıl yapacağız peki?

`Command.h`' başlığının üstünde ileriye dönük bir bildirim kullanıyoruz:

```
class SequentialCommandGroup;

class Command { ... };
```

Ve sonra `SequentialCommandGroup.h` yi `Command.cpp` içine dahil ediyoruz. Bu dekoratör işlevleri şablon haline getirildiyse, `.cpp` dosyalarına tanım yazamayız ve bu da döngüsel bir bağımlılıkla sonuçlanır.

Java vs. C++ Sözdizimi Kuralları

Bu dekoratörler genellikle Java'da C++'dan daha fazla ayrıntıyı kaydeder (çünkü Java, ham `new` -yeni çağrılar gerektirir), bu nedenle genel olarak; komut grubunu, kullanıcı kodunda manuel olarak oluşturursanız, C++'da söz dizimsel bir fark yaratmaz.

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:

```

33 frc2::CommandPtr autos::ComplexAuto(DriveSubsystem* drive,
34                                     HatchSubsystem* hatch) {
35     return frc2::cmd::Sequence(
36         // Drive forward the specified distance
37         frc2::FunctionalCommand(
38             // Reset encoders on command start
39             [drive] { drive->ResetEncoders(); },
40             // Drive forward while the command is executing
41             [drive] { drive->ArcadeDrive(kAutoDriveSpeed, 0); },
42             // Stop driving at the end of the command
43             [drive](bool interrupted) { drive->ArcadeDrive(0, 0); },
44             // End the command when the robot's driven distance exceeds the
45             // desired value
46             [drive] {
47                 return drive->GetAverageEncoderDistance() >=
48                     kAutoDriveDistanceInches;
49             },
50             // Requires the drive subsystem
51             {drive})
52         .ToPtr(),
53         // Release the hatch
54         hatch->ReleaseHatchCommand(),
55         // Drive backward the specified distance
56         // Drive forward the specified distance
57         frc2::FunctionalCommand(
58             // Reset encoders on command start
59             [drive] { drive->ResetEncoders(); },
60             // Drive backward while the command is executing
61             [drive] { drive->ArcadeDrive(-kAutoDriveSpeed, 0); },
62             // Stop driving at the end of the command
63             [drive](bool interrupted) { drive->ArcadeDrive(0, 0); },
64             // End the command when the robot's driven distance exceeds the
65             // desired value
66             [drive] {
67                 return drive->GetAverageEncoderDistance() <=
68                     kAutoBackupDistanceInches;
69             },
70             // Requires the drive subsystem
71             {drive})
72         .ToPtr());
73 }
```

To avoid breakage, command compositions still use `unique_ptr<Command>`, so CommandPtr instances can be destructured into a `unique_ptr<Command>` using the `Unwrap()` rvalue-this method. For vectors, the static `CommandPtr::UnwrapVector(vector<CommandPtr>)` function

exists.

26.10 PIDSubsystems ve PIDCommands üzerinden PID Kontrol

Not: Bu komut tabanlı sarmalayıcılar tarafından kullanılan WPILib PID kontrol özelliklerinin bir açıklaması için, bakınız [WPILib’de PID Kontrolü](#)

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 PIDSubsystems

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.

PIDSubsystem oluşturma

Not: If `periodic` is overridden when inheriting from [PIDSubsystem](#), make sure to call `super.periodic()`! Otherwise, PID functionality will not work properly.

[PIDSubsystem](#) alt sınıfını oluştururken, kullanıcılar sınıfın normal işleminde kullanacağı işlevselliği sağlamak için iki soyut yöntemi geçersiz kılmalıdır:

`getMeasurement()`

Java

```
protected abstract double getMeasurement();
```

C++

```
virtual double GetMeasurement() = 0;
```

getMeasurement yöntemi, süreç değişkeninin mevcut ölçümünü döndürür. PIDSubsystem bu yöntemi periodic() bloğundan otomatik olarak çağırarak ve değerini kontrol döngüsüne aktaracaktır.

Kullanıcılar, işlem değişken ölçümü olarak kullanmak istedikleri sensör okumasını döndürmek için bu yöntemi geçersiz kılmalıdır.

useOutput()**Java**

```
protected abstract void useOutput(double output, double setpoint);
```

C++

```
virtual void UseOutput(double output, double setpoint) = 0;
```

useOutput() yöntemi, PID denetleyicisinin çıkışı ve mevcut ayar noktası durumunu (bu genellikle bir ileri beslemeyi hesaplamak için yararlıdır) kullanır. PIDSubsystem, bu yöntemi periodic() bloğundan otomatik olarak çağırarak ve ona kontrol döngüsünün hesaplanmış çıktısını iletecektir.

Kullanıcılar, son hesaplanmış kontrol çıkışı alt sistemlerinin motorlarına geçirmek için bu yöntemi geçersiz kılmalıdır.

Controller'a Geçirme

Kullanıcılar ayrıca, alt sınıflarının üst sınıf yapıcı çağırısı aracılığıyla PIDSubsystem temel sınıfına bir PIDController geçirmelidir. Bu, PID kazançlarının yanı sıra periyodu (kullanıcı standart olmayan bir main robot looğ döngü periyodu kullanıyorsa) belirlemeye yarar.

getController() çağırısı yapılarak, yapıcı gövdesindeki denetleyicide ek değişiklikler (e.g. enabling continuous input- ör. sürekli girişi etkinleştirmek) yapılabilir.

PIDSubsystem kullanımı

Bir PIDSubsystem alt sınıfının bir örneği oluşturulduktan sonra, aşağıdaki yöntemler aracılığıyla komutlar tarafından kullanılabilir:

setSetpoint()

`setSetpoint()` yöntemi, `PIDSubsystem` in ayar noktasını ayarlamak için kullanılabilir. Alt sistem, tanımlanan çıktıyı kullanarak ayar noktasını otomatik olarak izleyecektir:

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() ve disable()

`enable()` ve `disable()` yöntemleri `PIDSubsystem` in PID kontrolünü etkinleştirir ve devre dışı bırakır. Alt sistem etkinleştirildiğinde, kontrol döngüsünü otomatik olarak çalıştıracak ve ayar noktasını izleyecektir. Devre dışı bırakıldığında hiçbir kontrol gerçekleştirilmez.

Ek olarak, `enable()` yöntemi dahili `PIDController` i sıfırlar ve `disable()` yöntemi hem çıktı hem de ayar noktası `` olarak ayarlanmış şekilde kullanıcı tanımlı `useOutput()` yöntemini ``0``a ayarlar.

Tam PIDSubsystem Örneği

Pratikte kullanıldığında bir ``PIDS alt sistemi`` neye benzer? Aşağıdaki örnekler, FrisbeeBot örnek projesinden alınmıştır (Java, C++):

Java

```
5 package edu.wpi.first.wpilibj.examples.frisbeebot.subsystems;
6
7 import edu.wpi.first.math.controller.PIDController;
8 import edu.wpi.first.math.controller.SimpleMotorFeedforward;
9 import edu.wpi.first.wpilibj.Encoder;
10 import edu.wpi.first.wpilibj.examples.frisbeebot.Constants.ShooterConstants;
11 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
12 import edu.wpi.first.wpilibj2.command.PIDSubsystem;
13
14 public class ShooterSubsystem extends PIDSubsystem {
15     private final PWMSparkMax m_shooterMotor = new PWMSparkMax(ShooterConstants.
16         ↪ kShooterMotorPort);
17     private final PWMSparkMax m_feederMotor = new PWMSparkMax(ShooterConstants.
18         ↪ kFeederMotorPort);
19     private final Encoder m_shooterEncoder =
20         new Encoder(
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

19     ShooterConstants.kEncoderPorts[0],
20     ShooterConstants.kEncoderPorts[1],
21     ShooterConstants.kEncoderReversed);
22     private final SimpleMotorFeedforward m_shooterFeedforward =
23         new SimpleMotorFeedforward(
24             ShooterConstants.kSVolts, ShooterConstants.kVVoltsSecondsPerRotation);
25
26     /** The shooter subsystem for the robot. */
27     public ShooterSubsystem() {
28         super(new PIDController(ShooterConstants.kP, ShooterConstants.kI,
29             ShooterConstants.kD));
30         getController().setTolerance(ShooterConstants.kShooterToleranceRPS);
31         m_shooterEncoder.setDistancePerPulse(ShooterConstants.kEncoderDistancePerPulse);
32         setSetpoint(ShooterConstants.kShooterTargetRPS);
33     }
34
35     @Override
36     public void useOutput(double output, double setpoint) {
37         m_shooterMotor.setVoltage(output + m_shooterFeedforward.calculate(setpoint));
38     }
39
40     @Override
41     public double getMeasurement() {
42         return m_shooterEncoder.getRate();
43     }
44
45     public boolean atSetpoint() {
46         return m_controller.atSetpoint();
47     }
48
49     public void runFeeder() {
50         m_feederMotor.set(ShooterConstants.kFeederSpeed);
51     }
52
53     public void stopFeeder() {
54         m_feederMotor.set(0);
55     }

```

C++

```

5     #pragma once
6
7     #include <frc/Encoder.h>
8     #include <frc/controller/SimpleMotorFeedforward.h>
9     #include <frc/motorcontrol/PWMSparkMax.h>
10    #include <frc2/command/PIDSubsystem.h>
11    #include <units/angle.h>
12
13    class ShooterSubsystem : public frc2::PIDSubsystem {
14    public:
15        ShooterSubsystem();
16
17        void UseOutput(double output, double setpoint) override;

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

18     double GetMeasurement() override;
19
20     bool AtSetpoint();
21
22     void RunFeeder();
23
24     void StopFeeder();
25
26 private:
27     frc::PWMSparkMax m_shooterMotor;
28     frc::PWMSparkMax m_feederMotor;
29     frc::Encoder m_shooterEncoder;
30     frc::SimpleMotorFeedforward<units::turns> m_shooterFeedforward;
31 };
32

```

C++ (Source)

```

5  #include "subsystems/ShooterSubsystem.h"
6
7  #include <frc/controller/PIDController.h>
8
9  #include "Constants.h"
10
11 using namespace ShooterConstants;
12
13 ShooterSubsystem::ShooterSubsystem()
14     : PIDSubsystem{frc::PIDController{kP, kI, kD}},
15       m_shooterMotor(kShooterMotorPort),
16       m_feederMotor(kFeederMotorPort),
17       m_shooterEncoder(kEncoderPorts[0], kEncoderPorts[1]),
18       m_shooterFeedforward(kS, kV) {
19     m_controller.SetTolerance(kShooterToleranceRPS.value());
20     m_shooterEncoder.SetDistancePerPulse(kEncoderDistancePerPulse);
21     SetSetpoint(kShooterTargetRPS.value());
22 }
23
24 void ShooterSubsystem::UseOutput(double output, double setpoint) {
25     m_shooterMotor.SetVoltage(units::volt_t{output} +
26                               m_shooterFeedforward.Calculate(kShooterTargetRPS));
27 }
28
29 bool ShooterSubsystem::AtSetpoint() {
30     return m_controller.AtSetpoint();
31 }
32
33 double ShooterSubsystem::GetMeasurement() {
34     return m_shooterEncoder.GetRate();
35 }
36
37 void ShooterSubsystem::RunFeeder() {
38     m_feederMotor.Set(kFeederSpeed);
39 }
40

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

41 void ShooterSubsystem::StopFeeder() {
42     m_feederMotor.Set(0);
43 }

```

Komutlarla bir PIDSubsystem kullanmak çok basit olabilir:

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++

```

45 frc2::CommandPtr m_spinUpShooter =
46     frc2::cmd::RunOnce([this] { m_shooter.Enable(); }, {&m_shooter});
47
48 frc2::CommandPtr m_stopShooter =
49     frc2::cmd::RunOnce([this] { m_shooter.Disable(); }, {&m_shooter});

```

C++ (Source)

```

25 // We can bind commands while keeping their ownership in RobotContainer
26
27 // Spin up the shooter when the 'A' button is pressed
28 m_driverController.A().OnTrue(m_spinUpShooter.get());
29
30 // Turn off the shooter when the 'B' button is pressed
31 m_driverController.B().OnTrue(m_stopShooter.get());

```

26.10.2 PIDCommand

The PIDCommand class allows users to easily create commands with a built-in PIDController.

Bir PIDCommand oluşturma

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.

Not: If subclassing PIDCommand and overriding any methods, make sure to call the super version of those methods! Otherwise, PID functionality will not work properly.

Her iki durumda da, gerekli parametreleri kurucusuna ileterek bir PIDCommand oluşturulur (eğer bir alt sınıf tanımlıyorsa, bu bir *super()* çağrısıyla yapılabilir):

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 = {});
```

Kontrolör - Controller

Controller parametresi, komut tarafından kullanılacak olan PIDController nesnesidir. Bunu bilerek, kullanıcılar PID kazanımlarını, ve kontrolör için süreyi belirleyebilir (eğer kullanıcı standart olmayan bir ana robot döngü periyodu kullanıyorsa).

PIDCommand alt sınıfını oluştururken, `getController()` çağrısı yapılarak yapıcı gövdesindeki denetleyicide ek değişiklikler (ör. Sürekli girişi etkinleştirmek) yapılabilir.

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`.

`PIDCommand` alt sınıfını oluştururken, ileri düzey kullanıcılar, sınıfın `m_measurement` alanını değiştirerek ayar noktası tedarikçisini daha da değiştirebilir.

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.

`PIDCommand` alt sınıfını oluştururken, ileri düzey kullanıcılar, sınıfın `m_setpoint` alanını değiştirerek ayar noktası tedarikçisini daha da değiştirebilir.

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`.

`PIDCommand` alt sınıfını oluştururken, ileri düzey kullanıcılar, sınıfın `m_useOutput` alanını değiştirerek çıktı tüketicisini daha da değiştirebilir.

Gereksinimler

Tüm satır içi komutlar gibi, `PIDCommand` kullanıcının alt sistem gereksinimlerini bir constructor parametresi olarak belirtmesine izin verir.

Tam PIDCommand Örneği

Pratikte kullanıldığında bir ``PIDCommand`` neye benzer? Aşağıdaki örnekler GyroDriveCommands örnek projesinden alınmıştır (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/GyroDriveCommands>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/GyroDriveCommands>>):

Java

```

5 package edu.wpi.first.wpilibj.examples.gyrodrivecommands.commands;
6
7 import edu.wpi.first.math.controller.PIDController;
8 import edu.wpi.first.wpilibj.examples.gyrodrivecommands.Constants.DriveConstants;
9 import edu.wpi.first.wpilibj.examples.gyrodrivecommands.subsystems.DriveSubsystem;
10 import edu.wpi.first.wpilibj2.command.PIDCommand;
11
12 /** A command that will turn the robot to the specified angle. */
13 public class TurnToAngle extends PIDCommand {
14     /**
15      * Turns to robot to the specified angle.
16      *
17      * @param targetAngleDegrees The angle to turn to
18      * @param drive The drive subsystem to use
19      */
20     public TurnToAngle(double targetAngleDegrees, DriveSubsystem drive) {
21         super(
22             new PIDController(DriveConstants.kTurnP, DriveConstants.kTurnI,
23 ↪ DriveConstants.kTurnD),
24             // Close loop on heading
25             drive::getHeading,
26             // Set reference to target
27             targetAngleDegrees,
28             // Pipe output to turn robot
29             output -> drive.arcadeDrive(0, output),
30             // Require the drive
31             drive);
32
33             // Set the controller to be continuous (because it is an angle controller)
34             getController().enableContinuousInput(-180, 180);
35             // Set the controller tolerance - the delta tolerance ensures the robot is
36 ↪ stationary at the
37             // setpoint before it is considered as having reached the reference
38             getController()
39                 .setTolerance(DriveConstants.kTurnToleranceDeg, DriveConstants.
40 ↪ kTurnRateToleranceDegPerS);
41         }
42
43     @Override
44     public boolean isFinished() {
45         // End when the controller is at the reference.
46         return getController().atSetpoint();
47     }
48 }

```

C++

```

5  #pragma once
6
7  #include <frc2/command/CommandHelper.h>
8  #include <frc2/command/PIDCommand.h>
9
10 #include "subsystems/DriveSubsystem.h"
11
12 /**
13  * A command that will turn the robot to the specified angle.
14  */
15 class TurnToAngle : public frc2::CommandHelper<frc2::PIDCommand, TurnToAngle> {
16 public:
17     /**
18      * Turns to robot to the specified angle.
19      *
20      * @param targetAngleDegrees The angle to turn to
21      * @param drive               The drive subsystem to use
22      */
23     TurnToAngle(units::degree_t target, DriveSubsystem* drive);
24
25     bool IsFinished() override;
26 };

```

C++ (Source)

```

5  #include "commands/TurnToAngle.h"
6
7  #include <frc/controller/PIDController.h>
8
9  using namespace DriveConstants;
10
11 TurnToAngle::TurnToAngle(units::degree_t target, DriveSubsystem* drive)
12     : CommandHelper{frc::PIDController{kTurnP, kTurnI, kTurnD},
13                     // Close loop on heading
14                     [drive] { return drive->GetHeading().value(); },
15                     // Set reference to target
16                     target.value(),
17                     // Pipe output to turn robot
18                     [drive](double output) { drive->ArcadeDrive(0, output); },
19                     // Require the drive
20                     {drive}} {
21     // Set the controller to be continuous (because it is an angle controller)
22     m_controller.EnableContinuousInput(-180, 180);
23     // Set the controller tolerance - the delta tolerance ensures the robot is
24     // stationary at the setpoint before it is considered as having reached the
25     // reference
26     m_controller.SetTolerance(kTurnTolerance.value(), kTurnRateTolerance.value());
27
28     AddRequirements(drive);
29 }
30
31 bool TurnToAngle::IsFinished() {
32     return GetController().AtSetpoint();

```

(sonraki sayfaya devam)

33 }

And, for an *inlined* example:

Java

```

64 // Stabilize robot to drive straight with gyro when left bumper is held
65 new JoystickButton(m_driverController, Button.kL1.value)
66   .whileTrue(
67     new PIDCommand(
68       new PIDController(
69         DriveConstants.kStabilizationP,
70         DriveConstants.kStabilizationI,
71         DriveConstants.kStabilizationD),
72       // Close the loop on the turn rate
73       m_robotDrive::getTurnRate,
74       // Setpoint is 0
75       0,
76       // Pipe the output to the turning controls
77       output -> m_robotDrive.arcadeDrive(-m_driverController.getLeftY(),
↪output),
78       // Require the robot drive
79       m_robotDrive));

```

C++

```

34 // Stabilize robot to drive straight with gyro when L1 is held
35 frc2::JoystickButton(&m_driverController, frc::PS4Controller::Button::kL1)
36   .WhileTrue(
37     frc2::PIDCommand(
38       frc::PIDController{dc::kStabilizationP, dc::kStabilizationI,
39                          dc::kStabilizationD},
40       // Close the loop on the turn rate
41       [this] { return m_drive.GetTurnRate(); },
42       // Setpoint is 0
43       0,
44       // Pipe the output to the turning controls
45       [this](double output) {
46         m_drive.ArcadeDrive(m_driverController.GetLeftY(), output);
47       },
48       // Require the robot drive
49       {&m_drive});

```

26.11 TrapezProfileSubsystems ve TrapezoidProfileCommands ile Hareket Profillemesi

Not: Bu komut tabanlı sarmalayıcılar tarafından kullanılan WPILib hareket profili oluşturma özelliklerinin bir açıklaması için bakınız [WPILib’de Trapezoid Hareket Profilleri](#)

Not: TrapezoidProfile`` komut sarmalayıcıları genellikle özel veya harici denetleyicilerle kompozisyon için tasarlanmıştır. Trapezoidal hareket profillemeyi WPILib’in PIDController ile birleştirmek için, bakınız [Komut Tabanlı Hareket Profillemesi ve PID’yi Birleştirme](#).

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++).

WPILib, takımların komut tabanlı robot projelerine hareket profillemeyi entegre etmelerine daha fazla yardımcı olmak için, TrapezoidProfile sınıfı için iki kullanışlı sarmalayıcı içerir: TrapezoidProfileSubsystem, otomatik olarak periodic() içinde hareket profilleri oluşturulan ve TrapezoidProfileCommand, kullanıcı tarafından sağlanan tek bir TrapezoidProfile çalıştırır.

26.11.1 TrapezoidProfileSubsystem

Not: C++’da, TrapezoidProfileSubsystem sınıfı, açısal veya doğrusal olabilen mesafe ölçümleri için kullanılan birim türüne göre şablonlanır. İletilen değerler *zorunlu* mesafe birimleriyle tutarlı birimlere sahip olmalıdır, aksi takdirde bir derleme zamanı hatası atılır. C++ birimleri hakkında daha fazla bilgi için bakınız [C++ Ünite Kitaplığı](#).

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.

TrapezoidProfileSubsystem Oluşturma

Not: If periodic is overridden when inheriting from TrapezoidProfileSubsystem, make sure to call super.periodic()! Otherwise, motion profiling functionality will not work properly.

TrapezoidProfileSubsystem alt sınıfını oluştururken, kullanıcılar sınıfın olağan işleminde kullanacağı işlevselliği sağlamak için tek bir soyut methodu geçersiz kılmalıdır:

useState()

Java

```
protected abstract void useState(TrapezoidProfile.State state);
```

C++

```
virtual void UseState(State state) = 0;
```

useState() metodu, hareket profilinin mevcut durumunu tüketir. TrapezoidProfileSubsystem, bu yöntemi periodic() bloğundan otomatik olarak çağırarak ve alt sistemin hareket profili boyunca mevcut ilerlemesine karşılık gelen hareket profili durumunu iletecektir.

Kullanıcılar bu durumla istediklerini yapabilir; tipik bir kullanım durumu (*Full TrapezoidProfileSubsystem Example* 'da gösterildiği gibi), harici "akıllı" bir motor denetleyicisi için bir ayar noktası ve ileri besleme elde etmek için kullanmaktır.

Constructor Parametreleri

Kullanıcılar, alt sınıflarının superclass yapıcı çağrısı aracılığıyla TrapezoidProfile.Constraints temel sınıfına bir TrapezoidProfile.Constraints kümesi göndermelidir. Bu, otomatik olarak oluşturulan profilleri belirli bir maksimum hız ve ivmeyle sınırlamaya yarar.

Kullanıcılar ayrıca mekanizma için bir başlangıç pozisyonuna geçmelidir.

Standart olmayan bir ana loop periyodu kullanılıyorsa, gelişmiş kullanıcılar loop periyodu için alternatif bir değer geçebilir.

TrapezoidProfileSubsystem Kullanma

Bir TrapezoidProfileSubsystem alt sınıfının bir örneği oluşturulduktan sonra, aşağıdaki yöntemler aracılığıyla komutlar tarafından kullanılabilir:

setGoal()

Not: Hedefi, örtük hedef hızı sıfır olan basit bir mesafeye ayarlamak isterseniz, tam hareket profili durumu yerine tek bir mesafe değeri alan bir setGoal() aşırı yükü vardır.

setGoal() metodu, TrapezoidProfileSubsystem in hedef durumunu ayarlamak için kullanılabilir. Alt sistem, her yinelemede mevcut durumu sağlanan *useState()* metoduna geçirerek hedefe otomatik olarak bir profil yürütür.

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.
examplePIDSubsystem.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.

Tam TrapezoidProfileSubsystem Örneği

Pratikte kullanıldığında bir ``TrapezoidProfileSubsystem`` neye benziyor? Aşağıdaki örnekler, ArmbotOffboard örnek projesinden alınmıştır (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/ArmBotOffboard>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/ArmBotOffboard>>):

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.trjectory.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(
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

27     new TrapezoidProfile.Constraints(
28         ArmConstants.kMaxVelocityRadPerSecond, ArmConstants.
↪ kMaxAccelerationRadPerSecSquared),
29         ArmConstants.kArmOffsetRads);
30     m_motor.setPID(ArmConstants.kP, 0, 0);
31 }
32
33 @Override
34 public void useState(TrapezoidProfile.State setpoint) {
35     // Calculate the feedforward from the sepoint
36     double feedforward = m_feedforward.calculate(setpoint.position, setpoint.
↪ velocity);
37     // Add the feedforward to the PID output to get the motor output
38     m_motor.setSetpoint(
39         ExampleSmartMotorController.PIDMode.kPosition, setpoint.position, feedforward_
↪ / 12.0);
40 }
41
42 public Command setArmGoalCommand(double kArmOffsetRads) {
43     return Commands.runOnce(() -> setGoal(kArmOffsetRads), this);
44 }
45 }

```

C++ (Header)

```

5  #pragma once
6
7  #include <frc/controller/ArmFeedforward.h>
8  #include <frc2/command/Commands.h>
9  #include <frc2/command/TrapezoidProfileSubsystem.h>
10 #include <units/angle.h>
11
12 #include "ExampleSmartMotorController.h"
13
14 /**
15  * A robot arm subsystem that moves with a motion profile.
16  */
17 class ArmSubsystem : public frc2::TrapezoidProfileSubsystem<units::radians> {
18     using State = frc::TrapezoidProfile<units::radians>::State;
19
20 public:
21     ArmSubsystem();
22
23     void UseState(State setpoint) override;
24
25     frc2::CommandPtr SetArmGoalCommand(units::radian_t goal);
26
27 private:
28     ExampleSmartMotorController m_motor;
29     frc::ArmFeedforward m_feedforward;
30 };

```

C++ (Source)

```

5  #include "subsystems/ArmSubsystem.h"
6
7  #include "Constants.h"
8
9  using namespace ArmConstants;
10 using State = frc::TrapezoidProfile<units::radians>::State;
11
12 ArmSubsystem::ArmSubsystem()
13     : frc2::TrapezoidProfileSubsystem<units::radians>(
14         {kMaxVelocity, kMaxAcceleration}, kArmOffset),
15         m_motor(kMotorPort),
16         m_feedforward(kS, kG, kV, kA) {
17     m_motor.SetPID(kP, 0, 0);
18 }
19
20 void ArmSubsystem::UseState(State setpoint) {
21     // Calculate the feedforward from the sepoint
22     units::volt_t feedforward =
23         m_feedforward.Calculate(setpoint.position, setpoint.velocity);
24     // Add the feedforward to the PID output to get the motor output
25     m_motor.SetSetpoint(ExampleSmartMotorController::PIDMode::kPosition,
26         setpoint.position.value(), feedforward / 12_V);
27 }
28
29 frc2::CommandPtr ArmSubsystem::SetArmGoalCommand(units::radian_t goal) {
30     return frc2::cmd::RunOnce([this, goal] { this->SetGoal(goal); }, {this});
31 }

```

Komutlarla bir TrapezoidProfileSubsystem kullanmak oldukça basit olabilir:

Java

```

52 // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
53 m_driverController.a().onTrue(m_robotArm.setArmGoalCommand(2));
54
55 // Move the arm to neutral position when the 'B' button is pressed.
56 m_driverController
57     .b()
58     .onTrue(m_robotArm.setArmGoalCommand(Constants.ArmConstants.kArmOffsetRads));

```

C++

```

25 // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
26 m_driverController.A().OnTrue(m_arm.SetArmGoalCommand(2_rad));
27
28 // Move the arm to neutral position when the 'B' button is pressed.
29 m_driverController.B().OnTrue(
30     m_arm.SetArmGoalCommand(ArmConstants::kArmOffset));

```

26.11.2 TrapezoidProfileCommand

Not: C++ 'da, TrapezoidProfileCommand sınıfı, açısal veya doğrusal olabilen mesafe ölçümleri için kullanılan birim türüne göre şablonlanır. İletilen değerler *zorunlu* mesafe birimleriyle tutarlı birimlere sahip olmalıdır, aksi takdirde bir derleme zamanı hatası atılır. C++ birimleri hakkında daha fazla bilgi için bakınız *C++ Ünite Kitaplığı*.

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.

TrapezoidProfileCommand Oluşturma

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.

Not: 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.

Her iki durumda da, bir TrapezoidProfileCommand, constructor'a gerekli parametreler iletilerek oluşturulur (eğer bir alt sınıf tanımlıyorsa, bu bir super() çağrısıyla yapılabilir):

Java

```

28  * Creates a new TrapezoidProfileCommand that will execute the given {@link
↪TrapezoidProfile}.
29  * Output will be piped to the provided consumer function.
30  *
31  * @param profile The motion profile to execute.
32  * @param output The consumer for the profile output.
33  * @param goal The supplier for the desired state
34  * @param currentState The current state
35  * @param requirements The subsystems required by this command.
36  */
37  @SuppressWarnings("this-escape")
38  public TrapezoidProfileCommand(
39      TrapezoidProfile profile,
40      Consumer<State> output,
41      Supplier<State> goal,
42      Supplier<State> currentState,
43      Subsystem... requirements) {

```

C++

```

35  /**
36   * Creates a new TrapezoidProfileCommand that will execute the given
37   * TrapezoidalProfile. Output will be piped to the provided consumer function.
38   *
39   * @param profile      The motion profile to execute.
40   * @param output       The consumer for the profile output.
41   * @param goal         The supplier for the desired state
42   * @param currentState The current state
43   * @param requirements The list of requirements.
44   */
45  TrapezoidProfileCommand(frc::TrapezoidProfile<Distance> profile,
46                        std::function<void(State)> output,
47                        std::function<State()> goal,
48                        std::function<State()> currentState,
49                        Requirements requirements = {})

```

profil

The profile parameter is the TrapezoidProfile object that will be executed by the command. By passing this in, users specify the motion constraints of the profile that the command will use.

çıktı

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.

gereksinimler

Tüm satır içi komutlar gibi, TrapezoidProfileCommand kullanıcının alt sistem gereksinimlerini bir constructor parametresi olarak belirtmesine izin verir.

Tam TrapezoidProfileCommand Örneği

Pratikte kullanıldığında bir TrapezoidProfileSubsystem neye benziyor? Aşağıdaki örnekler DriveDistanceOffboard örnek projesinden alınmıştır (Java, C ++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/DriveDistanceOffboard>>):

Java

```
5 package edu.wpi.first.wpilibj.examples.drivedistanceoffboard.commands;
6
7 import edu.wpi.first.math.trajectory.TrapezoidProfile;
8 import edu.wpi.first.wpilibj.examples.drivedistanceoffboard.Constants.DriveConstants;
9 import edu.wpi.first.wpilibj.examples.drivedistanceoffboard.subsystems.DriveSubsystem;
10 import edu.wpi.first.wpilibj2.command.TrapezoidProfileCommand;
11
12 /** Drives a set distance using a motion profile. */
13 public class DriveDistanceProfiled extends TrapezoidProfileCommand {
14     /**
15      * Creates a new DriveDistanceProfiled command.
16      *
17      * @param meters The distance to drive.
18      * @param drive The drive subsystem to use.
19      */
20     public DriveDistanceProfiled(double meters, DriveSubsystem drive) {
21         super(
22             new TrapezoidProfile(
23                 // Limit the max acceleration and velocity
24                 new TrapezoidProfile.Constraints(
25                     DriveConstants.kMaxSpeedMetersPerSecond,
26                     DriveConstants.kMaxAccelerationMetersPerSecondSquared)),
27             // Pipe the profile state to the drive
28             setpointState -> drive.setDriveStates(setpointState, setpointState),
29             // End at desired position in meters; implicitly starts at 0
30             () -> new TrapezoidProfile.State(meters, 0),
31             // Current position
32             TrapezoidProfile.State::new,
33             // Require the drive
34             drive);
35         // Reset drive encoders since we're starting at 0
36         drive.resetEncoders();
37     }
38 }
```

C++ (Header)

```

5 #pragma once
6
7 #include <frc2/command/CommandHelper.h>
8 #include <frc2/command/TrapezoidProfileCommand.h>
9
10 #include "subsystems/DriveSubsystem.h"
11
12 class DriveDistanceProfiled
13 : public frc2::CommandHelper<frc2::TrapezoidProfileCommand<units::meters>,
14                               DriveDistanceProfiled> {
15 public:
16   DriveDistanceProfiled(units::meter_t distance, DriveSubsystem* drive);
17 };

```

C++ (Source)

```

5 #include "commands/DriveDistanceProfiled.h"
6
7 #include "Constants.h"
8
9 using namespace DriveConstants;
10
11 DriveDistanceProfiled::DriveDistanceProfiled(units::meter_t distance,
12                                              DriveSubsystem* drive)
13 : CommandHelper{
14     frc::TrapezoidProfile<units::meters>{
15         // Limit the max acceleration and velocity
16         {kMaxSpeed, kMaxAcceleration}},
17     // Pipe the profile state to the drive
18     [drive](auto setpointState) {
19         drive->SetDriveStates(setpointState, setpointState);
20     },
21     // End at desired position in meters; implicitly starts at 0
22     [distance] {
23         return frc::TrapezoidProfile<units::meters>::State{distance, 0_mps};
24     },
25     [] { return frc::TrapezoidProfile<units::meters>::State{}; },
26     // Require the drive
27     {drive}} {
28     // Reset drive encoders since we're starting at 0
29     drive->ResetEncoders();
30 }

```

And, for an *inlined* example:

Java

```

66 // Do the same thing as above when the 'B' button is pressed, but defined inline
67 m_driverController
68     .b()
69     .onTrue(
70         new TrapezoidProfileCommand(
71             new TrapezoidProfile(
72                 // Limit the max acceleration and velocity
73                 new TrapezoidProfile.Constraints(
74                     DriveConstants.kMaxSpeedMetersPerSecond,
75                     DriveConstants.kMaxAccelerationMetersPerSecondSquared)),
76                 // Pipe the profile state to the drive
77                 setpointState -> m_robotDrive.setDriveStates(setpointState,
↪ setpointState),
78                 // End at desired position in meters; implicitly starts at 0
79                 () -> new TrapezoidProfile.State(3, 0),
80                 // Current position
81                 TrapezoidProfile.State::new,
82                 // Require the drive
83                 m_robotDrive)
84                 .beforeStarting(m_robotDrive::resetEncoders)
85                 .withTimeout(10));

```

C++

```

37 DriveDistanceProfiled(3_m, &m_drive).WithTimeout(10_s));
38
39 // Do the same thing as above when the 'B' button is pressed, but defined
40 // inline
41 m_driverController.B().OnTrue(
42     frc::TrapezoidProfileCommand<units::meters>(
43         frc::TrapezoidProfile<units::meters>(
44             // Limit the max acceleration and velocity
45             {DriveConstants::kMaxSpeed, DriveConstants::kMaxAcceleration}),
46             // Pipe the profile state to the drive
47             [this](auto setpointState) {
48                 m_drive.SetDriveStates(setpointState, setpointState);
49             },
50             // End at desired position in meters; implicitly starts at 0
51             [] {
52                 return frc::TrapezoidProfile<units::meters>::State{3_m, 0_mps};
53             },
54             // Current position
55             [] { return frc::TrapezoidProfile<units::meters>::State{}; },
56             // Require the drive
57             {&m_drive})
58             // Convert to CommandPtr
59             .ToPtr()
60             .BeforeStarting(

```

26.12 Komut Tabanlı Hareket Profileleme ve PID'yi Birleştirme

Not: Bu komut tabanlı sarmalayıcılar tarafından kullanılan WPILib PID kontrol özelliklerinin bir açıklaması için, bakınız: ref: *docs / software / advanced-controls / controller / pidcontroller: WPILib'de PID Control*.

Ortak bir FRC ® kontrol çözümü, ayar noktası oluşturma için trapezoidal hareket profilini ayar noktası izleme için bir PID denetleyicisi ile eşleştirmektir. Bunu kolaylaştırmak için WPILib kendi: ref: *ProfiledPIDController <docs/software/advanced-controls/controllers/profiled-pidcontroller:Combining Motion Profiling and PID Control with ProfiledPIDController>* sınıfını içerir. Ekiplerin bu işlevselliği robotlarına entegre etmelerine yardımcı olmak için komut tabanlı çerçeve, *ProfiledPIDController* sınıfı için iki kolaylık sarmalayıcı içerir: denetleyiciyi bir alt sisteme entegre eden *ProfiledPIDSubsystem* ve *ProfiledPIDCommand*, denetleyiciyi bir komuta entegre eden.

26.12.1 ProfiledPIDSubsystem

Not: C ++ 'da, *ProfiledPIDSubsystem* sınıfı, açısal veya doğrusal olabilen mesafe ölçümleri için kullanılan birim türüne göre şablonlanır. İletilen değerler *zorunlu* mesafe birimleriyle tutarlı birimlere sahip olmalıdır, aksi takdirde derleme zamanı hatası atılır. C ++ birimleri hakkında daha fazla bilgi için bkz.: ref: *docs / software / basic-programlama / 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.

ProfiledPIDSubsystem oluşturma

Not: If *periodic* is overridden when inheriting from *ProfiledPIDSubsystem*, make sure to call *super.periodic()*! Otherwise, control functionality will not work properly.

ProfiledPIDSubsystem alt sınıfını oluştururken, kullanıcılar sınıfın normal işleminde kullanacağı işlevselliği sağlamak için iki soyut yöntemi geçersiz kılmalıdır:

getMeasurement ()

Java

```
protected abstract double getMeasurement();
```

C++

```
virtual Distance_t GetMeasurement() = 0;
```

getMeasurement yöntemi, süreç değişkeninin mevcut ölçümünü döndürür. ``PIDSubsystem`` bu yöntemi periodic() bloğundan otomatik olarak çağırarak ve değerini kontrol döngüsüne aktaracaktır.

Kullanıcılar, proses değişken ölçümü olarak kullanmak istedikleri sensör okumasını döndürmek için bu yöntemi geçersiz kılmalıdır.

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.

Kullanıcılar, son hesaplanmış kontrol çıkışını alt sistemlerinin motorlarına geçirmek için bu yöntemi geçersiz kılmalıdır.

Kontrol Cihazına Geçiş

Kullanıcılar ayrıca, alt sınıflarının süper sınıf yapıcı çağrısı aracılığıyla ProfiledPIDSubsystem temel sınıfa bir ProfiledPIDController geçirmelidir. Bu, PID kazanımlarını, hareket profili kısıtlamalarını ve periyodu (kullanıcı standart olmayan bir ana robot döngüsü periyodu kullanıyorsa) belirlemeye yarar.

getController() çağrısı yapılarak, yapıcı gövdesindeki denetleyicide ek değişiklikler (e.g. enabling continuous input- ör. sürekli girişi etkinleştirmek) yapılabilir.

ProfiledPIDSubsystem Kullanılması

Bir PIDSubsystem alt sınıfının bir örneği oluşturulduktan sonra, aşağıdaki yöntemler aracılığıyla komutlar tarafından kullanılabilir:

setGoal()

Not: Hedefi, örtük hedef hızı sıfır olan basit bir mesafeye ayarlamak isterseniz, tam hareket profili durumu yerine tek bir mesafe değeri alan bir `setGoal()` aşırı yükü vardır.

`setGoal()` yöntemi, ``PIDSubsystem`` in ayar noktasını ayarlamak için kullanılabilir. Alt sistem, tanımlanan çıktıyı kullanarak ayar noktasını otomatik olarak izleyecektir:

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});
```

etkinleştir (enable) ve devre dışı bırak (disable)

`enable()` ve `disable()` yöntemleri, ProfiledPIDSubsystem in otomatik kontrolünü etkinleştirir ve devre dışı bırakır. Alt sistem etkinleştirildiğinde, hareket profilini ve kontrol döngüsünü otomatik olarak çalıştıracak ve hedefe giden yolu izleyecektir. Devre dışı bırakıldığında hiçbir kontrol gerçekleştirilmez.

Ek olarak, `enable()` yöntemi dahili ProfiledPIDController i sıfırlar ve `disable()` yöntemi hem çıktı hem de ayar noktası ``olarak ayarlanmış şekilde kullanıcı tanımlı *useOutput()* yöntemini ``0``a ayarlar.

Tam ProfiledPIDSubsystem Örneği

Pratikte kullanıldığında bir PIDS alt sistemi neye benzer? Aşağıdaki örnekler ArmBot örnek projesinden alınmıştır (Java, C ++):

Java

```

5 package edu.wpi.first.wpilibj.examples.armbot.subsystems;
6
7 import edu.wpi.first.math.controller.ArmFeedforward;
8 import edu.wpi.first.math.controller.ProfiledPIDController;
9 import edu.wpi.first.math.trjectory.TrapezoidProfile;
10 import edu.wpi.first.wpilibj.Encoder;
11 import edu.wpi.first.wpilibj.examples.armbot.Constants.ArmConstants;
12 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
13 import edu.wpi.first.wpilibj2.command.ProfiledPIDSubsystem;
14
15 /** A robot arm subsystem that moves with a motion profile. */
16 public class ArmSubsystem extends ProfiledPIDSubsystem {
17     private final PWMSparkMax m_motor = new PWMSparkMax(ArmConstants.kMotorPort);
18     private final Encoder m_encoder =
19         new Encoder(ArmConstants.kEncoderPorts[0], ArmConstants.kEncoderPorts[1]);
20     private final ArmFeedforward m_feedforward =
21         new ArmFeedforward(
22             ArmConstants.kSVolts, ArmConstants.kGVolts,
23             ArmConstants.kVVoltSecondPerRad, ArmConstants.kAVoltSecondSquaredPerRad);
24
25     /** Create a new ArmSubsystem. */
26     public ArmSubsystem() {
27         super(
28             new ProfiledPIDController(
29                 ArmConstants.kP,
30                 0,
31                 0,
32                 new TrapezoidProfile.Constraints(
33                     ArmConstants.kMaxVelocityRadPerSecond,
34                     ArmConstants.kMaxAccelerationRadPerSecSquared)),
35             0);
36         m_encoder.setDistancePerPulse(ArmConstants.kEncoderDistancePerPulse);
37         // Start arm at rest in neutral position
38         setGoal(ArmConstants.kArmOffsetRads);
39     }
40
41     @Override
42     public void useOutput(double output, TrapezoidProfile.State setpoint) {
43         // Calculate the feedforward from the setpoint
44         double feedforward = m_feedforward.calculate(setpoint.position, setpoint.
45         ↪ velocity);
46         // Add the feedforward to the PID output to get the motor output
47         m_motor.setVoltage(output + feedforward);
48     }
49
50     @Override
51     public double getMeasurement() {
52         return m_encoder.getDistance() + ArmConstants.kArmOffsetRads;
53     }
54 }

```

C++ (Header)

```

5  #pragma once
6
7  #include <frc/Encoder.h>
8  #include <frc/controller/ArmFeedforward.h>
9  #include <frc/motorcontrol/PWMSparkMax.h>
10 #include <frc2/command/ProfiledPIDSubsystem.h>
11 #include <units/angle.h>
12
13 /**
14  * A robot arm subsystem that moves with a motion profile.
15  */
16 class ArmSubsystem : public frc2::ProfiledPIDSubsystem<units::radians> {
17     using State = frc::TrapezoidProfile<units::radians>::State;
18
19     public:
20         ArmSubsystem();
21
22         void UseOutput(double output, State setpoint) override;
23
24         units::radian_t GetMeasurement() override;
25
26     private:
27         frc::PWMSparkMax m_motor;
28         frc::Encoder m_encoder;
29         frc::ArmFeedforward m_feedforward;
30 };

```

C++ (Source)

```

5  #include "subsystems/ArmSubsystem.h"
6
7  #include "Constants.h"
8
9  using namespace ArmConstants;
10 using State = frc::TrapezoidProfile<units::radians>::State;
11
12 ArmSubsystem::ArmSubsystem()
13     : frc2::ProfiledPIDSubsystem<units::radians>(
14         frc::ProfiledPIDController<units::radians>(
15             kP, 0, 0, {kMaxVelocity, kMaxAcceleration})),
16         m_motor(kMotorPort),
17         m_encoder(kEncoderPorts[0], kEncoderPorts[1]),
18         m_feedforward(kS, kG, kV, kA) {
19     m_encoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
20     // Start arm in neutral position
21     SetGoal(State{kArmOffset, 0_rad_per_s});
22 }
23
24 void ArmSubsystem::UseOutput(double output, State setpoint) {
25     // Calculate the feedforward from the sepoint
26     units::volt_t feedforward =
27         m_feedforward.Calculate(setpoint.position, setpoint.velocity);
28     // Add the feedforward to the PID output to get the motor output

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
29 m_motor.SetVoltage(units::volt_t{output} + feedforward);
30 }
31
32 units::radian_t ArmSubsystem::GetMeasurement() {
33     return units::radian_t{m_encoder.GetDistance()} + kArmOffset;
34 }
```

Komutlarla bir ProfiledPIDSubsystem kullanmak çok basit olabilir:

Java

```
55 // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
56 m_driverController
57     .a()
58     .onTrue(
59         Commands.runOnce(
60             () -> {
61                 m_robotArm.setGoal(2);
62                 m_robotArm.enable();
63             },
64             m_robotArm));
```

C++

```
32 // Move the arm to 2 radians above horizontal when the 'A' button is pressed.
33 m_driverController.A().OnTrue(frc2::cmd::RunOnce(
34     [this] {
35         m_arm.SetGoal(2_rad);
36         m_arm.Enable();
37     },
38     {&m_arm}));
```

26.12.2 ProfiledPIDCommand

Not: C++ 'da, ProfiledPIDCommand sınıfı, açısal veya doğrusal olabilen mesafe ölçümleri için kullanılan birim türüne göre şablonlanır. İletilen değerler *zorunlu* mesafe birimleriyle tutarlı birimlere sahip olmalıdır, aksi takdirde derleme zamanı hatası atılır. C++ birimleri hakkında daha fazla bilgi için bkz. *C++ Ünite Kitablığı*.

The ProfiledPIDCommand class (Java, C++) allows users to easily create commands with a built-in ProfiledPIDController.

Bir PIDCommand oluşturma

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.

Not: If subclassing `ProfiledPIDCommand` and overriding any methods, make sure to call the super version of those methods! Otherwise, control functionality will not work properly.

Her iki durumda da, gerekli parametreleri yapıcısına ileterek bir `ProfiledPIDCommand` oluşturulur (bir alt sınıf tanımlıyorsa, bu bir *super ()* çağrısıyla yapılabilir):

Java

```

29  /**
30   * Creates a new PIDCommand, which controls the given output with a
    ↪ ProfiledPIDController. Goal
31   * velocity is specified.
32   *
33   * @param controller the controller that controls the output.
34   * @param measurementSource the measurement of the process variable
35   * @param goalSource the controller's goal
36   * @param useOutput the controller's output
37   * @param requirements the subsystems required by this command
38   */
39  public ProfiledPIDCommand(
40      ProfiledPIDController controller,
41      DoubleSupplier measurementSource,
42      Supplier<State> goalSource,
43      BiConsumer<Double, State> useOutput,
44      Subsystem... requirements) {

```

C++

```

38  /**
39   * Creates a new PIDCommand, which controls the given output with a
40   * ProfiledPIDController.
41   *
42   * @param controller the controller that controls the output.
43   * @param measurementSource the measurement of the process variable
44   * @param goalSource the controller's goal
45   * @param useOutput the controller's output
46   * @param requirements the subsystems required by this command
47   */
48  ProfiledPIDCommand(frc::ProfiledPIDController<Distance> controller,
49      std::function<Distance_t()> measurementSource,
50      std::function<State()> goalSource,
51      std::function<void(double, State)> useOutput,
52      Requirements requirements = {})

```

Kontrolör - Controller

Controller parametresi, komut tarafından kullanılacak olan `ProfiledPIDController` nesnesidir. Bunu geçerek, kullanıcılar PID kazanımlarını, hareket profili kısıtlamalarını ve kontrolör için süreyi belirleyebilir (eğer kullanıcı standart olmayan bir ana robot döngü periyodu kullanıyorsa).

`ProfiledPIDCommand` alt sınıfını oluştururken, `getController()` çağrısı yapılarak yapıcı gövdesindeki denetleyicide ek değişiklikler (ör. Sürekli girişi etkinleştirmek) yapılabilir.

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`.

`ProfiledPIDCommand` alt sınıfını oluştururken, ileri düzey kullanıcılar, sınıfın `m_measurement` alanını değiştirerek ölçüm tedarikçisini daha da değiştirebilir.

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.

`ProfiledPIDCommand` alt sınıfını oluştururken, ileri düzey kullanıcılar, sınıfın `m_goal` alanını değiştirerek ayar noktası tedarikçisini daha da değiştirebilir.

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`.

`ProfiledPIDCommand` alt sınıfını oluştururken, ileri düzey kullanıcılar, sınıfın `m_useOutput` alanını değiştirerek çıktı tüketicisini daha da değiştirebilir.

Gereksinimler

Tüm satır içi olmayan komutlar gibi, `ProfiledPIDCommand` kullanıcının alt sistem gereksinimlerini bir yapıcı parametresi olarak belirtmesine izin verir.

Tam ProfiledPIDCommand Örneği

Pratikte kullanıldığında bir ProfiledPIDCommand neye benziyor? Aşağıdaki örnekler GyroDriveCommands örnek projesinden alınmıştır (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/GyroDriveCommands>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/GyroDriveCommands>>):

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(
24                 DriveConstants.kTurnP,
25                 DriveConstants.kTurnI,
26                 DriveConstants.kTurnD,
27                 new TrapezoidProfile.Constraints(
28                     DriveConstants.kMaxTurnRateDegPerS,
29                     DriveConstants.kMaxTurnAccelerationDegPerSSquared)),
30             // Close loop on heading
31             drive::getHeading,
32             // Set reference to target
33             targetAngleDegrees,
34             // Pipe output to turn robot
35             (output, setpoint) -> drive.arcadeDrive(0, output),
36             // Require the drive
37             drive);
38
39         // Set the controller to be continuous (because it is an angle controller)
40         getController().enableContinuousInput(-180, 180);
41         // Set the controller tolerance - the delta tolerance ensures the robot is
42         // stationary at the
43         // setpoint before it is considered as having reached the reference
44         getController().setTolerance(DriveConstants.kTurnToleranceDeg, DriveConstants.
45         kTurnRateToleranceDegPerS);
46     }

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

47  @Override
48  public boolean isFinished() {
49      // End when the controller is at the reference.
50      return getController().atGoal();
51  }
52  }

```

C++ (Header)

```

5  #pragma once
6
7  #include <frc2/command/CommandHelper.h>
8  #include <frc2/command/ProfiledPIDCommand.h>
9
10 #include "subsystems/DriveSubsystem.h"
11
12 /**
13  * A command that will turn the robot to the specified angle using a motion
14  * profile.
15  */
16 class TurnToAngleProfiled
17     : public frc2::CommandHelper<frc2::ProfiledPIDCommand<units::radians>,
18                               TurnToAngleProfiled> {
19 public:
20     /**
21      * Turns to robot to the specified angle using a motion profile.
22      *
23      * @param targetAngleDegrees The angle to turn to
24      * @param drive               The drive subsystem to use
25      */
26     TurnToAngleProfiled(units::degree_t targetAngleDegrees,
27                         DriveSubsystem* drive);
28
29     bool IsFinished() override;
30 };

```

C++ (Source)

```

5  #include "commands/TurnToAngleProfiled.h"
6
7  #include <frc/controller/ProfiledPIDController.h>
8
9  using namespace DriveConstants;
10
11 TurnToAngleProfiled::TurnToAngleProfiled(units::degree_t target,
12                                          DriveSubsystem* drive)
13     : CommandHelper{
14         frc::ProfiledPIDController<units::radians>{
15             kTurnP, kTurnI, kTurnD, {kMaxTurnRate, kMaxTurnAcceleration}},
16         // Close loop on heading
17         [drive] { return drive->GetHeading(); },
18         // Set reference to target

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

19     target,
20     // Pipe output to turn robot
21     [drive](double output, auto setpointState) {
22         drive->ArcadeDrive(0, output);
23     },
24     // Require the drive
25     {drive}} {
26     // Set the controller to be continuous (because it is an angle controller)
27     GetController().EnableContinuousInput(-180_deg, 180_deg);
28     // Set the controller tolerance - the delta tolerance ensures the robot is
29     // stationary at the setpoint before it is considered as having reached the
30     // reference
31     GetController().SetTolerance(kTurnTolerance, kTurnRateTolerance);
32
33     AddRequirements(drive);
34 }
35
36 bool TurnToAngleProfiled::IsFinished() {
37     return GetController().AtGoal();
38 }

```

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++)

Uyarı: 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++](#).

27.1 Introduction to Kinematics and The ChassisSpeeds Class

Not: Kinematics and odometry uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.1.1 Kinematik nedir?

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 Odometri nedir?

Odometri, robotun sahadaki konumunun bir tahminini oluşturmak için robot üzerindeki sensörlerin kullanılmasını içerir. FRC'de bu sensörler tipik olarak birkaç kodlayıcıdır (tam sayı sürücü tipine bağlıdır) ve robot açısını ölçmek için bir jiroskoptur. Odometri sınıfları, robotun sahadaki konumunun bir tahminini oluşturmak için hızlarla (ve sapma durumunda açılar) ilgili periyodik kullanıcı girdileriyle birlikte kinematik sınıflarını kullanır.

27.1.3 The ChassisSpeeds Class

ChassisSpeeds nesnesi, yeni WPILib kinematik ve odometri takımı için çok önemlidir. ChassisSpeeds nesnesi, bir robot kasasının hızlarını temsil eder. Bu yapının üç bileşeni vardır:

- `vx` : Robotun x (ileri) yönündeki hızı.
- `vy` : Robotun y (yana doğru) yönündeki hızı. (Pozitif değerler, robotun sola hareket ettiği anlamına gelir).
- `omega` : Robotun saniyede radyan cinsinden açısal hızı.

Not: Holonomik olmayan bir aktarma organı (yani, yana doğru hareket edemeyen bir aktarma organı, örneğin bir diferansiyel tahrik), yanlara doğru hareket edememesi nedeniyle sıfır `vy` bileşenine sahip olacaktır.

27.1.4 ChassisSpeeds nesnesi oluşturma

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 Saha-bağıl hızlardan ChassisSpeeds Nesnesi Oluşturma

Robot açısı verildiğinde, bir dizi alana bağlı hızdan bir ChassisSpeeds nesnesi de oluşturulabilir. Bu, alana göre bir dizi istenen hızı (örneğin, karşı ittifak istasyonuna doğru ve sağ alan sınırına doğru) robot çerçevesine göre hızları temsil eden bir ChassisSpeeds nesnesine dönüştürür. Bu, bir sapma veya mecanum tahrik robotu için saha odaklı kontrolleri uygulamak için kullanışlıdır.

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))
```

Not: Açısal hız, açık bir şekilde “sahaya göre” olarak belirtilmemiştir çünkü açısal hız, bir alan perspektifinden veya bir robot perspektifinden ölçülen ile aynıdır.

27.2 Differential Drive Kinematiği

DifferentialDriveKinematics sınıfı, ChassisSpeeds nesnesi ile bir diferansiyel sürücü robotunun sol ve sağ tarafları için hızları içeren bir DifferentialDriveWheelSpeeds nesnesi arasında dönüşüm sağlayan kullanışlı bir araçtır. .

27.2.1 Kinematik Nesnesini Oluşturmak

DifferentialDriveKinematics nesnesi, robotun iz genişliği olan bir yapıcı argümanını kabul eder. Bu, bir diferansiyel sürücü üzerindeki iki tekerlek seti arasındaki mesafeyi temsil eder.

Not: 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 Şasi Hızlarını Tekerlek Hızlarına Dönüştürme

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 (v_x) and an angular velocity (ω) 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 Tekerlek Hızlarını Şasi Hızlarına Dönüştürme

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
# 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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
# Angular velocity
angularVelocity = chassisSpeeds.omega
```

27.3 Differential Drive Odometrisi

Bir kullanıcı, *odometry* gerçekleştirmek için diferansiyel sürücü kinematik sınıflarını kullanabilir. WPILib, sahada diferansiyel tahrik robotunun konumunu izlemek için kullanılacak bir `DifferentialDriveOdometry` sınıfı içerir.

Not: Bu yöntemde yalnızca kodlayıcılar ve bir jiroskop kullanıldığı için, robotun sahadaki konumunun tahmini, özellikle oyun sırasında diğer robotlarla temas kurduğunda zamanla kayacaktır. Bununla birlikte, odometri genellikle bağımsız dönemde çok doğrudur.

27.3.1 Odometri Nesnesini Oluşturma

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 $x = 0$, $y = 0$, $\theta = 0$.

Not: 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 Robot Duruşunu Güncelleme

Robotun sahadaki konumunu güncellemek için update yöntemi kullanılabilir. Bu yöntem periyodik olarak, tercihen periodic() yönteminde *Subsystem* ile çağrılmalıdır. update yöntemi, robotun yeni güncellenmiş konumunu döndürür. Bu yöntem, sol kodlayıcı mesafesi ve sağ kodlayıcı mesafesi ile birlikte robotun cayo açısını alır.

Not: 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 Robot Duruşunu Sıfırlama

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.

Önemli: 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.

Not: A full example of a differential drive robot with odometry is available here: [C++ / Java / Python](#)

In addition, the `GetPose` (C++) / `getPoseMeters` (Java / Python) methods can be used to retrieve the current robot pose without an update.

27.4 Swerve Drive Kinematığı

``SwerveDriveKinematics`` sınıfı, bir ``ChassisSpeeds`` nesnesi ile bir swerve sürücü robotunun her bir dönüş modülü için hızları ve açıları içeren birkaç ``SwerveModuleState`` nesnesi arasında dönüşüm sağlayan kullanışlı bir araçtır.

Not: Swerve drive kinematics uses a common coordinate system. You may wish to reference the [Coordinate System](#) section for details.

27.4.1 Swerve modülü durum sınıfı

SwerveModuleState sınıfı, bir swerve sürücüsünün tekil modülünün hızı ve açısı hakkında bilgi içerir. Bir SwerveModuleState in constructor'ı iki argüman alır: modüldeki çarkın hızı ve modülün açısı.

Not: 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.

Not: 0 açı, öne bakan modüllere karşılık gelir.

27.4.2 Kinematik Nesnesinin Oluşturulması

SwerveDriveKinematics sınıfı, değişken sayıda yapıcı argümanı kabul eder, her argüman robot merkezine göre bir swerve modülünün konumu olur (Translation2d olarak. Constructor argümanlarının sayısı, swerve modüllerinin sayısına karşılık gelir.

Not: Swerve sürücüsünün 2 veya daha fazla modülü olmalıdır.

Not: C++ 'da, sınıf modül sayısına göre şablonlanır. Bu nedenle, bir sınıfın üye değişkeni olarak bir SwerveDriveKinematics nesnesi oluştururken, modül sayısı bir şablon argümanı olarak iletilmelidir. Örneğin, dört modüllü tipik bir swerve sürücü için kinematik nesnesi şu şekilde oluşturulmalıdır: `frc::SwerveDriveKinematics<4> m_kinematics{...}`.

Modüllerin konumları, robotun merkezine göre olmalıdır. Pozitif x değerleri robotun önüne doğru hareket etmeyi temsil ederken, pozitif y değerleri robotun soluna doğru hareket etmeyi temsil eder.

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 Şase hızlarını modül durumlarına dönüştürme

The `toSwerveModuleStates(ChassisSpeeds speeds)` (Java / Python) / `ToSwerveModuleStates(ChassisSpeeds speeds)` (C++) method should be used to convert a `ChassisSpeeds` object to 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.

Bu yöntemle döndürülen dizideki öğeler, kinematik nesnesinin oluşturulduğu sırayla aynıdır. Örneğin, kinematik nesnesi ön sol modül konumu, ön sağ modül konumu, arka sol modül konumu ve bu sırayla sağ arka modül konumu ile oluşturulmuşsa, dizideki öğeler ön sol modül durumu, ön bu sırayla sağ modül durumu, arka sol modül durumu ve arka sağ modül durumu olacaktır.

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)
```

Modül açısı optimizasyonu

Başlıktaki değişikliği en aza indirmek için. SwerveModuleState sınıfı, belirli bir SwerveModuleState in hız ve açı ayar noktasını “optimize etmek” için kullanılan statik bir ``optimize()`` (Java) / Optimize() (C++) yöntemini içerir. Örneğin, belirli bir modülün ters kinematikten açısal set noktası 90 derece ise, ancak mevcut açınız -89 derecedir, bu yöntem modülün ayar noktasının hızını otomatik olarak geçersiz kılar ve modülün gitmesi gereken mesafeyi azaltmak için açısal ayar noktasını -90 derece yapar.

Bu yöntem iki parametre alır: istenen durum (genellikle toSwerveModuleStates metodundan) ve geçerli açı. Geri bildirim kontrol döngünüzde ayar noktası olarak kullanabileceğiniz yeni optimize edilmiş durumu döndürecektir.

JAVA

```
var frontLeftOptimized = SwerveModuleState.optimize(frontLeft,
    new Rotation2d(m_turningEncoder.getDistance()));
```

C++

```
auto flOptimized = frc::SwerveModuleState::Optimize(fl,
    units::radian_t(m_turningEncoder.GetDistance()));
```

PYTHON

```
from wpimath.kinematics import SwerveModuleState
from wpimath.geometry import Rotation2d

frontLeftOptimized = SwerveModuleState.optimize(frontLeft,
    Rotation2d(self.m_turningEncoder.getDistance()))
```

Field-oriented / Saha-odaklı sürüş

Recall bir ChassisSpeeds nesnesinin istenen saha odaklı hızlardan oluşturulabileceğini hatırlayın. Bu özellik, istenen saha odaklı hızlardan modül durumlarını elde etmek için kullanılabilir.

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));
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
// 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)
```

Özel döndürme merkezlerini kullanma

Bazen, belirli kaçamak manevralar için belirli bir köşenin etrafında dönmek istenebilir. Bu tür davranış WPILib sınıfları tarafından da desteklenir. Aynı ``ToSwerveModuleStates()`` yöntemi, dönüş merkezi için ikinci bir parametre kabul eder (``Translation2d`` olarak). Tıpkı tekerlek konumlarında olduğu gibi, dönme merkezini temsil eden ``Translation2d`` de robot merkezine göre olmalıdır.

Not: Tüm robotlar sabit bir çerçeve olduğundan, ChassisSpeeds nesnesinden sağlanan vx ve vy hızları, robotun tamamı için geçerli olmaya devam edecektir. Bununla birlikte, ChassisSpeeds nesnesindeki omega, dönüş merkezinden ölçülecektir.

Örneğin, belirli bir modülde dönme merkezi ayarlanabilir ve sağlanan ChassisSpeeds nesnesinin bir vx ve vy si sıfır ve sıfır olmayan bir omega ise robot, söz konusu dönüş modülü etrafında dönüyormuş gibi görünecektir.

27.4.4 Modül durumlarını şase hızlarına dönüştürme

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

```
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(
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    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

```

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(

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

std::vector{
    frontLeftState,
    frontRightState,
    backLeftState,
    backRightState
}
);
}
};

```

PYTHON

```

import ntcore
from wpimath.kinematics import SwerveModuleState

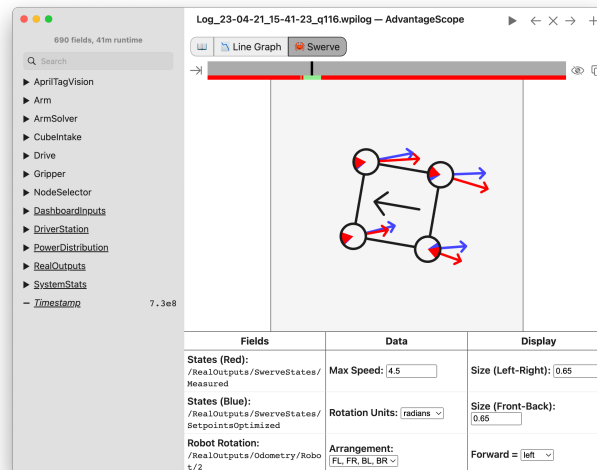
# get the default instance of NetworkTables
nt = ntcore.NetworkTableInstance.getDefault()

# 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 Swerve Sürüş Odometrisi

Bir kullanıcı gerçekleştirmek için swerve sürücü kinematik sınıflarını kullanabilir [odometry](#). WPILib, sahada bir Swerve sürüş robotunun konumunu izlemek için kullanılacak bir SwerveDriveOdometry sınıfı içerir.

Not: Bu yöntem yalnızca kodlayıcılar ve bir jiroskop kullandığından, robotun sahadaki konumunun tahmini, özellikle oyun sırasında diğer robotlarla temas kurduğunda zamanla kayacaktır. Bununla birlikte, odometri genellikle özerk dönemde çok doğrudur.

27.5.1 Odometri nesnesini oluşturma

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 $x = 0$, $y = 0$, $\theta = 0$.

Not: 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
);
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
// 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)
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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 Robot duruşunu güncelleme

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.

Bu update metodu periyodik olarak, tercihen `periodic()` metodunda bir *Subsystem* ile çağrılmalıdır. update metodu, robotun yeni güncellenmiş pozunu döndürür.

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 Robot Duruşunu Sıfırlama

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.

Önemli: 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.

Not: 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 Mecanum Drive Kinematik

MecanumDriveKinematics sınıfı, bir ChassisSpeeds nesnesi ile bir mecanum sürücüsündeki dört tekerleğin her biri için hızları içeren bir MecanumDriveWheelSpeeds nesnesi arasında dönüşüm sağlayan kullanışlı bir araçtır.

Not: Mecanum kinematics uses a common coordinate system. You may wish to reference the *Coordinate System* section for details.

27.6.1 Kinematik Nesnesinin Oluşturulması

MecanumDriveKinematics sınıfı dört constructor argümanı kabul eder, her argüman robot merkezine göre bir tekerleğin konumu olur (Translation2d olarak). Argümanların sırası sol ön, sağ ön, sol arka ve sağ arka şeklindedir. Tekerleklerin yerleri, robotun merkezine göre olmalıdır. Pozitif x değerleri robotun önüne doğru hareket etmeyi temsil ederken, pozitif y değerleri robotun soluna doğru hareket etmeyi temsil eder.

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 Şasi Hızlarını Tekerlek Hızlarına Dönüştürme

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
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)
```

Field-oriented / Saha-odaklı sürüş

Recall bir ChassisSpeeds nesnesinin istenen saha odaklı hızlardan oluşturulabileceğini hatırlayın. Bu özellik, istenen saha odaklı hızlardan tekerlek hızları elde etmek için kullanılabilir.

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)

```

Özel döndürme merkezlerini kullanma

Bazen, belirli bir kaçamak manevraları için belirli bir köşenin etrafında dönmek istenebilir. Bu tür davranış, WPILib sınıfları tarafından da desteklenir. Aynı ToWheelSpeeds()'' yöntemi, dönüş merkezi için ikinci bir parametre kabul eder (``Translation2d olarak). Tıpkı tekerlek konumları gibi, dönme merkezini temsil eden Translation2d de robot merkezine göre olmalıdır.

Not: Tüm robotlar sabit bir çerçeve olduğundan, ChassisSpeeds nesnesinden sağlanan vx ve vy hızları, robotun tamamı için geçerli olmaya devam edecektir. Bununla birlikte, ChassisSpeeds nesnesindeki omega, dönüş merkezinden ölçülecektir.

Örneğin, belirli bir tekerlek üzerinde dönme merkezi ayarlanabilir ve sağlanan ChassisSpeeds nesnesi sıfır olan bir vx ve vy ye ve sıfır olmayan bir omega ya sahipse, robot, o tekerleğin etrafında dönüyormuş gibi görünecektir.

27.6.3 Tekerlek hızlarını şasi hızlarına dönüştürme

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

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
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 Mecanum Sürüş Odometrisi

Bir kullanıcı, aşağıdakileri gerçekleştirmek için mecanum drive kinematics - mecanum sürücü kinematik sınıflarını kullanabilir *odometry*. WPILib, sahada bir mecanum tahrik robotunun konumunu izlemek için kullanılabilecek bir MecanumDriveOdometry sınıfı içerir.

Not: Bu yöntemde yalnızca kodlayıcılar ve bir jiroskop kullanıldığı için, robotun sahadaki konumunun tahmini, özellikle oyun sırasında diğer robotlarla temas kurduğunda zamanla kayacaktır. Bununla birlikte, odometri genellikle bağımsız dönemde çok doğrudur.

27.7.1 Odometri nesnesini oluşturma

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 $x = 0$, $y = 0$, $\theta = 0$.

Not: 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.
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)

# 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(),
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

MecanumDriveWheelPositions(
    self.frontLeftEncoder.getDistance(), self.frontRightEncoder.getDistance(),
    self.backLeftEncoder.getDistance(), self.backRightEncoder.getDistance()
),
Pose2d(5.0, 13.5, Rotation2d())
)

```

27.7.2 Robot duruşunu güncelleme

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 Robot Duruşunu Sıfırlama

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.

Önemli: 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.

Not: 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.

This section outlines the details of using the NetworkTables (v4) API to communicate information across the robot network.

Önemli: 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 NetworkTables nedir

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.

Not: NetworkTables has changed substantially in 2023. For more information on migrating pre-2023 code to use the new features, see [Migrating from NetworkTables 3.0 to NetworkTables 4.0](#).

28.1.1 NetworkTables Concepts

First, let's define some terms:

- **Topic:** a named data channel. Topics have a fixed data type (for the lifetime of the topic) and *mutable* properties.
- **Publisher:** defines the topic and creates and sends timestamped data values.
- **Subscriber:** receives timestamped data value updates to one or more topics.
- **Entry:** a combined publisher and subscriber. The subscriber is always active, but the publisher is not created until a publish operation is performed (e.g. a value is “set”, aka published, on the entry). This may be more convenient than maintaining a separate publisher and subscriber.
- **Property:** named information (metadata) about a topic stored and updated separately from the topic's data. A topic may have any number of properties. A property's value can be any data type that can be represented in JSON.

NetworkTables supports a range of data types, including *boolean*, numeric, string, and arrays of those types. Supported numeric data types are single or double precision *floating point*, or 64-bit integer. There is also the option of storing raw data (an array of bytes), which can be used for representing binary encoded structured data. Types are represented as strings for efficiency reasons. There is also an *enumeration* for the most common types in the NetworkTables API.

Topics are created when the first publisher announces the topic and are removed when the last publisher stops publishing. It's possible to subscribe to a topic that has not yet been created/published.

Topics have properties. Properties are initially set by the first publisher, but may be changed at any time. Similarly to values, property changes to a topic are propagated to all subscribers to that topic. Properties are structured data (JSON), but at the top level are simply a key/value store (a JSON map). Some properties have defined behavior, but arbitrary ones can be set by the application.

Publishers specify the topic's data type; while there can be multiple publishers to a single topic, they must all be publishing the same data type. This is enforced by the NetworkTables server (the first publisher “wins”). Typically single-topic subscribers also specify what data type they're expecting to receive on a topic and thus won't receive value updates of other types.

The [Network Tables Protocol Specification](#) contains detailed documentation on the current wire protocol.

28.1.2 Retained and Persistent Topics

While by default topics are *transitory* and disappear after the last publisher stops publishing, topics can be marked as *retained* (via setting the “retained” property to true) to prevent them from disappearing. For retained topics, the server acts as an implicit publisher of the last value, and will keep doing so as long as the server is running. This is primarily useful for configuration values; e.g. an autonomous mode selection published by a dashboard should set the topic as retained so its value is preserved in case the dashboard disconnects.

Additionally, topics can be marked as *persistent* via setting the “persistent” property to true. These operate similarly to retained topics, but in addition, persistent topic values are automa-

tically saved to a file on the server and when the server starts up again, the topic is created and its last value is published by the server.

28.1.3 Value Propagation

The server keeps a copy of the last published value for every topic. When a subscriber initially subscribes to a topic, the server sends the last published value. After that initial value, new value updates are communicated to subscribers each time the publisher sends a new value.

NetworkTables is a client/server system; clients do not talk directly to each other, but rather communicate via the server. Typically, the robot program is the server, and other pieces of software on other computers (e.g. the driver station or a coprocessor) are clients that connect to it. Thus, when a coprocessor (client) publishes a value, the value is sent first from the coprocessor (client) to the robot program (server), and then the robot program distributes that value to any subscribers (e.g. the robot program local program, or other clients such as dashboards).

The server does not send topic changes or value updates to clients that have not subscribed to the topic.

By default, NetworkTables sends value updates periodically, batching the data to help limit the number of small packets being sent over the network. Also, by default, only the most recent value is transmitted; any intermediate value changes made between network transmissions are discarded. This behavior can be changed via publish/subscribe options—publishers and subscribers can indicate that all value updates should be preserved and communicated via the “send all” option. In addition, it is possible to force NetworkTables to “flush” all current updates to the network; this is useful for minimizing latency.

28.1.4 Timestamps

All NetworkTable value updates are timestamped at the time they are published. Timestamps in NetworkTables are measured in integer microseconds.

NetworkTables automatically synchronizes time between the server and clients. Each client maintains an offset between the client local time and the server time, so when a client publishes a value, it stores a timestamp in local time and calculates the equivalent server timestamp. The server timestamp is what is communicated over the network to any subscribers. This makes it possible e.g. for a robot program to get a reasonable estimation of the time when a value was published on a coprocessor relative to the current time.

Because of this, two timestamps are visible through the API: a server timestamp indicating the time (estimated) on the server, and a local timestamp indicating the time on the client. When the RoboRIO is the NetworkTables server, the server timestamp is the same as the FPGA timestamp returned by `Timer.getFPGATimestamp()` (except the units are different: NetworkTables uses microseconds, while `getFPGATimestamp()` returns seconds).

28.1.5 NetworkTables Organization

Data is organized in NetworkTables in a hierarchy much like a filesystem's folders and files. There can be multiple subtables (folders) and topics (files) that may be nested in whatever way fits the data organization desired. At the top level (NetworkTableInstance: [Java](#), [C++](#), [Python](#)), topic names are handled similar to absolute paths in a filesystem: subtables are represented as a long topic name with slashes ("/") separating the nested subtable and value names. A NetworkTable ([Java](#), [C++](#), [Python](#)) object represents a single subtable (folder), so topic names are relative to the NetworkTable's base path: e.g. for a root table called "SmartDashboard" with a topic named "xValue", the same topic can be accessed via NetworkTableInstance as a topic named "/SmartDashboard/xValue". However, unlike a filesystem, subtables don't really exist in the same way folders do, as there is no way to represent an empty subtable on the network—a subtable "appears" only as long as there are topics published within it.

[OutlineViewer](#) is a utility for exploring the values stored in NetworkTables, and can show either a flat view (topics with absolute paths) or a nested view (subtables and topics).

There are some default tables that are created automatically when a robot program starts up:

Tablo ismi	Kullanım
/SmartDashboard	SmartDashboard.put() yöntem kümesi kullanılarak SmartDashboard veya Shuffleboard'a yazılan değerleri depolamak için kullanılır.
/LiveWindow	Test modu (Sürücü İstasyonunda Test) değerlerini saklamak için kullanılır. Tipik olarak bunlar Alt sistemler ve ilgili sensörler ve aktüatörlerdir.
/FMSInfo	Driver Station ve Saha Yönetim Sisteminden gelen şu anda devam eden maç hakkında bilgiler

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. Boolean-Publisher: [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
    }
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

dblPub.set(1.0);
dblPub.set(2.0, 0); // 0 = use current time

// publish a value with a specific timestamp; NetworkTablesJNI.now() can
// be used to get the current time. On the roboRIO, this is the same as
// the FPGA timestamp (e.g. RobotController.getFPGATime())
long time = NetworkTablesJNI.now();
dblPub.set(3.0, time);

// publishers also implement the appropriate Consumer functional
interface;
// this example assumes void myFunc(DoubleConsumer func) exists
myFunc(dblPub);
}

// often not required in robot code, unless this class doesn't exist for
// the lifetime of the entire robot program, in which case close() needs
to be
// called to stop publishing
public void close() {
    // stop publishing
    dblPub.close();
}
}

```

C++

```

class Example {
    // the publisher is an instance variable so its lifetime matches that of
    the class
    // publishing is automatically stopped when dblPub is destroyed by the
    class destructor
    nt::DoublePublisher dblPub;

public:
    explicit Example(nt::DoubleTopic dblTopic) {
        // start publishing; the return value must be retained (in this case, via
        // an instance variable)
        dblPub = dblTopic.Publish();

        // publish options may be specified using PubSubOptions
        dblPub = dblTopic.Publish({.keepDuplicates = true});

        // PublishEx provides additional options such as setting initial
        // properties and using a custom type string. Using a custom type string
        for
        // types other than raw and string is not recommended. The properties
        must
        // be a JSON map.
        dblPub = dblTopic.PublishEx("double", {{"myprop", 5}});
    }

    void Periodic() {
        // publish a default value
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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
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.
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

self.dblPub = dblTopic.publishEx("double", '{"myprop": 5}')
```

```

def periodic(self):
    # publish a default value
    self.dblPub.setDefault(0.0)

    # publish a value with current timestamp
    self.dblPub.set(1.0)
    self.dblPub.set(2.0, 0) # 0 = use current time

    # publish a value with a specific timestamp with microsecond
    ↪resolution.
    # On the roboRIO, this is the same as the FPGA timestamp (e.g.
    # RobotController.getFPGATime())
    self.dblPub.set(3.0, ntcore._now())

    # often not required in robot code, unless this class doesn't exist for
    # the lifetime of the entire robot program, in which case close() needs
    ↪to be
    # called to stop publishing
    def close(self):
        # stop publishing
        self.dblPub.close()
```

28.3.2 Subscribing to a Topic

A *subscriber* receives value updates made to a topic. Similar to publishers, NetworkTable subscribers are represented as type-specific Subscriber classes (e.g. BooleanSubscriber: Java, C++, Python) that must be stored somewhere to continue subscribing.

Subscribers have a range of different ways to read received values. It's possible to just read the most recent value using `get()`, read the most recent value, along with its timestamp, using `getAtomic()`, or get an array of all value changes since the last call using `readQueue()` or `readQueueValues()`.

Java

```

public class Example {
    // the subscriber is an instance variable so its lifetime matches that of
    ↪the class
    final DoubleSubscriber dblSub;

    public Example(DoubleTopic dblTopic) {
        // start subscribing; the return value must be retained.
        // the parameter is the default value if no value is available when
        ↪get() is called
        dblSub = dblTopic.subscribe(0.0);

        // subscribe options may be specified using PubSubOption
        dblSub =
            dblTopic.subscribe(0.0, PubSubOption.keepDuplicates(true),
            ↪PubSubOption.pollStorage(10));
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    // subscribeEx provides the options of using a custom type string.
    // Using a custom type string for types other than raw and string is not
    →recommended.
    dblSub = dblTopic.subscribeEx("double", 0.0);
}

public void periodic() {
    // simple get of most recent value; if no value has been published,
    // returns the default value passed to the subscribe() function
    double val = dblSub.get();

    // get the most recent value; if no value has been published, returns
    // the passed-in default value
    double val = dblSub.get(-1.0);

    // subscribers also implement the appropriate Supplier interface, e.g.
    →DoubleSupplier
    double val = dblSub.getAsDouble();

    // get the most recent value, along with its timestamp
    TimestampedDouble tsVal = dblSub.getAtomic();

    // read all value changes since the last call to readQueue/
    →readQueueValues
    // readQueue() returns timestamps; readQueueValues() does not.
    TimestampedDouble[] tsUpdates = dblSub.readQueue();
    double[] valUpdates = dblSub.readQueueValues();
}

// often not required in robot code, unless this class doesn't exist for
// the lifetime of the entire robot program, in which case close() needs
    →to be
    // called to stop subscribing
    public void close() {
        // stop subscribing
        dblSub.close();
    }
}

```

C++

```

class Example {
    // the subscriber is an instance variable so its lifetime matches that of
    →the class
    // subscribing is automatically stopped when dblSub is destroyed by the
    →class destructor
    nt::DoubleSubscriber dblSub;

public:
    explicit Example(nt::DoubleTopic dblTopic) {
        // start subscribing; the return value must be retained.
        // the parameter is the default value if no value is available when
        →get() is called
        dblSub = dblTopic.Subscribe(0.0);
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

// subscribe options may be specified using PubSubOptions
dblSub =
    dblTopic.subscribe(0.0,
        {.pollStorage = 10, .keepDuplicates = true});

// SubscribeEx provides the options of using a custom type string.
// Using a custom type string for types other than raw and string is not
↳recommended.
dblSub = dblTopic.SubscribeEx("double", 0.0);
}

void Periodic() {
    // simple get of most recent value; if no value has been published,
    // returns the default value passed to the Subscribe() function
    double val = dblSub.Get();

    // get the most recent value; if no value has been published, returns
    // the passed-in default value
    double val = dblSub.Get(-1.0);

    // get the most recent value, along with its timestamp
    nt::TimestampedDouble tsVal = dblSub.GetAtomic();

    // read all value changes since the last call to ReadQueue/
    ↳ReadQueueValues
    // ReadQueue() returns timestamps; ReadQueueValues() does not.
    std::vector<nt::TimestampedDouble> tsUpdates = dblSub.ReadQueue();
    std::vector<double> valUpdates = dblSub.ReadQueueValues();
}
};

```

C++ (Handle-based)

```

class Example {
    // the subscriber is an instance variable, but since it's a handle, it's
    // not automatically released, so we need a destructor
    NT_Subscriber dblSub;

public:
    explicit Example(NT_Topic dblTopic) {
        // start subscribing
        // Using a custom type string for types other than raw and string is not
        ↳recommended.
        dblSub = nt::Subscribe(dblTopic, NT_DOUBLE, "double");

        // subscribe options may be specified using PubSubOptions
        dblSub =
            nt::Subscribe(dblTopic, NT_DOUBLE, "double",
                {.pollStorage = 10, .keepDuplicates = true});
    }

    void Periodic() {
        // get the most recent value; if no value has been published, returns
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

// the passed-in default value
double val = nt::GetDouble(dblSub, 0.0);

// get the most recent value, along with its timestamp
nt::TimestampedDouble tsVal = nt::GetAtomic(dblSub, 0.0);

// read all value changes since the last call to ReadQueue/
↪ReadQueueValues
// ReadQueue() returns timestamps; ReadQueueValues() does not.
std::vector<nt::TimestampedDouble> tsUpdates =
↪nt::ReadQueueDouble(dblSub);
std::vector<double> valUpdates = nt::ReadQueueValuesDouble(dblSub);
}

~Example() {
    // stop subscribing
    nt::Unsubscribe(dblSub);
}

```

C

```

// This code assumes that a NT_Topic dblTopic variable already exists

// start subscribing
// Using a custom type string for types other than raw and string is not
↪recommended.
NT_Subscriber dblSub = NT_Subscribe(dblTopic, NT_DOUBLE, "double", NULL, 0);

// subscribe options may be specified using NT_PubSubOptions
struct NT_PubSubOptions options;
memset(&options, 0, sizeof(options));
options.structSize = sizeof(options);
options.keepDuplicates = 1; // true
options.pollStorage = 10;
NT_Subscriber dblSub = NT_Subscribe(dblTopic, NT_DOUBLE, "double", &options);

// get the most recent value; if no value has been published, returns
// the passed-in default value
double val = NT_GetDouble(dblSub, 0.0);

// get the most recent value, along with its timestamp
struct NT_TimestampedDouble tsVal;
NT_GetAtomic(dblSub, 0.0, &tsVal);
NT_DisposeTimestamped(&tsVal);

// read all value changes since the last call to ReadQueue/ReadQueueValues
// ReadQueue() returns timestamps; ReadQueueValues() does not.
size_t tsUpdatesLen;
struct NT_TimestampedDouble* tsUpdates = NT_ReadQueueDouble(dblSub, &
↪tsUpdatesLen);
NT_FreeQueueDouble(tsUpdates, tsUpdatesLen);

size_t valUpdatesLen;
double* valUpdates = NT_ReadQueueValuesDouble(dblSub, &valUpdatesLen);

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
NT_FreeDoubleArray(valUpdates, valUpdatesLen);

// stop subscribing
NT_Unsubscribe(dblSub);
```

Python

```
class Example:
    def __init__(self, dblTopic: ncore.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, ncore.PubSubOptions(keepDuplicates=True, pollStorage=10)
        )

        # subscribeEx provides the options of using a custom type string.
        # Using a custom type string for types other than raw and string is
        ↪not recommended.
        dblSub = dblTopic.subscribeEx("double", 0.0)

    def periodic(self):
        # simple get of most recent value; if no value has been published,
        # returns the default value passed to the subscribe() function
        val = self.dblSub.get()

        # get the most recent value; if no value has been published, returns
        # the passed-in default value
        val = self.dblSub.get(-1.0)

        # get the most recent value, along with its timestamp
        tsVal = self.dblSub.getAtomic()

        # read all value changes since the last call to readQueue
        # readQueue() returns timestamps
        tsUpdates = self.dblSub.readQueue()

        # often not required in robot code, unless this class doesn't exist for
        # the lifetime of the entire robot program, in which case close() needs
        ↪to be
        # called to stop subscribing
        def close(self):
            # stop subscribing
            self.dblSub.close()
```

28.3.3 Using Entry to Both Subscribe and Publish

An *entry* is a combined publisher and subscriber. The subscriber is always active, but the publisher is not created until a publish operation is performed (e.g. a value is “set”, aka published, on the entry). This may be more convenient than maintaining a separate publisher and subscriber. Similar to publishers and subscribers, NetworkTable entries are represented as type-specific Entry classes (e.g. BooleanEntry: [Java](#), [C++](#), [Python](#)) that must be retained to continue subscribing (and publishing).

Java

```
public class Example {
    // the entry is an instance variable so its lifetime matches that of the
    ↪class
    final DoubleEntry dblEntry;

    public Example(DoubleTopic dblTopic) {
        // start subscribing; the return value must be retained.
        // the parameter is the default value if no value is available when
        ↪get() is called
        dblEntry = dblTopic.getEntry(0.0);

        // publish and subscribe options may be specified using PubSubOption
        dblEntry =
            dblTopic.getEntry(0.0, PubSubOption.keepDuplicates(true),
        ↪PubSubOption.pollStorage(10));

        // getEntryEx provides the options of using a custom type string.
        // Using a custom type string for types other than raw and string is not
        ↪recommended.
        dblEntry = dblTopic.getEntryEx("double", 0.0);
    }

    public void periodic() {
        // entries support all the same methods as subscribers:
        double val = dblEntry.get();
        double val = dblEntry.get(-1.0);
        double val = dblEntry.getAsDouble();
        TimestampedDouble tsVal = dblEntry.getAtomic();
        TimestampedDouble[] tsUpdates = dblEntry.readQueue();
        double[] valUpdates = dblEntry.readQueueValues();

        // entries also support all the same methods as publishers; the first
        ↪time
        // one of these is called, an internal publisher is automatically created
        dblEntry.setDefault(0.0);
        dblEntry.set(1.0);
        dblEntry.set(2.0, 0); // 0 = use current time
        long time = NetworkTablesJNI.now();
        dblEntry.set(3.0, time);
        myFunc(dblEntry);
    }

    public void unublish() {
        // you can stop publishing while keeping the subscriber alive
    }
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    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);
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    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);
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

~Example() {
    // stop publishing and subscribing
    nt::ReleaseEntry(dblEntry);
}

```

C

```

// This code assumes that a NT_Topic dblTopic variable already exists

// start subscribing
// Using a custom type string for types other than raw and string is not
// recommended.
NT_Entry dblEntry = NT_GetEntryEx(dblTopic, NT_DOUBLE, "double", NULL, 0);

// publish and subscribe options may be specified using NT_PubSubOptions
struct NT_PubSubOptions options;
memset(&options, 0, sizeof(options));
options.structSize = sizeof(options);
options.keepDuplicates = 1; // true
options.pollStorage = 10;
NT_Entry dblEntry = NT_GetEntryEx(dblTopic, NT_DOUBLE, "double", &options);

// entries support all the same methods as subscribers:
double val = NT_GetDouble(dblEntry, 0.0);

struct NT_TimestampedDouble tsVal;
NT_GetAtomic(dblEntry, 0.0, &tsVal);
NT_DisposeTimestamped(&tsVal);

size_t tsUpdatesLen;
struct NT_TimestampedDouble* tsUpdates = NT_ReadQueueDouble(dblEntry, &
    tsUpdatesLen);
NT_FreeQueueDouble(tsUpdates, tsUpdatesLen);

size_t valUpdatesLen;
double* valUpdates = NT_ReadQueueValuesDouble(dblEntry, &valUpdatesLen);
NT_FreeDoubleArray(valUpdates, valUpdatesLen);

// entries also support all the same methods as publishers; the first time
// one of these is called, an internal publisher is automatically created
NT_SetDefaultDouble(dblPub, 0.0);
NT_SetDouble(dblPub, 1.0);
NT_SetDouble(dblPub, 2.0, 0); // 0 = use current time
int64_t time = NT_Now();
NT_SetDouble(dblPub, 3.0, time);

// you can stop publishing while keeping the subscriber alive
// it's not necessary to call this before NT_ReleaseEntry()
NT_Unpublish(dblEntry);

// stop subscribing
NT_ReleaseEntry(dblEntry);

```


Python

```

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);
    }
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

// 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");

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

// 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

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

# 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.

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    ↪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();
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    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(

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

        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.

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. **unsubscribe/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 RAI
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_Instance inst = nt::GetDefaultInstance();
nt::StartServer(inst, "networktables.json", "", NT_DEFAULT_PORT3, NT_DEFAULT_
↪PORT4);
```

C

```
NT_Instance 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
        }
    }
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

        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);
        }
    }
}

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

    }

    // 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.

Uyarı: 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");
    }
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

// 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"

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

        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)

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

# 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() {
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

// 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()

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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.

Temel bir istemci programı aşağıdaki örneğe benzer.

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);

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

        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_Instance 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”.

Then this instance is started as a NetworkTables client with the team number (the roboRIO is always the server). Additionally, if the program is running on the Driver Station computer, by using the startDSClient() method, NetworkTables will get the robot IP address from the Driver Station.

Daha sonra bu örnek program, saniyede bir döngü yapar ve x ve y değerlerini alır ve bunları konsolda yazdırır. Daha gerçekçi bir programda, istemci robotun tüketmesi için değerleri işliyor veya üretiyor olabilir.

28.8.1 Gradle kullanarak inşa etme

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'
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

11 }
12
13 wpilibTools.deps.wpilibVersion = wpi.versions.wpilibVersion.get()
14
15 def nativeConfigName = 'wpilibNatives'
16 def nativeConfig = configurations.create(nativeConfigName)
17
18 def nativeTasks = wpilibTools.createExtractionTasks {
19     configurationName = nativeConfigName
20 }
21
22 nativeTasks.addToSourceSetResources(sourceSets.main)
23 nativeConfig.dependencies.add wpilibTools.deps.wpilib("wpimath")
24 nativeConfig.dependencies.add wpilibTools.deps.wpilib("wpinet")
25 nativeConfig.dependencies.add wpilibTools.deps.wpilib("wpiutil")
26 nativeConfig.dependencies.add wpilibTools.deps.wpilib("ntcore")
27 nativeConfig.dependencies.add wpilibTools.deps.wpilib("cscore")
28 nativeConfig.dependencies.add wpilibTools.deps.wpilibOpenCv("frc" + wpi.
    ↪ frcYear.get(), wpi.versions.opencvVersion.get())
29
30 dependencies {
31     implementation wpilibTools.deps.wpilibJava("wpiutil")
32     implementation wpilibTools.deps.wpilibJava("wpimath")
33     implementation wpilibTools.deps.wpilibJava("wpinet")
34     implementation wpilibTools.deps.wpilibJava("ntcore")
35     implementation wpilibTools.deps.wpilibJava("cscore")
36     implementation wpilibTools.deps.wpilibJava("cameraserver")
37     implementation wpilibTools.deps.wpilibOpenCvJava("frc" + wpi.frcYear.
    ↪ get(), wpi.versions.opencvVersion.get())
38
39     implementation group: "com.fasterxml.jackson.core", name: "jackson-
    ↪ annotations", version: wpi.versions.jacksonVersion.get()
40     implementation group: "com.fasterxml.jackson.core", name: "jackson-core",
    ↪ version: wpi.versions.jacksonVersion.get()
41     implementation group: "com.fasterxml.jackson.core", name: "jackson-
    ↪ databind", version: wpi.versions.jacksonVersion.get()
42
43     implementation group: "org.ejml", name: "ejml-simple", version: wpi.
    ↪ versions.ejmlVersion.get()
44     implementation group: "us.hebi.quickbuf", name: "quickbuf-runtime",
    ↪ version: wpi.versions.quickbufVersion.get();
45 }
46
47 shadowJar {
48     archiveBaseName = "TestApplication"
49     archiveVersion = ""
50     exclude("module-info.class")
51     archiveClassifier.set(wpilibTools.currentPlatform.platformName)
52 }
53
54 wrapper {
55     gradleVersion = '8.5'
56 }

```

C++

Uncomment the appropriate platform as highlighted.

```

1  plugins {
2      id "cpp"
3      id "edu.wpi.first.GradleRIO" version "2024.2.1"
4  }
5
6  // Disable local cache, as it won't have the cross artifact necessary
7  wpi.maven.useLocal = false
8
9  // Set to true to run simulation in debug mode
10 wpi.cpp.debugSimulation = false
11
12 def appName = "TestApplication"
13
14 nativeUtils.withCrossLinuxArm64()
15 //nativeUtils.withCrossLinuxArm32() // Uncomment to build for arm32.
16 //targetPlatform below also needs to be fixed
17
18 model {
19     components {
20         "${appName}"(NativeExecutableSpec) {
21             //targetPlatform wpi.platforms.desktop // Uncomment to build on
22             //whatever the native platform currently is
23             targetPlatform wpi.platforms.linuxarm64
24             //targetPlatform wpi.platforms.linuxarm32 // Uncomment to build
25             //for arm32
26
27             sources.cpp {
28                 source {
29                     srcDir 'src/main/cpp'
30                     include '**/*.cpp', '**/*.cc'
31                 }
32                 exportedHeaders {
33                     srcDir 'src/main/include'
34                 }
35             }
36
37             // Enable run tasks for this component
38             wpi.cpp.enableExternalTasks(it)
39
40             wpi.cpp.deps.wpilibStatic(it)
41         }
42     }
43 }
44
45 wrapper {
46     gradleVersion = '8.5'
47 }

```


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();
    }
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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

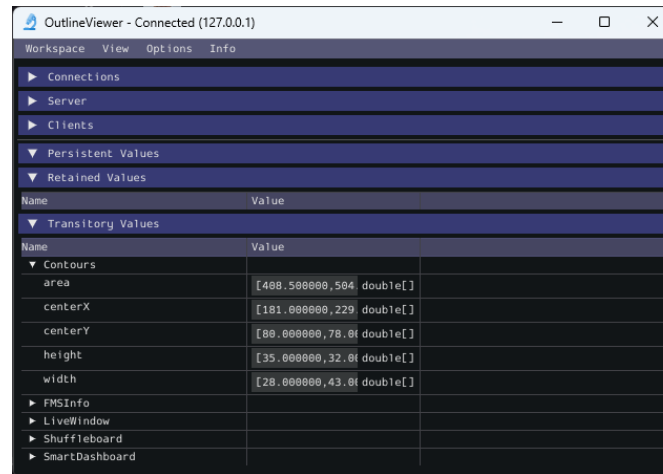
C++ values are now returned/used as value objects (plain `nt::Value`) instead of shared pointers to them (`std::shared_ptr<nt::Value>`).

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.

Çoğunlukla değerler, hedefler veya oyun parçaları gibi bir veya daha fazla ilgi alanı içindir ve birden çok örnek döndürülür. Aşağıdaki örnekte, görüntü işlemcisi tarafından birkaç x, y, genişlik, yükseklik ve alan döndürülür ve robot programı, daha sonraki işlemlerle döndürülen değerlerden hangisinin ilgi çekici olduğunu belirleyebilir.

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.

Aşağıdaki örneklerin her ikisi de, sadece NetworkTables'ın kullanımını gösteren, son derece basitleştirilmiş programlardır. Tüm kod robotInit() yöntemindedir, bu nedenle yalnızca program başladığında çalıştırılır. Programlarınızda, otonom veya teleop dönemlerinde robotu bir komutta veya bir kontrol döngüsünde hangi yöne hedefleyeceğinizi değerlendiren koddaki değerleri büyük olasılıkla alırsınız.

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) {
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
    System.out.print(area + " ");
}

System.out.println();
}
```

C++

```
nt::DoubleArraySubscriber areasSub;

void Robot::RobotInit() override {
    auto table = nt::NetworkTableInstance::GetDefault().GetTable("GRIP/myContoursReport");
    areasSub = table->GetDoubleArrayTopic("area").Subscribe({});
}

void Robot::TeleopPeriodic() override {
    std::cout << "Areas: ";

    std::vector<double> arr = areasSub.Get();

    for (double val : arr) {
        std::cout << val << " ";
    }

    std::cout << std::endl;
}
```

PYTHON

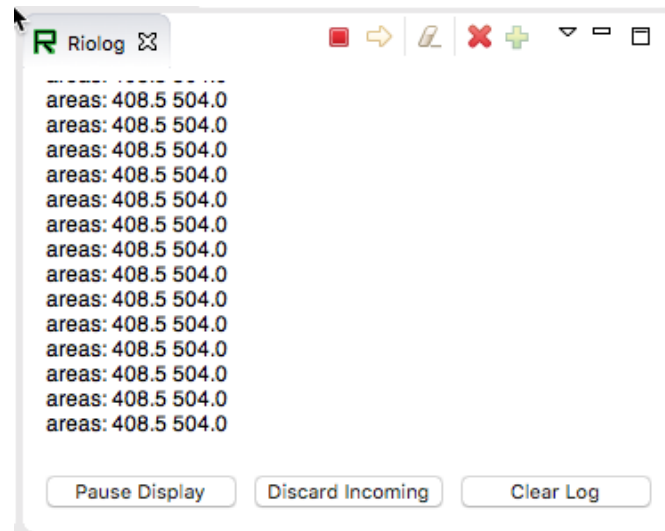
```
def robotInit(self):
    table = ntcore.NetworkTableInstance.getDefault().getTable("GRIP/myContoursReport")
    self.areasSub = table.getDoubleArrayTopic("area").subscribe([])

def teleopPeriodic(self):
    areas = self.areasSub.get()
    print("Areas:", areas)
```

Değerleri alma ve bu programda yazdırmanın adımları şunlardır:

1. Değerlere sahip alt tablo örneğini tutacak tablo değişkenini bildirin.
2. Daha sonra değerleri almak için kullanılabilmesi için alt tablo örneğini başlatın.
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



Bu durumda program yalnızca alan dizisine bakmaktadır, ancak gerçek bir örnekte tüm değerler büyük olasılıkla kullanılacaktır. Riolog'u VS Code veya Driver Station günlüğünde kullanarak, değerleri alınırken görebilirsiniz. Bu program, alanların değişmemesi için örnek bir statik görüntü kullanıyor, ancak robotunuzdaki bir kamera ile değerlerin sürekli değişeceğini hayal edebilirsiniz.

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 Rota Eğitimi

Bu, yörünge oluşturmayı uygulamak ve diferansiyel tahrikli bir robotu takip etmek için tam bir eğitimidir. Bu öğreticide kullanılan tam kod, RamseteCommand örnek projesinde bulunabilir. (Java, C++).

Trajectory Tutorial Overview

Not: 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.

Not: 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

Not: For detailed instructions on using the System Identification tool, see its [dedicated documentation](#).

Not: 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.

Not: 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:

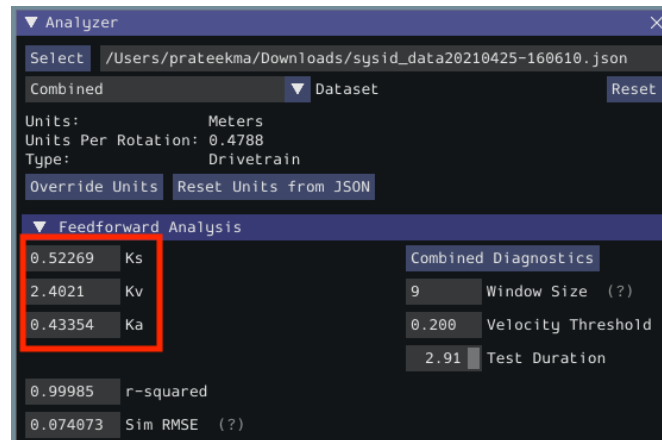


As our data look reasonably linear, and the fit metrics are within acceptable parameters, we proceed to the next step.

Record Feedforward Gains

Not: 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:

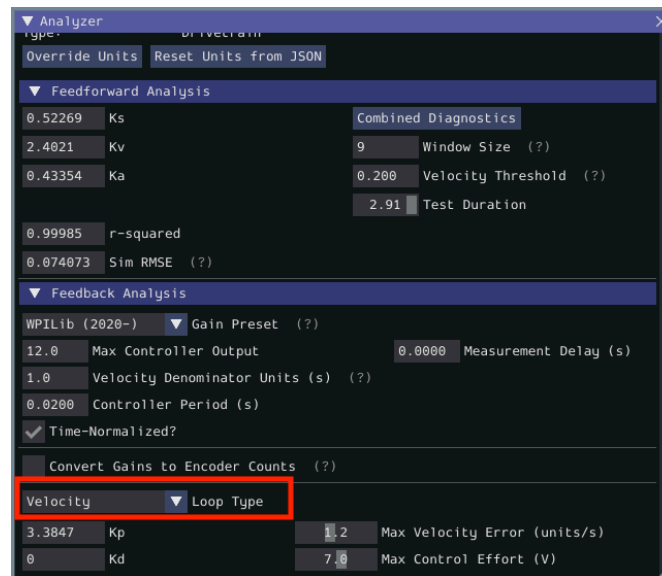
- kS: Volts
- kV: Volts * Seconds / Meters
- kA: Volts * Seconds² / Meters

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.

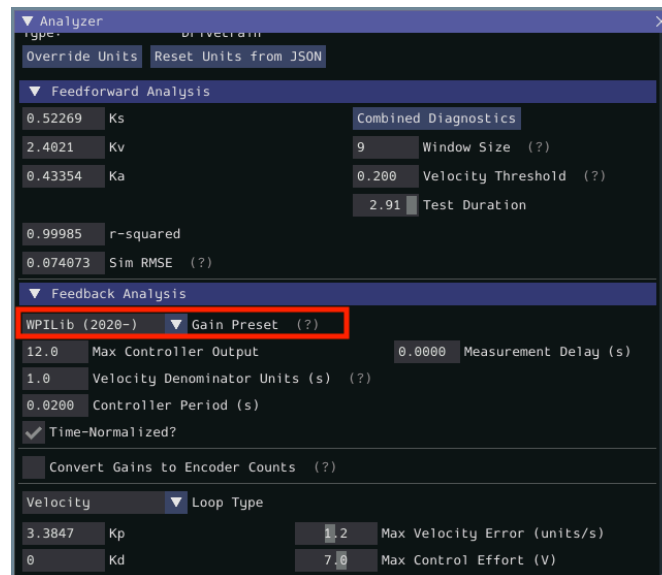
Calculate Feedback Gains

Not: Feedback gains do *not*, in general, transfer across robots. Do *not* use the gains from this tutorial for your own robot.

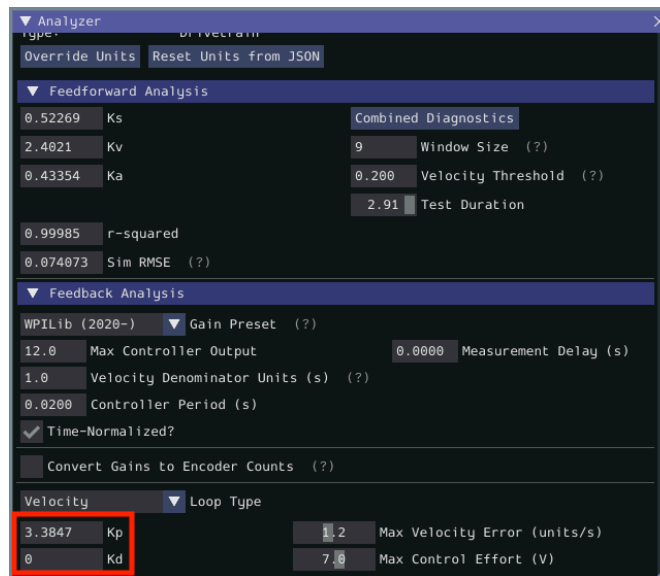
We now *calculate the feedback gains* for the PID control that we will use to follow the path. Trajectory following with WPILib’s RAMSETTE controller uses velocity closed-loop control, so we first select Velocity mode in the identification tool:



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

Not: In C++, it is important that the feedforward constants be entered as the correct unit type. For more information on C++ units, see *C++ Ünite Kitaplığı*.

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.

Not: Feedforward and feedback gains do *not*, in general, transfer across robots. Do *not* use the gains from this tutorial for your own robot.

JAVA

```
39 // These are example values only - DO NOT USE THESE FOR YOUR OWN ROBOT!
40 // These characterization values MUST be determined either experimentally or
↳ theoretically
41 // for *your* robot's drive.
42 // The Robot Characterization Toolsuite provides a convenient tool for obtaining
↳ these
43 // values for your robot.
44 public static final double ksVolts = 0.22;
45 public static final double kvVoltSecondsPerMeter = 1.98;
46 public static final double kaVoltSecondsSquaredPerMeter = 0.2;
47
48 // Example value only - as above, this must be tuned for your drive!
49 public static final double kPDriveVel = 8.5;
```

C++

```
47 // These are example values only - DO NOT USE THESE FOR YOUR OWN ROBOT!
48 // These characterization values MUST be determined either experimentally or
49 // theoretically for *your* robot's drive. The Robot Characterization
50 // Toolsuite provides a convenient tool for obtaining these values for your
51 // robot.
52 inline constexpr auto ks = 0.22_V;
53 inline constexpr auto kv = 1.98 * 1_V * 1_s / 1_m;
54 inline constexpr auto ka = 0.2 * 1_V * 1_s * 1_s / 1_m;
55
56 // Example value only - as above, this must be tuned for your drive!
57 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

```
29 public static final double kTrackwidthMeters = 0.69;
30 public static final DifferentialDriveKinematics kDriveKinematics =
31     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.

Uyarı: 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 [Ramsete Denetleyici Nesnesinin-Object Oluşturulması](#).

JAVA

```

60 // Reasonable baseline values for a RAMSETE follower in units of meters and
↪seconds
61 public static final double kRamseteB = 2;
62 public static final double kRamseteZeta = 0.7;

```

C++

```

64 // Reasonable baseline values for a RAMSETE follower in units of meters and
65 // seconds
66 inline constexpr auto kRamseteB = 2.0 * 1_rad * 1_rad / (1_m * 1_m);
67 inline constexpr auto kRamseteZeta = 0.7 / 1_rad;

```

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 *subclass*.

The full drive class from the RamseteCommand Example Project (Java, C++) can be seen below. The rest of the article will describe the steps involved in writing this class.

Java

```

5  package edu.wpi.first.wpilibj.examples.ramsetecommand.subsystems;
6
7  import edu.wpi.first.math.geometry.Pose2d;
8  import edu.wpi.first.math.kinematics.DifferentialDriveOdometry;
9  import edu.wpi.first.math.kinematics.DifferentialDriveWheelSpeeds;
10 import edu.wpi.first.util.sendable.SendableRegistry;
11 import edu.wpi.first.wpilibj.ADXRS450_Gyro;
12 import edu.wpi.first.wpilibj.Encoder;
13 import edu.wpi.first.wpilibj.drive.DifferentialDrive;
14 import edu.wpi.first.wpilibj.examples.ramsetecommand.Constants.DriveConstants;
15 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
16 import edu.wpi.first.wpilibj2.command.SubsystemBase;
17
18 public class DriveSubsystem extends SubsystemBase {
19     // The motors on the left side of the drive.
20     private final PWMSparkMax m_leftLeader = new PWMSparkMax(DriveConstants.
21 ↪ kLeftMotor1Port);
22     private final PWMSparkMax m_leftFollower = new PWMSparkMax(DriveConstants.
23 ↪ kLeftMotor2Port);
24
25     // The motors on the right side of the drive.
26     private final PWMSparkMax m_rightLeader = new PWMSparkMax(DriveConstants.
27 ↪ kRightMotor1Port);
28     private final PWMSparkMax m_rightFollower = new PWMSparkMax(DriveConstants.
29 ↪ kRightMotor2Port);
30
31     // The robot's drive
32     private final DifferentialDrive m_drive =
33         new DifferentialDrive(m_leftLeader::set, m_rightLeader::set);
34
35     // The left-side drive encoder
36     private final Encoder m_leftEncoder =
37         new Encoder(
38             DriveConstants.kLeftEncoderPorts[0],
39             DriveConstants.kLeftEncoderPorts[1],
40             DriveConstants.kLeftEncoderReversed);

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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 /**

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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.
97         ↪ getRate());
98     }
99
100    /**
101     * Resets the odometry to the specified pose.
102     *
103     * @param pose The pose to which to set the odometry.
104     */
105    public void resetOdometry(Pose2d pose) {
106        m_odometry.resetPosition(
107        ↪ m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
108        ↪ getDistance(), pose);
109    }
110
111    /**
112     * Drives the robot using arcade controls.
113     *
114     * @param fwd the commanded forward movement
115     * @param rot the commanded rotation
116     */
117    public void arcadeDrive(double fwd, double rot) {
118        m_drive.arcadeDrive(fwd, rot);
119    }
120
121    /**
122     * Controls the left and right sides of the drive directly with voltages.
123     *
124     * @param leftVolts the commanded left output
125     * @param rightVolts the commanded right output
126     */
127    public void tankDriveVolts(double leftVolts, double rightVolts) {
128        m_leftLeader.setVoltage(leftVolts);
129        m_rightLeader.setVoltage(rightVolts);
130        m_drive.feed();
131    }
132
133    /** Resets the drive encoders to currently read a position of 0. */
134    public void resetEncoders() {
135        m_leftEncoder.reset();
136        m_rightEncoder.reset();
137    }
138
139    /**
140     * Gets the average distance of the two encoders.
141     *
142     * @return the average of the two encoder readings
143     */
144    public double getAverageEncoderDistance() {
145        return (m_leftEncoder.getDistance() + m_rightEncoder.getDistance()) / 2.0;
146    }

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

145
146 /**
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
166  * slowly.
167  *
168  * @param maxOutput the maximum output to which the drive will be constrained
169  */
170 public void setMaxOutput(double maxOutput) {
171     m_drive.setMaxOutput(maxOutput);
172 }
173
174 /** Zeroes the heading of the robot. */
175 public void zeroHeading() {
176     m_gyro.reset();
177 }
178
179 /**
180  * Returns the heading of the robot.
181  *
182  * @return the robot's heading in degrees, from -180 to 180
183  */
184 public double getHeading() {
185     return m_gyro.getRotation2d().getDegrees();
186 }
187
188 /**
189  * Returns the turn rate of the robot.
190  *
191  * @return The turn rate of the robot, in degrees per second
192  */
193 public double getTurnRate() {
194     return -m_gyro.getRate();
195 }

```

C++ (Header)

```
5 #pragma once
6
7 #include <frc/ADXRS450_Gyro.h>
8 #include <frc/Encoder.h>
9 #include <frc/drive/DifferentialDrive.h>
10 #include <frc/geometry/Pose2d.h>
11 #include <frc/kinematics/DifferentialDriveOdometry.h>
12 #include <frc/motorcontrol/PWMSparkMax.h>
13 #include <frc2/command/SubsystemBase.h>
14 #include <units/voltage.h>
15
16 #include "Constants.h"
17
18 class DriveSubsystem : public frc2::SubsystemBase {
19 public:
20     DriveSubsystem();
21
22     /**
23      * Will be called periodically whenever the CommandScheduler runs.
24      */
25     void Periodic() override;
26
27     // Subsystem methods go here.
28
29     /**
30      * Drives the robot using arcade controls.
31      *
32      * @param fwd the commanded forward movement
33      * @param rot the commanded rotation
34      */
35     void ArcadeDrive(double fwd, double rot);
36
37     /**
38      * Controls each side of the drive directly with a voltage.
39      *
40      * @param left the commanded left output
41      * @param right the commanded right output
42      */
43     void TankDriveVolts(units::volt_t left, units::volt_t right);
44
45     /**
46      * Resets the drive encoders to currently read a position of 0.
47      */
48     void ResetEncoders();
49
50     /**
51      * Gets the average distance of the TWO encoders.
52      *
53      * @return the average of the TWO encoder readings
54      */
55     double GetAverageEncoderDistance();
56
57     /**
58      * Gets the left drive encoder.
59      */
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

60  * @return the left drive encoder
61  */
62  frc::Encoder& GetLeftEncoder();
63
64  /**
65   * Gets the right drive encoder.
66   *
67   * @return the right drive encoder
68   */
69  frc::Encoder& GetRightEncoder();
70
71  /**
72   * Sets the max output of the drive. Useful for scaling the drive to drive
73   * more slowly.
74   *
75   * @param maxOutput the maximum output to which the drive will be constrained
76   */
77  void SetMaxOutput(double maxOutput);
78
79  /**
80   * Returns the heading of the robot.
81   *
82   * @return the robot's heading in degrees, from -180 to 180
83   */
84  units::degree_t GetHeading() const;
85
86  /**
87   * Returns the turn rate of the robot.
88   *
89   * @return The turn rate of the robot, in degrees per second
90   */
91  double GetTurnRate();
92
93  /**
94   * Returns the currently-estimated pose of the robot.
95   *
96   * @return The pose.
97   */
98  frc::Pose2d GetPose();
99
100  /**
101   * Returns the current wheel speeds of the robot.
102   *
103   * @return The current wheel speeds.
104   */
105  frc::DifferentialDriveWheelSpeeds GetWheelSpeeds();
106
107  /**
108   * Resets the odometry to the specified pose.
109   *
110   * @param pose The pose to which to set the odometry.
111   */
112  void ResetOdometry(frc::Pose2d pose);
113
114  private:
115  // Components (e.g. motor controllers and sensors) should generally be

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

116 // declared private and exposed only through public methods.
117
118 // The motor controllers
119 frc::PWMSparkMax m_left1;
120 frc::PWMSparkMax m_left2;
121 frc::PWMSparkMax m_right1;
122 frc::PWMSparkMax m_right2;
123
124 // The robot's drive
125 frc::DifferentialDrive m_drive{[&](double output) { m_left1.Set(output); },
126                               [&](double output) { m_right1.Set(output); }};
127
128 // The left-side drive encoder
129 frc::Encoder m_leftEncoder;
130
131 // The right-side drive encoder
132 frc::Encoder m_rightEncoder;
133
134 // The gyro sensor
135 frc::ADXRS450_Gyro m_gyro;
136
137 // Odometry class for tracking robot pose
138 frc::DifferentialDriveOdometry m_odometry;
139 };

```

C++ (Source)

```

5  #include "subsystems/DriveSubsystem.h"
6
7  #include <frc/geometry/Rotation2d.h>
8  #include <frc/kinematics/DifferentialDriveWheelSpeeds.h>
9
10 using namespace DriveConstants;
11
12 DriveSubsystem::DriveSubsystem()
13 : m_left1{kLeftMotor1Port},
14   m_left2{kLeftMotor2Port},
15   m_right1{kRightMotor1Port},
16   m_right2{kRightMotor2Port},
17   m_leftEncoder{kLeftEncoderPorts[0], kLeftEncoderPorts[1]},
18   m_rightEncoder{kRightEncoderPorts[0], kRightEncoderPorts[1]},
19   m_odometry{m_gyro.GetRotation2d(), units::meter_t{0}, units::meter_t{0}} {
20   wpi::SendableRegistry::AddChild(&m_drive, &m_left1);
21   wpi::SendableRegistry::AddChild(&m_drive, &m_right1);
22
23   m_left1.AddFollower(m_left2);
24   m_right1.AddFollower(m_right2);
25
26   // We need to invert one side of the drivetrain so that positive voltages
27   // result in both sides moving forward. Depending on how your robot's
28   // gearbox is constructed, you might have to invert the left side instead.
29   m_right1.SetInverted(true);
30
31   // Set the distance per pulse for the encoders

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

32     m_leftEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
33     m_rightEncoder.SetDistancePerPulse(kEncoderDistancePerPulse.value());
34
35     ResetEncoders();
36 }
37
38 void DriveSubsystem::Periodic() {
39     // Implementation of subsystem periodic method goes here.
40     m_odometry.Update(m_gyro.GetRotation2d(),
41                     units::meter_t{m_leftEncoder.GetDistance()},
42                     units::meter_t{m_rightEncoder.GetDistance()});
43 }
44
45 void DriveSubsystem::ArcadeDrive(double fwd, double rot) {
46     m_drive.ArcadeDrive(fwd, rot);
47 }
48
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 }
54
55 void DriveSubsystem::ResetEncoders() {
56     m_leftEncoder.Reset();
57     m_rightEncoder.Reset();
58 }
59
60 double DriveSubsystem::GetAverageEncoderDistance() {
61     return (m_leftEncoder.GetDistance() + m_rightEncoder.GetDistance()) / 2.0;
62 }
63
64 frc::Encoder& DriveSubsystem::GetLeftEncoder() {
65     return m_leftEncoder;
66 }
67
68 frc::Encoder& DriveSubsystem::GetRightEncoder() {
69     return m_rightEncoder;
70 }
71
72 void DriveSubsystem::SetMaxOutput(double maxOutput) {
73     m_drive.SetMaxOutput(maxOutput);
74 }
75
76 units::degree_t DriveSubsystem::GetHeading() const {
77     return m_gyro.GetRotation2d().Degrees();
78 }
79
80 double DriveSubsystem::GetTurnRate() {
81     return -m_gyro.GetRate();
82 }
83
84 frc::Pose2d DriveSubsystem::GetPose() {
85     return m_odometry.GetPose();
86 }
87

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
88 frc::DifferentialDriveWheelSpeeds DriveSubsystem::GetWheelSpeeds() {
89     return {units::meters_per_second_t{m_leftEncoder.GetRate()},
90            units::meters_per_second_t{m_rightEncoder.GetRate()}};
91 }
92
93 void DriveSubsystem::ResetOdometry(frc::Pose2d pose) {
94     m_odometry.ResetPosition(m_gyro.GetRotation2d(),
95                             units::meter_t{m_leftEncoder.GetDistance()},
96                             units::meter_t{m_rightEncoder.GetDistance()}, pose);
97 }
```

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

```
31 // The left-side drive encoder
32 private final Encoder m_leftEncoder =
33     new Encoder(
34         DriveConstants.kLeftEncoderPorts[0],
35         DriveConstants.kLeftEncoderPorts[1],
36         DriveConstants.kLeftEncoderReversed);
37
38 // The right-side drive encoder
39 private final Encoder m_rightEncoder =
40     new Encoder(
41         DriveConstants.kRightEncoderPorts[0],
42         DriveConstants.kRightEncoderPorts[1],
43         DriveConstants.kRightEncoderReversed);
```

C++ (Source)

```
17 m_leftEncoder{kLeftEncoderPorts[0], kLeftEncoderPorts[1]},
18 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:

Önemli: 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.
97     ↪ getRate());
98 }
```

C++ (Source)

```
88 frc::DifferentialDriveWheelSpeeds DriveSubsystem::GetWheelSpeeds() {  
89     return {units::meters_per_second_t{m_leftEncoder.GetRate()},  
90             units::meters_per_second_t{m_rightEncoder.GetRate()}};  
91 }
```

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

```
45 // The gyro sensor  
46 private final ADXRS450_Gyro m_gyro = new ADXRS450_Gyro();
```

C++ (Header)

```
134 // The gyro sensor  
135 frc::ADXRS450_Gyro m_gyro;
```

Gyroscope Accessor Method

To access the current heading measured by the gyroscope, we include the following method:

Java

```
178 /**  
179  * Returns the heading of the robot.  
180  *  
181  * @return the robot's heading in degrees, from -180 to 180  
182  */  
183 public double getHeading() {  
184     return m_gyro.getRotation2d().getDegrees();  
185 }
```

C++ (Source)

```

76 units::degree_t DriveSubsystem::GetHeading() const {
77     return m_gyro.GetRotation2d().Degrees();
78 }

```

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

```
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.
79         ↪ getDistance());
80 }
```

C++ (Source)

```
38 void DriveSubsystem::Periodic() {
39     // Implementation of subsystem periodic method goes here.
40     m_odometry.Update(m_gyro.GetRotation2d(),
41                     units::meter_t{m_leftEncoder.GetDistance()},
42                     units::meter_t{m_rightEncoder.GetDistance()});
43 }
```

Odometry Accessor Method

To access the robot's current computed pose, we include the following method:

Java

```
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 }
```

C++ (Source)

```
84 frc::Pose2d DriveSubsystem::GetPose() {
85     return m_odometry.GetPose();
86 }
```

Önemli: 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.

Uyarı: `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.
117     ↪ kRamseteZeta),

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

117     new SimpleMotorFeedforward(
118         DriveConstants.kSVolts,
119         DriveConstants.kVVoltsSecondsPerMeter,
120         DriveConstants.kAVoltsSecondsSquaredPerMeter),
121     DriveConstants.kDriveKinematics,
122     m_robotDrive::getWheelSpeeds,
123     new PIDController(DriveConstants.kPDriveVel, 0, 0),
124     new PIDController(DriveConstants.kPDriveVel, 0, 0),
125     // RamseteCommand passes volts to the callback
126     m_robotDrive::tankDriveVolts,
127     m_robotDrive);
128
129     // Reset odometry to the initial pose of the trajectory, run path following
130     // command, then stop at the end.
131     return Commands.runOnce(() -> m_robotDrive.resetOdometry(exampleTrajectory.
132         ↪getInitialPose()))
133         .andThen(ramseteCommand)
134         .andThen(Commands.runOnce(() -> m_robotDrive.tankDriveVolts(0, 0)));
135 }

```

C++ (Source)

```

45 frc2::CommandPtr RobotContainer::GetAutonomousCommand() {
46     // Create a voltage constraint to ensure we don't accelerate too fast
47     frc::DifferentialDriveVoltageConstraint autoVoltageConstraint{
48         frc::SimpleMotorFeedforward<units::meters>{
49             DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
50         DriveConstants::kDriveKinematics, 10_V};
51
52     // Set up config for trajectory
53     frc::TrajectoryConfig config{AutoConstants::kMaxSpeed,
54                                 AutoConstants::kMaxAcceleration};
55     // Add kinematics to ensure max speed is actually obeyed
56     config.SetKinematics(DriveConstants::kDriveKinematics);
57     // Apply the voltage constraint
58     config.AddConstraint(autoVoltageConstraint);
59
60     // An example trajectory to follow. All units in meters.
61     auto exampleTrajectory = frc::TrajectoryGenerator::GenerateTrajectory(
62         // Start at the origin facing the +X direction
63         frc::Pose2d{0_m, 0_m, 0_deg},
64         // Pass through these two interior waypoints, making an 's' curve path
65         {frc::Translation2d{1_m, 1_m}, frc::Translation2d{2_m, -1_m}},
66         // End 3 meters straight ahead of where we started, facing forward
67         frc::Pose2d{3_m, 0_m, 0_deg},
68         // Pass the config
69         config);
70
71     frc2::CommandPtr ramseteCommand{frc2::RamseteCommand(
72         exampleTrajectory, [this] { return m_drive.GetPose(); },
73         frc::RamseteController{AutoConstants::kRamseteB,
74                                 AutoConstants::kRamseteZeta},
75         frc::SimpleMotorFeedforward<units::meters>{

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

76         DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
77         DriveConstants::kDriveKinematics,
78         [this] { return m_drive.GetWheelSpeeds(); },
79         frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
80         frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
81         [this](auto left, auto right) { m_drive.TankDriveVolts(left, right); },
82         [&m_drive]);
83
84     // Reset odometry to the initial pose of the trajectory, run path following
85     // command, then stop at the end.
86     return frc2::cmd::RunOnce(
87         [this, initialPose = exampleTrajectory.InitialPose()] {
88             m_drive.ResetOdometry(initialPose);
89         },
90         {})
91     .AndThen(std::move(ramseteCommand))
92     .AndThen(
93         frc2::cmd::RunOnce([this] { m_drive.TankDriveVolts(0_V, 0_V); }, {}));
94 }

```

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

```

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);

```

C++ (Source)

```

46 // Create a voltage constraint to ensure we don't accelerate too fast
47 frc::DifferentialDriveVoltageConstraint autoVoltageConstraint{
48     frc::SimpleMotorFeedforward<units::meters>{
49         DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
50     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

```

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);

```

C++ (Source)

```

52 // Set up config for trajectory
53 frc::TrajectoryConfig config{AutoConstants::kMaxSpeed,
54                             AutoConstants::kMaxAcceleration};
55 // Add kinematics to ensure max speed is actually obeyed
56 config.SetKinematics(DriveConstants::kDriveKinematics);
57 // Apply the voltage constraint
58 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

```
100 // An example trajectory to follow. All units in meters.
101 Trajectory exampleTrajectory =
102     TrajectoryGenerator.generateTrajectory(
103         // Start at the origin facing the +X direction
104         new Pose2d(0, 0, new Rotation2d(0)),
105         // Pass through these two interior waypoints, making an 's' curve path
106         List.of(new Translation2d(1, 1), new Translation2d(2, -1)),
107         // End 3 meters straight ahead of where we started, facing forward
108         new Pose2d(3, 0, new Rotation2d(0)),
109         // Pass config
110         config);
```

C++ (Source)

```
60 // An example trajectory to follow. All units in meters.
61 auto exampleTrajectory = frc::TrajectoryGenerator::GenerateTrajectory(
62     // Start at the origin facing the +X direction
63     frc::Pose2d{0_m, 0_m, 0_deg},
64     // Pass through these two interior waypoints, making an 's' curve path
65     {frc::Translation2d{1_m, 1_m}, frc::Translation2d{2_m, -1_m}},
66     // End 3 meters straight ahead of where we started, facing forward
67     frc::Pose2d{3_m, 0_m, 0_deg},
68     // Pass the config
69     config);
```

Not: 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

```
129 // Reset odometry to the initial pose of the trajectory, run path following
130 // command, then stop at the end.
131 return Commands.runOnce(() -> m_robotDrive.resetOdometry(exampleTrajectory.
    ↪ getInitialPose()))
```

C++ (Source)

```

84 // Reset odometry to the initial pose of the trajectory, run path following
85 // command, then stop at the end.
86 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 [Yörüngeleri Dönüştürmek](#).

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

```

112 RamseteCommand ramseteCommand =
113     new RamseteCommand(
114         exampleTrajectory,
115         m_robotDrive::getPose,
116         new RamseteController(AutoConstants.kRamseteB, AutoConstants.
117             ↪kRamseteZeta),
118         new SimpleMotorFeedforward(
119             DriveConstants.ksVolts,
120             DriveConstants.kvVoltSecondsPerMeter,
121             DriveConstants.kaVoltSecondsSquaredPerMeter),
122         DriveConstants.kDriveKinematics,
123         m_robotDrive::getWheelSpeeds,
124         new PIDController(DriveConstants.kPDriveVel, 0, 0),
125         new PIDController(DriveConstants.kPDriveVel, 0, 0),
126         // RamseteCommand passes volts to the callback
127         m_robotDrive::tankDriveVolts,
128         m_robotDrive);

```

C++ (Source)

```

71 frc2::CommandPtr ramseteCommand{frc2::RamseteCommand(
72     exampleTrajectory, [this] { return m_drive.GetPose(); },
73     frc::RamseteController{AutoConstants::kRamseteB,
74         AutoConstants::kRamseteZeta},
75     frc::SimpleMotorFeedforward<units::meters>{
76         DriveConstants::ks, DriveConstants::kv, DriveConstants::ka},
77     DriveConstants::kDriveKinematics,
78     [this] { return m_drive.GetWheelSpeeds(); },
79     frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
80     frc::PIDController{DriveConstants::kPDriveVel, 0, 0},
81     [this](auto left, auto right) { m_drive.TankDriveVolts(left, right); },
82     {&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

Not: Users may find a community driven project [PathPlanner](#) as potentially more useful. PathPlanner improves upon traditional pathplanning applications with an intuitive user interface and swerve path following support. Note that WPILib offers no support for community projects.

Introduction to PathWeaver

Not: 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.

Autonomous is an important section of the match; it is exciting when robots do impressive things in autonomous. In order to score, the robot usually need to go somewhere. The faster the robot arrives at that location, the sooner it can score points! The traditional method for autonomous is driving in a straight line, turning to a certain angle, and driving in a straight line again. This approach works fine, but the robot spends a non-negligible amount of time stopping and starting again after each straight line and turn.

A more advanced approach to autonomous is called “path planning”. Instead of driving in a straight line and turning once the line is complete, the robot continuously moves, driving with a curve-like motion. This can reduce turning stoppage time.

WPILib contains a trajectory generation suite that can be used by teams to generate and follow trajectories. This series of articles will go over how to generate and visualize trajectories using PathWeaver. For a comprehensive tutorial on following trajectories, please visit the [end-to-end trajectory tutorial](#).

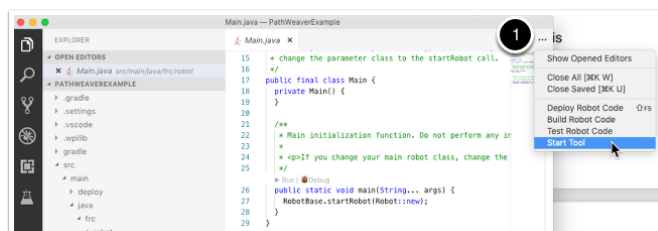
Not: [Trajectory following](#) code is required to use PathWeaver. We recommend that you start with Trajectory following and get that working with simple paths. From there you can continue on to testing more complicated paths generated by PathWeaver.

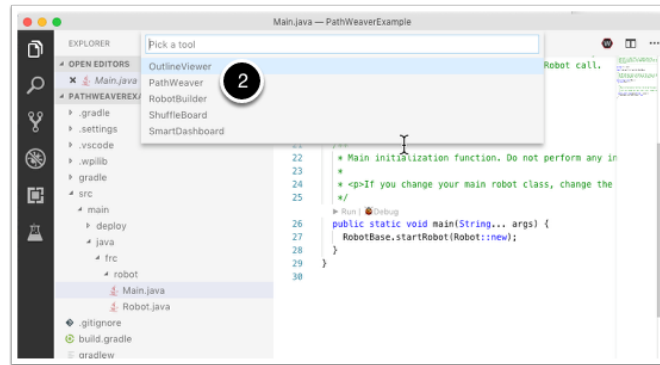
Creating a Pathweaver Project

PathWeaver is the tool used to draw the paths for a robot to follow. The paths for a single program are stored in a PathWeaver project.

Starting PathWeaver

PathWeaver is started by clicking on the ellipsis icon in the top right of the corner of the Visual Studio Code interface. You must select a source file from the WPILib project to see the icon. Then click on “Start tool” and then click on “PathWeaver” as shown below.





Creating the Project

To create a PathWeaver project, click on “Create project” and then fill out the project creation form. Notice that hovering over any of the fields in the form will display more information about what is required.

A screenshot of the 'PathWeaver - 2021.1.2' 'Create Project...' dialog box. The dialog contains several input fields and dropdown menus: 'Project Directory' with a 'Browse' button, 'Output Directory' with a 'Browse' button, 'Game' set to 'Infinite Recharge', 'Length Unit' set to 'Meter', 'Export Unit' set to 'Always Meters', 'Max Velocity' with a unit of 'm/sec', 'Max Acceleration' with a unit of 'm/sec²', and 'Wheel Base' with a unit of 'm'. At the bottom are 'Cancel' and 'Create Project' buttons.

Project Directory: This is normally the top level project directory that contains the build.gradle and src files for your robot program. Choosing this directory is the expected way to use PathWeaver and will cause it to locate all the output files in the correct directories for automatic path deployment to your robot.

Output directory: The directory where the paths are stored for deployment to your robot. If you specified the top level project folder of our robot project in the previous step (as recommended) filling in the output directory is optional.

Game: The game (which FRC® game is being used) will cause the correct field image overlay to be used. You can also create your own field images and the procedure will be described later in this series.

Length Unit: The units to be used in describing your robot and for the field measurements when visualizing trajectories using PathWeaver.

Export Unit: The units to be used when exporting the paths and waypoints. If you are planning to use WPILib Trajectories, then you should choose *Always Meters*. Choosing *Same as*

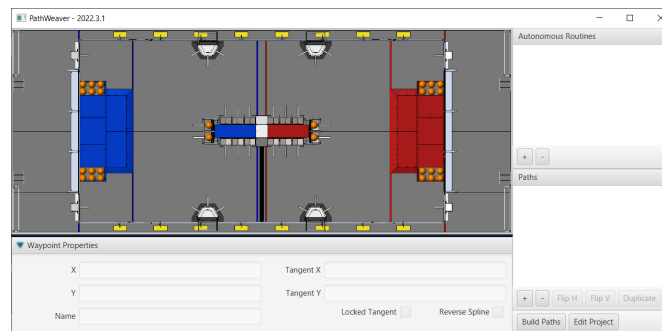
Project will export in the same units as *Length Unit* above.

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/sec²* is a good place to start if you don't know your drivetrain's characteristics.

Wheel Base: The distance between the left and right wheels of your robot. This is used to ensure that no wheel on a differential drive will go over the specified max velocity around turns.

PathWeaver User Interface

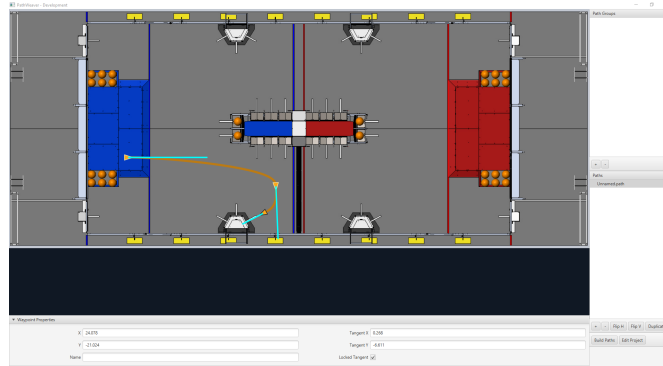


The PathWeaver user interface consists of the following:

1. The field area in the top left corner, which takes up most of the PathWeaver window. Trajectories are drawn on this part of the program.
2. The properties of the currently selected waypoint are displayed in the bottom pane. These properties include the X and Y, along with the tangents at each point.
3. A group of paths (or an “autonomous” mode) is displayed on the upper right side of the window. This is a convenient way of seeing all of the trajectories in a single auto mode.
4. The individual paths that a robot might follow are displayed in the lower right side of the window.
5. The “Build Paths” button will export the trajectories in a JSON format. These JSON files can be used from the robot code to follow the trajectory.
6. The “Edit Project” button allows you to edit the project properties.

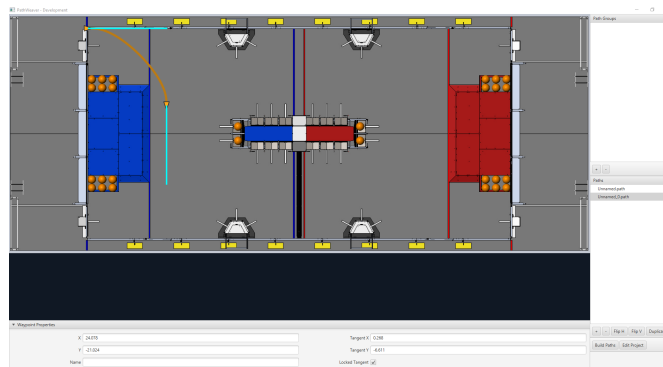
Visualizing PathWeaver Trajectories

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

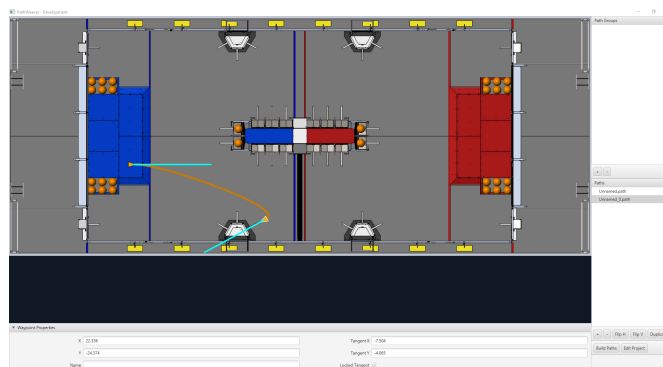
To start creating a trajectory, click the + (plus) button in the path window. A default trajectory will be created that probably does not have the proper start and end points that you desire. The path also shows the tangent vectors (teal lines) for the start and end points. Changing the angle of the 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.

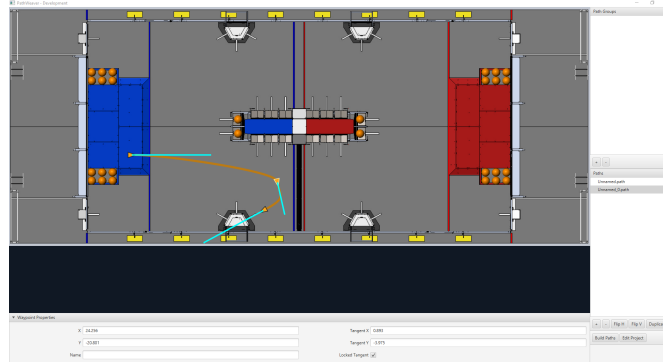
Changing a Waypoint Heading

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.

Importing a PathWeaver JSON

The TrajectoryUtil class can be used to import a PathWeaver JSON into robot code to follow it. This article will go over importing the trajectory. Please visit the [end-to-end trajectory tutorial](#) for more information on following the trajectory.

The fromPathweaverJson (Java) / FromPathweaverJson (C++) static methods in TrajectoryUtil can be used to create a trajectory from a JSON file stored on the roboRIO file system.

Önemli: To be compatible with the Field2d view in the simulator GUI, the coordinates for the exported JSON have changed. Previously (before 2021), the range of the y-coordinate was from -27 feet to 0 feet whereas now, the range of the y-coordinate is from 0 feet to 27 feet (with 0 being at the bottom of the screen and 27 feet being at the top). This should not affect teams who are correctly *resetting their odometry to the starting pose of the trajectory* before path following.

Not: PathWeaver places JSON files in src/main/deploy/paths which will automatically be placed on the roboRIO file system in /home/lvuser/deploy/paths and can be accessed using getDeployDirectory as shown below.

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.
        (sonraki sayfaya devam))
```

(önceki sayfadan devam)

```

    ↪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());
}

```

In the examples above, YourPath should be replaced with the name of your path.

Uyarı: Loading a PathWeaver JSON from file in Java can take more than one loop iteration so it is highly recommended that the robot load these paths on startup.

Adding field images to PathWeaver

Here are instructions for adding your own field image using the 2019 game as an example.

Games are loaded from the ~/PathWeaver/Games on Linux and macOS or %USERPROFILE%/PathWeaver/Games directory on Windows. The files can be in either a game-specific subdirectory, or in a zip file in the Games directory. The ZIP file must follow the same layout as a game directory; the JSON file must be in the root of the ZIP file (cannot be in a subdirectory).

Download the example *FIRST* Destination Deep Space field definition [here](#). Other field definitions are available in the [allwpilib GitHub repository](#).

File Layout

```

~/PathWeaver
  /Games
    /Custom Game
      custom-game.json
      field-image.png
    OtherGame.zip

```

JSON Format

```
{
  "game": "game name",
  "field-image": "relative/path/to/img.png",
  "field-corners": {
    "top-left": [x, y],
    "bottom-right": [x, y]
  },
  "field-size": [width, length],
  "field-unit": "unit name"
}
```

The path to the field image is relative to the JSON file. For simplicity, the image file should be in the same directory as the JSON file.

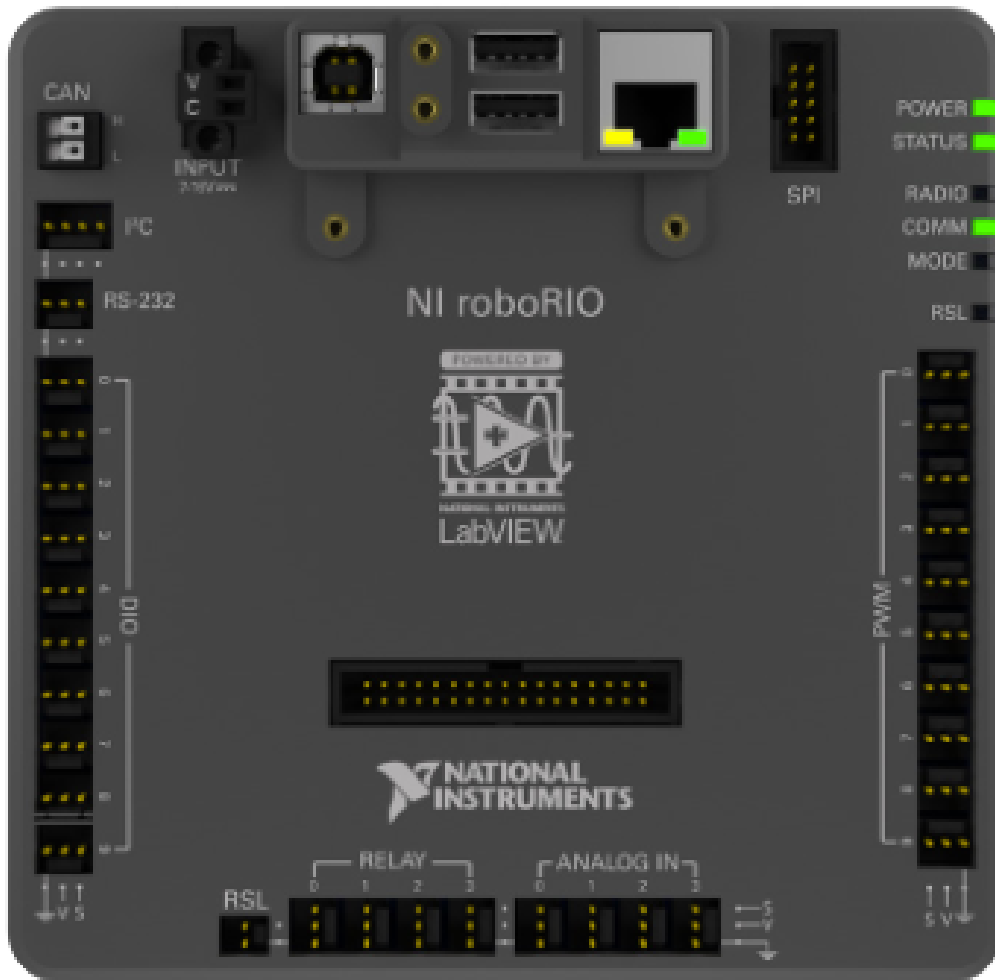
The field corners are the X and Y coordinates of the top-left and bottom-right pixels defining the rectangular boundary of the playable area in the field image. Non-rectangular playing areas are not supported.

The field size is the width and length of the playable area of the field in the provided units.

The field units are case-insensitive and can be in meters, cm, mm, inches, feet, yards, or miles. Singular, plural, and abbreviations are supported (e.g. "meter", "meters", and "m" are all valid for specifying meters)

Not: When making a new field image, a border (minimum of 20 pixels is recommended) should be left around the outside so that waypoints on the field edge are accessible.

30.1 roboRIO Giriş



RoboRIO, özellikle FIRST düşünülerek tasarlanmıştır. RoboRIO, Gerçek Zamanlı işlemciler + FPGA'nın (sahada programlanabilir kapı dizisi) temel mimarisine sahiptir, ancak endüstride

kullanılan bazı benzer sistemlerden daha güçlü, daha hafif ve daha küçüktür.

RoboRIO, entegre devreler (I2C), seri çevresel arabirimler (SPI), RS232, USB, Ethernet, darbe genişlik modülasyonu (PWM) için yerleşik bağlantı noktaları ve ortak sensörleri hızlı bir şekilde bağlamak için röleler içeren yeniden yapılandırılabilir bir robotik denetleyicidir. ve robotikte kullanılan aktüatörler. Denetleyicide LED'ler, düğmeler, yerleşik bir ivmeölçer ve özel bir elektronik bağlantı noktası bulunur. Yerleşik bir çift çekirdekli ARM gerçek zamanlı Cortex - A9 işlemciye ve özelleştirilebilir Xilinx FPGA'ya sahiptir.

Detailed information on the roboRIO can be found in the [roboRIO User Manual](#) and in the [roboRIO technical specifications](#).

Before deploying programs to your roboRIO, you must first image the roboRIO: [roboRIO 1](#) [roboRIO 2](#).

30.2 roboRIO Web Dashboard - Web Kontrol Paneli

RoboRIO web kontrol paneli, roboRIO'nun durumunu kontrol etmek ve ayarları güncellemek için kullanılabilen roboRIO'da yerleşik bir web sayfasıdır.

30.2.1 Webdash'i açma

172.22.11.2: System Configuration

Save

Settings

Hostname: roboRIO-330-FRC

IP Address: 0.0.0.0 (Ethernet)
172.22.11.2 (Ethernet)

DNS Name:

Vendor: National Instruments

Model: roboRIO

Serial Number: 03063C6E

Firmware Version: 8.8.0f0

Operating System: NI Linux Real-Time ARMv7-A 4.14.146-rt67

Status: Running

System Start Time: Thu Jul 01 2021 10:33:34 GMT-0700 (Pacific Daylight Time)

Image Title: roboRIO Image

Image Version: FRC_roboRIO_2024_v2.0

Comments:

Locale: English

Update Firmware

Startup Settings

☐ Force Safe Mode

☐ Enable Console Out

☐ Disable RT Startup App

☐ Disable FPGA Startup App

☒ LabVIEW Project Access

Web kontrol panelini açmak için bir web tarayıcısı açın ve adres çubuğuna roboRIO'nun adresini girin (USB için 172.22.11.2 veya "roboRIO-####-FRC.local" burada #### takım nu-

maranızdır, her iki arayüz için de başında sıfır olmadan). mDNS ve roboRIO ağ oluşturma hakkında daha fazla ayrıntı için bu belgeye bakın: [IP Yapılandırmaları](#)

30.2.2 Sistem Yapılandırması Sekmesi

Web panosunun ana ekranı, 5 ana bölümden oluşan System Configuration-Sistem Yapılandırması sekmesidir:

1. Gezinme Çubuğu - Bu bölüm, web panosunun farklı bölümlerine gitmenize olanak tanır. Bu gezinme çubuğundan erişilebilen farklı sayfalar aşağıda tartışılmıştır.
2. System Settings - Bu bölüm Sistem Ayarları ile ilgili bilgileri içerir. Ana bilgisayar adı alanı manuel olarak değiştirilmemelidir, bunun yerine ana bilgisayar adını takım numaranıza göre ayarlamak için roboRIO Imaging Tool aracını kullanın. Bu bölüm, aygıt IP'si, donanım yazılımı sürümü ve imaj sürümü gibi bilgileri içerir.
3. Başlangıç Ayarları - Bu bölüm roboRIO için Startup - Başlangıç ayarlarını içerir. Bunlar aşağıdaki alt adımda açıklanmaktadır
4. System Resources (resimde gösterilmemiştir) - Bu bölüm, bellek ve CPU yükü gibi sistem kaynaklarının anlık görüntüsünü sağlar.

Startup Settings-Başlangıç Ayarları

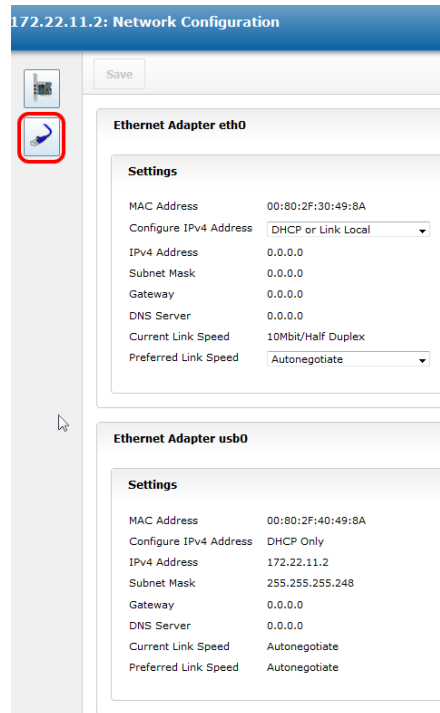


Startup Settings

- ☐ Force Safe Mode
- ☒ Enable Console Out
- ☐ Disable RT Startup App
- ☐ Disable FPGA Startup App
- ☒ Enable Secure Shell Server (sshd)
- ☒ LabVIEW Project Access

- Güvenli Modu Zorla - Denetleyiciyi Safe Mode-Güvenli Mod'a zorlar. Bu, imaj yükleme sorunlarını gidermek için kullanılabilir, ancak bunun yerine cihazı Güvenli Mod'a geçirmek için roboRIO üzerindeki Sıfırla düğmesini kullanmanız önerilir (güç zaten uygulanmış haldeyken rest düğmesini 5 saniye basılı tutun). **Varsayılan olarak işaretli değildir.**
- Enable Console Out - This enables the on-board RS232 port to be used as a Console output. This port uses RS232 levels and should not be connected to many microcontrollers which use TTL levels). **Default is unchecked.**
- Disable RT Startup App - Bu kutuyu işaretlemek, kodun başlangıçta çalışmasını devre dışı bırakır. RoboRIO'nun yeni program indirmeye yanıt vermediğini fark ederseniz, bu sorun giderme için kullanılabilir. Varsayılan işaretli değil
- Disable FPGA Startup App - **Bu kutu işaretlenmemelidir.**
- LabVIEW Project Access - **It is recommended to leave this box checked.** This setting allows LabVIEW projects to access the roboRIO.

30.2.3 Ağ Yapılandırması



172.22.11.2: Network Configuration

Save

Ethernet Adapter eth0

Settings

MAC Address	00:80:2F:30:49:8A
Configure IPv4 Address	DHCP or Link Local
IPv4 Address	0.0.0.0
Subnet Mask	0.0.0.0
Gateway	0.0.0.0
DNS Server	0.0.0.0
Current Link Speed	10Mbit/Half Duplex
Preferred Link Speed	Autonegotiate

Ethernet Adapter usb0

Settings

MAC Address	00:80:2F:40:49:8A
Configure IPv4 Address	DHCP Only
IPv4 Address	172.22.11.2
Subnet Mask	255.255.255.248
Gateway	0.0.0.0
DNS Server	0.0.0.0
Current Link Speed	Autonegotiate
Preferred Link Speed	Autonegotiate

Bu sayfa roboRIO'nun ağ bağdaştırıcılarının yapılandırmasını gösterir. **Bu sayfadaki herhangi bir ayarı değiştirmeniz önerilmez.** roboRIO ağ oluşturma hakkında daha fazla bilgi için şu makaleye bakın: [IP Yapılandırmaları](#)

30.3 roboRIO FTP

Not: RoboRIO'da hem SFTP hem de anonim FTP etkin. Bu makale, roboRIO dosya sistemine erişmek için her birinin nasıl kullanılacağını açıklamaktadır.

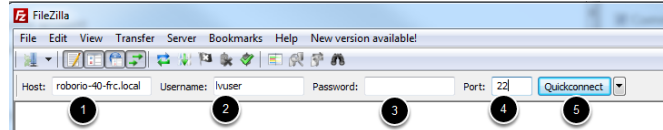
30.3.1 SFTP

SFTP, roboRIO dosya sistemine erişmenin önerilen yoludur. Programınızın altında çalışacağı hesabı kullanacağınız için, üzerine kopyalanan dosyalar her zaman kodunuzla uyumlu izinlere sahip olmalıdır.

Yazılım

SFTP için ücretsiz olarak kullanılabilen bir dizi program vardır. Bu makale FileZilla'nın kullanımını tartışacaktır. Devam etmeden önce [FileZilla](#) indirip kurabilir veya aşağıdaki talimatları SFTP istemcinize aktarabilirsiniz.

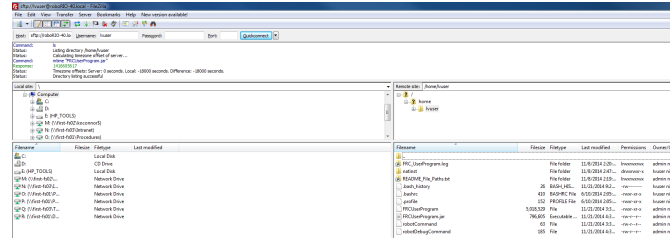
RoboRIO'ya bağlanma



RoboRIO'nuza bağlanmak için:

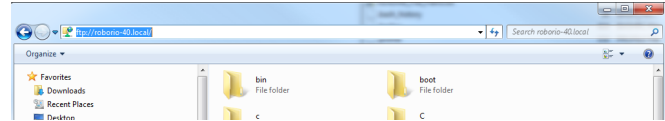
1. "Host" kutusuna mDNS adını (roboRIO-TEAM-frc.local) girin
2. Kullanıcı Adı kutusuna "lvuser" girin (bu, programınızın altında çalıştığı hesaptır)
3. Parola kutusunu boş bırakın
4. Bağlantı noktası kutusuna "22" girin (SFTP varsayılan bağlantı noktası)
5. Quickconnect 'e tıklayın

RoboRIO dosya sistemine göz atma



RoboRIO'ya bağlandıktan sonra Filezilla, \home\lvuser dizinini açacaktır. Sağdaki bölme uzaktaki sistemdir (roboRIO), sol bölme yerel sistemdir (bilgisayarınız). Her bölmenin üst bölümü, göz atmakta olduğunuz geçerli dizinin hiyerarşisini gösterir, alt bölüm dizinin içeriğini gösterir. Dosyaları aktarmak için, bir taraftan diğerine tıklayıp sürüklemeniz yeterlidir. RoboRIO'da dizinler oluşturmak için sağ tıklayın ve "Create Directory" u seçin.

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 Kullanıcı Hesapları ve SSH

Not: Bu belge, tipik FRC ® programlama için gerekli olmayan gelişmiş konuları içerir.

RoboRIO görüntüsü bir dizi hesap içerir, bu makale FRC için kullanılan ikisini vurgulayacak ve bunların amaçları hakkında bazı ayrıntılar sağlayacaktır. Ayrıca roboRIO'ya SSH üzerinden nasıl bağlanılacağı da açıklanacaktır.

30.4.1 roboRIO Kullanıcı Hesapları

RoboRIO görüntüsü bir dizi kullanıcı hesabı içerir, ancak FRC için birincil ilgi alanı iki tanedir.

admin

“Admin-Yönetici” hesabının sisteme kök erişimi vardır ve işletim sistemi dosyalarını veya ayarlarını değiştirmek için kullanılabilir. RoboRIO’nun işletim sistemini bozabilecek ayarların ve dosyaların değiştirilmesine izin verdiği için ekipler bu hesabı kullanırken dikkatli olmalıdır. Bu hesabın kimlik bilgileri:

Kullanıcı adı: admin

Şifre:

Not: Parola kasıtlı olarak boştur.

lvuser

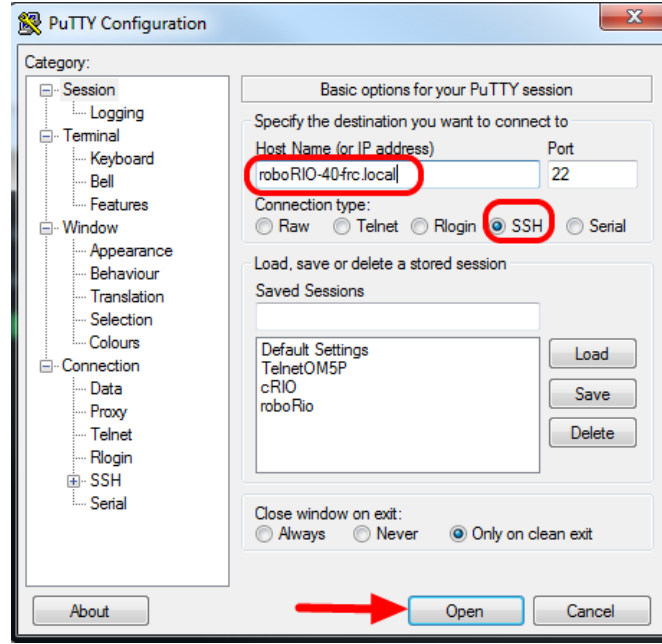
“Lvuser” hesabı, her üç dil için de kullanıcı kodunu çalıştırmak için kullanılan hesaptır. Bu hesabın kimlik bilgileri değiştirilmemelidir. Takımlar, roboRIO ile çalışırken, kodlarının altında çalışacağı aynı hesapta herhangi bir dosya veya ayar değişikliğinin yapılmasını sağlamak için bu hesabı (ssh veya sftp aracılığıyla) kullanmak isteyebilir.

Tehlike: Changing the default ssh passwords for either “lvuser” or “admin” will prevent C++, Java, and Python teams from uploading code.

30.4.2 SSH

SSH (Güvenli Kabuk), güvenli veri iletişimi için kullanılan bir protokoldür. Bir Linux sistemiyle ilgili olarak geniş anlamda bahsedildiğinde (roboRIO üzerinde çalışan sistem gibi), genellikle SSH protokolü kullanılarak komut satırı konsoluna erişim anlamına gelir. Bu, uzak sistemde komutları yürütmek için kullanılabilir. SSH için kullanılacak ücretsiz bir istemci PuTTY’dir: <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

Putty'i aç



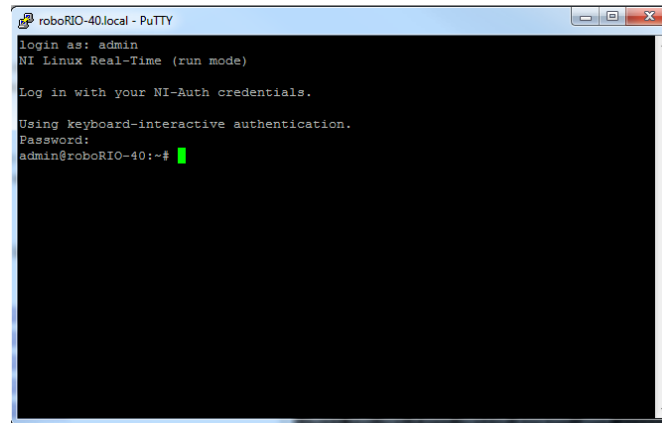
Putty'yi açın (herhangi bir güvenlik isteminde Tamam'a tıklayın). Ardından aşağıdaki ayarları yapın:

1. Host Name: roboRIO-TEAM-frc.local (TEAM takım numaranız olduğunda, örnek takım 40'ı gösterir)
2. Bağlantı Türü: SSH

Diğer ayarlar varsayılan olarak bırakılabilir. Bağlantıyı açmak için Aç'a tıklayın. SSH anahtarlarıyla ilgili bir istem görürseniz Tamam'ı tıklayın.

USB üzerinden bağlıysanız, ana bilgisayar adı olarak 172.22.11.2'yi kullanabilirsiniz. RoboRIO'nuz statik bir IP'ye ayarlanmışsa, Ethernet / kablosuz üzerinden bağlıysa bu IP'yi ana bilgisayar adı olarak kullanabilirsiniz.

Log In-Oturum aç



İstemi gördüğünüzde, istenen kullanıcı adını girin (açıklama için yukarıya bakın) ve ardından enter tuşuna basın. Parola isteminde enter tuşuna basın (her iki hesabın parolası boştur).

30.5 RoboRIO Kesintisi ve Mevcut Çekişi Anlama

Yüksek akım çekme olayları sırasında kendisini ve radyo gibi diğer kontrol sistemi bileşenlerini korumak için ve akü voltajının korunmasına yardımcı olmak için roboRIO, aşamalı bir kesinti koruma şeması içerir. Bu makale, bu şemayı açıklar, sistem akım çekmesi için proaktif olarak planlama hakkında bilgi sağlar ve robotunuzda meydana gelirse elektrik kesintisi olaylarını anlamak için PDP'nin yeni işlevselliğinin ve DS Günlük Dosyası Görüntüleyicisinin nasıl kullanılacağını açıklar.

30.5.1 roboRIO Kesinti Koruması

RoboRIO, batarya voltajını tehlikeli derecede düşük çeken büyük akım durumunda cihaz sıfırlamalarını önlemek ve giriş voltajını kendisine ve diğer kontrol sistemi bileşenlerine korumaya çalışmak için aşamalı bir kesinti koruma şeması kullanır.

Aşama 1 - 6V çıkış düşüşü

** Voltage Trigger - Gerilim Tetikleyici - 6,8 V **

When the voltage drops below 6.8V, the 6V output on the *PWM* pins will start to drop.

Aşama 2 - Çıkışı Devre Dışı Bırakma

** Voltage Trigger - Gerilim Tetikleyici - 6,3V **

Voltaj 6,3V'un altına düştüğünde, kontrol cihazı voltaj düşürme koruma durumuna girecektir. Aşağıdaki göstergeler bu durumun ortaya çıktığını gösterecektir:

- RoboRIO üzerindeki güç LED'i kehribar rengine dönecektir
- Sürücü istasyonundaki voltaj göstergesinin arka planı kırmızıya dönecek
- Sürücü İstasyonundaki mod ekranı "Gerilim Kesintisi" olarak değişecektir
- DS'nin CAN/Güç sekmesi 12V arıza sayacını 1 artıracaktır.
- DS, DS günlüğüne bir kesinti olayı kaydeder.

Denetleyici, pil voltajını korumaya çalışmak için aşağıdaki adımları atacaktır:

- PWM çıkışları devre dışı bırakılacaktır. Nötr değerini ayarlamış olan PWM çıkışları için (WPILib'deki tüm motor kontrolörleri), çıkış devre dışı bırakılmadan önce tek bir nötr puls gönderilecektir.
- 6V, 5V, 3.3V User Rails devre dışı hale getirilir. (Bu, PWM pinlerindeki 6V çıkışları, DIO konektör bankasındaki 5V pinleri, Analog bankadaki 5V pinleri, SPI ve I2C bankasındaki 3.3V pinleri ve MXP bankasındaki 5V ve 3.3V pinlerini içerir)
- Çıkışlar olarak yapılandırılan GPIO, High-Z'ye gider
- Röle Çıkışları devre dışı bırakılır (düşük sürülür)

- CAN tabanlı motor kontrolörlerine açık bir devre dışı bırakma komutu gönderilir
- Pneumatic Devices such as the CTRE Pneumatics Control Module and REV Pneumatic Hub are disabled

Kontrol cihazı, voltaj 7,5V'nin üzerine çıkana veya voltaj kesintisinin bir sonraki aşaması için tetiğin altına düşene kadar bu durumda kalır

Aşama 3 - Cihaz Karartması

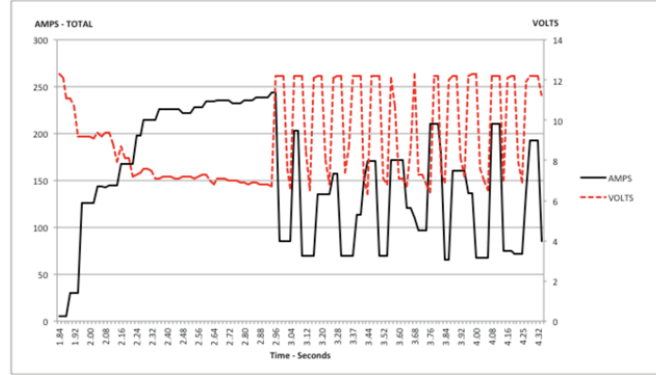
Voltage Trigger - Gerilim Tetikleyici - 4.5V

4.5V'nin altında cihaz elektrik kesintisine uğrayabilir. Kesin voltaj bundan daha düşük olabilir ve cihazdaki yüke bağlıdır.

Cihaz normal önyükleme sırasına başlayacağı zaman, kontrol cihazı voltaj 4,65V'nin üzerine çıkana kadar bu durumda kalacaktır.

30.5.2 Kesintiyi Önleme - Proaktif Akım Çekiş Planlaması

PLOT 1 – AMPS and VOLTS v. Time – 2.5 Second Window



Bir kesinti durumundan kaçınmanın anahtarı, robotunuzun mevcut çekimini proaktif olarak planlamaktır. Bunu yapmanın en iyi yolu, bir çeşit güç bütçesi yaratmaktır. Bu, güç kullanımını ve dolayısıyla bir maçın sonundaki pil durumunu en iyi şekilde anlamak için hem tahmini akım çekimini hem de süreyi ölçmeye çalışan karmaşık bir belge olabilir veya mevcut kullanımın basit bir envanteri olabilir. Bunu yapmak için:

1. Establish the max “sustained” current draw (with sustained being loosely defined here as not momentary). This is probably the most difficult part of creating the power budget. The exact current draw a battery can sustain while maintaining a voltage of 7+ volts is dependent on a variety of factors such as battery health (see [this](#) article for measuring battery health) and state of charge. As shown in the [NP18-12 data sheet](#), the terminal voltage chart gets very steep as state of charge decreases, especially as current draw increases. This datasheet shows that at 3CA continuous load (54A) a brand new battery can be continuously run for over 6 minutes while maintaining a terminal voltage of over 7V. As shown in the image above (used with permission from [Team 234s Drive System Testing document](#)), even with a fresh battery, drawing 240A for more than a second or two is likely to cause an issue. This gives us some bounds on setting our sustained current draw. For the purposes of this exercise, we’ll set our limit at 180A.
2. Robotunuzun aktarma organı, manipülatör, ana oyun mekanizması vb. Gibi farklı işlevlerini listeleyin.

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. Analizinizde karşılıklı olarak birbirini dışlayan işlevler belirlediyseniz, yazılımda dışlamayı zorunlu kılmayı düşünün. Ayrıca, çıktı limitleri veya istisnaları dinamik olarak sağlamak için (örneğin, aktarma organı akımı X’in üzerinde olduğunda bir mekanizma motorunu çalıştırmayın veya yalnızca motora izin vermeyin), robot programınızda PDP’nin akım izlemesini (aşağıda daha ayrıntılı olarak ele alınmıştır) kullanabilirsiniz. aktarma organı akımı Y’nin üzerinde olduğunda çıkışın yarısına kadar çalıştırın.

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 Kesintileri Tanımlama



Bir voltaj düşmesini belirlemenin en kolay yolu, DS'nin CAN \ Güç sekmesine tıklamak ve 12V hata sayısını kontrol etmektir. Alternatif olarak, Driver Station Log Viewer'ı kullandıktan sonra Driver Station Log'u gözden geçirebilirsiniz. Günlük, yukarıdaki görüntüdeki gibi parlak turuncu bir çizgi ile kesintileri belirleyecektir (bu kesintilerin bir tezgah üstü beslemeyle indüklendiğini ve tipik bir FRC robotundaki kesintilerin süresini ve davranışını yansıtmayabileceğini unutmayın).

30.6 Güvenli Modu kullanarak bir roboRIO'yu kurtarma

Bazen bir roboRIO, normal önyükleme ve görüntüleme işlemi kullanılarak kurtarılamayacak kadar bozulabilir. RoboRIO'yu Güvenli Modda başlatmak, aygıtın başarılı bir şekilde yeniden görüntülenmesini sağlayabilir.

Önemli: 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.

GradleRIO, robot kodunun roboRIO'ya dağıtımını sağlayan mekanizmadır. GradleRIO, popüler Gradle dependency ve derleme yönetim sistemi üzerine kurulmuştur. Bu bölümde, ekiplerin iş akışlarını geliştirmek için kullanabilecekleri **gelişmiş** yapılandırmalar vurgulanmaktadır.

31.1 Robot Koduyla Harici Kitaplıklar Kullanma

Uyarı: Harici kitaplıkları kullanmak, robot kodunuzla istenmeyen davranışlara neden olabilir! Ne yaptığının farkında değilseniz tavsiye edilmez!

Genellikle bir ekip, robot kodlarıyla kullanım için harici Java veya C++ kitaplıkları eklemek isteyebilir. Bu makale, Gradle bağımlılıklarınıza Java kitaplıkları eklemeyi veya C++ ekiplerinin sahip olduğu seçenekleri vurgulamaktadır.

31.1.1 Java

Not: Yerel kitaplıklara (JNI) dayanan herhangi bir dış bağımlılık muhtemelen çalışmayacaktır.

Java, harici bağımlılıklar eklemek için oldukça basittir. Sadece gerekli depoları ve bağımlılıkları eklemeniz yeterlidir.

Robot projelerinin varsayıldığı gibi `build.gradle` dosyasında bir `repositories {}` bloğu yoktur. Bunu kendiniz eklemek zorunda kalacaksınız. `dependencies {}` bloğunun üstüne lütfen aşağıdakileri ekleyin:

```
repositories {  
    mavenCentral()  
    ...  
}
```

`mavenCentral()`, içe aktarmak istediğiniz kitaplığın kullandığı herhangi bir depoya değiştirilebilir. Şimdi bağımlılığı kütüphanenin kendisine eklemelisiniz. Bu, `dependencies {}` bloğunuza gerekli satırı ekleyerek yapılır. Aşağıdaki örnek, Gradle projenize Apache Commons eklemeyi göstermektedir.

```
dependencies {  
    implementation 'org.apache.commons:commons-lang3:3.6'  
    ...  
}
```

Şimdi bir yapı çalıştırıyorsunuz ve bağımlılıkların indirildiğinden emin oluyorsunuz. Bir yapı çalıştırılana kadar Intellisense düzgün çalışmayabilir!

31.1.2 C ++

Robot projenize C++ bağımlılıkları eklemek, roboRIO için derlemeye ihtiyaç duyması nedeniyle önemsiz değildir. Birkaç seçeneğin var.

1. İsteddiğiniz kitaplığın kaynak kodunu robot projenize kopyalayın.
2. Örnek olarak `satıcı dep şablonunu` kullanın ve bir satıcı departmanı oluşturun.

Kaynak Kodu Kopyalama

Gerekli kaynağı ve/veya başlıkları robot projenize kopyalamanız yeterlidir. Daha sonra aşağıdaki gibi gerekli platform argümanlarını yapılandırabilirsiniz:

```
nativeUtils.platformConfigs.named("linuxx86-64").configure {  
    it.linker.args.add('-lstdc++fs') // links in C++ filesystem library  
}
```

Bir Satıcı Departmanı Oluşturma

Lütfen `vendordep deposu` içindeki talimatlara uyun.

31.2 GitHub Eylemlerini Kullanarak Robot Kodu için CI kurmak

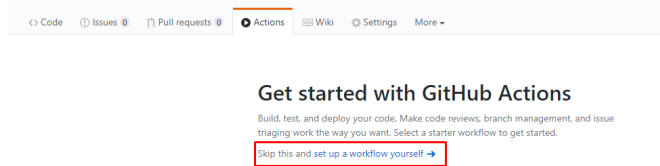
Bir ekip ortamında çalışmanın önemli bir yönü, GitHub gibi merkezi bir depoya gönderilen kodu test edebilmektir. Örneğin, proje yöneticisi veya lider geliştirici, kodu birleştirmeden önce bir dizi birim testini çalıştırmak isteyebilir veya tüm kodun çalışır durumda olduğundan emin olmak isteyebilir.

'GitHub Eylemleri <<https://github.com/features/actions>>' ekiplerin ve bireylerin çekme isteklerinde çeşitli dallardaki kod üzerinde birim testleri oluşturmaya ve çalıştırmasına olanak tanıyan bir hizmettir. Bu tip servisler genel olarak "Continuous Integration-Sürekli Entegrasyon" hizmetleri şeklinde bilinir. Bu öğretici yazı size GitHub Aksiyonlarını robot kodu projelerinde nasıl kurulacağını göstericektir.

Not: This tutorial assumes that your team's robot code is being hosted on GitHub. For an introduction to Git and GitHub, please see this [introduction guide](#).

31.2.1 Eylem oluşturma

CI sürecini gerçekleştirme talimatları bir YAML dosyasında depolanır. Bunu oluşturmak için, deponun sağ üstünde bulunan “Actions_Eylemler” sekmesine tıklayın. Ardından “set up a workflow yourself-kendi başınıza bir iş akışı oluşturun” bağlantısına tıklayın.



Şimdi bir metin editörü ile karşılanacaksınız. Tüm varsayılan metni aşağıdakilerle değiştirin:

```
# 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

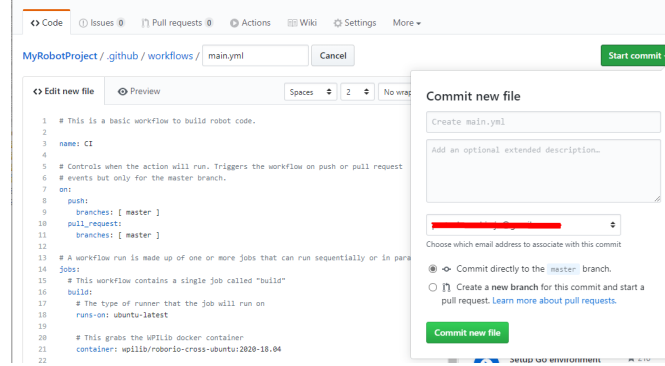
      # Grant execute permission for gradlew
      - name: Grant execute permission for gradlew
        run: chmod +x gradlew
```

(sonraki sayfaya devam)

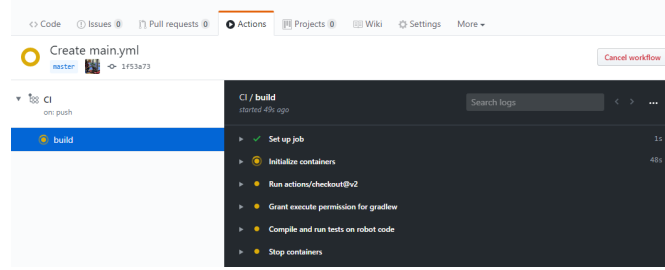
(önceki sayfadan devam)

```
# Runs a single command using the runners shell
- name: Compile and run tests on robot code
  run: ./gradlew build
```

Ardından, ekranın sağ üst köşesinde bulunan “Start commit-işlemeye başla” butonuna tıklayarak değişiklikleri kaydedin. İsterseniz varsayılan kaydetme mesajını düzenleyebilirsiniz. Sonrasında, “Commit new file-yeni dosya işle” yeşil butonuna tıklayın.



Ana bilgisayarda bir işlem gerçekleştiğinde, GitHub artık her zaman otomatik olarak çalıştıracaktır. Herhangi bir yapının durumunu izlemek için ekranın üst kısmındaki “Actions - eylemler” sekmesine tıklayabilirsiniz.



31.2.2 Eylemler YAML Dosyasının Dökümü

İşte yukarıdaki YAML dosyasının bir dökümü. Her satırın tam olarak anlaşılması gerekmesede, bir miktar anlayış, daha fazla özellik eklemenize ve ortaya çıkabilecek olası sorunları gidermenize yardımcı olacaktır.

```
# 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 ]
```

Bu kod bloğu, eylemin ne zaman çalışacağını belirler. Şu anda eylem, işlem yürütüldüğünde yada istendiğinde çalışacaktır.

```
# A workflow run is made up of one or more jobs that can run sequentially or in
→ parallel
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

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

```

Her aksiyonun iş akışı, sıralı olarak (biri ardına) veya paralel (aynı anda) çalışan bir veya daha fazla işten oluşur. İş akışımızda sadece bir “yapım” işi vardır.

Şunu belirtmek isteriz ki , işin bir Ubuntu sanal makinesinde ve sanallaştırılmış bir ‘JDK, C ++ derleyici ve roboRIO araç zincirleri içeren bir Docker konteynerinde <<https://www.docker.com/resources/what-container>>`_ çalışmasını istiyoruz.

```

# 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

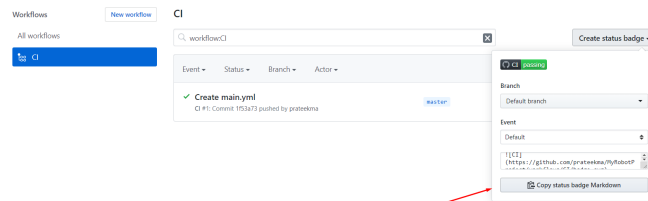
# Runs a single command using the runners shell
- name: Compile and run tests on robot code
  run: ./gradlew build

```

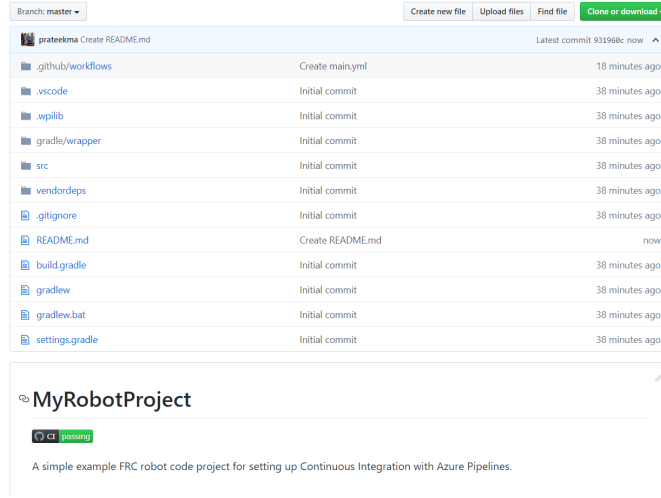
Each job has certain steps that will be executed. This job has four steps. The first step involves checking out the repository to access the robot code. The second step is a workaround for a [GitHub Actions issue](#). The third step involves giving the virtual machine permission to execute gradle tasks using ./gradlew. The final step runs ./gradlew build to compile robot code and run any unit tests.

31.2.3 README.md dosyasına Yapı Durum işareti ekleme

Main üzerinde en son derlemenin durumunu hızlı bir şekilde kontrol etmek için deponuzun README dosyasının üstüne bir CI durum işareti eklemek yararlıdır. Bunu yapmak için, ekranın üst kısmındaki “Actions-eylemeler” sekmesine tıklayın ve ekranın sol tarafındaki “CI” sekmesini seçin. Ardından, sağ üstteki “Create status badge-durum işareti oluşturun” düğmesini tıklayın ve durum işareti Markdown kodunu kopyalayın.



Son olarak, Kopyaladığınız Markdown kodunu README dosyanızın en üstüne yapıştırın, uygulayın ve değişikliklerinizi gönderin. Şimdi, ana depo sayfanızda GitHub Aksiyonları durum rozetini görmelisiniz.



31.3 Kod Biçimlendirici - Code Formatter Kullanma

Code formatters exist to ensure that the style of code written is consistent throughout the entire codebase. This is used in many major projects; from Android to OpenCV. Teams may wish to add a formatter throughout their robot code to ensure that the codebase maintains readability and consistency throughout.

Bu makalede; Java kullanan takımlar için [Spotless](https://github.com/wpilibsuite/styleguide/blob/main/wpiformat/README.rst), C++ kullanan takımlar için de [__](https://github.com/wpilibsuite/styleguide/blob/main/wpiformat/README.rst) kullanmayı vurgulayacağız

31.3.1 Spotless

Yapılandırma

Necessary build.gradle changes are required to get Spotless functional. In the plugins {} block of your build.gradle, add the Spotless plugin so it appears similar to the below.

```
plugins {
    id "java"
    id "edu.wpi.first.GradleRIO" version "2024.1.1"
    id 'com.diffplug.spotless' version '6.20.0'
}
```

Then ensure you add a required spotless {} block to correctly configure spotless. This can just get placed at the end of your build.gradle.

```
spotless {
    java {
        target fileTree('.') {
            include '**/*.java'
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

        exclude '**/build/**', '**/build-*/**'
    }
    toggleOffOn()
    googleJavaFormat()
    removeUnusedImports()
    trimTrailingWhitespace()
    endWithNewline()
}
groovyGradle {
    target fileTree('.') {
        include '**/*.gradle'
        exclude '**/build/**', '**/build-*/**'
    }
    greclipse()
    indentWithSpaces(4)
    trimTrailingWhitespace()
    endWithNewline()
}
format 'xml', {
    target fileTree('.') {
        include '**/*.xml'
        exclude '**/build/**', '**/build-*/**'
    }
    eclipseWtp('xml')
    trimTrailingWhitespace()
    indentWithSpaces(2)
    endWithNewline()
}
format 'misc', {
    target fileTree('.') {
        include '**/*.md', '**/.gitignore'
        exclude '**/build/**', '**/build-*/**'
    }
    trimTrailingWhitespace()
    indentWithSpaces(2)
    endWithNewline()
}
}

```

Spotless Çalıştırmak

Spotless can be ran using `./gradlew spotlessApply` which will apply all formatting options. You can also specify a specific task by just adding the name of formatter. An example is `./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

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
runs-on: ubuntu-latest
# Steps represent a sequence of tasks that will be executed as part of the job
steps:
  # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
  - uses: actions/checkout@v4
    with:
      fetch-depth: 0
  - uses: actions/setup-java@v4
    with:
      distribution: 'zulu'
      java-version: 17
  - run: ./gradlew spotlessCheck
```

Seçeneklerin Açıklaması

Each format section highlights formatting of custom files in the project. The java and groovyGradle are natively supported by spotless, so they are defined differently.

Breaking this down, we can split this into multiple parts.

- Java'yı Biçimlendirmek
- Gradle Dosyalarını Biçimlendirmek
- XML Dosyalarını Biçimlendirmek
- Miscellaneous - Çeşitli Dosyaları Biçimlendirmek

They are all similar, except for some small differences that will be explained. The below example will highlight the java {} block.

```
java {
  target fileTree('.') {
    include '**/*.java'
    exclude '**/build/**', '**/build-*/**'
  }
  toggleOff0n()
  googleJavaFormat()
  removeUnusedImports()
  trimTrailingWhitespace()
  endWithNewline()
}
```

Seçeneklerin her birinin ne anlama geldiğini açıklayalım.

```
target fileTree('.') {
  include '**/*.java'
  exclude '**/build/**', '**/build-*/**'
}
```

The above example tells spotless where our Java classes are and to exclude the build directory. The rest of the options are fairly self-explanatory.

- `toggleOff0n()` adds the ability to have spotless ignore specific portions of a project. The usage looks like the following

```
// format:off

public void myWeirdFunction() {

}

// format:on
```

- `googleJavaFormat()` tells spotless to format according to the [Google Style Guide](#)
- `removeUnusedImports()` will remove any unused imports from any of your Java classes
- `trimTrailingWhitespace()` will remove any extra whitespace at the end of your lines
- `endWithNewline()` will add a newline character to the end of your classes

In the `groovyGradle` block, there is a `greclipse` option. This is the formatter that spotless uses to format gradle files.

Additionally, there is a `eclipseWtp` option in the `xml` block. This stands for “Gradle Web Tools Platform” and is the formatter to format xml files. Teams not using any XML files may wish to not include this configuration.

Not: A full list of configurations is available on the [Spotless README](#)

Satır Sonları ile ilgili sorunlar

Spotless will attempt to apply line endings per-OS, which means Git diffs will be constantly changing if two users are on different OSes (Unix vs Windows). It's recommended that teams who contribute to the same repository from multiple OSes utilize a `.gitattributes` file. The following should suffice for handling line endings.

```
*.gradle text eol =lf
*.java text eol =lf
*.md text eol =lf
*.xml text eol =lf
```

31.3.2 wpiformat

Gereksinimler

- Python 3.6 veya daha yüksek

You can install `wpiformat` by typing `pip3 install wpiformat` into a terminal or command prompt.

Kullanım

wpiformat can be ran by typing wpiformat in a console. This will format with clang-format. Three configuration files are required (.clang-format, .styleguide, .styleguide-license). These must exist in the project root.

- .clang-format: İndir
- .styleguide-license: İndirme

Örnek bir stil kılavuzu aşağıda gösterilmiştir:

```
cppHeaderFileInclude {
  \.h$
  \.hpp$
  \.inc$
  \.inl$
}

cppSrcFileInclude {
  \.cpp$
}

modifiableFileExclude {
  gradle/
}
```

Not: Teams can adapt .styleguide and .styleguide-license however they wish. It's important that these are not deleted, as they are required to run wpiformat!

You can turn this into a [CI check](#) by running `git --no-pager diff --exit-code HEAD`, as shown in the example GitHub Actions workflow below:

```
name: Lint and Format

on:
  pull_request:
  push:
jobs:
  wpiformat:
    name: "wpiformat"
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - 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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
- 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 $T00L$` - Runs the tool `$T00L$` (Replace `$T00L$` with the name of the tool. IE, Glass, Shuffleboard, etc)

`./gradlew $T00L$Install` - Installs the java tool `$T00L$` (Replace `$T00L$` 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 [3. Taraf Kütüphaneleri](#)

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.

Not: For Python teams, Git metadata is always copied to your robot during the deploy process. You can use `wplib.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";
    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`.

Uyari: 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 it looks like.

```

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.
20             // You can use getTeamOrDefault(team) instead of getTeamNumber if you
21             // want to store a team number in this file.
22             team = project.frc.getTeamNumber()
23             debug = project.frc.getDebugOrDefault(false)
24
25             artifacts {
26                 // First part is artifact name, 2nd is artifact type
27                 // getTargetTypeClass is a shortcut to get the class type using a
28                 ↪ string
29                 frcJava(getArtifactTypeClass('FRCJavaArtifact')) {
30                 }
31
32                 // Static files artifact
33                 frcStaticFileDeploy(getArtifactTypeClass('FileTreeArtifact')) {
34                     files = project.fileTree('src/main/deploy')
35                     directory = '/home/lvuser/deploy'
36                 }
37             }
38         }
39     }
40 }

```

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 roboRIO's team number. Be sure to replace this.

Önemli: The hostname when connected via USB-B should be 172.22.11.2.

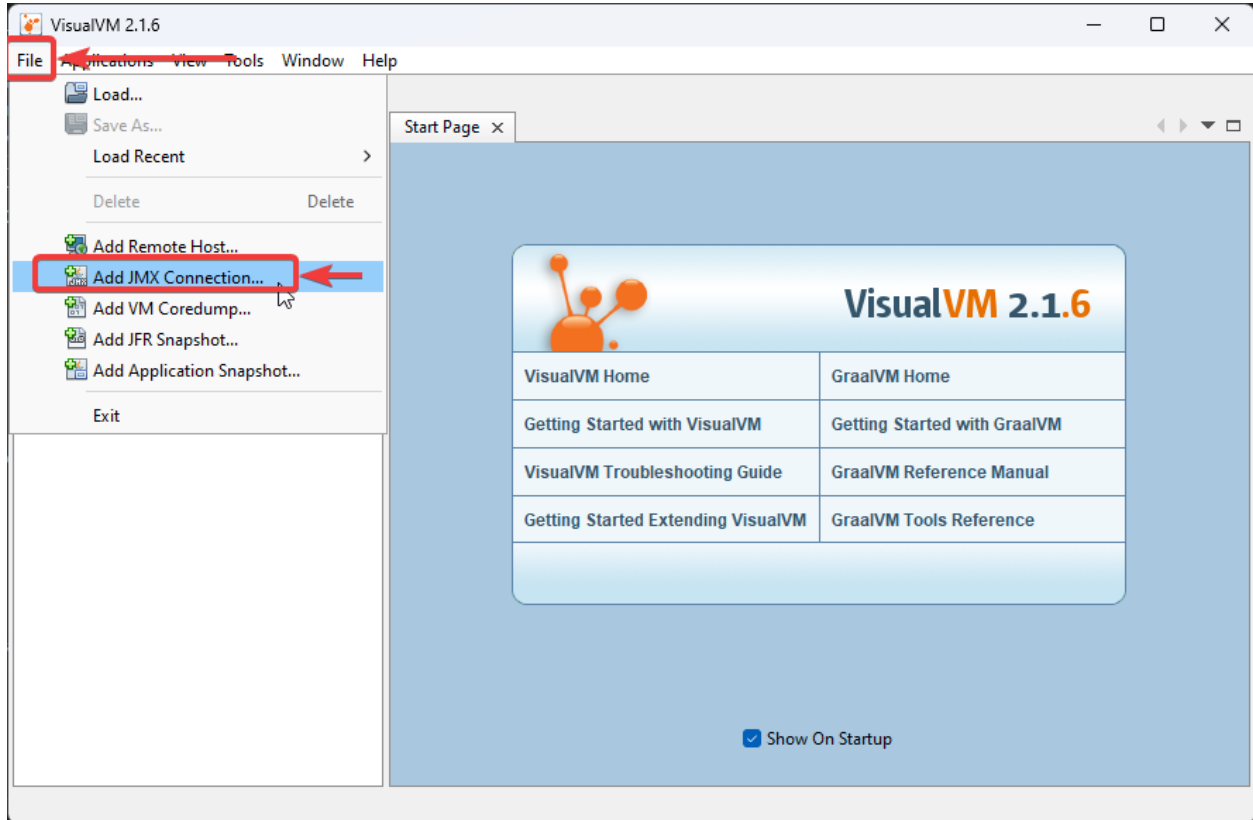
31.7.3 Running VisualVM

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:

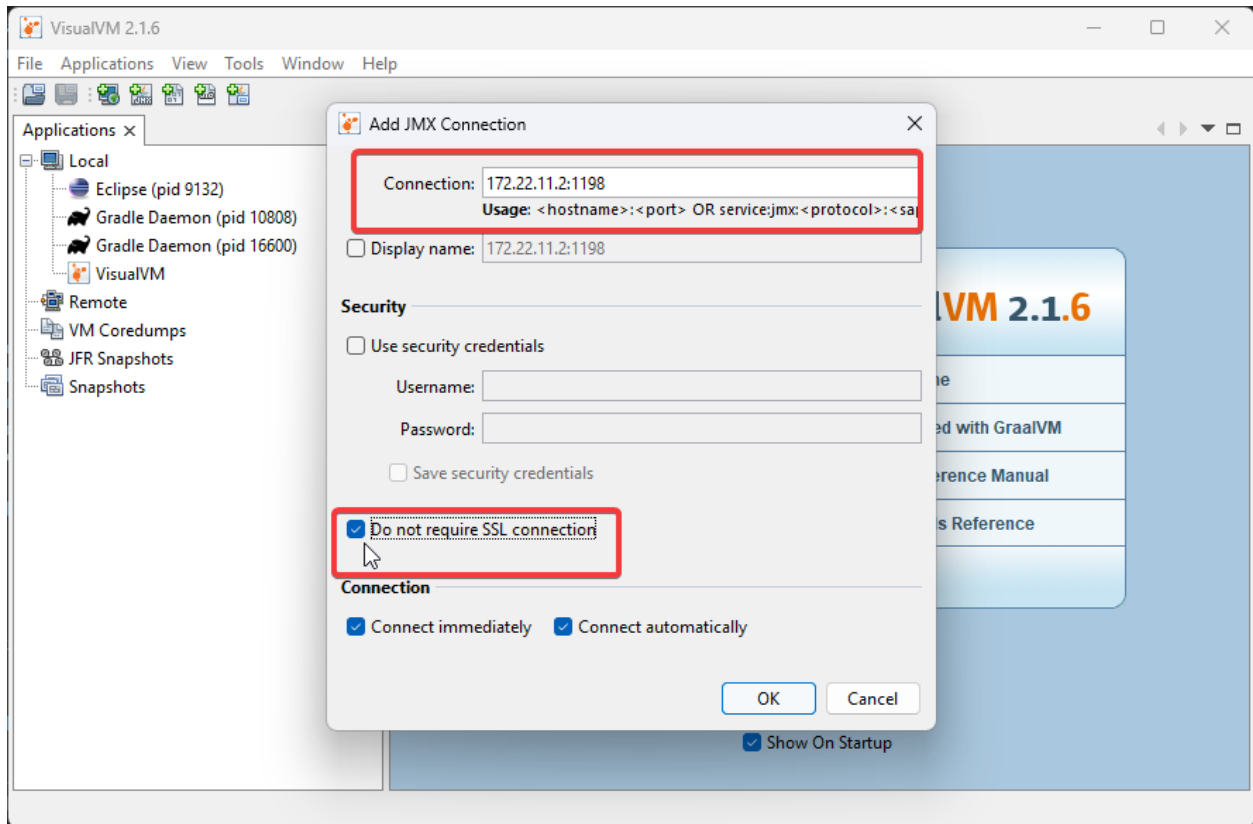
```
cd "C:\Users\Public\wpilib\visualvm_217\bin"
./visualvm --jdkhome "C:\Users\Public\wpilib\2024\jdk"
```

Önemli: The exact path `visualvm_217` may vary and depends on the version of VisualVM downloaded.

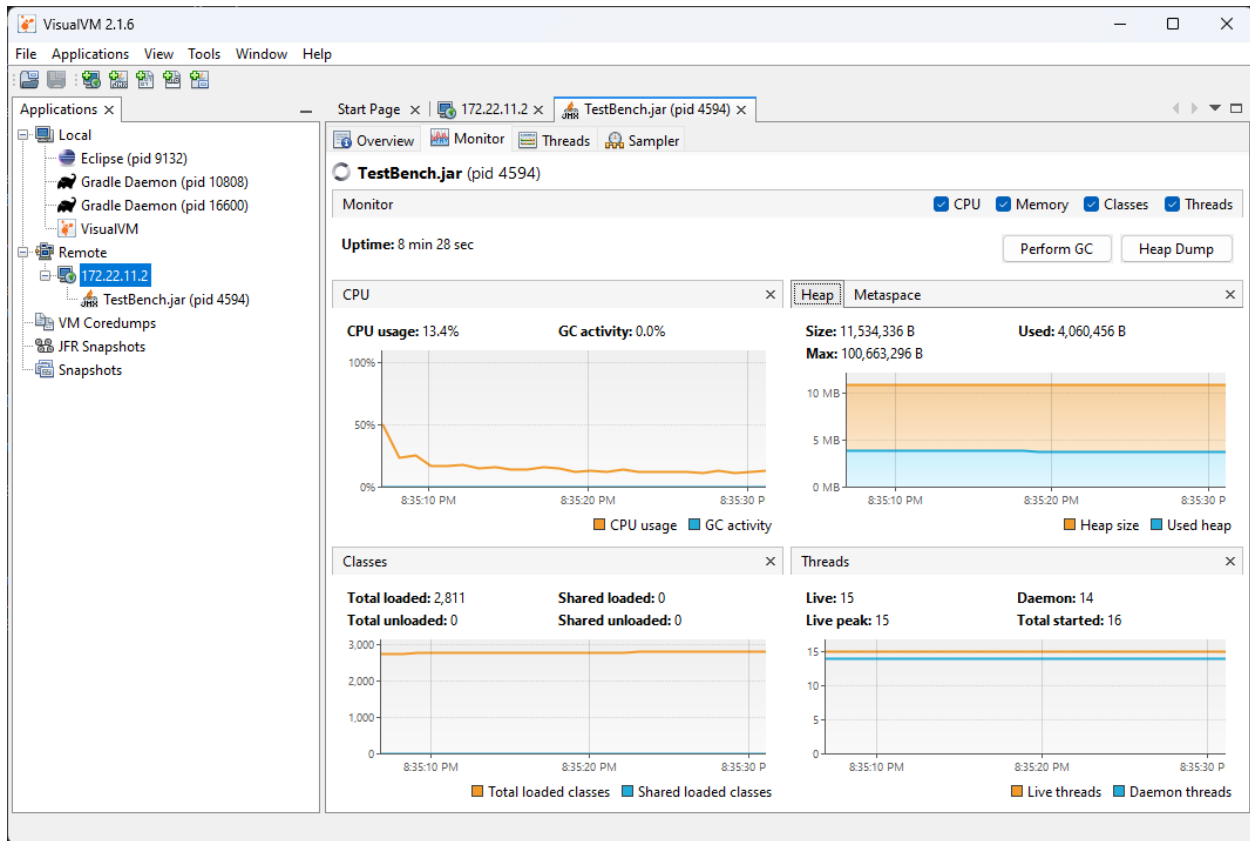
This should launch VisualVM. Once launched, open the *Add JMX Connection* dialog.



Once opened, configure the connection details and hostname. Ensure that *Do not require SSL connection* is ticked.

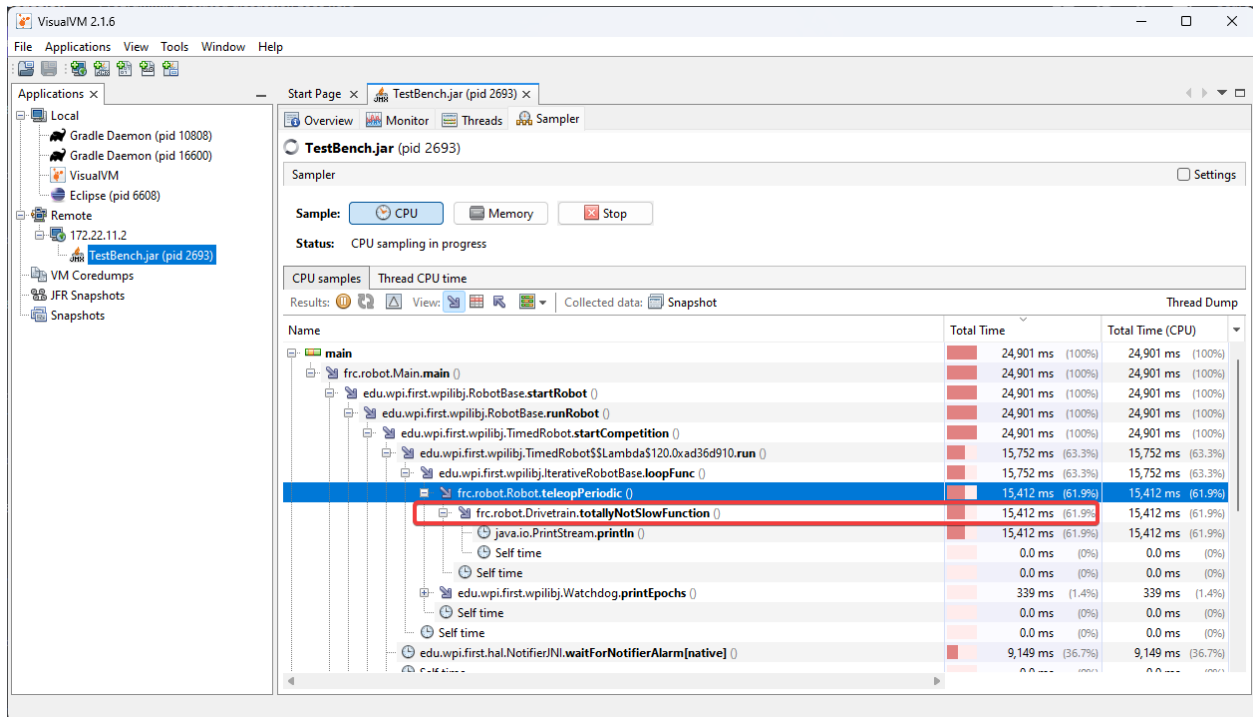


If correctly done, a new menu option in the left-hand sidebar will appear. Clicking on it will show you a detailed dashboard of the running JVM application.



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.

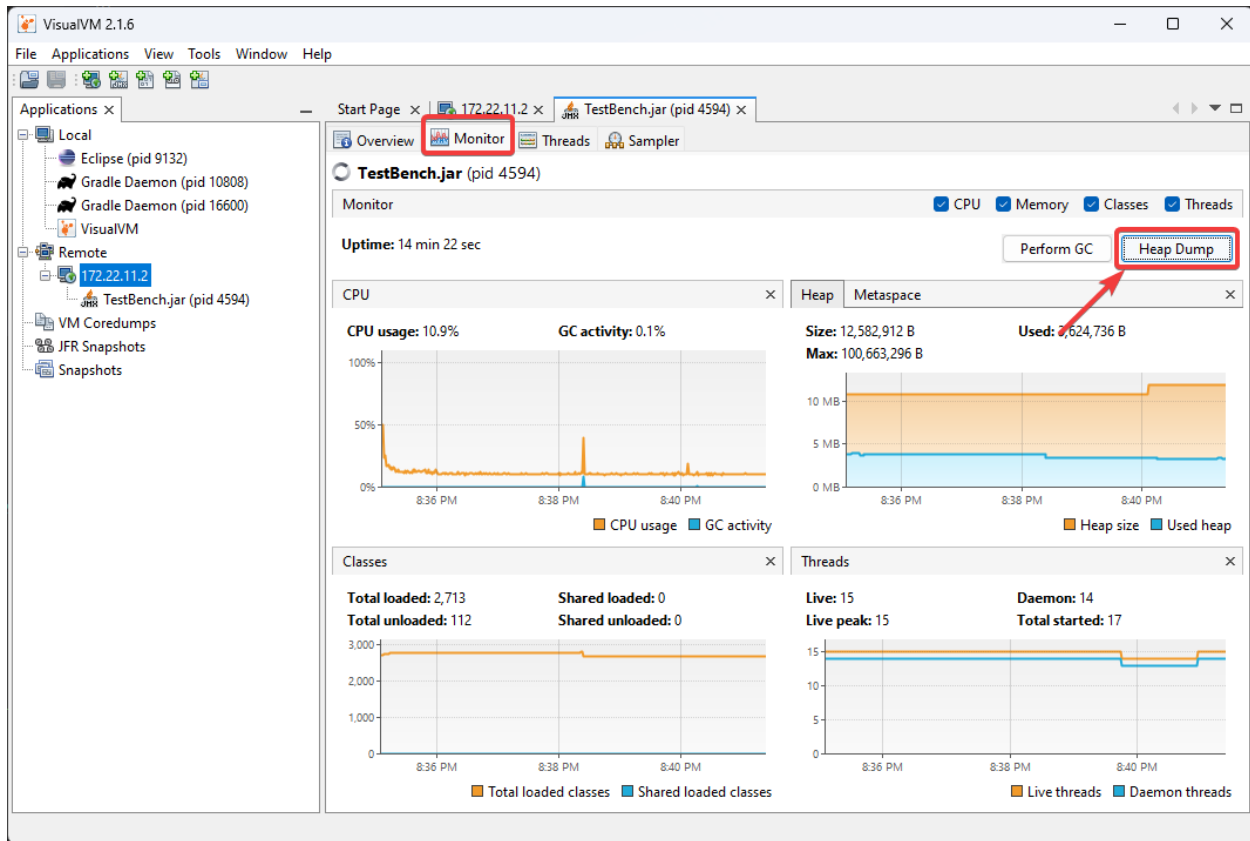
```
public static void totallyNotSlowFunction() {
    for (int i = 0; i < 2000; i++) {
        System.out.println("HAHAHAHA");
    }
}
```

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.

Tüyo: 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.

VisualVM 2.1.6

File Applications View Tools Window Help

Applications x

Start Page x | 172.22.11.2 x | TestBench.jar (pid 4594) x | [heapdump] 8:42:08 PM x

[heapdump] 8:42:08 PM

Heap Dump

Summary

Heap

Size: 3,267,072 B

Classes: 2,964

Instances: 76,477

Classloaders: 108

GC Roots: 2,752

Objects Pending for Finalization: 0

Environment

System: Linux (4.14.146-rt67)

Architecture: arm 32bit

Java Home: /usr/local/jre

Java Version: 17.0.3.7-frc 2022-04-19

Java Name: c+0-2023-17.0.5u7-1, mixed mode, emulated-client

Java Vendor: N/A

JVM Uptime: 15 min 07 sec

JVM Arguments [show]

Enabled Modules [show]

System Properties [show]

Classes by Number of Instances [view all]

Class	Count	Percentage
byte[]	15,213	(19.9%)
java.lang.String	14,666	(19.2%)
java.util.HashMap\$Node	3,738	(4.9%)
java.util.concurrent.ConcurrentHashMap\$Node	3,308	(4.3%)
java.lang.Object[]	3,188	(4.2%)

Classes by Size of Instances [view all]

Class	Count	Percentage
byte[]	1,308,048 B	(40%)
java.lang.String	351,984 B	(10.8%)
java.lang.Object[]	161,944 B	(5%)
int[]	90,824 B	(2.8%)
java.util.HashMap\$Node	89,712 B	(2.7%)

Instances by Size [view all]

Class	Count	Percentage
byte[]#1937 : 310,072 items	310,088 B	(9.5%)
int[]#292 : 9,504 items	38,032 B	(1.2%)

Dominators by Retained Size [view all]

Retained sizes must be computed first:

Compute Retained Sizes

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:

VisualVM 2.1.6

File Applications View Tools Window Help

Applications x

Start Page x | 172.22.11.2 x | TestBench.jar (pid 4594) x | [heapdump] 8:42:08 PM x

[heapdump] 8:42:08 PM

Heap Dump

Objects

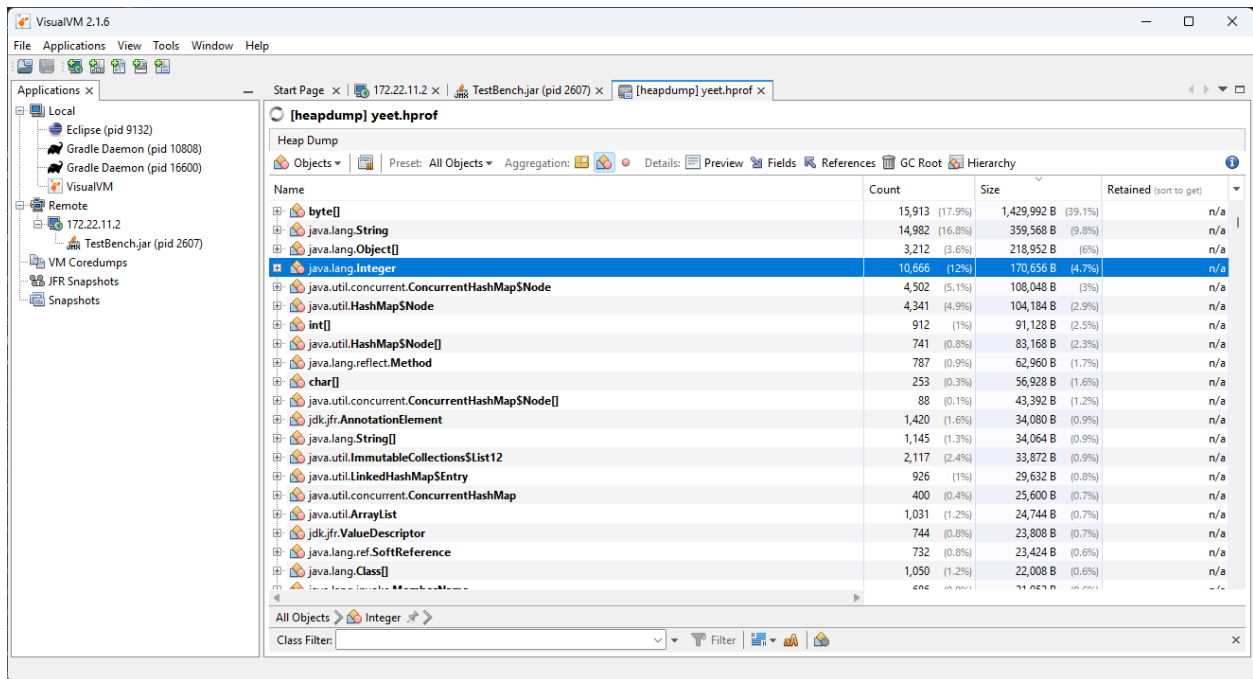
Presets: All Objects Aggregation Details Preview Fields References GC Root Hierarchy

Name	Count	Size	Retained (sort to get)
byte[]	15,213 (19.9%)	1,308,048 B (40%)	n/a
java.lang.String	14,666 (19.2%)	351,984 B (10.8%)	n/a
java.lang.Object[]	3,188 (4.2%)	161,944 B (5%)	n/a
int[]	908 (1.2%)	90,824 B (2.8%)	n/a
java.util.HashMap\$Node	3,738 (4.9%)	89,712 B (2.7%)	n/a
java.lang.reflect.Method	1,020 (1.3%)	81,600 B (2.5%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node	3,308 (4.3%)	79,392 B (2.4%)	n/a
java.util.HashMap\$Node[]	740 (1%)	79,056 B (2.4%)	n/a
char[]	250 (0.3%)	55,888 B (1.7%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node[]	85 (0.1%)	35,088 B (1.1%)	n/a
jdk.jfr.AnnotationElement	1,420 (1.9%)	34,080 B (1%)	n/a
java.util.ImmutableCollections\$List12	2,115 (2.8%)	33,840 B (1%)	n/a
java.lang.String[]	1,144 (1.5%)	31,648 B (1%)	n/a
java.util.LinkedHashMap\$Entry	926 (1.2%)	29,632 B (0.9%)	n/a
java.util.concurrent.ConcurrentHashMap	423 (0.6%)	27,072 B (0.8%)	n/a
java.lang.Class[]	1,319 (1.7%)	26,792 B (0.8%)	n/a
java.util.ArrayList	1,038 (1.4%)	24,912 B (0.8%)	n/a
jdk.jfr.ValueDescriptor	744 (1%)	23,808 B (0.7%)	n/a
java.lang.ref.SoftReference	740 (1%)	23,680 B (0.7%)	n/a
java.lang.reflect.Constructor	304 (0.4%)	21,888 B (0.7%)	n/a

All Objects

Class Filter:

with an ArrayList of ~10000 integers.



31.7.7 Additional Info

For more information on VisualVM, check out the [VisualVM documentation pages](#).

Bu bölüm, çeşitli geri besleme/ileri besleme kontrol algoritmaları ve rota izleme gibi WPILib'deki gelişmiş kontrol özelliklerini kapsar.

32.1 FRC'de Otonomun Modele Dayalı Doğrulamasının Videosu

WPILib ekibinden Tyler Veness, 2020'de "WPI Tarafından Sunulan RSN Bahar Konferansı"nda FRC®'da Model Bazlı Otonom Doğrulaması üzerine bir sunum yaptı.

The link to the presentation is available [here](#).

32.2 Gelişmiş Kontrollere Giriş

32.2.1 Kontrol Sistemi Temelleri

Not: This article includes sections of [Controls Engineering in FRC](#) by Tyler Veness with permission.

The Need for Control Systems

Kontrol sistemleri her yeredir ve onlarla her gün etkileşim halindeyiz. Görmüş olabileceğinizin küçük bir listesi, termostatlı ısıtıcılar ve klimalar, hız kontrolünü ve otomobillerde kilitlenmeyi önleyici fren sistemini (ABS) ve modern dizüstü bilgisayarlarda fan hızı modülasyonunu içerir. Kontrol sistemleri, bu gibi sistemlerin davranışını izler veya kontrol eder ve onları doğrudan kontrol eden insanlardan (manuel kontrol) veya yalnızca makinelerden (otomatik kontrol) oluşabilir.

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.

Örneğin, otonom bir arabadaki kapalı döngü denetleyicilerin belirsizlik durumunda güvenli bir şekilde davranacağını ve istenen performans özelliklerini karşılayacağını nasıl kanıtlayabiliriz? Kontrol teorisi, sistemlerin davranışını analiz etmek ve tahmin etmek, onları istediğimiz şekilde yanıt vermelerini sağlamak ve onları gürültülere ve belirsizliğe karşı sağlam kılmak için kullanılan bir matematik ve geometri uygulamasıdır.

Kontrol mühendisliği, basitçe ifade etmek gerekirse, kontrol teorisine uygulanan mühendislik sürecidir. Bu nedenle, uygulamalı matematikten daha fazlasıdır. Kontrol teorisinin arkasında güzel bir matematik bulunsa da, kontrol mühendisliği diğer herhangi bir mühendislik disiplini gibi değişik tokuşlarla doludur. Kontrol teorisinin verdiği çözüm her zaman mantık kontrolünden geçirilmeli ve performans özellikleri tarafından değerlendirilmelidir. Mükemmel olmamıza gerek yok; sadece şartnamelerimizi karşılayacak kadar iyi olmamız gerekiyor.

Terminoloji - isimlendirme

İleri mühendislik konuları için çoğu kaynak, gerekli olanın çok üzerinde bir bilgi düzeyini varsayar. Sorunun bir kısmı, jargon kullanımıdır. Fikirleri sahadakilere verimli bir şekilde iletirken, bu jargona aşina olmayan insanlar konu içinde kaybolur.

Bir kontrol sistemi tarafından kontrol edilen aktüatör sistemi veya koleksiyonu, *plant-tesis* olarak adlandırılır. Plant-tesis mevcut durumundan istenen bir duruma (reference-referans) sürmek için bir kontrolör kullanılır. Plant-tesis çıktılarından ölçülen bilgileri içermeyen denetleyicilere açık döngü-open-loop denetleyiciler denir.

Tesisin çıkışından geri beslenen bilgileri içeren denetleyicilere kapalı döngü - closed-loop denetleyiciler veya geri bildirim-feedback denetleyicileri denir.

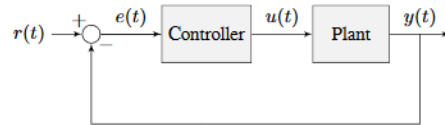


Figure 1.1: Control system nomenclature for a closed-loop system

$r(t)$	reference	$u(t)$	control input
$e(t)$	error	$y(t)$	output

Not: Bir sistemin girdi ve çıktıları plant bakış açısından tanımlanır. Gösterilen negatif geri besleme kontrolörü, hata olarak da bilinen referans ve çıkış arasındaki farkı sıfıra doğru sürüyor.

Kazanç-Gain nedir?

Kazanç-Gain, sabit durumda bir giriş sinyalinin büyüklüğü ile bir çıkış sinyalinin büyüklüğü arasındaki ilişkiyi gösteren orantılı bir değerdir. Birçok sistem, sisteme az ya da çok “güç-power” sağlayan, kazancın değiştirilebileceği bir yöntem içerir.

Aşağıdaki şekil varsayımsal bir girdi ve çıktıya sahip bir sistemi göstermektedir. Çıkış, girişin genliğinin iki katı olduğu için, sistemin kazancı ikidir.

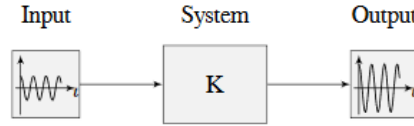


Figure 1.2: Demonstration of system with a gain of $K = 2$

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.

Blok Diyagramlar

Bir kontrol sistemini tasarlarken veya analiz ederken, onu grafiksel olarak modellemek faydalıdır. Bu amaçla blok diyagramlar kullanılır. Sistematik olarak manipüle edilebilir ve basitleştirilebilirler.

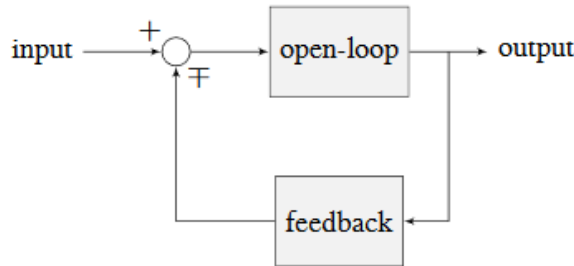


Figure 1.3: Block diagram with nomenclature

Açık döngü-open-loop kazancı, girişteki (daire) toplam düğümünden çıkış dalına olan toplam kazançtır. Bu, geri besleme döngüsünün bağlantısı kesildiğinde sistemin kazancı olacaktır. Geri besleme kazancı, çıkıştan giriş toplam düğümüne geri gelen toplam kazançtır. Bir toplam düğümünün çıktısı, girdilerinin toplamıdır.

Aşağıdaki şekil, geri besleme konfigürasyonunda daha resmi gösterime sahip bir blok diyagramdır.

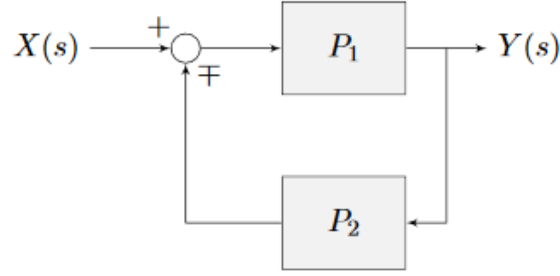


Figure 1.4: Feedback block diagram

± “eksi veya artı” anlamına gelir; burada eksi, olumsuz geri bildirimi temsil eder.

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

Not: 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:

Not: 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 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 k_V and k_P) 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

Not: For a guide on implementing PID control in code with WPILib, see [WPILib'de Feedforward Kontrolü](#).

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”.

$$V = K_s \cdot \text{sgn}(\dot{d}) + K_v \cdot \dot{d} + K_a \cdot \ddot{d}$$

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 K_s . 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 K_v , 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:

$$V = K_g + K_s \cdot \text{sgn}(\dot{d}) + K_v \cdot \dot{d} + K_a \cdot \ddot{d}$$

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):

$$V = K_g \cdot \cos(\theta) + K_s \cdot \text{sgn}(\dot{\theta}) + K_v \cdot \dot{\theta} + K_a \cdot \ddot{\theta}$$

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’ye Giriş

Not: For a guide on implementing PID control with WPILib, see [WPILib’de PID Kontrolü](#).

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.

Not: 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.

Proportional Term - Oransal terimi

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*.

$$u(t) = K_p e(t)$$

burada K_p orantılı kazançtır ve $e(t)$ geçerli zamandaki t hatadır.

Aşağıdaki şekil, bir P kontrolörü tarafından kontrol edilen bir *system* için bir blok diyagramı göstermektedir.

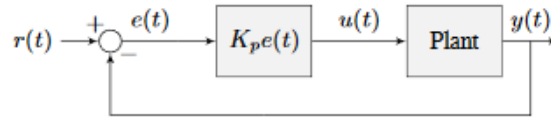


Figure 2.1: P controller block diagram

Orantılı kazançlar, *system* i istenen konuma çeken “yazılım tanımlı yaylar” gibi hareket eder. Fizikten şunu hatırlayın yayı modellemek için $F = -kx$ burada F uygulanan kuvvettir, k is a orantısal bir sabit ve x denge noktasından yer değiştirmedir. Bu başka bir şekilde yazılabilir $F = k(0 - x)$ burada 0 denge noktasıdır. Denge noktasının geri besleme denetleyicimizin *setpoint* olmasına izin verirse, denklemlerin bire bir karşılığı olur.

$$F = k(r - x)$$
$$u(t) = K_p e(t) = K_p (r(t) - y(t))$$

dolayısıyla orantısal denetleyicinin *systemin output* u : term: *setpoint* yönünde çektiği “kuvvet”, bir yay gibi *error* ile orantılıdır.

Derivative- Türev Terimi

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*.

$$u(t) = K_p e(t) + K_d \frac{de}{dt}$$

burada K_p oransal kazançtır, K_d türevsel kazançtır ve $e(t)$ şu andaki t zamanındaki hatadır.

Aşağıdaki şekil, bir PD kontrolörü tarafından kontrol edilen bir *system* için bir blok diyagramı göstermektedir.

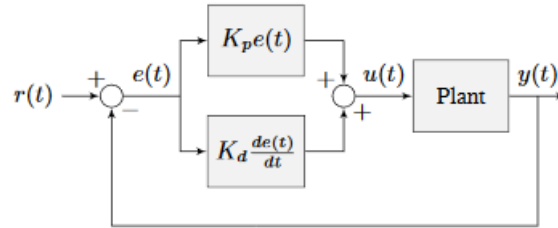


Figure 2.2: PD controller block diagram

Bir PD denetleyicisinde konum için oransal bir denetleyici (K_p) ve hız için oransal denetleyici (K_d) bulunur. Hız ile ilgili *setpoint*, örtülü olarak pozisyon *setpoint* inin zaman içinde nasıl değiştiğiyle sağlanır. Bunu kanıtlamak için, bir PD kontrolörü için denklemi yeniden düzenleyeceğiz.

$$u_k = K_p e_k + K_d \frac{e_k - e_{k-1}}{dt}$$

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 .

$$u_k = K_p(r_k - x_k) + K_d \frac{(r_k - x_k) - (r_{k-1} - x_{k-1})}{dt}$$

$$u_k = K_p(r_k - x_k) + K_d \frac{r_k - x_k - r_{k-1} + x_{k-1}}{dt}$$

$$u_k = K_p(r_k - x_k) + K_d \frac{r_k - r_{k-1} - x_k + x_{k-1}}{dt}$$

$$u_k = K_p(r_k - x_k) + K_d \frac{(r_k - r_{k-1}) - (x_k - x_{k-1})}{dt}$$

$$u_k = K_p(r_k - x_k) + K_d \left(\frac{r_k - r_{k-1}}{dt} - \frac{x_k - x_{k-1}}{dt} \right)$$

Nasıl olduğuna dikkat edin: $\frac{r_k - r_{k-1}}{dt}$, : term : *setpoint* in hızıdır. Aynı nedenden ötürü, $\frac{x_k - x_{k-1}}{dt}$, *systemin* belirli bir zaman adımıdaki hızıdır. Bu, PD kontrol cihazının K_d teriminin, tahmini hızı *setpoint* hızına sürdüğü anlamına gelir.

setpoint sabitse, örtük hız *setpoint* sıfırdır, dolayısıyla K_d terimi *system* i yavaşlatır. Bu, “yazılım tanımlı bir tampon” gibi davranır. Bunlar genellikle kapı kapatıcılarda görülür ve sönümleme kuvveti hız ile doğrusal olarak artar.

İntegral Terimi

Önemli: 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.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

burada K_p oransal kazanç, K_i integral kazançtır, $e(t)$ geçerli zamandaki hatadır t ve τ entegrasyon değişkenidir.

İntegral zaman 0 dan şimdiki zaman t ye kadar integral alır. Entegrasyon için τ kullanırız çünkü integral boyunca birden fazla değeri alacak bir değişkene ihtiyacımız vardır, ancak şunu kullanamayız t çünkü bunu şimdiki zaman olarak tanımlamıştık.

Aşağıdaki şekil, bir PI kontrolörü tarafından kontrol edilen bir *system* için bir blok diyagramı gösterir.

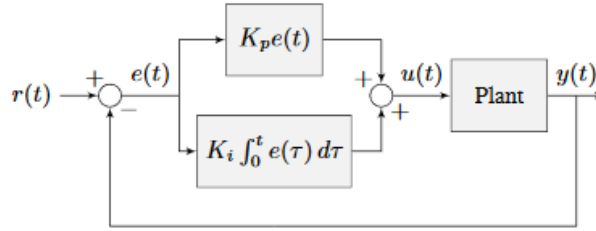


Figure 2.3: PI controller block diagram

system kararlı durumda *setpoint* i kapattığında, oransal terim *output* u **terim: `setpoint`** na çekmek için çok küçük olabilir, ve türev terimi sıfırdır. Bu, şekil 2.4'te gösterildiği gibi *steady-state error* ile sonuçlanabilir.

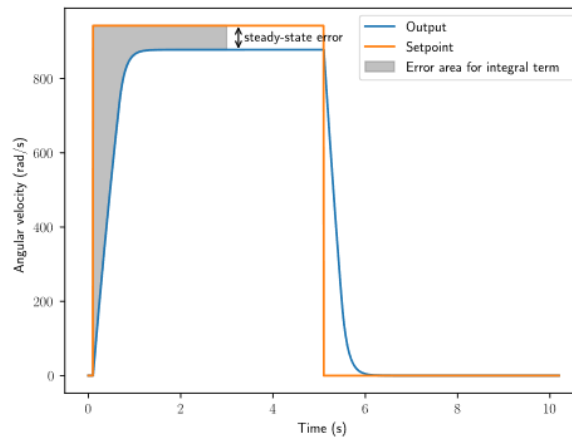


Figure 2.4: P controller with steady-state error

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.

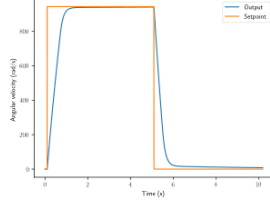


Figure 2.5: PI controller without steady-state error

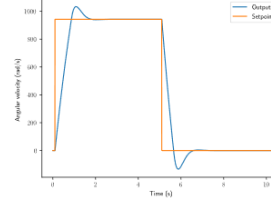


Figure 2.6: PI controller with overshoot from large K_i gain

Putting It All Together

Not: WPILib tarafından sağlanan PIDController'ı kullanma hakkında bilgi için bkz [ilgili makale](#).

When these terms are combined by summing them all together, one gets the typical definition for a PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt}$$

burada K_p oransal kazançtır, K_i integral kazançtır, K_d türevsel kazançtır, $e(t)$ şu andaki hatadır t ve τ entegrasyon değişkenidir.

Aşağıdaki şekil, bir PID kontrolörü için bir blok diyagramı göstermektedir.

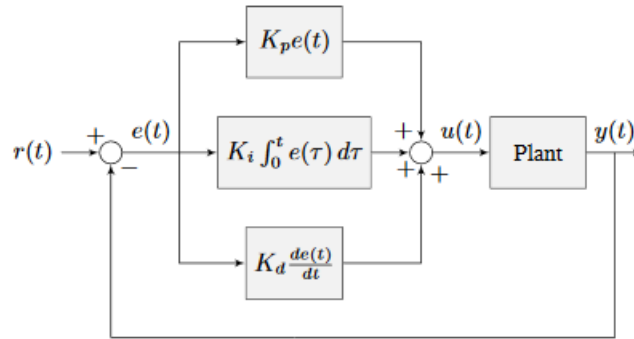


Figure 2.7: PID controller block diagram

Yanıt Türleri

PID denetleyicisi tarafından çalıştırılan *system* genellikle üç tür yanıt içerir: yetersiz sönümlü, aşırı sönümlü ve kritik sönümlü. Bunlar şekil 2.8’de gösterilmektedir.

Resim 2.7 de *step responses* için; *rise time*, *system`in` :term:`step input`u uyguladıktan sonra referansa ulaşmaya başlaması için gereken zamandır. :term:`Settling time`:* *system`in` :term:`step input`u uyguladıktan sonra :term:`reference`a oturmaya başlaması için gereken zamandır.*

Yetersiz sönümlenmiş-underdamped bir yanıt, oturmadan önce *reference* etrafında salınır. *Aşırı sönümlenmiş-overdamped* bir yanıt

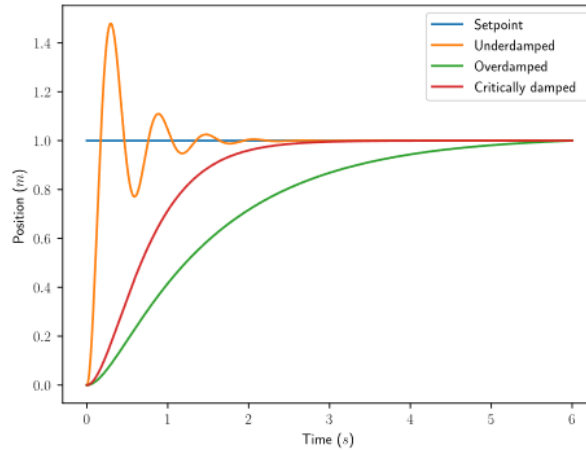


Figure 2.8: PID controller response types

yavaş yükselir ve *reference* yi aşmaz. *Kritik olarak sönümlenen-critically damped* bir yanıt, *reference* öğesini aşmadan en hızlı *rise time* a sahiptir.

32.2.5 WPI Tarafından Hazırlanan PID’ye Giriş Videosu

Hızlı hareket eden ve tam olarak istenen bir konumda duran bir robot sistemi tasarlarken hiç sorun yaşadınız mı? Sabit mesafeler veya hızlarda sürerken, bir kolu veya asansörü çalıştırırken, veya belirli hareket gerektiren başka bir motor kontrollü sistemi kullanırken zorluklar ortaya çıkabilir. Bu videoda, WPI Profesörü Dmitry Berenson robot kontrollerinden ve PID kontrollerinin nasıl çalıştığından bahsediyor.

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

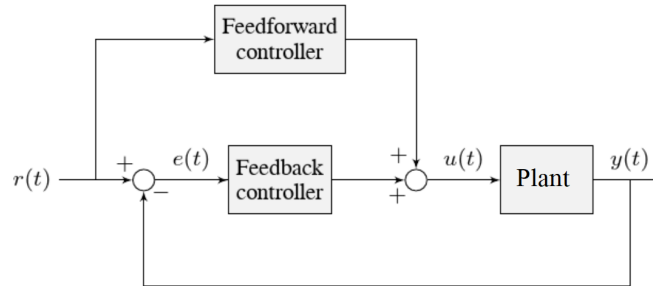
In this section, we will tune a simple velocity controller for a flywheel. The tuning principles explained here will also work for almost any velocity control scenario.

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 WPI-Lib’s *SimpleMotorFeedforward*, with additional adjustment for sensor delay and gearbox inefficiency. The simulation assumes the plant is controlled by feedforward and feedback controllers, composed in this fashion:



Where:

- The plant’s *output* $y(t)$ is the flywheel rotational velocity
- The controller’s *setpoint* $r(t)$ is the desired velocity of the flywheel
- The controller’s *control effort*, $u(t)$ is the voltage applied to the motor driving the flywheel’s motion

Not: 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.

Not: This is fundamentally different from the *vertical arm* and *turret* controllers, which both control *position*.

¹ For this simulation, we model a ball being injected to the flywheel as a velocity-dependant (frictional) torque fighting the spinning of the wheel for one quarter of a wheel rotation, right around the 5 second mark. This is a very simplistic way to model the ball, but is sufficient to illustrate the controller’s behavior under a sudden load. It would not be sufficient to predict the ball’s trajectory, or the actual “pulldown” in *output* for the system.

The tutorials below will demonstrate the behavior of the system under bang-bang, pure feed-forward, pure feedback (PID), and combined feedforward-feedback control strategies. Follow the instructions to learn how to manually tune these controllers, and expand the “tuning solution” to view an optimal model-based set of tuning parameters.

Bang-Bang Control

Interact with the simulation below to see how the flywheel system responds when controlled by a bang-bang controller.

The “Bang-Bang” controller is a simple controller which applies a binary (present/not-present) force to a mechanism to try to get it closer to a setpoint. A more detailed description (and documentation for the corresponding WPILib implementation) can be found [here](#).

There are no tuneable controller parameters for a bang-bang controller - you can only adjust the setpoint. This simplicity is a strength, and also a weakness.

Try adjusting the setpoint up and down. You should see that for almost all values, the output converges to be somewhat near the setpoint.

Common Issues with Bang-Bang Controllers

Note that the system behavior is not perfect, because of delays in the control loop. These can result from the nature of the sensors, measurement filters, loop iteration timers, or even delays in the control hardware itself. Collectively, these cause a cycle of “overshoot” and “undershoot”, as the output repeatedly goes above and below the setpoint. This oscillation is unavoidable with a bang-bang controller.

Typically, the steady-state oscillation of a bang-bang controller is small enough that it performs quite well in practice. However, rapid on/off cycling of the control effort can cause mechanical issues - the cycles of rapidly applying and removing forces can loosen bolts and joints, and put a lot of stress on gearboxes.

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*.

Not: PID-only control is not a very good control scheme for flywheel velocity! Do not be surprised if/when the simulation below does not behave well, even when the "optimal" constants are used.

In this particular example, for a setpoint of 300, values of $K_p = 0.1$, $K_i = 0.0$, and $K_d = 0.0$ will produce somewhat reasonable results. Since this control strategy is not very good, it will not work well for all setpoints. You can attempt to improve this behavior by incorporating some K_i , but it is very difficult to achieve good behavior across a wide range of setpoints.

Issues with Feedback Control Alone

Because a non-zero amount of *control effort* is required to keep the flywheel spinning, even when the *output* and *setpoint* are equal, this feedback-only strategy is flawed. In order to optimally control a flywheel, a combined feedforward-feedback strategy is needed.

Combined Feedforward and Feedback Control

Interact with the simulation below to see how the flywheel system responds under simultaneous feedforward and feedback (PID) control.

Tuning the combined flywheel controller is simple - we first tune the feedforward controller following the same procedure as in the feedforward-only section, and then we tune the PID controller following the same procedure as in the feedback-only section. Notice that PID portion of the controller is *much* easier to tune "on top of" an accurate feedforward.

In this particular example, for a setpoint of 300, values of $K_v = 0.0075$ and $K_p = 0.1$ will produce very good results across all setpoints. Small changes to K_p will change the controller behavior to be more or less aggressive - the optimal choice depends on your problem constraints.

Note that the combined feedforward-feedback controller works well across all setpoints, and recovers very quickly after the external disturbance of the ball contacting the flywheel.

Tuning Conclusions

Applicability of Velocity Control

A gamepiece-launching flywheel is one of the most visible applications of velocity control. It is also applicable to drivetrain control - following a pre-defined path in autonomous involves controlling the velocity of the wheels with precision, under a variety of different loads.

Choice of Control Strategies

Because we are controlling velocity, we can achieve fairly good performance with a *pure feedforward controller*. This is because a permanent-magnet DC motor's steady-state velocity is roughly proportional to the voltage applied, and is the reason that you can drive your robot around with joysticks without appearing to use any control loop at all - in that case, you are implicitly using a proportional feedforward model.

Because we must apply a constant control voltage to the motor to maintain a velocity at the setpoint, we cannot successfully use a *pure feedback (PID) controller* (whose output typically disappears when you reach the setpoint) - in order to effectively control velocity, a feedback controller must be *combined with a feedforward controller*.

Bang-bang control can be combined with feedforward control much in the way PID control can - for the sake of brevity we do not include a combined feedforward-bang-bang simulation.

Tuning with only feedback can produce reasonable results in cases where no *control effort* is required to keep the *output* at the *setpoint*. This may work for mechanisms like turrets, or swerve drive steering. However, as seen above, it does not work well for a flywheel, where the back-EMF and friction both act to slow the motor even when it is sustaining motion at the setpoint. To control this system, we need to combine the PID controller with a feedforward controller.

K_d is not useful for velocity control with a constant setpoint - it is only necessary when the setpoint is changing.

Adding an integral gain to the *controller* is often a sub-optimal way to eliminate *steady-state error* - you can see how sloppy and "laggy" it is in the simulation above! As we will see soon, a better approach is to combine the PID controller with a feedforward controller.

Velocity and Position Control

Velocity control also differs from position control in the effect of inertia - in a position controller, inertia tends to cause the mechanism to swing past the setpoint even if the control voltage drops to zero near the setpoint. This makes aggressive control strategies infeasible, as they end up wasting lots of energy fighting self-induced oscillations. In a velocity controller, however, the effect is different - the rotor shaft stops accelerating as soon as you stop applying a control voltage (in fact, it will slow down due to friction and back-EMF), so such overshoots are rare (in fact, overshoot typically occurs in velocity controllers only as a result of loop delay). This enables the use of an extremely simple, extremely aggressive control strategy called *bang-bang control*.

Feedforward Simplifications

For the sake of simplicity, the simulations above omit the K_s term from the WPILib SimpleMotorFeedforward equation. On actual mechanisms, however, this can be important - especially if there's a lot of friction in the mechanism gearing. A flywheel with a lot of static friction will not have a linear control voltage-velocity relationship unless the feedforward controller includes a K_s term to cancel it out.

To measure K_s manually, slowly increase the voltage to the mechanism until it starts to move. The value of K_s is the largest voltage applied before the mechanism begins to move.

Additionally, there is no need for a K_a term in the feedforward for velocity control unless the setpoint is changing - for a flywheel, this is not a concern, and so the gain is omitted here.

Footnotes

32.2.8 Tuning a Turret Position Controller

In this section, we will tune a simple position controller for a turret. The tuning principles explained here will also work for almost any position-control scenarios under no external loading.

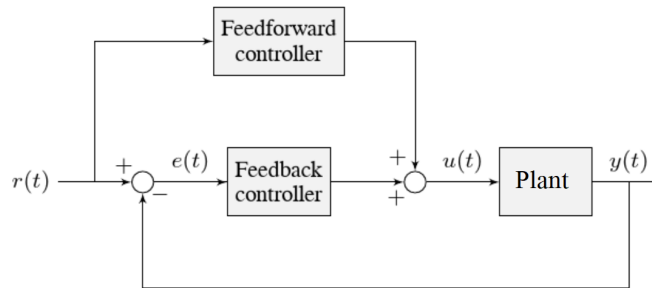
Turret Model Description

A turret rotates some mechanism side-to-side to position it for scoring gamepieces.

Our “turret” consists of:

- A rotating inertial mass (the turret)
- A motor and gearbox driving the mass

For the purposes of this tutorial, this plant is modeled with the same equation used by WPILib's *SimpleMotorFeedforward*, with additional adjustment for sensor delay and gearbox inefficiency. The simulation assumes the plant is controlled by feedforward and feedback controllers, composed in this fashion:



Where:

- The plant's *output* $y(t)$ is the turret's position
- The controller's *setpoint* $r(t)$ is the desired position of the turret
- The controller's *control effort*, $u(t)$ is the voltage applied to the motor driving the turret

Picking the Control Strategy for a Turret Position Controller

In general: the more voltage that is applied to the motor, the faster the motor (and turret) will spin. Once voltage is removed, friction and back-EMF slowly decrease the spinning until the turret stops. We want to make the turret rotate to a given position.

The tutorials below will demonstrate the behavior of the system under pure feedforward, pure feedback (PID), and combined feedforward-feedback control strategies. Follow the instructions to learn how to manually tune these controllers, and expand the “tuning solution” to view an optimal model-based set of tuning parameters. Even though WPILib tooling can provide you with optimal gains, it is worth going through the manual tuning process to see how the different control strategies interact with the mechanism.

This simulation does not include any motion profile generation, so acceleration setpoints are not very well-defined. Accordingly, the kA term of the feedforward equation is not used by the controller. This means there will be some amount of delay/lag inherent to the feedforward-only response.

Pure Feedforward Control

Interact with the simulation below to examine how the turret system responds when controlled only by a feedforward controller.

Not: 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.

Not: Feedforward-only control is not a viable control scheme for turrets! Do not be surprised if/when the simulation below does not behave well, even when the “correct” constants are used.

The exact gain used by the plant is $K_v = 0.2$. Note that due to timing inaccuracy in browser simulations, the K_v that works best in the simulation may be somewhat smaller than this.

Issues with Feed-Forward Control Alone

As mentioned above, our simulated mechanism perfectly obeys the WPILib *SimpleMotorFeedforward* equation (as long as the “system noise” option is disabled). We might then expect, like in the case of the *flywheel velocity controller*, that we should be able to achieve perfect convergence-to-setpoint with a feedforward loop alone.

However, our feedforward equation relates *velocity* and *acceleration* to voltage - it allows us to control the *instantaneous motion* of our mechanism with high accuracy, but it does not allow us direct control over the *position*. This is a problem even in our simulation (in which the feedforward equation is the *actual* equation of motion), because unless we employ a *motion profile* to generate a sequence of velocity setpoints we can ask the turret to jump immediately from one position to another. This is impossible, even for our simulated turret.

The resulting behavior from the feedforward controller is to output a single “voltage spike” when the position setpoint changes (corresponding to a single loop iteration of very high velocity), and then zero voltage (because it is assumed that the system has already reached the setpoint). In practice, we can see in the simulation that this results in an initial “impulse” movement towards the target position, that stops at some indeterminate position in-between. This kind of response is called a “kick,” and is generally seen as undesirable.

You may notice that *smooth* motion below the turret’s maximum achievable speed can be followed accurately in the simulation with feedforward alone. This is misleading, however, because no real mechanism perfectly obeys its feedforward equation. With the “system noise” option enabled, we can see that even smooth, slow motion eventually results in compounding position errors when only feedforward control is used. To accurately converge to the setpoint, we need to use a feedback (PID) controller.

Pure Feedback Control

Interact with the simulation below to examine how the turret system responds when controlled only by a feedback (PID) controller.

Perform the following:

1. Set K_p , K_i , K_d , and K_v to zero.
2. Increase K_p until the mechanism responds to a sudden change in setpoint by moving sharply to the new position. If the controller oscillates too much around the setpoint, reduce K_p until it stops.
3. Increase K_d to reduce the amount of “lag” when the controller tries to track a smoothly moving setpoint (reminder: click and drag the turret’s directional indicator to move it smoothly). If the controller starts to oscillate, reduce K_d until it stops.

Gains of $K_p = 0.3$ and $K_d = 0.05$ yield rapid and stable convergence to the setpoint. Other, similar gains will work nearly as well.

Issues with Feedback Control Alone

Note that even with system noise enabled, the feedback controller is able to drive the turret to the setpoint in a stable manner over time. However, it may not be possible to smoothly track a moving setpoint without lag using feedback alone, as the feedback controller can only respond to errors once they have built up. To get the best of both worlds, we need to combine our feedback controller with a feedforward controller.

Combined Feedforward and Feedback Control

Interact with the simulation below to examine how the turret system responds under simultaneous feedforward and feedback control.

Tuning the combined turret controller is simple - we first tune the feedforward controller following the same procedure as in the feedforward-only section, and then we tune the PID controller following the same procedure as in the feedback-only section. Notice that PID portion of the controller is *much* easier to tune “on top of” an accurate feedforward.

The optimal gains for the combined controller are just the optimal gains for the individual controllers: gains of $K_v = 0.15$, $K_p = 0.3$, and $K_d = 0.05$ yield rapid and stable convergence to the setpoint and relatively accurate tracking of smooth motion. Other, similar gains will work nearly as well.

Once tuned properly, the combined controller should accurately track a smoothly moving setpoint, and also accurately converge to the setpoint over time after a “jump” command.

Tuning Conclusions

Choice of Control Strategies

Like in the case of the *vertical arm*, and unlike the case of the *flywheel*, we are trying to control the *position* rather than the *velocity* of our mechanism.

In the case of the flywheel *velocity* controller we could achieve good control performance with feedforward alone. However, it is very hard to predict how much voltage will cause a certain total change in *position* (time can turn even small errors in velocity into very big errors in position). In this case, we cannot rely on feedforward control alone - as with the vertical arm, we will need a feedback controller.

Unlike in the case of the vertical arm, though, there is no voltage required to keep the mechanism at the setpoint once it's there. As a consequence, it is often possible to effectively control a turret without any feedforward controller at all, relying only on the output of the feedback controller (if the mechanism has a lot of friction, this may not work well and both a feedforward and feedback controller may be needed). Simple position control in the absence of external forces is one of the only cases in which pure feedback control works well.

Controlling a mechanism with only feedback can produce reasonable results in cases where no *control effort* is required to keep the *output* at the *setpoint*. On a turret, this can work acceptably - however, it may still run into problems when trying to follow a moving setpoint, as it relies entirely on the controller transients to control the mechanism's intermediate motion between position setpoints.

We saw in the feedforward-only example above that an accurate feedforward can track slow, smooth velocity setpoints quite well. Combining a feedforward controller with the feedback

controller gives the smooth velocity-following of a feedforward controller with the stable long-term error elimination of a feedback controller.

Reasons for Non-Ideal Performance

This simulation does not include any motion profile generation, so acceleration setpoints are not very well-defined. Accordingly, the kA term of the feedforward equation is not used by the controller. This means there will be some amount of delay/lag inherent to the feedforward-only response.

A Note on Feedforward and Static Friction

For the sake of simplicity, the simulations above omit the K_s term from the WPILib SimpleMotorFeedforward equation. On actual mechanisms, however, this can be important - especially if there's a lot of friction in the mechanism gearing. A turret with a lot of static friction will be very hard to control accurately with feedback alone - it will get "stuck" near (but not at) the setpoint when the loop output falls below K_s .

To measure K_s manually, slowly increase the voltage to the mechanism until it starts to move. The value of K_s is the largest voltage applied before the mechanism begins to move.

It can be mildly difficult to *apply* the measured K_s to a position controller without motion profiling, as the WPILib SimpleMotorFeedforward class uses the velocity setpoint to determine the direction in which the K_s term should point. To overcome this, either use a motion profile, or else add K_s manually to the output of the controller depending on which direction the mechanism needs to move to get to the setpoint.

32.2.9 Dikey Kol Pozisyon Kontrolörünü Ayarlama

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.

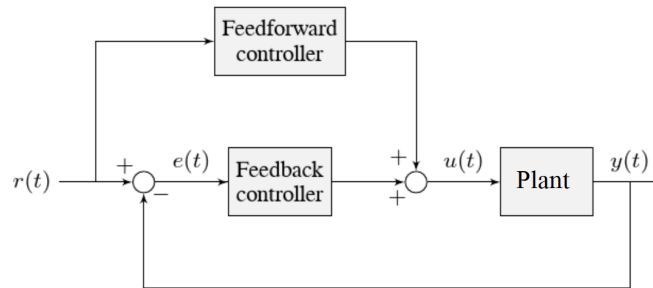
Arm Model Description

Vertical arms are commonly used to lift gamepieces from the ground up to a scoring position. Other similar examples include shooter hoods and elevators.

"Dikey kolumuz" şunlardan oluşur:

- 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 [ArmFeedforward](#), with additional adjustment for sensor delay and gearbox inefficiency. The simulation assumes the plant is controlled by feedforward and feedback controllers, composed in this fashion:



Where:

- The plant's *output* $y(t)$ is the arm's rotational position
- The controller's *setpoint* $r(t)$ is the desired angle of the arm
- The controller's *control effort*, $u(t)$ is the voltage applied to the motor driving the arm

Picking the Control Strategy for a Vertical Arm

Applying voltage to the motor causes a force on the mechanism that drives the arm up or down. If there is no voltage, gravity still acts on the arm to pull it downward. Generally, it is desirable to fight this effect, and keep the arm at a specific angle.

The tutorials below will demonstrate the behavior of the system under pure feedforward, pure feedback (PID), and combined feedforward-feedback control strategies. Follow the instructions to learn how to manually tune these controllers, and expand the “tuning solution” to view an optimal model-based set of tuning parameters. Even though WPILib tooling can provide you with optimal gains, it is worth going through the manual tuning process to see how the different control strategies interact with the mechanism.

Pure Feedforward Control

Interact with the simulation below to examine how the turret system responds when controlled only by a feedforward controller.

Not: 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.

Not: 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 *ArmFeed-forward* 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.

Not: 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 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.

Önemli: 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.

Aktüatör Doygunluğu

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 Filtreler

Not: Bu bölümde çeşitli gösterge grafiklerini oluşturmak için kullanılan veriler bulunabilir: indir: [buradan <resources/filterdemo.csv>](#).

Bu bölüm, WPILib ile birlikte gürültü azaltma ve/veya girdinin istikrarlı hale getirilmesi için yararlı olan bir dizi filtreyi açıklamaktadır.

32.3.1 Filtrelere Giriş

Filtreler, modern teknolojiye kullanılan en yaygın araçlardan bazılarıdır ve hem sinyal işlemede hem de kontrollerde robotikte çok sayıda uygulama bulur. Bir filtre kavramını anlamak, WPILib tarafından sağlanan çeşitli filtre türlerinin faydasını anlamak için çok önemlidir.

Filtre Nedir?

Not: Bu makalenin iyiliği için, tüm verilerin tek boyutlu zaman serisi verileri olduğunu varsayacağız. Açıkçası, ilgili kavramlar bundan daha geneldir - ancak sinyallerin ve filtrelemenin tam / titiz bir tartışması bu belgenin kapsamı dışındadır.

Öyleyse, tam olarak filtre *nedir* ? Basitçe ifade etmek gerekirse, bir filtre, bir giriş akışından bir çıkış akışına eşlemedir. Diğer bir deyişle, bir filtreden (prensipte olarak) çıkan değer, yalnızca girdinin *mevcut* değerine değil, aynı zamanda *tüm geçmiş ve gelecekteki değerler kümesine* (tabii ki pratikte filtreler, WPILib tarafından sağlananlar gerçek zamanlı olarak akış verilerinde uygulanabilir; buna göre, bunlar yalnızca girdinin *geçmiş* değerlerine bağlı olabilir ve gelecekteki değerlere bağlı değildir) bağlı olabilir. Bu önemli bir kavramdır, çünkü genellikle bir sinyalden istenmeyen *dinamikleri* kaldırmak / azaltmak için filtreler kullanırız. Bir sinyali filtrelediğimizde, *sinyalin zaman içinde nasıl değiştiğini* kontrol etmekle ilgileniyoruz.

Filtre Kullanmanın Etkileri

Gürültü Azaltma

Bir filtrenin en tipik kullanımlarından biri gürültüyü azaltmaktır. Gürültüyü azaltan bir filtreye *alçak-geçiren* filtre denir (çünkü yüksek frekansları bloke ederken düşük frekansların “geçmesine” izin verir). Şu anda WPILib’de bulunan filtrelerin çoğu, etkili bir şekilde alçak geçiren filtrelerdir.

Oran Sınırlama

Filtreler ayrıca bir sinyalin değişme hızını azaltmak için de yaygın olarak kullanılır. Bu, gürültünün azaltılmasıyla yakından ilgilidir ve gürültüyü azaltan filtreler de çıktıların değişim oranını sınırlama eğilimindedir.

Kenar algılama

Alçak geçiren filtrenin karşılığı, yalnızca yüksek frekansların çıkışa geçmesine izin veren yüksek geçiren filtredir. Yüksek geçiren filtrelerin sezgileri oluşturmak için biraz zor olabilir, ancak yüksek geçiren bir filtrenin yaygın bir kullanımı kenar algılamadır - yüksek geçiren filtreler, daha yavaş değişiklikleri göz ardı ederken girdideki ani değişiklikleri yansıtacağından, sinyaldeki keskin süreksizliklerin yerini belirlemek için yararlıdırlar.

Faz gecikmesi

Gerçek zamanlı bir düşük geçiş filtresinin kaçınılmaz bir olumsuz etkisi, “faz gecikmesi” nin ortaya çıkmasıdır. Daha önce belirtildiği gibi, gerçek zamanlı bir filtre yalnızca sinyalin geçmiş değerlerine bağlı olabileceğinden (gelecekteki değerleri elde etmek için zamanda yolculuk yapamayız), filtrelenen değer giriş değişmeye başladığında “yakalanması” biraz zaman alır. Gürültü azaltma ne kadar büyükse, eklenen gecikme o kadar büyük olur. Bu, birçok yönden, gerçek zamanlı filtrelemenin *temel* tan etkisidir ve filtre tasarımınızın birincil itici faktörü olmalıdır.

İlginç bir şekilde, yüksek geçişli filtreler, giriş değerindeki yerel değişiklikleri şiddetlendirdiklerinden, bir faz gecikmesinin aksine bir faz *öncüsü* sunar.

32.3.2 Doğrusal Filtreler

WPILib’in desteklediği ilk (ve en yaygın kullanılan) filtre türü, *doğrusal bir filtre* veya daha spesifik olarak, doğrusal zamanla değişmeyen-linear time-invariant (LTI) filtredir.

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*.

impuls yanıtlarının iki geniş “sorts-türü” vardır: sonsuz dürtü yanıtları (IIR) ve sonlu dürtü yanıtları (FIR).

Sonsuz dürtü tepkilerinin sonsuz “desteği” vardır - yani, sonsuz büyüklükte bir bölgede sıfırdan farklıdırlar. Bu, genel olarak, sonsuz “hafızaya” sahip oldukları anlamına gelir - giriş akışında bir değer görüldüğünde, *sonraki tüm çıktıları sonsuza kadar* etkileyecektir. Bu, katı bir sinyal işleme perspektifinden tipik olarak istenmeyen bir durumdur, ancak sonsuz dürtü yanıtlarına sahip filtrelerin, basit özyineleme ilişkileri ile ifade edilebildikleri için hesaplanması çok kolay olma eğilimindedir.

Sonlu dürtü yanıtlarının sonlu “desteği” vardır - yani, sınırlı bir bölgede sıfırdan farklıdırlar. “Arketipik” FIR filtresi düz hareketli bir ortalamadır - yani, basitçe çıktıyı son n girdinin ortalamasına eşit olarak ayarlamaktır. FIR filtreleri, IIR filtrelerinden daha çok istenen özelliklere sahip olma eğilimindedir, ancak hesaplamaları daha maliyetlidir.

Linear filters are supported in WPILib through the `LinearFilter` class (Java, C++, , Python).

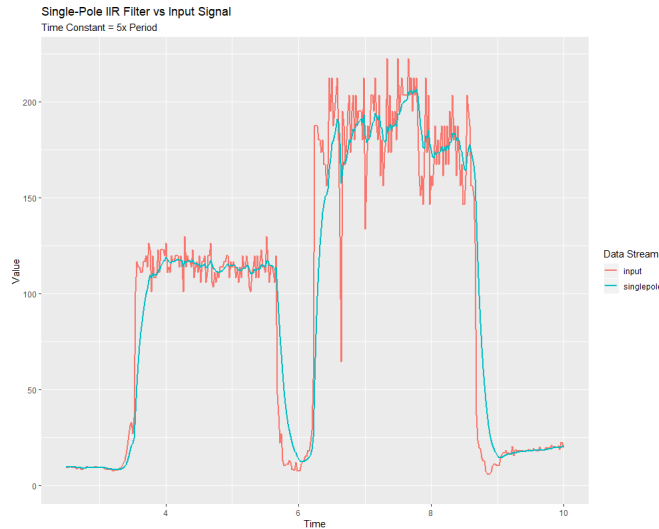
LinearFilter - Doğrusal Filtre Oluşturma

Not: C ++ `LinearFilter` sınıfı, girdi için kullanılan veri türüne göre şablonlanır.

Not: Filtrelerin “belleği” olduğundan, her giriş akışı kendi filtre nesnesini gerektirir. Birden çok giriş akışı için aynı filtre nesnesini kullanmaya * çalışmayın *.

Özel bir filtre oluşturmak için `` `LinearFilter` `` sınıfını doğrudan somutlaştırmak mümkün olsa da, bunun yerine sağlanan fabrika yöntemlerinden birini kullanmak çok daha uygundur (ve yaygındır):

singlePoleIIR - tek Kutuplu 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

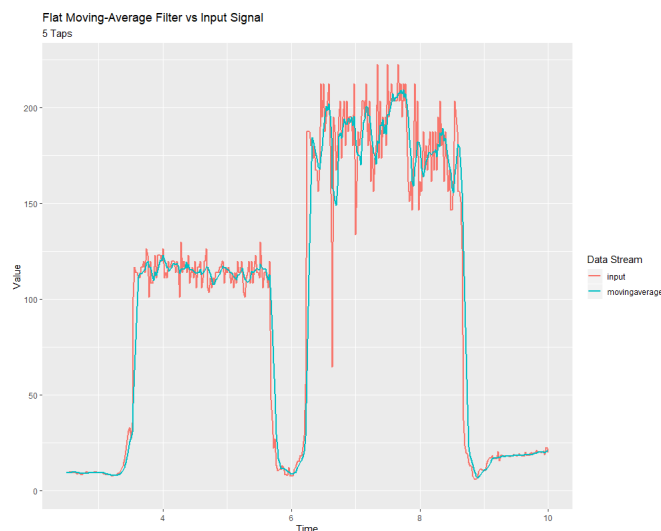
```
from wpimath.filter import LinearFilter

# 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)
```

"time constant- zaman sabiti" parametresi, filtrenin dürtü yanıtının "karakteristik zaman ölçeğini" belirler; filtre, bundan önemli ölçüde daha kısa zaman ölçeklerinde meydana gelen sinyal dinamiklerini iptal edecektir. Benzer şekilde, aynı zamanda tanımlanan yaklaşık zaman ölçeğidir: ref: "faz gecikmesi <docs/software/advanced-controls/filters/introduction:Phase Lag>". 2π ile çarpılan bu zaman ölçeğinin tersi, filtrenin "cutoff frequency-kesme frekansı" dır.

"period - dönem" parametresi, filtrenin ``calculate()`` yönteminin çağrılacağı dönemdir. Uygulamaların büyük çoğunluğu için bu, 0,02 saniyelik standart ana robot döngüsü süresi olacaktır. -

movingAverage - hareketliOrtalama



movingAverage fabrika yöntemi, basit bir düz hareketli ortalama filtre oluşturur. Bu, mümkün olan en basit düşük geçişli FIR filtresidir ve tek kutuplu IIR filtresiyle aynı bağlamların çoğunda kullanışlıdır. Hesaplamak biraz daha maliyetlidir, ancak genellikle biraz daha hoş bir şekilde davranır.

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);
```

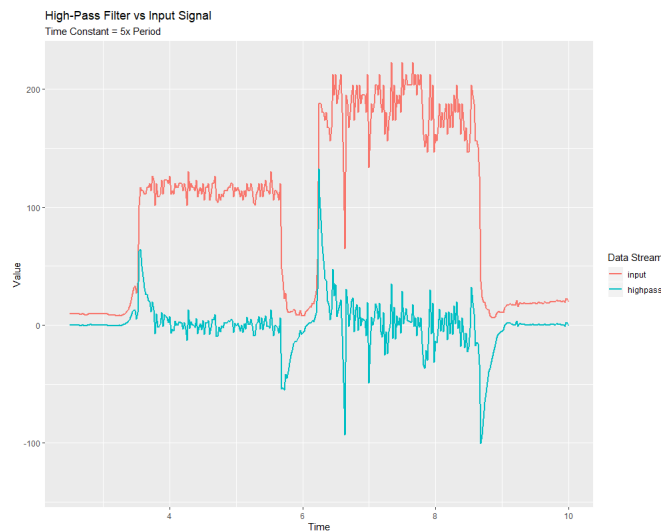
PYTHON

```
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)
```

The “taps” parameter is the number of samples that will be included in the flat moving average. This behaves similarly to the “time constant” above - the effective time constant is the number of taps times the period at which `calculate()` is called.

highPass-yüksekGeçiş



highPass-yüksekGeçiş fabrika yöntemi, basit bir birinci dereceden sonsuz dürtü tepkisi yüksek geçiren filtre oluşturur. Bu, “singlePoleIIR” nin “karşılığıdır”.

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)
```

“time constant-zaman sabiti” parametresi, filtrenin dürtü yanıtının “characteristic timescale-karakteristik zaman ölçeğini” belirler; filtre, bundan önemli ölçüde daha uzun zaman ölçeklerinde meydana gelen sinyal dinamiklerini iptal edecektir. Benzer şekilde, bu aynı zamanda tanımlanan yaklaşık zaman ölçeğidir: ref: “faz kurşun <docs/software/advanced-controls/filters/introduction:Phase lag>”. 2 pi ile çarpılan bu zaman ölçeğinin tersi, filtrenin “cutoff frequency-kesme frekansı” dır.

“period - dönem” parametresi, filtrenin ``calculate()`` yönteminin çağrılacağı dönemdir. Uygulamaların büyük çoğunluğu için bu, 0,02 saniyelik standart ana robot döngüsü süresi olacaktır. -

LinearFilter kullanma

Not: Oluşturulan filtrenin belirtilen zaman ölçeği parametresine uyması için, ``calculate()`` işlevinin * belirtilen sürede düzenli olarak çağrılması gerekir *. Herhangi bir nedenle ``hesapla()`` çağrılarında önemli bir hata olması gerekiyorsa, filtrenin ``reset()`` yöntemi daha fazla kullanılmadan önce çağrılmalıdır.

Filtreniz oluşturulduktan sonra kullanımı kolaydır - filtrelenmiş çıktıyı elde etmek için en son girdiyle calculate() yöntemini çağırmanız yeterlidir:

JAVA

```
// Calculates the next value of the output  
filter.calculate(input);
```

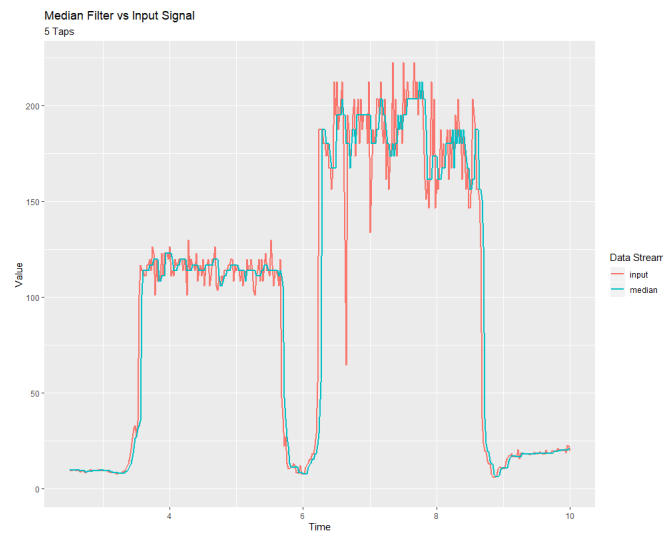
C++

```
// Calculates the next value of the output  
filter.Calculate(input);
```

PYTHON

```
# Calculates the next value of the output  
filter.calculate(input)
```

32.3.3 Median Filtresi



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.

Median filtresi, bir giriş akışından ara sıra aykırı değerleri kaldırmak için çok kullanışlıdır. Bu, ara sıra parazitlere eğilimli olan mesafe sensörlerinden gelen girdileri filtrelemek için özellikle uygun hale getirir. Hareketli bir ortalamanın aksine; median filtresi, ne kadar aşırı olursa olsun, az sayıdaki aykırı değerlerden tamamen etkilenmeyecektir.

The median filter is supported in WPILib through the MedianFilter class (Java, C++, , Python).

MedianFilter Oluşturma

Not: C++ MedianFilter sınıfı, girdi için kullanılan veri türüne göre şablonlanır.

Not: Filtrelerin “belleği” olduğundan, her giriş akışı kendi filtre nesnesini gerektirir. Birden çok giriş akışı için aynı filtre nesnesini kullanmaya *çalışmayın*.

Bir MedianFilter oluşturmak basittir:

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)
```

MedianFilter kullanma

Filtreniz oluşturulduktan sonra kullanımı kolaydır - filtrelenmiş çıktıyı elde etmek için en son girdiyle calculate() yöntemini çağırmanız yeterlidir:

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 Dönme Hızı Sınırlayıcı

FRC ® içindeki filtreler için yaygın bir kullanım kontrol girişlerinin davranışını yumuşatmak içindir (örneğin, sürücü kontrollerinizden joystick girişleri). Ne yazık ki, basit bir alçak geçiren filtre bu iş için pek uygun değildir; Düşük geçişli bir filtre, bir giriş akışının ani değişikliklere tepkisini yumuşatırken, aynı zamanda ince kontrol detayını temizler ve faz gecikmesine neden olur. Daha iyi bir çözüm, doğrudan kontrol girişinin değişim oranını sınırlamaktır. Bu, bir *slew rate limiter*-dönüş hızı sınırlayıcı - sinyalin maksimum değişim hızını sınırlayan bir filtre ile gerçekleştirilir.

WPILibWPILibBir dönüş hızı sınırlayıcı, bir tür ilkel hareket profili olarak düşünülebilir. Aslında, dönüş hızı sınırlayıcı, WPILib tarafından desteklenen: *Trapezoidal Motion Profile* ın birinci dereceden eşdeğeridir - ivme kısıtlamasının sonsuza eğilimli olmasına izin verildiğinde, tam da yamuk hareketin sınırlayıcı durumudur. Buna göre, dönüş hızı sınırlayıcı, bir hız ayar noktası akımına (veya genellikle hız ile yaklaşık olarak orantılı olan voltajlara) fiili bir hareket profili uygulamak için iyi bir seçimdir. Konumları kontrol eden giriş akışları için uygun bir trapezoidal profil kullanmak genellikle daha iyidir.

Slew rate limiting is supported in WPILib through the `SlewRateLimiter` class (Java, C++, Python).

Bir SlewRate Limiter oluşturma

Not: C++ ``SlewRateLimiter`` sınıfı, girdinin birim türüne göre şablonlanır. C++ birimleri hakkında daha fazla bilgi için bkz: ref: *docs / software / basic-programlama / cpp-units: C++ Units Library*.

Not: Filtrelerin “belleği” olduğundan, her giriş akışı kendi filtre nesnesini gerektirir. Birden çok giriş akışı için aynı filtre nesnesini kullanmaya çalışmayın.

Bir SlewRateLimiter oluşturmak basittir:

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)
```

Bir SlewRateLimiter' ı kullanma

Filtreniz oluşturulduktan sonra kullanımı kolaydır - filtrelenmiş çıktıyı elde etmek için en son girdiyle `` hesapla () `` yöntemini çağırmanız yeterlidir:

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

Not: 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.
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
↪ read as a True signal.  
if self.debounce.calculate(self.input.get()):  
    # Do something now that the DI is True.  
    pass
```

32.4 Geometri Sınıfları

Bu bölüm, WPILib'in geometri sınıflarını kapsar.

32.4.1 Ötelenme, Dönme ve Poz

Ötelenme

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.

Pisagor teoremini kullanarak mesafeyi başka bir Translation2d'ye döndüren `getDistance(Translation2d other)` kullanarak başka bir Translation2d nesnesine olan mesafeyi elde edebilirsiniz.

Not: 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.

Dönme

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.

Not: 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.

Not: 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.

Poz

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 Dönüşümler

Translation2d

Bir Translation2d üzerindeki işlemler, Translation2d ile temsil edilen vektör üzerinde işlemler gerçekleştirir.

- 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.
- Döndürme: Bir Translation2d nin saat yönünün tersine dönüşle döndürülmesi θ , rotateBy kullanılarak orijin etrafında yapılabilir. Bu, vektörün matrisle çarpılmasına eşdeğerdir $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
- Additionally, you can rotate a Translation2d by 180 degrees by using unaryMinus in Java, or the unary - operator in C++/Python.

Rotation2d

Rotation2d için dönüşümler, Rotation2d ile temsil edilen açı ölçüsündeki aritmetik işlemlerdir.

- 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 ve 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, **göreceli-relative** bir dönüşümü temsil eder. Bir öteleme ve döndürme bileşenine sahiptir. Bir Pose2d nin bir Transform2d ile dönüştürülmesi, dönüşümün çeviri bileşenini pozun dönüşüyle döndürür ve ardından döndürülen çeviri bileşenini ve döndürme bileşenini poza ekler. Başka bir deyişle, Pose2d.plus(Transform2d) şunu döndürür

$$\begin{bmatrix} x_p \\ y_p \\ \theta_p \end{bmatrix} + \begin{bmatrix} \cos\theta_p & -\sin\theta_p & 0 \\ \sin\theta_p & \cos\theta_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

Twist2d, bir yay boyunca mesafedeki bir değişikliği temsil eder. Genellikle, bu sınıf, bir aktarma sisteminin hareketini temsil etmek için kullanılır; burada x bileşeni, sürülen ileri mesafedir, y bileşeni, yana sürülen mesafedir (sol pozitif) ve θ bileşeni, başlıktaki değişiklik. Poz üstelini bulmanın ardında yatan matematik (pozu bükülmenin eğriliği boyunca ileriye doğru hareket ettirdikten sonra yeni poz) 10. bölümde [burada](#) _ bulunabilir.

Not: Holonomik olmayan aktarma organları için, bir Twist2d nin y bileşeni her zaman 0 olmalıdır.

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

Not: 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:

$$V = kS \cdot \text{sgn}(\dot{d}) + kV \cdot \dot{d} + kA \cdot \ddot{d}$$

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:

$$V = kG + kS \cdot \text{sgn}(\dot{d}) + kV \cdot \dot{d} + kA \cdot \ddot{d}$$

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:

$$V = kG \cdot \cos(\theta) + kS \cdot \text{sgn}(\dot{\theta}) + kV \cdot \dot{\theta} + kA \cdot \ddot{\theta}$$

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.

Not: 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

Not: Some familiarity with your language’s units library is recommended and knowing how to use `Consumers` is required. This 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 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.

Not: 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.

Not: 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.

Uyari: 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.

Uyari: 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:

Not: A drivetrain routine is shown below, but the same motions will occur on any mechanism.

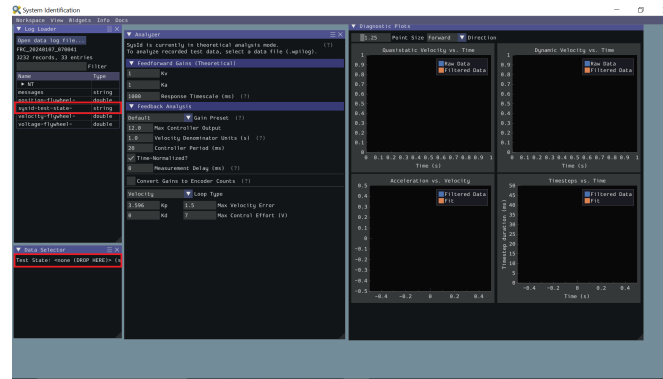
After all four tests have been completed, use the DataLogTool to retrieve the log file from the roboRIO.

32.5.4 Loading Data

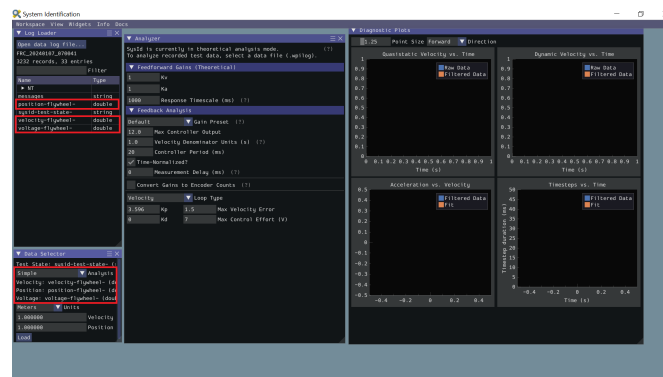
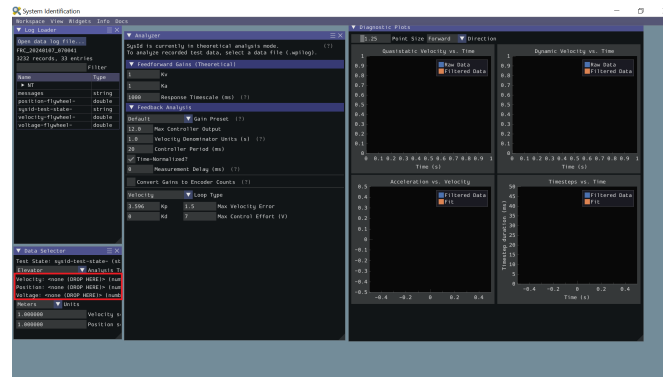
After downloading the WPILog containing the tests from the roboRIO, go to the Log Loader pane in SysId and click Open data log file....

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.

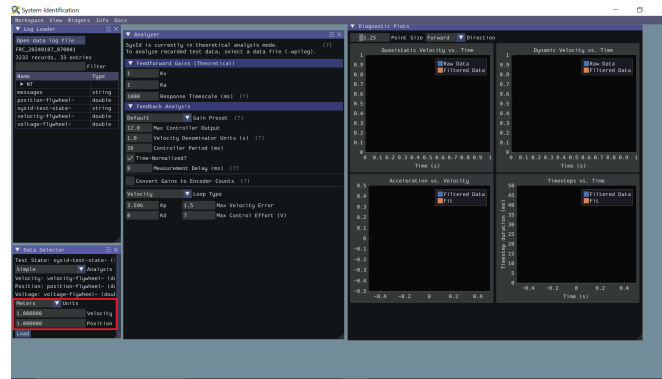
Not: SysIdRoutine will name the entry "sysid-test-state-mechanism", where "mechanism" is the name passed to the Mechanism constructor or the subsystem name.



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.

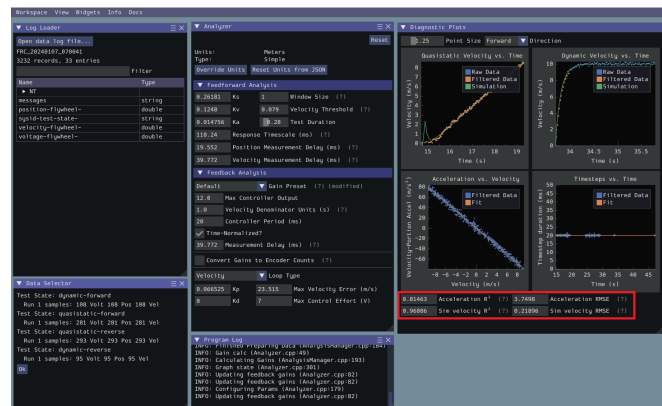


Ensure the correct analysis type has been selected, then click the Load button and move on to checking the fit diagnostics in the Diagnostics pane.

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*.



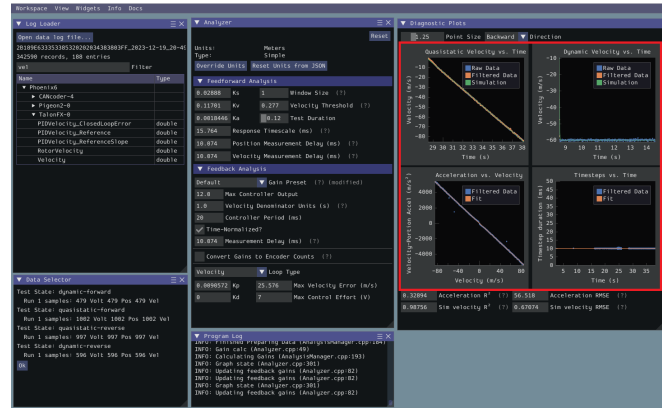
The acceleration *r-squared* is the fraction of the variance in measured acceleration (used as the independent variable in the SysId regression) explained by the linear model. This can be quite variable, because acceleration is very susceptible to system noise. Mechanisms tend to vibrate quite a bit, 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

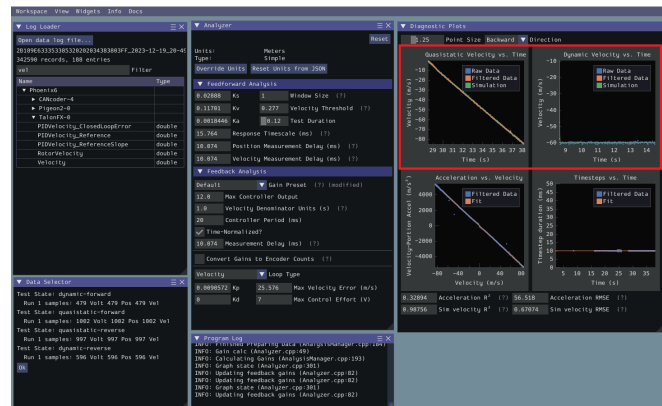
SysId also produces several diagnostic plots to help users evaluate the quality of their model fit.



Time-Domain Plots

Not: 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:



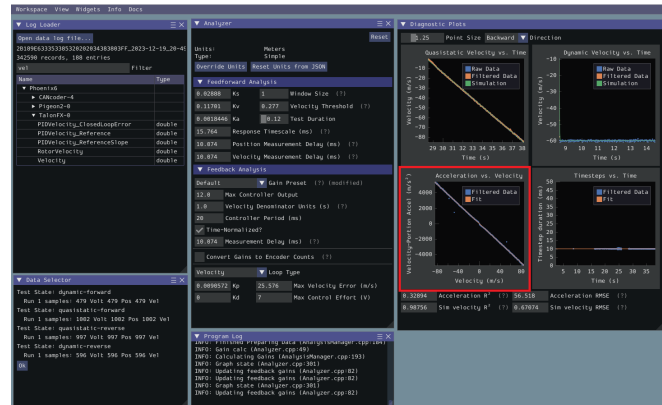
The velocity time domain plots contain three sets of data: Raw Data, Filtered Data, and Simulation. The Raw Data is the recorded data from your robot, the Filtered Data is the data after a median filter has been applied to the data, and the Simulation represents the velocity predictions of a model based 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.

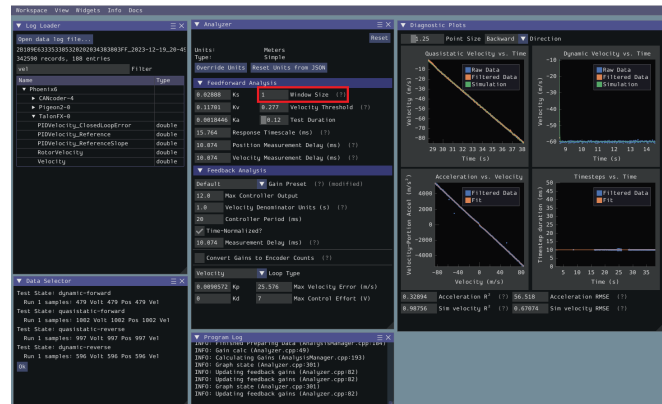
Deviation from this behavior is a sign of an *error*, either in your robot setup, analysis settings, or your test procedure.

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.



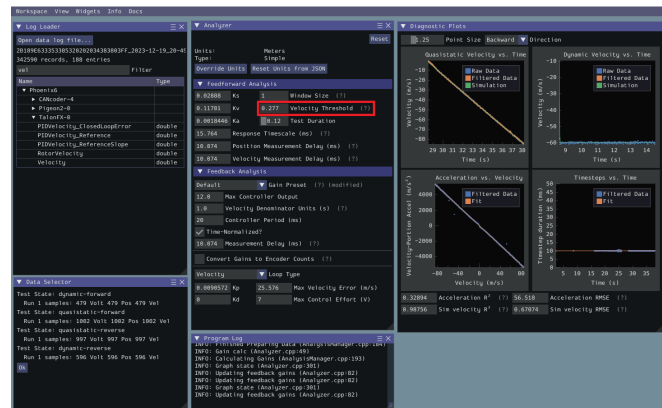
However, if your robot or mechanism has low mass compared to the motor power, this may “eat” what little meaningful acceleration data you have. In these cases k_A will tend towards zero and can be ignored for feedforward purposes. However, if k_A cannot be accurately measured, the calculated feedback gains are likely to be inaccurate, and manual tuning may be required.

Common Failure Modes

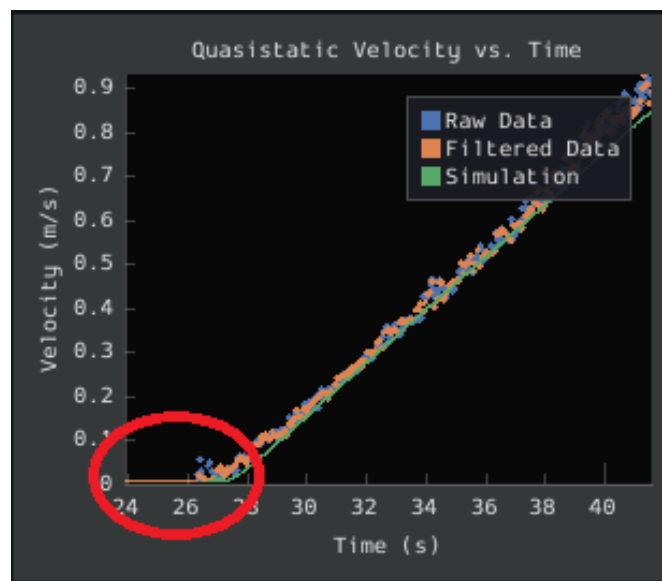
When something has gone wrong with the identification, diagnostic plots and console output provide crucial clues as to *what* has gone wrong. This section describes some common failures encountered while running the system identification tool, the identifying features of their diagnostic plots, and the 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.



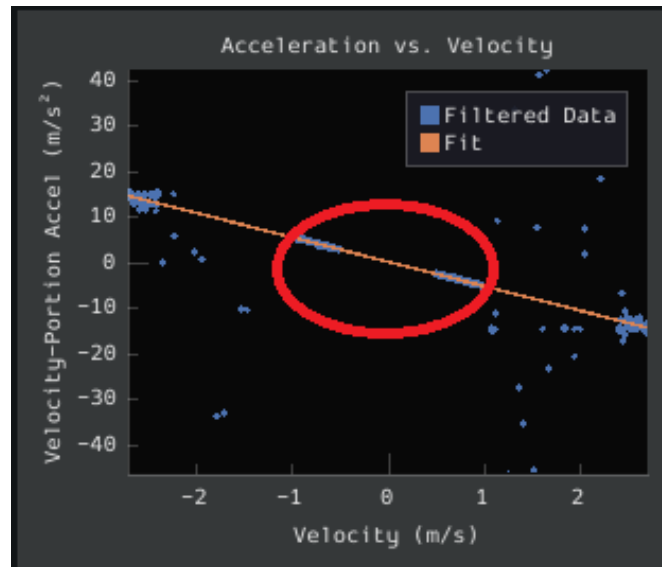
Velocity Threshold Too Low



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

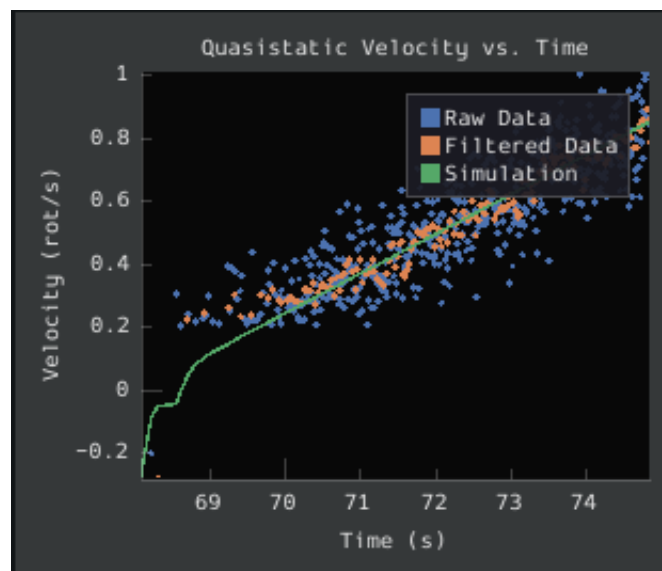


While not nearly as problematic as a too-low threshold, a velocity threshold that is too high will result in a large “gap” in the acceleration-versus-velocity plot.

To solve this, decrease the velocity threshold and re-analyze the data.

Noisy Velocity Signals

Not: 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).

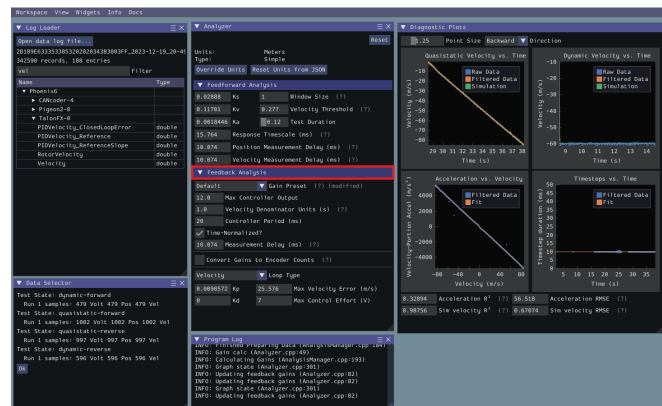
32.5.6 Analyzing Data

Feedforward Analysis

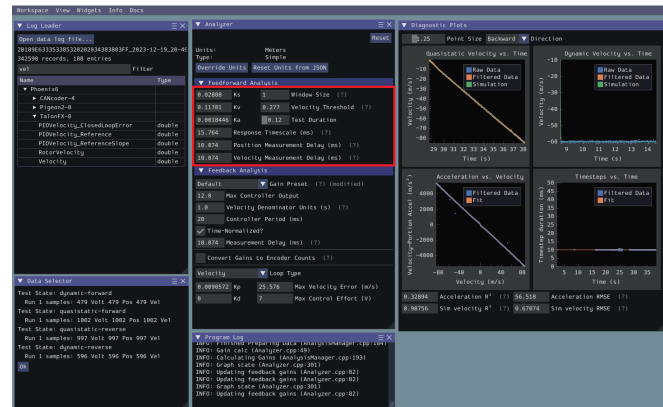
Not: For information on what the calculated feedback gains mean, see [The Permanent-Magnet DC Motor Feedforward Equation](#). For information on using the calculated feedback gains in code, see [feedforward control](#).

Click the dropdown arrow on the *Feedforward* Section.

Not: If you would like to change units, you will have to press the *Override Units* button and fill out the information on the popup.



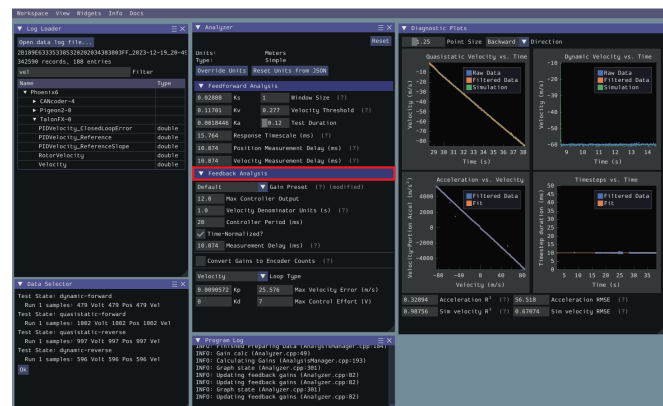
The computed mechanism system parameters will then be displayed.



Feedback Analysis

Önemli: 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.



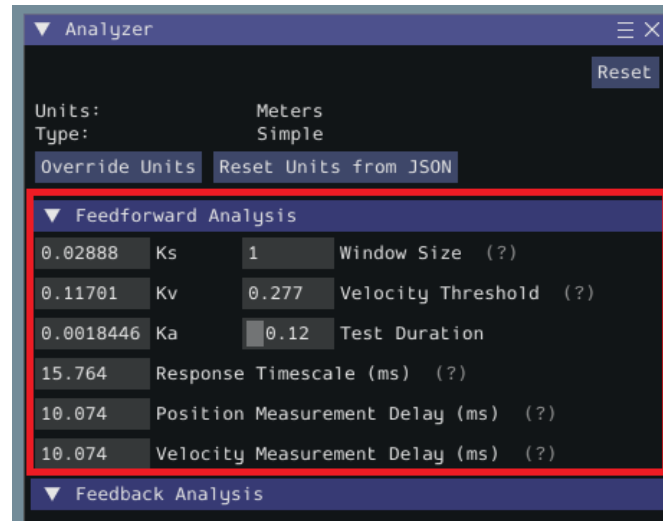
This view can be used to calculate optimal feedback gains for a PD or P controller for your mechanism (via *LQR*).

Enter Controller Parameters

Not: 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.

Not: 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

Not: 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*.

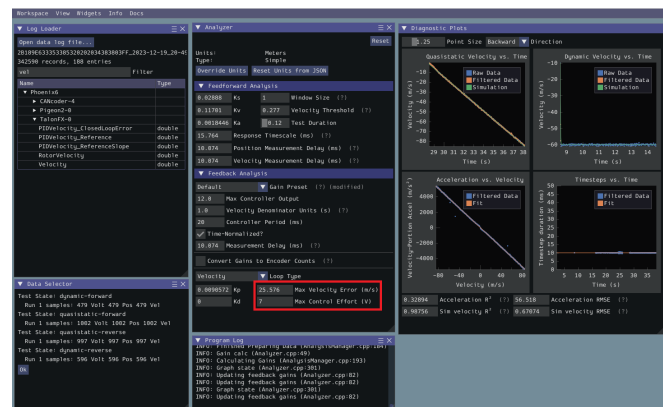
The following only applies if the user decides to implement their own custom filtering settings (e.g. adding a moving average filter to a WPILib PID loop or changing smart motorcontroller measurement period and/or measurement window size) as the measurement delay must be recalculated. Here is the general 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.

The screenshot shows the 'Analyzer' window with the following settings:

- Units:** Meters
- Type:** Simple
- Buttons:** Override Units, Reset Units from JSON, Reset
- Feedforward Analysis:**
 - 0.02888 Ks, 1 Window Size (?)
 - 0.11701 Kv, 0.277 Velocity Threshold (?)
 - 0.0018446 Ka, 0.12 Test Duration
 - 15.764 Response Timescale (ms) (?)
 - 10.074 Position Measurement Delay (ms) (?)
 - 10.074 Velocity Measurement Delay (ms) (?)
- Feedback Analysis:**
 - Default Gain Preset (?) (modified)
 - 12.0 Max Controller Output
 - 1.0 Velocity Denominator Units (s) (?)
 - 20 Controller Period (ms)
 - ☒ Time-Normalized?
 - 10.074 Measurement Delay (ms) (?)
 - ☐ Convert Gains to Encoder Counts (?)
- Loop Type:** Velocity (highlighted with a red box)
- Velocity Loop Parameters:**
 - 0.0090572 Kp, 25.576 Max Velocity Error (m/s)
 - 0 Kd, 7 Max Control Effort (V)

32.5.7 Additional Utilities and Tools

This page mainly covers useful information about additional functionality that this tool provides.

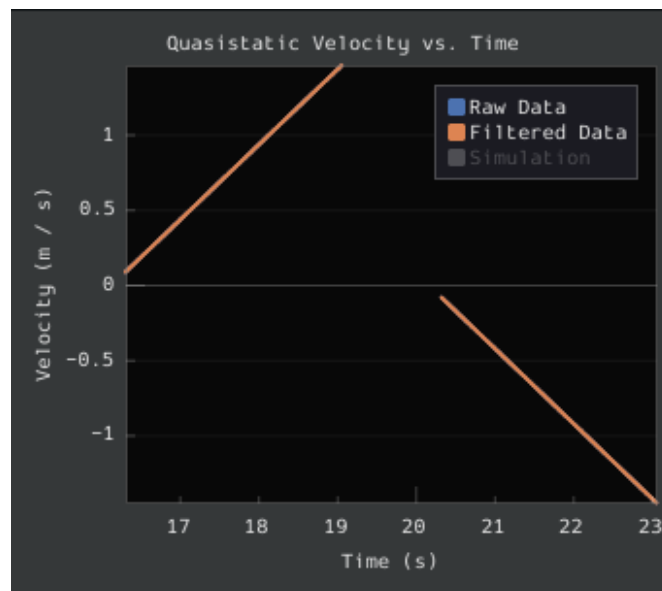
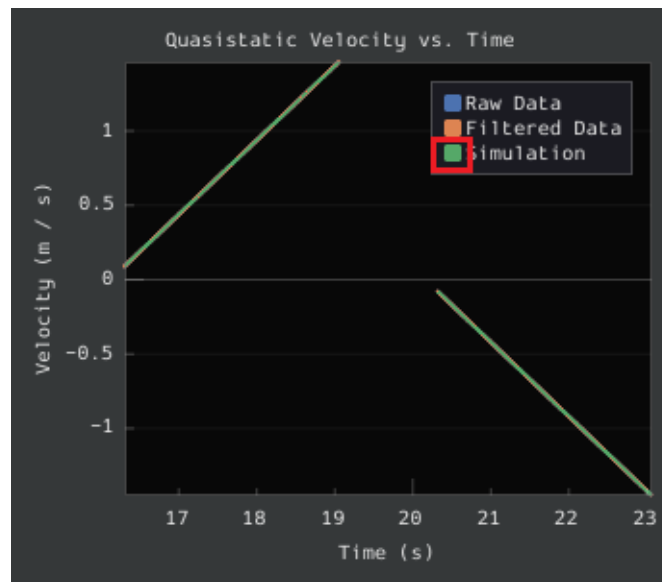
ImGui Tips

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.

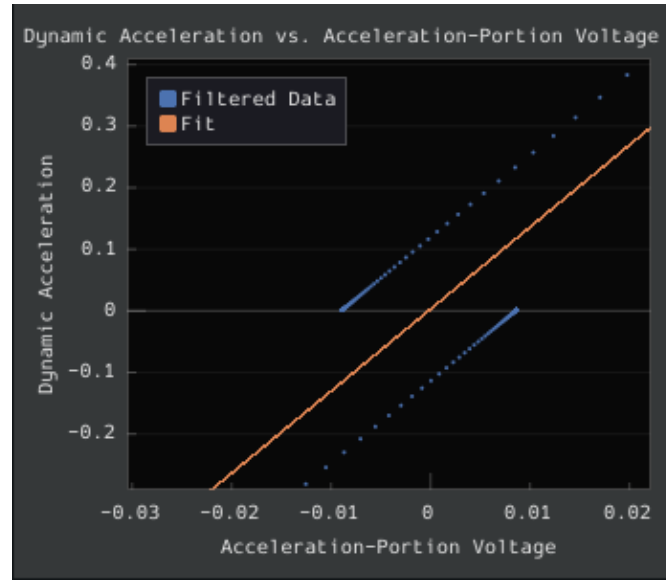
For example, if we want to hide sim data, we can click the green color box.



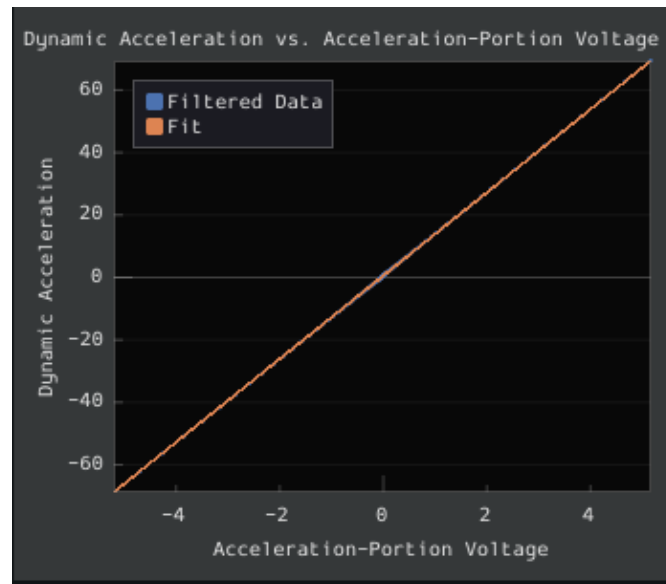
Auto Sizing Plots

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:



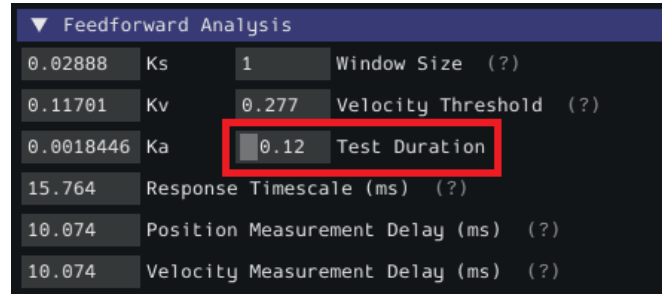
After double clicking, it is automatically resized:



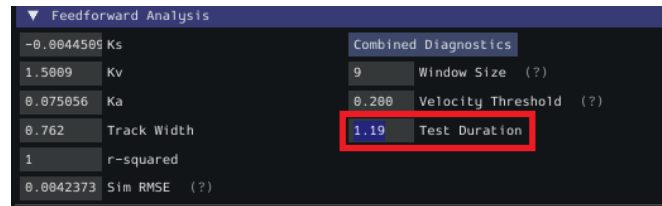
Setting Slider Values

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 Kontrolörler - Controllers

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'de PID Kontrolü

Not: This article focuses on in-code implementation of PID control in WPILib. For a conceptual explanation of the working of a PIDController, see [PID'ye Giriş](#)

Not: *Komut Tabanlı framework* aracılığıyla PID kontrolünün uygulanmasına ilişkin bir kılavuz için bkz *PIDSubsystems ve PIDCommands üzerinden PID Kontrol*.

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.

PIDController Sınıfını Kullanma

Bir PIDController Oluşturma

Not: PIDController eşzamansız olarak kullanılabilirken, herhangi güvenlik özelliği *sağlamaz* - güvenli çalışmanın tamamen kullanıcıya bırakılmasını sağlar ve bu nedenle eşzamansız kullanım yalnızca gelişmiş ekipler için önerilir.

WPILib'in PID kontrol fonksiyonelliğini kullanmak için, kullanıcılar önce istenen kazançlarla bir PIDController nesnesi oluşturmalıdır:

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)
```

Constructor'a, denetleyicinin çalıştırılacağı süreyi belirleyen isteğe bağlı dördüncü bir parametre sağlanabilir. PIDController nesnesi, esas olarak ana robot döngüsünden senkronize kullanım için tasarlanmıştır ve bu nedenle değeri varsayılan olarak 20ms'dir.

Geri Besleme Döngüsü Çıkışı Kullanma

Not: PIDController, calculate() yönteminin yapılandırılan periyotla tutarlı bir aralıkta düzenli olarak çağrıldığını varsayar. Bunun yapılmaması, istenmeyen döngü davranışına neden olacaktır.

Oluşturulan PIDController i kullanmak basittir: basitçe robotun ana döngüsünden calculate() yöntemini çağırın (örneğin, robotun autonomousPeriodic() yöntemi):

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))
```

Hataları Kontrol Etmek

Not: `getPositionError()` ve `getVelocityError()`, döngünün bir konumu kontrol ettiği varsayılarak adlandırılır - bir hızı kontrol eden bir döngü için bunlar sırasıyla hız hatasını ve hızlanma hatasını döndürür.

Ölçülen işlem değişkeninin mevcut hatası, ```getPositionError()``` işlevi tarafından döndürülür, türevi ise `getVelocityError()` işlevi tarafından döndürülür:

Toleransları Belirleme ve Kontrol Etme

Not: Yalnızca bir konum toleransı belirtilirse, hız toleransı varsayılan olarak sonsuzdur.

Not: Yukarıdaki gibi, `position` işlem değişken ölçümünü ve `velocity` bunun türevini ifade eder - dolayısıyla, bir hız döngüsü için bunlar aslında sırasıyla hız ve ivmedir.

Bazen, bir denetleyicinin ayar noktasını belirli bir tolerans dahilinde izleyip izlemediğini bilmek yararlıdır - örneğin, bir komutun sonlandırılması gerekip gerekmediğini veya (bir hareket profilini takip ederken) hareketin engellenip engellenmediğini belirlemek yada tekrar planlamak için.

Bunu yapmak için önce toleransları `setTolerance()` yöntemi ile belirlemeliyiz; daha sonra `atSetpoint()` metodu ile kontrol edebiliriz.

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()
```

Denetleyiciyi Sıfırlama

Artık geçerli olmayabileceğinden (örneğin, PIDController devre dışı bırakıldığında ve ardından yeniden etkinleştirildiğinde) bir PIDController in dahili durumunu (en önemlisi, integral akümülatörü) temizlemek istenebilir. . Bu, `reset()` yöntemi çağrılarak gerçekleştirilebilir.

Max Integrator Değeri Ayarlama

Not: Entegratörler, geri besleme döngü sistemlerine kararsızlık ve histerezis getirir. Kesinlikle başka hiçbir çözüm işe yaramayacaksa ekiplerin integral kazanımı kullanmaktan kaçınmaları şiddetle tavsiye edilir - çoğu zaman, bir entegratörle çözülebilecek sorunlar daha doğru bir *feedforward* kullanılarak daha iyi çözülebilir.

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)
```

Sürekli Girişi - Continuous Input Ayarlama

Uyarı: Mekanizmanız tamamen sürekli dönme kabiliyetine sahip değilse (örneğin, kabloları döndükçe bükülen, kayma halkası olmayan bir turret), mekanizmanın limitlerini geçmesini önlemek için ek bir güvenlik özelliği uygulamadıysanız sürekli girişi *etkinleştirmeyin* !

Uyarı: Sürekli giriş işlevi, giriş değerlerinizi otomatik olarak *kaydırmaz* - bu özelliği kullanırken, giriş değerlerinin asla belirtilen aralığın dışında olmadığından emin olun!

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.

Bunu otomatik olarak yapacak bir PIDController yapılandırmak için, `enableContinuousInput()` yöntemini kullanın:

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)
```

Kontrol Çıkışını Bağlama**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’de Feedforward Kontrolü

Not: 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, kullanıcıların mekanizmaları için doğru ileri besleme kontrolü uygulamalarına yardımcı olmak için bir dizi sınıf sağlar. Birçok yönden, doğru bir ileri besleme, bir mekanizmanın etkili kontrolü için geri bildirimden daha önemlidir. Çoğu FRC ® mekanizmalar, iyi anlaşılmış sistem denklemlerine yakından uyar, doğru bir ileri beslemeyle başlamak, doğru ve sağlam mekanizma kontrolü için hem kolay hem de büyük ölçüde faydalıdır.

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 şu anda feedforward denetimi için aşağıdaki üç yardımcı sınıfı sağlar:

- *SimpleMotorFeedforward* (Java, C++, Python)
- *ArmFeedforward* (Java, C++, Python)
- *ElevatorFeedforward* (Java, C++, Python)

SimpleMotorFeedforward

Not: C ++ 'da, SimpleMotorFeedforward sınıfı, açısal veya doğrusal olabilen mesafe ölçümleri için kullanılan birim türüne göre şablonlanır. Kabul edilen kazançlar, mesafe birimleriyle tutarlı birimlere sahip *olmalıdır*, yoksa bir derleme zamanı hatası alınır. kS birimleri volt, kV birimleri volt * saniye / mesafe ve kA'' birimleri ``volt * saniye^2 / mesafe olmalıdır. C ++ birimleri hakkında daha fazla bilgi için bkz *C ++ Ünite Kitaplığı*.

Not: Java feedforward bileşenleri, çıktıları, kullanıcı tarafından sağlanan feedforward kazançlarının birimleri cinsinden hesaplar. WPILibJ'in güvenli-tip bir unit sistemi olmadığı için kullanıcılar *unitleri tutarlı tutmaya özen göstermelidir*.

Not: 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 sınıfı, dönen tekerlekler ve robot sürücüler gibi sürtünme ve atalet dışında hiçbir harici yüke sahip olmayan sabit mıknatıslı DC motorlardan oluşan mekanizmalar için ileri beslemeyi hesaplar.

Bir SimpleMotorFeedforward oluşturmak için, basitçe gerekli kazançları kullanın:

Not: k_A kazancı ihmal edilebilir ve eğer öyleyse, varsayılan olarak sıfır değerine ayarlanır. Bu pek çok mekanizma için, özellikle az eylemsizliği olanlar için gerekli değildir.

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)
```

Feedforward hesaplamak için, istenen motor hızı ve ivmesiyle `calculate()` yöntemini çağırmanız yeterlidir:

Not: İvme argümanı `calculate()` çağrısından çıkarılabilir ve eğer öyleyse, varsayılan olarak sıfır değerine ayarlanır. Bu, açıkça tanımlanmış bir ivmelenme ayar noktası olmadığında yapılmalıdır.

JAVA

```
// Calculates the feedforward for a velocity of 10 units/second and an acceleration
// of 20 units/second^2
// Units are determined by the units of the gains passed in at construction.
feedforward.calculate(10, 20);
```

C++

```
// Calculates the feedforward for a velocity of 10 meters/second and an acceleration_
↳ of 20 meters/second^2
// Output is in volts
feedforward.Calculate(10_mps, 20_mps_sq);
```

PYTHON

```
# Calculates the feedforward for a velocity of 10 meters/second and an acceleration_
↳ of 20 meters/second^2
# Output is in volts
feedforward.calculate(10, 20)
```

ArmFeedforward

Not: 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 ++ Ünite Kitaplığı](#).

Not: Java feedforward bileşenleri, çıktıları, kullanıcı tarafından sağlanan feedforward kazançlarının birimleri cinsinden hesaplar. WPILibJ'in güvenli-tip bir unit sistemi olmadığı için kullanıcılar *unitleri tutarlı tutmaya özen göstermelidir*.

Not: 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 sınıfı, doğrudan kalıcı mıknatıslı bir DC motor tarafından kontrol edilen kollar için sürtünme, atalet ve kol kütlelerinin harici yükleme ile ileri beslemeyi hesaplar. Bu, FRC'deki çoğu kol için doğru bir modelidir.

Bir ArmFeedforward oluşturmak için, onu gerekli kazançlarla ayarlamanız yeterlidir:

Not: kA kazancı ihmal edilebilir ve eğer öyleyse, varsayılan olarak sıfır değerine ayarlanır. Bu pek çok mekanizma için, özellikle az eylemsizliği olanlar için gerekli değildir.

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)
```

Feedforward hesaplamak için, istenen kol konumu, hızı ve ivmeyle `calculate()` yöntemini çağırmanız yeterlidir:

Not: İvme argümanı `calculate()` çağrısından çıkarılabilir ve eğer öyleyse, varsayılan olarak sıfır değerine ayarlanır. Bu, açıkça tanımlanmış bir ivmelenme ayar noktası olmadığında yapılmalıdır.

JAVA

```
// Calculates the feedforward for a position of 1 units, a velocity of 2 units/second,
↪ and
// an acceleration of 3 units/second^2
// Units are determined by the units of the gains passed in at construction.
feedforward.calculate(1, 2, 3);
```

C++

```
// Calculates the feedforward for a position of 1 radians, a velocity of 2 radians/
↪ second, and
// an acceleration of 3 radians/second^2
// Output is in volts
feedforward.Calculate(1_rad, 2_rad_per_s, 3_rad/(1_s * 1_s));
```

PYTHON

```
# Calculates the feedforward for a position of 1 radians, a velocity of 2 radians/  
↪second, and  
# an acceleration of 3 radians/second^2  
# Output is in volts  
feedforward.calculate(1, 2, 3)
```

ElevatorFeedforward

Not: In C++, the passed-in gains *must* have units consistent with the distance units, or a compile-time error will be thrown. k_S and k_G should have units of volts, k_V should have units of volts * seconds / distance, and k_A should have units of volts * seconds² / distance. For more information on C++ units, see [C++ Ünite Kitablığı](#).

Not: Java feedforward bileşenleri, çıktıları, kullanıcı tarafından sağlanan feedforward kazançlarının birimleri cinsinden hesaplar. WPILibJ'in güvenli-tip bir unit sistemi olmadığı için kullanıcılar *unitleri tutarlı tutmaya özen göstermelidir*.

Not: 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 sınıfı, sürtünme, atalet ve asansör kütlesi ile yüklenen sabit mıknatıslı DC motorlardan oluşan asansörler için feedforward hesaplar. Bu, FRC'deki çoğu asansörün uygun bir modelidir.

ElevatorFeedforward oluşturmak için, onu gerekli kazançlarla yapılandırmanız yeterlidir:

Not: k_A kazancı ihmal edilebilir ve eğer öyleyse, varsayılan olarak sıfır değerine ayarlanır. Bu pek çok mekanizma için, özellikle az eylemsizliği olanlar için gerekli değildir.

JAVA

```
// Create a new ElevatorFeedforward with gains  $k_S$ ,  $k_G$ ,  $k_V$ , and  $k_A$   
ElevatorFeedforward feedforward = new ElevatorFeedforward( $k_S$ ,  $k_G$ ,  $k_V$ ,  $k_A$ );
```

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)
```

Feedforward hesaplamak için, istenen motor hızı ve ivmesiyle `calculate()` yöntemini çağırmanız yeterlidir:

Not: İvme argümanı `calculate()` çağrısından çıkarılabilir ve eğer öyleyse, varsayılan olarak sıfır değerine ayarlanır. Bu, açıkça tanımlanmış bir ivmelenme ayar noktası olmadığında yapılmalıdır.

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)
```


Kontrol Mekanizmaları için Feedforward Kullanma

Not: 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.

Feedforward kontrolü, bir geri bildirim denetleyicisi olmadan tamamen kendi başına kullanılabilir. Bu, “açık döngü” kontrolü olarak bilinir ve birçok mekanizma için (özellikle robot sürücüler) mükemmel şekilde tatmin edici olabilir. Bir robot sürücüsünü aşağıdaki şekilde kontrol etmek için bir `SimpleMotorFeedforward` kullanılabilir:

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 Feedforward - İleribesleme ve PID Kontrolünü Birleştirme

Not: Bu makale, WPILib’in sağladığı kütüphane sınıfları ile birlikte feedforward/PID kontrolünün kod içi uygulamasını kapsar. İlgili kavramları daha ayrıntılı olarak açıklayan belgeler yakında çıkacaktır.

Feedforward ve geri bildirim denetleyicilerinin her biri ayrı ayrı kullanılabilir, ancak birlikte kullanıldığında en etkilidir. Neyse ki, bu iki kontrol yöntemini birleştirmek *son derece* basittir - basitçe çıktıların bir araya getirir.

Feedforward'ı bir PIDController ile kullanma

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)
```

Dahası, ileri besleme, tamamen geri bildirimden ayrı bir özelliktir ve bu nedenle, endişelerin ayrılmasını ihlal ettiği için aynı denetleyici nesnesinde ele alınması için bir neden yoktur. WPILib, ortak FRC ® için doğru ileri besleme voltajlarını hesaplamak üzere birkaç yardımcı sınıfla birlikte gelir. mekanizmalar - daha fazla bilgi için bkz : ref: docs / software / advanced-controls / controller / feedforward: *Feedforward Control in WPILib*.

PID ile Feedforward Bileşenlerini Kullanma

Not: 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.

Feedforward / PID kontrolünün daha eksiksiz bir örneği neye benzeyebilir? Feedforward sayfasından [sürüş örneği](#) dikkate alınız. Geri bildirim kontrolünü dahil etmek için bunu kolayca değiştirebiliriz (SimpleMotorFeedforward bileşeni ile)

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)
    )
```

Diğer mekanizma türleri de benzer şekilde ele alınabilir.

32.6.4 WPILib’de Trapezoid Hareket Profilleri

Not: Bu makale, trapezoidal hareket profillerinin kod içi üretimini kapsar. İlgili kavramları daha ayrıntılı olarak açıklayan belgeler yakında çıkacaktır.

Not: TrapezoidProfile sınıfının *komut tabanlı framework* çerçevesinde uygulanmasına ilişkin bir kılavuz için bkz *TrapezProfileSubsystems ve TrapezoidProfileCommands ile Hareket Profilleme*.

Not: Kendi başına kullanılan TrapezoidProfile sınıfı, özel bir kontrolörle (yerleşik PID işlevselliğine sahip “akıllı” bir motor kontrolörü gibi) oluşturulduğunda çok kullanışlıdır. Bir WPILib PIDController ile entegre etmek için, bakınız *Motion Profiling ve PID Control’ü ProfiledPIDController ile Birleştirme*.

İleri besleme ve geri bildirim kontrolü, belirli bir ayar noktasına ulaşmak için uygun yollar sunsa da, çoğu zaman mekanizmalarımız için ayar noktaları oluşturma sorunuyla karşı karşıyayız. Bir mekanizmaya derhal istenen duruma getirme komutu yaklaşımı işe yarayabilirken, çoğu zaman yetersizdir. Mekanizmalarımızın işlevini iyileştirmek için, mekanizmalara, mevcut durumu ile istenen hedef durumu arasında sorunsuz bir şekilde geçiş yapan bir *dizi ayar noktasına* komut vermek isteriz.

To help users do this, WPILib provides a TrapezoidProfile class (Java, C++, Python).

Trapezoidal Profil Oluşturma

Not: C++ 'da, TrapezoidProfile 'sınıfı, açısal veya doğrusal olabilen mesafe ölçümleri için kullanılan birim tipine göre şablonlanır. İletilen değerler zorunlu mesafe birimleriyle tutarlı birimlere sahip *olmalıdır*; aksi takdirde derleme zamanı hatası atılır. C++ birimleri hakkında daha fazla bilgi için bkz . *C++ Ünite Kitaplığı*.

Kısıtlamalar

Not: Çeşitli *ileribesleme yardımcı sınıfları*, bir mekanizmanın aynı anda elde edilebilen maksimum hızını ve ivmesini hesaplamak için yöntemler sağlar. Bunlar, TrapezoidProfile için uygun hareket kısıtlamalarını hesaplamak için çok yararlı olabilir.

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)
```

Başlangıç ve Bitiş Durumları

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)
```

Hepsini bir araya koymak

Not: C++ genellikle iç sınıfların türünü çıkarabilir ve bu nedenle basit bir başlatıcı listesi (sınıf adı olmadan) parametre olarak gönderilebilir. Tam sınıf adları, netlik açısından aşağıdaki örnekte verilmiştir.

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))
```

Bir TrapezoidProfile Kullanmak**Profil Örnekleme**

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))
```

Profil Durumunun Kullanılması

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)
```

Tam Kullanım Örneği

Not: In this example, the initial state is re-computed every timestep. This is a somewhat different usage technique than is detailed above, but works according to the same principles - the profile is sampled at a time corresponding to the loop period to get the setpoint for the next loop iteration.

A more complete example of `TrapezoidProfile` usage is provided in the `ElevatorTrapezoidProfile` example project ([Java](#), [C++](#), [Python](#)):

JAVA

```

5 package edu.wpi.first.wpilibj.examples.elevatortrapezoidprofile;
6
7 import edu.wpi.first.math.controller.SimpleMotorFeedforward;
8 import edu.wpi.first.math.trajectory.TrapezoidProfile;
9 import edu.wpi.first.wpilibj.Joystick;
10 import edu.wpi.first.wpilibj.TimedRobot;
11
12 public class Robot extends TimedRobot {
13     private static double kDt = 0.02;
14
15     private final Joystick m_joystick = new Joystick(1);
16     private final ExampleSmartMotorController m_motor = new
17     ↪ ExampleSmartMotorController(1);
18     // Note: These gains are fake, and will have to be tuned for your robot.
19     private final SimpleMotorFeedforward m_feedforward = new SimpleMotorFeedforward(1,
20     ↪ 1.5);
21
22     // Create a motion profile with the given maximum velocity and maximum
23     // acceleration constraints for the next setpoint.
24     private final TrapezoidProfile m_profile =
25     ↪ new TrapezoidProfile(new TrapezoidProfile.Constraints(1.75, 0.75));
26     private TrapezoidProfile.State m_goal = new TrapezoidProfile.State();
27     private TrapezoidProfile.State m_setpoint = new TrapezoidProfile.State();
28
29     @Override
30     public void robotInit() {
31         // Note: These gains are fake, and will have to be tuned for your robot.
32         m_motor.setPID(1.3, 0.0, 0.7);
33     }
34
35     @Override
36     public void teleopPeriodic() {
37         if (m_joystick.getRawButtonPressed(2)) {
38             m_goal = new TrapezoidProfile.State(5, 0);
39         } else if (m_joystick.getRawButtonPressed(3)) {
40             m_goal = new TrapezoidProfile.State();
41         }
42
43         // Retrieve the profiled setpoint for the next timestep. This setpoint moves
44         // toward the goal while obeying the constraints.
45         m_setpoint = m_profile.calculate(kDt, m_setpoint, m_goal);
46
47         // Send setpoint to offboard controller PID
48         m_motor.setSetpoint(
49             ↪ ExampleSmartMotorController.PIDMode.kPosition,
50             ↪ m_setpoint.position,
51             ↪ m_feedforward.calculate(m_setpoint.velocity) / 12.0);
52     }
53 }

```


C++

```

5  #include <numbers>
6
7  #include <frc/Joystick.h>
8  #include <frc/TimedRobot.h>
9  #include <frc/controller/SimpleMotorFeedforward.h>
10 #include <frc/trajectory/TrapezoidProfile.h>
11 #include <units/acceleration.h>
12 #include <units/length.h>
13 #include <units/time.h>
14 #include <units/velocity.h>
15 #include <units/voltage.h>
16
17 #include "ExampleSmartMotorController.h"
18
19 class Robot : public frc::TimedRobot {
20 public:
21     static constexpr units::second_t kDt = 20_ms;
22
23     Robot() {
24         // Note: These gains are fake, and will have to be tuned for your robot.
25         m_motor.SetPID(1.3, 0.0, 0.7);
26     }
27
28     void TeleopPeriodic() override {
29         if (m_joystick.GetRawButtonPressed(2)) {
30             m_goal = {5_m, 0_mps};
31         } else if (m_joystick.GetRawButtonPressed(3)) {
32             m_goal = {0_m, 0_mps};
33         }
34
35         // Retrieve the profiled setpoint for the next timestep. This setpoint moves
36         // toward the goal while obeying the constraints.
37         m_setpoint = m_profile.Calculate(kDt, m_setpoint, m_goal);
38
39         // Send setpoint to offboard controller PID
40         m_motor.SetSetpoint(ExampleSmartMotorController::PIDMode::kPosition,
41                             m_setpoint.position.value(),
42                             m_feedforward.Calculate(m_setpoint.velocity) / 12_V);
43     }
44
45 private:
46     frc::Joystick m_joystick{1};
47     ExampleSmartMotorController m_motor{1};
48     frc::SimpleMotorFeedforward<units::meters> m_feedforward{
49         // Note: These gains are fake, and will have to be tuned for your robot.
50         1_V, 1.5_V * 1_s / 1_m};
51
52     // Create a motion profile with the given maximum velocity and maximum
53     // acceleration constraints for the next setpoint.
54     frc::TrapezoidProfile<units::meters> m_profile{{1.75_mps, 0.75_mps_sq}};
55     frc::TrapezoidProfile<units::meters>::State m_goal;
56     frc::TrapezoidProfile<units::meters>::State m_setpoint;
57 };
58
59 #ifndef RUNNING_FRC_TESTS

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

60 int main() {
61     return frc::StartRobot<Robot>();
62 }
63 #endif

```

PYTHON

```

8  import wpilib
9  import wpimath.controller
10 import wpimath.trajectory
11 import examplesmartmotorcontroller
12
13
14 class MyRobot(wpilib.TimedRobot):
15     kDt = 0.02
16
17     def robotInit(self):
18         self.joystick = wpilib.Joystick(1)
19         self.motor = examplesmartmotorcontroller.ExampleSmartMotorController(1)
20         # Note: These gains are fake, and will have to be tuned for your robot.
21         self.feedforward = wpimath.controller.SimpleMotorFeedforwardMeters(1, 1.5)
22
23         self.constraints = wpimath.trajectory.TrapezoidProfile.Constraints(1.75, 0.75)
24
25         self.goal = wpimath.trajectory.TrapezoidProfile.State()
26         self.setpoint = wpimath.trajectory.TrapezoidProfile.State()
27
28         # Note: These gains are fake, and will have to be tuned for your robot.
29         self.motor.setPID(1.3, 0.0, 0.7)
30
31     def teleopPeriodic(self):
32         if self.joystick.getRawButtonPressed(2):
33             self.goal = wpimath.trajectory.TrapezoidProfile.State(5, 0)
34         elif self.joystick.getRawButtonPressed(3):
35             self.goal = wpimath.trajectory.TrapezoidProfile.State(0, 0)
36
37         # Create a motion profile with the given maximum velocity and maximum
38         # acceleration constraints for the next setpoint, the desired goal, and the
39         # current setpoint.
40         profile = wpimath.trajectory.TrapezoidProfile(
41             self.constraints, self.goal, self.setpoint
42         )
43
44         # Retrieve the profiled setpoint for the next timestep. This setpoint moves
45         # toward the goal while obeying the constraints.
46         self.setpoint = profile.calculate(self.kDt)
47
48         # Send setpoint to offboard controller PID
49         self.motor.setSetPoint(
50             examplesmartmotorcontroller.ExampleSmartMotorController.PIDMode.kPosition,
51             self.setpoint.position,
52             self.feedforward.calculate(self.setpoint.velocity) / 12,
53         )

```

32.6.5 Motion Profiling ve PID Control'ü ProfiledPIDController ile Birleştirme

Not: ProfiledPIDController sınıfının *command-based framework* Çerçevesine uygulanmasına ilişkin bir kılavuz için, bkz: *Komut Tabanlı Hareket Profileleme ve PID'yi Birleştirme*.

Önceki makalede, trapezoidal hareket profili oluşturmak ve kullanmak için ``TrapezoidProfile`` sınıfının nasıl kullanılacağını gördük. Bu makaledeki örnek kod, bir “akıllı” motor kontrol cihazının harici PID kontrol özelliği ile TrapezoidProfile sınıfını manuel olarak oluşturmayı gösterir.

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.

ProfiledPIDController sınıfını kullanma

Not: C++’da, ``ProfiledPIDController`` sınıfı, açısal veya doğrusal olabilen mesafe ölçümleri için kullanılan birim türüne göre şablonlanır. İletilen değerler *zorunlu* mesafe birimleriyle tutarlı birimlere sahip olmalıdır, aksi takdirde derleme zamanı hatası atılır. C++ birimleri hakkında daha fazla bilgi için bkz *C++ Ünite Kitabı*..

Not: ProfiledPIDController işlevinin çoğu, ``PIDController`` ile etkin bir şekilde aynıdır. Buna göre, bu makale yalnızca hareket profili oluşturma işlevini barındırmak için büyük ölçüde değiştirilmiş özellikleri kapsayacaktır. Standart PIDController özellikleri hakkında bilgi için bkz.: ref: docs / software / advanced-controls / controllers / pidcontroller: PID Control in WPILib.

Profilli PIDController Oluşturma

Not: C++ genellikle iç sınıfların türünü çıkarabilir ve bu nedenle basit bir başlatıcı listesi (sınıf adı olmadan) parametre olarak gönderilebilir. Tam sınıf adı, netlik açısından aşağıdaki örnekte verilmiştir.

Bir ProfiledPIDController oluşturmak, *creating a PIDController* oluşturmak ile neredeyse aynıdır. Tek fark, dahili olarak oluşturulan ``TrapezoidProfile`` örneklerine otomatik olarak iletilecek olan bir dizi *trapezoidal profile constraints*, sağlama ihtiyacıdır:

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))
```

Hedef ve Ayar Noktası

Standart bir ``PIDController`` ile ``ProfiledPIDController`` arasındaki en büyük fark, kontrol döngüsünün gerçek* ayar noktasının * kullanıcı tarafından doğrudan belirtilmemesidir. Bunun yerine, kullanıcı bir *hedef* konumu veya durumu belirtir ve kontrolör için ayar noktası, mevcut durum ile hedef arasında oluşturulan hareket profilinden otomatik olarak hesaplanır. Dolayısıyla, kullanıcı tarafı arama çoğunlukla aynı görünürken:

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))
```

Belirtilen hedef değeri (sıfırdan farklı bir hız isteniyorsa, bir konum değeri veya bir TrapezProfile.State olabilir), döngünün *zorunlu* akım **ayar noktası* değildir - daha ziyade, oluşturulan profil sona erdiğinde **nihai * ayar noktasıdır*.

Ayar Noktasını Alma / Kullanma

ProfiledPIDController hedefi ayar noktasından farklı olduğundan, genellikle kontrolörün mevcut ayar noktasını sorgulamak istenir (örneğin, kullanılacak değerleri almak için: ref: *ileri besleme <docs/software/advanced-controls/controllers/combining-feedforward-feedback:Using Feedforward Components with PID>*). Bu, `getSetpoint ()` yöntemi ile yapılabilir.

Döndürülen ayar noktası daha sonra aşağıdaki örnekte olduğu gibi kullanılabilir:

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()

```

Tam Kullanım Örneği

A more complete example of ProfiledPIDController usage is provided in the ElevatorProfilePID example project (Java, C++, Python):

JAVA

```

5  package edu.wpi.first.wpilibj.examples.elevatorprofiledpid;
6
7  import edu.wpi.first.math.controller.ElevatorFeedforward;
8  import edu.wpi.first.math.controller.ProfiledPIDController;
9  import edu.wpi.first.math.trajectory.TrapezoidProfile;
10 import edu.wpi.first.wpilibj.Encoder;
11 import edu.wpi.first.wpilibj.Joystick;
12 import edu.wpi.first.wpilibj.TimedRobot;
13 import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
14
15 @SuppressWarnings("PMD.RedundantFieldInitializer")
16 public class Robot extends TimedRobot {
17     private static double kDt = 0.02;
18     private static double kMaxVelocity = 1.75;
19     private static double kMaxAcceleration = 0.75;
20     private static double kP = 1.3;
21     private static double kI = 0.0;
22     private static double kD = 0.7;
23     private static double kS = 1.1;
24     private static double kG = 1.2;
25     private static double kV = 1.3;
26
27     private final Joystick m_joystick = new Joystick(1);
28     private final Encoder m_encoder = new Encoder(1, 2);
29     private final PWMSparkMax m_motor = new PWMSparkMax(1);
30
31     // Create a PID controller whose setpoint's change is subject to maximum
32     // velocity and acceleration constraints.
33     private final TrapezoidProfile.Constraints m_constraints =
34         new TrapezoidProfile.Constraints(kMaxVelocity, kMaxAcceleration);
35     private final ProfiledPIDController m_controller =
36         new ProfiledPIDController(kP, kI, kD, m_constraints, kDt);
37     private final ElevatorFeedforward m_feedforward = new ElevatorFeedforward(kS, kG,
38         ↪ kV);
39
40     @Override
41     public void robotInit() {
42         m_encoder.setDistancePerPulse(1.0 / 360.0 * 2.0 * Math.PI * 1.5);
43     }
44
45     @Override
46     public void teleopPeriodic() {
47         if (m_joystick.getRawButtonPressed(2)) {
48             m_controller.setGoal(5);
49         } else if (m_joystick.getRawButtonPressed(3)) {
50             m_controller.setGoal(0);
51         }
52         // Run controller and update motor output

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

53     m_motor.setVoltage(
54         m_controller.calculate(m_encoder.getDistance())
55         + m_feedforward.calculate(m_controller.getSetpoint().velocity));
56     }
57 }

```

C++

```

5  #include <numbers>
6
7  #include <frc/Encoder.h>
8  #include <frc/Joystick.h>
9  #include <frc/TimedRobot.h>
10 #include <frc/controller/ElevatorFeedforward.h>
11 #include <frc/controller/ProfiledPIDController.h>
12 #include <frc/motorcontrol/PWMSparkMax.h>
13 #include <frc/trajectory/TrapezoidProfile.h>
14 #include <units/acceleration.h>
15 #include <units/length.h>
16 #include <units/time.h>
17 #include <units/velocity.h>
18 #include <units/voltage.h>
19
20 class Robot : public frc::TimedRobot {
21 public:
22     static constexpr units::second_t kDt = 20_ms;
23
24     Robot() {
25         m_encoder.SetDistancePerPulse(1.0 / 360.0 * 2.0 * std::numbers::pi * 1.5);
26     }
27
28     void TeleopPeriodic() override {
29         if (m_joystick.GetRawButtonPressed(2)) {
30             m_controller.SetGoal(5_m);
31         } else if (m_joystick.GetRawButtonPressed(3)) {
32             m_controller.SetGoal(0_m);
33         }
34
35         // Run controller and update motor output
36         m_motor.SetVoltage(
37             units::volt_t{
38                 m_controller.Calculate(units::meter_t{m_encoder.GetDistance()}) +
39                 m_feedforward.Calculate(m_controller.GetSetpoint().velocity));
40     }
41
42 private:
43     static constexpr units::meters_per_second_t kMaxVelocity = 1.75_mps;
44     static constexpr units::meters_per_second_squared_t kMaxAcceleration =
45         0.75_mps_sq;
46     static constexpr double kP = 1.3;
47     static constexpr double kI = 0.0;
48     static constexpr double kD = 0.7;
49     static constexpr units::volt_t kS = 1.1_V;
50     static constexpr units::volt_t kG = 1.2_V;

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

51  static constexpr auto kV = 1.3_V / 1_mps;
52
53  frc::Joystick m_joystick{1};
54  frc::Encoder m_encoder{1, 2};
55  frc::PWMSparkMax m_motor{1};
56
57  // Create a PID controller whose setpoint's change is subject to maximum
58  // velocity and acceleration constraints.
59  frc::TrapezoidProfile<units::meters>::Constraints m_constraints{
60      kMaxVelocity, kMaxAcceleration};
61  frc::ProfiledPIDController<units::meters> m_controller{kP, kI, kD,
62                                                         m_constraints, kDt};
63  frc::ElevatorFeedforward m_feedforward{kS, kG, kV};
64  };
65
66  #ifndef RUNNING_FRC_TESTS
67  int main() {
68      return frc::StartRobot<Robot>();
69  }
70  #endif

```

PYTHON

```

8  import wpilib
9  import wpimath.controller
10 import wpimath.trajectory
11 import math
12
13
14 class MyRobot(wpilib.TimedRobot):
15     kDt = 0.02
16
17     def robotInit(self) -> None:
18         self.joystick = wpilib.Joystick(1)
19         self.encoder = wpilib.Encoder(1, 2)
20         self.motor = wpilib.PWMSparkMax(1)
21
22         # Create a PID controller whose setpoint's change is subject to maximum
23         # velocity and acceleration constraints.
24         self.constraints = wpimath.trajectory.TrapezoidProfile.Constraints(1.75, 0.75)
25         self.controller = wpimath.controller.ProfiledPIDController(
26             1.3, 0, 0.7, self.constraints, self.kDt
27         )
28
29         self.encoder.setDistancePerPulse(1 / 360 * 2 * math.pi * 1.5)
30
31     def teleopPeriodic(self) -> None:
32         if self.joystick.getRawButtonPressed(2):
33             self.controller.setGoal(5)
34         elif self.joystick.getRawButtonPressed(3):
35             self.controller.setGoal(0)
36
37         # Run controller and update motor output
38         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

Uyarı: 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 Yörünge Oluşturma ve WPILib ile Takip Etme

Bu bölümde, parametrelili eğri yörüngeleri oluşturmak ve bu yörüngeleri tipik FRC ® ile izlemek için WPILib desteği açıklanmaktadır. robot sürücüler.

32.7.1 Yörünge Üretimi

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.

Bu makale, bir yörüngeyi nasıl oluşturulacağını anlatıyor. Bu dizideki sonraki birkaç makale, oluşturulan yörüngeyi gerçekte nasıl izleneceğini ele alacaktır. Yörüngeler dünyasına dalmadan önce robotunuzun sahip olması gereken birkaç şey var:

- Robotun her iki tarafının konumunu ve hızını ölçmenin bir yolu. Bunu yapmanın en iyi yolu kodlayıcıdır; ancak diğer seçenekler optik akış sensörleri vb. içerebilir.
- Robot kasasının açısını veya açısal oranını ölçmenin bir yolu. Bunu yapmanın en iyi yolu jiroskoptur. Açısal hız enkoder hızları kullanılarak hesaplanabilmesine rağmen, bu yöntem tekerlek dönme kaybı nedeniyle ÖNERİLMEZ.

If you are looking for a simpler way to perform autonomous navigation, see [the section on driving to a distance](#).

Spline'lar - Eğri setleri

Spline, noktalar arasında enterpolasyon yapan bir dizi eğriyi ifade eder. Eğriler dışında, noktaları birleştiren noktalar olarak düşünün. WPILib iki tür spline'ı destekler: hermit kenetli kübik ve hermit beşli.

- Hermite sıkıştırılmış kübik: Bu, çoğu kullanıcı için önerilen seçenektir. Bu eğri çizgiler kullanılarak yörüngelerin oluşturulması, tüm noktaların (x, y) koordinatlarının ve başlangıç ve bitiş yol noktalarındaki başlıkların belirlenmesini içerir. İç yol noktalarındaki başlıklar, sürekli eğriliği (yön değiştirme hızı) sağlamak için otomatik olarak belirlenir.
- Hermite quintic: Bu, kullanıcının * tüm * yol noktaları için (x, y) koordinatlarını ve başlıklarını belirtmesini gerektiren daha gelişmiş bir seçenektir. Bu, kenetlenmiş kübik yivlerin oluşturduğu yörüngelerden memnun değilseniz veya iç noktalardaki başlıkların daha iyi kontrolünü istiyorsanız kullanılmalıdır.

Spline'lar yörüngeler oluşturmak için bir araç olarak kullanılır; ancak, spline'ın kendisi hızlar ve ivmeler hakkında herhangi bir bilgiye sahip değildir. Bu nedenle, spline sınıflarını doğrudan kullanmanız önerilmez. Hızları ve ivmeleri olan düzgün bir yol oluşturmak için, bir * yörünge * oluşturulmalıdır.

Yörünge yapılandırmasını oluşturma

Bir yörünge oluşturmak için bir konfigürasyon oluşturulmalıdır. Yapılandırma, başlangıç hızı ve bitiş hızına ek olarak özel kısıtlamalar, maksimum hız, maksimum ivme hakkında bilgi içerir. Yapılandırma ayrıca yörüngeyi tersine çevrilmesi gerekip gerekmediği hakkında bilgi içerir (robot yol noktaları boyunca geriye doğru gider). Bir yapılandırma oluşturmak için ``TrajectoryConfig`` sınıfı kullanılmalıdır. Bu sınıfın yapıcısı iki argüman alır: maksimum hız ve maksimum ivme. Diğer alanlar (``startVelocity``, ``endVelocity``, ``reversed``, ``kısıtlamalar``) varsayılan değerlere (``0``, ``0``, ``false`` Nesne oluşturulduğunda, ``{}``). Bu alanlardan herhangi birinin değerlerini değiştirmek isterseniz, aşağıdaki yöntemleri çağırabilirsiniz:

- `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++)

Not: ``Ters-reversed`` özelliği, basitçe robotun geriye doğru hareket edip etmediğini gösterir. Dört yol noktası, a, b, c ve d belirtirseniz, robot, ``ters-reversed`` bayrağı ``doğru-true`` olarak ayarlandığında, yol noktaları boyunca aynı sırayla seyahat edecektir. Bu aynı zamanda, yol noktalarını sağlarken robotun yönünü hesaba katmanız gerektiği anlamına gelir. Örneğin, robotunuz ittifak istasyonu duvarınıza bakıyorsa ve bazı alan unsurlarına geri giderse, başlangıç yol noktasının 180 derece dönüşü olmalıdır.

Yörünge oluşturmak

Bir yörünge oluşturmak için kullanılan yöntem `generateTrajectory (...)` şeklindedir. Bu yöntem için dört aşırı yük vardır. Kenetli kübik spline kullanan iki tanesi ve beşli spline kullanan diğer ikisi. Her bir spline türü için bir yörünge oluşturmanın iki yolu vardır. En kolay yöntemler, Pose2d nesnelerini kabul eden aşırı yüklemelerdir.

Kelepçeli kübik eğriler için bu yöntem, biri başlangıç yol noktası ve diğeri bitiş yol noktası için olmak üzere iki Pose2d nesnesini kabul eder. Yöntem, iç yol noktalarını temsil eden `` Translation2d `` nesnelerinin bir vektörünü alır. Bu iç yol noktalarındaki başlıklar, sürekli eğriliği sağlamak için otomatik olarak belirlenir. Beşli spline'lar için, yöntem basitçe `` Pose2d `` nesnelerinin bir listesini alır ve her bir `` Pose2d `` , bir noktayı ve alandaki başlığı temsil eder.

Daha karmaşık aşırı yük, spline'lar için "kontrol vektörlerini-control vectors" kabul eder. Bu yöntem, her noktada teğet vektörün büyüklüğünü kontrol edebileceğiniz Pathweaver ile yörüngeler oluştururken kullanılır. `` ControlVector `` sınıfı iki `` çift-double `` diziden oluşur. Her dizi bir boyutu (x veya y) temsil eder ve onun elemanları bu noktadaki türevleri temsil eder. Örneğin, `` x `` dizisinin 0 ögesindeki değer x koordinatını (0. türev), 1. ögedeki değer x boyutundaki 1. türevi temsil eder ve bu böyle devam eder.

Sabitlenmiş kübik spline'lar kullanılırken, dizinin uzunluğu 2 (0. ve 1. türevler) olmalıdır, oysa beşli spline'lar kullanılırken, dizinin uzunluğu 3 (0., 1. ve 2. türev) olmalıdır. Tam olarak ne yaptığınızı bilmiyorsanız, yörüngeleri manuel olarak oluşturmak için ilk ve daha basit yöntem ŞİDDETLE önerilir. (yani Pathweaver JSON dosyalarını kullanmadığınızda).

2018 oyunu FIRST Power Up için sabitlenmiş kübik eğriler kullanarak bir yörünge oluşturmanın bir örneği:

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);
    }
}
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
}  
}
```

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  
    )
```

Not: The Java code utilizes the [Units](#) utility, for easy unit conversions.

Not: 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(traj)` function. C++/Python users can simply add (+) the two trajectories together.

Uyarı: It is up to the user to ensure that the end of the initial and start of the appended trajectory match. It is also the user's responsibility to ensure that the start and end velocities of their trajectories match.

JAVA

```
var trajectoryOne =
TrajectoryGenerator.generateTrajectory(
    new Pose2d(0, 0, Rotation2d.fromDegrees(0)),
    List.of(new Translation2d(1, 1), new Translation2d(2, -1)),
    new Pose2d(3, 0, Rotation2d.fromDegrees(0)),
    new TrajectoryConfig(Units.feetToMeters(3.0), Units.feetToMeters(3.0)));

var trajectoryTwo =
TrajectoryGenerator.generateTrajectory(
    new Pose2d(3, 0, Rotation2d.fromDegrees(0)),
    List.of(new Translation2d(4, 4), new Translation2d(6, 3)),
    new Pose2d(6, 0, Rotation2d.fromDegrees(0)),
    new TrajectoryConfig(Units.feetToMeters(3.0), Units.feetToMeters(3.0)));

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 Yörünge Kısıtlamaları

Önceki makale 'de, yörüngeleri oluştururken hiçbir özel kısıtlamanın eklenmediğini fark etmiş olabilirsiniz. Özel kısıtlamalar, kullanıcıların konuma ve eğriliğe bağlı olarak yörünge boyunca bulunan noktalarda hız ve ivmeye daha fazla kısıtlama getirmesine olanak tanır.

Örneğin, özel bir kısıtlama, belirli bir bölgede yörünge hızını belirli bir eşiğin altında tutabilir veya stabilite amacıyla robotu yakın dönüşlerde yavaşlatabilir.

WPILib Tarafından Sağlanan Kısıtlamalar

WPILib, kullanıcıların yörüngeler oluştururken kullanabilecekleri önceden tanımlanmış bir dizi kısıtlama içerir. WPILib tarafından sağlanan kısıtlamaların listesi aşağıdaki gibidir:

- **CentripetalAccelerationConstraint:** Robot yörünge boyunca ilerlerken merkezci ivmesini sınırlar. Bu, robotun sıkı dönüşlerde yavaşlamasına yardımcı olabilir.
- **DifferentialDriveKinematicsConstraint:** Robotun dönüşlerinin hızını, diferansiyel tahrikli bir robotun hiçbir tekerleği belirtilen maksimum hızın üzerine çıkmayacak şekilde sınırlar.
- **DifferentialDriveVoltageConstraint:** Bir diferansiyel sürücü robotunun ivmesini, komut verilen hiçbir gerilimin belirtilen maksimum değeri aşmayacak şekilde sınırlar.
- **EllipticalRegionConstraint:** Sahada yalnızca eliptik bir bölgede bir kısıtlama uygular.
- **MaxVelocityConstraint:** Bir maksimum hız kısıtlaması uygular. Bu, robotun hızını yalnızca belirli bir bölgede sınırlamak için **EllipticalRegionConstraint** veya **RectangularRegionConstraint** ile oluşturulabilir.
- **MecanumDriveKinematicsConstraint:** Robotun dönüş hızını, mecanum sürücülü bir robotun hiçbir tekerleği belirtilen maksimum hızın üzerine çıkmayacak şekilde sınırlar.

- **RectangularRegionConstraint:** Alan üzerinde sadece dikdörtgen bir bölgede bir kısıtlama uygular.
- **SwerveDriveKinematicsConstraint:** Robotun dönüşler sırasındaki hızını, dönüşlü bir robotun hiçbir tekerleği belirtilen maksimum hızın üzerine çıkmayacak şekilde sınırlar.

Not: **DifferentialDriveVoltageConstraint**, yalnızca teorik gerilim komutlarının *feedforward model* kullanarak belirtilen maksimum değeri aşmamasını sağlar. Robot izleme sırasında referanstan saparsa, komut verilen voltaj belirtilen maksimumdan daha yüksek olabilir.

Özel Bir Kısıtlama Oluşturma

Kullanıcılar, **TrajectoryConstraint** arayüzünü uygulayarak kendi kısıtlamalarını oluşturabilirler.

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 yöntemi, herhangi bir kısıtlama olmaksızın yörüngenin verilen poz, eğrilik ve orijinal hızı için izin verilen maksimum hızı döndürmelidir. MinMaxAcceleration methodu, verilen pozda, eğrilik ve kısıtlanmış hız için izin verilen minimum ve maksimum hızlanmayı döndürmelidir.

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 Yörüngeleri Manipüle Etmek

Bir yörünge oluşturulduktan sonra, belirli yöntemleri kullanarak ondan bilgi alabilirsiniz. Bu yöntemler, bu yörüngeleri izlemek için kod yazarken faydalı olacaktır.

Yörüngenin Toplam Süresini Elde Etmek

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()
```

Yörüngeyi Örnekleme

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)
```

`` Yörünge :: Örnek`` yapısı, örnek nokta hakkında birkaç parça bilgi içerir:

- `t`: Yörünge başlangıcından örnekleme noktasına kadar geçen süre.
- `velocity`: Numune noktasındaki hız.
- `acceleration`: Örnek noktadaki ivme.
- ``pose``: Örnek noktadaki poz (x, y, başlık).
- `curvature`: Numune noktasındaki eğrilik (yörünge boyunca mesafeye göre yön değişim oranı).

Not: Numune noktasındaki açısal hız, hızı eğrilik ile çarparak hesaplanabilir.

Yörünge Tümü Durumlarını Alma (Gelişmiş)

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 Yörüngeleri Dönüştürmek

Yörüngeler bir koordinat sisteminden diğerine dönüştürülebilir ve bir koordinat sistemi içinde “relativeTo” ve “transformBy” yöntemleri kullanılarak taşınabilir. Bu yöntemler, uzayda yörüngeleri hareket ettirmek veya başka bir referans çerçevesinde zaten var olan bir yörüngeyi yeniden tanımlamak için kullanışlıdır.

Not: Bu yöntemlerin hiçbirisi orijinal yörüngeyi şeklini değiştirmez.

“relativeTo” Metodu

“relativeTo” metodu, başka bir referans çerçevesinde zaten mevcut bir yörüngeyi yeniden tanımlamak için kullanılır. Bu yöntem bir argüman alır: yeni koordinat sisteminin başlangıcını temsil eden mevcut koordinat sistemine göre tanımlanan bir poz (bir “Pose2d” nesnesi aracılığıyla).

Örneğin, A koordinat sisteminde tanımlanan bir yörünge, orijini A koordinat sisteminde (3, 3, 30 derece) olan B koordinat sisteminde `relativeTo` yöntemi kullanılarak yeniden tanımlanabilir.

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)
```



In the diagram above, the original trajectory (aTrajectory in the code above) has been defined in coordinate system A, represented by the black axes. The red axes, located at (3, 3) and 30° with respect to the original coordinate system, represent coordinate system B. Calling `relativeTo` on aTrajectory will redefine all poses in the trajectory to be relative to coordinate system B (red axes).

transformBy Metodu

transformBy metodu, bir koordinat sistemi içindeki bir yörüngeyi hareket ettirmek (yani çevirmek ve döndürmek) için kullanılabilir. Bu yöntem bir argüman alır: yörüngeyi mevcut başlangıç konumunu aynı yörüngeyi istenen bir başlangıç konumuna eşleyen bir dönüşüm (bir Transform2d nesnesi aracılığıyla).

Örneğin, transformBy yöntemini kullanarak (2, 2, 30 derece) 'de başlayan bir yörüngeyi (4, 4, 50 derece) başlatması için dönüştürmek isteyebilirsiniz.

JAVA

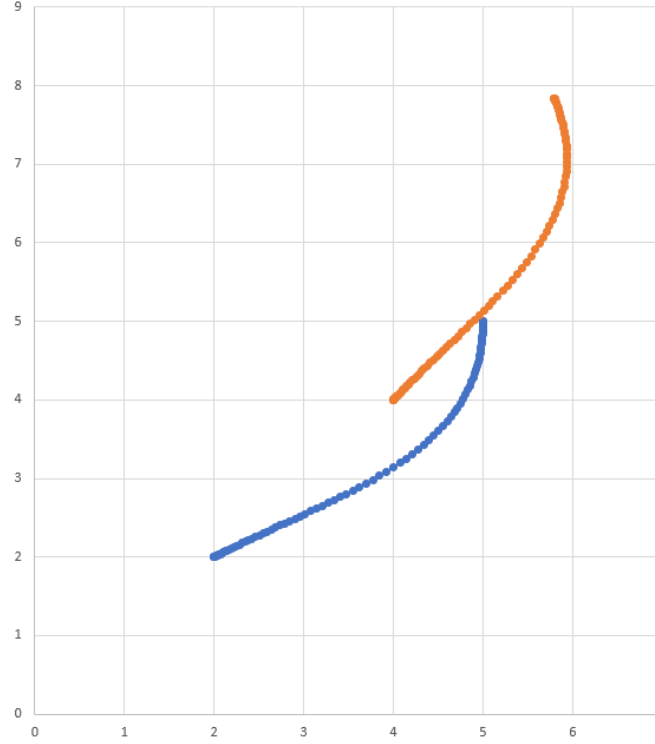
```
Transform2d transform = new Pose2d(4, 4, Rotation2d.fromDegrees(50)).minus(trajectory.  
↪getInitialPose());  
Trajectory newTrajectory = trajectory.transformBy(transform);
```

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)
```



Yukarıdaki diyagramda, (2, 2) 'de ve 30° 'de başlayan orijinal yörünge mavi olarak görülebilir. Yukarıdaki dönüşümü uyguladıktan sonra, ortaya çıkan yörünge başlangıç konumu 50° 'de (4, 4) olarak değiştirilir. Ortaya çıkan yörünge turuncu renkte görünür.

32.7.5 Ramsete Denetleyicisi

Ramsete Kontrol Cihazı, WPILib'e yerleşik bir yörünge izleyicidir. Bu izleyici, küçük rahatsızlıklar için düzeltme ile yörüngeleri doğru bir şekilde izlemek için kullanılabilir.

Ramsete Denetleyici Nesnesinin-Object Oluşturulması

Ramsete kontrolörü, b ve $zeta$ olmak üzere iki kazançla başlatılmalıdır. Daha büyük b değerleri yakınsamayı orantılı bir terim gibi daha agresif hale getirirken, daha büyük $zeta$ değerleri yanıtta daha fazla sönümlleme sağlar. Bu kontrolör kazançları, sadece kontrolörün ayarlanan hızları nasıl çıkaracağını belirler. Robotun gerçek hız takibini ETKİLEMEZ. Bu, bu denetleyici kazançlarının genellikle robottan bağımsız olduğu anlamına gelir.

Not: b ve $zeta$ için 2.0 ve 0.7 kazançları, tüm birimler metre cinsinden olduğunda istenen sonuçları üretmek için tekrar tekrar test edilmiştir. Bu nedenle, RamseteController için sıfır bağımsız değişkenli bir kurucu, bu değerlere varsayılan kazançlarla birlikte mevcuttur.

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)
```

Ayarlanmış Hızları Alma

Ramsete kontrolörü, robot bu hızları takip ettiğinde hedef noktasına doğru bir şekilde ulaşması için “ayarlanmış hızları” geri döndürür. Denetleyici, yeni hedefle periyodik olarak güncellenmelidir. Hedef, istenen bir pose, istenen doğrusal hız ve istenen açısal hızdan oluşur. Ayrıca robotun mevcut konumu da periyodik olarak güncellenmelidir. Kontrolör, ayarlanmış doğrusal ve açısal hızı döndürmek için bu dört argümanı kullanır. Kullanıcılar, optimum yö-rünge takibini elde etmek için robotlarına bu doğrusal ve açısal hızlara kumanda etmelidir.

Not: 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)
```

Bu hesaplamalar, güncellenmiş bir robot konumu ve hedefi ile her döngü yinelemesinde gerçekleştirilmelidir.

Ayarlanmış Hızları Kullanma

Ayarlanan hızlar, bir v_x (ileri yönde doğrusal hız), bir v_y (yan yönde doğrusal hız) ve bir ω (robot çerçevesinin merkezi etrafındaki açısal hız) içeren `ChassisSpeeds` tipindedir. Ramsete denetleyicisi, holonomik olmayan robotlar (yanlara doğru hareket edemeyen robotlar) için bir denetleyici olduğundan, ayarlanmış hız nesnesinin sıfır v_y değeri vardır.

Döndürülen ayarlanmış hızlar, aktarma sistemi tipiniz için kinematik sınıfları kullanılarak kullanılabilir hızlara dönüştürülebilir. Örneğin, ayarlanmış hızlar, bir `DifferentialDriveKinematics` nesnesi kullanılarak bir diferansiyel sürücü için sol ve sağ hızlara dönüştürülebilir.

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.

Komut Tabanlı Çerçeve - Command-Based Framework 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 [Rota Eğitimi](#).

32.7.6 Holonomik Sürücü Kontrolcüsü

Holonomik sürücü kontrolcüsü, holonomik aktarma organlarına sahip robotlar için bir yörünge izleyicidir (örn. sapma, mekanum, vb.). Bu, küçük sıkıntılar için düzeltme ile yörüngeleri doğru bir şekilde izlemek için kullanılabilir.

Holonomik Sürücü Kontrolcüsü Oluşturma

Holonomik sürücü kontrolcüsü, 2 PID kontrolcüsü ve 1 profilli PID kontrolcüsü ile somutlaştırılmalıdır.

Not: PID kontrolü hakkında daha fazla bilgi için bakınız [WPILib’de PID Kontrolü](#).

2 PID kontrolcü, sırasıyla alana-göre x ve y yönlerindeki hatayı düzeltmesi gereken kontrolcülerdir. Örneğin, ilk 2 bağımsız değişken sırasıyla PIDController (1, 0, 0) ve PIDController (1.2, 0, 0) ise, holonomik sürücü kontrolcüsü; x yönündeki her hata metresi için x’ e saniyede ek bir metre ekleyecektir ve y yönündeki her hata metresi için y’ ye saniyede ek 1,2 metre ekleyecektir.

Son parametre, robotun dönüşü için bir ProfiledPIDController’ dır. Bir holonomik aktarma organının dönüş dinamikleri x ve y yönlerindeki hareketten ayrıldığından, kullanıcılar bir yörüngeyi izlerken özel yön referansları ayarlayabilir. Bu rota referansları, ``ProfiledPIDController içinde ayarlanan parametrelere göre profilelenmiştir.

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.
```

Ayarlanmış Hızları Alma

Holonomik sürücü kontrolcüsü, robot bu hızları takip ettiğinde hedef noktasına doğru bir şekilde ulaşacak şekilde “ayarlanmış hızları” geri döndürür. Kontrolcü, yeni hedefle periyodik olarak güncellenmelidir. Hedef, istenen bir poz, doğrusal hız ve rotadan oluşur.

Not: 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 trajectory 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)
)
```

Ayarlanmış Hızları Kullanma

Ayarlanan hızlar ChassisSpeeds tipindedir, bir `` vx `` (ileri yönde doğrusal hız) içerir, a `` vy `` (yan yöndeki doğrusal hız) ve bir `` omega `` (robot çerçevesinin merkezi etrafındaki açılma hızı).

Döndürülen ayarlanmış hızlar, aktarma sistemi tipiniz için kinematik sınıfları kullanılarak kullanılabilir hızlara dönüştürülebilir. Aşağıdaki örnek kodda, bir swerve sürücü robotu varsayacağız; bununla birlikte, kinematik kodu, MecanumDriveKinematics kullanımı haricinde bir mecanum tahrik robotu için tamamen aynıdır.

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)
```

Bu sapma modülü durumları, hala hızlar ve açılar olduğundan; bu hızları ve açıları ayarlamak için PID kontrolcülerini kullanmanız gerekecektir.

32.7.7 Sorun giderme

Tam Arızaların Giderilmesi

Robotunuzun tamamen yanlış bir şey yapmasına neden olabilecek birkaç şey vardır. Aşağıdaki kontrol listesi bazı yaygın hataları kapsamaktadır.

- Robotum hareket etmiyor.
 - Gerçekten motorlarınıza çıktı mı veriyorsunuz?
 - Sürücü istasyonuna bir `MalformedSplineException` yazdırılıyor mu? Evet ise, aşağıdaki `MalformedSplineException` bölümüne gidin.
 - Yörüngeğiniz çok kısa mı yoksa yanlış birimlerde mi?
- Robotum, yörüngeyi diğer yöne bakacak şekilde sürmek için dönüyor.
 - Yörüngeinizin başlangıç ve bitiş yörüngeleri yanlış mı?
 - Robotunuzun jiroskopu yanlış yöne mi sıfırlanıyor?
 - *Ters bayrağı yanlış mı ayarlanmış?*
 - Gyro açılarınız saat yönünde pozitif mi? Eğer öyleyse, onları reddetmelisiniz.
- Robotum dönmesi gerekmesine rağmen sadece düz bir çizgide ilerliyor.
 - Gyro'nuz doğru bir şekilde kurulmuş ve iyi veriler döndürüyor mu?
 - Gyro başlığını odometri nesnenize doğru birimlerle mi geçiriyorsunuz?
 - İz genişliğiniz doğru mu? Doğru birimlerde mi?
- Sürücü istasyonunda bir `MalformedSplineException` çıktısı alıyorum ve robot hareket etmiyor.

- *Ters bayrağı yanlış mı ayarlanmış?*
- Yaklaşık olarak zıt başlıklarla birbirine çok yakın iki ara noktanız var mı?
- Aynı (veya neredeyse aynı) koordinatlara sahip iki ara noktanız var mı?
- Robotum çok uzağa gidiyor.
 - Kodlayıcı birimi dönüştürmeleriniz doğru ayarlanmış mı?
 - Kodlayıcılarınız bağlı mı?
- Robotum çoğunlukla doğru olanı yapıyor, ancak biraz yanlış.
 - Sonraki bölüme gidin.

Kötü Performans Sorunlarını Giderme

Not: Bu bölüm çoğunlukla, derleme hataları, dönen ve yanlış yöne giden robotlar veya `MalformedSplineException` lar gibi yıkıcı arızalar değil, bir metre hata gibi zayıf yörünge izleme performansının giderilmesiyle ilgilidir.

Not: Bu bölüm diferansiyel tahrikli robotlar için tasarlanmıştır, ancak fikirlerin çoğu swerve veya mecanum sürüşü yönlendirmek için uyarlanabilir.

Zayıf yörünge izleme performansının giderilmesi zor olabilir. Yörünge oluşturucu ve takipçinin kullanımı kolay ve kutudan çıkar çıkmaz performans göstermesi amaçlansa da, robotunuzun olması gerektiği yere tam olarak varmadığı durumlar vardır. Yörünge oluşturucu ve takipçilerin ayarlanması gereken çok sayıda düğmesi ve birçok hareketli parçası vardır, bu nedenle, özellikle robotun genel davranışından yörünge sorunlarının kaynağını bulmak zor olduğu için nereden başlayacağını bilmek zor olabilir.

Yörünge oluşturucu katmanını ve hatalı davranan takipçileri bulmak çok zor olabileceğinden, genel olarak kötü izleme performansı için sistematik, katman katman bir yaklaşım önerilir (örneğin, robot birkaç feet veya yirmi dereceden fazla uzakta) . Aşağıdaki adımlar, yapmanız gereken sıraya göre listelenmiştir; Farklı adımların etkilerini birbirinden ayırabilmeniz için bu sırayı takip etmeniz önemlidir.

Not: The below examples put diagnostic values onto *NetworkTables*. The easiest way to graph these values is to *use Shuffleboard's graphing capabilities*.

Odometriyi Doğrula

Odometreniz kötüyse, Ramsete kontrol cihazınız yanlış davranabilir, çünkü robotunuzun hedef hızlarını, odometrenizin robotun nerede olduğunu düşündüğüne bağlı olarak değiştirir.

Not: *Sending your robot pose and trajectory to field2d* can help verify that your robot is driving correctly relative to the robot trajectory.

1. Her odometre güncellemesinden sonra robotunuzun konumunu kaydetmek için kodunuzu ayarlayın:

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. Robotunuza paralel bir mezura yerleştirin ve robotunuzu şerit metre boyunca yaklaşık bir metre dışarı doğru itin. Y eksenini boyunca bir şerit metre yerleştirin ve baştan başlayın, robotunuzu X eksenini boyunca bir metre ve Y eksenini boyunca bir metre kaba bir yay çizerek itin.
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.

Feedforward - ileribesleme'yi Doğrula

İleri beslemeniz kötüyse, robotun her iki tarafındaki P denetleyicileri de takip etmeyecek ve `DifferentialDriveVoltageConstraint`, robotunuzun hızlanmasını doğru bir şekilde sınırlamayacaktır. İleriye doğru ilerlemeyi izole edip test edebilmek için çoğunlukla tekerlek P kontrol cihazlarını kapatmak istiyoruz.

1. İlk olarak, her tekerlek için P kontrolörünü devre dışı bırakmalıyız. Her kontrolör için P kazancını 0'a ayarlayın. `RamseteCommand` örneğinde, `kPDriveVel` değerini 0 olarak ayarlarsınız:

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,
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
...
);
```

3. Son olarak, istenen tekerlek hızını ve gerçek tekerlek hızını kaydetmemiz gerekir (Shuffleboard kullanıyorsanız veya grafik yazılımınız bu özelliğe sahipse, gerçek ve istenen hızları aynı grafiğe koymalısınız):

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);
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

frc2::RamseteCommand ramseteCommand(
    exampleTrajectory, [this]() { return m_drive.GetPose(); },
    frc::RamseteController(AutoConstants::kRamseteB,
                          AutoConstants::kRamseteZeta),
    frc::SimpleMotorFeedforward<units::meters>(
        DriveConstants::ks, DriveConstants::kv, DriveConstants::ka),
    DriveConstants::kDriveKinematics,
    [this] { return m_drive.GetWheelSpeeds(); }, leftController,
    rightController,
    [=](auto left, auto right) {
        auto leftReference = leftRef;
        auto leftMeasurement = leftMeas;
        auto rightReference = rightRef;
        auto rightMeasurement = rightMeas;

        m_drive.TankDriveVolts(left, right);

        leftMeasurement.SetDouble(m_drive.GetWheelSpeeds().left.value());
        leftReference.SetDouble(leftController.GetSetpoint());

        rightMeasurement.SetDouble(m_drive.GetWheelSpeeds().right.value());
        rightReference.SetDouble(rightController.GetSetpoint());
    },
    {&m_drive});

```

4. Run the robot on a variety of trajectories (curved and straight line), and check to see if the actual velocity tracks the desired velocity by looking at graphs from NetworkTables.
5. If the desired and actual are off by *a lot* then you should check if the wheel diameter and encoderEPR you used for system identification were correct. If you've verified that your units and conversions are correct, then you should try recharacterizing on the same floor that you're testing on to see if you can get better data.

P Kazancını Doğrula

Önceki adımı tamamladıysanız ve sorun ortadan kalktıysa, sorununuz muhtemelen sonraki adımlardan birinde bulunabilir. Bu adımda, tekerlek P kontrol cihazlarınızın iyi ayarlanmış olduğunu doğrulayacağız. Java kullanıyorsanız, Ramsete'yi kapatmak istiyoruz, böylece PF denetleyicilerimizi kendi başlarına görüntüleyebilelim.

1. ****P kazancının önceki sıfır olmayan değere ayarlanması dışında****, gerçek ve istenen hızı (ve Java kullanıyorsanız Ramsete'yi devre dışı bırakan kodu) günlüğe kaydeden önceki adımdaki tüm kodu tekrar kullanmanız gerekir.
2. Robotu çeşitli yörüngelerde tekrar çalıştırın ve gerçek ve istenen grafiklerin iyi göründüğünü kontrol edin.
3. Grafikler iyi görünmüyorsa (yani gerçek hız, istenenden çok farklıysa), P kazancınızı ayarlamayı ve test yörüngelerini yeniden çalıştırmayı denemelisiniz.

Kısıtlamaları Kontrol Et

Not: Bu adım için P kazancınızın sıfır olmadığından ve önceki adımlarda eklenen günlük kodunun hala bulunduğundan emin olun. Java kullanıyorsanız, Ramsete'yi devre dışı bırakmak için kodu kaldırmalısınız.

Doğruluk sorununuz önceki adımların tümünde devam ettiyse, kısıtlamalarınızla ilgili bir sorununuz olabilir. Aşağıda, farklı mevcut kısıtlamaların yetersiz ayarlandığında göstereceği belirtilerin bir listesi bulunmaktadır.

Her seferinde bir kısıtlamayı test edin! Diğer kısıtlamaları kaldırın, kalan kısıtlamalardan birini ayarlayın ve kullanmak istediğiniz her kısıtlama için bu işlemi tekrarlayın. Aşağıdaki kontrol listesi, bir seferde yalnızca bir kısıtlama kullandığınızı varsayar.

- **DifferentialDriveVoltageConstraint:**
 - Robotunuz çok yavaş hızlanırsa, bu kısıtlama için maksimum voltajın çok düşük olması mümkündür.
 - If your robot doesn't reach the end of the path then your system identification data may be problematic.
- **DifferentialDriveKinematicsConstraint:**
 - Robotunuz yanlış istikamette biterse, maksimum aktarma organı yan hızının çok düşük veya çok yüksek olması mümkündür. Bunu söylemenin tek yolu maksimum hızı ayarlamak ve ne olacağını görmektir.
- **CentripetalAccelerationConstraint:**
 - Robotunuz yanlış istikamette biterse, suçlu bu olabilir. Robotunuz yeterince dönmüyorsa, maksimum merkezci ivmeyi artırmalısınız, ancak hızlı dönüşler hızla dönmüyorsa, maksimum merkezci ivmeyi-maximum centripetal azaltmalısınız.

Yörünge Yol Noktalarını Kontrol Edin

Yörüngенizin kendisinin pek sürülebilir olmaması mümkündür. Keskin dönüşleri azaltmak için ara noktaları (ve varsa ara noktalardaki yönleri) hareket ettirmeyi deneyin.

32.8 State-Space ve WPILib ile Model Tabanlı Kontrol

Bu bölüm, state-space kontrolü için WPILib desteğine bir giriş seviyesinde bilgi sağlar ve bunu açıklar.

32.8.1 Durum Uzayı - State-space Kontrolüne Giriş

Not: Bu makale, Tyler Veness'in izniyle, [FRC](#) 'deki Controls Engineering'den alınmıştır.

PID'den Model Tabanlı-Model-Based Kontrol

PID denetleyicileri ayarlarken, mevcut, geçmiş ve gelecekle ilgili denetleyici parametreleriyle uğraşmaya odaklanıyoruz: temel sistem durumlarından ziyade *error* (P, I ve D terimleri). Bu yaklaşım pek çok durumda işe yarasa da, eksik bir dünya görüşüdür.

Model tabanlı kontrol, kontrol etmeye çalıştığımız *system* (mechanism) için doğru bir model geliştirmeye odaklanır. Bu modeller keyfi bir orantılı *gains* yerine sistemin fiziksel yanıtlarına dayalı olarak geribildirim denetleyicileri tarafından seçilen testler yoluyla elde edilen *gains* bilgilendirmeye yardımcı olur. Bu, sadece bir sistemin nasıl tepki vereceğini önceden tahmin etmemize değil, aynı zamanda kontrolörlerimizi fiziksel bir robot olmadan test etmemize ve basit hataların ayıklanmasında zamandan tasarruf etmemize olanak tanır.

Not: 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.

Kelime bilgisi

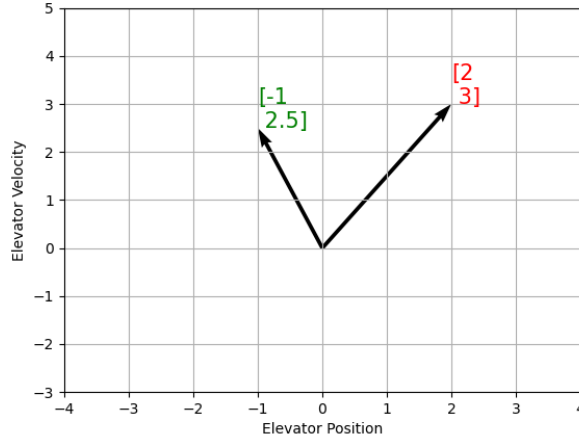
For the background vocabulary that will be used throughout this article, see the [Glossary](#).

Doğrusal Cebire Giriş

Lineer Cebirin temel kavramlarına kısa ve sezgisel bir giriş için, [3Blue1Brown's Essence of linear algebra series](#) in 1'den 4'e kadar olan bölümlerini öneriyoruz (Vektörler, bunlar ne bile ?, Doğrusal kombinasyonlar, aralık ve taban vektörleri, Doğrusal dönüşümler ve matrisler ve kompozisyon olarak Matris çarpımı).

Durum Uzayı nedir?

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]:



Bu görüntüde, durum uzayındaki durumları temsil eden vektörler oklardır. Şu andan itibaren bu vektörler, vektörün ucundaki bir nokta ile temsil edilecek, ancak vektörün geri kalanının hala orada olduğunu unutmayın.

state, :term:`girişleri` <input> :term:`outputs` <output> çıkışlarına ek olarak vektörler olarak temsil edilir. Mevcut durumlardan ve girdilerden durumdaki değişime haritalama bir denklemler sistemi olduğundan, bunu matris biçiminde yazmak doğaldır. Bu matris denklemi durum uzayı gösteriminde yazılabilir.

Durum Uzayı Gösterimi nedir?

Durum uzayı gösterimi, bir sistemin zaman içinde nasıl gelişeceğini açıklayan bir dizi matris denklemdir. Bu denklemler, $\dot{\mathbf{x}}$, and the *output* \mathbf{y} , değişikliklerini mevcut durum vektörünün doğrusal kombinasyonlarıyla ilişkilendirir \mathbf{x} and *input* vector \mathbf{u} .

Durum alanı kontrolü, sürekli zamanlı ve ayrık zamanlı sistemlerle başa çıkabilir. Sürekli zaman durumunda, sistemin durumunun değişim oranı: $\dot{\mathbf{x}}$, geçerli durumun doğrusal bir kombinasyonu olarak ifade edilir: $\dot{\mathbf{x}}$ and input \mathbf{u} .

Buna karşılık, ayrık zamanlı sistemler, bir sonraki zaman adımımızda sistemin durumunu şu anki duruma göre ifade eder. \mathbf{x}_{k+1} ve mevcut duruma göre: \mathbf{x}_k ve girdi: \mathbf{u}_k , burada k geçerli zaman adımımızdır ve $k + 1$ sonraki zaman adımımızdır.

Hem sürekli hem de ayrık zaman formlarında, *output* vector \mathbf{y} , current: term: "state" ve:term:input ifadelerinin doğrusal bir kombinasyonu olarak ifade edilir. Çoğu durumda, çıktı, sistem durumunun bir alt kümesidir ve akım girişinden hiçbir katkısı yoktur.

Sistemleri modellerken, ilk olarak sürekli zaman gösterimini türetiyoruz çünkü hareket denklemleri doğal olarak bir sistemin durumunun değişim hızı olarak mevcut durumu ve girdi-

lerinin doğrusal bir kombinasyonu olarak yazılır. Bu gösterimi robot üzerinde ayırık zamana dönüştürüyoruz çünkü sistemi sürekli yerine orada ayrı zaman adımlarında güncelliyoruz.

Aşağıdaki iki denklem seti, sürekli zaman ve ayırık zaman durum uzayı gösteriminin standart biçimidir:

$$\begin{aligned}\text{Continuous: } \dot{\mathbf{x}} &= \mathbf{Ax} + \mathbf{Bu} \\ \mathbf{y} &= \mathbf{Cx} + \mathbf{Du}\end{aligned}$$

$$\begin{aligned}\text{Discrete: } \mathbf{x}_{k+1} &= \mathbf{Ax}_k + \mathbf{Bu}_k \\ \mathbf{y}_k &= \mathbf{Cx}_k + \mathbf{Du}_k\end{aligned}$$

A	system matrix	x	state vector
B	input matrix	u	input vector
C	output matrix	y	output vector
D	feedthrough matrix		

Sürekli zaman durum uzay sistemi, ayrıklaştırma adı verilen bir süreçle ayırık zamanlı bir sisteme dönüştürülebilir.

Not: Ayırık zamanlı formda, sistemin durumu güncellemeler arasında sabit tutulur. Bu, rahatsızlıklara yalnızca durum tahminimiz güncellendiğinde tepki verebileceğimiz anlamına gelir. Tahminimizi daha hızlı güncellemek, performansın bir noktaya kadar iyileştirilmesine yardımcı olabilir. WPILib'in Notifier sınıfı, ana robot döngüsünden daha hızlı güncellemeler isteniyorsa kullanılabilir.

Not: Bir sistemin sürekli zaman ve ayırık zaman matrisleri A, B, C ve D aynı adlara sahipken, bunlar eşdeğer değildir. Sürekli zaman matrisleri, durum değişim oranını açıklarken: \mathbf{A} , ayırık zaman matrisleri ise sistemin durumunu bir sonraki zaman adımıyla mevcut durum ve girdinin bir fonksiyonu olarak tanımlar.

Önemli: WPILib'in LinearSystem'i, sürekli zaman sistem matrislerini alır ve bunları gerektiğinde dahili olarak ayırık zaman biçimine dönüştürür.

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.

Bu denklemi durum uzayı gösterimine dönüştürebiliriz. Bir durum (hız), bir *input* (voltage) ve bir *output* (velocity) olan bir sistem oluşturabiliriz. Hızın ilk türevinin ivme olduğunu hatırlayarak, denklemimizi aşağıdaki gibi yazabiliriz, hızı şu ile değiştirebiliriz: \mathbf{x} , acceleration with \mathbf{x} ve voltaj: V ile : \mathbf{u} :

$$\dot{\mathbf{x}} = \begin{bmatrix} -K_v \\ K_a \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1 \\ K_a \end{bmatrix} \mathbf{u}$$

The output and state are the same, so the output equation is the following:

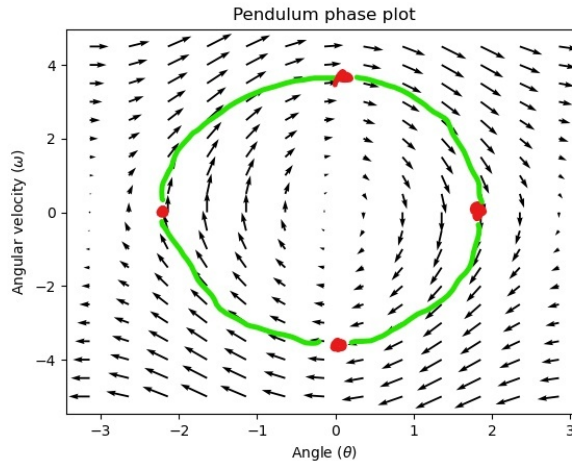
$$\mathbf{y} = [1] \mathbf{x} + [0] \mathbf{u}$$

That's it! That's the state-space model of a system for which we have the K_v and K_a constants. This same math is used in system identification to model flywheels and drivetrain velocity systems.

Durum Uzayı Yanıtlarını Görselleştirme: Faz Portresi

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 \mathbf{x} in state-space, and point in the direction of $\dot{\mathbf{x}}$, the direction that the system will evolve over time. This example shows a model of a pendulum with the states of angle and angular velocity.

Bir sistemin durum uzayında alabileceği potansiyel bir yörüngeyi izlemek için, başlamak üzere bir nokta seçin ve etrafındaki okları takip edin. Bu örnekte, $[-2, 0]$ ile başlayabiliriz. Oradan, biz dikeyde döndükçe hız artar ve salınımın zıt ucuna ulaşana kadar azalmaya başlar. Kökenle ilgili bu dönme döngüsü sonsuza kadar tekrar eder.



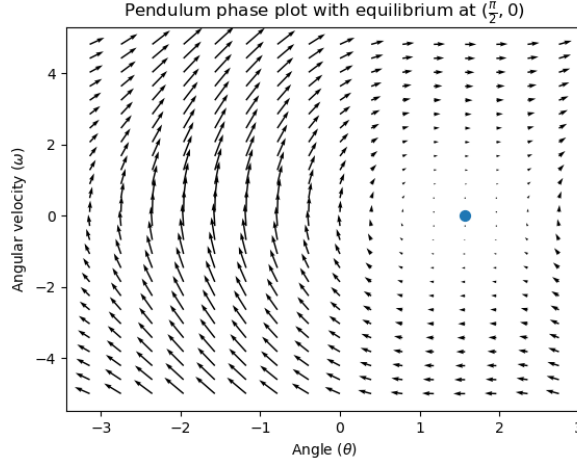
Faz portresinin kenarlarının yakınında, X ekseninin saat yönünün tersine: math: π radyan dönüşü olarak sarıldığına ve π radyanının saat yönünde bir dönüşünün aynı noktada sona ereceğine dikkat edin.

Diferansiyel denklemler ve faz portreleri hakkında daha fazla bilgi için, bkz. `3Blue1Brown Diferansiyel Denklemler videosu <https://www.youtube.com/watch?v=p_di4Zn4wz4>` - saat 15: 30'da sarkaç faz uzayını canlandırmak için harika bir iş çıkarırlar.

Feedforward'ı Görselleştirme

Bu aşama portresi, sistemin "open loop" tepkilerini, yani devletin doğal olarak gelişmesine izin verirken nasıl tepki vereceğini gösterir. Sarkacı yatay olarak dengelemek istiyorsak (at $(\frac{\pi}{2}, 0)$ in state space), bir şekilde bir kontrol uygulamamız gerekir: term: *input* Sarkacın açık döngü eğiliminin aşağı doğru sallanma eğilimine karşı koymak için. Feedforward'ın yapmaya çalıştığı budur - bunu, faz portremizin durum uzayında *reference* AMkonumunda (veya ayar noktasında) bir dengeye sahip olmasını sağlayın.

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 $\frac{T}{I}$, 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 $(\frac{\pi}{2}, 0)$:



Geri bildirim kontrolü

Bir DC motor söz konusu olduğunda, sadece matematiksel bir modelle ve sistemin tüm mevcut durumlarının bilgisiyle (örneğin, açısal hız), gelecekteki voltaj girişleri verildiğinde gelecekteki tüm durumları tahmin edebiliriz. Ancak sistem, bir yük veya beklenmedik sürtünme gibi denklemlerimiz tarafından modellenmeyen herhangi bir şekilde bozulursa, motorun açısal hızı zamanla modelden sapacaktır. Bununla mücadele etmek için, motora bir geri besleme kontrolörü kullanarak düzeltici komutlar verebiliriz.

PID kontrolörü, bir geri besleme kontrolü biçimidir. Durum uzayı kontrolü genellikle şunları kullanır *control law*, burada \mathbf{K} bazı kontrolcü *gain* matrisi, \mathbf{r} *reference* durumu ve **matematik: \mathbf{x}** durum uzayındaki mevcut durumdur. Bu iki vektör arasındaki fark $\mathbf{r} - \mathbf{x}$, *error* dır.

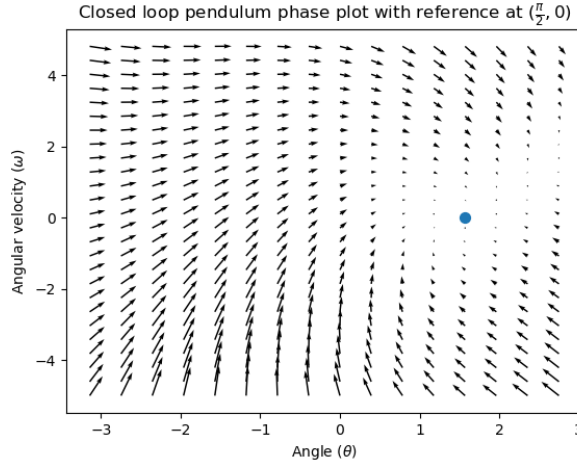
$$\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$$

Bu *control law*, sistemimizin her durumu için oransal bir denetleyicidir. Oransal denetleyiciler, sistemimizin durumunu durum uzayındaki referans durumuna çeken yazılım tanımlı yaylar oluşturur. Kontrol edilen sistemin konum ve hız durumlarına sahip olması durumunda, yukarıdaki **terim: *control law***, aynı zamanda konumu ve hız hatasını sıfıra çekmeye çalışan bir PD kontrolörü gibi davranacaktır.

Bu kontrol yasasının uygulamada bir örneğini göstereyim. Sarkaç sistemini yukarıdan kullanacağız, burada sallanan sarkaç durum uzayında orijini daire içine aldı. \mathbf{K} sıfır matrisinin (tümü sıfır olan bir matris), sıfırın P ve D kazançlarını toplama gibi olacağı durum - kontrol yok girdi uygulanabilir ve aşama portresi yukarıdakiyle aynı görünecektir.

Geri bildirim eklemek için, rasgele olarak bir \mathbf{K} of $[2, 2]$ seçiyoruz; burada sarkaç için bizim *input* açısal ivmedir. Bu \mathbf{K} , her bir konumun radyan hata için, açısal ivmenin saniyede 2 radyan kare olacağı anlamına gelir; benzer şekilde, **terim: *hata*** nın her saniyesinde her radyan

için saniyede 2 radyan kare hızlanırız. Durum uzayında bir yerden içeriye doğru bir oku takip etmeyi deneyin - başlangıçtaki koşullar ne olursa olsun, durum, saf ileri beslemeyle sonsuza kadar daire çizmek yerine referans a yerleşecektir.



Ancak, sistemimiz için bir optimal **terim: `kazanç`** matrisi K 'yi nasıl seçebiliriz? Manuel olarak şu seçimi yapabilirsek de **terim: <gain> kazanır** ve sistem yanıtını simüle edebilir veya bir PID kontrolörü gibi robot üzerinde ayarlayabilirken, modern kontrol teorisinin daha iyi bir cevabı var :Linear-Quadratic Regulator (LQR).

Linear-Quadratic Regulator / Doğrusal-Kuadratik Düzenleyici

Model tabanlı kontrol, bir başlangıç koşulu ve gelecekteki kontrol girdileri verilen bir sistemin gelecekteki durumlarını tahmin edebileceğimiz anlamına geldiğinden, matematiksel olarak en uygun olanı seçebiliriz **gain** matrix K . Bunu yapmak için, önce “iyi” veya “kötü” nün neye benzeyeceğini tanımlamalıyız K . Bunu, hata karesini ve zaman içindeki kontrol girdisini toplayarak yaparız, bu da bize kontrol yasamızın ne kadar “kötü” olacağını gösteren bir sayı verir. Bu miktarı en aza indirirsek, optimal kontrol yasasına ulaşmış olacağız.

LQR: Tanım

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 $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$.

$$J = \sum_{k=0}^{\infty} (\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k)$$

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**.

Not: LQR tasarımının \mathbf{Q} ve \mathbf{R} matrisleri ayrıklaştırmaya ihtiyaç duymaz, ancak \mathbf{K} sürekli zaman ve ayrık zaman için hesaplanır systems farklı olacaktır.

LQR: ayarlama

PID kontrolörlerinin kazançlarını değiştirilerek ayarlanabildiği gibi, kontrol yasamızın hatamızı ve girdimizi nasıl dengelediğini de değiştirmek istiyoruz. Örneğin, bir uzay gemisi belirli bir referansa ulaşmak için harcadığı yakıtı en aza indirmek isteyebilirken, yüksek hızlı bir robotik kolun bozulmalara hızlı tepki vermesi gerekebilir.

LQR'mizdeki hata ve kontrol çabasını **Q** ve **R** matrisleriyle ağırlıklandırabiliriz. Maliyet fonksiyonumuzda (kontrol yasamızın ne kadar “kötü” performans göstereceğini açıklar), **Q** ve **R**, hatamızı ve kontrol girdimizi birbirine göre ağırlıklandırır. Yukarıdaki uzay gemisi örneğinde, hatayı çok fazla cezalandırmak istemediğimizi göstermek için nispeten küçük sayılarla \mathbf{Q} kullanabiliriz, oysa **R** yakıt harcamanın istenmeyen olduğunu göstermek için büyük olabilir.

WPILib ile LQR sınıfı, istenen maksimum durum sapmalarını ve kontrol çabalarının bir vektörünü alır ve bunları dahili olarak Bryson kuralıyla tam **Q** ve **R** matrislerine dönüştürür. Bu vektörlere atıfta bulunmak için genellikle küçük harf **q** ve **r** kullanır ayrıca matrisleri belirtmek için **Q** ve **R** kullanır.

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.

Örneğin, konum ve hız durumlarına sahip bir asansör sistemi için aşağıdaki **Q** ve **R**'yi kullanabiliriz.

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
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
{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: örnek uygulama

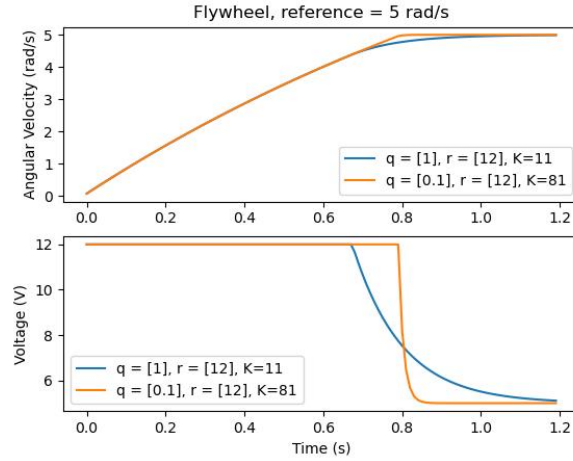
Let's apply a Linear-Quadratic Regulator to a real-world example. Say we have a flywheel velocity system determined through system identification to have $K_v = 1 \frac{\text{volts}}{\text{radian per second}}$ and $K_a = 1.5 \frac{\text{volts}}{\text{radian per second squared}}$. Using the flywheel example above, we have the following linear system:

$$\mathbf{x} = \begin{bmatrix} -K_v \\ K_a \end{bmatrix} v + \begin{bmatrix} 1 \\ K_a \end{bmatrix} V$$

[12 volts] nın $q = [0.1 \text{ rad/sec}]$, ve \mathbf{r} si ile İstenilen bir durum sapmasını (maksimum hata) keyfi olarak seçeriz. 20 ms'lik bir zaman adımı ile ayrıklaştırmadan sonra, $\mathbf{K} = 81$ nın bir *gain* kazancını buluruz. Bu \mathbf{K} *gain* volanın hızı üzerindeki bir PID döngüsünün orantılı bileşeni olarak işlev görür.

Let's adjust \mathbf{q} and \mathbf{r} . We know that increasing the \mathbf{q} elements or decreasing the \mathbf{r} elements we use to create \mathbf{Q} and \mathbf{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 \mathbf{q} from 0.1 to 1.0, our *gain* matrix \mathbf{K} drops from ~81 to ~11. Similarly, decreasing our maximum voltage r from 12.0 to 1.2 decreases \mathbf{K} .

Aşağıdaki grafik, volanın açısal hızını ve zaman içinde uygulanan voltajı iki farklı *gain* ile gösterir. Daha yüksek bir *gain* in sistemin referansa daha hızlı ($t = 0,8$ saniyede) ulaşmasını sağlarken, motorumuzu daha uzun süre 12V'de doygun tutacağını görebiliriz. Bu, bir PID kontrol cihazının P kazancını ~8x faktörü ile artırmakla tamamen aynıdır.



LQR ve Ölçüm Gecikme Telafisi

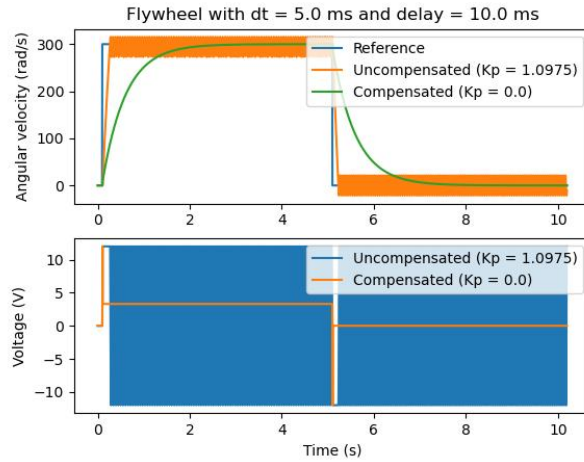
Çoğu zaman, sensörlerimizin ölçümleriyle ilişkili bir gecikmeye sahiptir. Örneğin CAN üzerinden SPARK MAX motor kontrolörü, hız ölçümleriyle ilişkili 30 ms'ye kadar gecikmeye sahip olabilir.

Bu gecikme, geri bildirim denetleyicimizin geçmişteki durum tahminlerine göre voltaj komutları oluşturacağı anlamına gelir. Bu, genellikle aşağıdaki grafikte gösterildiği gibi sistemimize kararsızlık ve salınımlar getirme etkisine sahiptir.

However, we can model our controller to control where the system's *state* is delayed into the future. This will reduce the LQR's *gain* matrix **K**, trading off controller performance for stability. The below formula, which adjusts the *gain* matrix to account for delay, is also used in system identification.

$$\mathbf{K}_{\text{compensated}} = \mathbf{K} \cdot (\mathbf{A} - \mathbf{BK})^{\text{delay}/dt}$$

A - BK ile **K** çarpmak temelde kazançları bir zaman adımı ilerletir. Bu durumda, kazançları ölçüm gecikmesiyle ilerletmek için $(\mathbf{A} - \mathbf{BK})^{\text{delay}/dt}$ ile çarpıyoruz.



Not: Bu, **K** 'yi sıfıra düşürerek geribildirim kontrolünü etkin bir şekilde devre dışı bırakma etkisine sahip olabilir.

Not: SPARK MAX motor kontrolörü, 19,5 ms gecikmeli 40-kademe FIR filtresi kullanır ve durum çerçeveleri varsayılan olarak her 20 ms'de bir gönderilir.

Aşağıdaki kod, sensör giriş gecikmeleri için LQR kontrol cihazının K kazancının nasıl ayarlanacağını gösterir:

JAVA

```
// Adjust our LQR's controller for 25 ms of sensor input delay. We
// provide the linear system, discretization timestep, and the sensor
// input delay as arguments.
controller.latencyCompensate(elevatorSystem, 0.02, 0.025);
```

C++

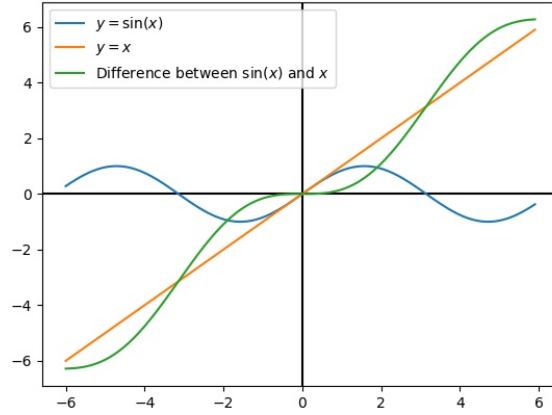
```
// Adjust our LQR's controller for 25 ms of sensor input delay. We
// provide the linear system, discretization timestep, and the sensor
// input delay as arguments.
controller.LatencyCompensate(elevatorSystem, 20_ms, 25_ms);
```

PYTHON

```
# Adjust our LQR's controller for 25 ms of sensor input delay. We
# provide the linear system, discretization timestep, and the sensor
# input delay as arguments.
controller.latencyCompensate(elevatorSystem, 0.020, 0.025)
```

Doğrusallaştırma

Doğrusallaştırma, doğrusal olmayan fonksiyonları ve durum uzay sistemlerini doğrusal olanları kullanarak kestirmek için kullanılan bir araçtır. İki boyutlu uzayda doğrusal fonksiyonlar düz çizgilerdir, doğrusal olmayan fonksiyonlar ise eğri olur. Doğrusal olmayan bir fonksiyonun yaygın bir örneği ve buna karşılık gelen doğrusal yaklaşımı şöyledir $y = \sin x$. Bu fonksiyon şu şekilde yaklaştırılabilir $y = x$ sıfıra yakın. Bu yaklaşım, $x = 0$ a yakın iken doğrudur, ancak doğrusallaştırma noktasından uzaklaştıkça doğruluğu kaybeder. Örneğin, $\sin x \approx x$ yaklaşımı $y = 0$ 0,5 radyan içinde 0,02 aralığında doğrudur, ancak bundan sonra doğruluğu hızla kaybeder. Aşağıdaki resimde şunu görüyoruz $y = \sin x$, $y = x$ ve yaklaşık ile gerçek değeri arasındaki fark $\sin x$ at x .



Doğrusal olmayan dynamics” ile durum uzay sistemlerini de doğrusallaştırabiliriz. Bunu durum uzayında bir nokta \mathbf{x} seçerek ve bunu doğrusal olmayan fonksiyonlarımızın girdisi olarak kullanarak yapıyoruz. Yukarıdaki örnekte olduğu gibi, bu, sistemin doğrusallaştırıldığı noktaya yakın durumlar için iyi çalışır, ancak bu durumdan hızla uzaklaşabilir.

32.8.2 State-Space Kontrolörü Çözüm Yolu

Not: Bu öğreticiyi izlemeden önce, okuyucuların bir durum-uzay kontrolüne girişi okumaları önerilir [Durum Uzayı - State-space Kontrolüne Giriş](#)

Bu öğreticinin amacı, bir volan için bir durum-uzay kontrolörünün uygulanmasına ilişkin “uçtan uca” talimatlar sağlamaktır. Bu öğreticiyi takip ederek okuyucular şunları öğreneceklerdir:

1. Create an accurate state-space model of a flywheel using *system identification* or *CAD* software.
2. Enkoder hızı ölçümlerini gecikmeden filtrelemek için bir Kalman Filtresi uygulayın.
3. Bir *LQR* geri besleme denetleyicisi uygulayın; bu, model tabanlı ileri besleme ile birleştirildiğinde, çarkı a *reference* “referans” a sürmek için voltaj *inputs* oluşturur.

Bu öğretici, çok fazla programlama uzmanlığı olmayan ekipler için ulaşılabilir olması amaçlanmıştır. WPILib kitaplığı, durum-uzay kontrol özelliklerinin uygulanma biçiminde önemli bir esneklik sunarken, bu eğitimde ana hatları verilen uygulamayı yakından takip etmek, ekiplere çeşitli durum-uzay sistemleri için yeniden kullanılabilir temel bir yapı sağlamalıdır.

The full example is available in the state-space flywheel ([Java/C++/Python](#)) and state-space flywheel system identification ([Java/C++/Python](#)) example projects.

State-Space Control-Durum Uzaýı Kontrolü Neden Kullanılır?

State-Space Control-Durum-uzay kontrolü, sistemimizin doğru bir modelini oluşturmaya odaklandığından, *model* will respond to control inputs 1 e nasıl tepki vereceğini doğru bir şekilde tahmin edebiliriz. Bu, mekanizmalarımızı fiziksel bir robota erişim olmadan simüle etmemize ve aynı zamanda kolayca seçim yapmamıza olanak tanır *gains* iyi çalışacağını bildiğimiz kazançlardır. Bir modele sahip olmak, sensör okumalarını en iyi şekilde filtrelemek için Kalman Filtreleri gibi gecikmesiz filtreler oluşturmamıza da olanak tanır.

Flywheel-Volanımızın Modellenmesi

Recall that continuous state-space systems are modeled using the following system of equations:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{Ax} + \mathbf{Bu} \\ \mathbf{y} &= \mathbf{Cx} + \mathbf{Du}\end{aligned}$$

Where \dot{x} is the rate of change of the *system's state*, \mathbf{x} is the system's current state, \mathbf{u} is the *input* to the system, and \mathbf{y} is the system's *output*.

Let's use this system of equations to model our flywheel in two different ways. We'll first model it using *system identification* using the SysId toolsuite, and then model it based on the motor and flywheel's *moment of inertia*.

Durum uzaýı sistemimizi oluşturma'nın ilk adımı, sistemimizin durumlarını seçmektir. Devlet olarak istediğimiz her şeyi seçebiliriz - istersek tamamen ilgisiz durumları seçebiliriz - ama önemli olan durumları seçmeye yardımcı olur. Bizim durumumuzdaki *hidden states* ifadesini dahil edebiliriz (sadece konumunu ölçebilseydik asansör hızı gibi) ve Kalman Filtremizin değerlerini tahmin etmesine izin verebiliriz. Seçtiğimiz durumların, geribildirim denetleyicisi tarafından ilgili *references* e doğru yönlendirileceğini unutmayın (tipik olarak : ref: Doğrusal Karesel Düzenleyici <docs/software/advanced-controls/state-space/state-space-intro:The Linear-Quadratic Regulator>, çünkü optimaldir).

Volanımız için yalnızca tek bir durumu önemsiyoruz : hızı. İvmesini de modellemeyi seçebilirken, bu durumun dahil edilmesi sistemimiz için gerekli değildir.

Daha sonra, : terimini tanımlıyoruz: `sistemimize <input> girişleri. Girdiler, durumunu değiştirmek için sistemimize "into-içine" koyabileceğimiz şeyler olarak düşünülebilir. Volan (ve FRC ® 'deki diğer birçok tek eklemlili mekanizma) durumunda, sadece bir girişimiz var: motora uygulanan voltaj. Girişimiz olarak voltajı seçerek (motor görev döngüsü gibi bir şey üzerinde), akü yükü arttıkça akü voltajındaki düşüşü telafi edebiliriz.

Sürekli zaman durum-uzay sistemi şunu yazar \dot{x} veya sistemin *system'* durumunun anlık değişim oranı, şu anki *state* ile orantılı olarak ve $\text{inputs} <\text{input}>$. Durumumuz açısız hız olduğundan, $\dot{\mathbf{x}}$ volanın açısız ivmesi olacaktır.

Daha sonra, çarkımızı sürekli zaman durum-uzay sistemi olarak modelleyeceğiz. WPILib'in LinearSystem' sistemi bunu dahili olarak ayırık zamana dönüştürecektir. Gözden geçirme : ref: "durum uzaýı gösterimi <docs/software/advanced-controls/state-space/state-space-intro:What is State-Space Notation?>", sürekli zamanlı ve ayırık zamanlı sistemler hakkında daha fazla bilgi için.

Sistem Tanımlama ile Modelleme

To rewrite this in state-space notation using *system identification*, we recall from the flywheel *state-space notation example*, where we rewrote the following equation in terms of \mathbf{a} .

$$V = kV \cdot \mathbf{v} + kA \cdot \mathbf{a}$$

$$\mathbf{a} = \mathbf{v} = \left[\frac{-kV}{kA} \right] v + \left[\frac{1}{kA} \right] V$$

Burada: \mathbf{v} , volan hızıdır; \mathbf{a} ve $\dot{\mathbf{v}}$, volan ivmesidir ve V voltajdır. Bunu, durum vektörü için standart \mathbf{x} ve giriş vektörü için \mathbf{u} standart kuralıyla yeniden yazarak şunu buluruz:

$$\dot{\mathbf{x}} = \left[\frac{-kV}{kA} \right] \mathbf{x} + \left[\frac{1}{kA} \right] \mathbf{u}$$

Durum uzayı notasyonunun ikinci bölümü, sistemin o anki *state* ve *girdileri* ile *output* ile ilişkilidir. Bir volan durumunda, çıktı vektörümüz \mathbf{y} (veya sensörlerimizin ölçebildiği şeyler) volanımızın hızıdır ve bu da bizim *state* \mathbf{x} vektörümüzün bir unsuru olur. Bu nedenle, çıktı matrisimiz $\mathbf{C} = [1]$ ve sistem besleme matrisimiz $\mathbf{D} = [0]$ olur. Bunu sürekli zaman durum uzayı gösteriminde yazmak aşağıdakileri verir.

$$\dot{\mathbf{x}} = \left[\frac{-kV}{kA} \right] \mathbf{x} + \left[\frac{1}{kA} \right] \mathbf{u}$$

$$\mathbf{y} = [1] \mathbf{x} + [0] \mathbf{u}$$

LinearSystem sınıfı, şu şekilde tanımlanmış durum uzayı sistemlerini kolayca oluşturmak için yöntemler içerir *system identification*. Bu örnek, kV değeri 0,023 ve kA değeri 0,001 olan bir volan modelini göstermektedir:

Java

```

33 // Volts per (radian per second)
34 private static final double kFlywheelKv = 0.023;
35
36 // Volts per (radian per second squared)
37 private static final double kFlywheelKa = 0.001;
38
39 // The plant holds a state-space model of our flywheel. This system has the
40 // following properties:
41 //
42 // States: [velocity], in radians per second.
43 // Inputs (what we can "put in"): [voltage], in volts.
44 // Outputs (what we can measure): [velocity], in radians per second.
45 //
46 // The Kv and Ka constants are found using the FRC Characterization toolsuite.
47 private final LinearSystem<N1, N1, N1> m_flywheelPlant =
    LinearSystemId.identifyVelocitySystem(kFlywheelKv, kFlywheelKa);

```

C++

```

17 #include <frc/system/plant/LinearSystemId.h>

30 // Volts per (radian per second)
31 static constexpr auto kFlywheelKv = 0.023_V / 1_rad_per_s;
32
33 // Volts per (radian per second squared)
34 static constexpr auto kFlywheelKa = 0.001_V / 1_rad_per_s_sq;
35
36 // The plant holds a state-space model of our flywheel. This system has the
37 // following properties:
38 //
39 // States: [velocity], in radians per second.
40 // Inputs (what we can "put in"): [voltage], in volts.
41 // Outputs (what we can measure): [velocity], in radians per second.
42 //
43 // The Kv and Ka constants are found using the FRC Characterization toolsuite.
44 frc::LinearSystem<1, 1, 1> m_flywheelPlant =
45     frc::LinearSystemId::IdentifyVelocitySystem<units::radian>(kFlywheelKv,
46                                                                kFlywheelKa);

```

Python

```

23 # Volts per (radian per second)
24 kFlywheelKv = 0.023
25
26 # Volts per (radian per second squared)
27 kFlywheelKa = 0.001

37 # The plant holds a state-space model of our flywheel. This system has the
38 ↪ following properties:
39 #
40 # States: [velocity], in radians per second.
41 # Inputs (what we can "put in"): [voltage], in volts.
42 # Outputs (what we can measure): [velocity], in radians per second.
43 #
44 # The Kv and Ka constants are found using the FRC Characterization toolsuite.
45 self.flywheelPlant = (
46     wpimath.system.plant.LinearSystemId.identifyVelocitySystemRadians(
47         kFlywheelKv, kFlywheelKa
48     )
49 )

```

Volan Atalet Momenti ve Dişli Kullanarak Modelleme

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).

Not: WPILib'in durum uzayı sınıfları için dişli, giriş üzerinden çıkış olarak yazılır - yani, volan motorlardan daha yavaş dönüyorsa, bu sayı birden büyük olmalıdır.

Not: C++ LinearSystem sınıfı, birim karışıklıklarını önlemek ve boyutluluğu ileri sürmek için *the C++ Units Library* kullanır.

Java

```

33 private static final double kFlywheelMomentOfInertia = 0.00032; // kg * m^2
34
35 // Reduction between motors and encoder, as output over input. If the flywheel
36 // spins slower than the motors, this number should be greater than one.
37 private static final double kFlywheelGearing = 1.0;
38
39 // The plant holds a state-space model of our flywheel. This system has the
40 // following properties:
41 //
42 // States: [velocity], in radians per second.
43 // Inputs (what we can "put in"): [voltage], in volts.
44 // Outputs (what we can measure): [velocity], in radians per second.
45 private final LinearSystem<N1, N1, N1> m_flywheelPlant =
46     LinearSystemId.createFlywheelSystem(
47         DCMotor.getNEO(2), kFlywheelMomentOfInertia, kFlywheelGearing);

```

C++

```

17 #include <frc/system/plant/LinearSystemId.h>
31
32 #include <frc/system/plant/LinearSystemId.h>
33 static constexpr units::kilogram_square_meter_t kFlywheelMomentOfInertia =
34     0.00032_kg_sq_m;
35
36 // Reduction between motors and encoder, as output over input. If the flywheel
37 // spins slower than the motors, this number should be greater than one.
38 static constexpr double kFlywheelGearing = 1.0;
39
40 // The plant holds a state-space model of our flywheel. This system has the

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

40 // following properties:
41 //
42 // States: [velocity], in radians per second.
43 // Inputs (what we can "put in"): [voltage], in volts.
44 // Outputs (what we can measure): [velocity], in radians per second.
45 frc::LinearSystem<1, 1, 1> m_flywheelPlant =
46     frc::LinearSystemId::FlywheelSystem(
47         frc::DCMotor::NEO(2), kFlywheelMomentOfInertia, kFlywheelGearing);

```

Python

```

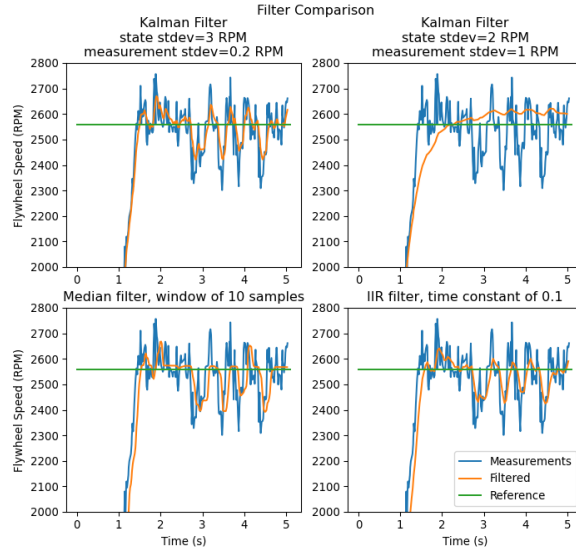
21 kFlywheelMomentOfInertia = 0.00032 # kg/m^2
22
23 # Reduction between motors and encoder, as output over input. If the flywheel spins
24 # slower than
25 kFlywheelGearing = 1
26
27 # The plant holds a state-space model of our flywheel. This system has the
28 # following properties:
29 #
30 # States: [velocity], in radians per second.
31 # Inputs (what we can "put in"): [voltage], in volts.
32 # Outputs (what we can measure): [velocity], in radians per second.
33 self.flywheelPlant = wpimath.system.plant.LinearSystemId.flywheelSystem(
34     wpimath.system.plant.DCMotor.NEO(2),
35     kFlywheelMomentOfInertia,
36     kFlywheelGearing,
37 )

```

Kalman Filtreleri: Volan Durumunu Gözlemleme

Kalman filtreleri, bir durum tahmini oluşturmak için durum uzayı modelimizi kullanarak hız ölçümlerimizi filtrelemek için kullanılır : \hat{x} . Volan modelimiz doğrusal olduğundan, volanın hızını tahmin etmek için bir Kalman filtresi kullanabiliriz. WPILib’in Kalman filtresi, model ve sensör ölçümlerinin standart sapmalarıyla birlikte bir LinearSystem (yukarıda bulduğumuz) alır. Bu ağırlıkları ayarlayarak durum tahminimizin ne kadar “smooth-düzgün” olduğunu ayarlayabiliriz. Daha büyük durum standart sapmaları, filtrenin durumumuza “distrust-güvenmemesine” ve yeni ölçümleri daha fazla tercih etmesine neden olurken, daha büyük ölçüm standart sapmaları bunun tersini yapacaktır.

Bir volan durumunda, 3 rad / s’lik bir durum standart sapması ve 0.01 rad / s’lik bir ölçüm standardı sapması ile başlarız. Bu değerler kullanıcının seçmesine bağlıdır - bu ağırlıklar bir miktar gürültüye toleranslı olan ancak durum tahmini a volan için dış parazitlere hızlı bir şekilde tepki veren bir filtre üretti ve aşağıdakiler için iyi davranan bir filtre oluşturmak için ayarlanmalıdır. özel volanız. Zaman içindeki durumları, ölçümleri, girdileri, referansları ve çıktıları grafiklemek, Kalman filtrelerini ayarlamak için harika bir görsel yoldur.



Yukarıdaki grafik, iki farklı şekilde ayarlanmış Kalman filtresinin yanı sıra bir: *single-pole IIR filter* ve *Median Filtresi* gösterir. Bu veriler, bir atıcı ile ~ 5 saniyede toplandı ve dört top, atıcıdan geçirildi (hızdaki dört düşüşte görüldüğü gibi). İyi durum ve ölçüm standardı sapmalarını seçme konusunda katı kurallar olmamakla birlikte, bunlar genel olarak, harici rahatsızlıklara hızlı tepki verirken gürültüyü reddedecek kadar modele güvenecek şekilde ayarlanmalıdır.

Because the feedback controller computes error using the \hat{x} estimated by the Kalman filter, the controller will react to disturbances only as quickly the filter's state estimate changes. In the above chart, the upper left plot (with a state standard deviation of 3.0 and measurement standard deviation of 0.2) produced a filter that reacted quickly to disturbances while rejecting noise, while the upper right plot shows a filter that was barely affected by the velocity dips.

Java

```

48 // The observer fuses our encoder data and voltage inputs to reject noise.
49 private final KalmanFilter<N1, N1, N1> m_observer =
50     new KalmanFilter<>(
51         Nat.N1(),
52         Nat.N1(),
53         m_flywheelPlant,
54         VecBuilder.fill(3.0), // How accurate we think our model is
55         VecBuilder.fill(0.01), // How accurate we think our encoder
56         // data is
57         0.020);

```

C++

```

13 #include <frc/estimator/KalmanFilter.h>

48 // The observer fuses our encoder data and voltage inputs to reject noise.
49 frc::KalmanFilter<1, 1, 1> m_observer{
50     m_flywheelPlant,
51     {3.0}, // How accurate we think our model is
52     {0.01}, // How accurate we think our encoder data is
53     20_ms};

```

Python

```

48 # The observer fuses our encoder data and voltage inputs to reject noise.
49 self.observer = wpimath.estimator.KalmanFilter_1_1_1(
50     self.flywheelPlant,
51     [3], # How accurate we think our model is
52     [0.01], # How accurate we think our encoder data is
53     0.020,
54 )

```

Kalman filtreleri : ref: docs / software / advanced-controls / state-space / state-space-observers: Predict step’de durum uzayı modelimizi kullandığından, modelimizin mümkün olduğu kadar doğru olması önemlidir. Bunu doğrulamanın bir yolu, bir volanın giriş voltajını ve zaman içindeki hızını kaydetmek ve bu verileri Kalman filtresinde yalnızca predict-tahmin olarak adlandırarak yeniden yürütmektir. Daha sonra, kV ve kA kazançları (veya eylemsizlik momenti ve diğer sabitler), model kaydedilen verilerle yakından eşleşene kadar ayarlanabilir.

Doğrusal-Kuadratik Düzenleyiciler ve Plant Inversion Feedforward-İleri Besleme

Linear-Quadratic Regulator / Doğrusal-Kuadratik Düzenleyici, volanımızı *system* i *reference* a sürmek için bir geri bildirim denetleyicisi bulur . Volanımızın sadece bir durumu olduğu için, LQR’miz tarafından seçilen kontrol yasası şu biçimde olacaktır $\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$ burada \mathbf{K} , bir 1x1 matrisdir; başka bir deyişle, LQR tarafından seçilen kontrol kuralı basitçe orantılı bir kontrolör veya sadece bir P kazancı olan bir PID kontrolörüdür. Bu kazanç, LQR’miz tarafından, geçtiğimiz durum gezi ve kontrol çabalarına göre seçilir. LQR denetleyicilerinin ayarlanması hakkında daha fazla bilgi şu adreste bulunabilir [LQR uygulama örneği](#).

SimpleMotorFeedforward gibi, kS, kV ve kA sabitleri verilen ileri besleme voltaj girişleri oluşturmak için kullanılabilir, Plant Inversion Feedforward sınıfı, bir durum uzay sistemi verilen *feedforward* voltaj girişlerini üretir. `` LinearSystemLoop`` sınıfı tarafından üretilen gerilim komutları ileri besleme ve geri besleme girişlerinin toplamıdır.

Java

```

59 // A LQR uses feedback to create voltage commands.
60 private final LinearQuadraticRegulator<N1, N1, N1> m_controller =
61     new LinearQuadraticRegulator<>{
62         m_flywheelPlant,
63         VecBuilder.fill(8.0), // qelms. Velocity error tolerance, in radians per
↪second. Decrease
64         // this to more heavily penalize state excursion, or make the controller
↪behave more
65         // aggressively.
66         VecBuilder.fill(12.0), // relms. Control effort (voltage) tolerance.
↪Decrease this to more
67         // heavily penalize control effort, or make the controller less aggressive.
↪12 is a good
68         // starting point because that is the (approximate) maximum voltage of a
↪battery.
69         0.020); // Nominal time between loops. 0.020 for TimedRobot, but can be
70         // lower if using notifiers.

```

C++

```

11 #include <frc/controller/LinearQuadraticRegulator.h>

```

```

54 // A LQR uses feedback to create voltage commands.
55 frc::LinearQuadraticRegulator<1, 1> m_controller{
56     m_flywheelPlant,
57     // qelms. Velocity error tolerance, in radians per second. Decrease this
58     // to more heavily penalize state excursion, or make the controller behave
59     // more aggressively.
60     {8.0},
61     // relms. Control effort (voltage) tolerance. Decrease this to more
62     // heavily penalize control effort, or make the controller less
63     // aggressive. 12 is a good starting point because that is the
64     // (approximate) maximum voltage of a battery.
65     {12.0},
66     // Nominal time between loops. 20ms for TimedRobot, but can be lower if
67     // using notifiers.
68     20_ms};
69
70 // The state-space loop combines a controller, observer, feedforward and plant
71 // for easy control.
72 frc::LinearSystemLoop<1, 1, 1> m_loop{m_flywheelPlant, m_controller,
73     m_observer, 12_V, 20_ms};

```

Python

```

56     # A LQR uses feedback to create voltage commands.
57     self.controller = wpimath.controller.LinearQuadraticRegulator_1_1(
58         self.flywheelPlant,
59         [8], # qelms. Velocity error tolerance, in radians per second. Decrease
60         # this to more heavily penalize state excursion, or make the controller
↪ behave more
61         # aggressively.
62         [12], # relms. Control effort (voltage) tolerance. Decrease this to more
63         # heavily penalize control effort, or make the controller less aggressive.
↪ 12 is a good
64         # starting point because that is the (approximate) maximum voltage of a
↪ battery.
65         0.020, # Nominal time between loops. 0.020 for TimedRobot, but can be
↪ lower if using notifiers.
66     )

```

Hepsini Bir Araya Getirin : LinearSystemLoop

LinearSystemLoop, daha önce oluşturduğumuz sistemimizi, denetleyicimizi ve gözlemcimizi birleştirir. Gösterilen kurucu ayrıca bir PlantInversionFeedforward başlatacaktır.

Java

```

72     // The state-space loop combines a controller, observer, feedforward and plant for
↪ easy control.
73     private final LinearSystemLoop<N1, N1, N1> m_loop =
74         new LinearSystemLoop<>(m_flywheelPlant, m_controller, m_observer, 12.0, 0.020);

```

C++

```

15 #include <frc/system/LinearSystemLoop.h>

71 // The state-space loop combines a controller, observer, feedforward and plant
72 // for easy control.
73 frc::LinearSystemLoop<1, 1, 1> m_loop{m_flywheelPlant, m_controller,
74                                     m_observer, 12_V, 20_ms};

```

Python

```

68     # The state-space loop combines a controller, observer, feedforward and plant
↪ for easy control.
69     self.loop = wpimath.system.LinearSystemLoop_1_1_1(
70         self.flywheelPlant, self.controller, self.observer, 12.0, 0.020
71     )

```


LinearSystemLoop umuza sahip olduğumuzda, geriye kalan tek şey onu çalıştırmaktır. Bunu yapmak için, Kalman filtremizi yeni enkoder hız ölçümlerimizle periyodik olarak güncelleyeceğiz ve ona yeni voltaj komutları uygulayacağız. Bunu yapmak için, önce *reference* yi, ardından mevcut volan hızıyla *correct*`, bir sonraki zaman adımına Kalman filtresini *predict* ve *getU* kullanılarak oluşturulan girdileri uyguluyoruz. .

Java

```
95  @Override
96  public void teleopPeriodic() {
97      // Sets the target speed of our flywheel. This is similar to setting the setpoint
98      // of a
99      // PID controller.
100     if (m_joystick.getTriggerPressed()) {
101         // We just pressed the trigger, so let's set our next reference
102         m_loop.setNextR(VecBuilder.fill(kSpinupRadPerSec));
103     } else if (m_joystick.getTriggerReleased()) {
104         // We just released the trigger, so let's spin down
105         m_loop.setNextR(VecBuilder.fill(0.0));
106     }
107
108     // Correct our Kalman filter's state vector estimate with encoder data.
109     m_loop.correct(VecBuilder.fill(m_encoder.getRate()));
110
111     // Update our LQR to generate new voltage commands and use the voltages to
112     // predict the next
113     // state with out Kalman filter.
114     m_loop.predict(0.020);
115
116     // Send the new calculated voltage to the motors.
117     // voltage = duty cycle * battery voltage, so
118     // duty cycle = voltage / battery voltage
119     double nextVoltage = m_loop.getU(0);
120     m_motor.setVoltage(nextVoltage);
121 }
```

C++

```
5  #include <numbers>
6
7  #include <frc/DriverStation.h>
8  #include <frc/Encoder.h>
9  #include <frc/TimedRobot.h>
10 #include <frc/XboxController.h>
11 #include <frc/controller/LinearQuadraticRegulator.h>
12 #include <frc/drive/DifferentialDrive.h>
13 #include <frc/estimator/KalmanFilter.h>
14 #include <frc/motorcontrol/PWMSparkMax.h>
15 #include <frc/system/LinearSystemLoop.h>
16 #include <frc/system/plant/DCMotor.h>
17 #include <frc/system/plant/LinearSystemId.h>
```

```

92 void TeleopPeriodic() override {
93     // Sets the target speed of our flywheel. This is similar to setting the
94     // setpoint of a PID controller.
95     if (m_joystick.GetRightBumper()) {
96         // We pressed the bumper, so let's set our next reference
97         m_loop.SetNextR(frc::Vectord<1>{kSpinup.value()});
98     } else {
99         // We released the bumper, so let's spin down
100        m_loop.SetNextR(frc::Vectord<1>{0.0});
101    }
102
103    // Correct our Kalman filter's state vector estimate with encoder data.
104    m_loop.Correct(frc::Vectord<1>{m_encoder.GetRate()});
105
106    // Update our LQR to generate new voltage commands and use the voltages to
107    // predict the next state with out Kalman filter.
108    m_loop.Predict(20_ms);
109
110    // Send the new calculated voltage to the motors.
111    // voltage = duty cycle * battery voltage, so
112    // duty cycle = voltage / battery voltage
113    m_motor.SetVoltage(units::volt_t{m_loop.U(0)});
114 }

```

Python

```

87 def teleopPeriodic(self) -> None:
88     # Sets the target speed of our flywheel. This is similar to setting the
89     ↪setpoint of a
90     # PID controller.
91     if self.joystick.getTriggerPressed():
92         # We just pressed the trigger, so let's set our next reference
93         self.loop.setNextR([kSpinUpRadPerSec])
94
95     elif self.joystick.getTriggerReleased():
96         # We just released the trigger, so let's spin down
97         self.loop.setNextR([0.0])
98
99     # Correct our Kalman filter's state vector estimate with encoder data.
100    self.loop.correct([self.encoder.getRate()])
101
102    ↪Update our LQR to generate new voltage commands and use the voltages to
103    ↪predict the next
104    # state with out Kalman filter.
105    self.loop.predict(0.020)
106
107    # Send the new calculated voltage to the motors.
108    # voltage = duty cycle * battery voltage, so
109    # duty cycle = voltage / battery voltage
110    nextVoltage = self.loop.U()
111    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;
```

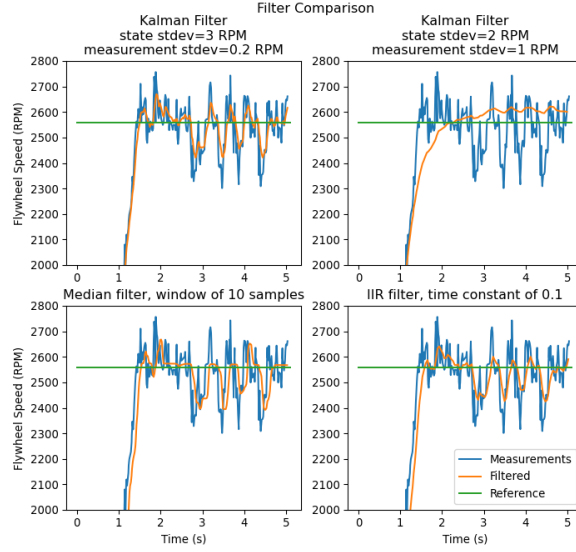
PYTHON

```
error = lqr.R() - x
error[0] = wpimath.angleModulus(error[0])
u = lqr.K() * error
```

32.8.3 Durum denetleyicisi ve Kalman Filtreleri

Durum gözlemcileri, sistemin doğru *state* nu tahmin etmek için bir sistemin davranışı ve harici ölçümler hakkındaki bilgileri birleştirir. Doğrusal sistemler için kullanılan yaygın bir gözlemci Kalman Filtresidir. Kalman filtreleri diğerlerine göre avantajlıdır *filters*, bir sistemin durumunu en iyi şekilde tahmin etmek için sistemin durum-uzay modeliyle bir veya daha fazla sensörden alınan ölçümleri birleştirir.

Bu görüntü, çeşitli farklı filtrelerden geçen zaman içindeki döner hıza ölçümlerini gösterir. İyi ayarlanmış bir Kalman filtresinin döner hıza dönüşü sırasında hiçbir ölçüm gecikmesi göstermediğini ve gürültülü verileri hala reddederken ve toplar içinden geçerken rahatsızlıklara hızlı tepki verdiğini unutmayın. Filtrelerle ilgili daha fazla bilgi *filters section* bulunabilir.



Gauss Fonksiyonları

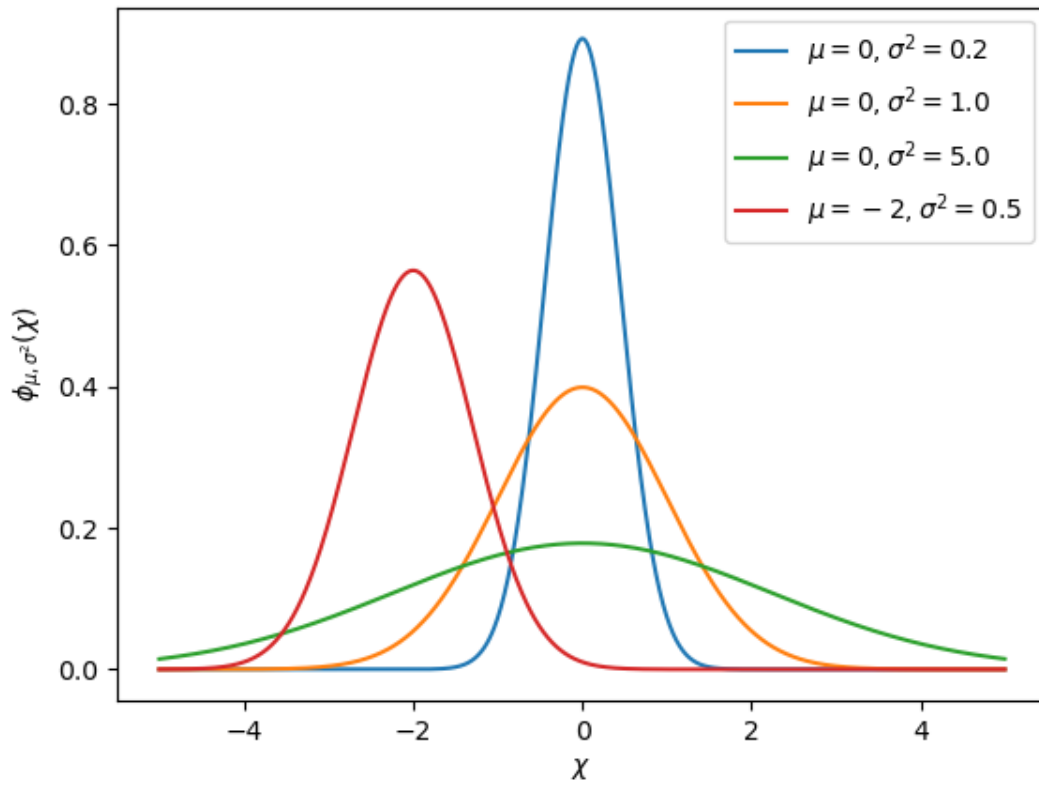
Kalman filters utilize a *Gaussian distribution* to model the noise in a process¹. In the case of a Kalman filter, the estimated *state* of the system is the mean, while the variance is a measure of how certain (or uncertain) the filter is about the true *state*.

Varyans ve kovaryans fikri, bir Kalman filtresinin işlevinin merkezinde yer alır. Kovaryans, iki rastgele değişkenin nasıl ilişkilendirildiğinin bir ölçüsüdür. Tek durumlu bir sistemde kovaryans matrisi basitçe $\text{var}(\mathbf{x}_1)$ veya varyansı içeren bir matristir $\text{var}(\mathbf{x}_1)$ durum x_1 . Bu varyansın büyüklüğü, mevcut durum tahminini açıklayan Gauss fonksiyonunun standart sapmasının karesidir. Kovaryans için nispeten büyük değerler gürültülü verileri gösterebilirken, küçük kovaryanslar filtrenin tahmini konusunda daha emin olduğunu gösterebilir. Varyans veya kovaryans için “büyük” ve “küçük” değerlerin kullanılan temel birime göre olduğunu unutmayın - örneğin, eğer \mathbf{x}_1 metre cinsinden ölçüldüyse, $\text{cov}(\mathbf{x}_1, \mathbf{x}_1)$ metre kare cinsinden olacaktır.

Kovaryans matrisleri aşağıdaki biçimde yazılır:

$$\Sigma = \begin{bmatrix} \text{COV}(x_1, x_1) & \text{COV}(x_1, x_2) & \dots & \text{COV}(x_1, x_n) \\ \text{COV}(x_2, x_1) & \text{COV}(x_2, x_2) & \dots & \text{COV}(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{COV}(x_n, x_1) & \text{COV}(x_n, x_2) & \dots & \text{COV}(x_n, x_n) \end{bmatrix}$$

¹ In a real robot, noise comes from all sorts of sources. Stray electromagnetic radiation adds extra voltages to sensor readings, vibrations and temperature variations throw off inertial measurement units, gear lash causes encoders to have inaccuracies when directions change... all sorts of things. It's important to realize that, by themselves, each of these sources of “noise” aren't guaranteed to follow any pattern. Some of them might be the “white noise” random vibrations you've probably heard on the radio. Others might be “pops” or single-loop errors. Others might be nominally zero, but strongly correlated with events on the robot. However, the *Central Limit Theorem* shows mathematically that regardless of how the individual sources of noise are distributed, as we add more and more of them up their combined effect eventually is distributed like a Gaussian. Since we do not know the exact individual sources of noise, the best choice of a model we can make is indeed that Gaussian function.



Kalman Filtreleri

Önemli: Bir Kalman filtresinin gerçekte ne yaptığına dair bir sezgi geliştirmek önemlidir. [Kalman and Bayesian Filters in Python by Roger Labbe](#) tarafından kitap Bayes filtrelerine harika bir görsel ve etkileşimli giriş sağlar. WPILib'deki Kalman filtreleri matematiği kolay hale getirmek için doğrusal cebir kullanır, ancak fikirler tek boyutlu duruma benzer. Bu filtrelerin ne yaptığına dair bir fikir edinmek için Bölüm 4'ü okumanızı öneririz.

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).

Aşağıda, ayrık zamanlı bir Kalman filtresinin denklemleri gösterilmektedir:

Predict step

$$\begin{aligned}\hat{\mathbf{x}}_{k+1}^- &= \mathbf{A}\hat{\mathbf{x}}_k^+ + \mathbf{B}\mathbf{u}_k \\ \mathbf{P}_{k+1}^- &= \mathbf{A}\mathbf{P}_k^-\mathbf{A}^T + \mathbf{Q}\mathbf{Q}^T\end{aligned}$$

Update step

$$\begin{aligned}\mathbf{K}_{k+1} &= \mathbf{P}_{k+1}^-\mathbf{C}^T(\mathbf{C}\mathbf{P}_{k+1}^-\mathbf{C}^T + \mathbf{R})^{-1} \\ \hat{\mathbf{x}}_{k+1}^+ &= \hat{\mathbf{x}}_{k+1}^- + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{C}\hat{\mathbf{x}}_{k+1}^- - \mathbf{D}\mathbf{u}_{k+1}) \\ \mathbf{P}_{k+1}^+ &= (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C})\mathbf{P}_{k+1}^-\end{aligned}$$

A	system matrix	$\hat{\mathbf{x}}$	state estimate vector
B	input matrix	\mathbf{u}	input vector
C	output matrix	\mathbf{y}	output vector
D	feedthrough matrix	\mathbf{Q}	process noise intensity vector
P	error covariance matrix	\mathbf{Q}	process noise covariance matrix
K	Kalman gain matrix	R	measurement noise covariance matrix

Durum tahmini \mathbf{x} , ile birlikte **P**, filtremizin sistemin gerçek durumuna ilişkin tahminini tanımlayan Gauss işlevinin ortalamasını ve kovaryansını açıklar.

Gürültü Kovaryans Matrislerinin İşlenmesi ve Ölçümü

Süreç ve ölçüm gürültüsü kovaryans matrisleri **Q** ve \mathbf{R} durumlarımızın ve ölçümlerimizin her birinin varyansını tanımlar. Bir Gauss fonksiyonu için varyansın, fonksiyonun standart sapmasının karesi olduğunu unutmayın. Bir WPILib'de **Q** ve **R**, köşegenleri ilgili varyanslarını içeren köşegen matrisleridir. Örneğin, $\begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix}$ ve ölçümler $[\text{position}]$ durum standart sapmalarıyla $\begin{bmatrix} 0.1 & 1.0 \end{bmatrix}$ ve ölçüm standart sapması $[0.01]$ şunlara sahip olacaktır **Q** and **R** matrisler:

$$\mathbf{Q} = \begin{bmatrix} 0.01 & 0 \\ 0 & 1.0 \end{bmatrix}, \mathbf{R} = [0.0001]$$

Error Covariance Matrix-Hata Kovaryans Matrisi

Hata kovaryans matrisi \mathbf{P} , durum tahmininin kovaryansını açıklar $\hat{\mathbf{x}}$. Gayri resmi olarak, \mathbf{P} , tahmin edilen *state* hakkındaki kesinliğimizi tanımlar. Eğer \mathbf{P} büyükse, gerçek durum hakkındaki belirsizliğimiz büyüktür. Tersine, daha küçük elemanlara sahip \mathbf{P} , gerçek durumumuz hakkında daha az belirsizlik anlamına gelir.

Modeli ileriye doğru yansıtırken, sistemin gerçek durumu hakkındaki kesinliğimiz azaldıkça \mathbf{P} artar.

Tahmin adımı

Tahminde, durum tahminimiz doğrusal sistem dinamiklerine göre güncellenir $\mathbf{x} = \mathbf{Ax} + \mathbf{Bu}$. Ayrıca, hata kovaryansımız \mathbf{P} , işlem gürültü kovaryans matrisi \mathbf{Q} ile artar. Daha büyük \mathbf{Q} değerleri, hata kovaryansımızın \mathbf{P} daha hızlı büyümesini sağlayacaktır. Bu \mathbf{P} , modeli ve ölçümleri ağırlıklandırmak için düzeltme adımında kullanılır.

Doğru adım

Doğru adımda, durum tahminimiz yeni ölçüm bilgilerini içerecek şekilde güncellenir. Bu yeni bilgi, Kalman kazancı \mathbf{K} tarafından $\hat{\mathbf{x}}$ 'durum tahminine göre ağırlıklandırılır. \mathbf{K} 'nın büyük değerleri gelen ölçümleri daha fazla ağırlıklandırırken, \mathbf{K} 'nın daha küçük değerleri durum tahminimize daha fazla ağırlık verir. Çünkü \mathbf{K} , \mathbf{P} ile ilgilidir, \mathbf{P} 'nin daha büyük değerleri \mathbf{K} 'nın arttırılacaktrvelmleridahaarlıklandractr.rnein, uzunbirsreiiñbirfiltretahminedilirse, byk : \mathbf{P} 'nin yeni bilgileri büyük ölçüde ağırlaştıracaktır.

Son olarak, hata kovaryansı \mathbf{P} durum tahminine olan güvenimizi artırmak için azalır.

Kalman Filtrelerini Ayarlama

WPILib'in Kalman Filtresi sınıflarının constructorları, doğrusal bir sistemi, işlem gürültüsü standart sapmalarının bir vektörünü ve ölçüm gürültüsü standart sapmalarını alır. Bunlar, köşegenleri her durum veya ölçümün standart sapmalarının veya varyanslarının karesiyle doldurarak \mathbf{Q} ve \mathbf{R} matrislerine dönüştürülür. Bir durumun standart sapmasını (ve dolayısıyla, \mathbf{Q} içindeki karşılık gelen girdisini) azaltarak, filtre gelen ölçümlere daha fazla güvenmeyecektir. Benzer şekilde, bir durumun standart sapmasını artırmak, gelen ölçümlere daha çok güvenecektir. Aynısı, ölçüm standart sapmaları için de geçerlidir - bir girişi azaltmak, filtrenin ilgili durum için gelen ölçüme daha fazla güvenmesini sağlarken, artması ölçüme olan güveni azaltacaktır.

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

```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

56     // data is
57     0.020);

```

C++

```

5  #include <numbers>
6
7  #include <frc/DriverStation.h>
8  #include <frc/Encoder.h>
9  #include <frc/TimedRobot.h>
10 #include <frc/XboxController.h>
11 #include <frc/controller/LinearQuadraticRegulator.h>
12 #include <frc/drive/DifferentialDrive.h>
13 #include <frc/estimator/KalmanFilter.h>
14 #include <frc/motorcontrol/PWMSparkMax.h>
15 #include <frc/system/LinearSystemLoop.h>
16 #include <frc/system/plant/DCMotor.h>
17 #include <frc/system/plant/LinearSystemId.h>
18 #include <units/angular_velocity.h>

```

```

48 // The observer fuses our encoder data and voltage inputs to reject noise.
49 frc::KalmanFilter<1, 1, 1> m_observer{
50     m_flywheelPlant,
51     {3.0}, // How accurate we think our model is
52     {0.01}, // How accurate we think our encoder data is
53     20_ms};

```

Python

```

48 # The observer fuses our encoder data and voltage inputs to reject noise.
49 self.observer = wpimath.estimator.KalmanFilter_1_1_1(
50     self.flywheelPlant,
51     [3], # How accurate we think our model is
52     [0.01], # How accurate we think our encoder data is
53     0.020,
54 )

```

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 $[x \ y \ \theta]^T$, 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

```
86 private final DifferentialDrivePoseEstimator m_poseEstimator =
87     new DifferentialDrivePoseEstimator(
88         m_kinematics,
89         m_gyro.getRotation2d(),
90         m_leftEncoder.getDistance(),
91         m_rightEncoder.getDistance(),
92         new Pose2d(),
93         VecBuilder.fill(0.05, 0.05, Units.degreesToRadians(5)),
94         VecBuilder.fill(0.5, 0.5, Units.degreesToRadians(30)));
```

C++

```
158 frc::DifferentialDrivePoseEstimator m_poseEstimator{
159     m_kinematics,
160     m_gyro.GetRotation2d(),
161     units::meter_t{m_leftEncoder.GetDistance()},
162     units::meter_t{m_rightEncoder.GetDistance()},
163     frc::Pose2d{},
164     {0.01, 0.01, 0.01},
165     {0.1, 0.1, 0.1}};
```

Add odometry measurements every loop by calling `Update()`.

JAVA

```
227 m_poseEstimator.update(
228     m_gyro.getRotation2d(), m_leftEncoder.getDistance(), m_rightEncoder.
    ↪ getDistance());
```

C++

```

84     m_poseEstimator.Update(m_gyro.GetRotation2d(),
85                           units::meter_t{m_leftEncoder.GetDistance()},
86                           units::meter_t{m_rightEncoder.GetDistance()});

```

Add vision pose measurements occasionally by calling `AddVisionMeasurement()`.

JAVA

```

236     // Compute the robot's field-relative position exclusively from vision
    ↪ measurements.
237     Pose3d visionMeasurement3d =
238         objectToRobotPose(m_objectInField, m_robotToCamera, m_cameraToObjectEntry);
239
240     // Convert robot pose from Pose3d to Pose2d needed to apply vision measurements.
241     Pose2d visionMeasurement2d = visionMeasurement3d.toPose2d();
242
243     // Apply vision measurements. For simulation purposes only, we don't input a
    ↪ latency delay -- on
244     // a real robot, this must be calculated based either on known latency or
    ↪ timestamps.
245     m_poseEstimator.addVisionMeasurement(visionMeasurement2d, Timer.
    ↪ getFPGATimestamp());

```

C++

```

93     // Compute the robot's field-relative position exclusively from vision
94     // measurements.
95     frc::Pose3d visionMeasurement3d = ObjectToRobotPose(
96         m_objectInField, m_robotToCamera, m_cameraToObjectEntryRef);
97
98     // Convert robot's pose from Pose3d to Pose2d needed to apply vision
99     // measurements.
100     frc::Pose2d visionMeasurement2d = visionMeasurement3d.ToPose2d();
101
102     // Apply vision measurements. For simulation purposes only, we don't input a
103     // latency delay -- on a real robot, this must be calculated based either on
104     // known latency or timestamps.
105     m_poseEstimator.AddVisionMeasurement(visionMeasurement2d,
106         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 Durum Uzayı Modellerinde ve Denetleyicilerinde Hata Ayıklama

İşaretleri Kontrol Etmek

Durum alanı denetleyicilerindeki hataların en yaygın nedenlerinden biri, işaretlerin ters çevrilmesidir. Örneğin, WPILib’e dahil olan modeller, pozitif voltajın pozitif bir ivmeyle sonuçlanmasını bekler ve bunun tersi de geçerlidir. Pozitif voltaj uygulanması mekanizmanın ileriye doğru hızlanmasına neden olmazsa veya “ileri” hareket etmek kodlayıcıyı (veya diğer sensör okumalarını) düşürürse, pozitif voltaj girişinin pozitif kodlayıcı okumasıyla sonuçlanması için ters çevrilmelidirler. Örneğin, kodlayıcılarımızın pozitif bir hız okuması için diferansiyel aktarma sistemime bir *input* of $[12, 12]^T$ (sol ve sağ motorlar için tam ileri) uygularsam, tekerleklerim robotumu “ileri” (yerel olarak + X eksenini boyunca) itmeli.

Önemli: 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.

Grafiklerin Önemi

Reliable data of the *system’s states*, *inputs* and *outputs* 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.

Not: Varsayılan olarak, NetworkTables 10 Hz güncelleme hızıyla sınırlıdır. Test için, 100 Hz’ye kadar veri göndermek için aşağıdaki kod parçacığı ile bu atlanabilir. Yeni verileri zorla yayınlamak için bu kod periyodik olarak çalıştırılmalıdır.

Tehlike: Bu, NetworkTables üzerinden fazladan veri (100 Hz’ye kadar) gönderecek ve bu, hem kullanıcı kodu hem de robot gösterge tablolarında gecikmeye neden olabilir. Bu aynı zamanda ağ kullanımını da artıracaktır. Yarışmalar sırasında bunu devre dışı bırakmak genellikle iyi bir fikirdir.

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()
```

Giriş Gecikmesini Telafi Etme

Çoğu zaman, bazı sensör giriş verileri (yani hız okumaları), akıllı motor kontrol cihazlarının gerçekleştirme eğiliminde olduğu dahili filtreleme nedeniyle gecikebilir. Varsayılan olarak, LQR'nin K kazancı herhangi bir giriş gecikmesi olmadığını varsayar, bu nedenle onlarca milisaniye düzeyinde önemli bir gecikme uygulamak istikrarsızlığa neden olabilir. Bununla mücadele etmek için, LQR'nin K kazancı, istikrar için performansın takas edilmesi suretiyle azaltılabilir. Bu gecikmenin matematiksel olarak titiz bir şekilde nasıl telafi edileceğine dair bir kod örneği mevcuttur: [ref:here <docs/software/advanced-controls/state-space/state-space-intro:LQR and Measurement Latency Compensation>](#).

32.9 Kontroller Sözlüğü

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.

Kontrol çabası - control effort

Control signal

Kontrol kanunu - control law

Varolan *state* dikkate alınarak bir *system* i istenilen *state* a getirmek için gerekli *inputs* ları üreten matematiksel formüldür. Yaygın bir kontrol yasası örneği $\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$

Kontrolör - Controller

reference sinyali ile *output* arasındaki farkı sıfıra sürerek istenen *system state* yi oluşturmak için *plant* i pozisyonlamak veya negatif geri beslemede kullanılır .

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.

dinamikler - dynamics

Kuvvetlerin etkisi altındaki cisimlerin hareketiyle ilgilenen bir fizik dalı. Modern kontrolden sistemler dinamiklerine göre gelişir.

derivative

A mathematical operation which evaluates the “rate-of-change” of a function at a given point. See [derivative](#) on Wikipedia for more info.

hata - error

Reference minus an *output* veya *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.

kazanç - gain

A scalar value that relates the magnitude of an input signal to the magnitude of an output signal. For example, gain in output = gain * input. A gain greater than one would

amplify an input signal, while a gain less than one would dampen an input signal. A negative gain would negate the input signal.

Gaussian distribution

A special mathematical function that describes distributions of averages. The graph of a Gaussian function is a “bell curve” shape. This function is described by its mean (the location of the “peak” of the bell curve) and variance (a measure of how “spread out” the bell curve is). See [Gaussian distribution](#) on Wikipedia for more info.

gradient

The *derivative*, but applied to a function with multiple inputs. As a result, the output is both the magnitude of the rate of change, and the vector direction along which it occurs.

gizli durum - hidden state

state terimi doğrudan ölçülemeyen, ancak *dynamics* terimi diğer durumlarla ilgili olabilen durum.

giriş - input

plant'in state - durumu değiştirmek için kullanılabilen *plant* (adı ile ilgili) için bir girdi.

- Örn. Bir tekerleği kontrol eden motorun: 1 voltaj girişi olacaktır.
- Örn. Bir sürüş tasarımının, sol ve sağ motorların voltajları olarak: 2 girişi olabilir .

Girişler genellikle **u** değişkeni ile temsil edilir, sütun vektörü *system* için *input* başına bir giriş içerir.

least-squares regression

A curve-fitting technique which picks a curve to minimize the *square* of the error between the fitted curve, and the actual measured data. See [ordinary least-squares regression](#) on Wikipedia for more info.

LQR

Linear-Quadratic Regulator - A feedback control scheme which seeks to operate a system in a “most optimal” or “lowest cost” manner, in the sense of minimizing the square of some “cost function” that represents a combination of system error and control effort. This requires an accurate mathematical model of the system being controlled, and function describing the “cost” of any given system state. See [LQR](#) on Wikipedia for more info.

ölçüm - measurement

Ölçümler şunlardır *outputs* , sensörler kullanılarak bir *plant* veya fiziksel sistemden ölçülür.

model

Fiziksel bir *system'in* davranışının bazı yönlerini yansıtan bir dizi matematiksel denklem.

onserver - gözlemci

Kontrol teorisinde, gerçek bir *system* in dahili *state* için verilen gerçek *input* ve *output* ölçümlerinden tahmini *state* sağlayan bir sistemdir. WPILib, doğrusal sistemleri gözlemlemek için bir Kalman Filtresi sınıfı ve doğrusal olmayan sistemler için ExtendedKalmanFilter ve UnscentedKalmanFilter sınıflarını içerir.

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.

çıkıtı

Sensörlerden ölçümler. Durumlardan - state daha fazla ölçüm olabilir. Bu çıktılar Kalman Filtrelerinin “correct-doğru” adımıyla kullanılır.

- Örn. Bir döner tekerin, hızını ölçen bir encoderin 1 *output* u olabilir.
- Örn. Bir sürüş organı, sahadaki x/y/baş konumunu bulmak için solvePNP ve V-SLAM kullanabilir. 6 ölçüm (solvePNP x/y/baş ve V-SLAM x/y/heading) ve 3 durum (robot x/y/heading) olması sorun değil.

system in çıktıları genellikle *y* değişkeni kullanılarak, her bir *output* (veya ölçebildiğimiz şey) başına bir giriş içeren bir sütun vektörü ile temsil edilir. Örneğin, *system* hız ve ivme için durumlara sahipse ancak sensörümüz yalnızca hızı ölçebiliyorsa, bizim *output* vektörümüz yalnızca *system* 'in hızını içerecektir.

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.

plant - tesis

system veya kontrol edilen aktüatörlerin toplamı.

süreç değişkeni - process variable

PID kontrolü bağlamında *plant* in çıktısını açıklamak için kullanılan terim.

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.

referans - reference

İstenilen durum. Bu değer, bir kontrol cihazının hata hesaplaması için referans noktası olarak kullanılır.

Yükseliş zamanı - rise time

: term: “system”, a: term: “step input” uygulandıktan sonra: term: “reference” ye ulaşmak için gereken süre.

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.

ayar noktası - set point

Bir PID kontrolörünün *reference* nı açıklamak için kullanılan terim.

yerleşme zamanı - settling time

step input uygulandıktan sonra *system* in *reference* ye yerleşmesi için geçen süre.

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.

durum - state

system 'in (örneğin hız) term:*sistem'in* <system> gelecekteki davranışını belirlemek için kullanılabilen bir özelliği. Durum-uzayı gösteriminde, bir sistemin durumu, durum uzayındaki konumunu tanımlayan bir sütun vektörü olarak yazılır.

- Örn. Bir sürüş organı sistemi, alandaki konumunu açıklamak için şu durumlara sahip

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

- Örn. Bir elevator sistemi şu durumlara sahip olabilir, mevcut yüksekliğini ve hızını tanımlamak için $\begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix}$.

systemin durumu genellikle: \mathbf{x} değişkeni ile temsil edilir, her bir: *state* için bir giriş içeren bir sütun vektörü.

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.

kararlı durum hatası - steady-state error

sistem dengeye ulaştıktan sonraki *Error* .

adım girişi - step input

system input yani 0 için $t < 0$ ve şundan büyük bir $t \geq 0$ için sabit 0 . $t \geq 0$ için 1 olan bir adım girdisine step input denir.

adım yanıtı - step response

system in *step input* a yanıtı.

sistem - system

Tek bir varlık olarak değerlendirilen *plant* ve bir:term:controller ve *observer* ile etkileşimini kapsayan bir terim. Matematiksel olarak konuşursak, *system* , *states* durumlarının doğrusal bir kombinasyonu aracılığıyla *inputs* 'u *outputs* a bağlar.

sistem kimliği - system identification

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.

sistem yanıtı - system response

Belirli bir *input* için *system* in zaman içindeki davranışı.

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

$\dot{\mathbf{x}}$ veya x-dot, *state* vektörünün türevi \mathbf{x} . Eğer *system* sadece bir hıza sahipse *state*, o zaman $\dot{\mathbf{x}}$ *system* ivmesini temsil eder.

x-hat

$\hat{\mathbf{x}}$ veya x-hat: bir *observer* tarafından tahmin edildiği gibi, bir sistemin tahmini *state* dir.

Kullanışlı Özellikler

Bu bölüm, diğer gelişmiş programlama özellikleriyle birlikte kullanılan bazı genel kolaylık özelliklerini kapsar.

33.1 İşlevleri Özel Frekanslarda Planlama

TimedRobot'un addPeriodic() metodu kişinin varsayılan TimedRobot periyodik güncelleme hızından (20 ms) daha hızlı bir hızda özel yöntemleri çalıştırmasına izin verir. Daha önce, ekipler geri bildirim denetleyicilerini 20 ms'lik TimedRobot döngü süresinden daha sık çalıştırmak için bir Notifier yapmak zorundaydı (TimedRobot u bundan daha sık çalıştırmak önerilmez). Artık kullanıcılar geri besleme denetleyicilerini ana robot döngüsünden daha sık çalıştırabilir, ancak TimedRobot periyodik işlevleriyle eş zamanlı olarak yapmak potansiyel iş parçacığı güvenliği sorunlarını ortadan kaldırabilir.

AddPeriodic() (Java)/ AddPeriodic() (C++) metodu, istenen süre ve ortak başlangıç zamanında isteğe bağlı bir farkla birlikte bir lambda (çalıştırılacak işlev) alır. İsteğe bağlı üçüncü argüman, bir işlevi diğer ``TimedRobot`` periyodik metodlarına bağlı olarak farklı bir zaman diliminde zamanlamak için kullanışlıdır.

Not: Peryod ve ofsetin birimleri Java'da saniyedir. C++ 'da, <docs/software/basic-programming/cpp-units:The C++ Units Library> birim kitaplığı, herhangi bir zaman biriminde bir dönem ve uzaklık belirtmek için kullanılabilir.

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() {
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```

        addPeriodic(() -> {
            m_motor.set(m_controller.calculate(m_encoder.getRate()));
        }, 0.01, 0.005);
    }

    @Override
    public teleopPeriodic() {
        if (m_joystick.getRawButtonPressed(1)) {
            if (m_controller.getSetpoint() == 0.0) {
                m_controller.setSetpoint(30.0);
            } else {
                m_controller.setSetpoint(0.0);
            }
        }
    }
}

```

C++ (Header)

```

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);
        }
    }
}

```

The `teleopPeriodic()` method in this example runs every 20 ms, and the controller update is run every 10 ms with an offset of 5 ms from when `teleopPeriodic()` runs so that their timeslots don't conflict (e.g., `teleopPeriodic()` runs at 0 ms, 20 ms, 40 ms, etc.; the controller runs at 5 ms, 15 ms, 25 ms, etc.).

33.2 Event-Based Programming With EventLoop

Many operations in robot code are driven by certain conditions; buttons are one common example. Conditions can be polled with an *imperative programming* style by using an `if` statement in a periodic method. As an alternative, WPILib offers an *event-driven programming* style of API in the shape of the `EventLoop` and `BooleanEvent` classes.

Not: 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(); }
```

Uyari: 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.

Uyarı: 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++

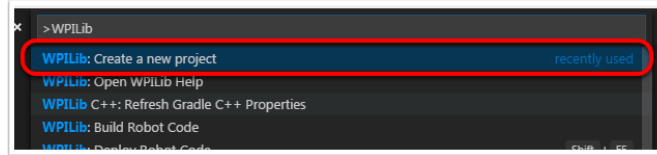
```
frc2::Trigger trigger = booleanEvent.CastTo<frc2::Trigger>();
```

Not: 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.

WPILib Örnek Projeler

Uyarı: WPILib örneklerini işlevsel tutmak için her girişimde bulunulsa da, “olduğu gibi” kullanılması *amaçlanmamıştır*. Kodun bir kullanıcı robotunda çalışması için en azından robota özgü sabitlerin değiştirilmesi gerekecektir. Birçok ampirik sabitin değerleri, gösterim amacıyla “sahte” hale getirilmiştir. Kullanıcıların, örnek kodu kopyalamak yerine kendi kodlarını (sıfırdan veya mevcut bir şablondan) yazmaları şiddetle tavsiye edilir.

WPILib örnek projeleri, çok sayıda kitaplık özelliği ve kullanım desenleri gösterir. Projeler, tek bir işlevin basit gösterimlerinden eksiksiz, rekabete uygun robot programlarına kadar çeşitlilik gösterir. Bu örneklerin tümü VS Code’da şu şekilde kullanılabilir Ctrl+Shift+P, ardından *WPILib: Create a new project* ve örnek seçilerek.



34.1 Temel Örnekler

Bu örnekler, temel / minimum robot işlevselliğini gösterir. Robot programlamaya ilk aşına olan, ancak işlevsellik açısından oldukça sınırlı olan yeni başlayan ekipler için kullanışlıdır.

- **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 Kontrol Örnekleri

Bu örnekler, yaygın robot kontrollerinin WPILib uygulamalarını gösterir. Sensörler mevcut olabilir, ancak bu örneklerin vurgulanan konsepti değildir.

- **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 Sensör Örnekleri

Bu örnekler, WPILib kullanarak sensör okumayı ve veri işlemeyi gösterir. Mekanizma kontrolü mevcut olabilir, ancak bu örneklerin vurgulanan konsepti değildir.

- **Axis Camera Sample** (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/org/roboio/axiscamera/AxisCameraExample.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/AxisCameraExample.cpp>>): Yakalanan bir video beslemesinde bir dikdörtgeni kaplamak ve bunu kontrol paneline aktarmak için OpenCV ve bir Axis Netcam kullanımını gösterir.
- **Power Distribution CAN Monitoring** (Java, C++, Python): Demonstrates obtaining sensor information from a Power Distribution module over CAN using the `PowerDistribution` class.
- **Duty Cycle Encoder** (Java, C++, Python): Demonstrates the use of the `DutyCycleEncoder` class to read values from a PWM-type absolute encoder.
- **DutyCycleInput** (Java, C++, Python): Demonstrates the use of the `DutyCycleInput` class to read the frequency and fractional duty cycle of a *PWM* input.
- **Encoder** (Java, C++, Python): Demonstrates the use of the `Encoder` class to read values from a quadrature encoder.
- **Gyro** (Java, C++, Python): Demonstrates the use of the `AnalogGyro` class to measure robot heading and stabilize driving.
- **Intermediate Vision** (Java, C++, Python): Demonstrates the use of OpenCV and a USB camera to overlay a rectangle on a captured video feed and stream it to the dashboard.
- **AprilTagsVision** (Java, C++): Demonstrates on-roboRIO detection of AprilTags using an attached USB camera.
- **Ultrasonic** (Java, C++, Python): Demonstrates the use of the `Ultrasonic` class to read data from an ultrasonic sensor in conjunction with the `MedianFilter` class to reduce signal noise.
- **SysIdRoutine** (Java, C++, Python): Demonstrates the use of the `SysIdRoutine` API to gather characterization data for a differential drivetrain.

34.4 Komut-Tabanlı Örnekler

Bu örnekler: *Command-Based framework*. 'ın kullanımını gösterir.

- **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 `PIDSubsystem` class.

- **Gears Bot** (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/examples/robot/Robot.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/gearsbot/main.cpp>>): WPI gösteri robotu GearsBot için eksiksiz bir robot kodu seti.
- **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 <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/examples/robot/Robot.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/mecanumcontrollercommand/main.cpp>>): TrajectoryGenerator ve MecanumControllerCommand sınıflarını kullanarak bir mecanum sürücü ile yörünge oluşturmayı ve takibi gösterir.
- **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 <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/examples/robot/Robot.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/swervecontrollercommand/main.cpp>>): TrajectoryGenerator ve SwerveControllerCommand sınıflarını kullanarak bir swerve sürücüsüyle yörünge oluşturmayı ve takip etmeyi gösterir.

34.5 State-Space Durum Uzayı Örnekleri

Bu örnekler, *State-Space Control*. 'in kullanımını gösterir.

- **StateSpaceFlywheel** (Java, C++, Python): Demonstrates state-space control of a flywheel.
- **StateSpaceFlywheelSysId** (Java, C++, Python): Demonstrates state-space control using SysId's System Identification for controlling a flywheel.
- **** StateSpaceElevator **** (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/examples/robot/Robot.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/statepaceelevator/main.cpp>>): Bir asansörün durum alanı kontrolünü gösterir.
- **** StateSpaceArm **** (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/examples/robot/Robot.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/statepacearm/main.cpp>>): Bir Kolum durum-uzay kontrolünü gösterir.
- **** StateSpaceDriveSimulation **** (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/java/edu/wpi/first/examples/robot/Robot.java>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/statepacedrivesimulation/main.cpp>>): Controller ve Field2d sınıfını takip eden bir RAMSETE yolu ile birlikte bir diferansiyel aktarma organının durum alanı kontrolünü gösterir.

34.6 Simülasyon Fiziği Örnekleri

Bu örnekler, fizik simülasyonunun kullanımını göstermektedir.

- **ElevatorSimulation** (Java, C++, Python): Demonstrates the use of physics simulation with a simple elevator.
- **ArmSimulation** (Java, C++, Python): Demonstrates the use of physics simulation with a simple single-jointed arm.
- **** StateSpaceDriveSimulation **** (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/StateSpaceDriveSimulation>>): Controller ve Field2d sınıfını takip eden bir RAMSETE yolu ile birlikte bir diferansiyel aktarma organının durum alanı kontrolünü gösterir.
- **** SimpleDifferentialDriveSimulation **** (Java <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibjExamples/src/main/cpp/examples/SimpleDifferentialDriveSimulation>>, C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/SimpleDifferentialDriveSimulation>>): Simülasyonda kullanılabilir temel bir aktarma organının barebone örneği.

34.7 Çeşitli Örnekler

Bu örnekler, yukarıdaki kategorilerin hiçbirine uymayan çeşitli WPILib işlevselliğini gösterir.

- **Addressable LED** (Java, C++, Python): Demonstrates the use of the AddressableLED class to control RGB LEDs for robot decoration and/or driver feedback.
- **DMA** (Java, C++): Demonstrates the use of DMA (Direct Memory Access) to read from sensors without using the RoboRIO's CPU.
- **** HAL **** (C++ <<https://github.com/wpilibsuite/allwpilib/tree/main/wpilibcExamples/src/main/cpp/examples/HAL>>): WPILib'in geri kalanını kullanmadan HAL (Donanım Soyutlama Katmanı) kullanımını gösterir. Bu örnek ileri düzey kullanıcılar içindir (yalnızca C++).
- **HID Rumble** (Java, C++, Python): Demonstrates the use of the "rumble" functionality for tactile feedback on supported HID's (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*.

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 [Destek Kaynakları](#) page.

- [Cross The Road Electronics \(CTRE\)](#)
- [Kauai Labs \(navX\)](#)
- [Limelight](#) (additional examples, called tutorials, can be found on the left)
- [PhotonVision](#)
- [REV Robotics](#)

36.1 En İyi Kablolama Uygulamaları

Tüyo: 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 Titreşim/Şok

Bir FRC® Robotu, titreşim ve şok yükleri söz konusu olduğunda inanılmaz derecede zorlu bir ortamdır. FRC’ye özgü elektroniklerin çoğu, bu koşullarda mekanik sağlamlık açısından kapsamlı bir şekilde test edilirken, radyo gibi birkaç bileşen, bir mobil platformda kullanılmak üzere özel olarak tasarlanmamıştır. Bu bileşenlerin maruz kaldığı şok ve titreşimi azaltmak için adımlar atmak arızaları azaltmaya yardımcı olabilir. Mekanik arızaları azaltabilecek bazı öneriler:

- Titreşim Yalıtımı - Kompresörler gibi aşırı titreşim yaratan tüm bileşenleri “titreşim yalıtıcıları” kullanarak izole ettiğinizden emin olun. Bu, robot üzerindeki titreşimi azaltmaya yardımcı olur, bu da bağlantı elemanlarını gevşetebilir ve bazı elektronik bileşenlerde erken yorulmaya neden olabilir.
- Bumpers - Tasarımınız için robotun mümkün olduğunca çoğunu örtmek için bumperları kullanın. Kurallar, robotunuzun köşelerinde özel bumper kapsamı gerektirse de, bumperların kullanımını en üst düzeye çıkarmak, tüm çarpışmaların bumperlarınız tarafından sönümlenme olasılığını artırır. Bumperlar, doğrudan sert bir robot yüzeyine çarpmaya kıyasla çarpışmada yaşanan kuvvetlerini önemli ölçüde azaltır, elektronik aksamın maruz kaldığı şoku azaltır ve şokla ilgili arıza olasılığını azaltır.
- Şok Montajı - Robot çarpışmalarında gördükleri kuvvetleri daha da azaltmak için elektronik bileşenlerinizin bir kısmını veya tamamını şokla monte etmeyi seçebilirsiniz. Bu, özellikle robot telsizi ve yardımcı işlemciler gibi, mobil platformlarda kullanılmak üzere tasarlanmamış olabilecek diğer elektronik cihazlar için yararlıdır. Titreşim izolatorleri, yaylar, köpükler veya esnek malzemelere montajın tümü, bu bileşenlerin gördüğü şok kuvvetlerini azaltabilir.

36.1.2 Yedeklilik

Ne yazık ki, FRC Kontrol Sisteminde yedeklilik için mümkün olduğu çok az yer vardır. Yedeklilik fırsatlarından yararlanmak, güvenilirliği artırabilir. Bunun birincil örneği, sağlanan PoE bağlantısına ek olarak güç adaptör konektörünün robot radiosuna bağlanmasıdır. Bu, kablolardan birinin hasar görmesi veya yerinden çıkması durumunda diğerinin radyoya giden gücü sürdürmesini sağlar. Robotunuzun kablolamasını yaparken ve programlarken yedeklilik sağlamak için diğer potansiyel alanlara dikkat edin.

36.1.3 Port Koruyucular

Robot veya Sürücü istasyonunda sıkça takılıp çıkarılabilen herhangi bir bağlantı için (DS oyun çubukları, DS Ethernet, roboRIO USB ve Ethernet bağlantısı gibi) bir “Port Koruyucu” veya “pigtail” kullanarak hasar verme potansiyelini önemli ölçüde azaltabilirsiniz. Bu tür bir cihaz, hem elektronik cihaz üzerindeki bağlantı noktasının gördüğü döngü sayısını azaltarak hem de bağlantıyı daha uygun bir konuma yeniden konumlandırarak çifte göreve hizmet edebilir. Bağlantı noktasının hasar görmesini önlemek için bağlantı noktası koruyucusunu sabitlediğinizden emin olun (sonraki maddeye bakın).

36.1.4 Kablo Yönetimi ve Gerilim Giderme

Robot güvenilirliği ve bakımı için en kritik bileşenlerden biri, iyi kablo yönetimi ve gerilim azaltmadır. İyi kablo yönetimi birkaç bileşenden oluşur:

- Kabloların doğru uzunlukta olduğundan emin olun. Herhangi bir fazla kablo uzunluğu sadece yönetilmesi gereken daha fazladır. COTS kablolamasında ek uzunluk nedeniyle fazladan kablounuz olması gerekiyorsa, telin geri kalanını sabitlemeden önce fazladan olanı ayrı kablo bağları kullanarak küçük bir demet halinde sabitleyin.
- Kabloların bağlantı noktalarına yakın bir yere sabitlendiğinden ve konektörleri zorlamamak için yeterince gevşek olduğundan emin olun. Buna gerilim azaltma denir ve bir kablounun fişten çekilmesi veya bir bağlantı noktasında bir telin kopması olasılığını en aza indirmek için kritiktir (bunlar genellikle gerilim yoğunlaştırıcılardır).
- Kabloları hareketli bileşenlerin yakınında sabitleyin. Hareketli bileşenler bükülecek veya aşırı hareket edecek olsa bile tüm kabloların güvenli ve hareketli bileşenlerden korunduğundan emin olun.
- Kabloları düzgün ve temiz tutmak için kabloları ek noktalarda gerektiği gibi sabitleyin. Kabloları çok fazla yerde sabitlememeye dikkat edin; kablolar çok fazla yerde sabitlenirse, aslında sorun giderme ve bakımı daha zor hale getirebilir.

36.1.5 Belgeleme

Bakımı kolaylaştırmanın harika bir yolu, robotun neresine bağlı olduğunu açıklayan belgeler oluşturmaktır. Bu tür bir dokümantasyon oluşturmamanın, eksiksiz bağlantı şemalarından excel tablolarına, hangi kanallara hangi fonksiyonların eklendiğini gösteren hızlı bir listeye kadar çeşitli yolları vardır. Birçok ekip de bu listeleri etiketlemeyle bütünleştirir (bir sonraki maddeye bakın).

Bir tel yanlışlıkla kesildiğinde veya bir mekanizma arızalandığında veya bir bileşen yandığında, kabloları baştan sona takip etmek zorunda kalmadan neyin nereye bağlı olduğunu size

söyleyecek bazı belgeleriniz varsa onarmak çok daha kolay olacaktır (kablo tesisatınız düzgün olsa bile!)

36.1.6 Etiketleme

Etiketleme, yukarıda açıklanan kablolama belgelerini tamamlamanın harika bir yoludur. Kablolama ve elektroniği etiketlemek için hepsi kendi artıları ve eksileri olan birçok farklı strateji vardır. Elektronikler için etiketler ve kablolar için bayraklar elle veya bir etiketleme makinesi kullanılarak yapılabilir (bazıları ısıyla daralan etiketlerle de yapılabilir) veya farklı şeyleri belirtmek için farklı renklerde elektrik bandı veya etiketleme bayrakları kullanabilirsiniz. Hangi sistemi seçerseniz seçin, dokümantasyonunuzu nasıl tamamladığınızı anladığınızdan ve ekibinizdeki herkesin buna aşina olduğundan emin olun.

36.1.7 Tüm kabloları ve bağlantıları kontrol edin

Robot üzerindeki tüm kablolama tamamlandıktan sonra, her şeyin sağlam olduğundan emin olmak için her bir bağlantıyı çekerek kontrol ettiğinizden emin olun. Ek olarak, herhangi bir bağlantı noktasından dışarıya kaçan tel “bıyıkları” çıkmadığından ve açıkta yalıtılmamış bağlantı olmadığından emin olun. Test sırasında herhangi bir bağlantı gevşerse veya herhangi bir “bıyık” fark edilirse, bağlantıyı yeniden yapın ve tamamlandığında ikinci bir kişinin kontrol etmesini sağlayın.

Kötü bağlantıların yaygın bir kaynağı vidalı veya somun ve cıvata bağlantı elemanlarıdır. Robot üzerinde bu türden herhangi bir bağlantı için (örneğin, pil bağlantıları, ana kesici, PDP, roboRIO), bağlantı elemanlarının sıkı olduğundan emin olun. Somun ve cıvata tarzı bağlantılar için, telin/terminalin elle döndürülemeyeceğinden emin olun; Akü telinizi veya ana kesici bağlantınızı terminali kavrayarak ve bükerek döndürebiliyorsanız, bağlantı yeterince sıkı değildir.

Diğer bir yaygın arıza kaynağı, PDP’nin sonundaki sigortalardır. Bu sigortaların tam olarak oturduğundan emin olun; onları tamamen oturtmak için beklediğinizden daha fazla kuvvet uygulamanız gerekebilir. Sigortalar düzgün bir şekilde yerleştirilirse, elle çıkarılması muhtemelen zor veya imkansız olacaktır.

SB-50 konektörü gibi geçmeli bağlantılar, darbeler sırasında gevşememelerini sağlamak için klipsler veya kablo bağları kullanılarak sabitlenmelidir.

36.1.8 Erken ve Sık Sık Kontrol Edin

İlk bir veya iki maçı oynadıktan (veya çok güçlü testler yaptıktan) sonra tüm elektrik sistemini mümkün olduğunca iyice kontrol edin. Robotun gördüğü ilk birkaç darbe, bağlantı elemanlarını gevşetebilir veya sorunları ortaya çıkarabilir.

Elektrik bağlantılarını düzenli olarak yeniden kontrol etmek için bir kontrol listesi oluşturun. Çok kaba bir başlangıç noktası olarak, pil ve PDP bağlantıları gibi rotasyonel bağlantı elemanları her 1-3 maçta bir kontrol edilmelidir. WAGO ve Weidmuller konektörleri gibi yay tipi bağlantıların muhtemelen olay başına yalnızca bir kez kontrol edilmesi gerekir. Ekibin kontrol listesini tamamlamaktan kimin sorumlu olduğunu ve bunun yapıldığını nasıl belgeleyeceğini bildiğinden emin olun.

36.1.9 Akü Bakımı

Akülerinize iyi bakın! Kötü bir akü, bir robotun bir maç sırasında kolayca kötü çalışmasına veya hiç çalışmamasına neden olabilir. Etkinlik sırasında kullanımınızı takip etmenize yardımcı olması için tüm akülerinizi etiketleyin. Birçok ekip bu etikete pilin yaşı gibi bilgileri ekler.

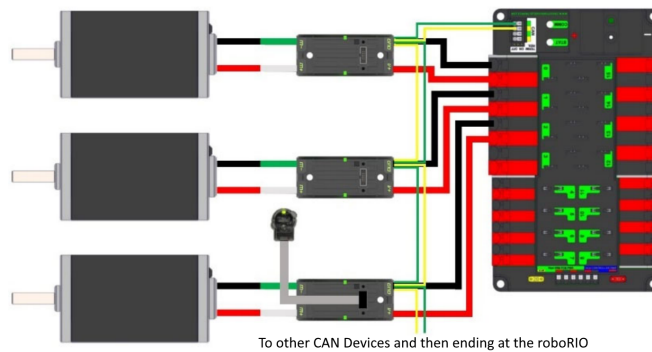
- Aküleri asla kablolarından kaldırmayın veya taşımayın! Pilleri tellerle taşımak, terminaler ve plakalar arasındaki iç bağlantıya zarar verme potansiyeline sahiptir, bu da iç direnci önemli ölçüde artırır ve performansı düşürür.
- Tam bir test yapıncaya kadar düşen aküyü kötü olarak işaretleyin. Bahsedilen terminal bağlantılarına ek olarak, bir akünün düşürülmesi de hücrelere zarar verme potansiyeline sahiptir. Bu hasar basit bir voltaj testinde anlaşılmayabilir, bunun yerine akü yük altına alınana kadar saklanabilir.
- Aküleri eşit şekilde değiştirin. Bu, pillerin şarj ve dinlenme için en fazla zamana sahip olmalarını ve eşit şekilde aşınmalarını sağlamaya yardımcı olur (eşit sayıda şarj/deşarj döngüsü)
- Mümkünse akülerin sağlamlığını test için yüke bağlayın. En az bir tanesi FRC için özel olarak tasarlanmış olmak üzere, takımların bataryalarını yüklemek için kullandığı ticari olarak temin edilebilen bir dizi ürün vardır. Bir yük testi, iç direnci ölçerek pil sağlığının bir göstergesi olabilir. Bu ölçüm, performans eşleşmesi söz konusu olduğunda, bir multimetre tarafından sağlanan basit bir yüksüz voltaj ölçümünden çok daha anlamlıdır.

36.1.10 DS Günlüklerini Kontrol Edin

Her maçtan sonra, pil voltajı ve akım kullanımının neye benzediğini görmek için DS günlüklerini inceleyin. Robotunuz için bu öğelerin normal aralığının ne olduğunu belirledikten sonra, potansiyel sorunları (bozuk piller, arızalı motorlar, mekanik tıkanmalar) kritik arızalara dönüşmeden önce tespit edebilirsiniz.

36.2 CAN Bağlantısının Temelleri

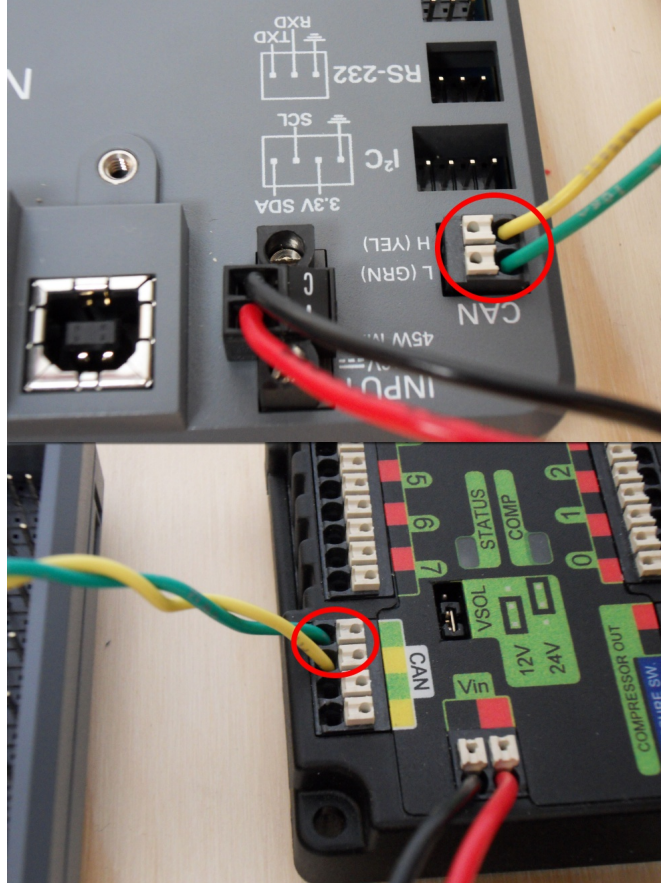
CAN is a two wire network that is designed to facilitate communication between multiple devices on your robot. It is recommended that CAN on your robot follow a “daisy-chain” topology. This means that the CAN wiring should usually start at your roboRIO and go into and out of each device successively until finally ending at the *PDP*.



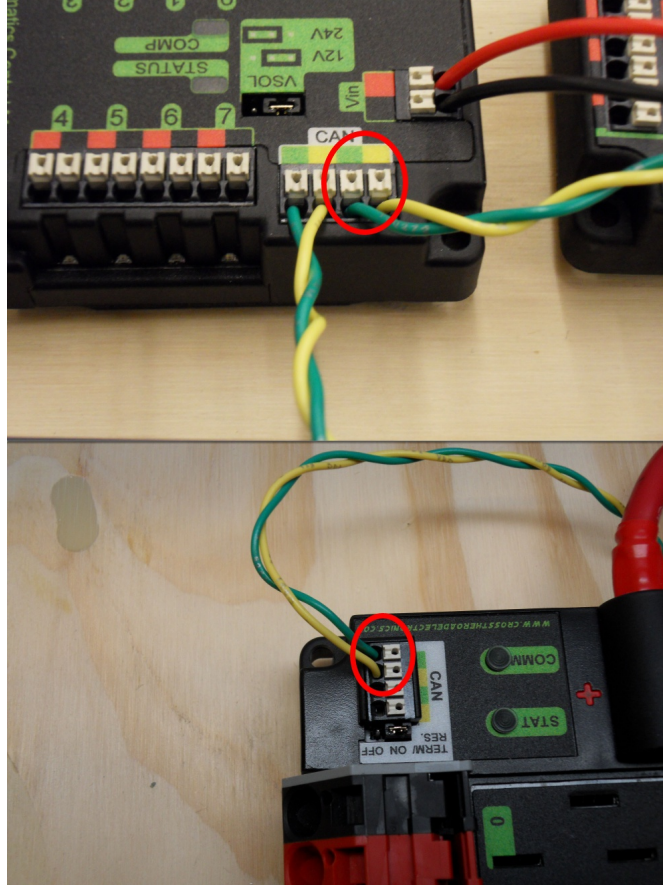
36.2.1 Standart Bağlantı

CAN genellikle sarı ve yeşil kablolarla, sarı CAN-Yüksek ve yeşil CAN-Alçak sinyalleri olacak şekilde bağlanır. Birçok cihaz, kabloların nasıl takılması gerektiğini belirtmek için bu sarı ve yeşil renk şemasını gösterir.

roboRIO'dan PCM'e CAN Bağlantısı



PCM'den PDP'ye CAN bağlantısı



36.2.2 Sonlandırma

CAN ağının 120 Ω dirençleri ile sonlandırılması gerektiğinden ve bunlar bu iki cihazda yerleşik olduğundan, kablolanmanın roboRIO'da başlayıp PDP'de bitmesi önerilir. PDP, "ON" konumunda CAN bus sonlandırma direnci jumper'ı ile birlikte gönderilir. Jumper'ı bu konumda bırakmanız ve ek CAN düğümlerini roboRIO ile PDP arasına yerleştirmeniz önerilir (PDP'yi veriyolunun sonu olarak bırakarak). PDP'yi veri yolunun ortasına yerleştirmek istiyorsanız (her iki PDP CAN terminali çiftini kullanarak) jumper'ı "KAPALI" konuma getirin ve kendi 120 Ω sonlandırma direncinizi CAN veri yolu zinciri kablonuzun sonuna yerleştirin.

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](#).

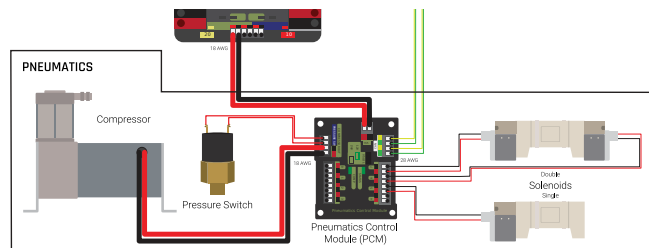
İpucu: For pneumatics safety & mechanical requirements, consult this year's Robot Construction rules. For mechanical design guidelines, the FIRST Pneumatics Manual is located [here](#)

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.

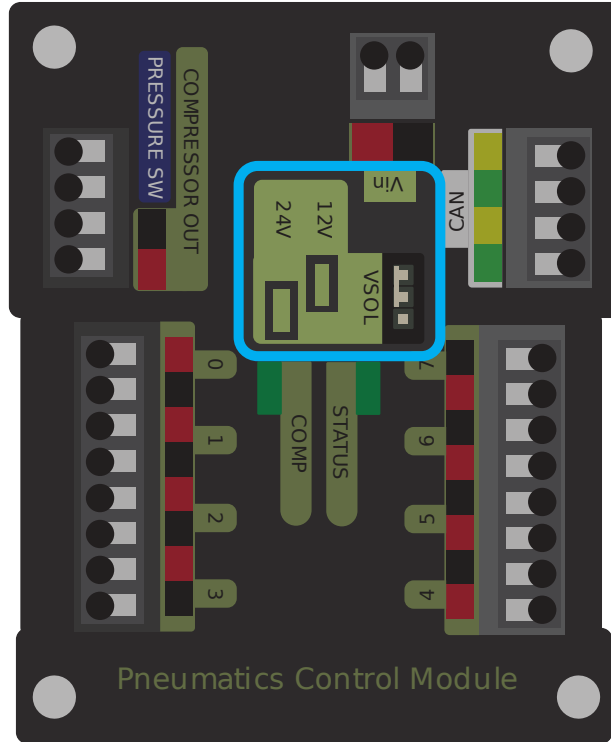
36.3.4 The Pressure Switch

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](#).

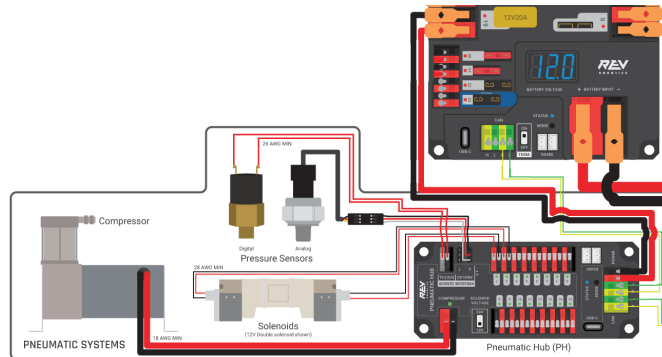
Ipucu: For pneumatics safety & mechanical requirements, consult this year's Robot Construction rules. For mechanical design guidelines, the FIRST Pneumatics Manual is located [here](#)

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.

Uyarı: 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 Durum Işığı için Hızlı Referans

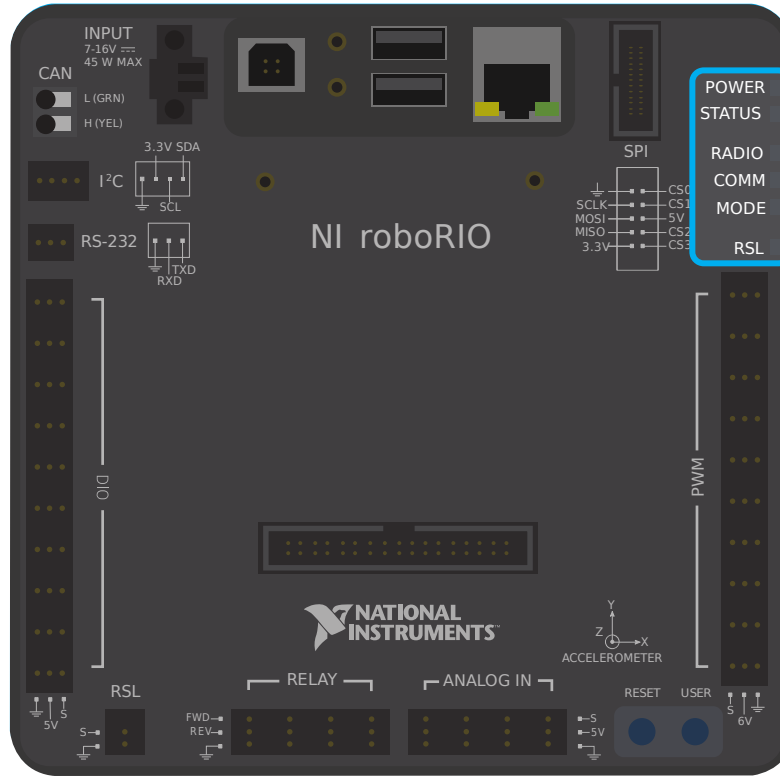
FRC® Kontrol Sistemi'nin birçok bileşeninde, robotunuzla ilgili sorunları hızlı bir şekilde teşhis etmek için kullanılabilecek gösterge ışıkları bulunur. Bu kılavuz, her bir donanım bileşenini gösterir ve göstergelerin anlamını açıklar. Fotoğraf ve bilgiler Innovation FIRST ve Cross the Road Electronics'ten alınmıştır.

36.5.1 Robot Sinyal Işığı - Robot Signal Light (RSL)



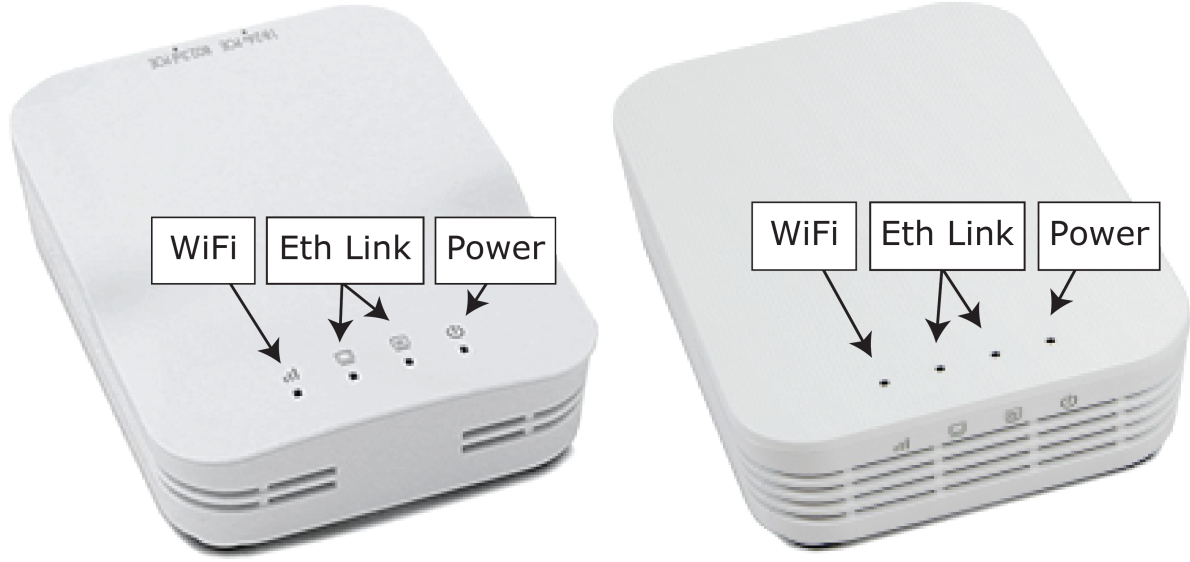
Sabit AÇIK	ON	-	Robot On - Açık ve Disabled - Devre Dışı
Yanıp Sönüyor			Robot On - Açık ve Enabled - Etkin
Off - Kapalı			Robot Off - Kapalı, roboRIO'ya güç verilmedi veya RSL uygun şekilde bağlanmadı

36.5.2 roboRIO



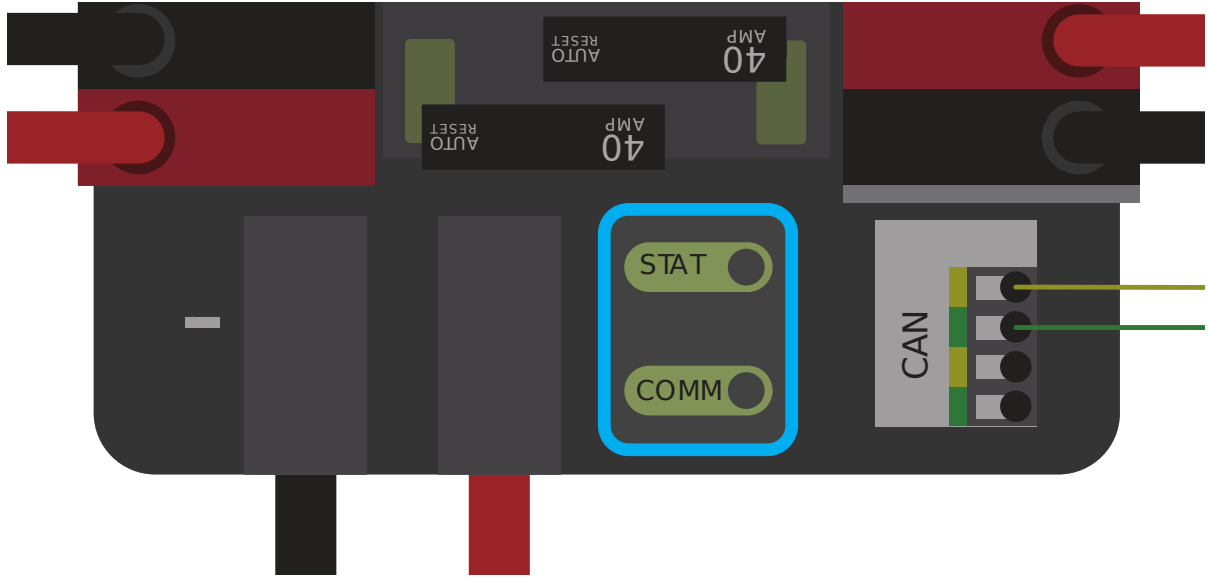
Po- wer	Yeşil	Güç iyi durumda
	Kehribar - Sarımsı Turuncu	Kısa devre-yanma koruması tetiklendi, çıkışlar devre dışı
	Kırmızı	Güç hatası, kısa devre için kullanıcı hatlarını kontrol edin
Sta- tus	Denetleyici başlatılırken açıktır, sonrasında sönmelidir	
	2 kere yanıp sönmeye	Yazılım hatası, roboRIO'ya yeniden imaj atın
	3 kez yanıp sönmeye	Safe mode - Güvenli Mod, roboRIO'yu yeniden başlatın, çözülmediyse yeniden imaj atın.
	4 kez yanıp sönmeye	Yazılım yeniden başlatılmadan iki kez çöktü, roboRIO'yu yeniden başlatın, çözülmediyse yeniden imaj atın
	Sürekli yanıp sönmeye veya sabit kalıyor	Kurtarılamaz hata
Ra- dio	Şu anda uygulanmıyor	
Com	Off - Kapalı	İletişim yok
	Sabit Kırmızı	DS ile iletişim var ancak çalışan kullanıcı kodu yok
	Kırmızı Yanıp Sönüyor	E-stop tetiklendi
	Yeşil Sabit	DS ile iletişim iyi
Mo- de	Off - Kapalı	Çıkışlar devre dışı (robot devre dışı modda, kesinti, vb.)
	Turuncu	Otonom Etkin
	Yeşil	Teleop Etkin
	Kırmızı	Test Etkin
RSL	Yukarıya bakın	

36.5.3 OpenMesh Radyo



Güç	Mavi	On - Açık veya Açılıyor
	Mavi Yanıp Sönüyor	Açılıyor
Eth Bağlantısı	Mavi	Bağlantı mevcut
	Mavi Yanıp Sönüyor	Trafik Mevcut
Kablosuz internet - WiFi	Off - Kapalı	Bridge - Köprü modu, Bağlantısız veya FRC olmayan ürün yazılımı
	Kırmızı	AP, Bağlantısız
	Sarı/Turuncu	AP, Bağlantılı
	Yeşil	Bridge - Köprü modu, Bağlantılı

36.5.4 Power Distribution Panel



PDP Status/Comm - İletişim LED'leri

LED	Flaş	Yavaş
Yeşil	Hata Yok - Robot Etkin	Hata Yok - Robot Devre Dışı
Turuncu	NA	Yapışkan Hata
Kırmızı	NA	CAN İletişimi yok

Tüyo: Bir PDP LED'i birden fazla renk gösteriyorsa, aşağıdaki PDP LED özel durum tablosuna bakın. PDP hatalarının çözümü hakkında daha fazla bilgi için PDP Kullanıcı Kılavuzuna bakın.

Not: Note that the No **CAN** Comm fault will occur if the PDP cannot communicate with the roboRIO via CAN Bus.

PDP Özel Durumları

LED Renkleri	Sorun
Kırmızı/Turuncu	Hasarlı Donanım
Yeşil/Turuncu	Bootloader'da
LED yok	Güç Yok/Yanlış Kutup bağlantısı

36.5.5 Power Distribution Hub



Not: These led patterns only apply to firmware version 21.1.7 and later

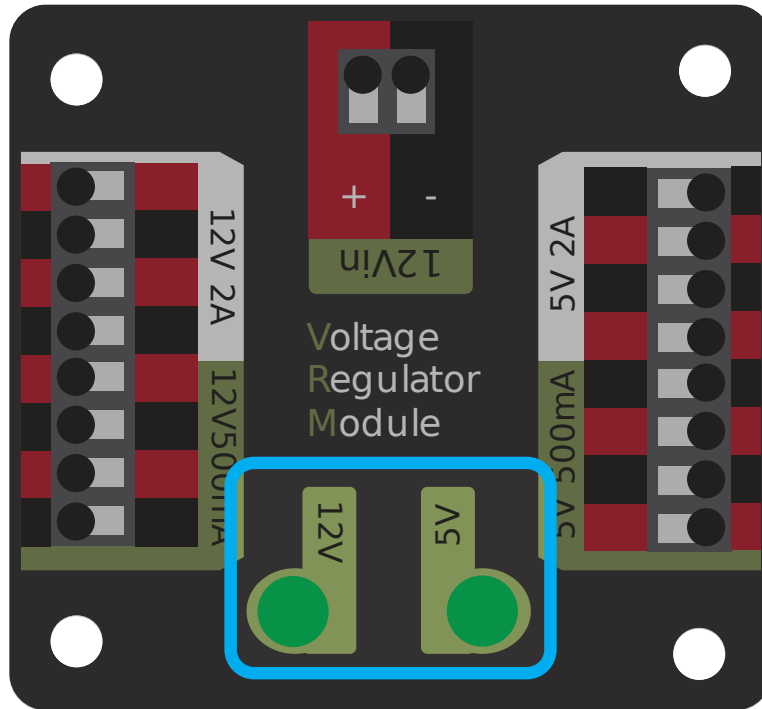
36.5.6 PDH Status LED

LED Color	Status - durum
Blue Solid	Device on but no communication established
Yeşil Sabit	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

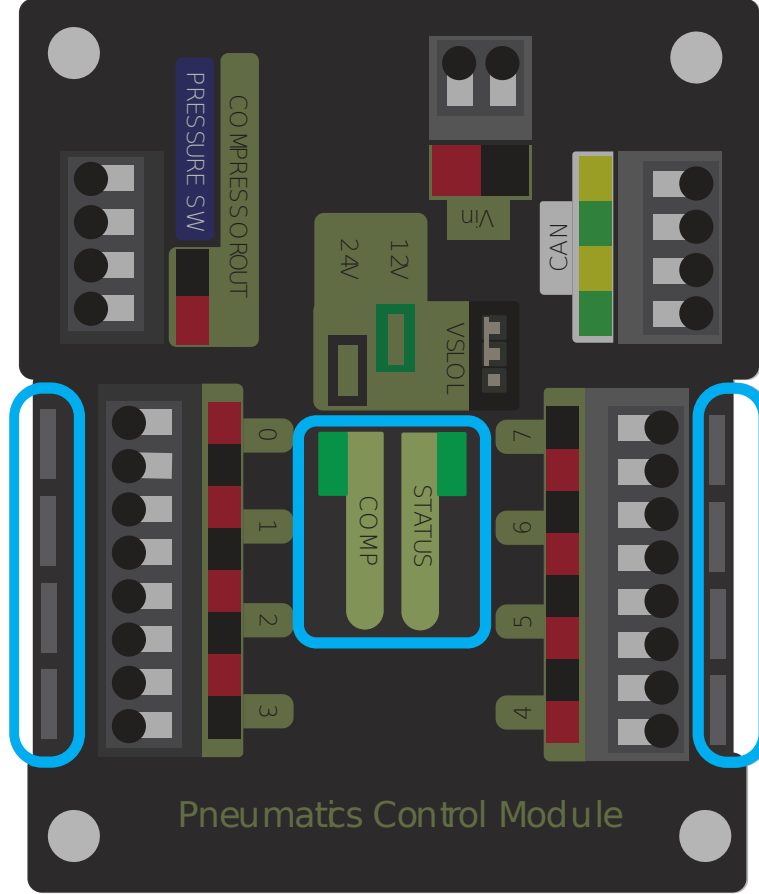
LED Color	Status - durum
Off - Kapalı	Channel has voltage and is operating as expected
Sabit Kırmızı	Channel has NO voltage and there is an active fault. Check for tripped or missing circuit breaker / fuse
Kırmızı Yanıp Sönüyor	Sticky fault on the channel. Check for tripped circuit breaker / fuse.

36.5.7 Voltage Regulator Module - Voltaj Regülatör Modülü



VRM üzerindeki durum LED'leri iki güç kaynağının durumunu gösterir. Besleme düzgün çalışıyorsa, LED parlak yeşil yanmalıdır. LED yanmıyorsa veya soluksa, çıkış kısa devre olabilir veya çok fazla akım çekebilir.

36.5.8 Pneumatics Control Module (PCM) - Pnömatik Kontrol Modülü



PCM Durum LED'i

LED	Flaş	Yavaş	Uzun
Yeşil	Hata Yok - Robot Etkin	Yapışkan Hata	NA
Tu-run-cu	NA	Yapışkan Hata	NA
Kır-mızı	NA	CAN İletişimi veya Solenoid Hatası Yok (Solenoid İndeksi Yanıp Sönüyor)	Kompresör Arızası

Tüyo: Bir PCM LED'i birden fazla renk gösteriyorsa, aşağıdaki PCM LED özel durum tablosuna bakın. PCM hatalarının çözümü hakkında daha fazla bilgi için PCM Kullanıcı Kılavuzuna bakın.

Not: CAN İletişimi Yok arızasının sadece cihaz başka bir cihazla iletişim kuramazsa değil, PCM ve PDP birbirleriyle iletişim kurup roboRIO ile iletişim kuramazsa da ortaya çıkacağına dikkat edin.

PCM LED Özel Durum Tablosu

LED	Sorunlar
Kırmızı/Turuncu	Hasarlı Donanım
Yeşil/Turuncu	Bootloader'da
LED yok	Güç Yok/Yanlış Kutup Bağlantısı

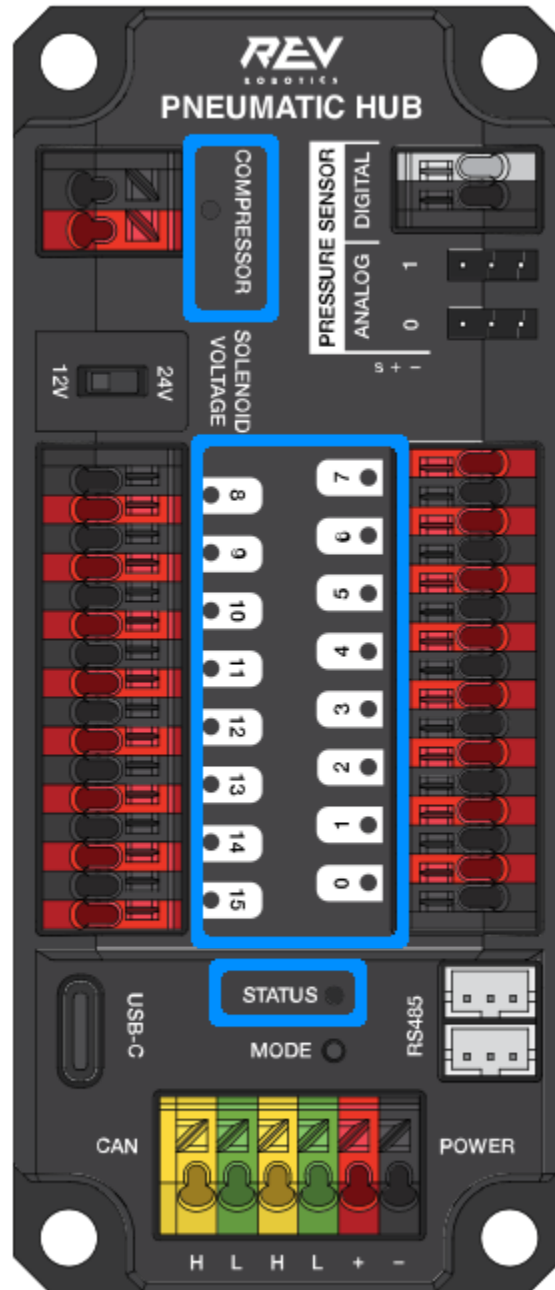
PCM Comp LED'i

Bu, Kompresör LED'idir. Bu LED, kompresör çıkışı aktif olduğunda yeşildir (kompresör şu anda açık) ve kompresör çıkışı aktif olmadığında kapalıdır.

PCM Solenoid Kanal LED'leri

Solenoid kanalı etkinse bu LED'ler kırmızı yanar ve devre dışı bırakılırsa yanmaz.

36.5.9 Pneumatic Hub



Not: These led patterns only apply to firmware version 21.1.7 and later

PH Status LED

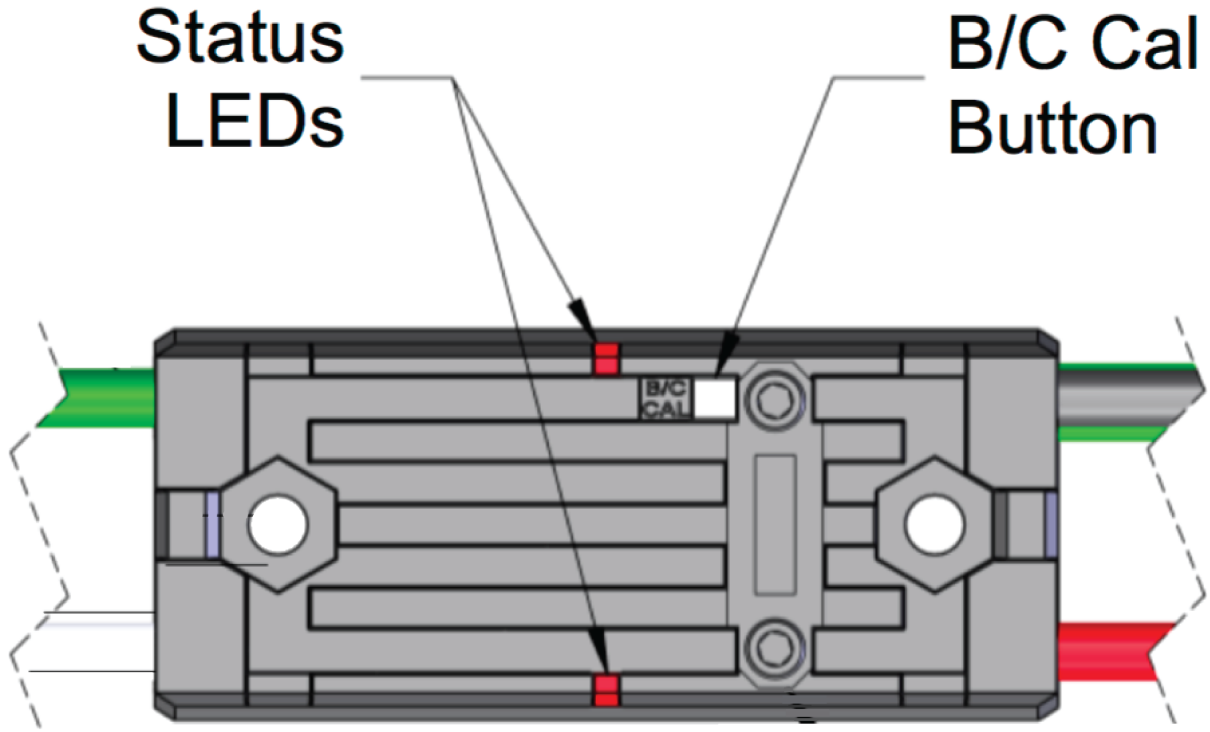
LED Color	Status - durum
Blue Solid	Device on but no communication established
Yeşil Sabit	Main Communication established
Magenta Blinking	Keep Alive Timeout
Solid Cyan	Secondary Heartbeat (connected to REV HW Client)
Orange/Blue Blinking	Hardware Fault
Orange/Yellow Blinking	CAN Fault
Orange/Red Blinking	Fail Safe
Orange/Magenta Blinking	Device Over Current
Orange/Green Blinking	Orange/Green Blinking

Compressor LED

LED Color	Status - durum
Yeşil Sabit	Compressor On
Black Solid	Compressor Off

Solenoid LEDs

LED Color	Status - durum
Yeşil Sabit	Solenoid On
Black Solid	Solenoid Off

36.5.10 Talon SRX & Victor SPX & Talon FX Motor Sürücüleri

Normal Çalışma Sırasında Status-Durum LED'leri

LED'ler	Renkler	Talon SRX Durumu
Her ikisi de	Yanıp Sönen Yeşil	İleri yönlü güç uygulandı. Yanıp sönme oranı, Görev Döngüsü-duty cycle ile orantılıdır.
Her ikisi de	Yanıp Sönen Kırmızı	Ters yönlü güç uygulandı. Yanıp sönme oranı, Görev Döngüsü-duty cycle ile orantılıdır.
Yok	Yok	Talon SRX'e güç uygulanmıyor
LED'ler değişiyor	Kapalı/Turuncu	CAN bus algılandı, robot devre dışı
LED'ler değişiyor	Kapalı/Yavaş Kırmızı	CAN veriyolu/PWM algılanmadı
LED'ler değişiyor	Kapalı/Hızlı Kırmızı	Arıza Tespit Edildi
LED'ler değişiyor	Kırmızı/Turuncu	Hasarlı Donanım
LED'ler (M-) yönünde yanar	Kapalı/Kırmızı	İleri Limit Anahtarı veya İleri Yumuşak Limit
LED'ler (M +) yönünde yanar	Kapalı/Kırmızı	Ters Limit Anahtarı veya Ters Yumuşak Limit
Yalnızca LED1 (M+/V+'ya en yakın)	Yeşil/Turuncu	Önyükleyicide
LED'ler (M +) yönünde yanar	Kapalı/Turuncu	Termal Hata / Kapatma (Yalnızca Talon FX)




















Kalibrasyon Sırasında Durum LED'leri

Durum LED'leri Yanıp Sönme Kodu	Talon SRX Durumu
Yanıp Sönen Kırmızı/Yeşil	Kalibrasyon Modu
Yanıp Sönen Yeşil	Başarılı Kalibrasyon
Yanıp Sönen Kırmızı	Başarısız Kalibrasyon

B/C CAL Yanıp Sönme Kodları

B/C CAL Düğme Rengi	Talon SRX Durumu
Sabit Kırmızı	Fren Modu
Off - Kapalı	Boşta Modu

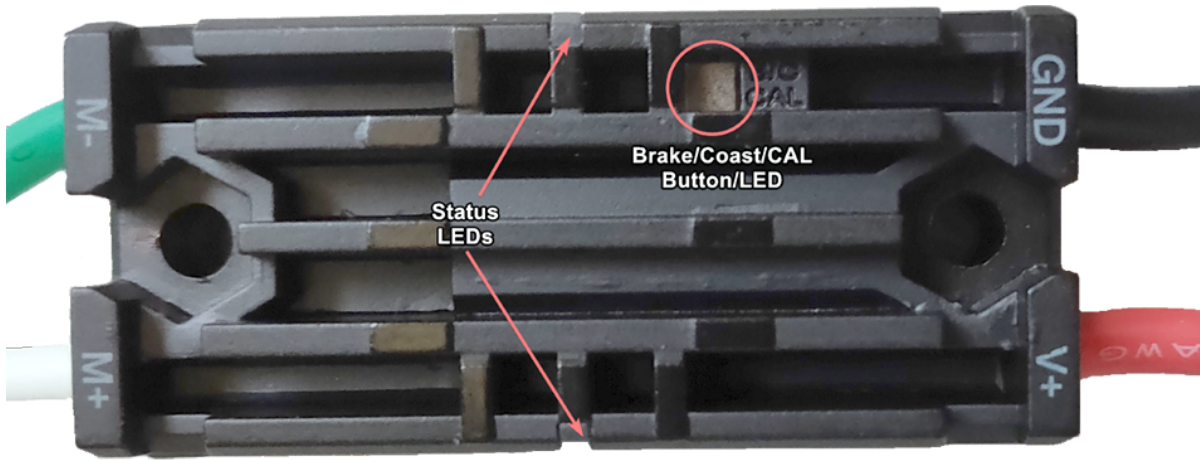
36.5.11 SPARK-MAX Motor Sürücüsü

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

		LED Status Code	
Time Scale		1 second	1 second
State		Normal Operation	
No Signal	Brake	[Blue][Black][Blue][Black][Blue][Black][Blue][Black]	
	Coast	[Yellow][Black][Yellow][Black][Yellow][Black][Yellow][Black]	
Full Forward		[Green][Green][Green][Green][Green][Green][Green][Green]	
Proportional Forward		[Green][Black][Green][Black][Green][Black][Green][Black]	
Neutral	Brake	[Blue][Blue][Blue][Blue][Blue][Blue][Blue][Blue]	
	Coast	[Yellow][Yellow][Yellow][Yellow][Yellow][Yellow][Yellow][Yellow]	
Proportional Reverse		[Red][Black][Red][Black][Red][Black][Red][Black]	
Full Reverse		[Red][Red][Red][Red][Red][Red][Red][Red]	
Forward Limit Tripped		[Green][Black][Green][Black][Green][Black][Green][Black]	
Reverse Limit Tripped		[Red][Black][Red][Black][Red][Black][Red][Black]	
		Calibration	
Calibration Mode		[Black][Black][Black][Black][Black][Black][Black][Black]	
Successful Calibration		[Green][Black][Green][Black][Green][Black][Green][Black]	
Failed Calibration		[Red][Black][Red][Black][Red][Black][Red][Black]	
		Factory Reset	
		Mode button held during power up	Mode button released
Reset to Factory Defaults		[Black][Black][Black][Black][Black][Black][Black][Black]	[Green][Green][Green][Green][Green][Green][Green][Green]

36.5.13 Victor-SP Motor Sürücüsü



Brake/Coast/Cal Düğmesi/LED - Denetleyici fren-brake modundaydı kırmızı, denetleyici boşta-coast modundaydı söner

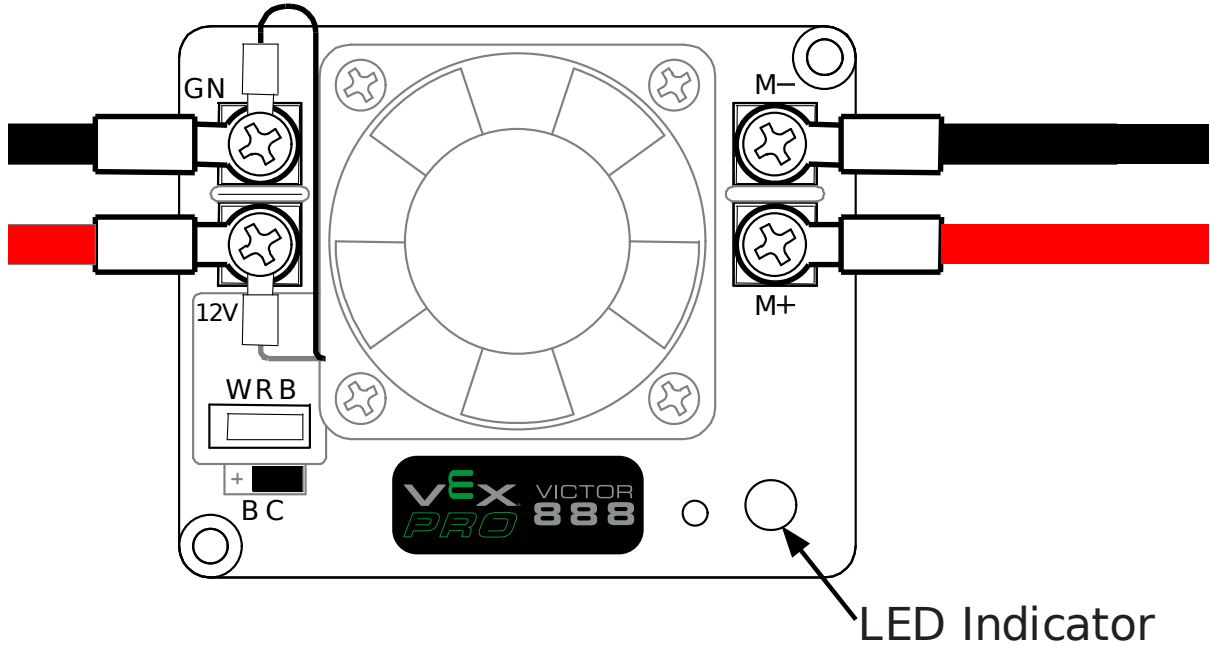
Status - durum

Yeşil	Sabit	Tam ileri çıkış
	Yanıp Sönüyor	İleri çıkış voltajı ile orantılı
Kırmızı	Sabit	Tam ters çıkış
	Yanıp Sönüyor	İleri çıkış voltajı ile orantılı
Turuncu	Sabit	FRC robotu devre dışı, PWM sinyali kayboldu veya sinyal ölü bant aralığında (+/-% 4 çıkış)
Kırmızı/yeşil	Yanıp Sönüyor	Kalibrasyona hazır. Birkaç yeşil yanıp sönme, başarılı kalibrasyonu gösterir ve birkaç kez kırmızı, başarısız kalibrasyonu gösterir.

36.5.14 Talon Motor Sürücüsü

Yeşil	Sabit	Tam ileri çıkış
	Yanıp Sönüyor	İleri çıkış voltajı ile orantılı
	Sabit	Tam ters çıkış
Kırmızı	Sabit	Tam ters çıkış
	Yanıp Sönüyor	Geri çıkış voltajı ile orantılı
	Sabit	Tam ters çıkış
Turuncu	Sabit	CAN cihazı bağlı değil
	Yanıp Sönüyor	Devre dışı durumu, PWM sinyali kayboldu, FRC robotu devre dışı bıraktı veya ölübant aralığında sinyal (+/-% 4 çıkış)
	Sabit	Tam ters çıkış
Off - Kapalı		Talon'a giriş gücü yok
Kırmızı/yeşil	Yanıp sönüyor	Kalibrasyona hazır. Birkaç yeşil yanıp sönme, başarılı kalibrasyonu gösterir ve birkaç kez kırmızı, başarısız kalibrasyonu gösterir.

36.5.15 Victor888 Motor Sürücüsü



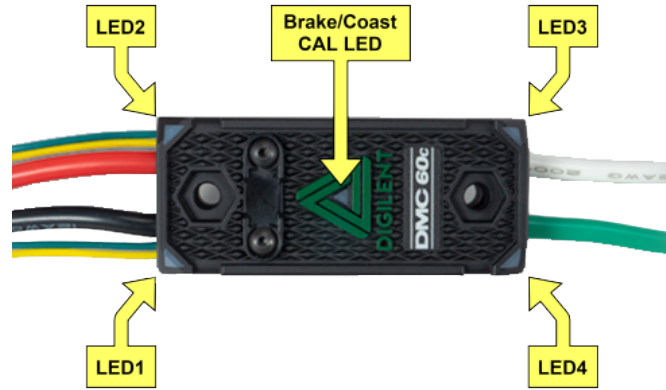
Yeşil	Sabit	Tam ileri çıkış
	Yanıp Sönüyor	Başarılı kalibrasyon
Kırmızı	Sabit	Tam ters çıkış
	Yanıp Sönüyor	Başarısız kalibrasyon
Turuncu	Sabit	Nötr/fren
Kırmızı/yeşil	Yanıp Sönüyor	Kalibrasyon modu

36.5.16 Jaguar Motor Sürücüsü



LED Durumu	Modül Durumu
Normal Çalışma Koşulları	
Sabit Sarı	Nötr (hız 0'a ayarlı)
Hızlı Yanıp Sönen Yeşil	İleri
Hızlı Yanıp Sönen Kırmızı	Geri
Sabit yeşil	Tam hızda ileri
Sabit Kırmızı	Tam hızlı geri
Arıza Durumları	
Yavaş Yanıp Sönen Sarı	Servo veya Ağ bağlantısı kaybı
Hızlı Yanıp Sönen Sarı	Geçersiz CAN ID'si
Yavaş Yanıp Sönen Kırmızı	Voltaj, Sıcaklık veya Limit Anahtarı arızası durumu
Yavaş Yanıp Sönen Kırmızı ve Sarı	Akım arıza durumu
Kalibrasyon veya CAN Durumları	
Yanıp Sönen Kırmızı ve Yeşil	Kalibrasyon modu etkin
Yanıp Sönen Kırmızı ve Sarı	Kalibrasyon modu hatası
Yanıp Sönen Yeşil ve Sarı	Kalibrasyon modu başarılı
Yavaş Yanıp Sönen Yeşil	CAN ID atama modu
Hızlı Yanıp Sönen Sarı	Mevcut CAN ID (ID'yi belirlemek için yanıp sönmeleri sayın)
Yanıp Sönen Sarı	CAN ID geçersiz (yani 0'a ayarlı) geçerli bir kimlik atamasını bekliyor

36.5.17 Digilent DMC-60



DMC60C, dört RGB (Kırmızı, Yeşil ve Mavi) LED ve bir Fren/Boşta- Brake/Coast CAL LED'i içerir. Dört RGB LED köşelerde bulunur ve normal çalışma sırasında, ayrıca bir arıza meydana geldiğinde durumu belirtmek için kullanılır. Fren/Boşta CAL LED'i, gövdenin ortasında bulunan üçgenin ortasında bulunur ve mevcut Fren/Boşta ayarını göstermek için kullanılır. Ortadaki LED söndüğünde, cihaz boşta modunda çalışmaktadır. Ortadaki LED yandığında, cihaz fren modunda çalışmaktadır. Fren/Boşta modu, üçgenin ortasına aşağıya doğru basılarak ve ardından düğme bırakılarak değiştirilebilir.

Açıldığında, RGB LED'ler Mavi renkte yanarak sürekli daha parlak hale gelir. Bu yaklaşık beş saniye sürer. Bu süre boyunca, motor sürücü bir giriş sinyaline yanıt vermeyecek ve çıkış sürücüleri etkinleştirilmeyecektir. İlk açılış tamamlandıktan sonra, cihaz normal çalışmaya başlar ve RGB LED'lerinde görüntülenen şey, uygulanan giriş sinyalinin yanı sıra mevcut arıza durumunun bir fonksiyonudur. Hata oluşmadığını varsayarsak, RGB LED'leri aşağıdaki gibi çalışır:

PWM Sinyali Uygulandı	LED Durumu
Giriş Sinyali Yok veya Geçersiz Giriş Sinyali Genişliği	Kırmızı ve Kapalı olarak yanan üst (LED1 ve LED2) ve alt (LED3 ve LED4) LED'ler arasında geçiş.
Nötr Giriş Sinyali Genişliği	4 LED'in tümü Turuncu renkte yanar.
Pozitif Giriş Sinyali Genişliği	LED'ler saat yönünde dairesel bir şekilde Yeşil renkte yanıp söner (LED1 → LED2 → LED3 → LED4 → LED1). LED güncelleme hızı, çıkışın görev döngüsü ile orantılıdır ve artan görev döngüsü ile artar. % 100 görev döngüsünde 4 LED'in tümü Yeşil yanar.
Negatif Giriş Sinyali Genişliği	LED'ler saat yönünün tersine dairesel bir düzende Kırmızı yanıp söner (LED1 → LED4 → LED3 → LED2 → LED1). LED güncelleme hızı, çıkışın görev döngüsü ile orantılıdır ve artan görev döngüsü ile artar. % 100 görev döngüsünde 4 LED'in tümü Kırmızı renkte yanar.

CAN Bus Kontrol Durumu	LED Durumu
Giriş Sinyali Yok veya CAN bus hatası tespit edildi	Kırmızı ve Kapalı olarak yanan üst (LED1 ve LED2) ve alt (LED3 ve LED4) LED'ler arasında geçiş.
Son 100 ms içinde hiçbir CAN Kontrol Frame'i alınması veya son kontrol çerçevesi modeNoDrive (Çıkış Devre Dışı) gönderdi.	Turuncu ve sönük olarak yanan üst (LED1 ve LED2) ve alt (LED3 ve LED4) LED'ler arasında geçiş.
Son 100 ms içinde geçerli CAN Control Frame alındı. Belirtilen kontrol modu, Motor Çıkışına bir Neutral Duty Cycle uygulanmasına neden oldu	4 LED'in tümü sürekli Turuncu renkte yanar.
Son 100 ms içinde geçerli CAN Control Frame alındı. Belirtilen kontrol modu, Positive Duty Cycle'in Motor Çıkışı olmasına neden oldu	LED'ler saat yönünde dairesel bir şekilde Yeşil renkte yanıp söner (LED1 → LED2 → LED3 → LED4 → LED1). LED güncelleme hızı, çıkışın görev döngüsü ile orantılıdır ve artan görev döngüsü ile artar. % 100 görev döngüsünde 4 LED'in tümü Yeşil yanar.
Son 100 ms içinde geçerli CAN Control Frame alındı. CAN Kontrol Çerçevesi. Belirtilen kontrol modu, Negative Duty Cycle'in Motor Çıkışı olmasına neden oldu	LED'ler saat yönünün tersine dairesel bir düzende Kırmızı yanıp söner (LED1 → LED4 → LED3 → LED2 → LED1). LED güncelleme hızı, çıkışın görev döngüsü ile orantılıdır ve artan görev döngüsü ile artar. % 100 görev döngüsünde 4 LED'in tümü Kırmızı renkte yanar.

Arıza Renk Göstergeleri

Bir arıza durumu tespit edildiğinde, çıkış görev döngüsü %0'a düşürülür ve arıza sinyali bildirilir. Çıkış daha sonra 3 saniye devre dışı kalır. Bu süre zarfında yerleşik LED'ler (LED1-4) arıza durumunu göstermek için kullanılır. Arıza durumu, kırmızı renkte yanan ve sönen üst (LED1 ve LED2) ve alt (LED3 ve LED4) LED'ler arasında geçiş yapılarak gösterilir. Altındaki LED'lerin rengi o anda hangi arızaların aktif olduğuna bağlıdır. Aşağıdaki tablo, alttaki LED'lerin renginin halihazırda aktif olan arızalarla nasıl eşleştiğini açıklamaktadır.

Renk	Aşırı Isınma	Düşük Gerilim
Yeşil	On	Off - Kapalı
Mavi	Off - Kapalı	On
Mavi / Açık Mavi	On	On

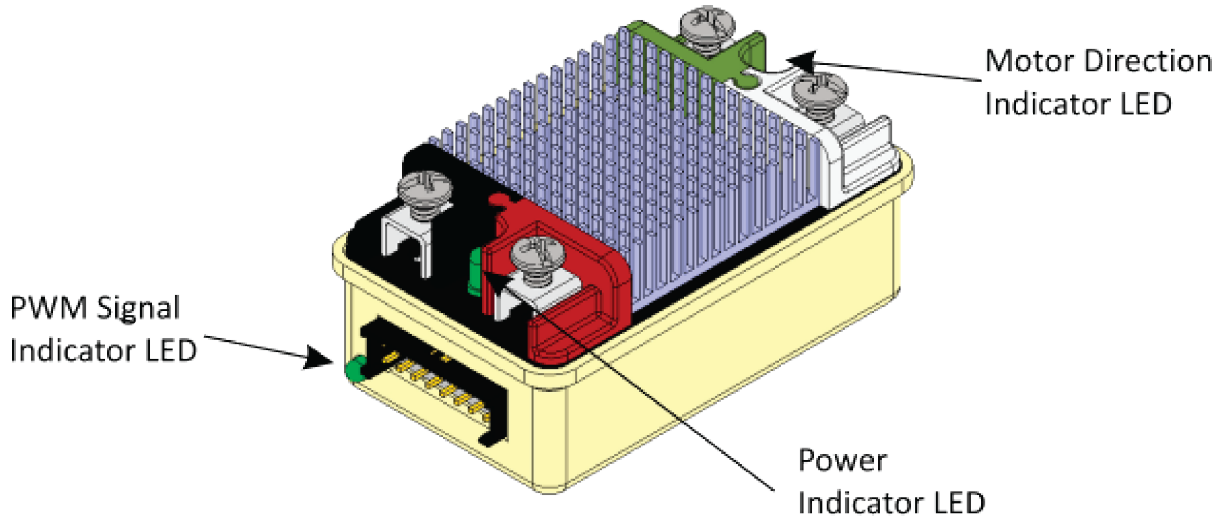
Fren/Boşta Modu

Ortadaki LED söndüğünde, cihaz boşta modunda çalışmaktadır. Ortadaki LED yandığında, cihaz fren modunda çalışmaktadır. Fren/Boşta modu, üçgenin ortasına basılarak ve ardından düğme bırakılarak değiştirilebilir.

36.5.18 Venom Motor Sürücüsü

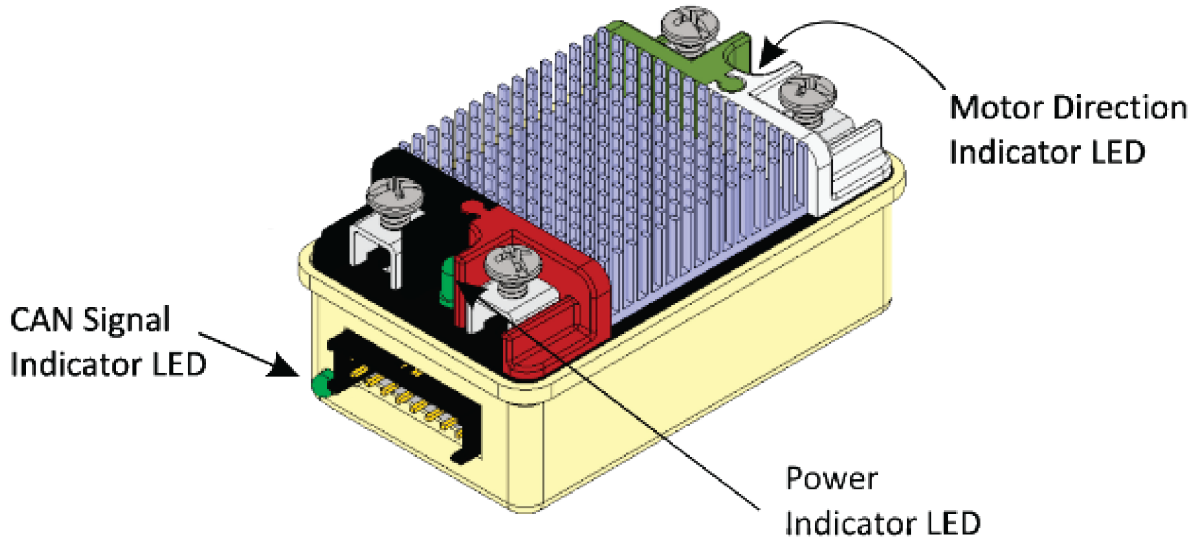
LED Pattern	Description
	Venom is initializing. This state should last less than 40ms after power up.
	The 'Identify Device' feature is active. This pattern is used to locate a particular Playing With Fusion device when multiple are installed on a robot. See the Motor Configuration section for more information.
	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 bus termination.
	Missing heartbeat in CAN control mode. Ensure device ID matches device ID used by CANVenom class. See the Motor Configuration section for more information and instructions to change/verify the device ID.
	Lead motor heartbeat is missing while in Follow The Leader mode.
	The lead motor ID is same as the motor ID. One Venom cannot follow itself. Ensure the leader and follower have different IDs.
	An invalid control mode was specified by the roboRIO. This should not occur when using PlayingWithFusionDriver. Contact PWF Technical support.
	Another Venom with the same device ID was detected on the CAN bus. All Venom device IDs must be unique.
	The forward limit switch is enabled and is active.
	The reverse limit switch is enabled and is active.
	Motor temperature is too high.
	Average motor current is too high.

36.5.19 Mindsensors SD540B (PWM)



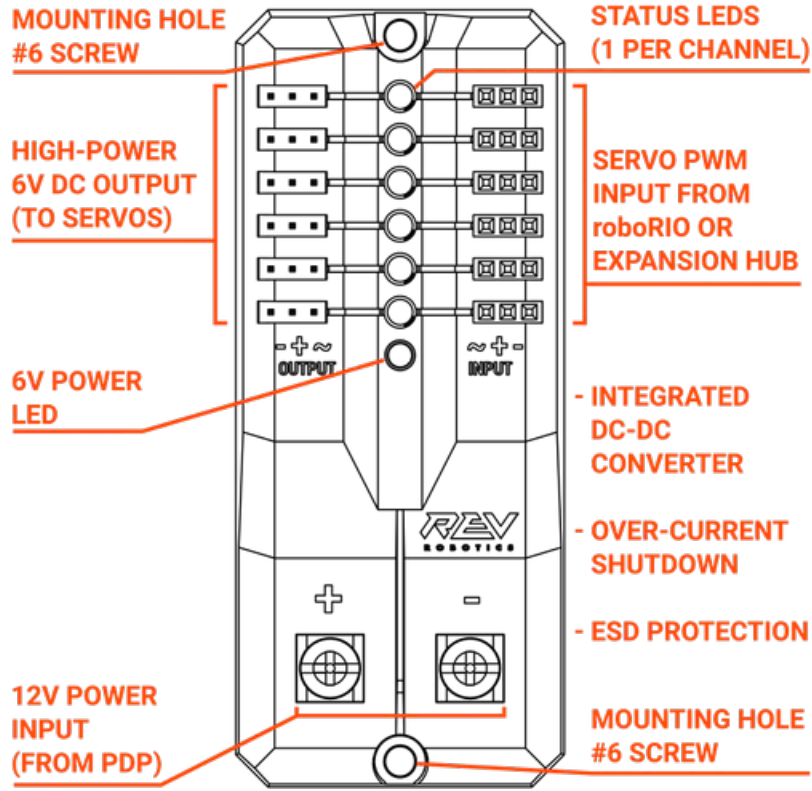
Power LED	Off - Kapalı	Güç uygulanmadı
	Kırmızı	Güç uygulandı
Motor LED	Kırmızı	İleri yönde
	Yeşil	Geri yön
PWM Sinyal LED	Kırmızı	Geçerli bir PWM sinyali algılanmadı
	Yeşil	Geçerli PWM sinyali algılandı

36.5.20 Mindsensors SD540C (CAN Bus)



Power LED	Off - Kapalı	Güç uygulanmadı
	Kırmızı	Güç uygulandı
Motor LED	Kırmızı	İleri yönde
	Yeşil	Geri yön
CAN Sinyal LED	Hızla yanıp sönüyor	CAN cihazı bağlı değil
	Off - Kapalı	RoboRIO'ya bağlı ve sürücü istasyonu açık

36.5.21 REV Robotics Servo Güç Modülü



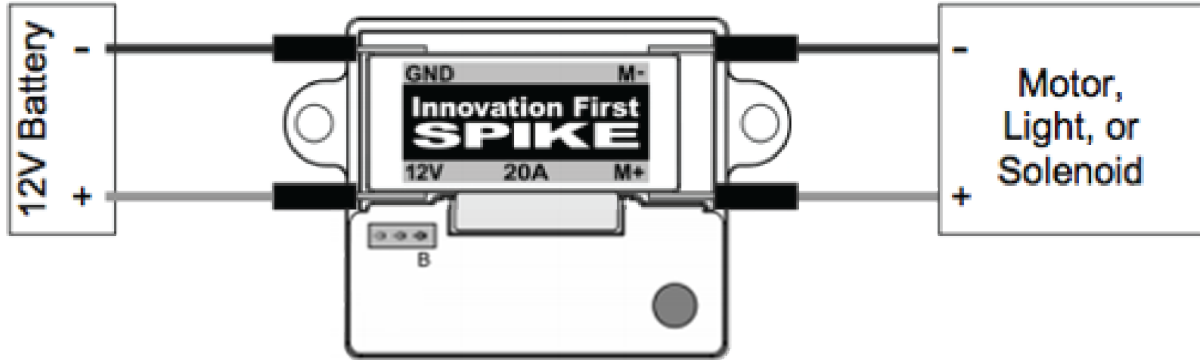
Durum LED'leri

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.

Durum	Desen
Sinyal yok	Yanıp Sönen Koyu sarı
Sol/Geri Sinyal	Sabit Kırmızı
Merkez/Nötr Sinyal	Sabit Koyu Sarı
Sağ/İleri Sinyal	Sabit yeşil

- 6V Güç LED'i kapalı, güç uygulandığında kararıyor veya titriyor = Aşırı akım koruma/kapatma

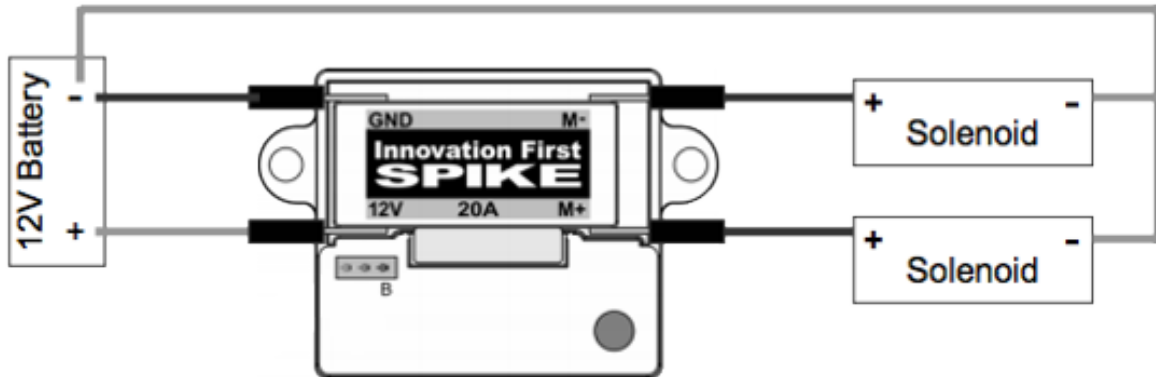
36.5.22 Motor, ışık veya solenoid anahtarı olarak yapılandırılmış Spike rölesi



Girişler		Çıkışlar		Gösterge	Motor fonksiyon
İleri (Be-yaz)	Geri (Kırmızı)	M+	M-		
Off - Kapalı	Off - Kapalı	GND	GND	Turuncu	Kapalı/Fren Durumu (varsayılan)
On	Off - Kapalı	+12v	GND	Yeşil	Motor tek yönde dönüyor
Off - Kapalı	On	GND	+12v	Kırmızı	Motor ters yönde dönüyor
On	On	+12v	+12v	Off - Kapa-lı	Kapalı/Fren Durumu

Not: 'Fren Durumu', motor girişlerindeki kısa devre nedeniyle motorun dinamik olarak durdurulmasını ifade eder. Bu koşul kapalı duruma geçerken isteğe bağlı değildir.

36.5.23 Bir veya iki solenoid için yapılandırılmış Spike rölesi



Girişler		Çıkışlar		Gösterge	Motor fonksiyon
İleri (Be-yaz)	Geri (Kırmızı)	M+	M-		
Off - Kapalı	Off - Kapalı	GND	GND	Turuncu	Her İki Solenoid Kapalı (varsayılan)
On	Off - Kapalı	+12v	GND	Yeşil	M+'ya bağlı solenoid AÇIK
Off - Kapalı	On	GND	+12v	Kırmızı	M-'ye bağlı solenoid AÇIK
On	On	+12v	+12v	Off - Kapalı	Her iki Solenoid AÇIK

36.5.24 CANCoder Encoder



LED Color	LED Brightness	CAN Bus detection	Magnet Field Strength	Description
Off - Kapalı	Off - Kapa-lı			CANCoder is not powered
Yel-low/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 Robot Önleyici Sorun Giderme

Not: FIRST® Robotik Yarışması'nda, robotlar sahada dolaşırken çok fazla baskıya mağruz kalır. Bağlantıların sıkı olduğundan, parçaların yerine iyice sabitlendiğinden ve her şeyin monte edildiğinden emin olmak önemlidir, böylece sahada gezinen bir robot kırılmaz.

36.6.1 Akü Bağlantılarını Kontrol Edin

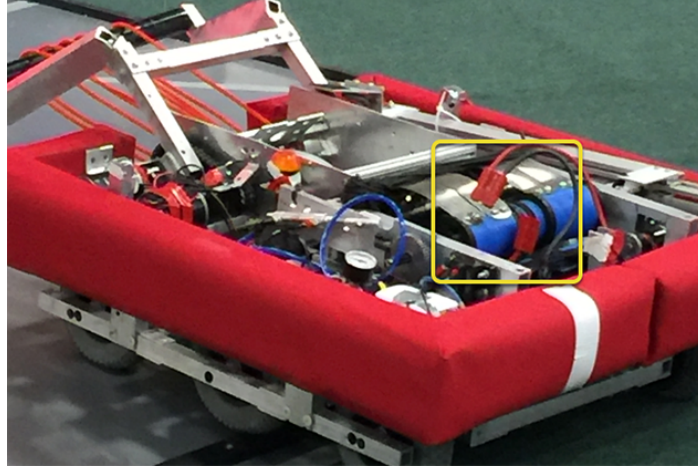


The tape that should be covering the battery connection in these examples has been removed to illustrate what is going on. On your robots, the connections should be covered.

Akü kablolarını sallayarak kontrol edin. Bazen vidalar gevşer veya bazen bağlantı tamamen kapanmaz. Yine de yalnızca çok kötü durumda olanları fark edeceksiniz çünkü çoğu zaman elektrik bandı bağlantıyı sağlam hissettirecek şekilde sertleştirir. Bir voltmetre veya Battery Beak kullanmak bu konuda yardımcı olacaktır.

Apply considerable force onto the battery cable at 90 degrees to try to move the direction of the cable leaving the battery, if successful the connection was not tight enough to begin with and it should be redone. This [article](#) has more detailed battery information.

36.6.2 Aküyü Robota Sabitlemek



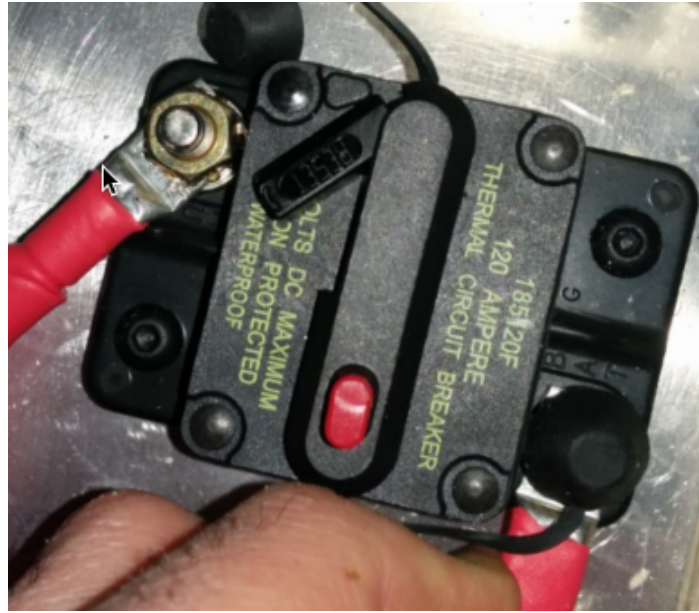
In almost every event we see at least one robot where a not properly secured battery connector (the large Anderson) comes apart and disconnects power from the robot. This has happened in championship matches on the Einstein and everywhere else. Its an easy to ensure that this doesn't happen to you by 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 Pil konektörünü ve ana güç kablolarının güvenliğini sağlamak

A loose robot-side battery connector (the large Anderson SB) can allow the main power leads to be tugged when the battery is replaced. If the main power leads are loose, that "tug" can get all the way back to the crimp lugs attached to the 120 Amp Circuit Breaker or Power Distribution Panel (PDP), bend the lug, and over time cause the lug end to break from fatigue. Putting a couple tie wraps attaching the main power leads to the chassis and bolting down the robot-side battery connector can prevent this, as well as make it easier to connect the battery.

36.6.4 Ana sigorta (120 Amper devre kesicisi)

Not: Somunların iyice sıkıldığından ve devre kesici anahtarın sert bir elemana bağlandığından emin olun.



Apply a strong twisting force to try to rotate the crimped lug. If the lug rotates then the nut is not tight enough. After tightening the nut, retest by once again trying to rotate the lug.

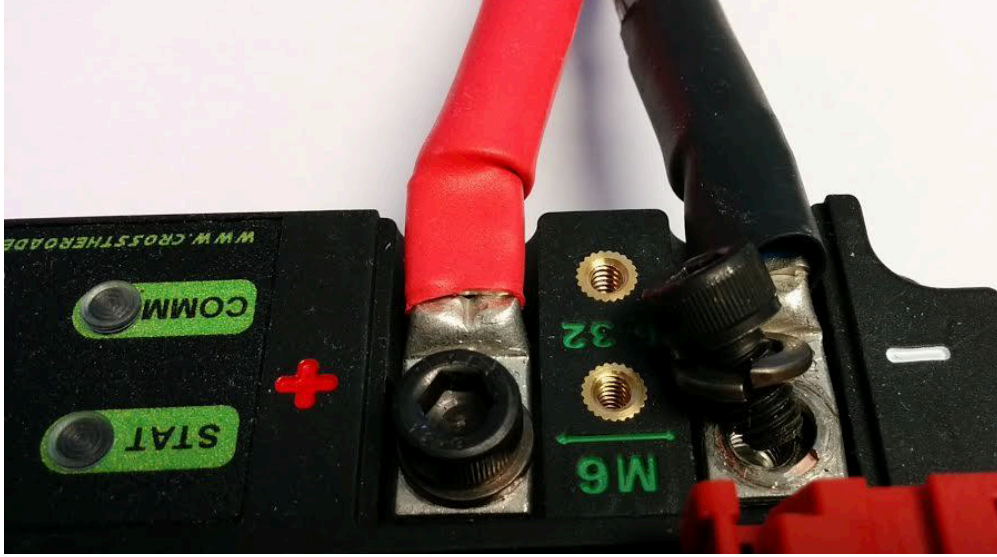
The original nut has a star locking feature, which can wear out over time: these may require checking every few matches, especially if your robot-side battery connector is not attached to the chassis.

The nut is normally a relatively uncommon 1/4-28 thread: ensure this is correct if the nut is replaced.

Because the metal stud is just molded into the case, every once in awhile you may break off the stud. Don't stress, just replace the assembly.

When subjected to multiple competition seasons, the Main Breaker is susceptible to fatigue damage from vibration and use, and can start opening under impact. Each time the thermal fuse function is triggered, it can become progressively easier to trip. Many veteran teams start each season with a fresh main breaker, and carry spares.

36.6.5 Power Distribution Panel (PDP) - Güç Dağıtım Paneli



PDP vidalarına yaylı rondela taktığınızdan emin olun, ancak bakarak anlamak kolay değildir ve bazen bunu yapamazsınız. Kapağını kaldırarak kontrol edebilirsiniz. Ayrıca kırmızı ve siyah kabloları birlikte tutup sıkıştırırsanız, bazen gerçekten gevşek bağlantıları yakalayabilirsiniz.

36.6.6 Çekme Testi





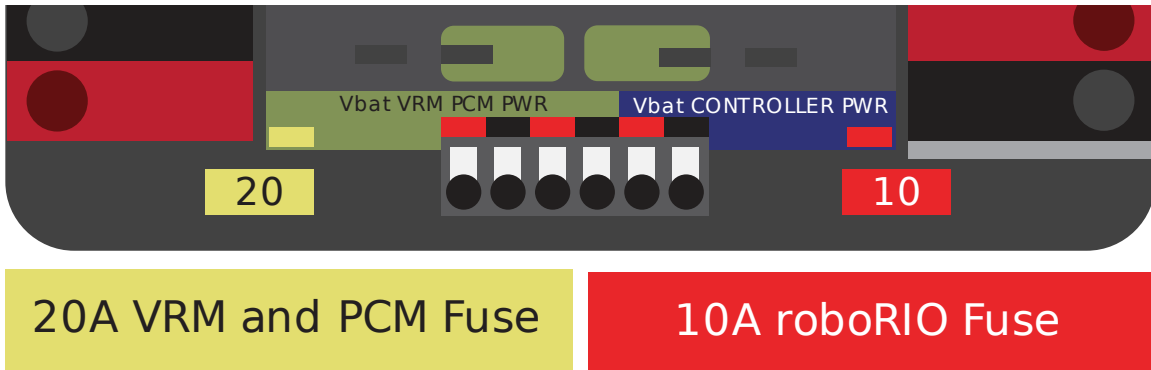
Güç, kompresör çıkışı, roboRIO güç konektörü ve radyo gücü için Weidmuller bağlantılarını gösterildiği gibi kabloları çekerek doğrulamak önemlidir. Bağlantılardan hiçbirinin çıkmadığından emin olun.

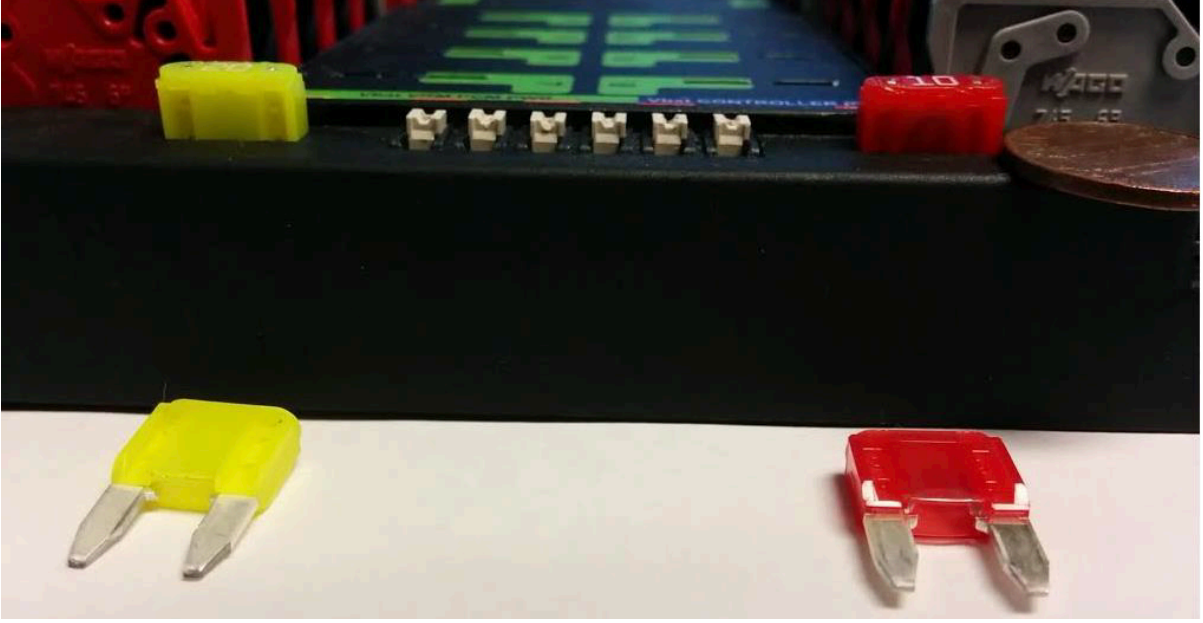
Birbirine yakın ve çok uzun kablo ucu uzunluklarına (ekstra uzun sıyrılmış kablolar) sahip Weidmuller bağlantılarını olası veya yaklaşan kısa devreler için kontrol edin.

Takıp çıkarılabilen kablo pabuçları da yanlış sıkıştırma nedeniyle bağlanamayabilir, bu yüzden bunları da çekerek test edin.

36.6.7 Bıçak Sigortalar

20A sigortayı (sarı) sola ve 10A sigortayı (kırmızı) sağa yerleştirdiğinizden emin olun.





Uyarı: Sigortaların sigorta yuvalarına tam olarak oturduğundan emin olun. Sigortalar en azından aşağıdaki şekil kadar yuvanın içine girmelidir (farklı marka sigortaların farklı bıçak uzunlukları vardır). Sigortayı çıplak elle çıkarmak neredeyse imkansız olmalıdır (pense kullanmadan). Bu doğru şekilde yapılmazsa, robot / radyo bağlantısı kesinti sorunları gösterebilir.

Bıçak sigortalarını elle çıkarabiliyorsanız, o zaman tamamen içeride değillerdir. Robot çalışması sırasında dışarı çıkmamaları için PDP'ye tamamen oturduklarından emin olun.

36.6.8 roboRIO üzerinde metal talaş

Metal talaş, bir parça işlenirken üretilen taş, metal veya diğer malzemelerin ince parçalarıdır. Bazen bütün kontrol sistemi parçaları üzerindeyken robotta değişiklikler yapmanız gerekir. RoboRIO'nun devre kartı uygun bir şekilde kaplanmıştır, ancak bu, metal talaşların kasa içindeki devre yolları veya bileşenlerine temas edip kısa devre yapmayacağını garanti etmez. Bu durumda, talaşların hiçbirinin roboRIO'ya veya diğer bileşenlerden herhangi birinin içine kaçmamasına özen göstermelisiniz. Özellikle, açıkta kalan 3 pinli başlık talaşların kasaya girebileceği yerlerdir. Bir el feneri ile dört bir tarafı hızlıca taramak, gerçekten kötü sızma alanlarını bulmak için genellikle yeterlidir.

36.6.9 Radyo Güç Bağlantı Elemanı

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 Ethernet kablosu

RIO'yu radyoya bağlayan ethernet kablosunun konnektörü kilitleyen klipsi yoksa, başka bir kablo alın. Bu, her yarışmada birkaç kez meydana gelen yaygın bir sorundur. Kablolarınızın sağlam olduğundan emin olun. Klips, özellikle de dar bir yerden geçirirken, bir şeye takılır ve kopar,

36.6.11 Gevşek Kablolar

Kablolar sıkılmalıdır, özellikle radyo gücü ve ethernet kablosu. Radyo güç kabloları çok fazla sıkılığa sahip değildir ve kablonun serbestçe salınmasına izin verilirse (doğru kablo olsa bile) düşecektir.

Ethernet kablosu da oldukça ağırdır, serbestçe sallanmasına izin verilirse, plastik klips ethernet pin konnektörlerini yuvada tutmak için yeterli olmayabilir.

36.6.12 Sorunları Pitte Simüle Etmek

Robot çalıştırılırken ve bağlıyken kabloların normal sallanmasının ötesinde, robotun bir tarafının kaldırılıp düşürülmesi önerilir. Sahada, özellikle de defans yapan takımlara karşı robot kullanmak, genellikle çok şiddetli olacaktır ve bu önlem, hiçbir şeyin düşmemesine yardımcı olur. Robotun maçın ortasındansa pitte sorun çıkarması daha iyidir.

Bu testi yaparken, USB bağlantılı değil, ethernet bağlantılı olmak önemlidir, aksi takdirde tüm kritik yolları test etmemiş olursunuz.

36.6.13 Fabrika Yazılım ve Sürümleri Kontrol Edin

Robot inspectorları bunu kontrol edeceklerdir, ancak siz de yapmalısınız, robot inspectorlarına yardımcı olur ve bu onları mutlu edecektir. Ayrıca bu işlem en son, düzeltilmiş kodla çalıştığınızı garanti eder. Robotunuzdaki eski bir kontrol sistemi yazılımı nedeniyle bir maçı kaybetmek istemezsiniz.

36.6.14 Driver Station Kontrolleri

Drivers Stationda sık sık sorunlar görüyoruz. Şunları yapmalısın:

- HER ZAMAN dizüstü bilgisayarınızın şarj kablosunu sahaya getirin, bataryanızın ne kadar iyi olduğu önemli değildir, sahada fişe takmanıza izin verilir.
- Güç ve uyku ayarlarını kontrol edin, uyku ve hazırda bekletme, ekran koruyucuları vb. kapatın.
- USB cihazları için güç yönetimini kapatın (aygıt yöneticisi)

- Ethernet bağlantı noktaları için güç yönetimini kapatın (aygıt yöneticisi)
- Windows Defender'ı kapatın
- Güvenlik duvarını kapatın
- Sahadayken DS/Dashboard dışındaki tüm uygulamaları kapatın.
- Başlat menüsünde (sağ alt taraf) uygulama menüsünde gereksiz hiçbir şeyin çalışmadığını doğrulayın.

36.6.15 Kullanışlı Araçlar



Robotların içinde hiçbir zaman görmek için yeterli ışık yokmuş gibi duruyor, en azından kritik bağlantı noktalarını incelemek için yeterli değil, bu nedenle robotunuzdaki bağlantıları incelemek için elde tutulan bir LED el feneri kullanmayı düşünebilirsiniz. Herhangi bir hırdavat/otomotiv mağazasından temin edilebilirler.

WAGO aleti, çok telli kablolarla Weidmuller bağlantılarını yeniden yapmak için güzel bir araçtır. Genellikle takıma göstermek için bir örnek yaparım ve ardından onlar çok telli kabloyu yerleştirirken beyaz pistonu aşağı bastırmak için WAGO aracını kullanarak gerisini onlara yaptırım. WAGO aracının açısı onu özellikle faydalı kılar.

36.7 Robot Battery Basics

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 Battery

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 Battery Safety & Handling

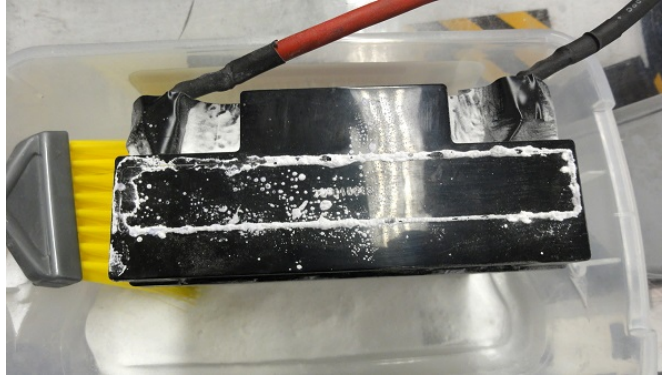
A healthy battery is **always** “On” and the terminals are **always** energized. If the polarities short together - for example, a wrench or aerosol can falls and bridges the gap between two bare terminals - all the stored energy will be released in a dangerous arc. This risk drives a wide range of best practices, such as covering terminals in storage, only uncovering and working on one terminal or polarity at a time, keeping SB contacts fully inserted in connectors, etc.

Do *NOT* carry a battery assembly by the cables, and always avoid pulling by them. Pulling on batteries by the cables will begin to damage the lugs, tabs, and the internal connection of the tab. Over time, fatigue damage can add up until the entire tab tears out of the housing! Even if it isn't clearly broken, internal fatigue damage can increase the battery internal resistance, prematurely wearing out the battery. The battery will not be able to provide the same amount of current with increased internal resistance or if the *connectors are loose*.



Dropping the batteries can bend the internal plates and cause performance issues, create bulges, or even crack the battery case open. While most FRC batteries use Absorbent Glass Mat [AGM] or Gel technology for safety and performance, when a cell is punctured it may still leak a small amount of battery acid. This is one of the reasons FIRST recommends teams have a battery spill kit available.

Finally, certain older battery chargers without “maintenance mode” features can *overcharge* the battery, resulting in boiling off some of the battery acid.



Damaged batteries should be safely disposed of as soon as possible. All retailers that sell large SLA batteries, like car batteries, should be able to dispose of it for you. They may charge a small fee, or provide a small “core charge refund”, depending on your state law.

Tehlike: DO NOT attempt to “repair” damaged or non-functional batteries.

36.7.3 Battery Construction & Tools

Battery Leads

Battery leads must be copper, minimum size (cross section) 6 AWG (16mm², 7 SWG) and maximum length 12”, color coded for polarity, with an Anderson SB connector. Standard 6AWG copper leads with Pink/Red SB50 battery leads often come in the Kit of Parts and are sold by FRC vendors.

Lead Cables

Tinned, annealed, or coated copper is allowed. Do not use CCA (copper clad aluminum), aluminum, or other non-copper base metal. The conductor metal is normally printed on the outside of the insulation with the other cable ratings.

Wire size 6AWG is sufficient for almost all robots and fits standard SB50 contacts. A small number of teams adopt larger wire sizes for marginal performance benefits.

Higher strand count wire (sometimes sold as “Flex” or “welding wire”) has a smaller bend radius, which makes it easier to route, and a higher fatigue limit. There is no strand count requirement, but 84/25 (84 strand “flex” hookup wire) and 259/30 (259 strand “welding wire”) will both be *much* easier to work with than 19/0.0372 (19 strand hookup wire).

The insulation must be color-coded per the Game Manual: as of 2021, the +12Vdc wire must be red, white, brown, yellow, or black w/stripe and the ground wire (return wire) must be black or blue. There is no explicit insulation temperature rating requirement, but any blackened or damaged insulation means the wire needs to be replaced: off hand, 105C is plenty and lower will work for almost all robots. There is no insulation voltage rating requirement, lower is better for thinner insulation.

SB Connector

The Anderson SB Connector may be the standard Pink/Red SB50, or another Anderson SB connector. Teams are *STRONGLY* recommended to use the Pink/Red SB50 for interoperability: the other colors and sizes of housings will not intermate, and you will be unable to borrow batteries or chargers.

Follow manufacturer's instructions to crimp contacts and assemble the leads into Anderson SB connectors. A small flathead screwdriver can help to insert the contacts (push on the contact, not on the wire insulation), or it can help to disengage the internal latch if the contact is in the wrong slot or upside down.

Battery Lugs

Compression lugs ("crimp lugs") for #10 bolt (or M5) battery tabs (~0.2" or ~5mm hole diameter) are available online and through electrical supply houses, sold by the accepted wire sizes in AWG (or mm²) and post diameter ("bolt size", "hole diameter"). Higher end vendors will also distinguish between Standard (~19) and Flex (>80) strand counts in their lug catalogs. Some vendors also offer right angle lugs, in addition to more common straight styles. Follow manufacturer's instructions to crimp the lugs.

Screw terminal lugs are legal, but not recommended. If using screw terminal lugs, use the correct tip size screwdriver to tighten the terminal. Check the terminal tightness frequently because they may loosen over time.

Battery Lead Lug To Post Connection

A #10 or M5 nut & bolt connect the battery lead lug to the battery tab.

Uyarı: The lug and tab must directly contact, copper to copper: do not put a washer of any kind separating them.



Some batteries come with tab bolts in the package: they may be used, or replaced with stronger alloy steel bolts. It is a good idea to add a functional lock washer, such as a #10 star washer or a nordlock washer system, in addition to a nylon locking (“nylock”) nut. Only use one style of lock washer in each connection. Even if the manufacturer provides split ring lock washers in the package, you are not required to use them.



These connections must be very tight for reliability. Any movement of the lug while in operation may interrupt robot power, resulting in robot reboots and field disconnections lasting 30 seconds or more.

This connection must also be completely covered for electrical safety; electrical tape will work, but heatshrink that fits over the entire connection is recommended. High shrink ratios (minimum 3:1, recommend 4:1) will make it easier to apply the heatshrink. Adhesive lined heat shrink is allowed. Be sure *all* the copper is covered! Heat shrink must be “touched up” with electrical tape if some copper shows.



Battery Chargers

There are many good COTS “smart” battery chargers designed for 12V SLA batteries, rated for 6A or less per battery, with ‘maintenance mode’ features. Chargers rated over 6A are not allowed in FRC pits.

Chargers used at competition are required to use Anderson SB connectors. Attaching a COTS SB connector battery lead to the charger leads using appropriately sized wire nuts or screw terminals is fast and simple (be sure to cover any exposed copper with heat shrink or electrical tape). SB Connector Contacts are also available for smaller wire sizes, if the team has crimping capability.

Uyarı: After attaching the SB, double check the charger polarities with a multimeter before plugging in the first battery.

Some FRC vendors sell chargers with red SB50 connectors pre-attached.

Battery Evaluation Tools

Battery Charger

If your battery charger has Maintenance Mode indicator, such as a GREEN LED, you can use that indicator to tell you whether you are READY. Some chargers will cycle between “CHARGING” and “READY” periodically. This is a “maintenance” behavior, sometimes associated with the battery cooling off and being able to accept more charge.

Driver Station Display and Log

When the robot is plugged in and connected to the driver station laptop, the battery voltage is displayed on the NI Driver Station software.

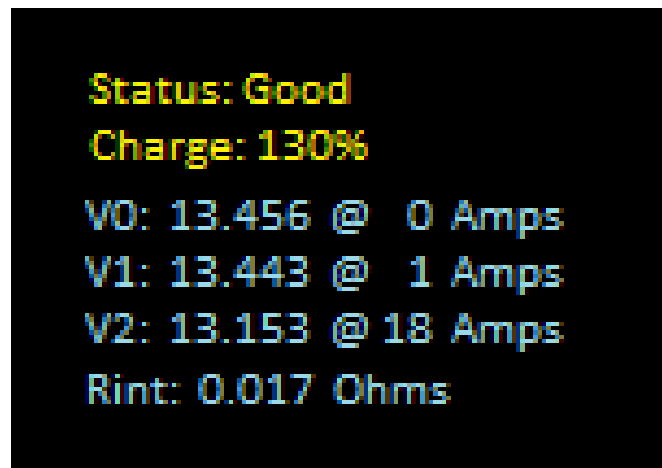
After you finish a driving session, you can *review the battery voltage in the Log Viewer*.

Hand-held Voltmeter or Multimeter

A voltage reading from probes on the SB connector of a disconnected battery will give you a snapshot of what the Voc (Voltage open circuit, or “float voltage”) is in the “Unloaded” state. In general the Voc is not a recommended method for understanding battery health: the open circuit voltage is not as useful as the combination of internal resistance and voltages at specific loads provided by a Load Tester (or Battery Analyzer).

Load Tester

A battery load tester can be used as a quick way to determine the detailed readiness of a battery. It may provide information like: open-load voltage, voltage under load, internal resistance, and state of charge. These metrics can be used to quickly confirm that a battery is ready for a match and even help to identify some long term problems with the battery.



Ideal internal resistance should be less than 0.015 Ohms. The manufacturer specification for most batteries is 0.011 Ohms. If a battery gets higher than 0.020 Ohms it is a good idea to consider not using that battery for competition matches.

If a battery shows significantly lower voltages at the higher test current loads, it may not be done charging, or it may need to be retired.

36.7.4 Understanding Battery Voltages

A “12V battery” is anything but 12.0V.

Fully charged, a battery can be anywhere from 12.7 to 13.5 volts open circuit (Voc). Open circuit voltage is measured with *nothing* connected.

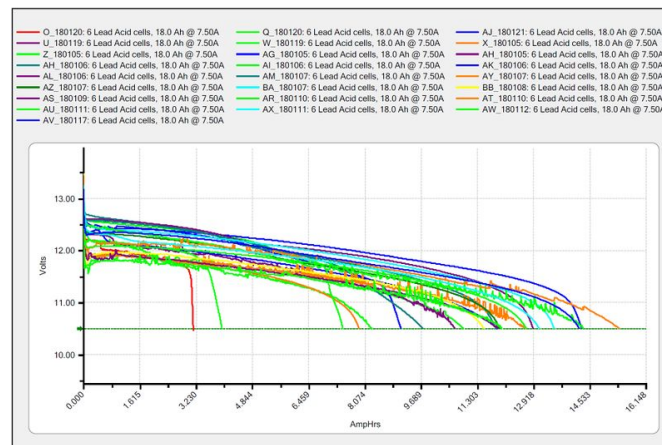
Once a load (like a robot) is connected, and any amount of current is flowing, the battery voltage will drop. So if you check a battery with a Voltmeter, and it reads 13.2, and then connect it to your robot and power on, it will read lower, maybe 12.9 on the Driver Station display. Those numbers will vary with every battery and specific robot, see Characterization below. Once your robot starts running, it will pull more current, and the voltage will drop further.

Batteries reading 12.5V on an idle robot should be swapped and charged before a match. Always swap the batteries before the robot starts reaching brownout safety thresholds (dwelling at low voltages on the Driver Station display), as frequently entering low voltage ranges risks permanent battery damage; this behavior can happen at a variety of Voc states depending on battery health, battery manufacturer, and robot design. The battery State of Charge should be kept over 50% for battery longevity.

Battery voltage and current also depends on temperature: cool batteries are happy batteries.

Battery Characterization

A battery analyzer can be used to give a detailed inspection and comparison of battery performance.



It will provide graphs of battery performance over time. This test takes significant time (roughly two hours) so it is less suited to testing during competition. It is recommended to run this test on each battery every year to monitor and track its performance. This will determine how it should be used: matches, practice, testing, or disposed of.

At the standard 7.5 amps test load, competition batteries should have at least a 11.5 amp hour rating. Anything less than that should only be used for practice or other less demanding use cases.

Battery Longevity

A battery is rated for about 1200 normal charge/recharge cycles. The high currents required for an FRC match reduce that lifespan to about 400 cycles. These cycles are intended to be relatively low discharge, from around 13.5 down to 12 or 12.5 volts. Deep cycling the battery (running it all the way down) will damage it.

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.

Batteries need to be kept away from both extreme heat and cold. This generally means storing the batteries in a climate controlled area: a classroom closet is usually fine, a parking lot shipping container is more risky.

36.7.5 Battery Best Practices

- Only use a charged battery for competition matches. If you are in a situation where you have run out of charged batteries, please ask a veteran team for help! Nobody wants to see a robot dead on the field (*brownout*) due to a bad or uncharged battery.
- Teams are strongly recommended to use properly rated tools and stringent quality control practices for crimping processes (ask local veteran teams or a commercial electrician for help), or use vendor-made Battery Leads.
- Wait for batteries to cool after the match before recharging: the case should not be warm to the touch, fifteen minutes is usually plenty.
- Teams should consider purchasing several new batteries each year to help keep their batteries fresh. Elimination matches can require many batteries and there may not be enough time to recharge.



- A multi bank battery charger allows you to charge more than one battery at a time. Many teams build a robot cart for their batteries and chargers allowing for easy transport and storage.
- It is a good idea to permanently identify each battery with at least: team number, year, and a unique identifier.
- 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.



- Using battery flags (a piece of plastic placed in the battery connector) is a common way to indicate that a battery has been charged. Battery flags can also be easily 3D printed.
- Handles for SB50 contacts can be purchased or 3D printed to help avoid pulling on the leads while connecting or disconnecting batteries. Do not use these handles to carry the weight of the battery.



- Some teams sew battery carrying straps from old seatbelts or other flat nylon that fit around the battery to help prevent carrying by leads.



- Cable tie edge clips can be used with 90 degree crimp lugs to strain relieve battery leads.



Not: Belgelerin bu bölümündeki sayfalar, yalnızca belgelerin web sürümünden görüntülenebilen medyayı içerir.

37.1 Robotik Uygulamalar için Motorlar

Takımların ilgilenmesi gereken en önemli tasarım kararlarından biri, robotlarındaki motor sistemlerini seçmek ve tasarlamaktır. Bu yüzden çoğu zaman, belirli bir tasarım için düşük performans sağlayan yanlış motor seçilir ve bazen daha da kötüsü, aşırı akım çekiminden dolayı arızalanan motorlar da olabilir. Bu video dizisinde, WPI Profesörü Ken Stafford, motorların nasıl çalıştığını, sistemlerin maksimum performansta çalışması için nasıl tasarlanacağını ve bir robot sistemi için örnek bir tasarımı anlatıyor.

37.2 Sensörler ve Algılama

Sensörler ve algılama olmadan robotlar gerçekte radyo kontrollü araçlardır. Sensörler, robotların, robotların mekanik sistemlerinin dahili işleyişini ve ayrıca robotun etrafındaki çevre ile etkileşim kurma yeteneğini anlamalarına olanak tanır. Bu videolarda, WPI Profesörü Craig Putnam bir dizi sensör sınıfını, bunların nasıl kullanıldığını anlatıyor ve uygulamalarınız için hangi sensörlerin en iyi olduğuna dair rehberlik sağlıyor.

37.3 Pnömatikler

Pnömatikler, robotlarda genellikle yeterince kullanılmayan, doğrusal hareket sistemleridir. Motor kullanımına göre pnömatiklerin farklı avantajları vardır. Bu videoda Profesör Ken Stafford, pnömatiğin özelliklerini, robotlarda kullanımlarını ve herhangi bir kullanımı için doğru büyüklükteki sistemi hesaplamayı anlatıyor.

37.4 Güç İletimi

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.

38.1 Sensöre Genel Bakış - Donanım

Not: This section covers sensor hardware, not the use of sensors in code. For a software sensor guide, see [Sensor Overview - Software](#).

Etkili olabilmek için, robotların çevreleri hakkında bilgi toplayabilmeleri genellikle hayati önem taşır. Robota, bulunduğu ortamın durumu hakkında geri bildirim sağlayan cihazlara “sensörler” denir. Sahada konumlandırmadan robot yönlendirmesine ve motor / mekanizma konumlandırmasına kadar her şeyi ölçmek için FRC takımlarının kullanabileceği çok çeşitli sensörler vardır. Sensörlerden yararlanmak, sahada başarı için kesinlikle çok önemli bir beceridir; FRC oyunlarının çoğu “kör” bir robotla gerçekleştirilebilecek görevlere sahipken, en iyi robotlar oyun görevlerini olabildiğince hızlı ve güvenilir bir şekilde yerine getirmek için büyük ölçüde sensörlere güvenir.

Ek olarak, sensörler robot güvenliği için son derece önemli olabilir - birçok robot mekanizması, yanlış kullanıldığında kendilerini kırabilir. Sensörler buna karşı bir koruma sağlar ve robotların, örneğin bir mekanizma bir ani duruşla karşılaşırsa bir motoru devre dışı bırakmasına izin verir.

38.1.1 Sensör Türleri

FRC’de kullanılan sensörler genel olarak iki farklı şekilde kategorize edilebilir: işleve göre ve iletişim protokolüne göre. Temel sınıflandırma, robot tasarımı ile ilgilidir; ikincisi kablolama ve programlama içindir.

İşlevlerine Göre Sensörler

Sensörler, robotun durumunun çeşitli yönleri hakkında geri bildirim sağlayabilir. FRC’de genel olan sensör işlevleri şunları içerir:

- *Yakınlık Anahtarları*
 - Mekanik yakınlık anahtarları (“sınır anahtarları”)
 - Manyetik yakınlık anahtarları
 - Endüktif yaklaşım anahtarları
 - Fotoelektrik yakınlık anahtarları
- Mesafe sensörleri
 - *Ultrasonic sensörler*
 - *Triangulating mesafe ölçerler*
 - *LIDAR*
- Mil dönüş sensörleri
 - *Enkoderler*
 - *Potansiyometreler*
- *Accelerometers-İvme ölçerler*
- *Jiroskoplar*

İletişim Protokolüne Göre Sensörler

Bir sensörün yararlı olabilmesi için roboRIO ile “konuşabilmesi” gerekir. Sensörlerin okumalarını roboRIO’ya iletebileceği birkaç ana yöntem vardır:

- *Analog giriş*
- *Dijital giriş*
- *Seri veri yolu*

Genel olarak, analog ve dijital girişlerle iletişim kuran sensörlerin desteği basittir, seri veri yolu üzerinden iletişim ise daha karmaşık olabilir.

38.2 Analog Girişler - Donanım

Not: This section covers analog input hardware. For a software guide to analog inputs, see [Analog Inputs - Software](#).

Bir *analog sinyal* <https://en.wikipedia.org/wiki/Analog_signal> `_, değeri herhangi bir aralıkta herhangi bir yerde bulunabilen bir sinyaldir. Bu, birkaç ayrıık değerden yalnızca birini alabilen :doc:`digital sinyal<digital-inputs-hardware>, ile tam bir tezat oluşturuyor. RoboRIO'nun analog giriş portları, 0V ile 5V arasındaki değerlere sahip analog sinyallerin ölçülmesine izin verir

Pratikte, bilgisayar gibi dijital bir cihazla (roboRIO gibi) “gerçek” bir analog sinyali ölçmenin bir yolu yoktur. Buna göre, analog girişler aslında 12 bitlik dijital sinyal olarak ölçülür - ancak bu oldukça yüksek bir çözünürlüktür [1] _.

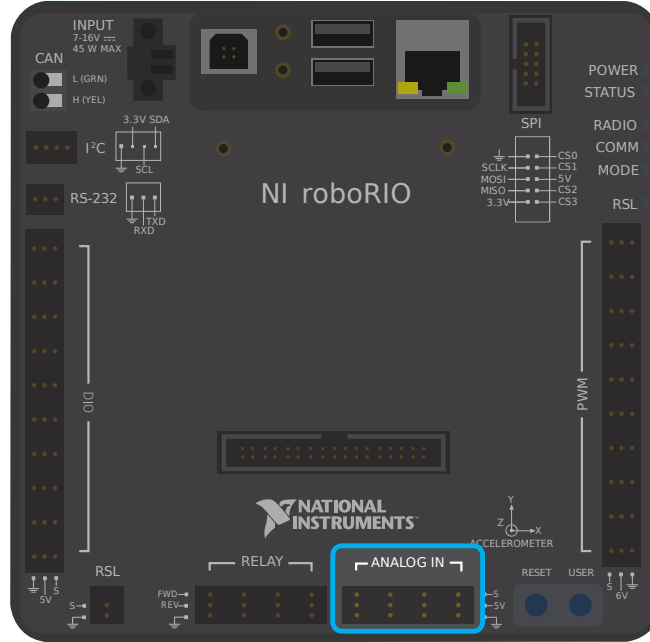
Analog girişler tipik olarak (ancak her zaman değil!), aşağıdaki gibi ölçümleri bir aralıkta sürekli değişen sensörler için kullanılır: *ultrasonic mesafe ölçer* ve *potansiyometre* ölçümleriyle orantılı bir voltaj çıkararak iletişim kurabilirler. .

38.2.1 RoboRIO analog giriş bağlantı noktalarına bağlanma

Not: “MXP” genişletme portu üzerinden ek dört analog giriş mümkündür. Bunları kullanmak için, MXP'ye bağlanan bir çeşit ara devre panosu gereklidir.

Uyarı: Her bir pime doğru kablonun bağlandığından emin olmak için, sensörü kablolamadan *önce* kullandığınız sensörün teknik özellik dökümanlarına daima başvurun. Bunun yapılmaması sensör veya RIO'nun zarar görmesine neden olabilir.

Uyarı: Hiçbir zaman güç pinini roboRIO üzerindeki herhangi bir bağlantı noktasındaki topraklama pinine doğrudan bağlamayın! Bu, roboRIO'daki koruma özelliklerini tetikleyecek ve beklenmeyen davranışlara neden olabilir.

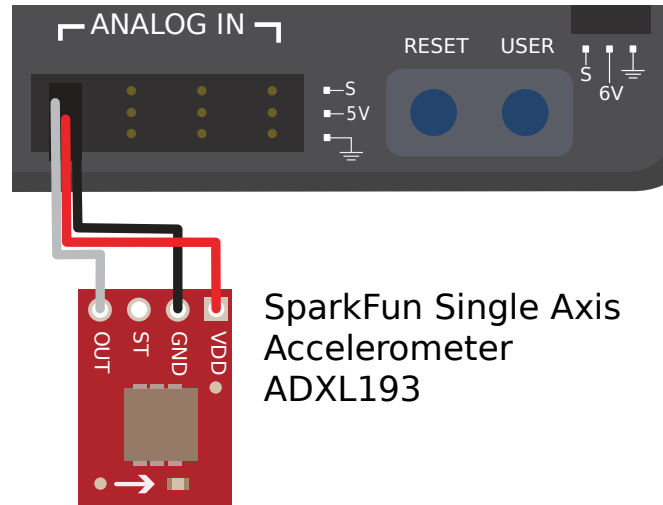


RoboRIO, yukarıdaki resimde görüldüğü gibi 4 dahili analog giriş bağlantı noktasına (0-3 numaralı) sahiptir. Her bağlantı noktasının üç pini vardır - sinyal ("S"), güç ("V") ve toprak ("|**toprak**|"). "Güç" ve "toprak" pinleri, analog giriş bağlantı noktalarına bağlanan dış sensörlere güç sağlamak için kullanılır - "güç" ve "toprak" pinleri [2] _ arasında sabit bir 5V potansiyel farkı vardır. Sinyal pini, sinyalin gerçekte ölçüldüğü pindir.

Bir sensörü tek bir analog giriş bağlantı noktasına bağlama

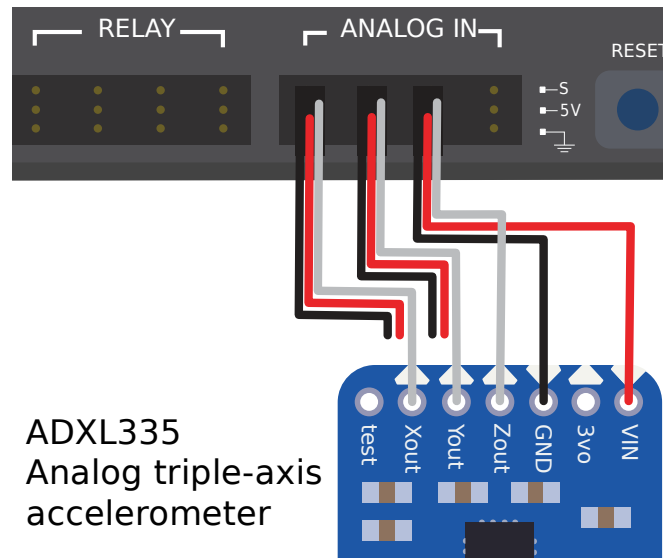
Not: Bazı sensörler (örneğin *potansiyometreler*), değiştirilebilir güç ve toprak bağlantılarına sahip olabilir.

Analog giriş bağlantı noktalarına bağlanan çoğu sensörde, analog giriş bağlantı noktalarının üç pinine tam olarak karşılık gelen üç kablo-sinyal, güç ve toprak-olacaktır. Buna göre bağlanmaları gerekir.



Bir sensörü birden çok analog giriş bağlantı noktasına bağlama

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 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 Dipnotlar

38.3 Analog Potansiyometreler - Donanım

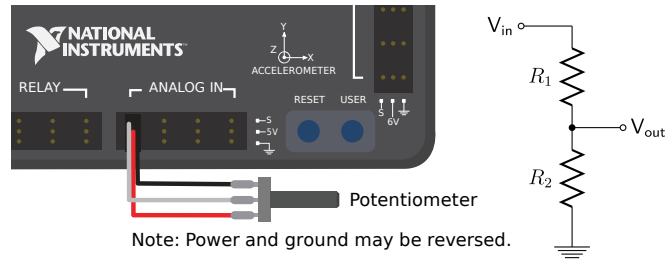
Not: This section covers analog potentiometer hardware. For a software guide to analog potentiometers, see [Analog Potentiometers - Software](#).

Uyarı: Potansiyometreler genellikle mekanik olarak sınırlı bir hareket aralığına sahiptir. Kullanıcılar, mekanizmalarının potansiyometrelerini maksimum hareket mesafesini aşmasına dikkat etmelidir çünkü bu, potansiyometreye zarar verir veya bozar.

quadrature enkoderlerden ayrı olarak, FRC robotlarında dönüşü ölçmenin diğer bir yaygın yolu analog potansiyometrelerdir. Bir potansiyometre basitçe değişken bir dirençtir - potansiyometrenin şaftı döndükçe direnç değişir (genellikle doğrusal olarak). Bu direncin bir *voltaj bölücüye* yerleştirilmesi, kullanıcının potansiyometre boyunca voltajı ölçerek direnci kolayca ölçmesine olanak tanır ve bu daha sonra şaftın dönüş konumunu hesaplamak için kullanılabilir.

38.3.1 Analog potansiyometre kablolama

İsimlerin önerdiği gibi, analog potansiyometreler roboRIO'lara bağlanır *analog giriş* portuna. Bununla birlikte, potansiyometrelerin tam olarak nasıl bağlanacağını anlamak için, iç devrelerini anlamak önemlidir.



Soldaki resim tipik bir potansiyometreyi göstermektedir. RIO'nun analog girişlerinde olduğu gibi üç pin vardır. Ortadaki pin sinyal pini iken, dış pinler hem güç *hem de* toprak olabilir.

Daha önce belirtildiği gibi, bir potansiyometre, sağdaki devre şemasında gösterildiği gibi bir voltaj bölücüdür. Potansiyometre mili döndükçe, R1 ve R2 dirençleri değişir; ancak toplamaları sabit kalır¹. Bu nedenle, tüm potansiyometre boyunca voltaj sabit kalır (roboRIO için bu 5 volt olacaktır), ancak sinyal pimi ile voltaj veya toprak pimi arasındaki voltaj, şaft döndükçe doğrusal olarak değişir.

Devre simetrik olduğu için tersine çevrilebilir - bu, kullanıcının dönüşün hangi ucunda ölçülen voltajın sıfır ve hangi ucta 5 volt olduğunu seçmesine izin verir. Sensörün yönünü tersine çevirmek için, basitçe geriye doğru kablolanabilir! Potansiyometrenizin yönünü bir multimetre ile kontrol ettiğinizden emin olun ve tellerinizi kontaklara lehimlemeden önce istenen yönde olduğundan emin olun.

¹ Bunun gerçekte çalışma şekli, genellikle şaftın, akımın aktığı sabit uzunluktaki dirençli bir "süpürücü" boyunca bir kontağın konumunu kontrol etmesini sağlamaktır - direnç, kontak ve süpürücü ucu arasındaki süpürücü uzunluğu ile orantılıdır. .

38.3.2 Dipnotlar

38.4 Dijital Girişler - Donanım

Not: This section covers digital input hardware. For a software guide to digital inputs, see *Digital Inputs - Software*.

Bir **digital sinyal**https://en.wikipedia.org/wiki/Digital_signal, birkaç farklı durumdan birinde olabilen bir sinyaldir. Çoğu durumda, sinyal bir teldeki voltajdır ve bir dijital sinyal için yalnızca iki durum vardır - yüksek veya düşük (ayrıca sırasıyla 1 ve 0 veya doğru ve yanlış olarak gösterilir).

RoboRIO'nun dahili dijital giriş-çıkış portları (veya "DIO") portları 5V üzerinde çalışır, bu nedenle "yüksek" 5V'luk bir sinyale ve "düşük" 0V¹² sinyaline karşılık gelir.

38.4.1 RoboRIO DIO bağlantı noktalarına bağlanma

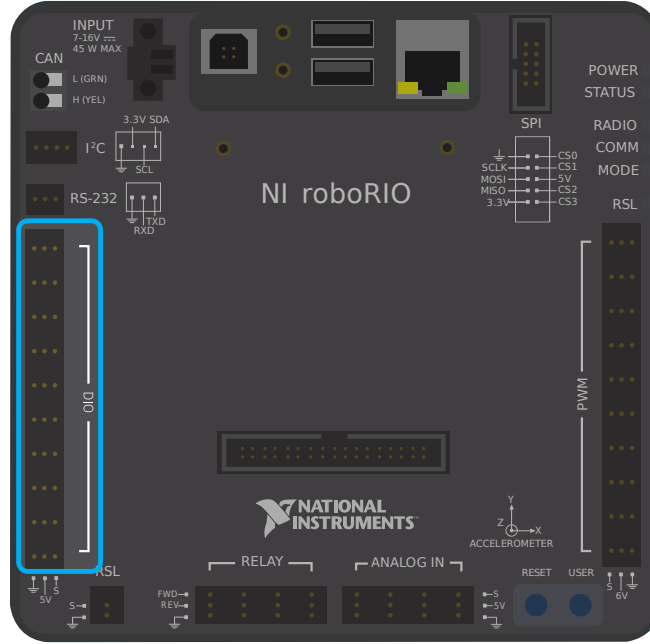
Not: Ek DIO bağlantı noktaları "MXP" genişletme bağlantı noktası aracılığıyla kullanılabilir. Bunları kullanmak için, MXP'ye bağlanan bir çeşit ara devre panosu gereklidir.

Uyarı: Her bir pine doğru kablonun bağlandığından emin olmak için, sensörü kablolmadan *önce* kullandığınız sensörün teknik özelliklerine daima başvurun. Bunun yapılmaması, cihazda hasara neden olabilir.

Uyarı: Hiçbir zaman güç pinini roboRIO üzerindeki herhangi bir bağlantı noktasındaki topraklama pinine doğrudan bağlamayın! Bu, roboRIO'daki koruma özelliklerini tetikleyecek ve beklenmeyen davranışlara neden olabilir.

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.



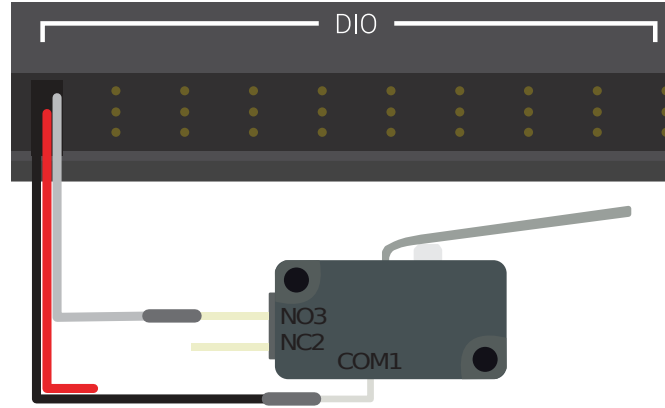
RoboRIO, yukarıdaki resimde görüldüğü gibi 10 dahili DIO bağlantı noktasına (0-9 numaralı) sahiptir. Her bağlantı noktasında üç pin bulunur - sinyal ("S"), güç ("V") ve toprak ("|**top-rak**|"). "Güç" ve "toprak" pinleri, DIO bağlantı noktalarına bağlanan dış sensörlere güç sağlamak için kullanılır - "güç" ve "toprak" pinleri [3] - "güç" arasında sabit bir 5V potansiyel farkı vardır pin "yüksek" duruma (5V) ve "toprak" dan "düşük" e (0V) karşılık gelir. Sinyal pini, sinyalin gerçekte ölçüldüğü pindir (veya bir çıkış olarak kullanıldığında, sinyali gönderen pindir).

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.

Basit bir anahtarı bir DIO bağlantı noktasına bağlama

Bir DIO portuna bağlanabilen en basit cihaz bir anahtardır (örneğin *limit anahtarı*). Bir DIO bağlantı noktasına bir anahtar doğru şekilde bağlandığında, bağlantı noktası devre açıkken "high" ve devre kapalıyken "low" okuyacaktır.

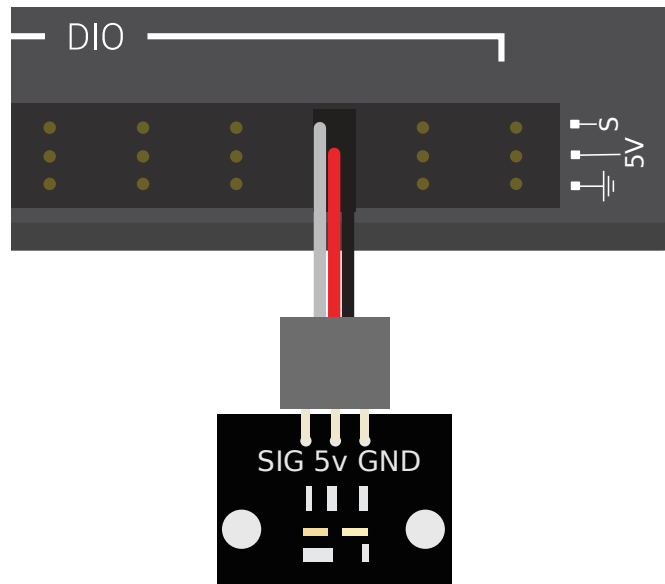
Basit bir anahtara güç verilmesine gerek yoktur ve bu nedenle sadece iki tele sahiptir. Anahtarlar, DIO portunun *sinyal* ve *toprak* pinleri arasına bağlanmalıdır. Anahtar devresi açık olduğunda, sinyal pini boşta ve kaldırma direnci "yüksek" değerini okumasını sağlar. Anahtar devresi kapatıldığında, doğrudan toprak rayına bağlanır ve böylece "düşük" okur.



Limit Switch or Micro Switch

Elektrikli bir sensörü bir DIO bağlantı noktasına bağlama

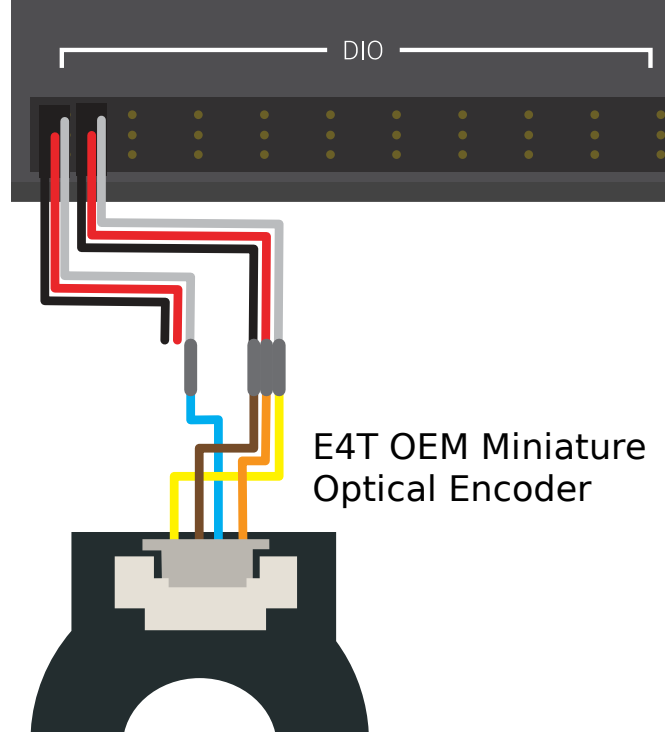
Çoğu dijital sensör (temassız yaklaşım anahtarlarının çoğu gibi) çalışmak için güce ihtiyaç duyar. Elektrikli bir sensörün genellikle üç kablosu vardır - sinyal, güç ve toprak. Bunlar, DIO portunun ilgili pinlerine bağlanmalıdır.



WCP Hall Effect Sensor

Birden çok DIO bağlantı noktası kullanan bir sensörü bağlama

Bazı sensörlerin (örneğin *quadrature encoders*) çalışması için birden çok DIO bağlantı noktasına bağlanması gerekebilir. Genel olarak, bu sensörler yalnızca tek bir güç ve tek bir toprak pini gerektirecektir - yalnızca ek bağlantı nokta(larının) sinyal pinlerine ihtiyaç duyulacaktır.



38.4.2 Dipnotlar

38.5 Yakınlık-Proximity Anahtarları - Donanım

Not: This section covers proximity switch hardware. For a guide to using proximity switches in software, see *Digital Inputs - Software*.

Bir robot üzerindeki en yaygın algılama görevlerinden biri, bir nesnenin (bir mekanizma, oyun parçası veya saha ögesi) robot üzerinde bilinen bir noktaya belirli bir mesafe içinde olup olmadığını algılamaktır. Bu tür bir algılama, “yakınlık anahtarı” ile gerçekleştirilir.

38.5.1 Yakınlık anahtarı çalışması

Yakınlık anahtarları switch-anahtardır - “açık” durum (devre boyunca *bağlantının olmadığı* *) ve “kapalı” bir (devrede **bağlantı olduğu*) nu belirtir. Bu nedenle, yakınlık anahtarları dijital bir sinyal üretir ve buna göre neredeyse her zaman roboRIO’ların:doc:digital giriş <digital-inputs-hardware> bağlantı noktalarına bağlanırlar.

Yakınlık anahtarları, anahtarın etkinleştirilmesinin devreyi kapattığı “normalde açık” veya anahtarın etkinleştirilmesinin devreyi açtığı “normalde kapalı” olabilir. Bazı anahtarlar * hem bir NO hem de aynı anahtara bağlı bir NC devresi sunar. Pratikte, bir NO ve bir NC anahtarı arasındaki etkili fark, bir kablolama arızası hemen hemen her zaman bir açık devre ile sonuçlanacağından, anahtara giden kablolanmanın başarısız olması durumunda sistemin davranışdır. NC anahtarları genellikle “daha güvenlidir”, çünkü bir kablo arızası sistemin anahtara basılmış gibi davranmasına neden olur - anahtarlar genellikle bir mekanizmanın kendisine zarar vermesini önlemek için kullanıldığından, bu durumda mekanizmanın zarar görme olasılığını azaltır. bir kablo arızası.

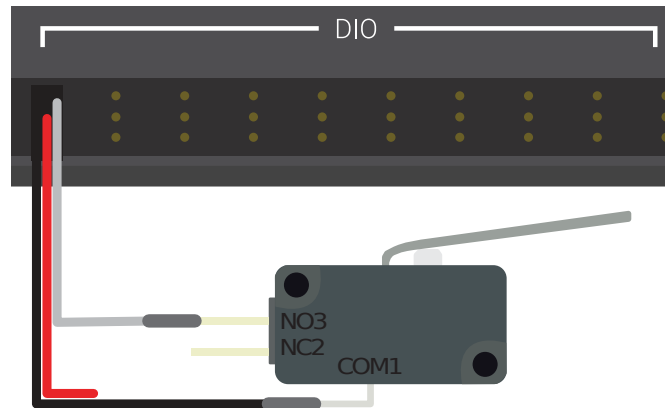
RoboRIO’daki dijital girişler, anahtar açıkken girişi yüksek (1 değeri) yapacak olan çekme dirençlerine sahiptir, ancak anahtar kapandığında, giriş artık toprağa bağlı olduğundan değer 0’a gider.

38.5.2 Yakınlık Anahtarı Türleri

There are several types of proximity switches that are commonly used in FRC®:

- *Mekanik Yakınlık Anahtarları (“limit anahtarları”)*
- *Manyetik Yakınlık Anahtarları*
- *Endüktif Yakınlık Anahtarları*
- *Fotoelektrik Yakınlık Anahtarları*
- *Uçuş Süresi Yakınlık Anahtarları*

Mekanik Yakınlık Anahtarları (“limit anahtarları”)



Limit Switch or Micro Switch

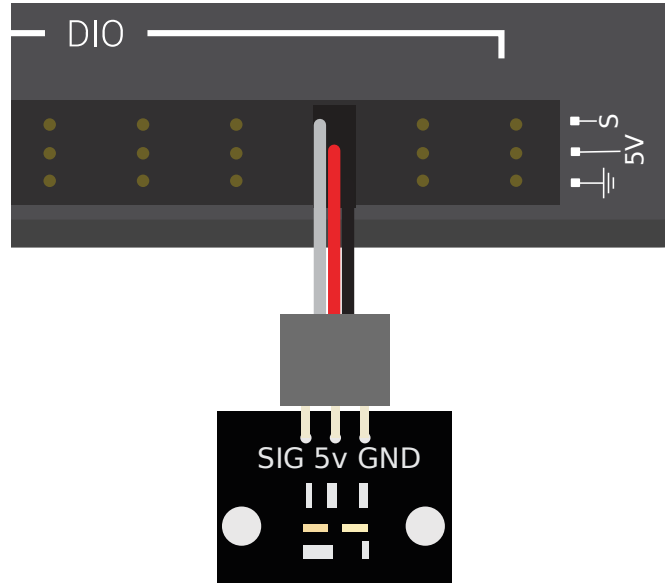
Mechanical proximity switches (more commonly known as “limit switches”) are probably the most commonly used proximity switch in FRC, due to their simplicity, ease-of-use, and low

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.

Limit anahtarlarının boyutu, anahtar kolunun geometrisi ve anahtarı etkinleştirmek için gereken “itış” miktarı değişir. Limit anahtarları oldukça ucuz olsa da, mekanik çalıştırma bazen temassız alternatiflere göre daha az güvenilirdir. Bununla birlikte, anahtar kolunu hareket ettirebilen herhangi bir fiziksel nesne tarafından tetiklenebildikleri için son derece fazla kullanım alanı bulurlar.

See this [article](#) for writing the software for Limit Switches.

Manyetik Yakınlık Anahtarları



WCP Hall Effect Sensor

Manyetik yakınlık anahtarları, sensörün belirli bir mesafesine bir mıknatıs yaklaştığında etkinleşir. Buna göre, “temassız” anahtarlardır - algılanan nesne ile temas gerektirmezler.

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](#).

Manyetik yakınlık anahtarları “tek kutuplu-unipolar”, “iki kutuplu-bipolar” veya “omnipolar” olabilir. Tek kutuplu bir anahtar, mıknatısın belirli bir kutbunun varlığına bağlı olarak etkinleştirilir ve devre dışı bırakılır (anahtara bağlı olarak mıknatısın N- kuzey veya S- güney kutpu). İki kutuplu bir anahtar, bir kutbun yakınlığından harekete geçer ve karşı kutbun yakınlığından devre dışı kalır. Omnipolar anahtarlar da her iki kutbun varlığında etkinleşir ve mıknatıs olmadığında devre dışı kalır.

Manyetik yakınlık anahtarları genellikle mekanik benzerlerinden daha güvenilir olsa da, kullanıcının algılanacak nesneye bir mıknatıs monte etmesini gerektirir - bu nedenle, çoğunlukla

mekanizma konumunu algılamak için kullanılırlar.

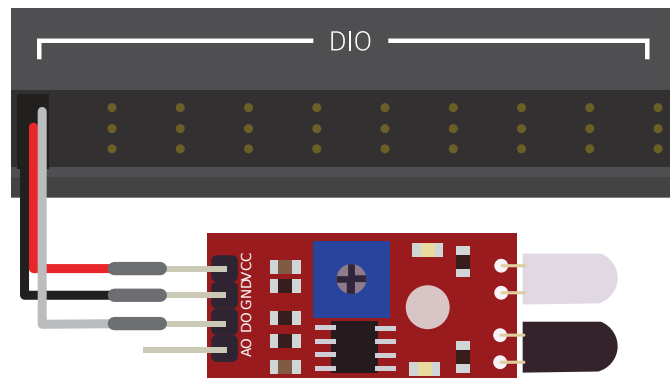
Endüktif Yakınlık Anahtarları



Endüktif yakınlık anahtarları, herhangi bir tür iletken sensörün belirli bir aralığına geldiğinde etkinleşir. Manyetik yakınlık anahtarları gibi, bunlar “temassız” anahtarlardır.

Endüktif yaklaşım anahtarları, manyetik yakınlık anahtarları ile aynı amaçların çoğu için uyumludur. Daha genel yapıları nedeniyle (sadece bir mıknatıs yerine herhangi bir iletkenin mevcudiyetinde aktif hale gelmeleri), uygulamanın doğasına bağlı olarak yardım veya engel teşkil edebilir.

Fotoelektrik-Photoelectric Yakınlık Anahtarları



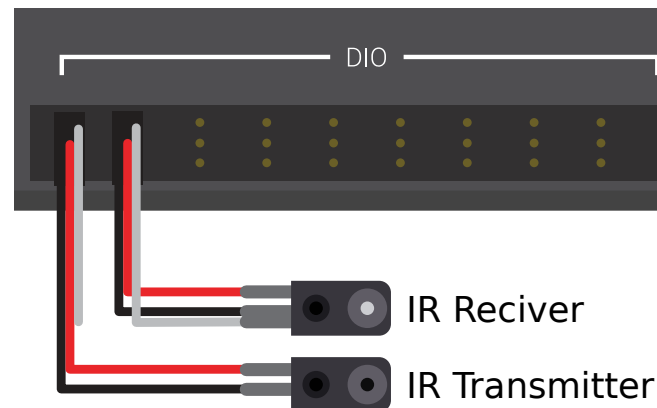
IR Digital Obstacle Avoidance Sensor

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 boun-

ces off of the sensor target) exceeds a given threshold. One such sensor is the [IR Obstacle Avoidance Module](#) pictured above.

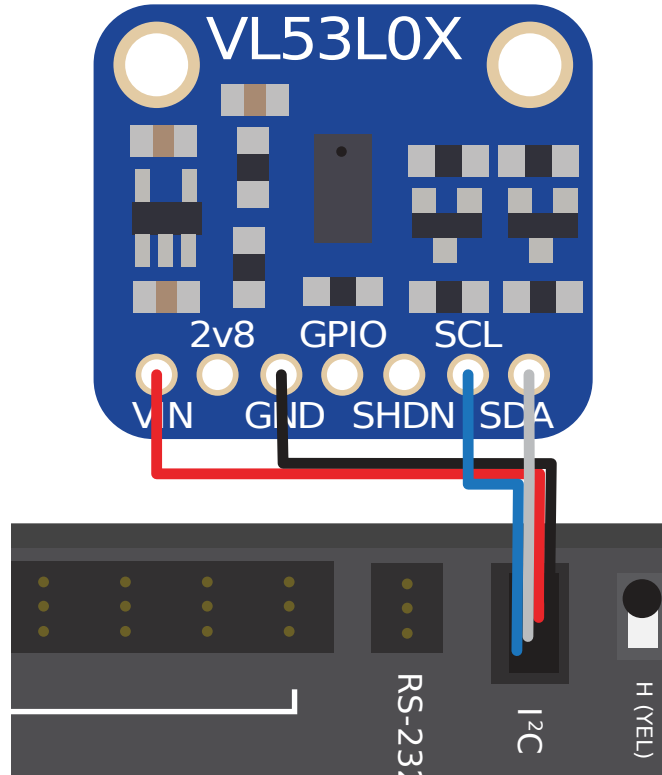
Since photoelectric proximity switches rely on measuring the amount of reflected light, they are often inconsistent in their triggering range between different materials - accordingly, most photoelectric sensors have an adjustable activation point (typically controlled by turning a screw somewhere on 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](#).



Uçuş Süresi -Time-of-flight Yakınlık Anahtarları

Time of Flight Distance Sensor



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 Kodlayıcılar-Donanım

Not: 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 Kodlayıcı Türleri

There are three main ways encoders connect physically that are typically used in FRC:

- *Shafted encoders*
- *On-shaft encoders*
- *Magnetic encoders*

Bu kodlayıcılar, söz konusu mekanizmaya nasıl monte edildiklerine göre değişir. Bu tür kodlayıcılara ek olarak, birçok FRC mekanizması tasarımlarına entegre edilmiş quadrature kodlayıcılarla birlikte gelir.

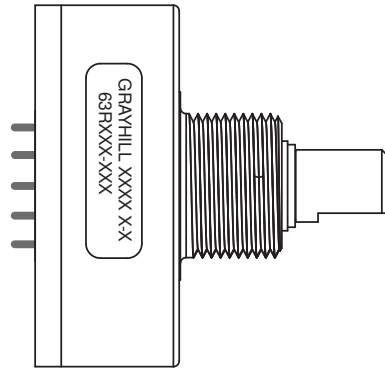
There are also three main ways the encoder data is communicated that are typically used in FRC:

- *Quadrature encoders*
- *Duty Cycle encoders*
- *Analog encoders*

Not: Some encoders may support more than one communication method

Shafted Encoders

Grayhill 63R Optical Encoder



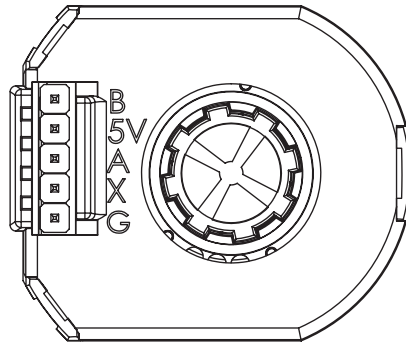
Shafted encoders have a sealed body with a shaft protruding out of it that must be coupled rotationally to a mechanism. This is often done with a helical beam coupling, or, more cheaply, with a length of flexible tubing (such as surgical tubing or pneumatic tubing), fastened with cable ties and/or adhesive at either end. Many commercial off-the-shelf FRC gearboxes have purpose-built mounting points for shafted encoders.

Examples of shafted encoders:

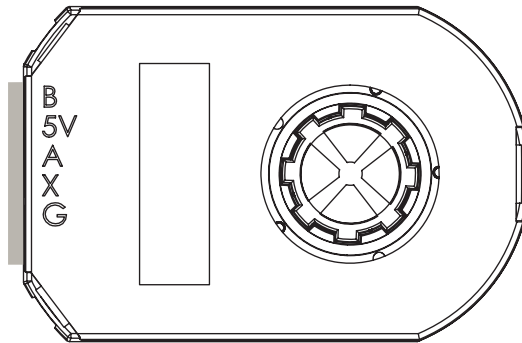
- [Grayhill 63r](#)
- [US Digital MA3](#)

On-shaft Encoders

AM10 Series Modular Incremental Encoder



AMT103



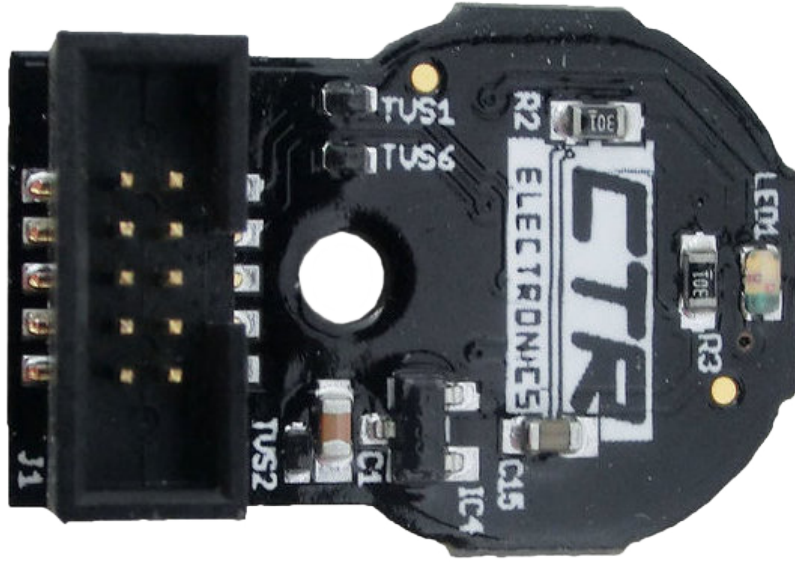
AMT102

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



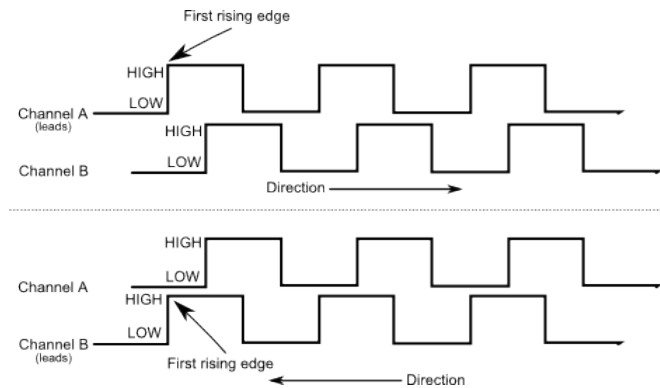
Magnetic encoders require no mechanical coupling to the shaft at all; rather, they track the orientation of a magnet fixed to the shaft. While the no-contact nature of magnetic encoders can be handy, they often require precise construction in order to ensure that the magnet is positioned correctly with respect to the encoder.

Examples of magnetic encoders:

- CTRE Mag Encoder
- Thrifty Absolute Magnetic Encoder
- Team 221 Lamprey2

Quadrature Encoders

“Quadrature” terimi, hareketin ölçüldüğü/kodlandığı yöntemi ifade eder. Bir quadrature kodlayıcı, aşağıdaki resimde görüldüğü gibi, birbirinden 90 derece faz farkı olan iki kare dalga darbesi üretir:



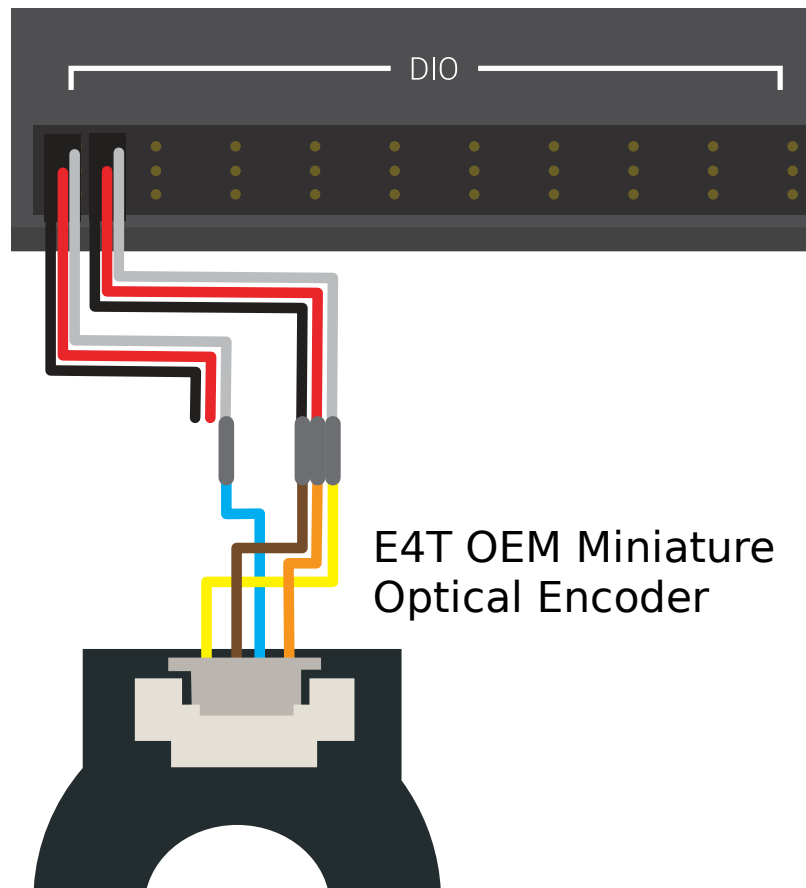
Bu nedenle, her iki kanalda, periyod başına toplam dört “kenar” vardır (dolayısıyla “dörtlülük”). İki faz farklı darbenin kullanılması, hareket yönünün, hangi darbenin diğerini “yönlendirdiği” net bir şekilde belirlenmesine izin verir.

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.

Quadrature Encoder Resolution

Uyarı: “CPR”-cycles per revolution ve “PPR” kısaltmaları *her ikisi de*, çeşitli kaynaklarda her kenarı ve devir başına döngüyü belirtmek için kullanılır, bu nedenle kısaltma tek başına ikisinden hangisinin geçerli olduğunu belirlemeye yetmez. Şüphe duyduğunuzda, kodlayıcınızın teknik kılavuzuna bakın.

Kodlayıcılar dönüşü dijital darbelerle ölçtüğü için, ölçümün doğruluğu, belirli bir dönme hareketi miktarı başına darbe sayısı ile sınırlıdır. Bu, kodlayıcının “çözünürlüğü” olarak bilinir ve geleneksel olarak iki farklı yoldan biriyle ölçülür: devir başına kenar veya devir başına döngü.

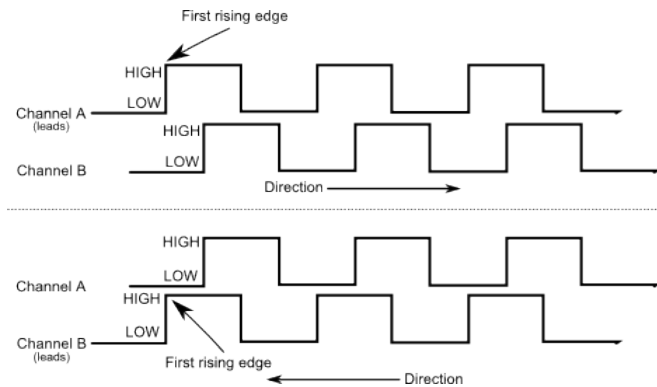
Devir başına kenarlar, kodlayıcı milinin her iki kanalında da yüksekten düşüğe veya düşükten yükseğe geçişlerin toplam sayısını ifade eder. Tam bir nokta *dört* kenar içerir.

Devir başına döngü, enkoder milinin dönüşü başına her iki kanalın toplam *tam dönem* sayısını ifade eder. Tam bir periyod *bir* döngüdür.

Bu nedenle, devir başına kenarlarda belirtilen bir çözünürlük, devir başına döngülerde belirtilen aynı çözünürlüğün dört katı bir değere sahiptir.

Genel olarak, kodlayıcınızın dönüş başına kenardaki çözünürlüğü, konumlandırmadaki kabul edilebilir en küçük hatanızdan biraz daha hassas olmalıdır. Dolayısıyla, mekanizmayı artı veya eksi bir derece bilmek istiyorsanız, dönüş başına 360 kenardan biraz daha yüksek çözünürlüğe sahip bir kodlayıcıya sahip olmalısınız.

Duty Cycle Encoders



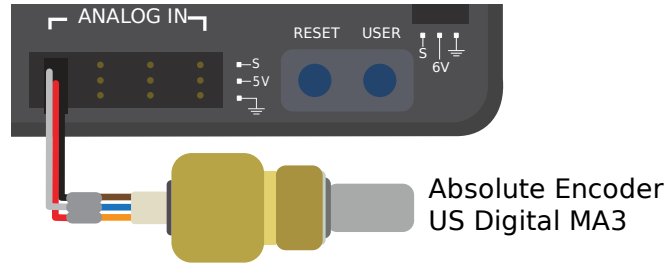
Duty cycle encoders connect to a single digital input on the roboRIO. They output a pulse where the length of a pulse is proportional to the absolute position of the encoder.

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 Gyroscopes - Donanım

Not: This section covers gyro hardware. For a software guide to gyros, see [Gyroscopes - Software](#).

Jiroskoplar (veya kısaca “jiroskoplar”) dönme oranını ölçen cihazlardır. Bunlar özellikle robot sürüşünü stabilize etmek için veya toplam açısız yer değiştirme ölçümünü elde etmek için hız ölçümlerini entegre ederek (toplayarak) yönü veya eğimi ölçmek için kullanışlıdır.

:ref: IMUlar <docs/hardware/sensors/accelerometers-hardware:IMUs (Inertial Measurement Units)>` (Atalet Ölçüm Birimleri) olarak bilinen birkaç popüler FRC® cihazı, 3 eksenli jiroskopları, ivmeölçerleri ve diğer konum sensörlerini tek bir cihazda birleştirir. Bazı popüler örnekler şunlardır:

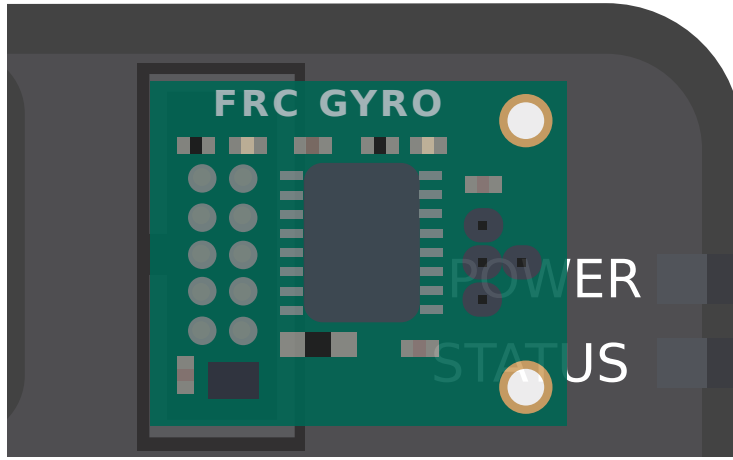
- Analog Devices ADIS16448 veADIS 16470 IMUs
- CTRE Pigeon IMU
- Kauai Labs NavX

38.7.1 Gyro Türleri

FRC’de yaygın olarak kullanılan iki tür Gyro vardır: tek eksenli jiroskoplar, üç eksenli jiroskoplar ve genellikle 3 eksenli jiroskop içeren IMU’lar.

Tek eksenli Gyrolar

Analog Devices 1-axis SPI Gyro



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.

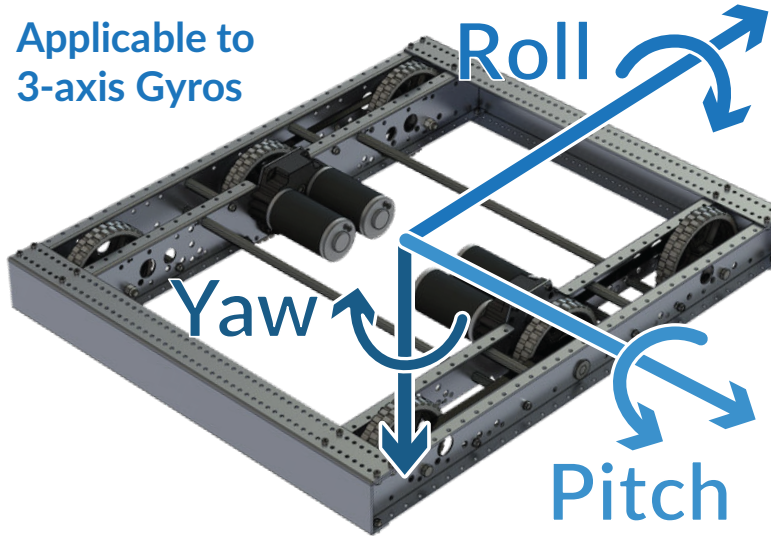
Son yıllarda FIRST Choice’de olan [Analog Devices ADXRS450 FRC Gyro Board](#), yaygın olarak kullanılan bir tek eksenli jiroskoptur.

Üç eksenli Gyrolar



Üç eksenli jiroskoplar, üç uzaysal eksenin (tipik olarak x, y ve z olarak etiketlenir) etrafındaki dönüş oranını ölçer. Bu eksen etrafındaki harekete, pitch-eğim, yaw-sapma ve roll-yuvarlanma denir.

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.



Not: 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.

Çevresel üç eksenli jiroskoplar, basitçe üç analog voltaj çıkışı sağlayabilir (ve böylece *analog giriş portları* bağlanabilir veya (daha yaygın olarak) roboRIO'nun *seri veri yollarından* biriyle iletişim kurabilirler

38.8 Ultrasonik - Donanım

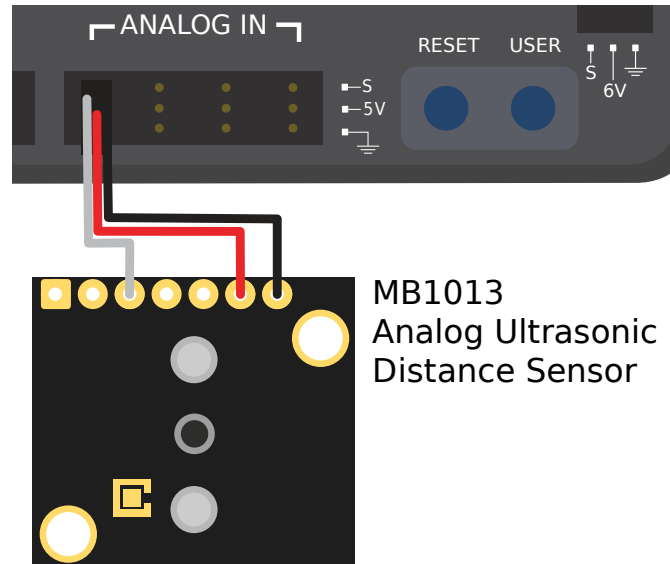
Not: This section covers ultrasonic sensor hardware. For a software guide to ultrasonics, see [Ultrasonics - Software](#).

Ultrasonik telemetreler, FRC®’de kullanılan en yaygın telemetrelerden bazılarıdır. Ucuz, kullanımı kolay ve oldukça güvenilirler. Ultrasonik telemetreler, yüksek frekanslı bir ses darbesi yayarak ve ardından yankının hedeften sıçradıktan sonra sensöre ulaşmasının ne kadar sürdüğünü ölçerek çalışır. Ölçülen zamandan ve havadaki ses hızından hedefe olan mesafeyi hesaplamak mümkündür.

38.8.1 Ultrasonik türleri

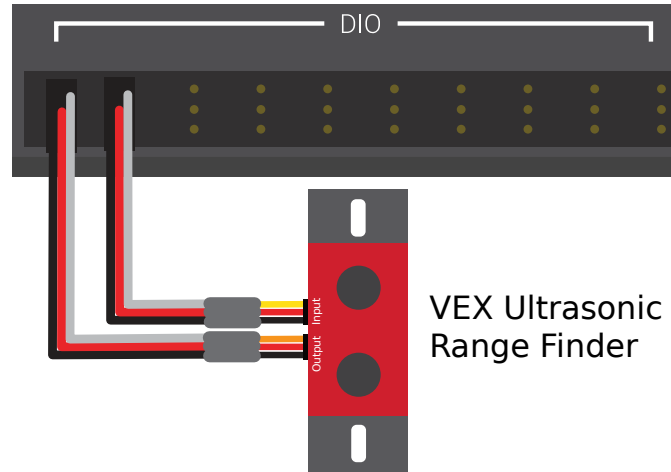
Tüm ultrasonik telemetreler yukarıda özetlenen “ping yanıtı” prensibine göre çalışırken, roboRIO ile iletişim kurma şekillerinde farklılık gösterebilir.

Analog ultrasonik



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](#).

Ping yanıtı ultrasonikleri



Bahsedildiği gibi, tüm ultrasonikler işlevsel olarak ping yanıtı cihazlar olsa da, bir “ping yanıtı” ultrasonik şunlara bağlanmak üzere yapılandırılmıştır *digital giriş ve digital çıkış*. Dijital çıkış ping göndermek için kullanılır, giriş ise yanıtı okumak için.

Seri ultrasonik



Bazı daha karmaşık ultrasonik sensörler, RIO ile şu şunlardan biri üzerinden iletişim kurabilir: RS-232 gibi *serial buses*

38.8.2 Uyarılar

Ultrasonik sensörlerin kullanımı genellikle oldukça kolaydır, ancak birkaç dikkat edilecek nokta vardır. Ultrasonik, darbe ile yankısı arasındaki süreyi ölçerek çalıştığından, genellikle yalnızca kendi aralıklarındaki *en yakın* hedefe olan mesafeyi ölçer. Bu nedenle, iş için doğru sensörü seçmek son derece önemlidir. Ultrasonik sensörler için dokümantasyon, genellikle ultrasoniğin bir hedefi tespit edeceği “pencerenin” şeklini gösteren bir “ışın modeli” resmini içerecektir - sensörünüzü seçerken buna çok dikkat edin.

Ultrasonik sensörler ayrıca diğer ultrasonik sensörlerden kaynaklanan parazitlere karşı hassastır. Bunu en aza indirmek için roboRIO, ping yanıtı ultrasoniklerini “round-robin” tarzında çalıştırabilir - ancak rekabet halinde, diğer robotlara monte edilmiş sensörlerden kaynaklanan parazitlerin oluşmamasını sağlamanın kesin bir yolu yoktur.

Son olarak, ultrasonik ses dalgalarını emen veya onları tuhaf şekillerde yönlendiren nesneleri tespit edemeyebilir. Bu nedenle, sert, düz nesneleri tespit etmek için en iyi şekilde çalışırlar.

38.9 İvmeölçerler - Donanım

İvmeölçerler, ivmeyi ölçmek için kullanılan yaygın sensörlerdir.

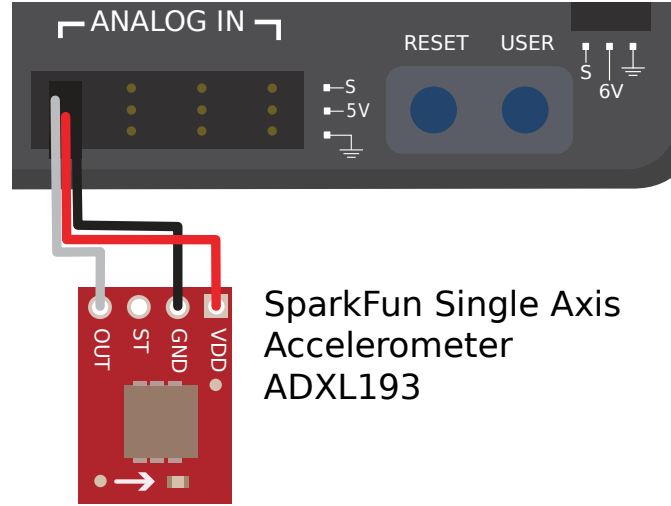
Prencip olarak, hassas ivme ölçümleri çift-entegre olabilir ve konumu izlemek için kullanılabilir (bir denge çarkının dönüş hızı ölçümünün istikameti belirlemek için nasıl entegre edilebileceğine benzer şekilde) - ancak pratikte yasal FRC’de bulunan ivmeölçerlerin fiyat aralığı bu kullanım için neredeyse doğru değildir. Bununla birlikte, ivmeölçerler, FRC’deki bir dizi görev için hala yararlıdır.

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 İvme ölçer türleri

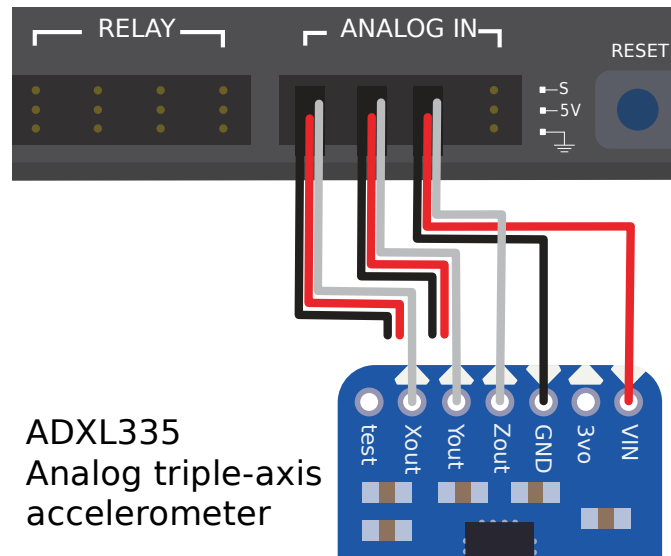
FRC’de yaygın olarak kullanılan üç tür ivmeölçer vardır: tek eksenli ivmeölçerler, çok eksenli ivmeölçerler ve IMU’lar.

Tek eksenli-single axis ivmeölçerler



Adlarına göre, tek eksenli ivmeölçerler, tek bir eksen boyunca ivmeyi ölçer. Bu eksen genellikle fiziksel cihaz üzerinde belirtilir ve istenen eksenin ölçülebilmesi için cihazın uygun yönde monte edilmesi oldukça önemlidir. Tek eksenli ivmeölçerler genellikle ölçülen ivmeye karşılık gelen bir analog voltaj üretir ve bu nedenle roboRIO'nun *analog giriş* bağlantı noktalarına bağlanır.

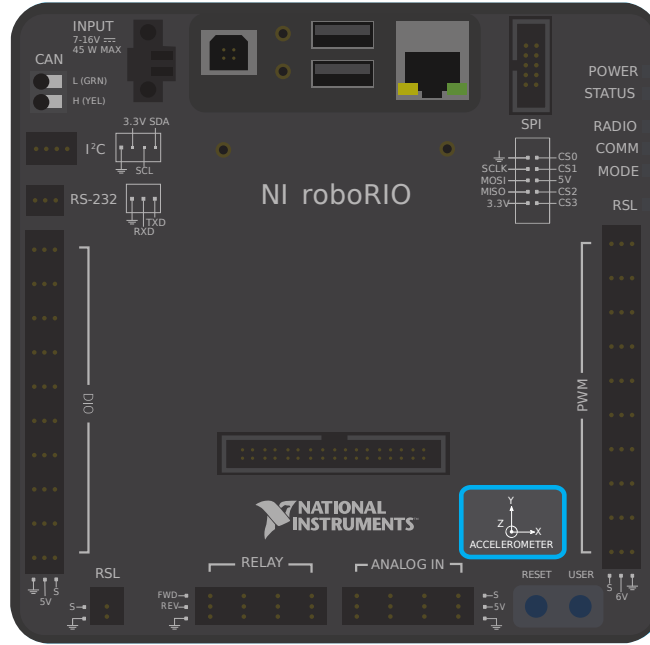
Çok eksenli-multi axis ivmeölçerler



Çok eksenli ivmeölçerler, birden çok uzaysal eksen boyunca ivmeyi ölçer. RoboRIO'nun yerleşik ivmeölçeri, üç eksenli bir ivmeölçerdir.

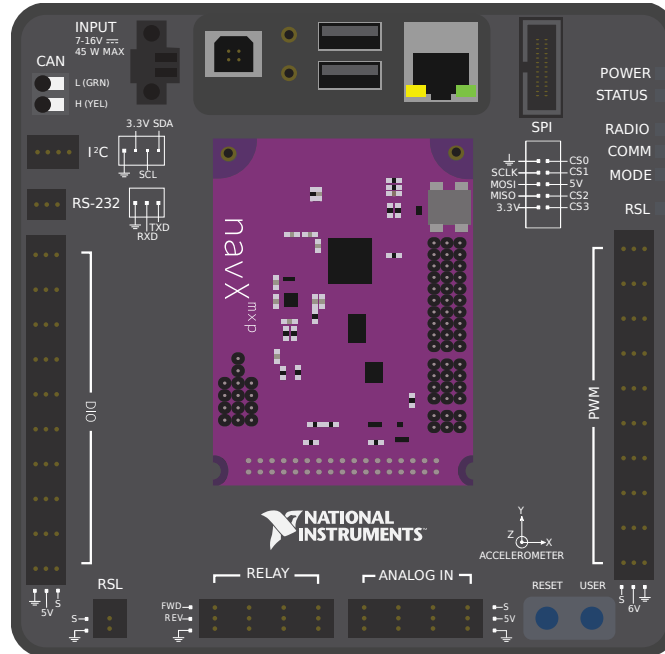
Peripheral multi-axis accelerometers may simply output multiple analog voltages (and thus connect to the *analog input ports*, or (more commonly) they may communicate with one of the roboRIO's *serial buses*.

roboRIO yerleşik ivme ölçer



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'lar (Atalet Ölçüm Cihazları)



Birkaç popüler FRC cihazı (“atalet ölçüm cihazları” veya “IMU’lar” olarak bilinir) hem ivme-ölçeri hem de jiroskopu birleştirir. Popüler FRC örneği şunları içerir:

- Analog Aygıtlar ADIS16448 ve ADIS 16470 IMU'lar
- CTRE Pigeon IMU
- Kauai Labs NavX

38.10 LIDAR - Donanım

LIDAR (ışık algılama ve menzil) sensörleri, FRC®'de kullanımı artan telemetri sistemleridir.

LIDAR sensörleri aşağıdakilere *ultrasonics* oldukça benzer şekilde çalışır ancak ses yerine ışığı kullanır. Bir lazer atımı olur ve sensör, darbenin geri dönmesine kadar geçen süreyi ölçer.

38.10.1 LIDAR türleri

Mevcut FRC'de yaygın olarak kullanılan iki tür LIDAR sensörü vardır: 1 boyutlu LIDAR ve 2 boyutlu LIDAR.

1 Boyutlu - 1 Dimentional LIDAR



1 boyutlu (1D) bir LIDAR sensörü, ultrasonik sensör gibi çalışır - en yakın nesneye olan mesafeyi, doğrusal bir çizgi boyunca ölçer. 1D LIDAR sensörleri, daha dar “ışın profillerine” sahip olduklarından ve parazitte daha az duyarlı olduklarından, genellikle ultrasonik sensörlerden daha güvenilirlerdir. Yukarıdaki resimde *Garmin LIDAR-Lite Optik Mesafe Sensörü* görülmektedir.

1D LIDAR sensörleri genellikle ölçülen mesafeye orantılı bir analog voltaj çıkarır ve böylece roboRIO'ların *analog input* bağlantı noktalarına veya *roboRIO seri iletişim portlarından* birine bağlanır.

2 Boyutlu LIDAR



2 boyutlu (2D) bir LIDAR sensörü, bir düzlemdeki tüm yönlerdeki mesafeyi ölçer. Genel olarak bu işlem (kabaca), sabit bir hızda dönen döner tablaya 1D LIDAR sensörü yerleştirilerek gerçekleştirilir.

Doğası gereği, 2D LIDAR sensörlerinin büyük miktarda veriyi roboRIO'ya geri göndermesi gerektiğinden, neredeyse her zaman roboRIO'nun *seri veri yollarından* birine direkt bağlanırlar .

38.10.2 Uyarılar

LIDAR sensörlerinin birkaç yaygın dezavantajı vardır:

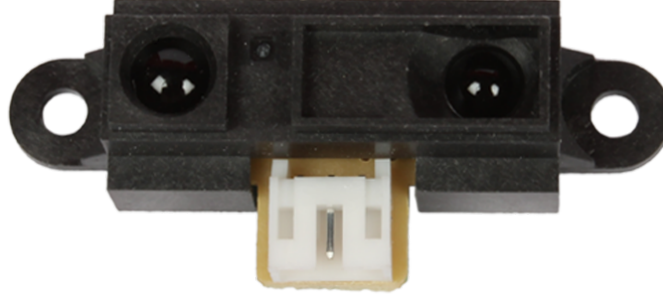
Ultrasonik olanlar gibi, LIDAR da yayılan darbenin sensöre geri yansımaları mantığına dayanır. Bu nedenle, LIDAR kritik olarak lazerin dalga boyundaki malzemenin yansıtıcılığına bağlıdır. FRC saha duvarları, kızılötesi dalga boyunda şeffaf olma eğiliminde olan polikarbonattan yapılmıştır (bu, genellikle FRC kullanımı için yasaldır). Bu nedenle, LIDAR saha duvarlarını tespit etmekte zorlanma eğilimindedir.

2D LIDAR sensörleri (FRC kullanımı için yasal fiyat aralığında) oldukça gürültülü olma eğilimindedir ve ölçülen verilerinin işlenmesi ("nokta bulutu" olarak bilinir) çok sayıda karmaşık yazılımdan faydalanır. Ek olarak, özellikle FRC için yapılmış çok az sayıda 2D LIDAR sensörü vardır, bu nedenle yazılım desteği az olma eğilimindedir.

2D LIDAR sensörleri çalışmak için bir döner tablaya bağlı olduğundan, veri güncelleme hızları, döner tablanın dönme hızıyla sınırlıdır. FRC için yasal fiyat aralığındaki sensörler için, bu genellikle değerlerini özellikle hızlı bir şekilde güncellemedikleri anlamına gelir; bu, robot (veya hedefler) hareket ederken bir sınırlama olabilir.

Ek olarak, 2D LIDAR sensörleri *açısız* çözünürlükle sınırlı olduğundan nokta bulutunun *uzamsal* çözünürlüğü, hedefler daha uzaktayken daha kötüdür.

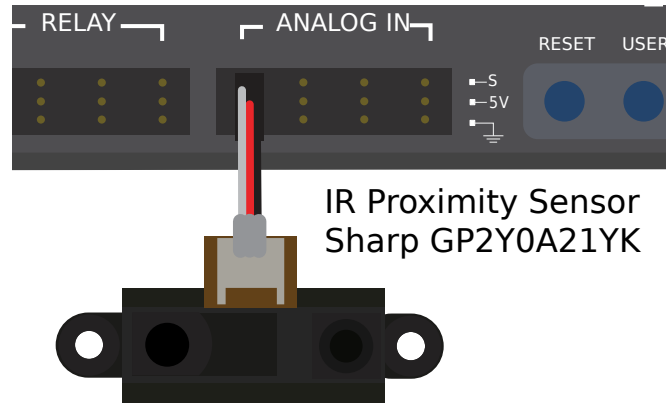
38.11 Triangulating Mesafe Ölçerler



Triangulating telemetreler (genellikle kızılötesi dalga boyu bandında çalıştıkları için “IR telemetreler” olarak adlandırılırlar), FRC®’de kullanılan bir başka yaygın telemetre türüdür. Yukarıda gösterilen sensör, *Genel bir Sharp marka sensördür*

Bunun aksine: doc: “LIDAR <lidar> ”, triangulating telemetreleri bir darbenin emisyonu ile bir yansımanın alınması arasındaki süreyi ölçmez. Bunun yerine, çoğu IR telemetre, hafif bir açıyla sabit bir ışın yayarak ve yansıyan ışının konumunu ölçerek çalışır. Yansıtılan ışının yayıcıya temas noktası ne kadar yakınsa, nesne sensöre o kadar yakın olur.

38.11.1 IR telemetreleri kullanma



IR Mesafe Ölçerler genellikle hedefe olan mesafeyle orantılı bir analog voltaj çıkarır ve bu nedenle RIO’daki *analog giriş* bağlantı noktalarına bağlanır.

38.11.2 Uyarılar

IR telemetreler, 1D LIDAR sensörlerine benzer dezavantajlardan muzdariptir - yayılan lazerin dalga boyundaki hedefin yansıtıcılığına çok duyarlıdır.

Ek olarak, kızılötesi telemetreler kısa mesafelerde ölçüm yaparken LIDAR sensörlerinden daha iyi çözünürlük sunma eğiliminde olduğundan, genellikle hedefin yönelimindeki farklılıklara karşı daha hassastır, *özellikle* hedef yüksek derecede yansıtıcıysa (ayna gibi).

38.12 Seri Veri Yolları

RoboRIO, *digital* ve *analog* girişlerine ek olarak, çevre aygıtlarıyla çeşitli seri iletişim yöntemleri de sunar.

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.

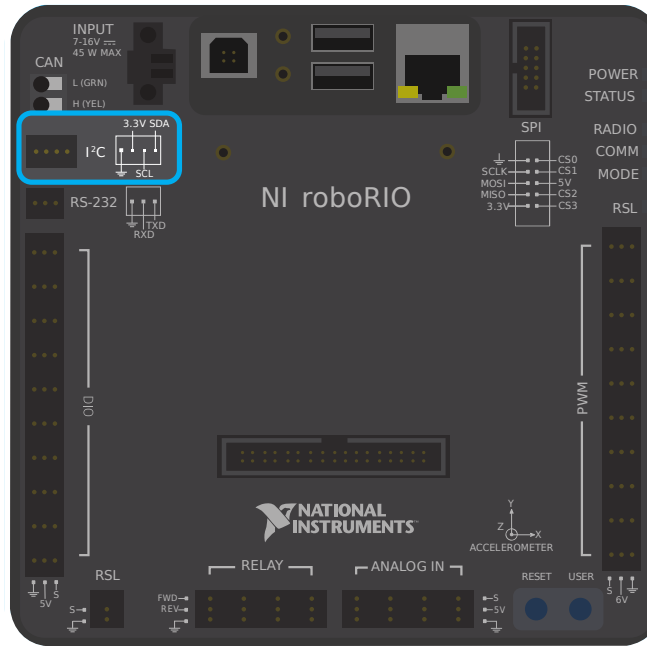
38.12.1 Desteklenen seri veri yolu türleri

RoboRIO, birçok temel seri iletişim türünü destekler:

- *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

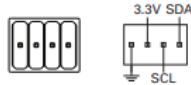


Table 5. I²C Port Signal Descriptions

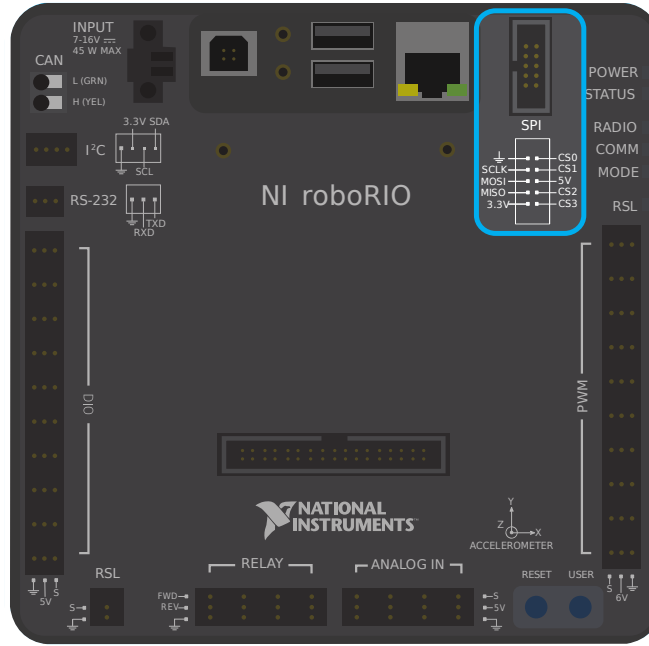
Signal Name	Direction	Description
GND	—	Reference for digital lines and +3.3 V power output.
3.3V	Output	+3.3 V power output.
SCL	Input or Output	I ² C lines with 3.3 V output, 3.3 V/5 V-compatible input. Refer to the I²C Lines section for more information.
SDA	Input or Output	

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.

Uyarı: Be sure to familiarize yourself on the following known issue before using the on-board 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

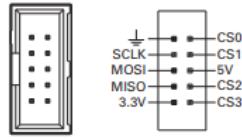


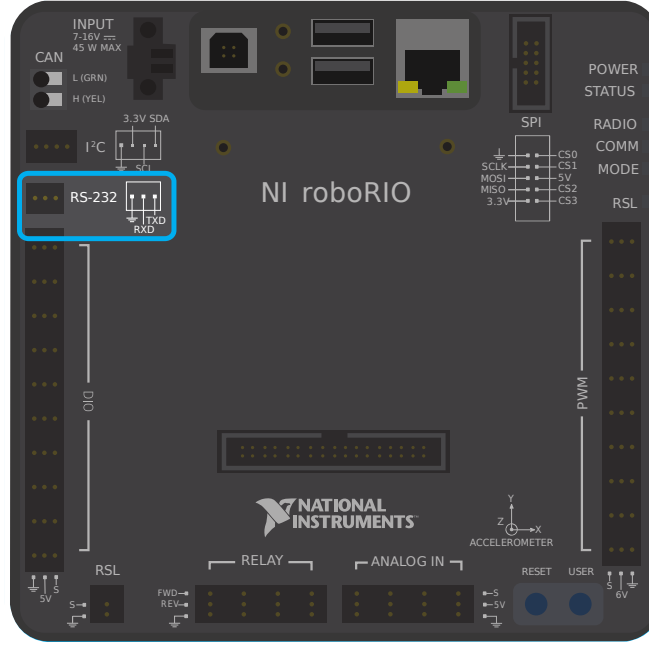
Table 12. SPI Port Signal Descriptions

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 veriyolu ayrıca [MXP expansion port](#) aracılığıyla da kullanılabilir. MXP bağlantı noktası, bağımsız saat ve giriş / çıkış hatları ve ek bir CS sağlar.

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.

Çevresel cihazlarla RS-232 üzerinden iletişim kurmak için, her bir pinin cihaz üzerindeki ilgili pimine bağlanması gerekir.

RS-232 veri yolu aynı zamanda MXP expansion port`_ aracılığıyla da kullanılabilir.

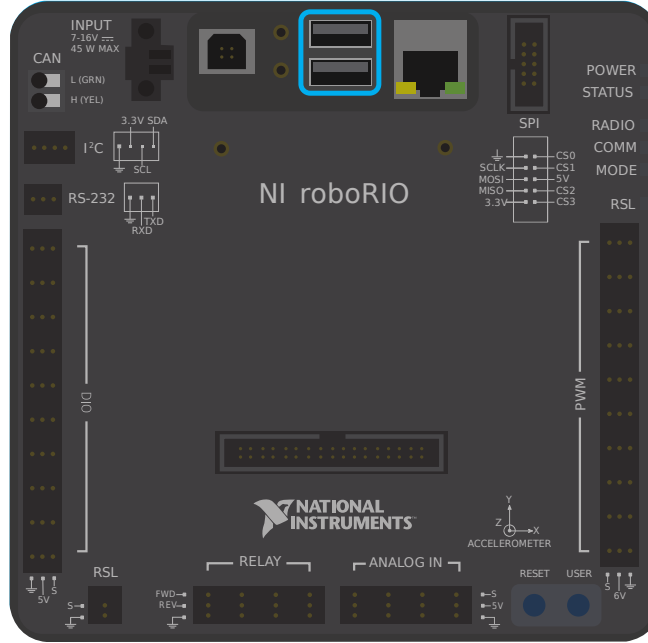
The roboRIO RS-232 serial port uses RS-232 signaling levels (+/- 15v). The MXP serial port uses CMOS signaling levels (+/- 3.3v).

Not: By default, the onboard RS-232 port is utilized by the roboRIO's serial console. In order to use it for an external device, the serial console must be disabled using the [Imaging Tool](#) or [roboRIO Web Dashboard - Web Kontrol Paneli](#).

38.12.5 USB Client

RoboRIO'daki USB bağlantı noktalarından biri USB-B veya USB bağlantı noktasıdır. Bu, standart bir USB kablosuyla Driver Station bilgisayarı gibi cihazlara bağlanabilir.

38.12.6 USB Host



RoboRIO'daki USB bağlantı noktalarından ikisi bir USB-A veya USB host bağlantı noktasıdır. Bunlar kameralar veya sensörler gibi cihazlara standart bir USB kablosuyla bağlanabilir.

38.12.7 MXP Expansion Port - MXP Genişleme Portu

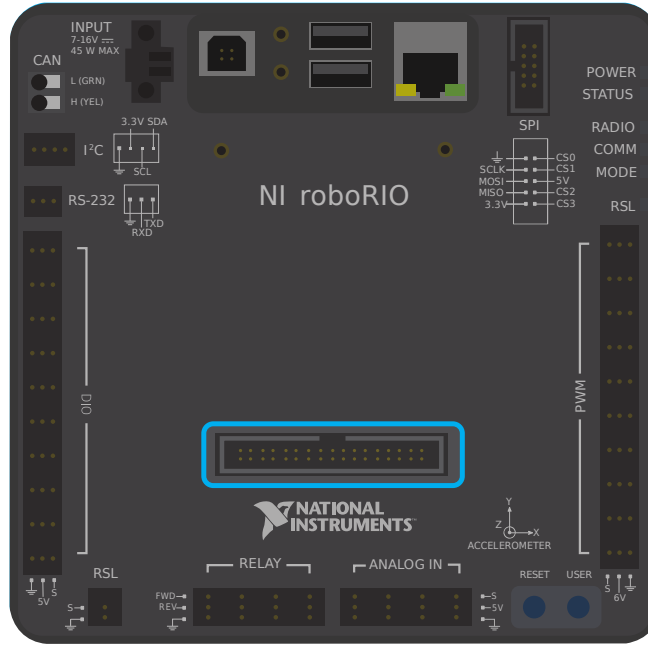
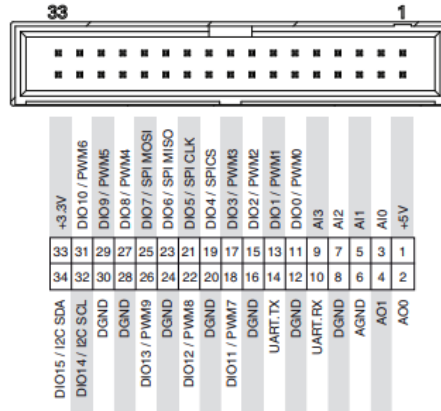


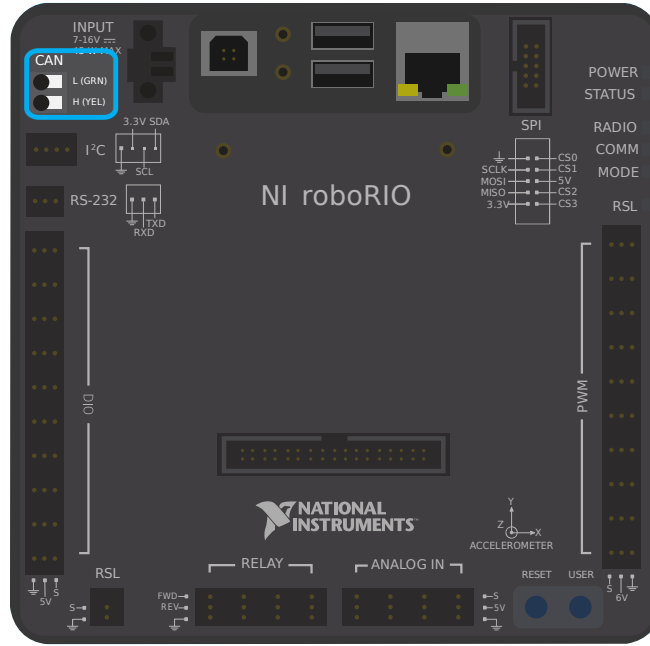
Figure 4. MXP Pinout



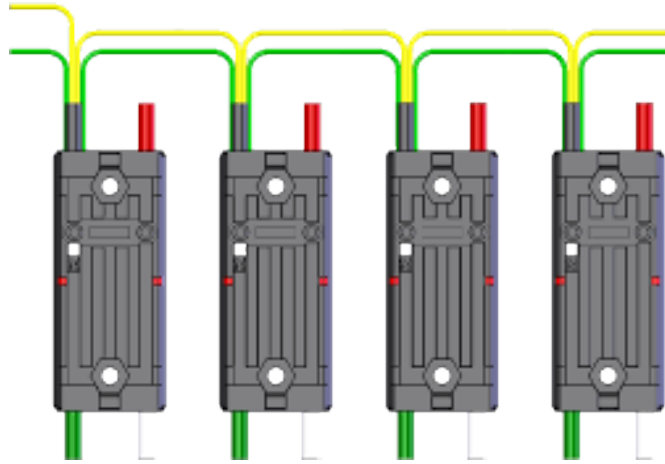
Several of the serial buses are also available for use through the roboRIO's [MXP](#) Expansion Port. This port allows users to make use of many additional *digital* and *analog* inputs, as well as the various serial buses.

Birçok çevresel aygıt, kullanıcı tarafından herhangi bir kablolama gerektirmeden, rahatlıkla doğrudan MXP bağlantı noktasına bağlanır.

38.12.8 CAN Bus Veri Yolu



Ek olarak roboRIO, CAN veriyolu üzerinden çevresel cihazlarla iletişimi destekler. Bununla birlikte, FRC CAN protokolü oldukça kendine özgü olduğundan, nispeten az sayıda çevresel sensör onu desteklemektedir (ancak motor kontrolörleri için yoğun bir şekilde kullanılmaktadır). CAN veri yolu protokolünü kullanmanın avantajlarından biri, cihazların aşağıda gösterildiği gibi zincirleme bağlanabilmesidir. Zincirdeki herhangi bir aygıttan güç kesilirse, veri sinyalleri yine de zincirdeki tüm aygıtlara erişebilir.



Bir çok sensör öncelikle CAN veriyolunu kullanır. Bazı örnekler şunları içerir:

- Playwithfusion.com'dan CAN Tabanlı Uçuş Süresi / Mesafe Sensörü
- 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 \(PDH\)](#)

CAN veriyoluna baęlı cihazları kullanma hakkında daha fazla bilgi řu makalede bulunabilir *CAN cihazların kullanımı*.

Romi'ye Giriş

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.

Tüyo: A course that teaches programming using the Romi Robot is available via Thinkscape. Information on this course is available [here](#)



39.1 Romi Donanım ve Montajı

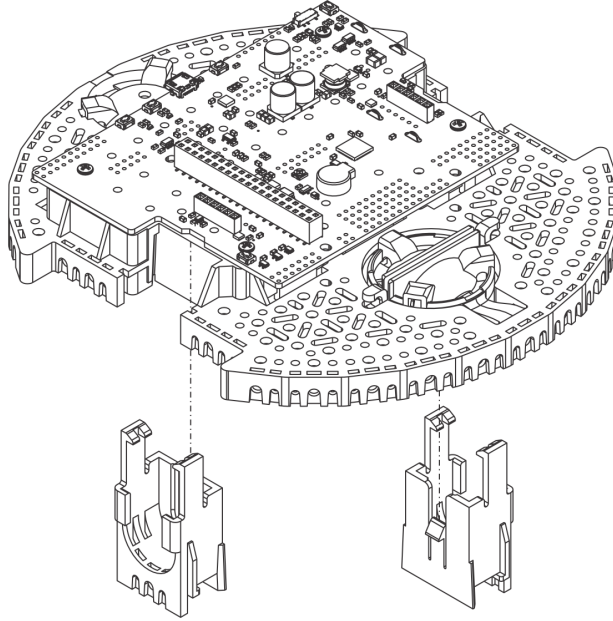
Romi'ye başlamak için gerekli donanıma sahip olmanız gerekecek.

1. Pololu'dan [Romi Kiti](#) - Sipariş ücretsiz gönderim için uygundur
2. *Raspberry Pi* <<https://www.amazon.com/gp/product/B07BFH96M3/>> __ - 3B + veya 4
3. [8GB \(or larger\) Micro SD card](#)
4. `Micro SD card reader <<https://www.amazon.com/gp/product/B0779V61XB/>>” __ - eğer zaten sizde yoksa
5. [6 AA batteries](#) - Şarj edilebilir en iyisidir (şarj cihazını unutmayın)

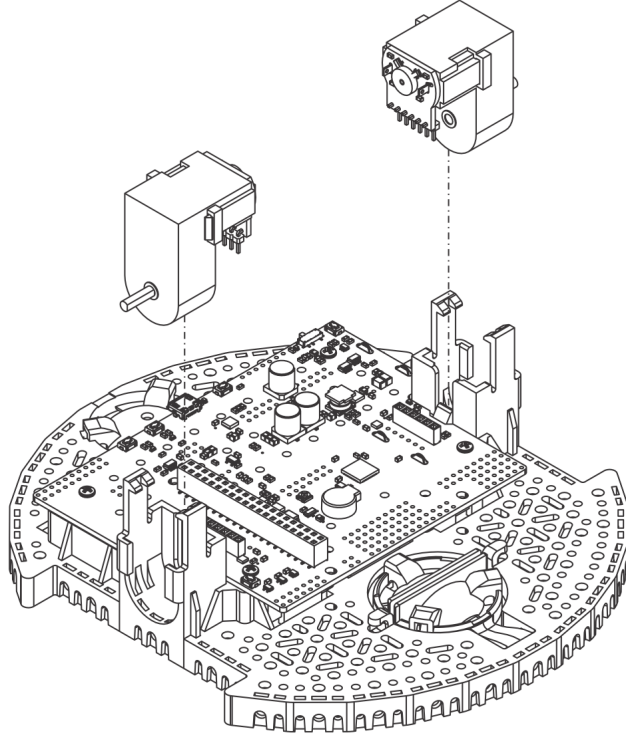
39.1.1 Montaj

FIRST için Romi Robot Kiti önceden lehimlenmiş olarak gelir ve kullanılmadan önce yalnızca bir araya getirilmesi gerekir. Tüm malzemeleri topladıktan sonra montaja başlayabilirsiniz:

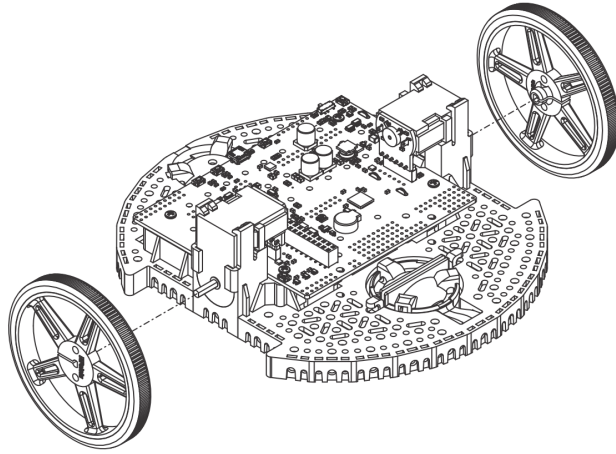
1. Motor klipslerini gösterildiği gibi kasayla hizalayın ve klipslerin altı kasanın alt kısmıyla aynı hizaya gelene kadar kasanın içine sıkıca bastırın (birkaç tıklama duyabilirsiniz).



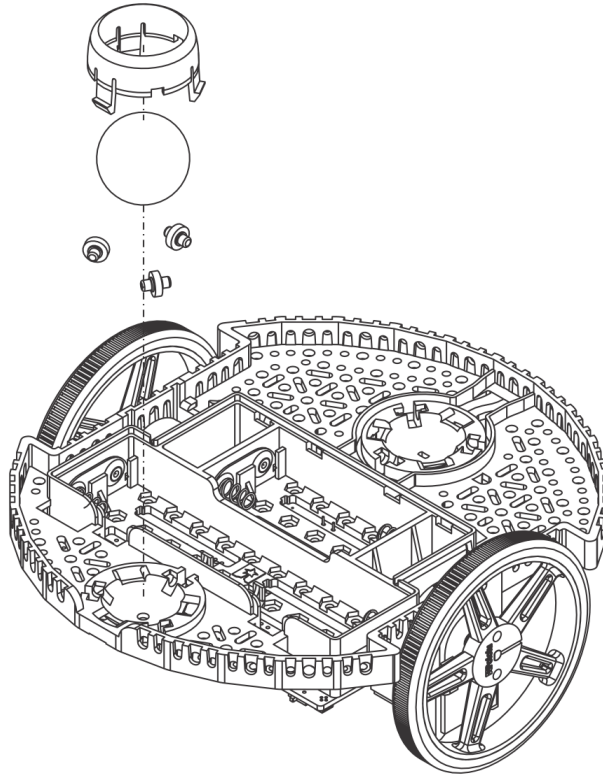
2. Mini Plastik Redüktörlü Motorları, yerine oturana kadar motor klipslerine itin. Motorun klips serbest bırakılmasını engellediğini unutmayın, bu nedenle daha sonra bir motor braketini çıkarmanız gerekirse, önce motoru çıkarmanız gerekir. Kitle birlikte gelen Mini Plastik Redüktörlü motorlar, konum geri bildirimi için dört evreli kodlayıcıları etkinleştirmek için genişletilmiş motor millerine sahiptir.



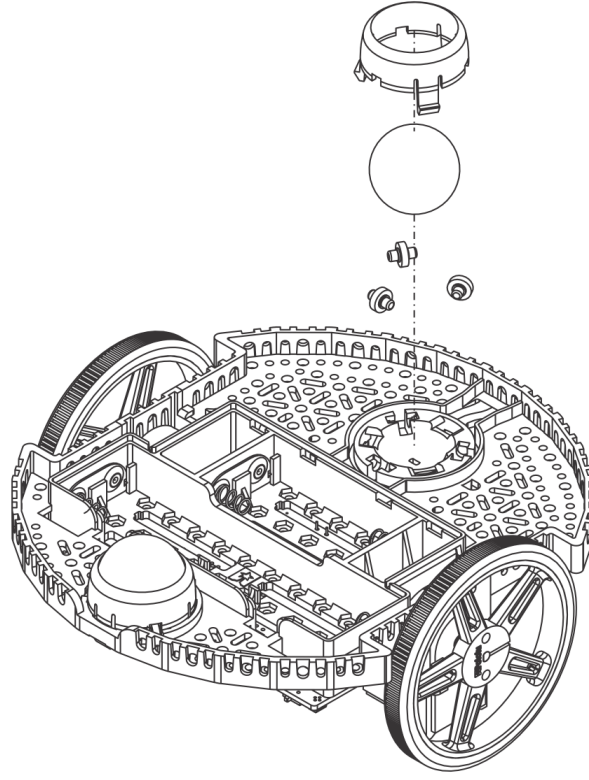
3. Motor shaftı tekerleğin dış yüzü ile aynı hizaya gelene kadar tekerlekleri motorların çıkış millerine bastırın. Bunu yapmanın bir yolu, tekerleği düz bir yüzeye yerleştirmek ve şasiyi bununla hizalamaktır, böylece motorun D-şaftının düz kısmı tekerlekle doğru şekilde hizalanır. Ardından, motor şaftını yüzeye temas edene kadar tekerleğin içine doğru bastırarak şasiyi indirin.



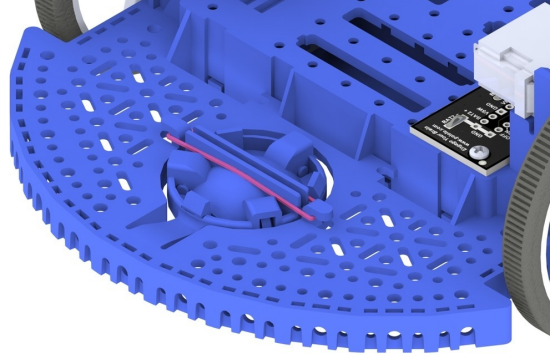
4. Şasiyi ters çevirin ve arka top tekerlek için üç silindiri şasideki oyuklara yerleştirin. 1 plastik topu üç silindirin üstüne yerleştirin. Ardından, bilyeli tekerlek tutma klipsini topun üzerinden şasinin içine itin, böylece üç bacak ilgili deliklere oturur.



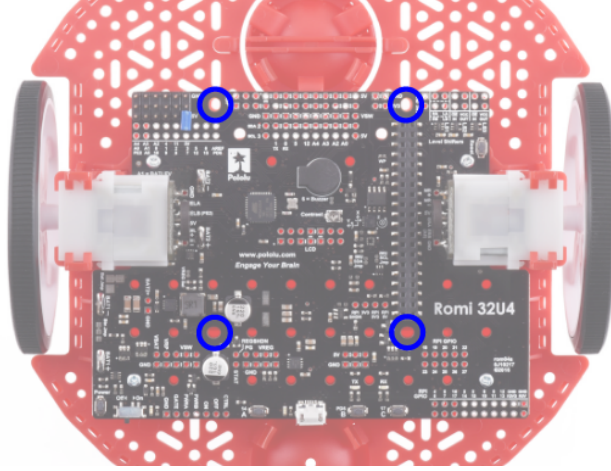
5. Ön top tekeri için tekrarlayın, böylece robotun önünde ve arkasında bir tekerlek olur.



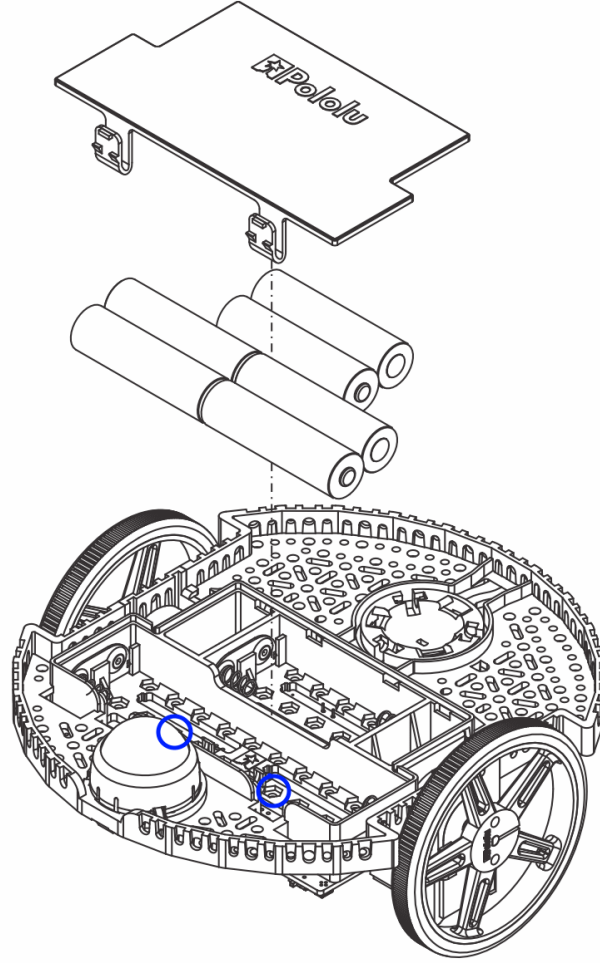
6. İsteğe bağlı: Ön bilyalı tekerlek, süspansiyon sistemi olarak işlev gören esnek bir kolla desteklenir. Daha sert hale getirmek istiyorsanız, şasinin üst tarafındaki top tekerleğinin her iki yanında bulunan iki kancanın etrafına bir lastik bant sarabilirsiniz.



7. Raspberry Pi kartını desteklemek için ayraçları takın. Romi kartının yan tarafındaki deliklere resimde gösterildiği gibi “Romi 32U4” etiketine en yakın iki çıkıntı (dişli tarafı aşağıda) monte edilir. Bu aralayıcıların somunları pil bölmesinin içindedir. Diğer iki çıkıntı, kartın diğer tarafındaki deliklere girer. Bunları takmak için, çıkıntıları vidalarken somunu tutmak için bir kargaburun pense ihtiyacınız olacaktır. Aşağıdaki resimde daire içine alınmış delikler, zıtlıkların nereye gitmesi gerektiğini göstermektedir.



8. Kasa, dört veya altı adet AA pil ile çalışır (şarj edilebilir AA NiMH pilleri kullanmanızı öneririz). Pillerin doğru yönü Romi kاسasındaki pil şeklindeki deliklerin yanı sıra kasanın kendisindeki + ve - göstergeleriyle gösterilir.



9. Raspberry Pi kartını, Pi üzerindeki 2x20 pinli konektörü Romi'deki 2x20 pin soketiyle dikkatlice hizalayarak baş aşağı takın. Pimlerin hiçbirini bükmemeye dikkat ederek eşit bir basınçla itin. Takıldıktan sonra, Raspberry Pi kartını önceki bir adımda takılan tır-

naklarla sabitlemek için verilen vidaları kullanın.

Not: Vidalardan ikisi, pil bölmesinin içindeki altıgen bir deliğe bir somun yerleştirilmesini gerektirecektir. Konumlar yukarıdaki resimde mavi dairelerle gösterilmiştir.



Romi kasanızın montajı artık tamamlandı!

39.2 Romi'nize İmaj atmak

Romi'nin 2 mikroişlemci kartı vardır:

1. A **Raspberry Pi** that handles high-level communication with the robot program running on the desktop and
2. A **Romi 32U4 Control Board** that handles low-level motor and sensor operation.

Both boards need to have firmware installed so that the robot operates properly.

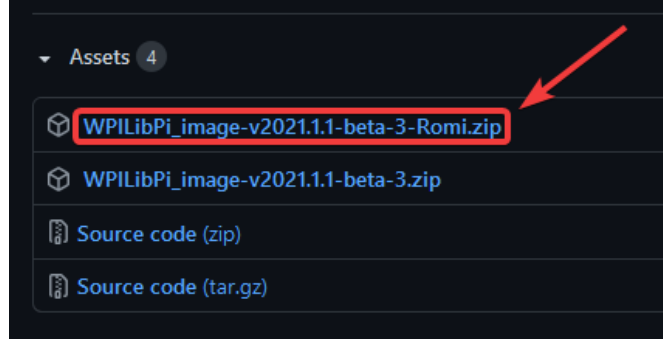
39.2.1 Raspberry Pi

İndirme

Raspberry Pi ürün yazılımı WPILibPi (eski adıyla FRCVision) tabanlıdır ve Raspberry Pi mikro SD kartına indirilip yazılmalıdır. Mevcut resim dosyalarını görmek için açıklamanın altındaki Assets-Varlıklar seçeneğine tıklayın:

[Romi WPILibPi](#)

WPILibPi'nin standart sürümünü değil, Romi sürümünü indirdiğinizden emin olun. Romi versiyonunun sonuna ``-Romi`` eklenmiştir. Örnek için aşağıdaki resme bakın.



İmaj Atma

Görüntünün kurulum prosedürü burada açıklanmaktadır: [WPILibPi Kurulumu](#).

Kablosuz Ağ Kurulumu

Raspberry Pi'nizi Romi ile kullanıma hazır hale getirmek için aşağıdaki adımları uygulayın:

1. Romi 32U4 kartındaki güç anahtarını açık konuma kaydırarak Romi'yi açın. Yeni bir görüntü ile ilk kez başlatıldığında, dosya sistemini yeniden boyutlandırırken ve yeniden başlatılırken önyüklemesi yaklaşık 2-3 dakika sürecektir. Sonraki zamanlarda, bir dakikadan daha kısa sürede açılacaktır.
2. Bilgisayarınızı kullanarak, WPA2 parolası `` WPILib2021! `` ile SSID WPILibPi-
<number> (burada <number> Raspberry Pi seri numarasına dayalıdır) kullanarak Romi WiFi ağına bağlanın.

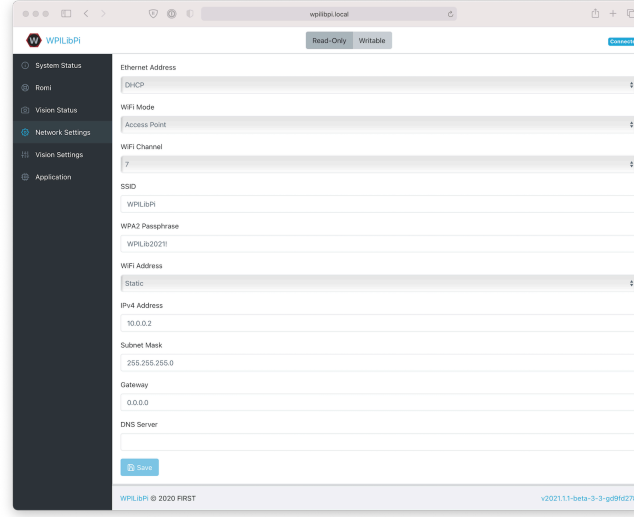
Not: Raspberry Pi'yi birden fazla WPILibPi çalıştıran Raspberry Pi's'in bulunduğu bir ortamda çalıştırıyorsanız, belirli bir Raspberry Pi'nin SSID'si de kulaklık bağlantı noktası aracılığıyla sesli olarak duyurulur. Varsayılan SSID, SD kartı (okuyucu aracılığıyla) bir bilgisayara takıp boot-önyükleme bölümünü açarak okunabilen /boot/default-ssid.txt dosyasına da yazılır.

3. Bir web tarayıcısı açın ve <http://10.0.0.2/> veya <http://wpilibpi.local/> adresinden Raspberry Pi panosuna bağlanın.

Not: Görüntü varsayılan olarak salt okunur olarak açılır, bu nedenle değişiklik yapmak için Writable düğmesine tıklamanız gerekir. Değişiklikleri yaptıktan sonra, bellek bozulmasını önlemek için Read-Only düğmesine tıklayın.

4. Kontrol paneli web sayfasının üst kısmındaki *Writable* ögesini seçin.
5. WPA2 Passphrase alanına yeni bir parola ayarlayarak Romi'niz için varsayılan parolayı değiştirin.
6. Değişiklikleri kaydetmek için sayfanın altındaki *Save* düğmesine basın.
7. Change the network SSID to a unique name if you plan on operating your Romi on a wireless network with other Romis.
8. Belirlediğiniz yeni şifre ile Romi'nin WiFi ağına yeniden bağlanın.

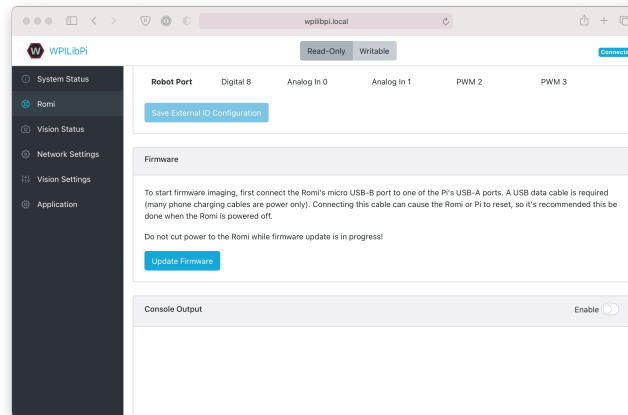
Tüm değişiklikler tamamlandığında Gösterge Panosunu Read-only- Salt Okunur olarak ayarladığınızdan emin olun.



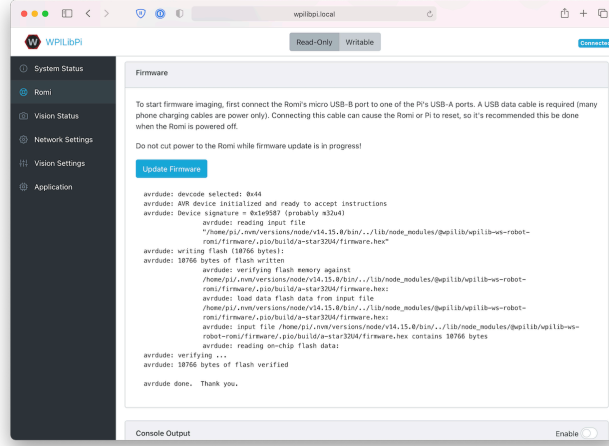
39.2.2 32U4 Kontrol Kartı

Raspberry Pi artık donanım yazılımı görüntüsünü 32U4 Kontrol Paneline yazmak için kullanılabilir.

1. Romi'yi kapatın
2. Raspberry Pi'nin USB bağlantı noktalarından birinden 32U4 Kontrol Panosu üzerindeki mikro USB bağlantı noktasına bir USB A - mikro-B kablosu bağlayın.
3. Romi'yi açın ve Wifi ağına bağlanın ve önceki adımlarda olduğu gibi web panosuna bağlanın.
4. Romi konfigürasyon sayfasında, *Update Firmware* düğmesine basın.



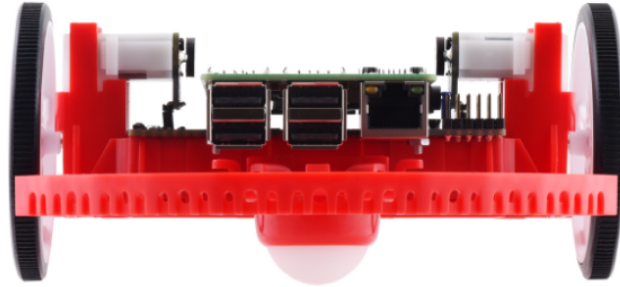
Aygıt yazılımı dağıtım işleminin günlüğünü gösteren bir konsol görünecektir. Firmware 32U4 Kontrol Panosuna dağıtıldıktan sonra, ``avrdude done. Thank you.`` görünecek.



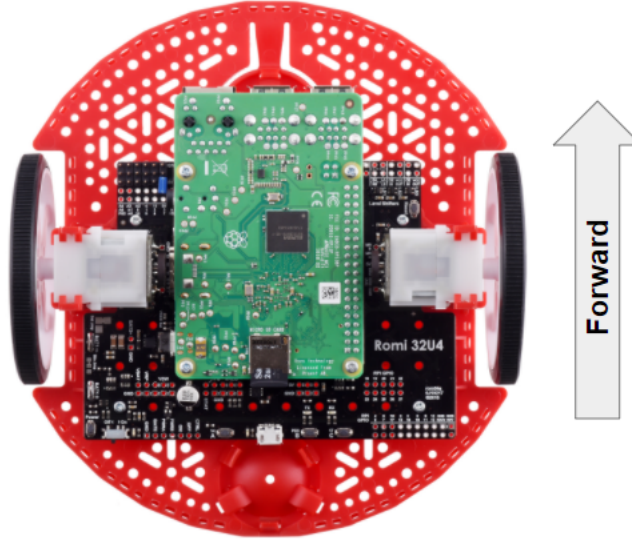
39.3 Romi Robotunu tanımak

39.3.1 Yön Konvansiyonları

Romi'nin önü, Raspberry Pi USB bağlantı noktalarının, GPIO pinlerinin ve suspended caster tekerleğinin bulunduğu yerdir.



Tüm Romi belgelerinde, ileriye doğru ilerlemeye yönelik referanslar yukarıdaki “front” tanımını kullanır.



39.3.2 Hardware, Sensors, and GPIO

Romi robotu, aşağıdaki yerleşik donanıma / çevre birimlerine sahiptir:

- Enkoderli 2x dişli motor
- 1x Atalet Ölçüm Birimi (IMU)
- 3x LED (yeşil, sarı, kırmızı)
- 3x buton (A, B ve C olarak işaretlenmiştir)
- 5x configurable GPIO channels (EXT)
- Buzzer

Not: The Buzzer is currently not supported by WPILib.

Motors, Wheels, and Encoders

Romi robotunda kullanılan motorlar 120: 1 dişli redüksiyonuna ve 4,5V'de 150 RPM yüksüz çıkış hızına sahiptir. Serbest akım 0,13 amperdir ve durma akımı 1,25 amperdir. Durma torku 25 oz-inçtir (0.1765 N-m), ancak yerleşik emniyet kavraması daha düşük torklarda kaymaya başlayabilir.

Tekerlekler 70 mm (2,75 ") çapa sahiptir ve 141 mm (5,55") iz genişliğine sahiptirler.

Enkoderlar doğrudan motor çıkış shaftına bağlanır ve 12 Devir Başına Sayıma (CPR) sahiptir. Sağlanan dişli oranıyla, bu, tekerlek dönüşü başına 1440 sayım yapar.

The motor *PWM* channels are listed in the table below.

Channel	Romi Donanım Bileşeni
PWM 0	Left Motor
PWM 1	Right Motor

Not: 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 Donanım Bileşeni
DIO 4	Sol Kodlayıcı Dörtlü Kanal A
DIO 5	Sol Enkoder Dörtlü Kanal B
DIO 6	Sağ Kodlayıcı Dörtlü Kanal A
DIO 7	Sağ Kodlayıcı Dörtlü Kanal B

Not: Varsayılan olarak, kodlayıcılar Romi ileri gittiğinde sayar.

Atalet ölçü birimi

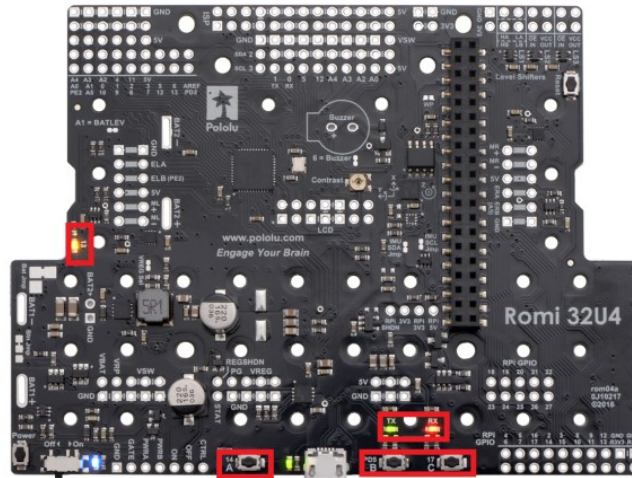
Romi robotu, 3 eksenli cayo ve 3 eksenli ivmeölçer içeren bir STMicroelectronics LSM6DS33 Atalet Ölçüm Birimi (IMU) içerir.

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 robotu web kullanıcı arayüzü ayrıca, robot koduyla kullanılmadan önce jiroskopu kalibre etmek ve sıfır ofsetlerini ölçmek için bir araç sağlar.

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.

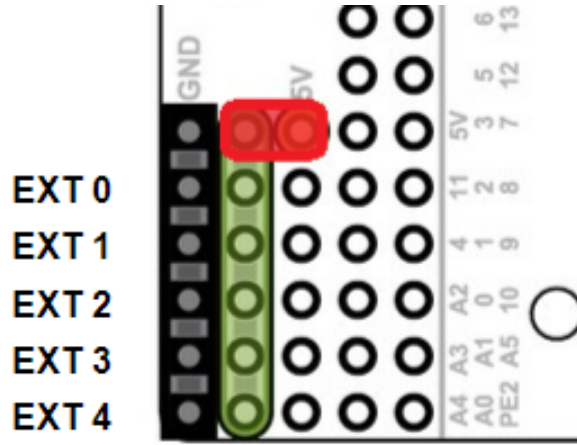


DIO Kanalı	Romi Donanım Bileşeni
DIO 0	Button A (yalnızca giriş)
DIO 1	Button B (giriş), Yeşil LED (çıkış)
DIO 2	Button C (giriş), Kırmızı LED (çıkış)
DIO 3	Sarı LED (yalnızca çıkış)

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 kanalları, Topraklama, Güç ve Sinyal bağlantılarıyla (Toprak bağlantısı kartın kenarına en yakın ve sinyal kartın içine en yakın olacak şekilde) 3 pimli, servo tarzı bir arabirim aracılığıyla gösterilir.

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.

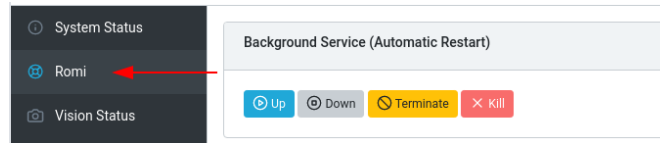
- PWM Motor Controllers (i.e. Spark)
- Encoder
- AnalogInput
- DigitalInput
- DigitalOutput
- Servo
- BuiltInAccelerometer

The following classes are provided by the [Romi Vendordep](#).

- RomiGyro
- RomiMotor
- OnboardIO

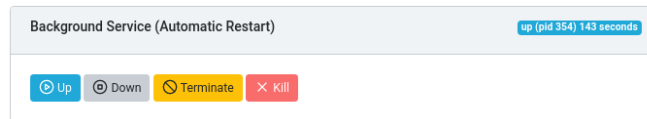
39.5 Romi Web Kullanıcı Arayüzü

Romi Web Kullanıcı Arayüzü, WPILibPi Raspberry Pi görüntüsünün bir parçası olarak yüklenir. Ana WPILibPi Web Kullanıcı Arayüzünün gezinme çubuğundaki Romi sekmesine tıklayarak erişilebilir.



Bu bölümün geri kalanı, Romi Web Kullanıcı Arayüzünün çeşitli bölümlerinden geçecek ve ilgili işlevselliği açıklayacaktır.

39.5.1 Arka Plan Hizmet Durumu



Romi Web Kullanıcı Arayüzünün bu bölümü, halihazırda çalışan Romi Web Hizmeti (WPILib'in Romi ile konuşmasına izin veren şey) hakkında bilgi sağlar. Kullanıcı arabirimi, hizmeti yukarı / aşağı getirmek için kontroller sağlar ve web hizmetinin mevcut çalışma süresini gösterir.

Not: Kullanıcıların bu bölümdeki işlevleri sık kullanmaları gerekmeyecektir, ancak sorun giderme için yararlı olabilir.

39.5.2 Romi Durumu

Romi Status	
	Value
Romi Service Version	0.0.12
Firmware Compatible	Yes
Battery Voltage	7.65

Bu bölüm, hizmet sürümü, pil voltajı ve Romi 32U4 kartında şu anda yüklü olan ürün yazılımının web hizmetinin mevcut sürümü ile uyumlu olup olmadığı dahil olmak üzere Romi hakkında bilgi sağlar.

Not: Donanım yazılımı uyumlu değilse, şu bölüme bakın :doc:`Imaging your Romi </docs/romi-robot/imaging-romi>`nizi görüntüleme “

39.5.3 Web Servis Güncellemesi

Web Service Update

To perform an offline update of the Romi webservice, obtain an appropriate version from the GitHub release page, and upload the .tgz file here.

Upload Romi Webservice Package

No file chosen

Not: Raspberry Pi'nin bu bölümün çalışması için ****Writable- Yazılabilir**** modunda olması gerekir.

Romi WPILibPi görüntüsü, Romi web hizmetinin en son (yayınlanma anında) sürümüyle birlikte gelir. Romi web hizmetinin daha yeni sürümlerine yükseltmeyi desteklemek için bu bölüm, kullanıcıların Romi web hizmeti [GitHub releases page](#).

Yükseltme gerçekleştirmek için GitHub Bültenleri sayfasından uygun .tgz dosyasını indirin. Ardından, indirilen .tgz dosyasını seçin ve tıklayın *Save* . Güncellenen web hizmeti paketi Raspberry Pi'ye yüklenecek ve yüklenecektir. Kısa bir süre sonra Romi Durumu bölümü kendisini en son sürüm bilgileriyle güncellemelidir.

39.5.4 Harici IO Yapılandırması

External IO Configuration

Each of the 5 external pins can be configured to perform one of three functions: DIO, Analog In or PWM (EXT 0 can only be set to DIO or PWM).

After saving the IO configuration, the *Robot Port* section will update with the appropriate channels to use in robot code.

Romi Pin	EXT 0	EXT 1	EXT 2	EXT 3	EXT 4
Setting	DIO	Analog	Analog	PWM	PWM
Robot Port	Digital 8	Analog In 0	Analog In 1	PWM 2	PWM 3

Save External IO Configuration

Bu bölüm, kullanıcıların Romi üzerindeki 5 harici GPIO kanalını yapılandırmasına olanak tanır.

Not: Raspberry Pi'nin bu bölümün çalışması için ****Writable- Yazılabilir **** modunda olması gerekir.

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.

“Robot Port-Robot Bağlantı Noktası” satırı, yapılandırılmış her GPIO kanalı için uygun WPILib eşlemesini sağlar. Örneğin, EXT 0 bir Dijital IO kanalı olarak yapılandırılmıştır ve WPILib’de bir DigitalInput (veya DigitalOutput) kanalı 8 olarak erişilebilir olacaktır.

39.5.5 IMU Kalibrasyonu

IMU Calibration

Most gyros will have some sort of zero offset. In order to get more accurate rate-of-turn readings, the gyro can be calibrated to calculate an appropriate zero offset.

To calibrate the gyro, place the Romi on a flat surface and click the “Calibrate Gyro” button. While the calibration is running, please do not touch the Romi.

Current Gyro Offsets

X Offset	Y Offset	Z Offset
0.683	-4.305	-2.817

Calibrate Gyro

Not: Raspberry Pi'nin bu bölümün çalışması için ****Writable- Yazılabilir **** modunda olması

gerekir.

Bu bölüm, kullanıcıların jiroskopu Romi üzerinde kalibre etmesine olanak tanır. Jiroskoplarda genellikle bir tür sıfır ofset hatası vardır ve kalibrasyon Romi'nin ofseti hesaplamasına ve hesaplamalarda kullanmasına izin verir.

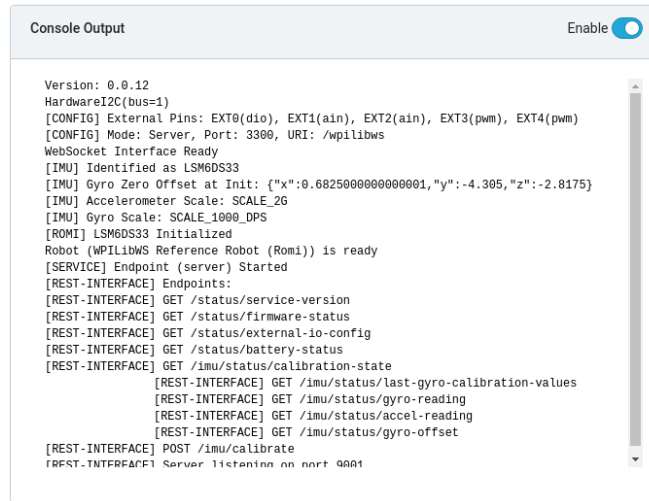
Kalibrasyona başlamak için Romi'yi düz, sabit bir yüzeye yerleştirin. Ardından *Calibrate Gyro* düğmesine tıklayın. Mevcut kalibrasyon sürecini gösteren bir ilerleme çubuğu görünecektir. Kalibrasyon tamamlandıktan sonra, en son ofset değerleri ekranda görüntülenecek ve Romi web hizmetine kaydedilecektir.

Bu ofset değerleri diske kaydedilir ve yeniden başlatmalar arasında kalır.

39.5.6 Firmware

Not: Şu bölüme bakın *Imaging your Romi* ınızı görüntüleme.

39.5.7 Konsol Çıkışı



```
Version: 0.0.12
HardwareI2C(bus=1)
[CONFIG] External Pins: EXT0(dio), EXT1(ain), EXT2(ain), EXT3(pwm), EXT4(pwm)
[CONFIG] Mode: Server, Port: 3300, URI: /wpilibws
WebSocket Interface Ready
[IMU] Identified as LSM6DS33
[IMU] Gyro Zero Offset at Init: {"x":0.6825000000000001,"y":-4.305,"z":-2.8175}
[IMU] Accelerometer Scale: SCALE_2G
[IMU] Gyro Scale: SCALE_1000_DPS
[ROMI] LSM6DS33 Initialized
Robot (WPILibWS Reference Robot (Romi)) is ready
[SERVICE] Endpoint (server) Started
[REST-INTERFACE] Endpoints:
[REST-INTERFACE] GET /status/service-version
[REST-INTERFACE] GET /status/firmware-status
[REST-INTERFACE] GET /status/external-io-config
[REST-INTERFACE] GET /status/battery-status
[REST-INTERFACE] GET /imu/status/calibration-state
[REST-INTERFACE] GET /imu/status/last-gyro-calibration-values
[REST-INTERFACE] GET /imu/status/gyro-reading
[REST-INTERFACE] GET /imu/status/accel-reading
[REST-INTERFACE] GET /imu/status/gyro-offset
[REST-INTERFACE] POST /imu/calibrate
[REST-INTERFACE] Server listening on port 9901
```

Etkinleştirildiğinde, bu bölüm kullanıcıların Romi web hizmetinin sağladığı ham konsol çıktısını görüntülemelerine olanak tanır. Bu, Romi ile ilgili sorunları gidermek veya sadece perde arkasında neler olduğu hakkında daha fazla bilgi edinmek için kullanışlıdır.

39.5.8 Köprü Modu

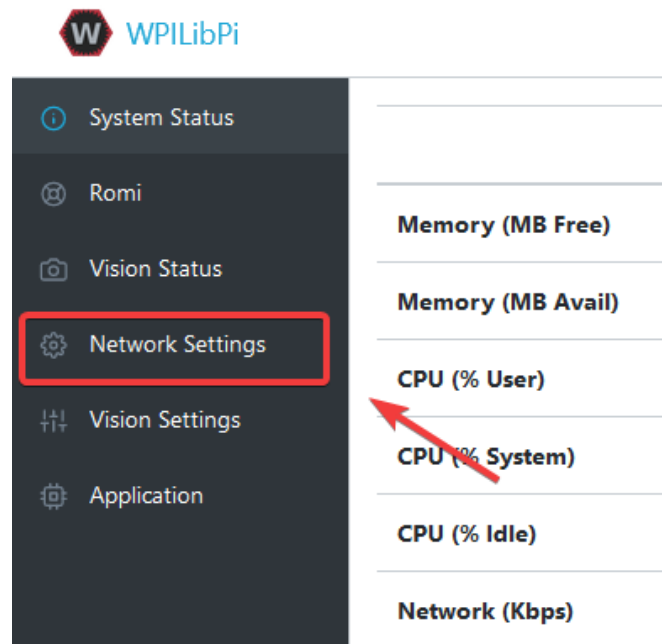
Köprü modu, Romi robotunuzun Erişim Noktası (AP) gibi davranmak yerine bir WiFi ağına bağlanmasına olanak tanır. Bu, Romi'yi ekstra donanım olmadan kullanırken interneti kullanabileceğiniz için özellikle uzaktan öğrenme ortamlarında kullanışlıdır.

Not: Köprü modunun kısıtlı ağ ortamlarında (Eğitim Kurumları) düzgün çalışması olası değildir.

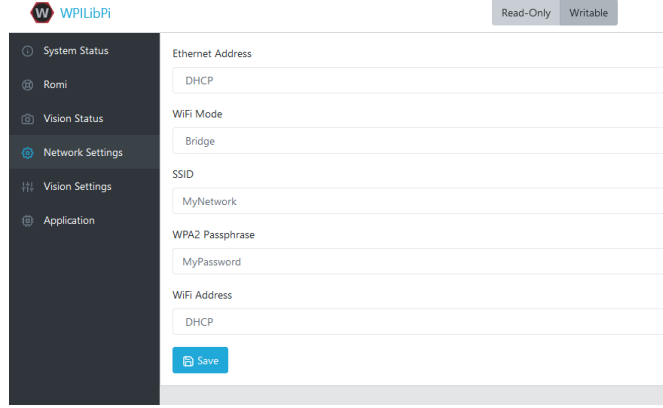
1. Üst menüde *Yazılabilir* seçeneğini etkinleştirin.



2. :guilabel:`Ağ Ayarları`na tıklayın.



3. Aşağıdaki ağ ayarları uygulanmalıdır:



- **Ethernet:** DHCP
- **WiFi Modu:** Köprü
- **SSID:** Ağınızın SSID'si (adı)
- **WPA2 Parolası:** Wi-Fi ağınızın parolası
- **WiFi Adresi:** DHCP

Ayarlar uygulandıktan sonra lütfen Romi'yi yeniden başlatın. Artık belirlediğiniz ağa bağlıyken web tarayıcınızda `wpilibpi.local` konumuna gidebilmelisiniz.

Romi'ye Erişilemiyor

Romi doğru köprü ayarlarına sahipse ve buna erişemiyorsanız, birkaç geçici çözümümüz var.

- Romi'ye Ethernet
- Romi'yi yeniden şekillendirin

Bazı kısıtlı ağlar, Romi çözümlemesinin ana bilgisayar adını etkileyebilir, IP adresini bulmak için [Angry IP Scanner](#) kullanarak bu sorunu çözebilirsiniz.

Uyarı: Angry IP Scanner, ağınızdaki cihazlara ping gönderirken bazı antivirüsler tarafından casus yazılım olarak işaretlenir! Güvenli bir uygulamadır!

39.6 Romi'yi Programlamak

Romi için bir program yazmak, normal bir FRC robotu için bir program yazmaya çok benzer. Aslında, tüm aynı araçlar (Visual Studio Code, Driver Station, SmartDashboard, vb.) Romi ile kullanılabilir.

39.6.1 Bir Romi Programı Oluşturmak

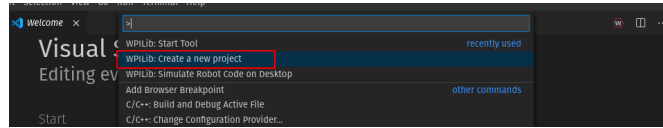
Creating a new program for a Romi is like creating a normal FRC program, similar to the [Zero To Robot](#) programming steps.

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.

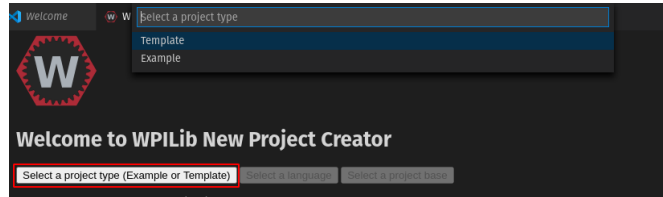
Not: In order to program the Romi using C++, a compatible C++ desktop compiler must be installed. See [Robot Simulation - Additional C++ Dependency](#).

Yeni bir WPILib Romi Projesi Oluşturma

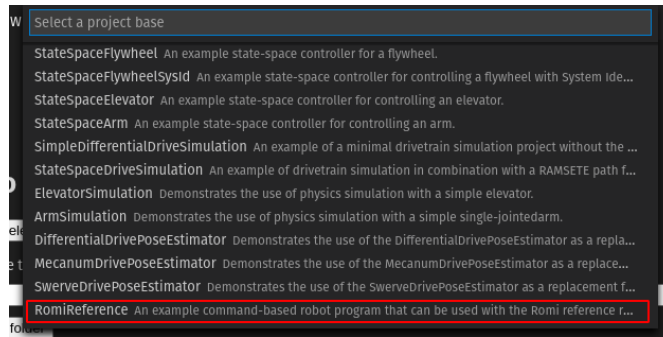
Visual Studio Code komut paletini şu şekilde getirin **Ctrl+Shift+P** ve komut istemine "New project-Yeni proje" yazın. "Create a new project-Yeni proje oluştur" komutunu seçin:



Bu, "New Project Creator Window-Yeni Proje Oluşturucu Penceresini" getirecektir. Buradan, "Select a project type-Bir proje türü seçin (Örnek veya Şablon) seçeneğini tıklayın ve beliren istemden "Example-Örnek "i seçin:



Ardından, bir örnek listesi görünecektir. "RomiReference" örneğini bulmak için listede ilerleyin:



"New Project Creator-Yeni Proje Oluşturucu" daki kalan alanları doldurun ve yeni robot projesini oluşturmak için "Generate Project-Proje Oluştur" u tıklayın.

Bir Romi Programı Çalıştırmak

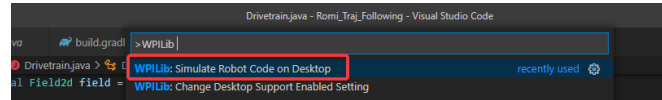
Once the robot project is generated, it is essentially ready to run. The project has a pre-built Drivetrain class and associated default command that lets you drive the Romi around using a joystick.

One aspect where a Romi project differs from a regular FRC robot project is that the code is not deployed directly to the Romi. Instead, a Romi project runs on your development computer and leverages the WPILib simulation framework to communicate with the Romi robot.

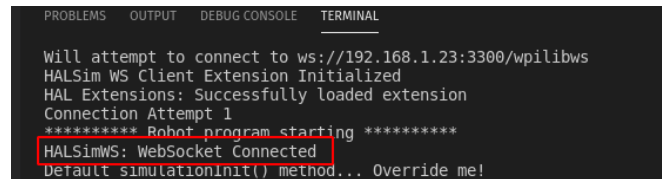
To run a Romi program, first, ensure that your Romi is powered on. Next, connect to the WPILibPi-<number> WiFi network broadcast by the Romi. If you changed the Romi network settings (for example, to connect it to your own WiFi network) you may change the IP address that your program uses to connect to the Romi. To do this, open the build.gradle file and update the wpi.sim.envVar line to the appropriate IP address.

```
43 //Sets the websocket client remote host.  
44 wpi.sim.envVar("HALSIMWS_HOST", "10.0.0.2")  
45 wpi.sim.addWebsocketsServer().defaultEnabled = true  
46 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.



Her şey yolunda giderse, konsol çıktısında “HALSimWS:WebSocket Connected” yazan bir satır görmelisiniz:



Romi kodunuz şimdi çalışıyor!

39.7 Romi Programlama(LabVIEW)

Romi için LabVIEW programı yazmak, herhangi bir roboRIO tabanlı robot programı yazmaya çok benzer. Aslında, bütün benzer araçlar Romi ile kullanılabilir.

39.7.1 Romi Projesi Oluşturmak

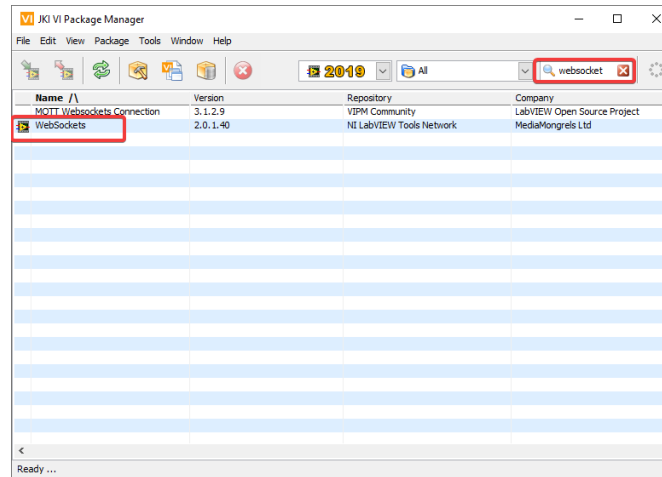
Romi için yeni bir program oluşturmak, normal bir FRC [reg] programı oluşturmaktan farklı değildir, *Sıfırdan Robota* programlama adımlarına benzer. Başlangıçta, Romi donanımı roboRIO robotunuzdan farklı bağlantı noktalarına bağlı olabileceğinden, sadece Romi üzerinde kullanmak için ayrı bir proje oluşturmak isteyebilirsiniz.

Romi Robot, sol ve sağ taraf için sırasıyla *PWM* 0 ve 1 portlarını kullanmıştır.

WebSockets VI'ı indirme

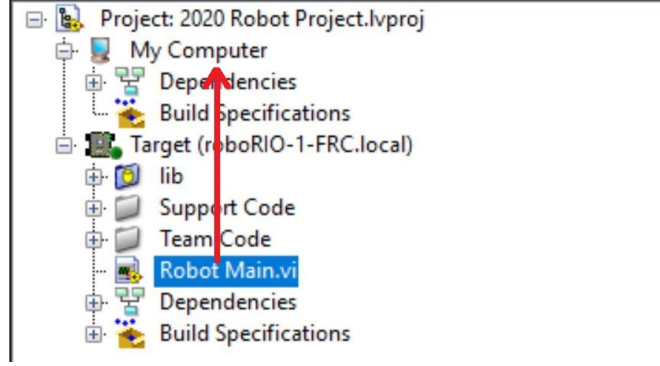
Romi projesinin normal bir FRC [reg] robot projesinden farklı olduğu bir nokta, kodun doğrudan Romi'ye dağıtılmamasıdır. Bunun yerine, bir Romi projesi geliştirme bilgisayarınızda çalışır ve Romi robotu ile iletişim kurmak için WPILib simülasyon çerçevesinden yararlanır. WebSockets, LabVIEW'in Romi ile iletişim kurmak için kullandığı protokoldür.

VI Package Manager uygulamasını aç. Sağ üstteki arama kutusuna websockets yaz. :guilabel:`LabVIEW Tools Network`ten VI'yı seç.



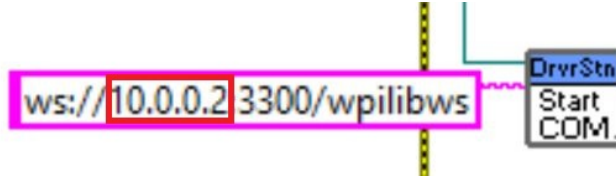
Proje Hedefini Değiştirme

LabVIEW programınızı Romi üzerinde çalıştırmak için gereken birincil adım, hedefi Masaüstü olarak değiştirmektir. Proje hedefini değiştirmek için, Proje Gezgini'nde Robot Main VI'yı bulun ve tıklayıp Hedef bölümünden Bilgisayarım bölümüne sürükleyin.



Hedeflenen IP'yi Belirlemek

Varsayılan olarak, LabVIEW programınız 10.0.0.2 IP adresine sahip bir Romi'ye bağlanmaya çalışacaktır. Farklı bir IP kullanmak isterseniz, bunu Robot Main içindeki Driver Station Start Communication VI için bir girdi olarak belirtebilirsiniz. Simulation URL si için pembe giriş terminalini bulun, ardından sağ tıklayın ve varsayılan değerle önceden doldurulmuş bir sabit oluşturmak için *Create Constant* seçeneğini seçin. Daha sonra metnin IP adresi kısmını değiştirebilir, protokol bölümünü (başlangıçta) ve port ve son eki (sonda) aynı bırakmaya dikkat edebilirsiniz.



Bir Romi Programı Çalıştırmak

Bir Romi programı çalıştırmak için ilk başta Romi'nin açık olduğuna emin olun. Romi'nin WPILibPi-<number> ağ yayınına bağlandığınızda bilgisayarınızda Romi programını çalıştırmak için beyaz *Çalıştır* okuna basın.

Artık Romi kodunuz çalışıyor! Program, otomatik olarak belirttiğiniz IP'ye veya bir IP belirtmediyseniz varsayılan IP'ye bağlanmayı çalışacak.

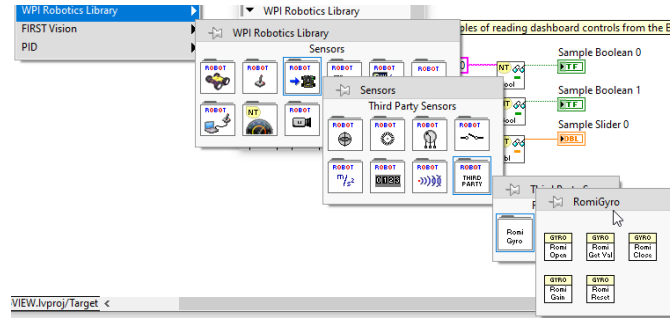
Driver Station yazılımının LabVIEW koduyla aynı bilgisayarda çalıştırılması önerilir. Programınız Driver Station'a başarıyla bağlandığında, Driver Station'a kodun Masaüstünde çalıştığını otomatik olarak bildirecek ve Driver Station'ın içindeki herhangi bir bilgiyi değiştirmenize gerek kalmadan Driver Station'ın bağlanmasını sağlayacaktır. Ardından, Driver Station'ı Romi'nize yönlendirmeniz gerekir. Bu, takım numarasını 127.0.0.1 olarak ayarlayarak yapılır. Daha sonra robot modunu ayarlamak ve normal şekilde etkinleştirmek/devre dışı bırakmak için Driver Station'daki kontrolleri kullanabilirsiniz.

Not: Robot kodunuz Romi'ye bağlanamazsa, Driver Station da bağlantı olmadığını gösterecektir.

Gyro veya Encoder'ı kullanmak

Romi'deki gyro, RomiGyro fonksiyonuyla kullanılabilir. Bu fonksiyon, şurada bulunabilir

- WPI Robotics Library
 - Sensors
 - Third Party Libraries
 - RomiGyro



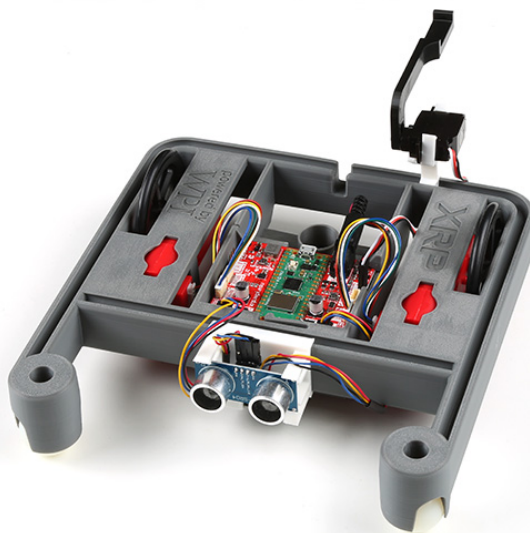
Encoder'lar standart encoder fonksiyonuyla kullanılabilir. DIO portları da şunlardır:

- Sol(4, 5)
- Sağ(6, 7)

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

Not: 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

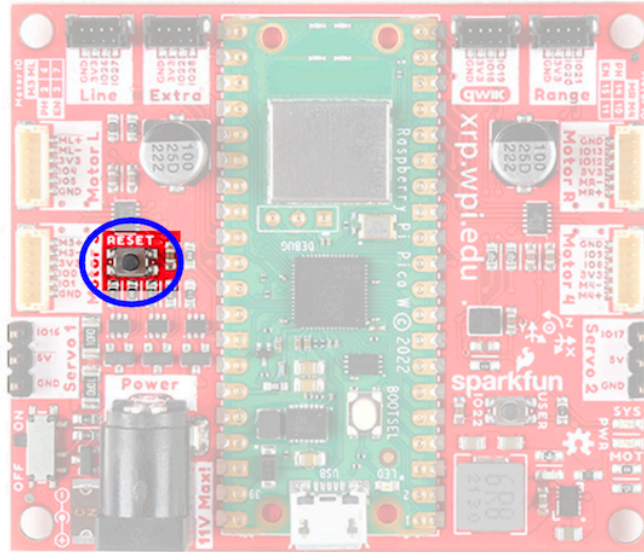
The XRP firmware must be downloaded and written to the Pico W. Click on Assets at the bottom of the description to see the available image files:

[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 B00TSEL button (the white button on the green Pico W, near the USB connector), quickly press the reset button (circled below), and then release the B00TSEL button.

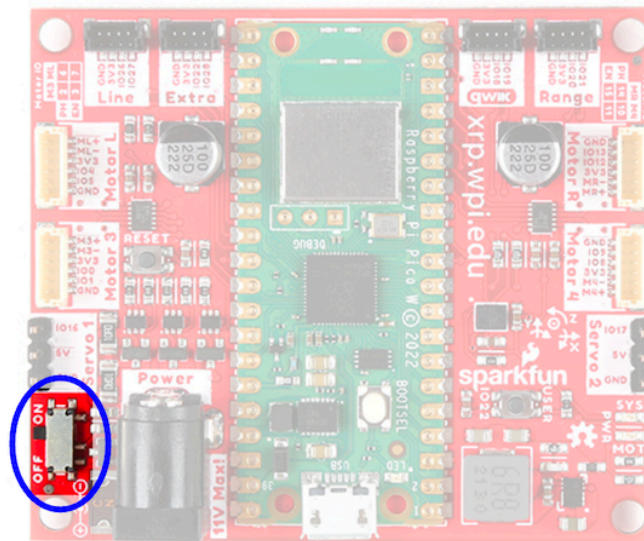


4. The board will temporarily disconnect from your computer, and then reconnect as a USB storage device named RPI-RP2.
5. Drag the .uf2 firmware file into the RPI-RP2 drive, and it will automatically update the firmware.
6. Once complete, the RPI-RP2 USB storage device will disconnect. At this point, 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.

Not: 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.

Not: 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.

Not: 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

Not: 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.

Not: 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

Not: 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

Not: 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

Not: The PWM motor controller classes (e.g. Spark) and Servo are not supported. The XRP requires use of specialized XRPMotor and XRPServo classes.

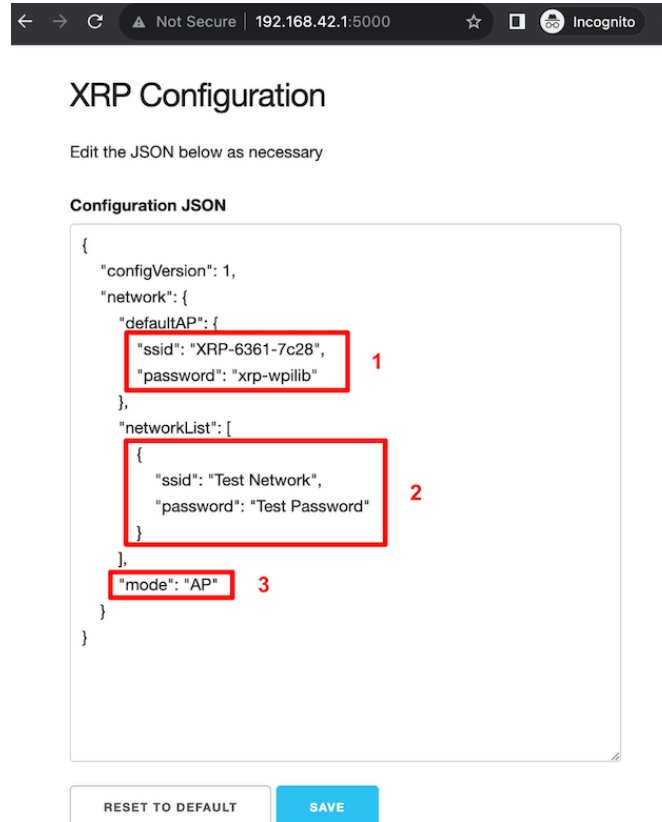
The following classes are provided by the XRP Vendordep (built-in to WPILib).

- 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.

Not: 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

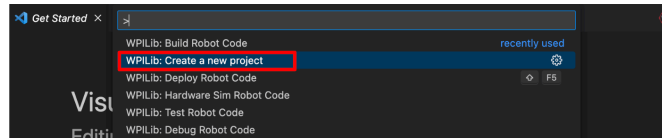
Creating a new program for an XRP is like creating a normal FRC program, similar to the *Zero To Robot* programming steps.

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.

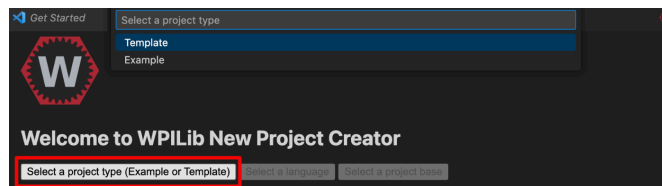
Not: In order to program the XRP using C++, a compatible C++ desktop compiler must be installed. See *Robot Simulation - Additional C++ Dependency*.

Creating a New WPILib XRP Project

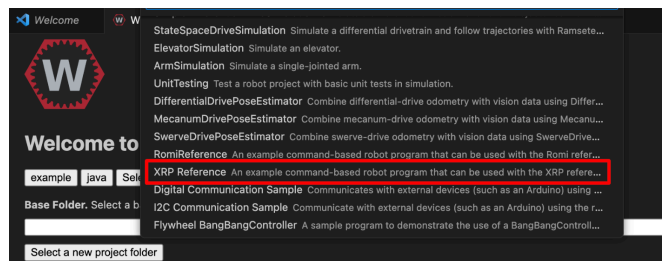
Bring up the Visual Studio Code command palette with `Ctrl+Shift+P`, and type “New project” into the prompt. Select the “Create a new project” command:



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

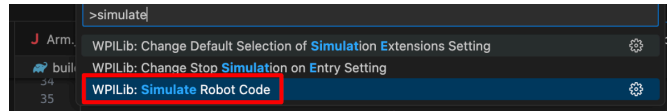
Once the robot project is generated, it is essentially ready to run. The project has a pre-built `Drivetrain` class and associated default command that lets you drive the XRP around using a joystick.

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.addXRPCClient().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.

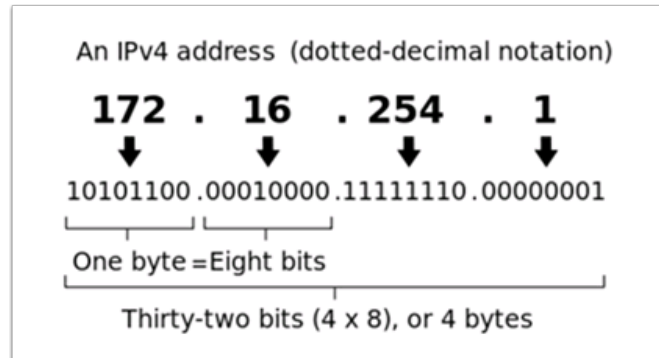
Your XRP code is now running!

Bu bölüm, sürücü istasyonu ile roboRIO arasındaki iletişimle ilgili temel robot yapılandırmasını ve kullanımını özetlemektedir.

41.1 Ağ Temelleri

41.1.1 IP Adresi nedir?

IP adresi, bir ağdaki her bir cihazı tanımlayan, noktalarla ayrılmış kendine has bir sayı dizisidir. Her IP adresi 0-255 arasında değişen 4 bölüme (oktetlere) bölünmüştür.



Yukarıda gösterildiği gibi, bu, her IP adresinin 32 bitlik bir adres olduğu anlamına gelir, yani 2^{32} adres veya yaklaşık 4.300.000.000 adres olasılığıdır. Ancak bunların çoğu, web sunucuları gibi şeyler için halka açık olarak kullanılmaktadır.

Bu, IP Adreslemenin **ilk anahtar noktasını** ortaya çıkarır: Ağdaki her cihazın benzersiz bir IP adresi olmalıdır. İki cihaz aynı IP adresine sahip olamaz, aksi takdirde çakışmalar meydana gelir.

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.

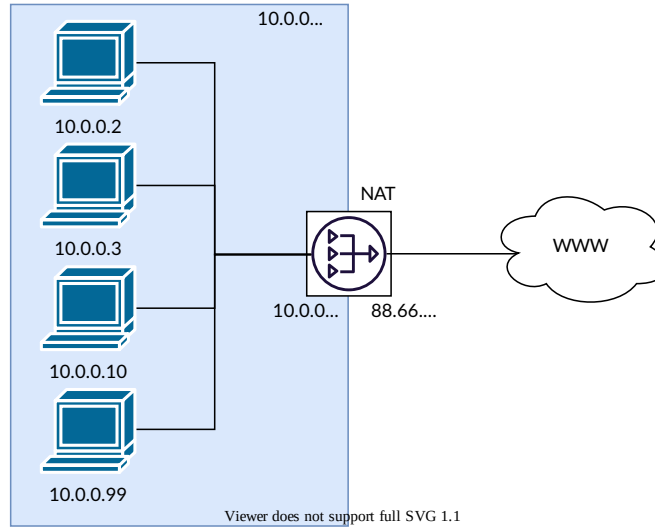
41.1.2 Genel ve Özel IP Adresleri

To be efficient with using IP Addresses, the idea of “Reserved IP Ranges” was implemented. In short, this means that there are ranges of IP Addresses that will never be assigned to web servers, and will only be used for local networks, such as those in your house.

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:

Sınıf	Bit Sayısı	Başlangıç Adresi	Bitiş Adresi	Adres Sayısı
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

These reserved ranges let us assign one “unreserved IP Address” to an entire house, and then use multiple addresses in a reserved range to connect more than one computer to the internet. A process on the home’s internet router known as **NAT** (Network Address Translation), handles the process of keeping track which private IP is requesting data, using the public IP to request that data from the internet, and then passing the returned data back to the private IP that requested it. This allows us to use the same reserved IP addresses for many local networks, without causing any conflicts. An image of this process is presented below.



Not: FRC için | reg | networks, 10.0.0.0`aralığını kullanacağız. Bu aralık, IP adresleri için ``10.TE.AM.xx formatını kullanmamıza izin verirken, Sınıf B veya C ağları kullanmak, yalnızca bir takım alt kümesinin formatı takip etmesine izin verir. Bu biçimlendirmenin bir örneği, FRC Team 1750 için 10.17.50.1 olabilir.

41.1.3 Bu adresler nasıl atanır?

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).

Dinamik Atama

IP adreslerini dinamik olarak atamak, ağdaki bir cihazın IP adresi atamalarını yönetmesine izin verdiğimiz anlamına gelir. Bu, Dinamik Ana Bilgisayar Yapılandırma Protokolü (DHCP) aracılığıyla yapılır. DHCP'nin birçok bileşeni vardır, ancak bu belgenin kapsamı için, bunu ağı otomatik olarak yöneten bir hizmet olarak düşüneceğiz. Ağa yeni bir aygıt bağladığınızda, DHCP hizmeti yeni aygıtı görür, ardından ona kullanılabilir bir IP adresi ve aygıtın iletişim kurması için gereken diğer ağ ayarlarını sağlar. Bu, her cihazın tam IP adresini bilmediğimiz zamanlar olduğu anlamına gelebilir.

DHCP sunucusu nedir?

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.

Bu, ağ cihazlarınıza IP adresleri atayan bir DHCP sunucusuna sahip olmanız gereken bir durumla karşılaşırsanız, en yakın ev yönlendiricisini bulup fişe takmak kadar basit olduğu anlamına gelir.

Statik Atama

Statik olarak IP adresleri atamak, ağdaki her cihaza sahip olmasını istediğimiz IP adresini manuel olarak söylediğimiz anlamına gelir. Bu yapılandırma, her cihazdaki bir ayar aracılığıyla gerçekleşir. Ağda DHCP'yi devre dışı bırakarak ve adresleri manuel olarak atayarak, ağdaki her bir cihazın tam IP adresini bilmenin avantajını elde ederiz, ancak her birini manuel olarak ayarladığımız ve kullanılan IP adreslerini takip eden bir hizmet olmadığından, bunu kendimiz takip etmeliyiz. IP adreslerini statik olarak ayarlarken, yinelenen adresler atamamaya dikkat etmeliyiz ve diğer ağ ayarlarını (alt ağ maskesi ve varsayılan ağ geçidi gibi) her cihazda doğru şekilde yaptığımızdan emin olmalıyız.

41.1.4 Yerel bağlantı nedir?

Bir cihazın IP adresi yoksa ağ üzerinden iletişim kuramaz. Adresini bir DHCP sunucusundan dinamik olarak alacak bir cihazımız varsa, ancak ağda DHCP sunucusu yoksa bu bir sorun haline gelebilir. Bunun bir örneği, doğrudan bir roboRIO'ya bağlı bir dizüstü bilgisayarınız olması ve her ikisinin de dinamik olarak bir IP adresi alacak şekilde ayarlanması olabilir. Hiçbir cihaz bir DHCP sunucusu değildir ve ağdaki tek iki cihaz oldukları için, IP adreslerine otomatik olarak atanmayacaktır.

Link-local addresses give us a standard set of addresses that we can “fall-back” to if a device set to acquire dynamically is not able to acquire an address. If this happens, the device will assign itself an IP address in the 169.254.xx.yy address range; this is a link-local address. In our roboRIO and computer example above, both devices will realize they haven't been assigned an IP address and assign themselves a link-local address. Once they are both assigned addresses in the 169.254.xx.yy range, they will be in the same network and will be able to communicate, even though they were set to dynamic and a DHCP server did not assign addresses.

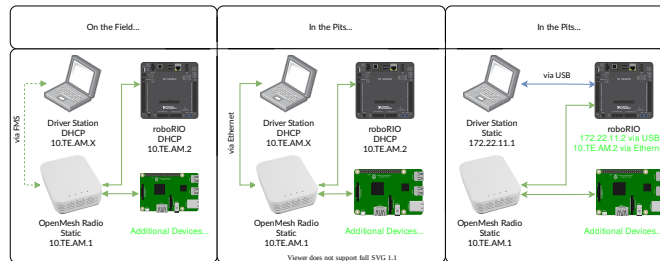
41.1.5 FRC için IP Adresleme

Daha fazla bilgi için bu belgeye bakın: [IP Ağ İletişim Makalesi](#) .

Dinamik ve Statik Yapılandırmaları Karıştırma

Ekip sahadayken, yukarıdaki bölümde belirtildiği gibi cihazların 10.TE.AM.xx aralığında statik olarak ayarlanması ve belirtilen IP adresi çakışması olmadığı sürece sahanın DHCP adreslerini ataması ile ilgili herhangi bir sorun fark etmemelidir.

Yarışma sırasında, bir ekip aşağıdaki nedenden dolayı Statik ve DHCP cihazlarını karıştırırsa sorunlarla karşılaşabilir. Yukarıda belirtildiği gibi, DHCP cihazları bir sunucu yoksa bağlantı yerel adresine (169.254.xx.yy) geri dönecektir. Statik cihazlar için IP adresi her zaman aynı olacaktır. DHCP sunucusu yoksa ve roboRIO, sürücü istasyonu ve dizüstü bilgisayar yerel bağlantı adreslerine geri dönerse, 10.TE.AM.xx aralığındaki statik olarak ayarlanmış cihazlar farklı bir ağda olacak ve yerel bağlantı adreslerine sahip olanlar tarafından görülmeyecektir. Bunun görsel bir açıklaması aşağıda verilmiştir:



Uyarı: USB aracılığıyla roboRIO'ya bağlandığında, OpenMesh radyoya bağlı cihazlara (yukarıda gösterilen yeşil ağda) erişmek için [Port Yönlendirme](#) yapılandırması gereklidir.

Available Network Ports

Please see R704 of the 2024 Game Manual for information regarding available network ports.

41.1.6 mDNS

mDNS veya multicast Domain Name System, ağda bir DNS sunucusu olmadan DNS'nin özelliklerinden yararlanmamızı sağlayan bir protokoldür. Bunu daha net hale getirmek için bir adım geri gidelim ve DNS'nin ne olduğu hakkında konuşalım.

DNS Nedir?

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.

Örnek 2: Ev ağınızda, dizüstü bilgisayarınızdan bağlanmak istediğiniz MYCOMPUTER adlı bir sunucunuz var. Ağınız DHCP kullanıyor, bu nedenle MYCOMPUTER IP Adresini bilmiyorsunuz, ancak DNS yalnızca MYCOMPUTER adını kullanarak bağlanmanıza izin veriyor. Ek olarak, DHCP atamaları her yenilendiğinde, MYCOMPUTER farklı bir adresle sonuçlanabilir, ancak belirli bir IP adresi yerine MYCOMPUTER adını kullanarak bağlandığınız için, DNS kaydı güncellenmiş olur ve siz yine de bağlanabilirsiniz.

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 için 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 bize geleneksel DNS ile aynı avantajları sağlar, ancak bir sunucu gerektirmeyen bir şekilde uygulanır. Bir kullanıcı bir cihaza kolay bir ad kullanarak bağlanmak istediğinde, mDNS bu adı taşıyan cihazdan kendisini tanıttırmasını isteyen bir mesaj gönderir. İsme sahip cihaz daha sonra IP adresini içeren bir geri dönüş mesajı gönderir, böylece ağdaki tüm cihazlar bilgilerini güncelleyebilir. mDNS, roboRIO'muza `roboRIO-TEAM-FRC.local` olarak atıfta bulunmamızı ve bir DHCP ağına bağlanmasını sağlayan şeydir.

Not: FRC için kullanılan bir cihaz mDNS'yi desteklemiyorsa, ona 10.0.0.20 - 10.0.0.255 aralığında bir IP Adresi atanacaktır, ancak bağlanmak için tam adresi bilmeyeceğiz ve eskisi gibi kolay adını kullanamayacağız. Bu durumda, cihazın statik bir IP Adresine sahip olması gerekir.

mDNS - İlkeleri

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 - Sağlayıcılar

mDNS'yi kullanmak için, PC'nize bir mDNS uygulamasının yüklenmesi gerekir. Her büyük platform için bazı yaygın mDNS uygulamaları şunlardır:

Windows:

- **NI mDNS Responder:** NI FRC Game Tools ile Yüklenir
- **Apple Bonjour:** iTunes ile yüklenir

OSX:

- **Apple Bonjour:** Varsayılan olarak yüklenir

Linux:

- **nss-mDNS/Avahi/Zeroconf:** Bazı Linux varyantlarında (Ubuntu veya Mint gibi) varsayılan olarak yüklenir ve etkinleştirilir. Başkalarına yüklenmesi veya etkinleştirilmesi gerekebilir (Arch gibi)

mDNS - Güvenlik Duvarları

Not: Bilgisayar yapılandırmanıza bağlı olarak, herhangi bir değişiklik gerekmez, bu bölüm sorun gidermeye yardımcı olmak için sağlanmıştır.

Düzenli çalışması için mDNS'nin güvenlik duvarınızdan geçmesine izin verilmelidir. Ağ trafiği mDNS uygulamasından geldiğinden ve doğrudan Driver Station veya IDE'den gelmediğinden, bu uygulamalara izin vermek yeterli olmayabilir. mDNS güvenlik duvarı sorunlarını çözmek için iki ana yolu vardır:

- mDNS uygulaması için bir uygulama/hizmet istisnası ekleyin (NI mDNS Responder ``C:\Program Files\National Instruments\Shared\mDNS Responder\mDNSResponder.exe`` dir)
- UDP 5353'e gelen/giden trafik için bir bağlantı noktası istisnası ekleyin. IP Aralıkları:
 - 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 - Tarayıcı desteği

Çoğu web tarayıcısı, bir mDNS sağlayıcısı kurulu olduğu sürece roboRIO web sunucusuna erişmek için mDNS adresini kullanabilmelidir. Bu tarayıcılar arasında Microsoft Edge, Firefox ve Google Chrome bulunur.

41.1.7 USB

USB arayüzünü kullanıyorsanız, ağ kurulumuna gerek yoktur (*FRC Game Tools Araçlarını Yükleme*'ye ihtiyacınız vardır). RoboRIO sürücüsü, ana bilgisayarın (bilgisayarınız) ve roboRIO'nun IP adresini otomatik olarak yapılandıracaktır ve yukarıda listelenen yazılım roboRIO'nuzu bulabilmeli ve kullanabilmelidir.

41.1.8 Ethernet/Wireless

The *Radyonuzu Programlama* 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 Özet

IP Adresleri, bir ağdaki cihazlarla iletişim kurmamızı sağlayan şeydir. FRC için, bir DHCP sunucusuna bağlıysak veya statik olarak atanmışlarsa, bu adresler 10.TE.AM.xx aralığında veya cihazlar DHCP olarak ayarlanmışsa, ancak sunucu mevcut değilse ``169.254.xx.yy`` aralığında olacaktır. IP Adreslerinin nasıl çalıştığı hakkında daha fazla bilgi için Microsoft'un [buradaki](#) makalesine bakın.

Ağdaki tüm cihazlar mDNS'yi destekliyorsa, tüm cihazlar DHCP'ye ayarlanabilir ve kolay adlarıyla anılabilir (örn. ``roboRIO-TEAM-FRC.local``). Bazı cihazlar mDNS'yi desteklemiyorsa, bunların statik adresleri kullanacak şekilde ayarlanması gerekecektir.

Tüm cihazlar DHCP veya Statik IP atamalarını (doğru statik ayarlarla) kullanacak şekilde ayarlanmışsa, iletişim herhangi bir değişiklik gerekmeden hem sahada hem de pit alanında çalışacaktır. Bazı Statik ve bazı DHCP cihazlarının bir karışımı varsa, Statik cihazlar sahaya bağlanacak, ancak pite alanında bağlanmayacaktır. Bu, tüm aygıtları statik ayarlara ayarlayarak veya mevcut ayarları bırakıp pit alanında bir DHCP sunucusu sağlayarak çözülebilir.

41.2 IP Yapılandırmaları

Not: Bu belge, hem sahalarda hem de pitlerde etkinliklerde kullanılan IP yapılandırmasını, olası sorunları ve geçici çözüm yapılandırmalarını açıklamaktadır.

41.2.1 TE.AM IP Gösterimi

TE.AM gösterimi, bu belgedeki birçok yerde IP'lerin bir parçası olarak kullanılmıştır. Bu gösterim, dört basamaklı takım numaranızı IP adresi için iki basamaklı çifte bölmeyi ifade eder.

Örnek: 10.TE.AM.2

Takım 12 - 10.0.12.2

Takım 122 - 10.1.22.2

Takım 1212 - 10.12.12.2

Team 1202 - 10.12.2.2

Team 1220 - 10.12.20.2

Takım 3456 - 10.34.56.2

41.2.2 Sahada

Bu bölümde maç oynamak için Saha Ağına bağlanıldığında ağ bağlantısı açıklanmaktadır.

Sahada DHCP Yapılandırması

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 veya OM5P-AC radyo - Kiosk tarafından programlanmış Statik 10.TE.AM.1
- roboRIO - Robot Radio tarafından atanan DHCP adresi 10.TE.AM.2
- Driver Station - DHCP ("Obtain an IP address automatically") 10.TE.AM.X assigned by field
- IP kamera (kullanılıyorsa) - Robot Radio tarafından atanan DHCP 10.TE.AM.Y
- Diğer cihazlar (kullanılıyorsa) - Robot Radio tarafından atanan DHCP 10.TE.AM.Z

Sahada Statik Yapılandırma

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 radyo - Kiosk tarafından programlanmış Statik ` 10.TE.AM.1
- roboRIO - Statik 10.TE.AM.2 makul bir seçim olacaktır, 255.255.255.0 alt ağ maskesi (varsayılan)
- 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 Kamera (kullanılıyorsa) - Statik 10.TE.AM.11 makul bir seçim olacaktır, 255.255.255.0 alt ağı uygun olacaktır
- Diğer cihazlar - Statik 10.TE.AM.6-.10 veya .12-.19 (.11 kamera yoksa) alt ağı 255.255.255.0

41.2.3 Pit Alanlarında

Not: 2018'deki yenilikler: Artık etkinlik yapılandırmasında Robot Radio' nun kablolu tarafında çalışan bir DHCP sunucusu var.

Pitlerde DHCP Yapılandırması

- OpenMesh radyo - Kiosk tarafından programlanan Statik 10.TE.AM.1 .
- roboRIO - Robot Radio tarafından atanan 10.TE.AM.2
- Driver Station - DHCP ("Obtain an IP address automatically"), 10.TE.AM.X, assigned by Robot Radio
- IP kamera (kullanılıyorsa) - Robot Radio tarafından atanan DHCP, 10.TE.AM.Y
- Diğer cihazlar (kullanılıyorsa) - Robot Radio tarafından atanan DHCP, 10.TE.AM.Z

Pitlerde Statik Yapılandırma

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 Ağ Sorunlarını Giderme

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 MDNS kullanarak roboRIO'yu pingleyin

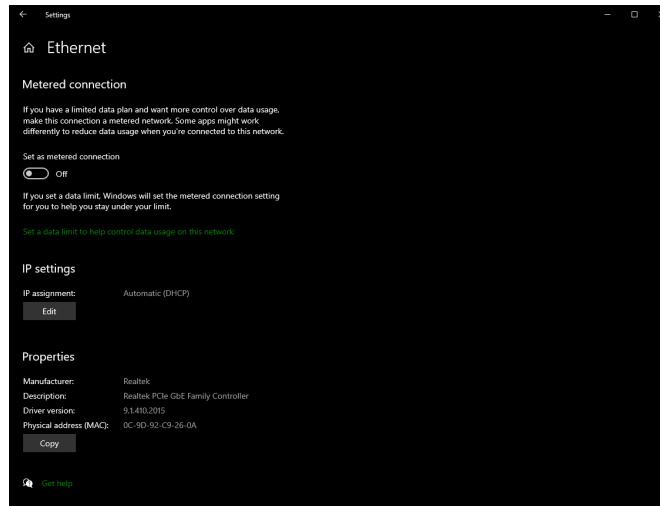
RoboRIO ağ iletişimi sorunlarını belirlemenin ilk adımı, bunun bir uygulama sorunu mu yoksa genel bir ağ sorunu mu olduğunu belirlemektir. Bunu yapmak için, komut istemi açmak için **Start -> cmd yazın -> Enter'a basın.** `ping roboRIO-####-FRC.local` yazın, burada #### takım numaranızdır (başında sıfır olmadan) ve enter tuşuna basın. Ping başarılı olursa, sorun büyük olasılıkla belirli bir uygulamadadır, uygulamadaki ekip numarası yapılandırmanızı doğrulayın ve güvenlik duvarı yapılandırmanızı kontrol edin.

41.3.2 RoboRIO IP Adresine ping atma

Yanıt yoksa, `10.TE.AM.2` (*TE.AM IP Notation*) yukarıda açıklandığı gibi hemen ping atmayı deneyin. Bu işe yararsa, PC'nizdeki mDNS adresini çözmede sorun yaşarsınız. En yaygın iki neden, sistemde kurulu bir mDNS çözücünün olmaması ve ağda .local adresini normal DNS kullanarak çözümlemeye çalışan bir DNS sunucusudur.

- 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](#).
- Bilgisayarınızın diğer tüm ağlarla olan bağlantısını kesin ve OM5P-AN'nin bir erişim noktası olarak yapılandırıldığından emin olun : ref: *FRC Radyo Yapılandırma Yardım-cı Programı* <docs/zero-to-robot/step-3/radio-programming:Programming your Radio>. Sistemden diğer yönlendiricilerin kaldırılması, soruna neden olan bir DNS sunucusu olmadığının doğrulanmasına yardımcı olacaktır.

41.3.3 Ping Başarısız



IP adresine doğrudan ping atmak başarısız olursa, bilgisayarın ağ yapılandırmasıyla ilgili bir sorununuz olabilir. PC **Automatic** olarak yapılandırılmalıdır. Bunu kontrol etmek için, tıklayın: *Start -> Settings -> Network & Internet* e tıklayınız. Ağınıza bağlı olarak şunu seçin *Wifi* veya *Ethernet*. Ardından bağlı ağınıza tıklayın. **IP settings** 'na gidin ve tıklayın *Edit* ve *Automatic (DHCP)* seçeneğinin seçili olduğundan emin olun.

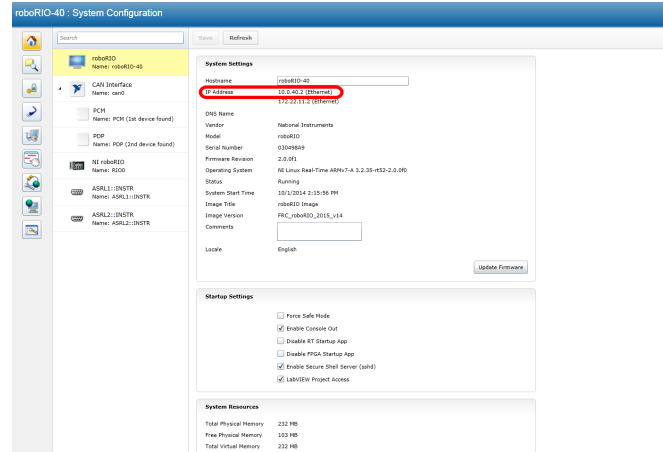
41.3.4 USB Bağlantısı Sorunlarını Giderme

USB bağlantısındaki sorunu gidermeye çalışıyorsanız, roboRIO'nun IP adresine ping atmayı deneyin. Bilgisayara bağlı yalnızca bir roboRIO olduğu sürece, 172.22.11.2 olarak yapılandırılmıştır. Bu ping başarısız olursa, roboRIO'nun bağlı ve çalıştığından ve NI FRC Game Tool araçlarının yüklü olduğundan emin olunuz. Game Tools, USB bağlantısı için gerekli roboRIO sürücülerini de yükler.

Bu ping başarılı olursa, ancak .local ping işlemi başarısız olursa, roboRIO ana bilgisayar adı yanlış yapılandırılmış olabilir veya .local adresini çözmeye çalışan bir DNS sunucusuna bağlanmış olabilirsiniz.

- Verify that your roboRIO has been imaged for your team number: *roboRIO 1 roboRIO 2*. This sets the hostname used by mDNS.
- *Diğer tüm ağ bağdaştırıcılarını devre dışı bırak*

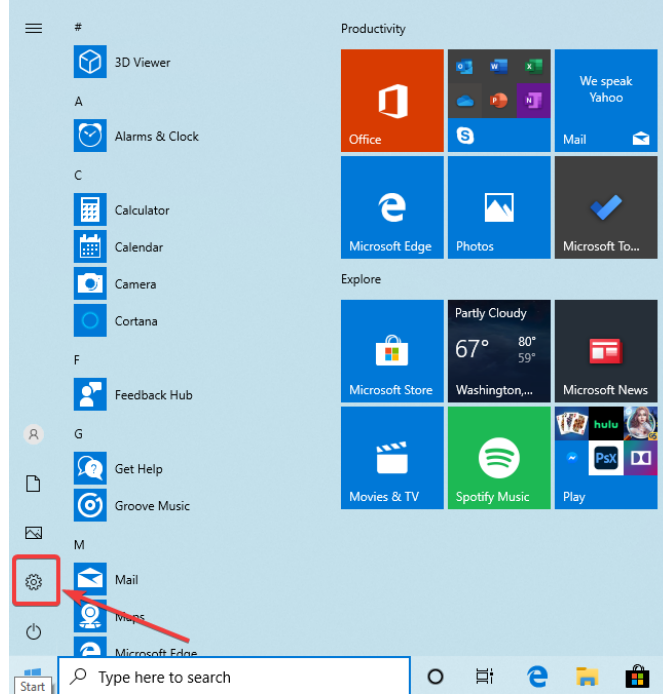
41.3.5 Ethernet Bağlantısı



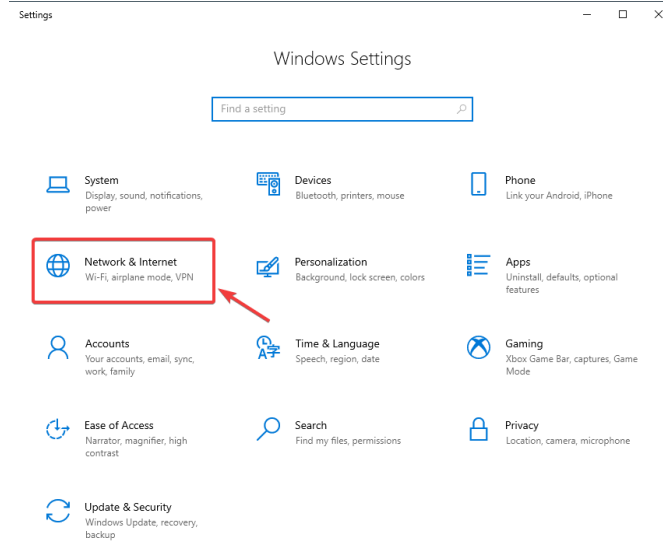
Bir Ethernet bağlantısında sorun gideriyorsanız, önce USB bağlantısını kullanarak roboRIO'ya bağlanabildiğinizden emin olmanız faydalı olabilir. USB bağlantısını kullanarak *roboRIO webdashboard* arayüzünü açın ve roboRIO'nun ethernet arayüzünde bir IP adresine sahip olduğunu doğrulayın. RoboRIO'ya doğrudan bağlantı kuruyorsanız, bu kendi kendine atanan bir 169.*.*.* adresi olmalıdır, OM5P-AN radyo ile bağlıysanız, 10.TE.AM.XX burada TEAM, dört haneli FRC takım numaranızdır. Buradaki tek IP adresi USB adresiyse, fiziksel roboRIO ethernet bağlantısını kontrol edin.

41.3.6 Ağ Adaptörlerini Devre Dışı Bırakma

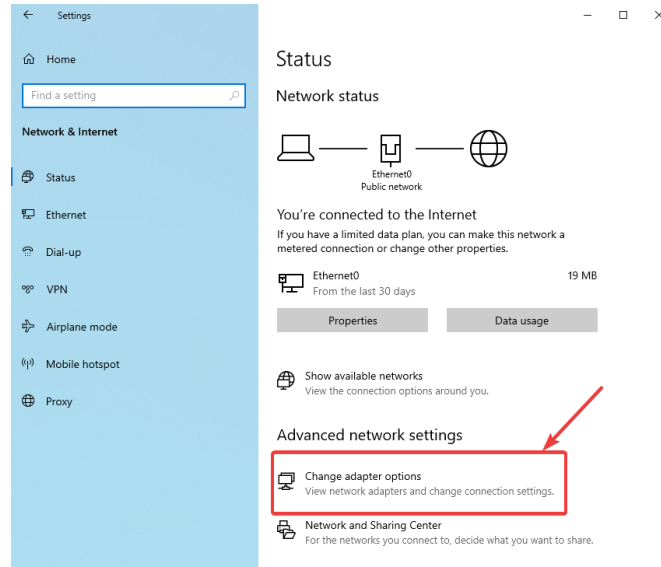
Bu, adaptörleri fiziksel bir düğme ile kapatmak veya bilgisayarı uçak moduna almakla her zaman aynı değildir. Aşağıdaki adımlar, adaptörlerin nasıl devre dışı bırakılacağı hakkında daha fazla ayrıntı sağlar.



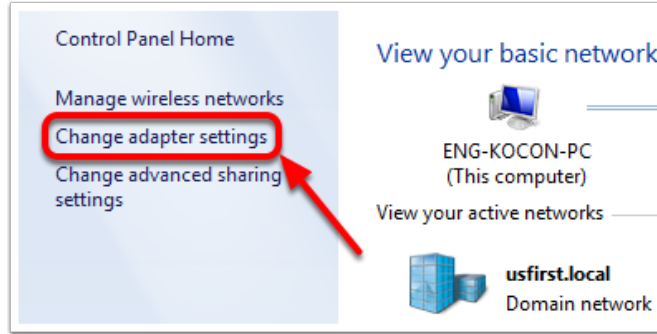
Ayarlar simgesine tıklayarak Ayarlar uygulamasını açın.



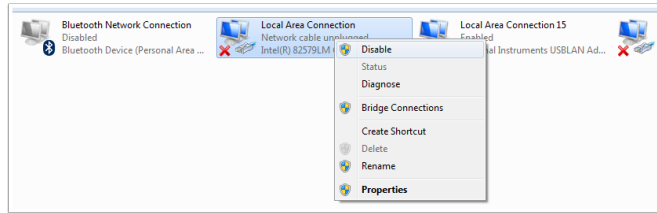
Network & Internet kategorisini seçin.



Şuna tıklayın *Change adapter options*.



Sol bölmede, tıklayın *Change Adapter Settings*.



Telsize bağlı olan dışındaki her bir adaptör için, adaptöre sağ tıklayın ve menüden *Disable* seçin.

41.3.7 Proxies-Vekil sunucular.

- Proxies. Etkin bir proxy'ye sahip olmak roboRIO ağ iletişimi ile ilgili sorunlara neden olabilir.

41.4 Windows Güvenlik Duvarı Yapılandırması

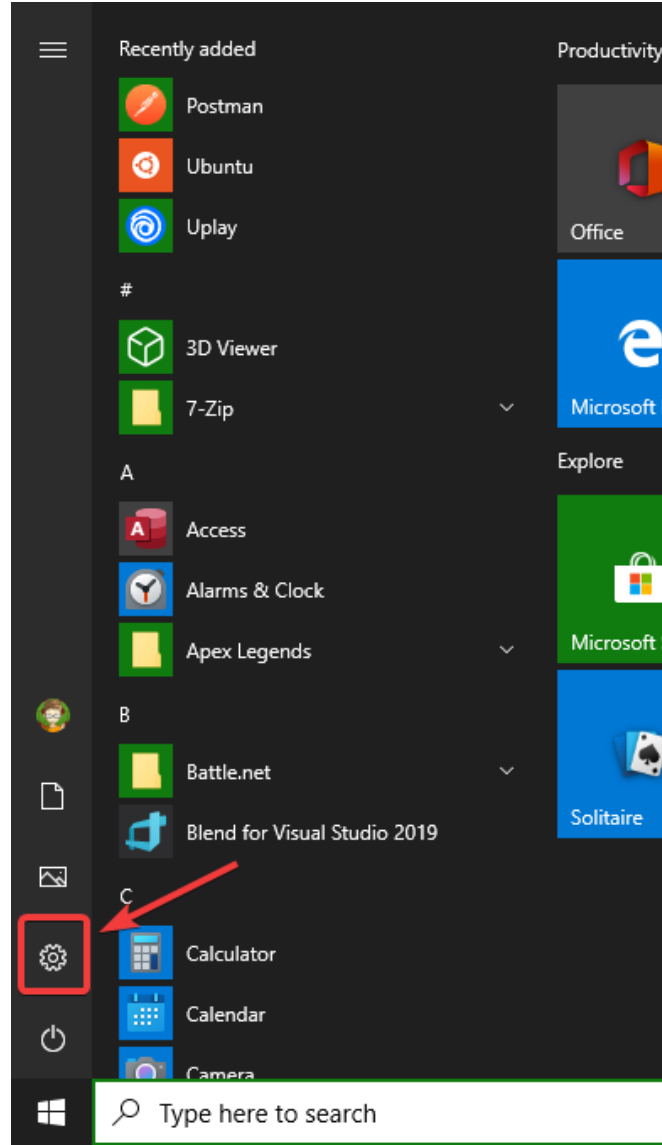
FRC ® 'de kullanılan programlama araçlarının çoğu çeşitli nedenlerle ağ erişimine ihtiyaç duyuyor. Tam yapılandırmaya bağlı olarak, Windows Güvenlik Duvarı bu programlardan biri veya daha fazlası için bu erişime potansiyel olarak müdahale edebilir.

41.4.1 Windows Güvenlik Duvarını Devre Dışı Bırakma

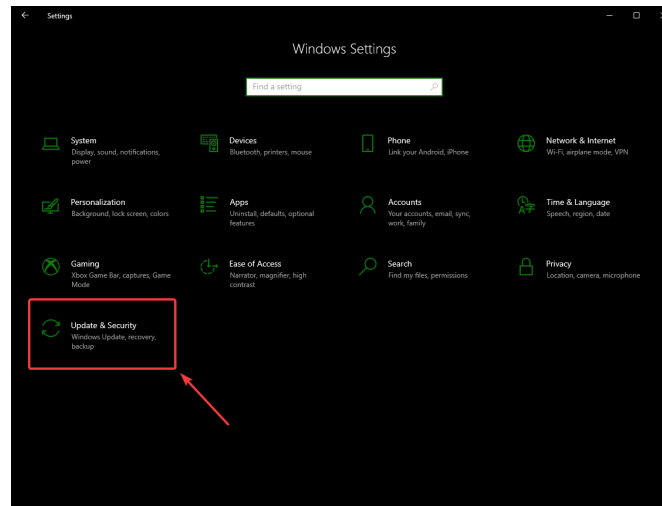
Önemli: Güvenlik duvarınızı devre dışı bırakmak, bilgisayarda yönetici ayrıcalıkları gerektirir. Ayrıca, internete bağlanan bilgisayarlar için güvenlik duvarının devre dışı bırakılmasının tavsiye edilmediğini unutmayın.

En kolay çözüm, Windows Güvenlik Duvarını devre dışı bırakmaktır. Ekipler, bunun, internete bağlanıldığında bilgisayarı kötü amaçlı yazılım saldırılarına karşı potansiyel olarak daha savunmasız hale getireceğine dikkat etmelidir.

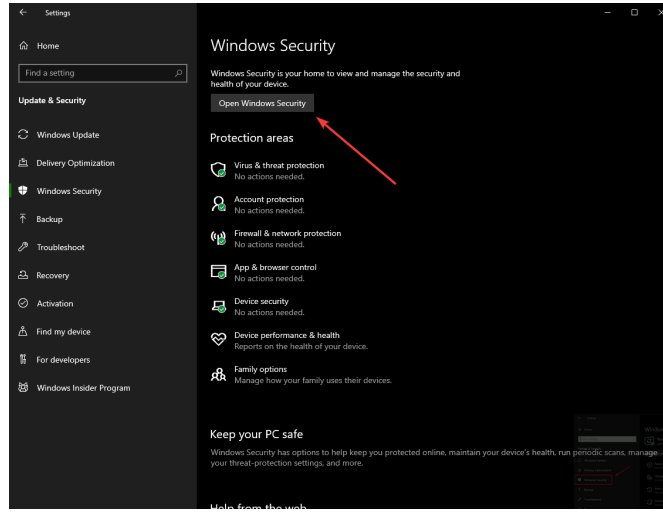
Click *Start* -> *Settings*



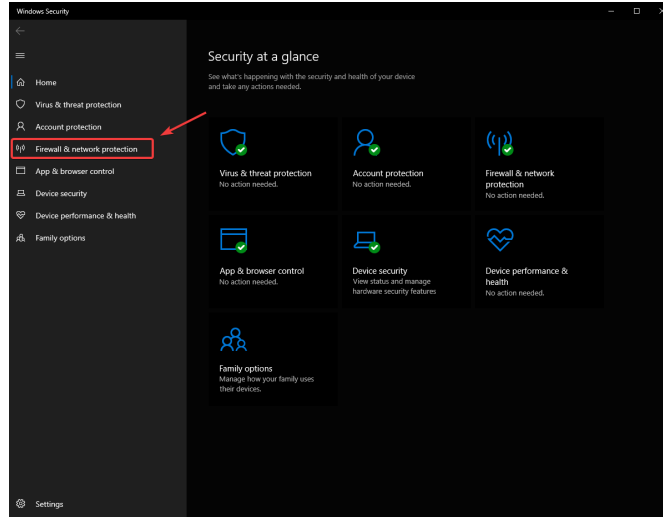
Update & Security tıklayınız



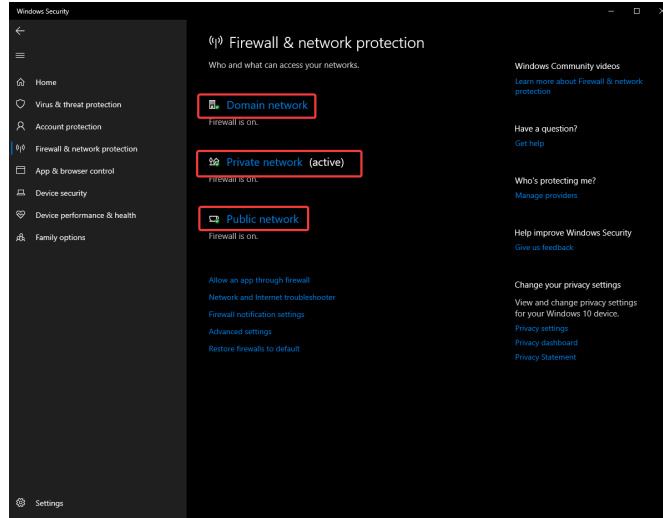
In the right pane, select *Open Windows Security*



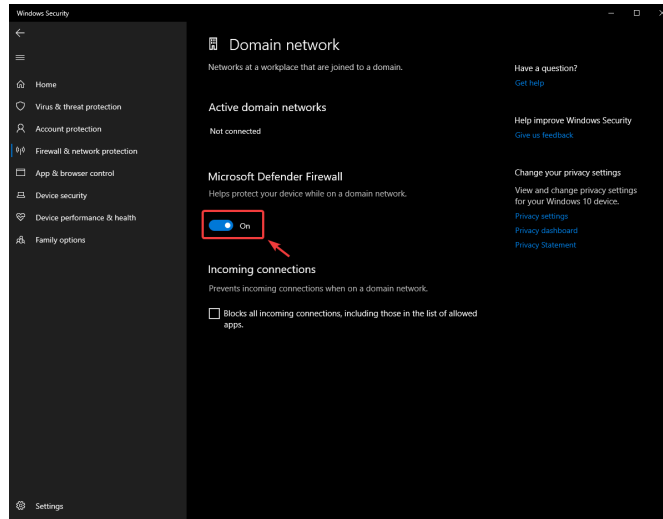
Sol bölmede, *Firewall and network protection* seçin



Vurgulanan seçeneklerden **her birine** tıklayın



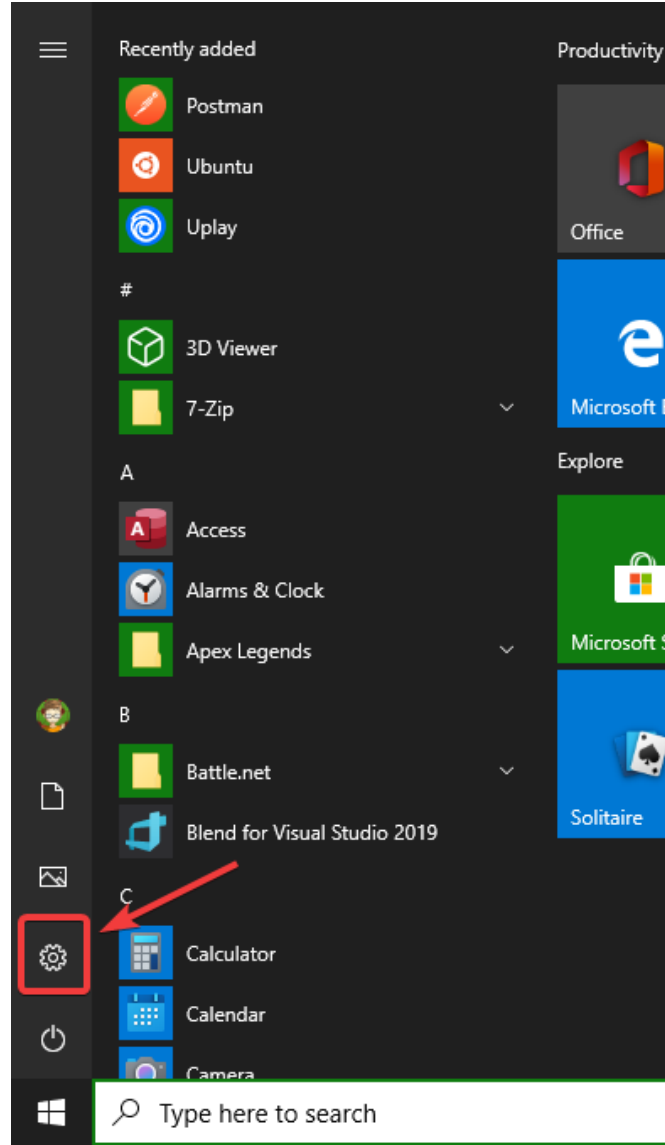
Ardından kapatmak için **On** düğmesine tıklayın.



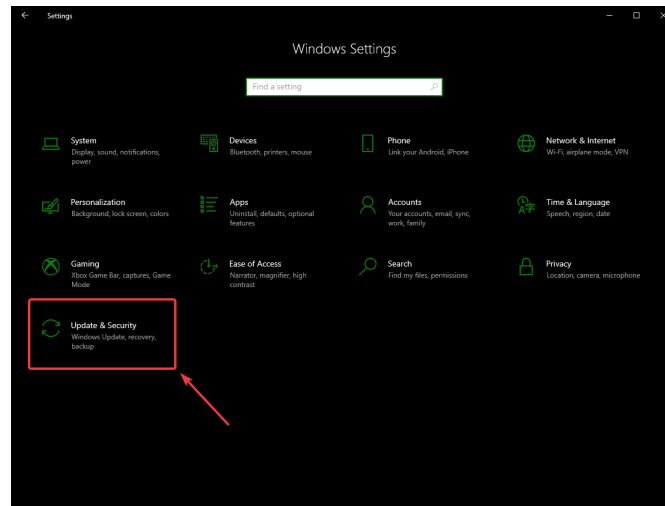
41.4.2 Beyaz Listeye Ekleyen Uygulamalar

Alternatif olarak, sorun yaşadığınız tüm FRC programları için Güvenlik Duvarına istisnalar ekleyebilirsiniz.

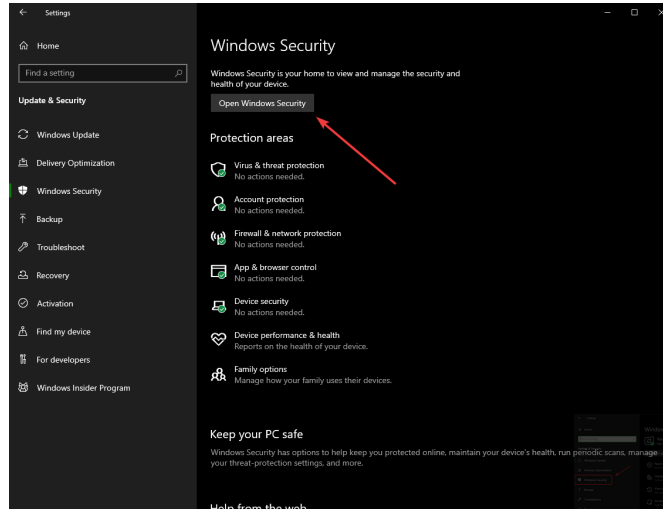
Click *Start* -> *Settings*



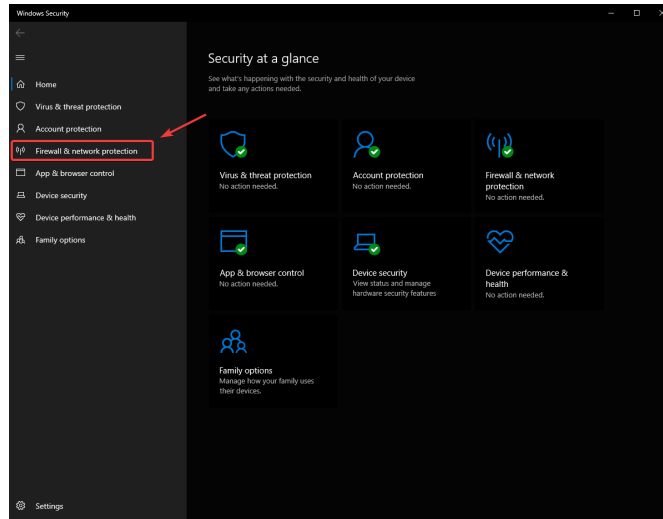
Update & Security tıklayınız



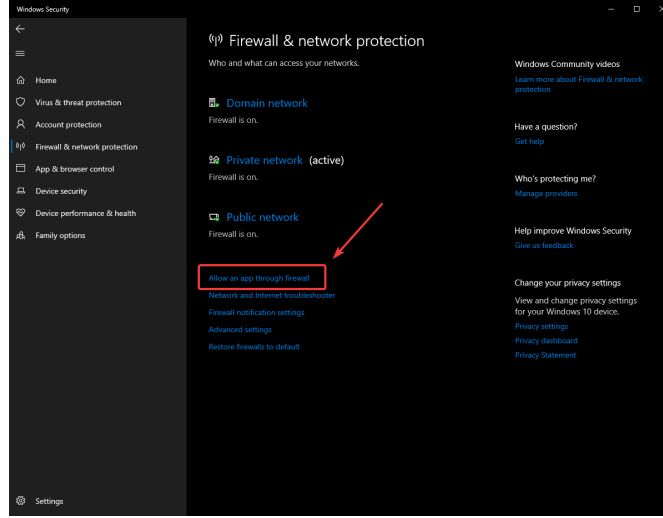
In the right pane, select *Open Windows Security*



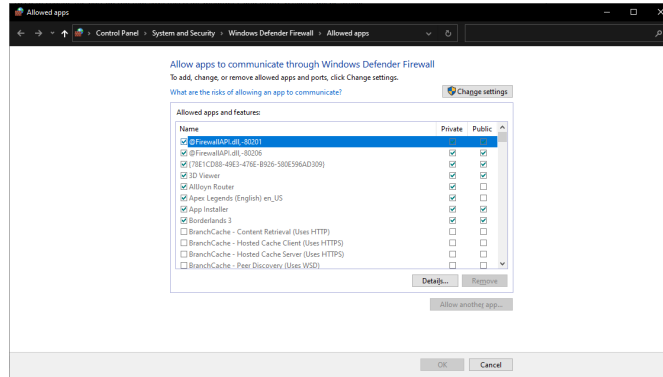
Sol bölmede, *Firewall and network protection* seçin



Pencerenin alt kısmında şunu seçin *Allow an app through firewall*



Sorun yaşadığınız her FRC yazılımı için, programın listede görüldüğünden ve 3 sütunun her birinde bir onayı bulunduğundan emin olun. Bir ayarı değiştirmeniz gerekirse, ayarları değiştirmeden önce sağ üstteki *Change settings* düğmesine tıklamanız gerekir. Program listede yoksa *Allow another program...* düğmesine tıklayın ve eklemek için programın konumuna göz atın.



41.5 Bant Genişliği Kullanımını Ölçme

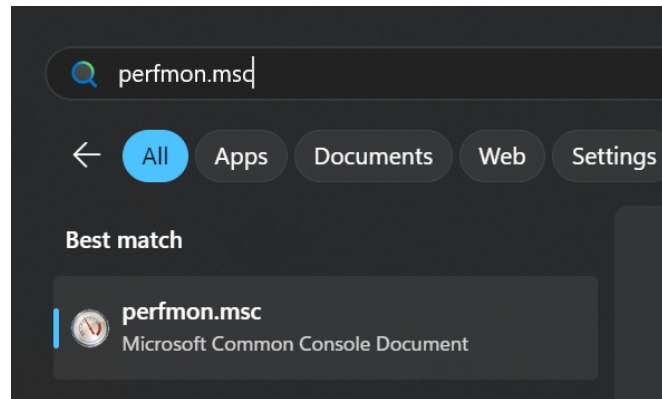
On the FRC® Field each team is allocated limited network bandwidth (see R704 in the 2024 manual). The [FMS Whitepaper](#) provides more information on determining the bandwidth usage of the Axis camera, but some teams may wish to measure their overall bandwidth consumption. This document details how to make that measurement.

Not: 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)

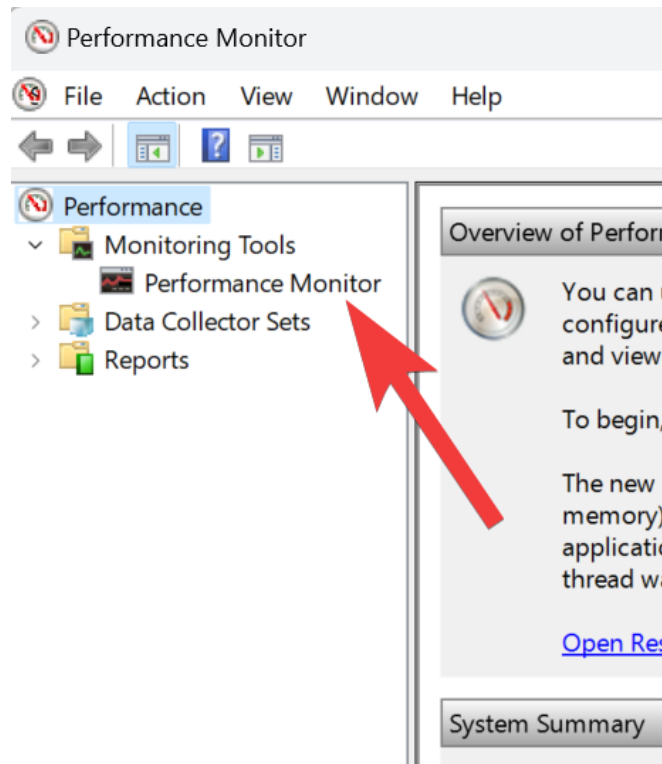
Windows contains a built-in tool called the Performance Monitor that can be used to monitor the bandwidth usage over a network interface.

Performance Monitor'ü Başlatma



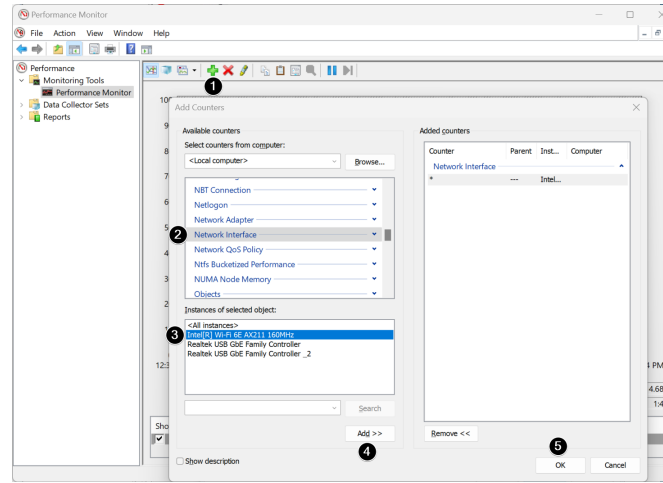
Open the Start menu and in the search box, type `perfmon.msc` and press Enter.

Real-Time- Gerçek Zamanlı Monitörü Aç



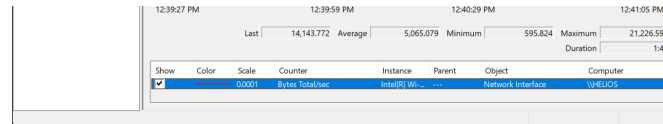
In the left pane, click *Performance Monitor* to display the real-time monitor.

Network Counter Ekle

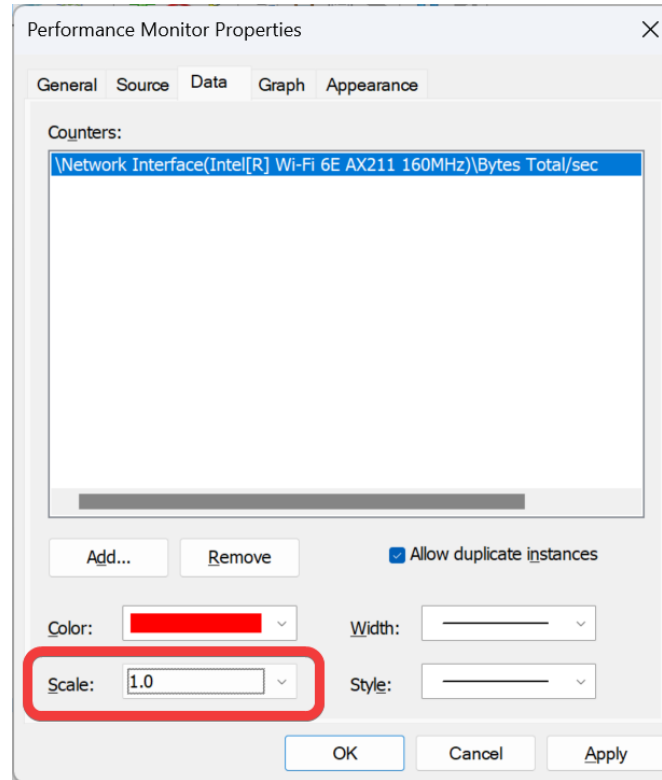


1. Sayaç eklemek için ekranın üst tarafındaki yeşil artı işaretini tıklayın
2. Sol üst bölmede, seçmek için Network Interface nü bulun ve tıklayın.
3. Sol alt bölmede, istenen ağ arayüzünü bulun (veya tüm arayüzleri izlemek için All instances kullanın)
4. Click *Add >>* to add the counter to the right pane.
5. Click *OK* to add the counters to the graph.

Ekstra Sayaçları Kaldırın

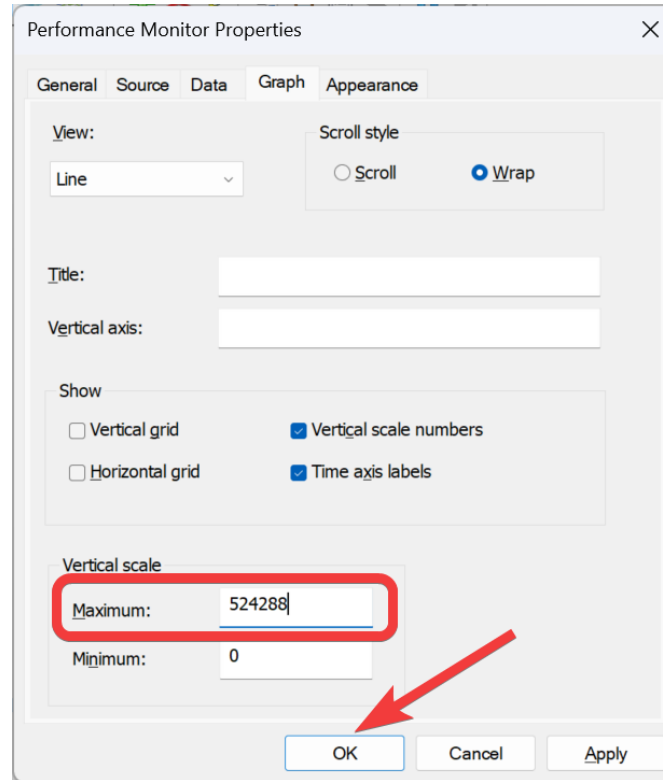


In the bottom pane, select each counter other than Bytes Total/sec and press the Delete key. The Bytes Total/sec entry should be the only entry remaining in the pane.

Data Properties-Veri Özelliklerini Yapılandırın

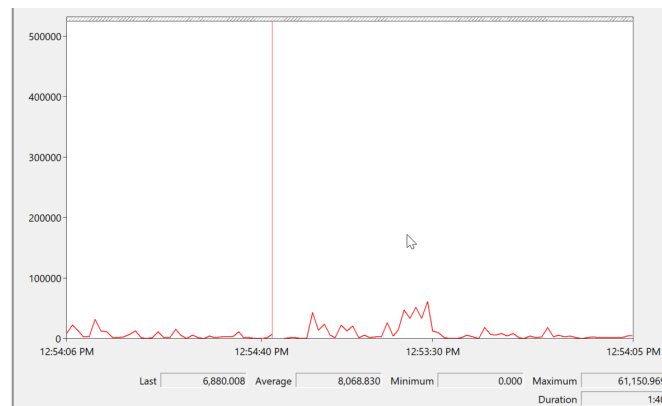
Press Ctrl+Q to bring up the Properties window. Click on the dropdown next to Scale and select 1.0. Then click on the *Graph* tab.

Grafik Özelliklerini Yapılandırın



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.

Bant Geniřlięi Kullanımını Görüntüleme

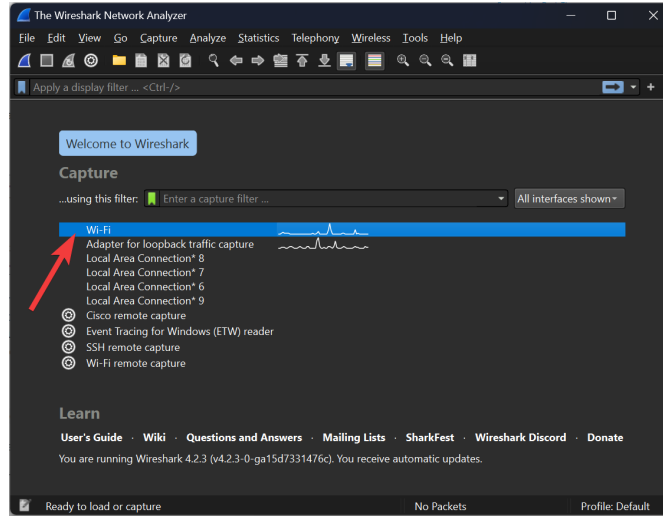


You may now connect to your robot as normal over the selected interface (if you haven't done so already). The graph will show the total bandwidth usage of the connection, with the bandwidth cap at the top of the graph. The Last, Average, Min and Max values are also displayed at the bottom of the graph. 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 Wireshark kullanarak Bant Genişliği Kullanımını Ölçme

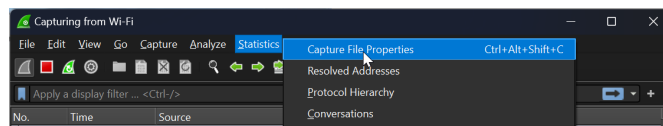
If you can not use performance monitor, you will need to install a 3rd party program to monitor bandwidth usage. One program that can be used for this purpose is Wireshark. Download and install the latest version of Wireshark for your version of Windows from [this page](#). After installation is complete, locate and open Wireshark. Connect your computer to your robot, open the Driver Station and any Dashboard or custom programs you may be using.

Arayüzü seçin ve yakalamayı başlatın



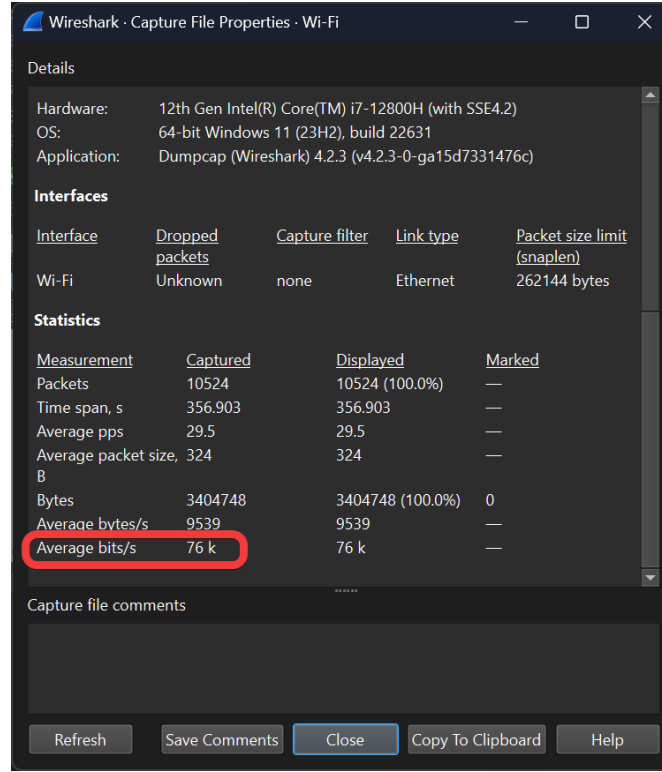
In the Wireshark program on the left side, double click the interface you are using to connect to the robot.

Open Capture File Properties



Let the capture run for at least 1 minute, then click *Statistics* then *Capture File Properties*.

Bant Geniřlięi Kullanımını Görüntüleyin



Average bandwidth usage, in bits/second is displayed near the bottom of the window.

41.6 OM5P-AC Radyo Modifikasyonu

OM5P-AC modem için amaçlanan kullanım durumu, onu FRC ® ‘de gördüğü ile çevreden gelen aynı şoklara ve kuvvetlere maruz bırakmaz. Modem kasanın altında önemli bir baskıya maruz kalırsa,modemin altındaki metal bir kalkanın, kartın altındaki bazı açıkta kalan metal kabloları kısa devre yaparak radyonun yeniden başlatılmasına neden olması mümkündür. Bu makale, bu senaryoyu önlemek için modemde yapılan bir deęişikliği ayrıntılarıyla anlatmaktadır.

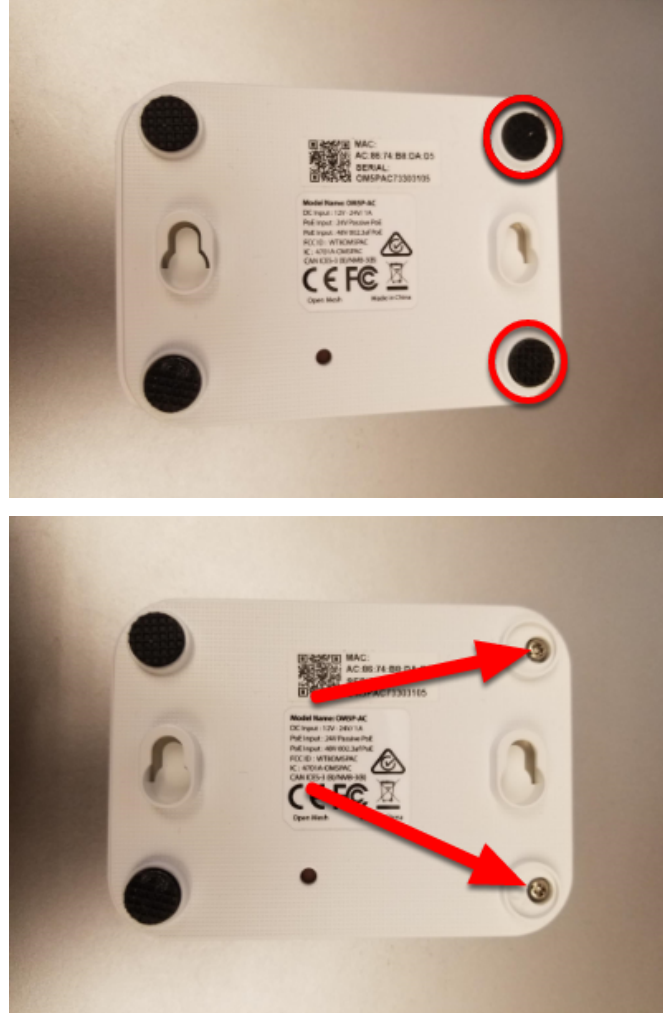
Uyarı: Bu şekilde yeniden başlatmaya neden olmak için kasanın altına önemli miktarda bir baskı uygulanması gerekir. Çoęu FRC radyo yeniden başlatma sorunu, bir şekilde güç kabloları yoluyla tespit edilebilir. Radyoyu açıp deęiřtirmek (ve hassas dahili bileřenlere zarar verme riski) yerine telsizin stratejik montajı yoluyla bu riski azaltmanızı öneririz:

- Telsizin altındaki “montaj baęlantı noktaları” özelliklerini kullanmaktan kaçının.
- Radyoyu biraz şok emilimine izin verecek şekilde monte etmek isteyebilirsiniz. Biraz uzun bir yol olarak, telsizi kanca veya plastik kelepçe kullanarak veya az miktarda esnek (plastik veya sac levha vb.) bir malzemeye robot yüzeyinde monte etmek, radyonun maruz kaldığı kuvvetleri önemli ölçüde azaltabilir.

41.6.1 Radyoyu Açmak

Not: OpenMesh OM5P-AC, kullanıcılar tarafından bakımı yapılabilen bir cihaz olarak tasarlanmamıştır. Kullanıcılar bu değişikliği riskleri kabul ederek gerçekleştirirler. Radyo anten kabloları gibi dahili bileşenlere zarar vermekten kaçınmak için yavaş ve dikkatli çalıştığınızdan emin olun.

Kasa Vidaları



Radyonun ön tarafındaki iki lastik ayağı bulun ve tırnak, küçük düz tornavida, vb. kullanarak radyodan ayırın. Küçük bir yıldız tornavida kullanarak, ayakların altındaki iki vidayı çıkarın.

Yan Mandallar



Radyonun kapağında, her bir uzun kenarın ortasına yakın küçük bir tırnak vardır (bu mandalları bir sonraki resimde daha net görebilirsiniz). Bir tırnak veya çok ince bir alet kullanarak, önden arkaya telsizin ortasına doğru kapak ve kasa arasındaki boşluk boyunca kaydırın, radyonun ortasına yaklaştığınızda küçük bir ses duymalısınız. Diğer tarafta da tekrarlayın (not: Bunu yaparken yanlışlıkla ilk taraf yeniden kilitlenebilir, devam etmeden önce her iki tarafın da açıldığından emin olun). Radyo kapağı şimdi yukarıdaki resimde gösterildiği gibi ön tarafta biraz açık olmalıdır.

Kapağı Kaldır

Uyarı: Isı emici pedler nedeniyle kart kapağa yapışmış olabilir. Panonun onunla gelip gelmediğini kontrol etmek için kapağı çıkarırken radyonun havalandırma deliklerinden bakın, eğer varsa, panoyu kapaktan ayırmak için küçük bir alet araya sokmanız gerekebilir. Küçük bir tornavida veya benzeri bir alet, ön köşesinden, vida deliğinin hemen yukarısına uygulanarak havalandırma deliklerinden geçebilir. Bu alanda panonun neye benzediğini görmek için kapağı kaldırılmış olarak gösterilen resme kaydırabilirsiniz.

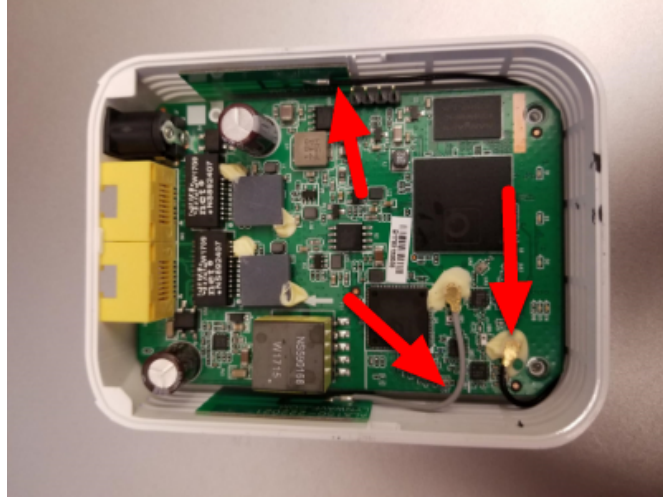


Kapağı çıkarmaya başlamak için, vida tutucular kasanın ön tarafına çarpana kadar öne doğru kaydırın (hafifçe kaldırarak) (bunu yaparken mandal bölgelerine baskı uygulamanız gerekebilir).

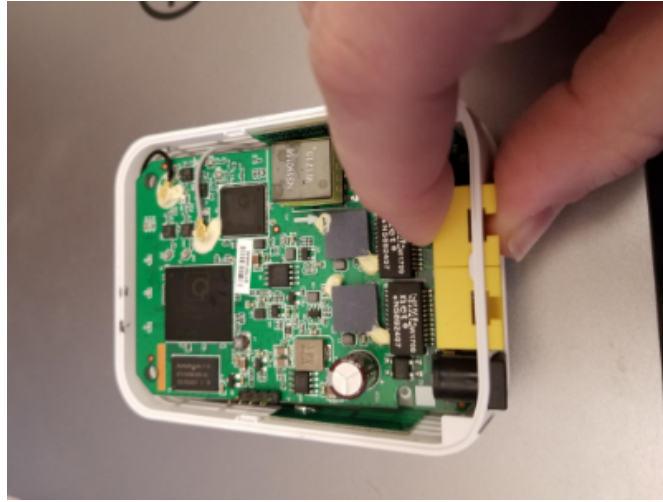


Ardından, kaldırmaya devam ederken gösterildiği gibi kapağı güç bağlantı noktası tarafından hafifçe çevirmeye başlayın. Bu, kapağı sağ üst köşede görünen küçük üçgenden çıkaracaktır. Sol üst köşedeki benzer bir özelliği açmak için sol üst köşeyi güç bağlantı noktasına doğru iterken bu yönde hafifçe dönmeye devam edin (bu adımda daha fazla kaldırmaya çalışmayın). Ardından kapağı tamamen gövdeden kaldırın.

Panoyu Kaldır



Uyarı: Yukarıdaki resimde gösterilen anten kablolarına dikkat edin. Bu teller ve konektörleri kırılındır, sonraki adımları gerçekleştirirken onlara zarar vermemeye özen gösterin.





Kartı çıkarmak için, ağ bağlantı noktalarından birini veya her ikisini parmaklarınızla tutmanızı (gösterildiği gibi) ve ağ bağlantı noktaları ve güç jakı kasadan kurtulana kadar içe (radyonun önüne doğru) ve yukarı doğru itmenizi öneririz.



Altındaki metal korumayı ortaya çıkarmak için kartı yukarı (kısa gri anten kablosuna doğru) yatırın.

Not: Bu adımı gerçekleştirdiğinizde, kartın alt tarafında kasadaki delikten daha büyük reset düğmesi olduğunu fark edebilirsiniz. FRC aygıt yazılımı kurulu haldeyken sıfırlama düğmesine basmanın hiçbir etkisi olmadığını ve radyo kasasını delmenin izin verilen bir değişiklik olmadığını unutmayın.

41.6.2 Bant Uygula



Ağ bağlantı noktası / güç bağlantı jackı açıklıklarının hemen içindeki alandaki metal korumaya bir parça elektrik bandı uygulayın. Bu, kartın altındaki açıkta kalan uçların bu plakada kısa devre yapmasını önleyecektir.

41.6.3 Radyoyu Yeniden Birleştirin

Açmak için talimatları tersine çevirerek radyoyu yeniden monte edin:

- Ön tarafa yakın vida delikleriyle hizalandığından ve güvenli bir şekilde oturduğundan emin olarak kartı geri yerleştirin
- Kapağı sağdan sola hareket ettirerek arka sol tutma tırnağına kaydırın. Bu alandaki kondansatöre dikkat edin
- Kapağı döndürün, aşağı doğru bastırın ve sağ arka tutma tırnağına oturtun
- Mandalları oturtmak için kapağın önüne / ortasına sıkıca bastırın
- Ön ayaklardaki 2 vidayı yerleştirin
- Ön ayakları yerleştirin

42.1 Port Yönlendirme

Bu bölüm, yerel bağlantı noktalarını başka bir bilgisayara/ bağlantı noktasına iletmenin kolay bir yolunu sağlar. Bu, roboRIO USB bağlantı noktasına bağlı bir bilgisayardan Ethernet bağlantılı cihazlara erişmenin bir yolunu sağlamak için kullanışlıdır. Bu sınıf, hem bir TCP bağlantı noktası ileticisi olarak işlev görür; bu, SSH gibi bağlantıları iletebileceğiniz anlamına gelir.

42.1.1 Uzak Bağlantı Noktası İletme

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)
```

Önemli: Yerel yönlendirilen bağlantı noktası olarak 1024'ten daha düşük bir bağlantı noktası **** can not-kullanamazsınız . Tam URL'leri ** can not-kullanamazsınız** (<http://wpilibpi.local>) ve yalnızca IP Adreslerini veya DNS adlarını kullanmanız gerektiğine de dikkat etmek önemlidir.

42.1.2 Yönlendirilen Bağlantı Noktasını kaldırma

Belirtilen bir bağlantı noktasında forward-yönlendirmeyi durdurmak için, bağlantı basitçe `remove(int port)` çağrısı yapmanız yeterlidir. Yönlendirilmemiş bir bağlantı noktasında `remove()` çağırırsanız, hiçbir şey olmaz.

JAVA

```
@Override
public void robotInit() {
    PortForwarder.remove(8888);
}
```

C++

```
void Robot::RobotInit {
    wpi::PortForwarder::GetInstance().Remove(8888);
}
```

PYTHON

```
wpinet.PortForwarder.getInstance().remove(8888)
```


frc-docs'a Katkıda Bulunmak

43.1 Katkı Yönergeleri

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](#).

Önemli: *FIRST*® sağlanan belgelere ve görüntülere ilişkin tüm hakları saklı tutar. Makaleler/güncellemeler için kredi [GitHub commit history](#). de olacaktır.

43.1.1 Görev Tanımı

WPILib Misyonu, *FIRST* Robotik ekiplerinin donanım detaylarına odaklanmak yerine oyuna özel yazılım yazmaya odaklanmasını sağlamaktır - "tavanı alçaltmayın, zemini yükseltin". Sınırlı programlama bilgisine ve/veya mentor deneyimine sahip ekiplerin, daha gelişmiş programlama yeteneklerine sahip ekiplerin yeteneklerini engellemeden olabildiğince başarılı olmasını sağlamak için çalışıyoruz. Kit of Parts kontrol sistemi bileşenlerini doğrudan kütüphane ile destekliyoruz. Ekiplerin belirli bir programlama dili seçerken dezavantajlı durumda kalmaması için her dilin (Java, C ++ ve NI's LabVIEW) ana özellikleri arasında eşitlik sağlamaya da çalışıyoruz.

Bu belgeler, tüm *FIRST* Robotics Competition ekipleri için bir öğrenme alanı sağlamaya hizmet eder. Projeye yapılacak katkılar bu temel ilkeleri takip etmelidir.

- Topluluk liderliğindeki belgeler. Belge kaynakları halka açık olarak barındırılır ve topluluk katkı sağlayabilir
- Yapılandırılmış, iyi biçimlendirilmiş, temiz belgeler. Dokümantasyon hem kaynak hem de yayın açısından temiz ve okunması kolay olmalıdır.
- İlgi. Belgeler, *FIRST* Robotics Competition'a odaklanmalıdır.

Belgelerinizin stiliyle ilgili bilgi için lütfen :ref:`docs/contributing/frc-docs/style-guide:Style Guide`'a bakın.

43.1.2 Yayınlama İşlemi

frc-docs, ana sayfa /stable/ ve geliştirme sayfası /latest/ nı yönetmek için özel bir yayınlama işlemi kullanır. Bu akış aşağıda detaylandırılmıştır.

During Season:

- Commit made to main branch
 - /stable/ ve /latest/ websitesinde güncellendi

End of Season:

- Repository is tagged with year, for archival purposes

Sezon Dışı :

- stable branch is locked to the last on-season commit
- Commit made to main branch
 - Sadece dokümantasyon sitesinde /latest/ güncellenir

43.1.3 PR yaratma

PRs should be made to the [frc-docs](#) repo on GitHub. They should point to the main branch and *not* stable.

43.1.4 Yeni İçerik Yaratma

frc-docs <<https://github.com/wpilibsuite/frc-docs>> __projesine katkıda bulunduğunuz için teşekkür ederiz! Başlamadan önce bilmeniz gereken birkaç şey var!

Makaleler nereye yerleştirilir?

Yeni makalelerin konumu oldukça tartışılabilir bir konu olabilir. Halihazırda bir konu kategorisine giren bağımsız makaleler, belirtilen konu kategorisine yerleştirilmelidir (simülasyonla ilgili belgeler simülasyon bölümüne yerleştirilmelidir). Bununla birlikte, bir makale mevcut iki ayrı bölümü birleştirdiğinde veya bunlara atıfta bulunduğunda işler oldukça karmaşık hale gelebilir. Bu durumda, PR'ı açmadan önce tartışmayı başlatmak için yazara havuzda bir konuyu tema seçmesini tavsiye ederiz.

Not: Tüm yeni makaleler arşivle birleştirilmeden önce bir inceleme sürecinden geçecektir. Bu inceleme süreci WPILib ekibinin üyeleri tarafından yapılacaktır. Yeni Makaleler resmi *FIRST* destekli Yazılım ve Donanım üzerinde olmalıdır. Resmi olmayan kitaplıklar veya sensörler ile ilgili belgeler *kabul edilmeyecektir*. Bu incelemenin tamamlanması biraz zaman alabilir, lütfen sabırlı olun.

Bölümler nereye yerleştirilir?

Bölümler, büyük miktarda içerik barındırdıkları için oldukça zordur. Yazara, bir PR açmadan önce tartışma toplamak için <https://github.com/wpilibsuite/frc-docs/issues> __ sayısını açmasını tavsiye ederiz.

Diğer Makaleleri Bağlamak

Makalenin başka bir makalede açıklanan içeriğe atıfta bulunması durumunda, yazar ilk referansta o makaleye bağlantı vermek için elinden geleni yapmalıdır.

Bir aktarma organı rehberinde aşağıdaki içeriğe sahip olduğumuzu hayal edin:

```
Teams may often need to test their robot code outside of a competition.
↪:ref:`Simulation <link-to-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.
```

Simulation'ın yalnızca ilk örneğinin nasıl bağlantılı olduğuna dikkat edin. Yazarın izlemesi gereken yapı budur. Bağlantılı bir makalenin farklı içerik konularına sahip olduğu durumlar vardır. Makalede farklı içerik türlerine atıfta bulunuyorsanız, her yeni referansa bir kez bağlantı vermelisiniz (yazarın aksini uygun gördüğü durumlar hariç).

43.2 Biçim Rehberi

Bu belge, frc-docs projesi için çeşitli RST/Sphinx'e özgü yönergeleri içerir. Çeşitli WPILib kod projeleriyle ilgili yönergeler için bkz. [WPILib GitHub](#)

43.2.1 Dosya İsimleri

Yalnızca küçük alfanümerik karakterler ve - (eksi) sembolünü kullanın.

Yazılım / Donanım ile ilgili belgeler için, belge adının sonuna "Hardware" veya "Software" ekleyin. Örneğin, ultrasonik donanım.rst

.Rst uzantısına sahip suffix dosya adları.

Not: 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 Metin

All text content should be on the same line. If you need readability, use the word-wrap function of your editor.

Bu terimler için aşağıdaki örnekleri kullanın:

- roboRIO (RoboRIO, roboRio yada RoboRio değil)
- LabVIEW (labview yada LabView değil)
- Visual Studio Code (VS Code) (vscode, VScode, vs code, v.b. değil)
- macOS (Mac OS, Mac OSX, Mac OS X, Mac, Mac OS, v.b. değil)
- GitHub (github, Github vb. Değil)
- PowerShell (powershell, Powershell vb. Değil)
- Linux (linux değil)
- Java (java değil)

İngilizce metin için ASCII karakter kümesini kullanın. Özel karakterler için (örneğin, Yunan sembolleri) *standart karakter kümelerini* [_kullanın](https://docutils.sourceforge.io/docs/ref/rst/definitions.html#character-entity-sets).

Bağımsız denklemler için `.. math ::` ve satır içi denklemler için `: math:` kullanın. Yararlı bir LaTeX denklem kısayol sayfası [_bu adreste bulunabilir](https://www.reed.edu/academic_support/pdfs/qskills/latexcheatsheet.pdf).

Dosya adları, fonsiyon ve değişken adları için genel geçer değerleri kullanın.

Tescilli ticari markaların kullanımı *FIRST*® ve FRC|reg| politikasını [_bu sayfadan](#) takip etmelisiniz. Spesifik olarak, mümkün olduğunda (yani başka bir işaretlemenin içine veya bir belge başlığına yerleştirilmemişse), ticari markaların ilk kullanımında ® sembolü ve tüm *FIRST* yazı örnekleri italik yazılmalıdır. ® sembolü, belgenin üst kısmındaki `..include::<isonum.txt>` kullanılarak ve ardından `*FIRST*\| reg |` veya `FRC\| reg|` kullanılarak eklenebilir.

Commonly used terms should be added to the *FRC Sözlüğü*. You can reference items in the glossary by using `:term:`deprecated``.

43.2.3 Whitespace - Beyazboşluk

Girinti

Yeni bir içerik bloğu *oluşturmadığınız sürece* girinti *her zaman* önceki girinti düzeyiyle eşleşmelidir.

İçerik direktiflerinin yeni satırının `..toctree::` olarak girintisi 3 boşluk olmalıdır.

Boş Satırlar

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.

İç Beyazboşluk

Cümleler arasında bir boşluk kullanınız.

43.2.4 Başlıklar

Başlıklar aşağıdaki yapıda olmalıdır. Başlık alt çizgileri, başlığın kendisiyle aynı sayıda karakterle eşleşmelidir.

1. Belge başlıkları için =. Bunu makale başına * birden fazla * *kullanmayın*.
2. - doküman bölümleri için
3. ^ alt doküman bölümleri için
4. ~ Doküman alt-alt bölümleri için
5. Daha düşük seviyelerde bir yapı kullanmanız gerekirse, bir şeyleri yanlış yapıyorsunuz demektir.

Ana başlıklar için başlığı kullanın.

43.2.5 Listeler

Listelerin her girinti düzeyi arasında yeni bir satır olması gerekir. En yüksek girinti 0 girintiye sahip olmalı ve sonraki alt listeler, önceki girintinin ilk karakterinden başlayan bir girintiye sahip olmalıdır.

- Block one
- Block two
- Block three
 - Sub 1
 - Sub 2
- Block four

43.2.6 Kod Blokları

Tüm kod bloklarının belirtilmiş bir dili olmalıdır.

1. İstisna: Biçimlendirmenin korunması gereken ve dili olmayan içerik. Bunun yerine metin kullanın.

C++ ve Java örnek kodu için [WPILib biçim kılavuzunu](#) takip ediniz. Örneğin, C++ ve Java'da girinti için iki boşluk kullanın.

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 Tavsiyeler

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.

Kullanın

```
.. warning:: This is a warning!
```

Yanlış

```
.. warning::
    This is a warning!
```

43.2.10 Linkler

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 *Derleme Yapısı* 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!
```

Bununla birlikte, aynı bağlantıya birden çok kez başvurulması gereken bazı durumlarda, aşağıdaki sözdizimi kabul edilir.

```
Hi there, `this is a link`_ and it's pretty cool!
```

```
.. _this is a link: https://example.com
```

43.2.11 Resimler

Görseller, içerik ve yönergeyi ayıran 1 yeni satırla oluşturulmalıdır.

Tüm görseller (vektörler dahil) boyut olarak 500 kilobyttan daha küçük olmalıdır. Lütfen daha küçük bir çözünürlük ve daha verimli sıkıştırma algoritmaları kullanın.

```
.. image:: images/my-article/my-image.png
:alt: Always add alt text here describing the image.
```

Resim dosyaları

Image files should be stored in the document directory, sub-directory of document-name/images.

Resim adları, görüntünün ne gösterdiğinin kısa bir açıklaması olduğu short-description.png adlandırma şemasına uymalıdır. Bu, 24 karakterden az olmalıdır.

They should be of the .png or .jpg image extension. .gif is unacceptable due to storage and accessibility concerns.

Not: Erişilebilirlik önemlidir! Resimler bir `:alt:` direktifiyle işaretlenmelidir.

```
.. image:: images/my-document/my-image.png
:alt: An example image
```

Vectorel Resimler

SVG dosyaları `svg2pdfconverter` Sphinx uzantısı ile desteklenir.

Bunları başka herhangi bir resim dosyasında yaptığınız gibi kullanın.

Not: Vektördeki gömülü görüntülerin vektörü 500KB sınırını aşacak şekilde şişirmediğinden emin olun.

```
.. image:: images/my-document/my-image.svg
   :alt: Always add alt text here describing the image.
```

Draw.io Diyagramları

Draw.io (aynı zamanda diagrams.net olarak da bilinir) diyagramları, gömülü `.drawio` meta verisine sahip `svg` dosyalarıyla desteklenir ve `svg` dosyası, diyagramların kaynak dosyası olarak işlev görür, normal bir vektör grafik dosyası gibi işlenir.

Bunları başka herhangi bir vektör görüntüsü veya başka bir görüntü gibi kullanın.

```
.. image:: diagrams/my-document/diagram-1.drawio.svg
   :alt: Always add alt text here describing the image.
```

Draw.io Dosyaları

Draw.io dosyaları, normal resimler ile neredeyse aynı adlandırma düzenini izler. Gömülü `.drawio` meta verisine sahip dosyaları takip etmek için, dosya adının sonuna, uzantının önüne bir `.drawio` ekleyin, yani dosyanın adı `document-title-1.drawio.svg` vb olmalıdır. Ek olarak, diyagramlar belge dizininde `diagrams` adlı bir alt klasörde saklanmalıdır.

Bir diyagramı meta verilerle `.svg` olarak kaydetmenin ayrıntıları için, buraya bakınız [Draw.io Kaydetme Talimatları](#).

Uyarı: Draw.io dışında herhangi bir programda `diagrams` klasöründe bulunan veya `.drawio.svg` ile biten herhangi bir dosyayı değiştirmedenizden emin olun, aksi takdirde dosyanın meta verilerini kırma riskiniz olabilir. , bu dosyayı düzenlenemez hale getirebilir.

43.2.12 Dosya uzantıları

Dosya uzantıları kod biçimlendirmesi kullanılmalıdır. Örneğin, şunu kullanın:

```
``.png``
```

Onun yerine

```
.png
".png"
"``.png``"
```

43.2.13 İçindekiler (TOC-Table of Contents)

Her kategori bir `index.rst` içermelidir. Bu dizin dosyası, `maxdepth` maksimum derinliği 1 içermelidir. Alt kategoriler `maxdepth 1` ile kabul edilebilir.

The category `index.rst` file can then be added to the root index file located at `source/index.rst`.

43.2.14 Örnekler

```
Title
====
This is an example article

.. code-block:: java

    System.out.println("Hello World");

Section
-----
This is a section!
```

43.2.15 Önemli Not!

Bu liste kapsamlı değildir ve yöneticiler değişiklik yapma hakkını saklı tutar. Değişiklikler bu belgeye yansıtılacaktır.

43.3 Derleme Yapısı

Bu belge, `frc-docs` sitesinin HTML, PDF ve EPUB sürümlerinin nasıl oluşturulacağı hakkında bilgiler içerir. `frc-docs`, dokümantasyon oluşturucu olarak Sphinx'i kullanır. Bu belge ayrıca `Git <https://git-scm.com/>` __ ve konsol komutları hakkında temel bilgiye sahip olduğunuzu varsayar.

43.3.1 Gereksinimler

`Git` 'in kurulu olduğundan ve `frc-docs` deposunun `git clone https://github.com/wpi/libsuite/frc-docs.git` kullanılarak klonlandığından emin olun. ``.

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.

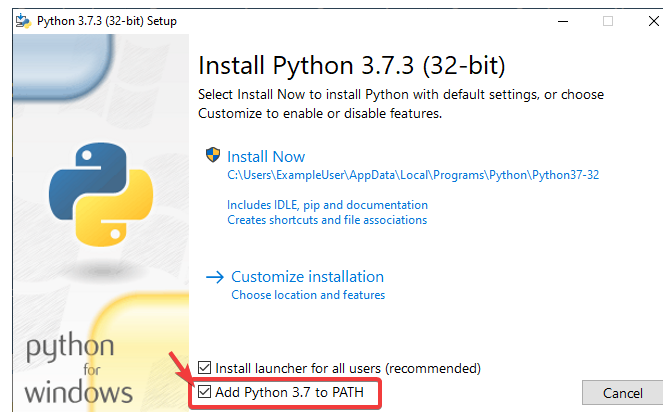
```
"restructuredtext.linter.doc8.extraArgs": [
  "--ignore D001"
]
```

Windows

Not: 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)

Python'u yüklerken **** Add Python to PATH**** seçimini yaparak sistem PATH 'inize eklendiğinden emin olun.



Python kurulduktan sonra Powershell'i açın. Ardından frc-docs dizinine gidin. Aşağıdaki komutu çalıştırın: `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 Derleme

Bir Powershell Penceresi veya terminali açın ve klonlanan frc-docs dizinine gidin.

```
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

Not: Lint Check, satır sonlarıyla ilgili bir hata nedeniyle Windows'ta satır sonlarını kontrol etmeyecektir. Daha fazla bilgi için `bu soruna <<https://bugs.launchpad.net/doc8/+bug/1756704>>`__bakın.

Linter ile yaptığınız değişiklikleri kontrol etmeniz önerilir. Bu başarılı olmazsa buildbot'ta başarısız **olacaktır**. Kontrol etmek için . \ Make lint komutunu çalıştırın.

Link Check - Bağlantı Denetleme

Bağlantı denetleyicisi, belgelerdeki tüm bağlantıların çözümlendiğinden emin olur. Bu başarılı olmazsa buildbot'ta başarısız **olacaktır**. Kontrol etmek için . \ Make linkcheck komutunu çalıştırın.

Resim boyutu denetimi

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.

Yönlendirme Kontrolü

Taşınan veya yeniden adlandırılan dosyaların yeni konumları (veya 404 ile değiştirilmiş) `` kaynak `` içindeki `` redirects.txt `` dosyasında bulunmalıdır.

Yönlendirme yazıcısı, yeniden adlandırılmış / taşınmış dosyaları yeniden yönlendirmeler dosyasına otomatik olarak ekleyecektir. Çalıştır `.\make rediraffewritediff`.

Not: bir dosya hem taşınırsa hem de önemli ölçüde değiştirilirse, yeniden yönlendirme yazıcısı onu `` redirects.txt `` dosyasına eklemeyecek ve redirects.txt dosyasının manuel olarak güncellenmesi gerekecektir.

Yeniden yönlendirme denetleyicisi, tüm dosyalar için geçerli yeniden yönlendirmeler olduğundan emin olur. Bu ******, başarılı olmazsa buildbot'ta başarısız olur ******. Kontrol etmek için, tüm dosyaların yeniden yönlendirildiğini doğrulamak için `.\make rediraffecheckdiff` komutunu çalıştırın. Ek olarak, tüm dosyaların düzgün bir şekilde yeniden yönlendirilmesini sağlamak için bir HTML yapısının çalıştırılması gerekebilir.

HTML Derleme

HTML içeriği oluşturmak için `.\make html` komutunu yazın. İçerik depo kök dizinindeki `build/html` dizininde bulunur.

43.3.3 PDF Oluşturma

Uyarı: Windows üzerinde PDF derlemesinin SVG içeriği için bozuk görüntülere neden olabileceğini lütfen unutmayın. Bunun nedeni, Windows'ta librsvg2-bin desteğinin olmasıdır.

PDF içeriği oluşturmak için `.\make Latekspdf` komutunu yazın. PDF, arşivin kök dizinindeki `build/latex` dizininde bulunur.

43.3.4 EPUB Derlemek

EPUB içeriği oluşturmak için `.\ Make epub` komutunu yazın. EPUB, deponun kök dizinindeki `build/epub` dizininde bulunur.

43.3.5 Python 3. Parti Kütüphaneleri ekleme

Önemli: After modifying `frc-docs` dependencies in any way, `requirements.txt` must be regenerated by running `poetry export -f requirements.txt --output source/requirements.txt --without-hashes` from the root of the repo.

`frc-docs` uses [Poetry](#) to manage its dependencies to make sure builds are reproducible.

Not: Poetry is **not** required to build and contribute to `frc-docs` content. It is *only* used for dependency management.

Poetry Kurulumu

Ensure that Poetry is installed. Run the following command: `pip install poetry`.

Adding a Dependency

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`.

Updating a Top-Level Dependency

Update the dependency's version in the `[tool.poetry.dependencies]` section of `pyproject.toml`. Then, run the following command: `poetry lock --no-update`.

Updating Hidden Dependencies

Run the following command: `poetry lock`.

43.4 Draw.io Kaydetme Talimatları

Uyarı: Make sure you don't modify any file that is in a diagrams folder, or ends in .drawio.svg in any program other than draw.io; otherwise you might risk breaking the metadata of the file, making it uneditable.

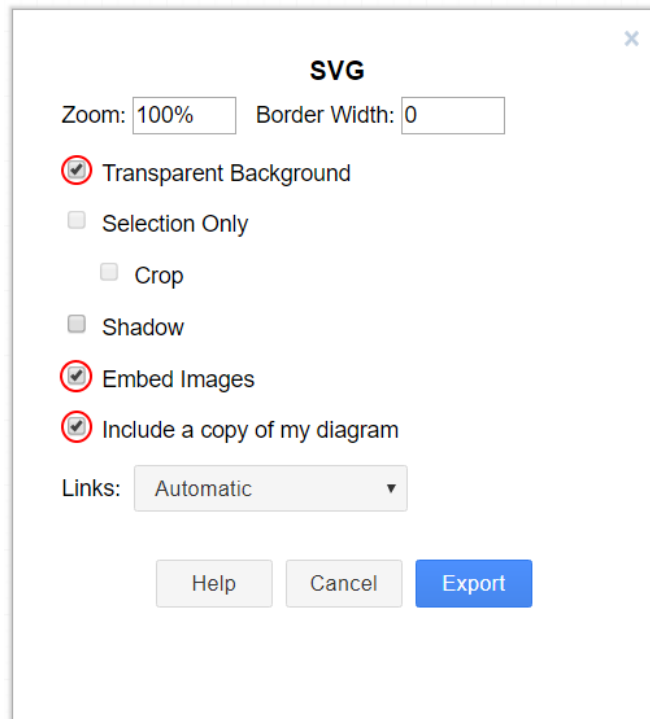
Draw.io (ayrıca diagrams.net olarak da bilinir), draw.io kaynak dosyası için katıştırılmış XML meta verileriyle birlikte svg dosyaları olarak kaydedildiğinde desteklenir (normalde .drawio olarak saklanır). Bu, bu görüntülerin hem ileride düzenlenebilecek diyagramlar için kaynak dosyaları olmasına hem de normal svg dosyaları olarak işlenmesine olanak tanır.

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.

Bu yöntem hem draw.io masaüstü hem de diagrams.net adresindeki web sürümü için geçerlidir.

Dışa aktarmak için File - Export as - SVG... seçeneğine gidin. Diyagram meta verilerini gömmek için Include a copy of my diagram seçeneğinin etkinleştirildiğinden ve Embed Images seçeneğinin etkinleştirildiğinden emin olun, böylece diyagramdaki resim dosyaları belgelerde işlenecek şekilde gömülür. Ek olarak, arka planın doğru görüntülendiğinden emin olmak için Transparent Background seçeneğini işaretleyin.

Dışa aktarma - export menüsü aşağıdaki gibi görünmelidir:



Ardından Export ı tıklayın, ardından dosyayı nereye kaydetmek istediğinizi seçin ve kaydedin.

Not: Kaydederken, şu adresteki stil kılavuzunu takip ettiğinizden emin olun [Draw.io Dosyaları](#)

43.5 Çeviriler

frc-docs, web tabanlı [Transifex](#) yardımcı programını kullanarak çevirileri destekler. frc-docs İspanyolca - Meksika (es_MX), Fransızca - Kanada (fr_CA) ve Türkçe - Türkiye'ye (tr_TR) çevrilmiştir. Çince - Çin (zh_CN), İbranice İsrail (he_IL) ve Portekizce - Brezilya (pt_BR) çevirileri devam etmektedir. *both-hem* İngilizce hem de belirtilen dillerden birini akıcı bir şekilde bilen çevirmenler, çevirilere katkıda bulunmaktan büyük memnuniyet duyacaktır. Bir çeviri tamamlandığında bile, frc-docs'daki değişikliklere ayak uydurmak için güncellenmesi gerekir.

43.5.1 İşakışı

Frc-docs çevirmek için izlenecek bazı adımlar.

1. [Transifex](#) için kaydolun ve [frc-docs projesine](#) katılmayı isteyin ve katkıda bulunmak istediğiniz dile erişim talep edin.
2. GitHub'a katılın *tartışmalarına* <<https://github.com/wpilibsuite/allwpilib/discussions>> __! Bu, WPILib ekibi ile doğrudan bir iletişim aracıdır. Bunu bize hızlı ve modern bir şekilde soru sormak için kullanabilirsiniz.
3. Frc-docs çeviri projesine erişim izni verilmeden önce sizinle iletişime geçilebilir ve katkıda bulunan dillerle ilgili sorular sorulabilir.
4. Dilinizi çevirin!

43.5.2 Linkler

Bağlantılar orijinal sözdizimlerinde korunmalıdır. Bir bağlantıyı çevirmek için, TRANSLATE ME metnini (bu, İngilizce başlıkla değiştirilecektir) uygun çeviriyle değiştirebilirsiniz.

Orijinal metnin bir örneği olabilir

```
For complete wiring instructions/diagrams, please see the :doc:`Wiring the FRC
↳Control System Document <Wiring the FRC Control System document>`.
```

burada Wiring the FRC Control System Document tercüme edilir.

```
For complete wiring instructions/diagrams, please see the :doc:`TRANSLATED TEXT
↳<Wiring the FRC Control System document>`.
```

Diger bir ornek aşağıda

```
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 Çevirileri Yayınlamak

Çeviriler Transifex'ten alınır ve her gün otomatik olarak yayınlanır.

43.5.4 Doğruluk

Çeviriler orijinal metne göre doğru olmalıdır. İngilizce metinde iyileştirmeler yapılabilir, **frc-docs** <<https://github.com/wpilibsuite/frc-docs>> __ deposunda bir PR veya sorun açın. Bunlar daha sonra birleştirilerek çevrilebilir.

43.6 En İyi Çevirmenler

43.6.1 Çince

- 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
- 曹 强
- 曹 强
- 曹 强
- 曹 强
- 曹 强
- 曹 强
- 曹 强
- 曹 Sherry

43.6.2 Fransızca

- 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 Portekizce

- 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 İspanyolca

- 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 Türkçe

- 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 İbranice

- Aric Radzin
- Dalton Smith
- Itay Ziv
- Ofek Ashery
- Shai Grossman
- Starlight220
- Yotam Shlomi

Allwpilib ile geliştirme

Önemli: Bu belge, *WPILib geliştiricileri* için bilgiler içerir. Bu, FRC® robotlarını programlamak için değildir.

Bu, [allwpilib](#) deposu için çeşitli belgelere bağlantıların bir listesidir.

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 Çekirdek Depo

44.3 NetworkTables

A

accelerometer, [577](#)
adım girişi - step input, [1217](#)
adım yanıtı - step response, [1217](#)
AM, [577](#)
AprilTags, [577](#)
ayar noktası - set point, [1216](#)

B

back-EMF, [577](#)
bang-bang control, [1213](#)
boolean, [577](#)

C

C ++, [578](#)
CAD, [577](#)
call stack, [577](#)
CAM, [577](#)
CAN, [577](#)
Cartesian coordinate system, [1213](#)
CC, [69](#)
CD, [578](#)
central limit theorem, [578](#)
churning losses, [1214](#)
CIM, [578](#)
Classical Mechanics, [578](#)
composition, [578](#)
control signal, [1214](#)
convolution, [1214](#)
COTS, [578](#)
counter-electromotive force, [1214](#)
CRTP, [578](#)
CSA, [578](#)
CTRE, [578](#)
current, [1214](#)
CXX, [69](#)

D

declarative programming, [578](#)
dependency injection, [578](#)
deprecated, [578](#)
derivative, [1214](#)
design pattern, [579](#)
DHCP, [579](#)
dinamikler - dynamics, [1214](#)

durum - state, [1217](#)

E

encapsulation, [579](#)
entry, [579](#)
enumeration, [579](#)
EPA, [579](#)
event-driven programming, [579](#)
exponential search, [1214](#)
exponential smoothing, [1214](#)

F

FIRST, [579](#)
FLL, [579](#)
floating point, [579](#)
FMS, [579](#)
FPGA, [579](#)
FRC, [580](#)
FTA, [580](#)
FTC, [580](#)

G

Gaussian distribution, [1215](#)
GDC, [580](#)
giriş - input, [1215](#)
gizli durum - hidden state, [1215](#)
GP, [580](#)
gradient, [1215](#)
GradleRIO, [580](#)
gyroscope, [580](#)

H

hata - error, [1214](#)

I

I2C, [580](#)
imperative programming, [580](#)
IMU, [580](#)

J

Java, [580](#)
JSON, [580](#)

K

kararlı durum hatası - steady-state error, [1217](#)
kazanç - gain, [1214](#)
Kontrol kanunu - control law, [1214](#)
Kontrol çabası - control effort, [1214](#)
Kontrolör - Controller, [1214](#)
KOP, [580](#)
KOP chassis, [581](#)

L

LabVIEW, [581](#)
least-squares regression, [1215](#)
LED, [581](#)
LQR, [1215](#)

M

mass, [581](#)
model, [1215](#)
moment of inertia, [581](#)
mutable, [581](#)
MXP, [581](#)

N

NetworkTables, [581](#)
no-op, [581](#)

O

odometry, [581](#)
onserver - gözlemci, [1215](#)
OPR, [581](#)
ortam değişkeni
 CC, [69](#)
 CXX, [69](#)
orthogonal, [1215](#)
otonom, [577](#)

P

PCM, [581](#)
PDH, [581](#)
PDP, [581](#)
permanent-magnet DC motor, [582](#)
persistent, [582](#)
PH, [582](#)
phase portrait, [1216](#)
PID, [1216](#)
plant - tesis, [1216](#)
pose, [582](#)
pose estimation, [582](#)
property, [582](#)
publisher, [582](#)
PWM, [582](#)
Python, [582](#)
Python Geliştirme Önerileri
 PEP 600, [69](#)

R

r-squared, [1216](#)
RAII, [582](#)
recursive composition, [582](#)
referans - reference, [1216](#)
retained, [582](#)
retro-reflection, [582](#)
REV, [583](#)
RMSE, [1216](#)

rota, [580](#)
RPM, [583](#)
RSL, [583](#)

S

serialized, [583](#)
signum function, [1216](#)
simülasyon, [583](#)
sistem - system, [1217](#)
sistem kimliği - system identification, [1217](#)
sistem yanıtı - system response, [1217](#)
software library, [583](#)
solenoid valve, [583](#)
SPI, [583](#)
state machine, [583](#)
statistically robust, [1217](#)
subscriber, [583](#)
süreç değişkeni - process variable, [1216](#)

T

TBA, [584](#)
telemetry, [584](#)
topic, [584](#)
torque, [584](#)
trajectory, [584](#)
transitory, [584](#)

U

uzaktan kontrol, [584](#)

V

viscous drag, [1217](#)
voltage, [1217](#)
VRM, [584](#)

W

WCP, [584](#)
WFA, [584](#)

X

x-dot, [1218](#)
x-hat, [1218](#)

Y

yerleşme zamanı - settling time, [1216](#)
Yükseliş zamanı - rise time, [1216](#)



çıktı, [1216](#)
ölçüm - measurement, [1215](#)