Corbels FTC Programming Lessons in Java

v0.1.13-rc.1-3-gfda2787

Mike Stitt

Oct 21, 2026

Table of Contents

1 003 welcome **1**

2 010 getting started **3**

3 011 how-to: get started on Windows **5**
3.1 the steps . . . 6

4 012 how-to: get started on macOS **17**
4.1 the steps . . . 18

5 013 how-to: get started on Ubuntu **27**
5.1 the steps . . . 28

6 017 how-to: navigate Android Studio's panes **39**

7 020 lesson: read the sticks **43**
7.1 the steps . . . 44

8 021 what's happening: assignments **53**

9 030 lesson: turn the sticks into speeds **59**
9.1 the steps . . . 60

10 035 what's happening: sticks into speeds **63**

11 040 lesson: drive like a tank **73**
11.1 the steps . . . 74

12 041 what's happening: calling a method **77**

13 050 lesson: drive the robot on blocks **79**
13.1 the steps . . . 80

14 060 lesson: shape the sticks **85**
14.1 the steps . . . 85

15 070 lesson: read the buttons **91**
15.1 the steps . . . 92

16 080 lesson: drive like an arcade game **97**
16.1 the steps . . . 98

17 090 lesson: slide it sideways **103**
17.1 the steps . . . 104

18 093 how-to: watch the numbers while it drives **111**
18.1 what you will have when this is done . . . 111
18.2 before you start . . . 111
18.3 the steps . . . 111
18.4 what you just did . . . 113
18.5 where next . . . 113

19 095 how-to: check your work without a robot **115**
19.1 what you will have when this is done . . . 115
19.2 before you start . . . 115
19.3 the steps . . . 115
19.4 what you just did . . . 117
19.5 where next . . . 117

20 110 lesson: hand the wheels to the path follower **119**
20.1 the steps . . . 120

21 115 lesson: mount the Pinpoint and plug it in **123**
21.1 the steps . . . 124

22 117 lesson: let AutoTune tune the robot **127**
22.1 the steps . . . 127

23 120 lesson: work out where the robot is from the drive wheels **131**
23.1 the steps . . . 132

24 125 lesson: the first autonomous **139**
24.1 the steps . . . 140

25 130 lesson: two legs, forward then sideways **143**
25.1 the steps . . . 144

26 140 lesson: make the sticks mean the field **147**
26.1 the steps . . . 148

27 150 lesson: give the driver both **151**
27.1 the steps . . . 152

28 160 lesson: hold the heading the driver left **155**
28.1 the steps . . . 156

29 170 lesson: press Y and the robot drives itself there **159**
29.1 the steps . . . 160

30 180 lesson: one OpMode a driver could take to a match **163**
30.1 the steps . . . 164

31 190 lesson: ask the wheels for a speed **167**
31.1 the steps . . . 168

32 195 lesson: measure the two numbers L190 trusted **173**
32.1 the steps . . . 174

33 900 reference: Pedro routes **177**
33.1 the shapes . . . 177
33.2 the heading rules . . . 182
33.3 `linear()` on a line was reversed before 3.0.1 . . . 183
33.4 `longLinear()` takes the long way round . . . 184
33.5 `facingPoint()` has one edge . . . 185
33.6 what this page does not cover . . . 186

34 910 reference: glossary **187**

35 920 reference: keyboard shortcuts **191**

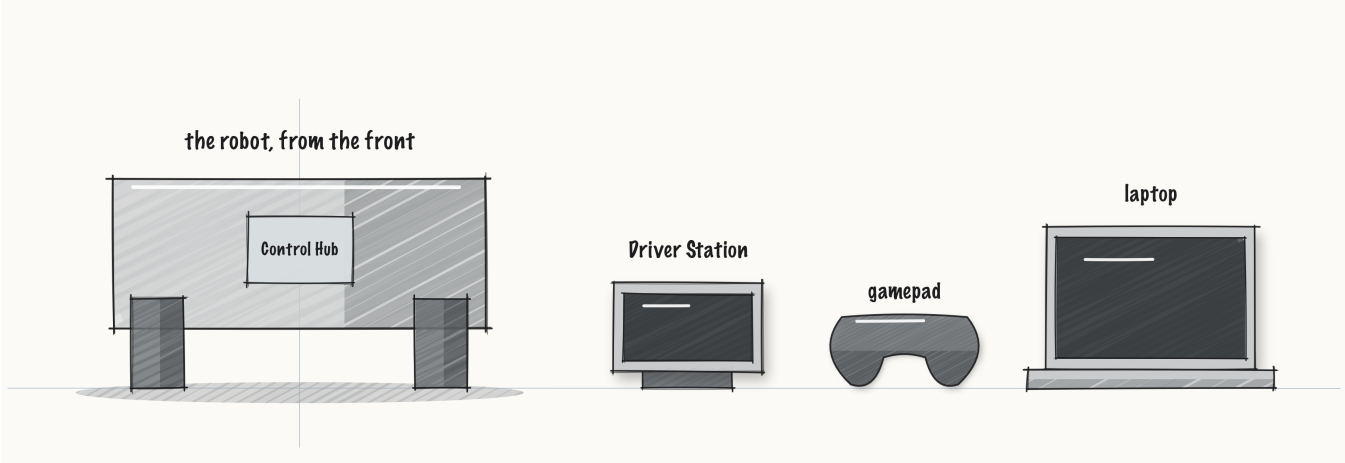
36 930 reference: cheat sheet **193**
36.1 what to run . . . 193
36.2 what shows up in AdvantageScope . . . 194

37 950 reference: answers **199**
37.1 L020SticksOpMode . . . 199
37.2 L030SpeedsOpMode . . . 199
37.3 L040TankOpMode . . . 200
37.4 L050BlocksOpMode . . . 200
37.5 L060ShapingOpMode . . . 201
37.6 L070ButtonsOpMode . . . 202
37.7 L080ArcadeOpMode . . . 202
37.8 L090MecanumOpMode . . . 203
37.9 L110FollowerDriveTrain . . . 204
37.10 L110FollowerOpMode . . . 204
37.11 L120CompareLocalizersOpMode . . . 205
37.12 L120EncoderLocalizer . . . 205
37.13 L125Drive24OpMode . . . 206
37.14 L130ForwardThenStrafeOpMode . . . 206
37.15 L140FieldRelativeOpMode . . . 207
37.16 L150RobotRelativeButtonOpMode . . . 207
37.17 L160HeadingHoldOpMode . . . 208
37.18 L170DriveToPoseOpMode . . . 208
37.19 L180CombinedOpMode . . . 209
37.20 L190SpeedDriveTrain . . . 209
37.21 L190VelocityDriveOpMode . . . 210
37.22 L195TicksDriveTrain . . . 211

38 990 where this came from **213**
38.1 the guide's own terms . . . 213
38.2 Game Manual 0 . . . 213
38.3 Pedro Pathing . . . 213
38.4 the FTC SDK . . . 213
38.5 AdvantageScope . . . 213
38.6 not affiliated with FIRST . . . 214

003 welcome

Welcome to Corbels FTC Programming Lessons in Java! It teaches you to program an FTC robot in Java, one page at a time.



This guide explains how to program FTC robots, one lesson at a time. Each lesson page says what you will have when it is done, what has to be true before you start, and the steps to get there. Every step tells you what you will see when it worked.

How to use it.

- **It's best to start at the beginning.** But you don't have to, other than getting the wheel directions right in lesson L050, Pinpoint installed in lesson L115, and Pedro Pathing tuned in lesson L117. Most can be accomplished without doing the previous lesson.
- **Do the step, then read the check.** The check tells you whether to go on.
- **The answers are in the back.** Use them when you have tried and need a hint.
- The code and the source to build these pages are at https://github.com/msb-stitt/ftc31459-26-27-biobuzz.
- Versions of this book are listed at https://msb-stitt.github.io/ftc31459-26-27-biobuzz/.

What this book was made from.

- The lessons and the code: `v0.1.13-rc.1-3-gfda2787`, in ftc31459-26-27-biobuzz.
- The screenshots, drawings and videos: `lessons-e035-pedestals` at `b0a4e88`, in corbels-lfs-lessons.

- The answers in the back: `solutions-auto-schedules.1`, in ftc31459-26-27-biobuzz.

A name like `v0.1.4` on its own is a release. More after it, like `-11-g24411ba`, means a copy made between releases, 11 changes after it. `-dirty` means it was made with changes that were not yet saved in git.

010 getting started

Put the robot's code on your laptop and build it.

There are two computers. One is inside the REV Control Hub, on the robot. The other is your laptop. It runs Windows, macOS or Ubuntu, and you write the program on it.

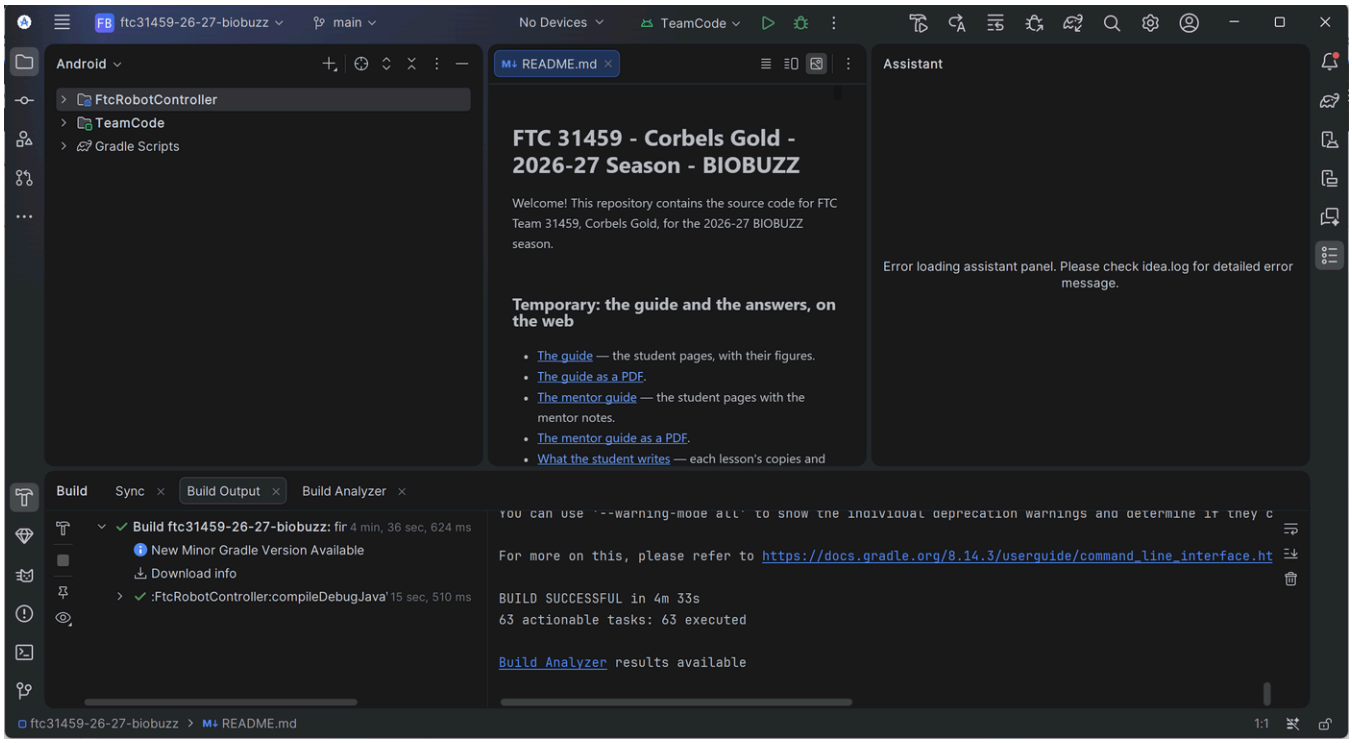
Android Studio is an **IDE**. That is short for Integrated Development Environment. You write the robot's program in it, in Java. Then it puts the program on the robot's computer. The program tells that computer what numbers to work out. It also says where to put them.

AdvantageScope is a **scope**, as in telescope or microscope. It shows the numbers inside the running program.

Use the lesson for your computer:

- Windows: H011
- a Mac: H012
- Ubuntu: H013

011 how-to: get started on Windows



Put the robot's code on a Windows laptop and build it. After this, every lesson starts in Android Studio with the project open.

Before you start.

- A Windows laptop, signed in to an account that may install programs.
- The internet.
- No robot and no gamepad are needed.

Nothing else has to be installed first. Android Studio fetches Java 17 for the project by itself, and git is installed in L011S040 when Android Studio asks for it.

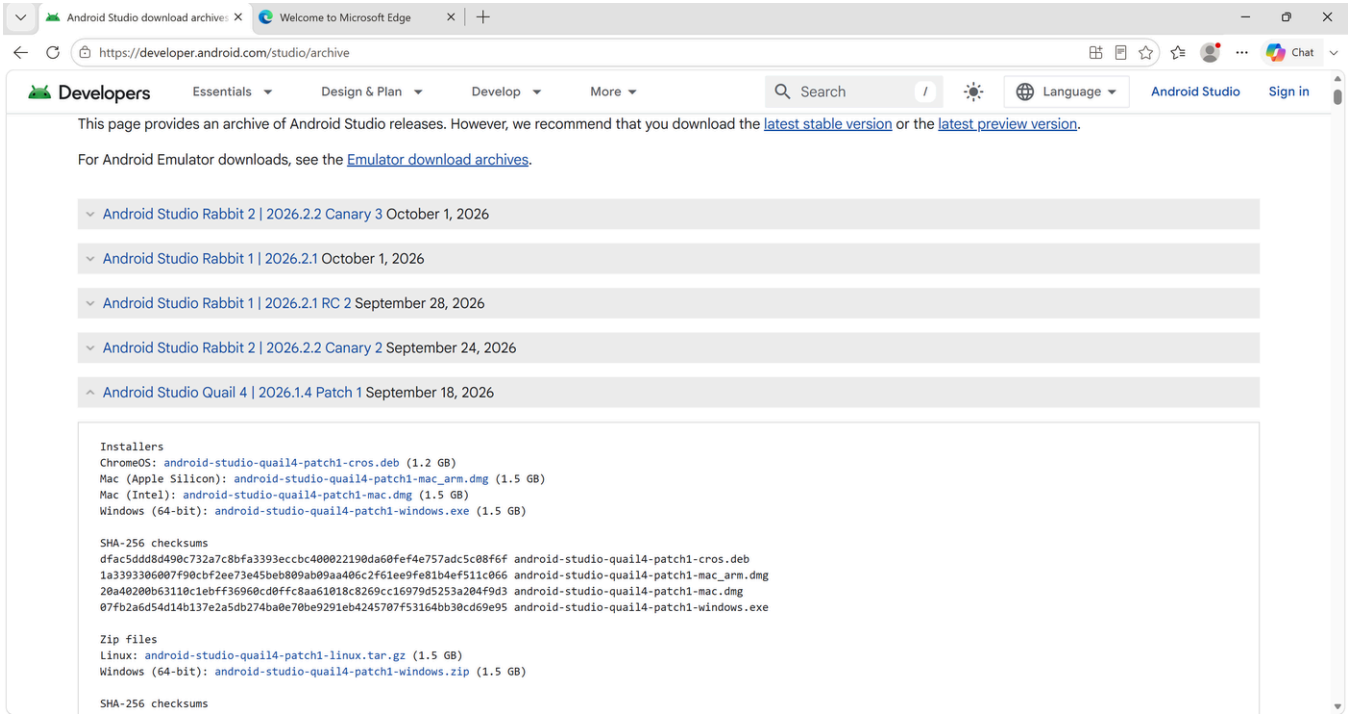
3.1 the steps

3.1.1 H011S010: install Android Studio

The robot's code is written in Android Studio. The team uses one version of it, **Quail 4**, so everyone sees the same screens. It is not the newest, so it comes from the page of older versions.

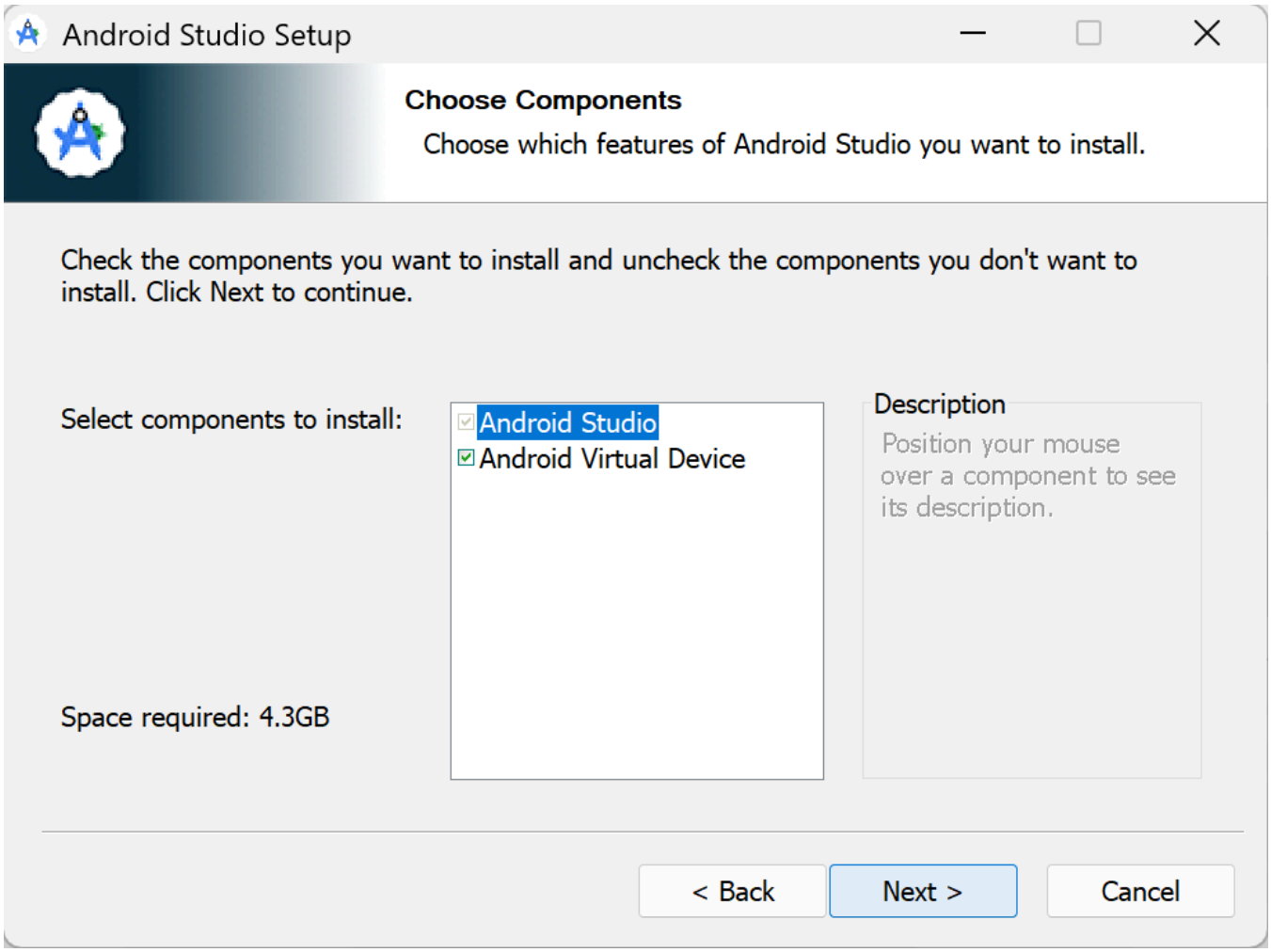
Download.

- Open https://developer.android.com/studio/archive and click **I agree** under the terms.
- Click **Android Studio Quail 4 | 2026.1.4 Patch 1** to open it.
- Click `android-studio-quail4-patch1-windows.exe`. It is about 1.5 GB.



Install.

- Open the downloaded file. Click **Yes** when Windows asks to let it change your device.
- In Android Studio Setup, click **Next** on each page, then **Install**.
- When it is done, click **Next**, then **Finish**.



Check.

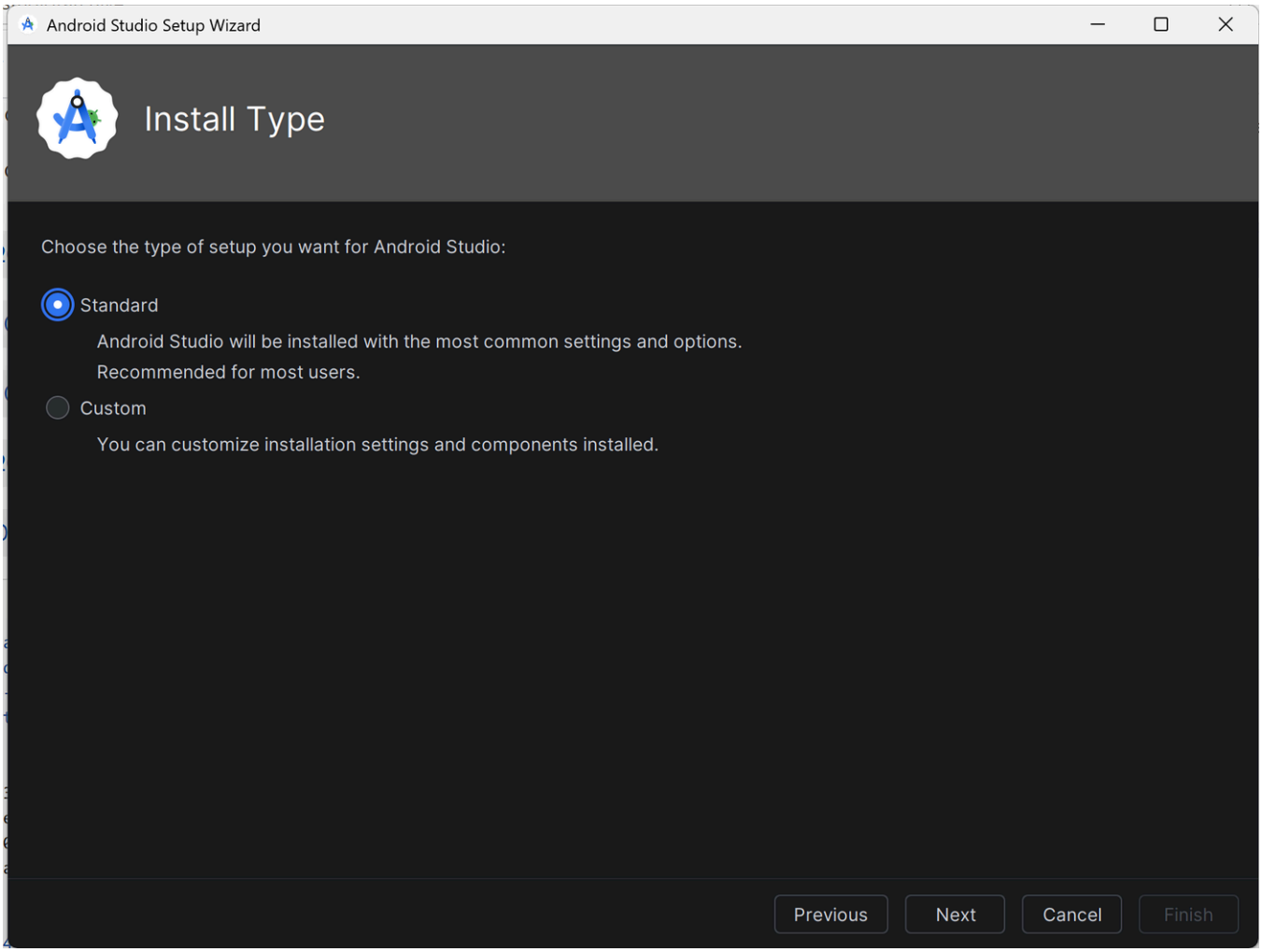
- Android Studio starts and asks to *Help improve Android Studio*. Click **Don't send**.

3.1.2 H011S020: let Android Studio fetch what it needs

The first time it starts, Android Studio's setup wizard downloads the Android tools the code is built with.

Setup.

- Click **Next** on *Welcome*.
- Pick **Standard** and click **Next**.
- Click **Next** on *Verify Settings*.
- Click each license on the left, pick **Accept**, and click **Next**.
- Wait while it downloads, then click **Finish**.



Check.

- The last page of the wizard reads *Android SDK is up to date*. A red line may be under it: *SDK emulator directory is missing*. That is fine.
- *Welcome to Android Studio* opens, with **New Project**, **Open** and **Clone Repository**.

If a note says *Rabbit 1 available*, leave it. The team stays on Quail 4.

3.1.3 H011S030: install AdvantageScope

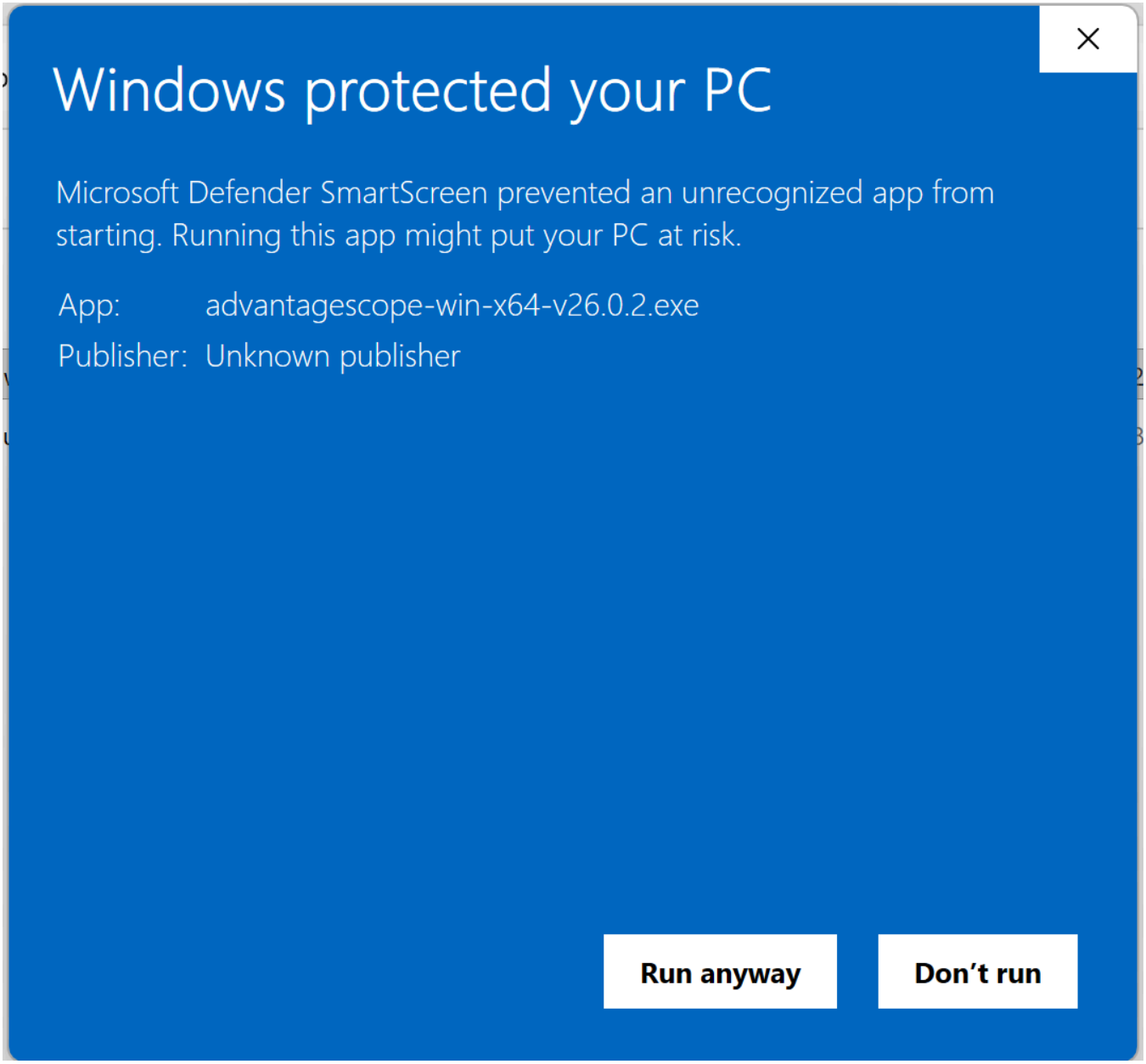
AdvantageScope draws what the robot is doing: where it is on the field, and graphs of its numbers.

Download. Open https://github.com/Mechanical-Advantage/AdvantageScope/releases/tag/v26.0.2. Under **Assets**, click `advantagescope-win-x64-v26.0.2.exe`.

Install.

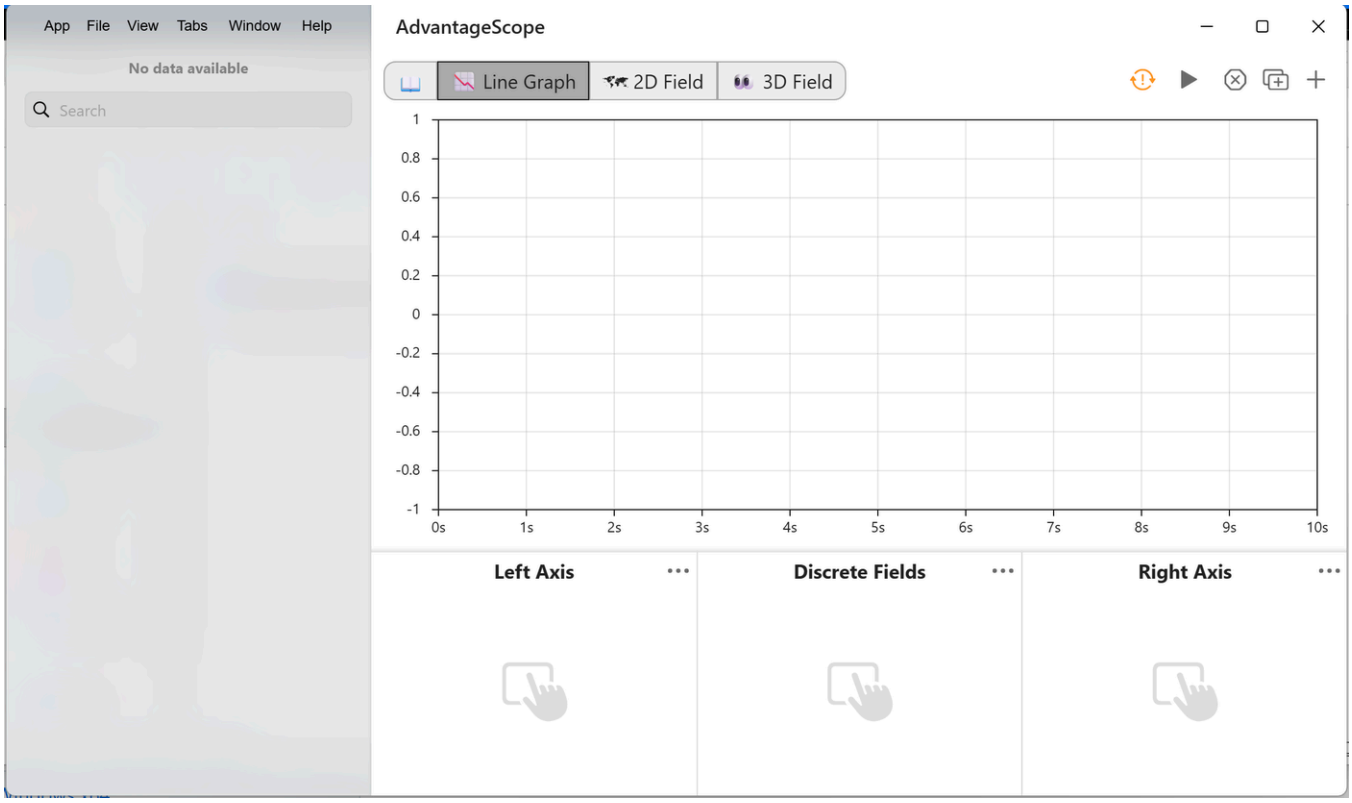
- Edge may say the file *isn't commonly downloaded*. Click the three dots by it, then **Keep**, then **Keep anyway**.
- Open the file from your Downloads folder.

- If you see *Windows protected your PC*, click **More info**, then **Run anyway**. It sets itself up and starts.



Check.

- AdvantageScope opens on its **Line Graph** tab, with *No data available* on the left.

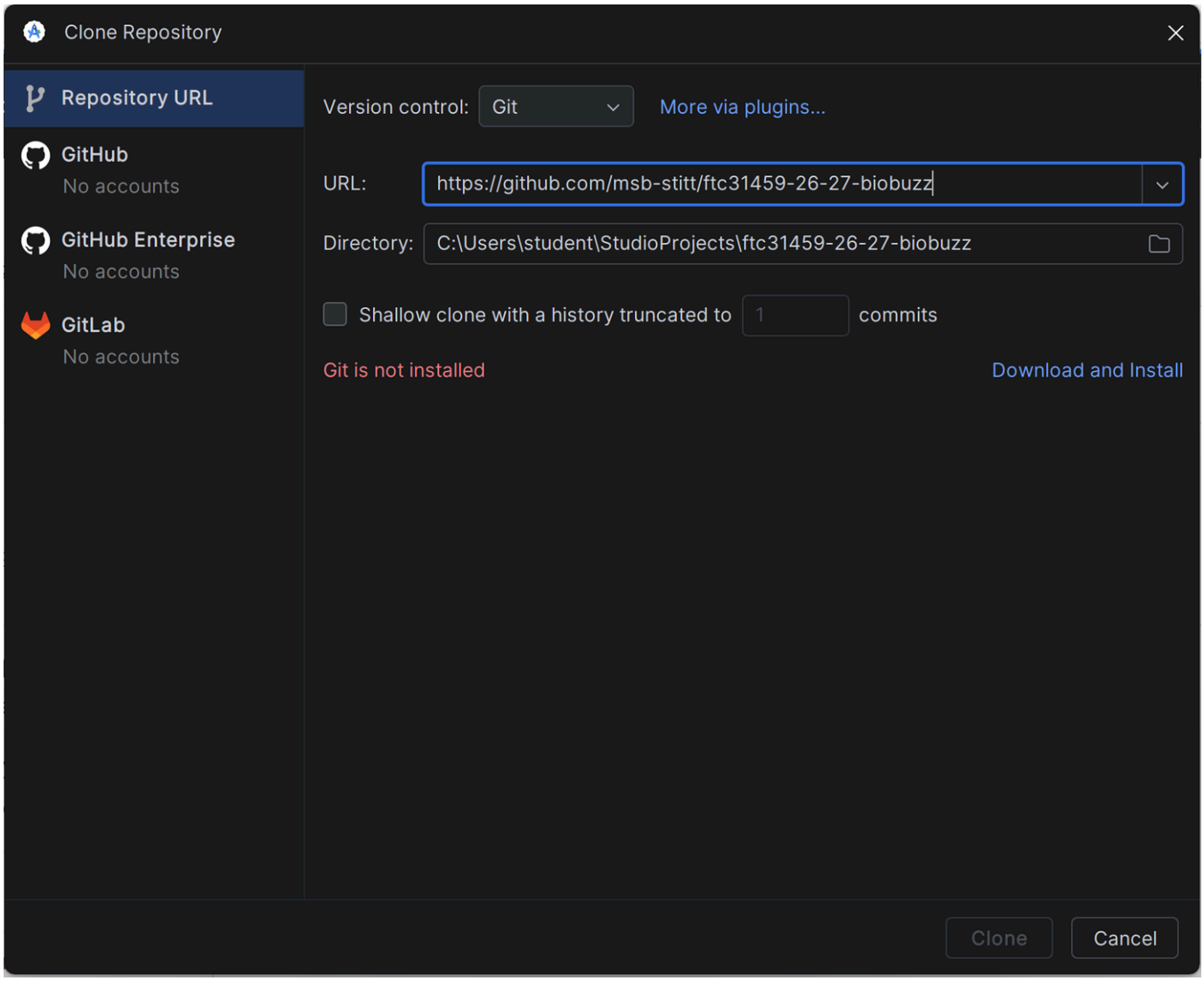


3.1.4 H011S040: get the project

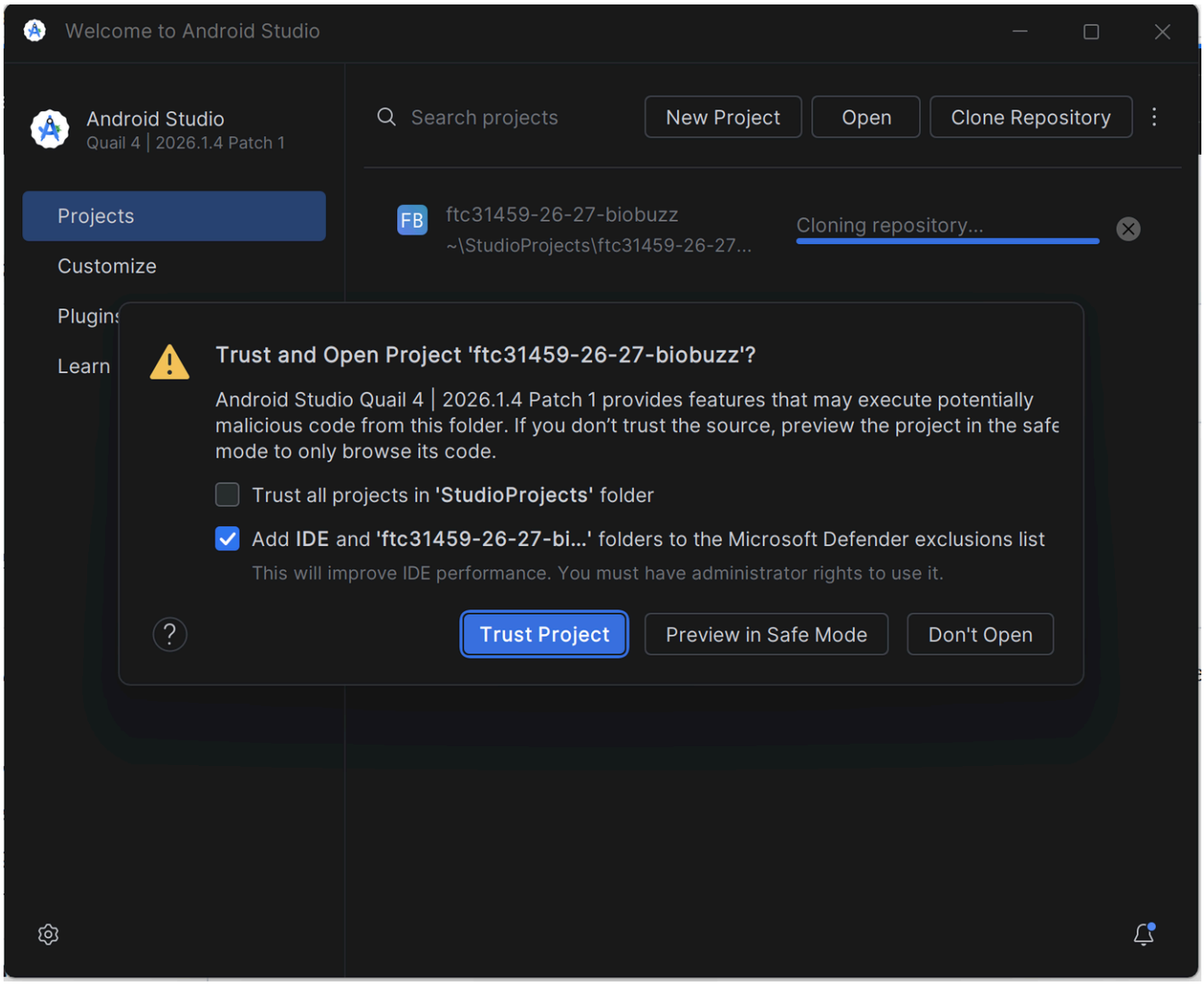
The robot's code lives on GitHub. Android Studio copies it to the laptop with a tool called **git**.

Clone.

- On *Welcome to Android Studio*, click **Clone Repository**.
- Into **URL**, type `https://github.com/msb-stitt/ftc31459-26-27-biobuzz`. Leave **Directory** as it is.
- The first time, it says *Git is not installed*. Click **Download and Install** beside it. Click **Yes** when Windows asks.
- When it says *Git has been installed*, click **Clone**.



Trust. Android Studio asks if you trust the project. A box about Microsoft Defender is ticked. Leave it ticked, and click **Trust Project**.



Check.

- The project opens, and the Project window lists `FtcRobotController` and `TeamCode`.

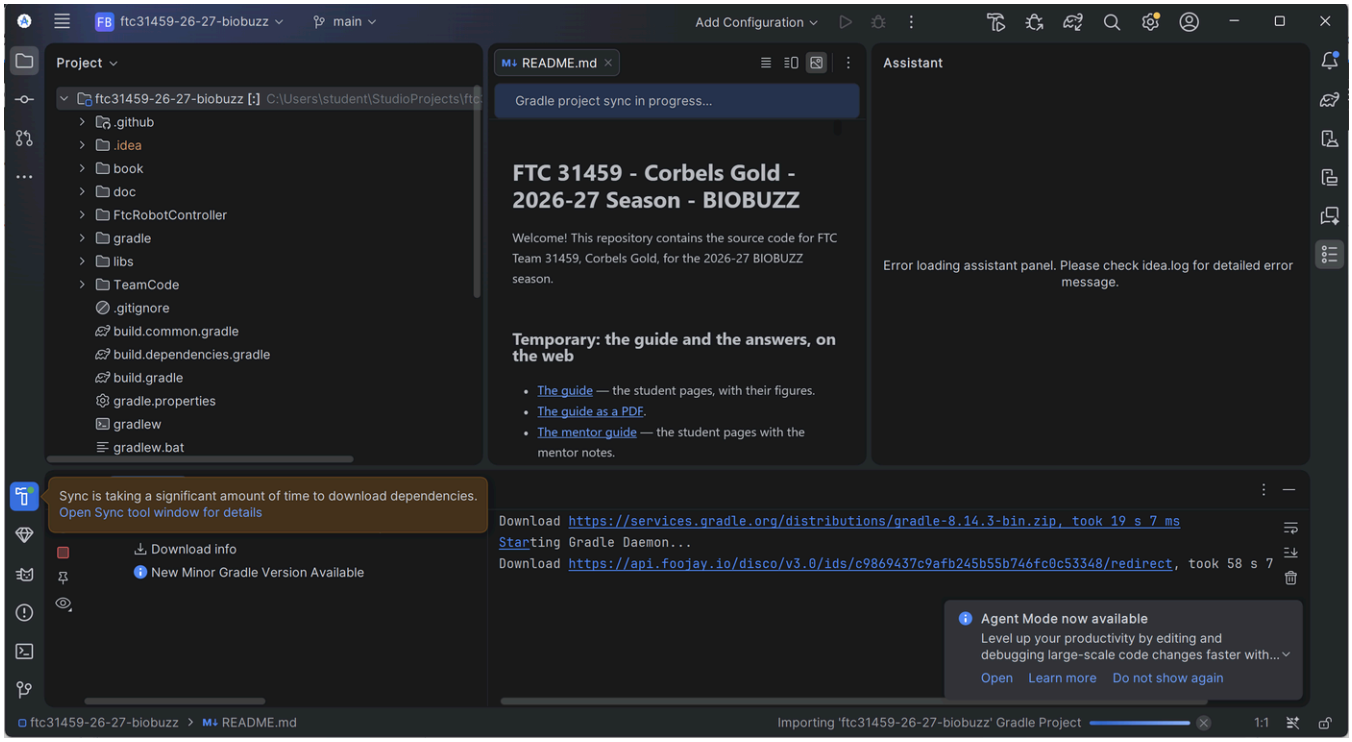
3.1.5 H011S050: answer what Android Studio offers

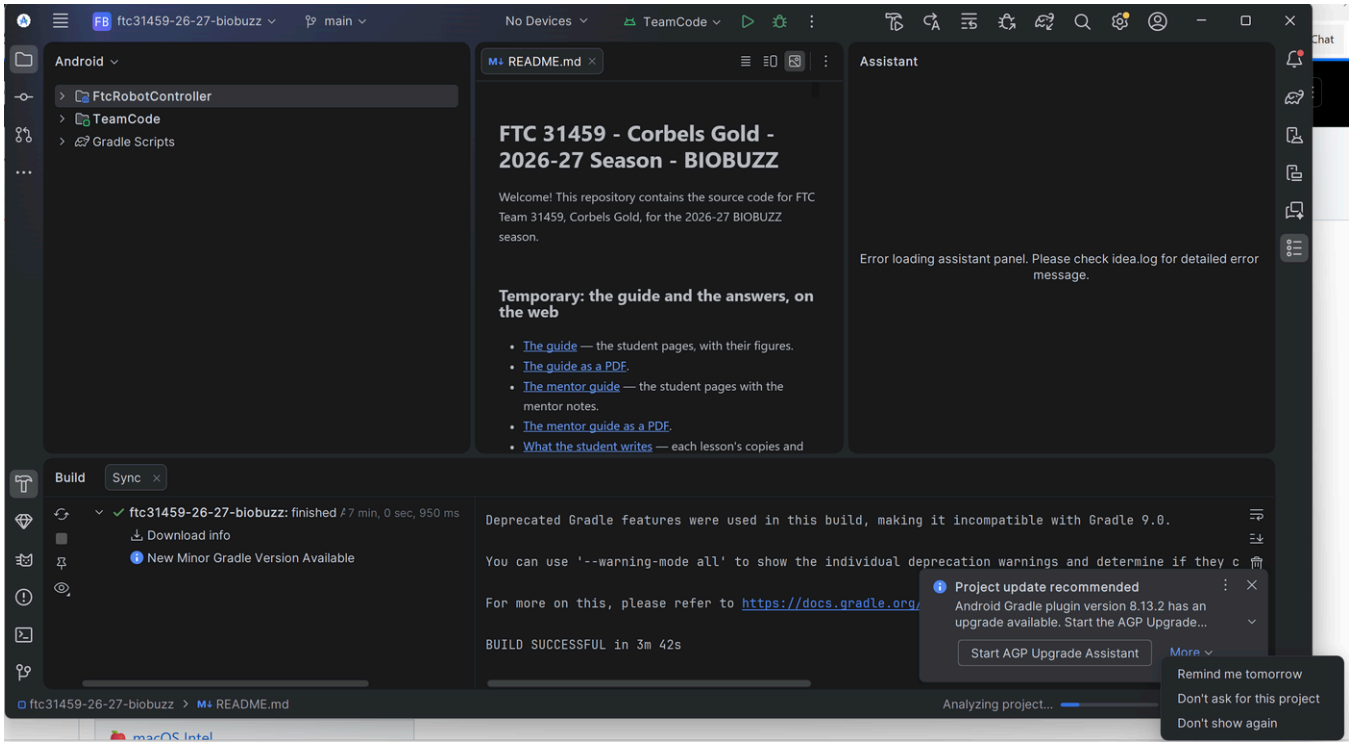
While the project loads, Android Studio reads it with **Gradle**, the tool that builds the code. This is called a *sync*. The first sync downloads Java 17 and more, and takes a few minutes.

Answer. Notes pop up in the bottom right while it works. Answer each one like this:

- **Agent Mode now available**: click **Do not show again**. The lessons do not use an AI helper.
- **Project update recommended**, *Android Gradle plugin version 8.13.2 has an upgrade available*: click **More**, then **Don't ask for this project**. The project needs version 8.13.2, so do not upgrade it.
- **Microsoft Defender may affect IDE**: click **Exclude folders**.

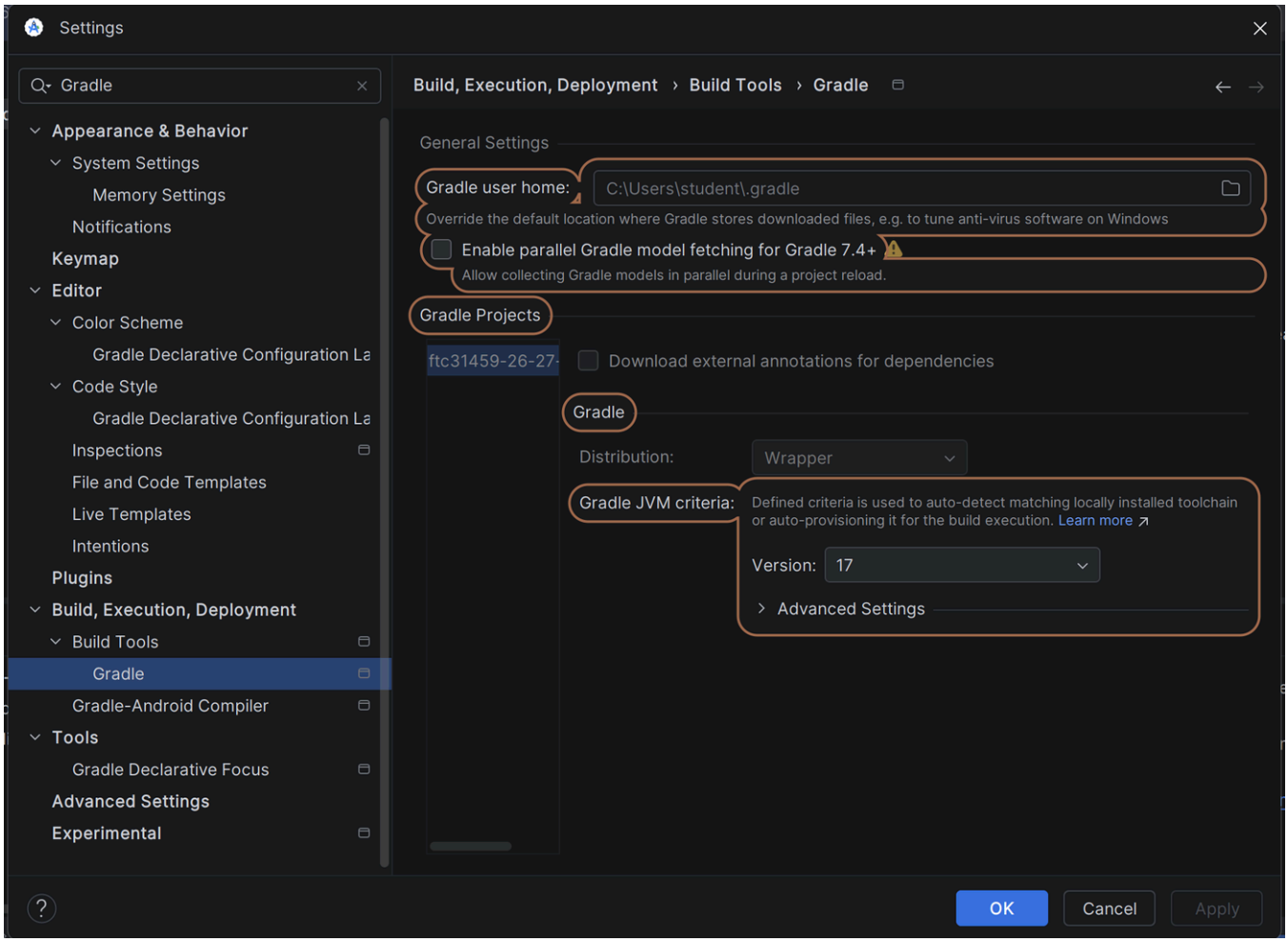
- If Windows asks whether `adb` can use the network, click **Allow**.





Check.

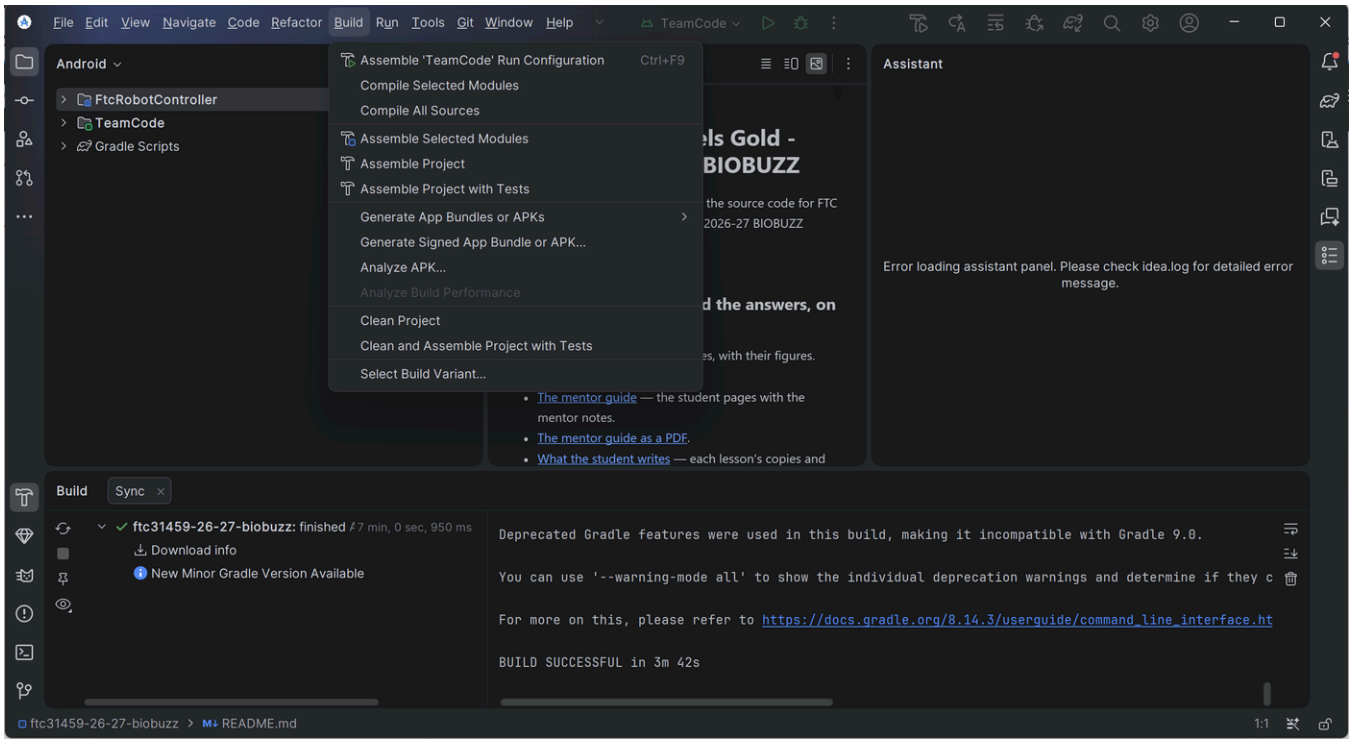
- The Build window's **Sync** tab has a green tick beside the project.
- Open **File → Settings**. Type `Gradle` in its search box and click **Gradle** under **Build Tools**. Under *Gradle JVM criteria*, **Version** reads `17`.



If it didn't, on a Windows laptop with an ARM chip, and the sync fails with `AccessDeniedException` while moving the downloaded JDK into `.gradle\jdks`, stop the Windows service `XtaCache` as an administrator and sync again with **File → Sync Project with Gradle Files**.

3.1.6 H011S060: build the code

Build. Click **Build → Assemble Project**. The first build takes a few minutes.



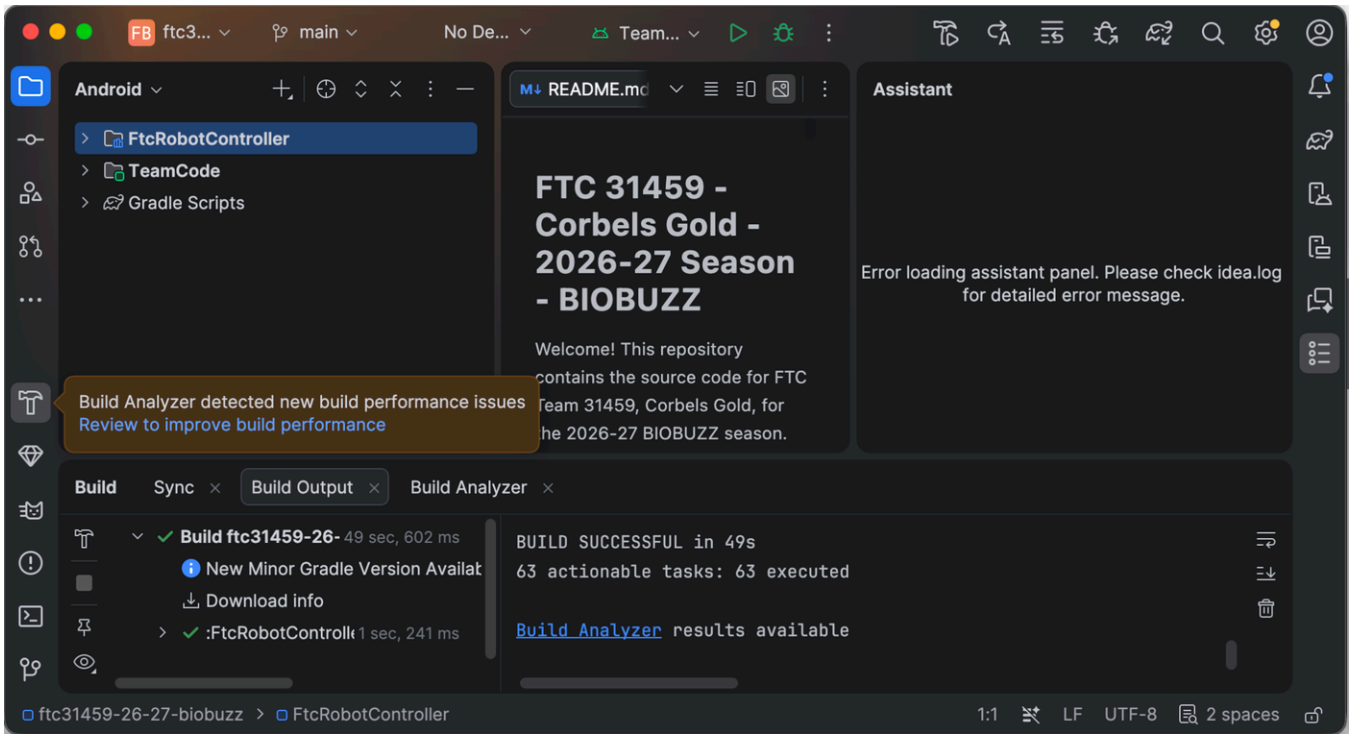
Check.

- The Build window's **Build Output** tab ends with *BUILD SUCCESSFUL* and *63 actionable tasks: 63 executed*.

What you just did. Android Studio is where the code is written, and Gradle turns the code into an app for the robot. Clone copied the team's repository from GitHub onto the laptop. The sync read the project's Gradle files and downloaded what they ask for: Gradle 8.14.3, the Android Gradle plugin 8.13.2 and Java 17 among them. Assemble Project built `FtcRobotController` and `TeamCode` into `TeamCode-debug.apk`, the app that is sent to the robot.

Where next. L020 copies the first starter into `mytry` and reads the gamepad's sticks.

012 how-to: get started on macOS



Put the robot's code on a Mac and build it. After this, every lesson starts in Android Studio with the project open.

Before you start.

- A Mac, signed in to an account that may install programs.
- The internet.
- No robot and no gamepad are needed.

Nothing else has to be installed first. Android Studio fetches Java 17 for the project by itself, and git is installed in L012S040 when macOS asks for it.

4.1 the steps

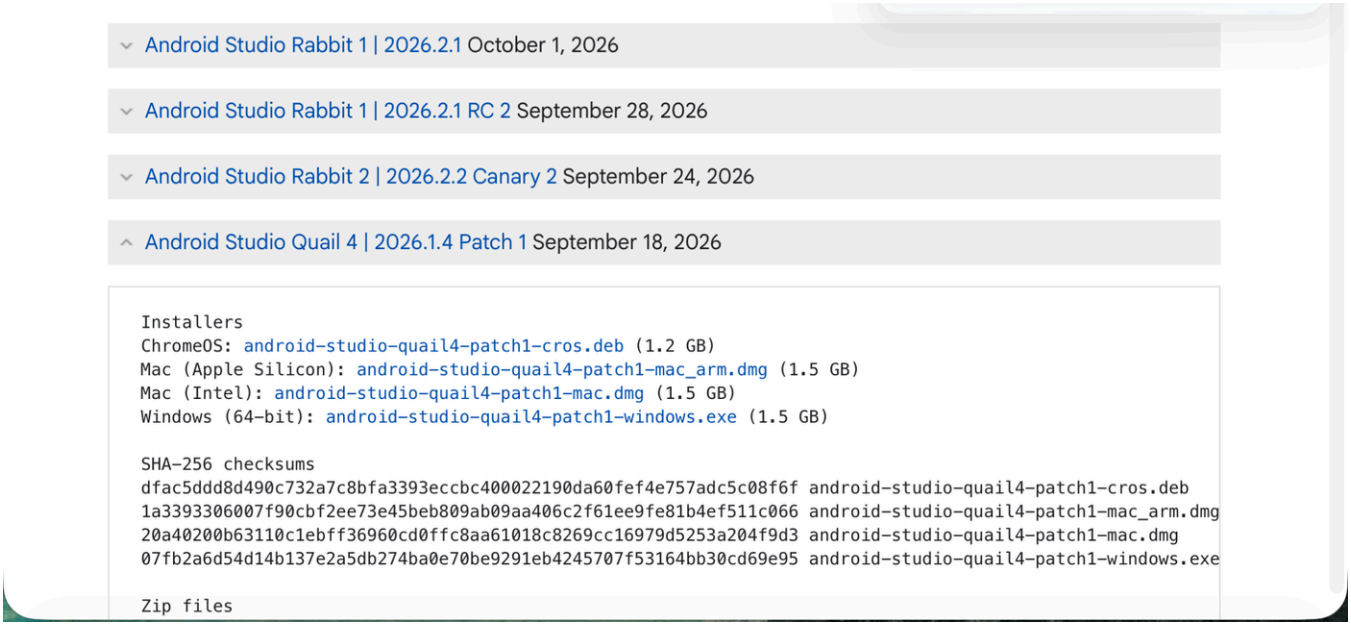
4.1.1 H012S010: install Android Studio

The robot's code is written in Android Studio. The team uses one version of it, **Quail 4**, so everyone sees the same screens. It is not the newest, so it comes from the page of older versions.

Chip. Click the Apple menu, then **About This Mac**. It shows whether the Mac has an Apple chip or an Intel chip. The files below differ for each.

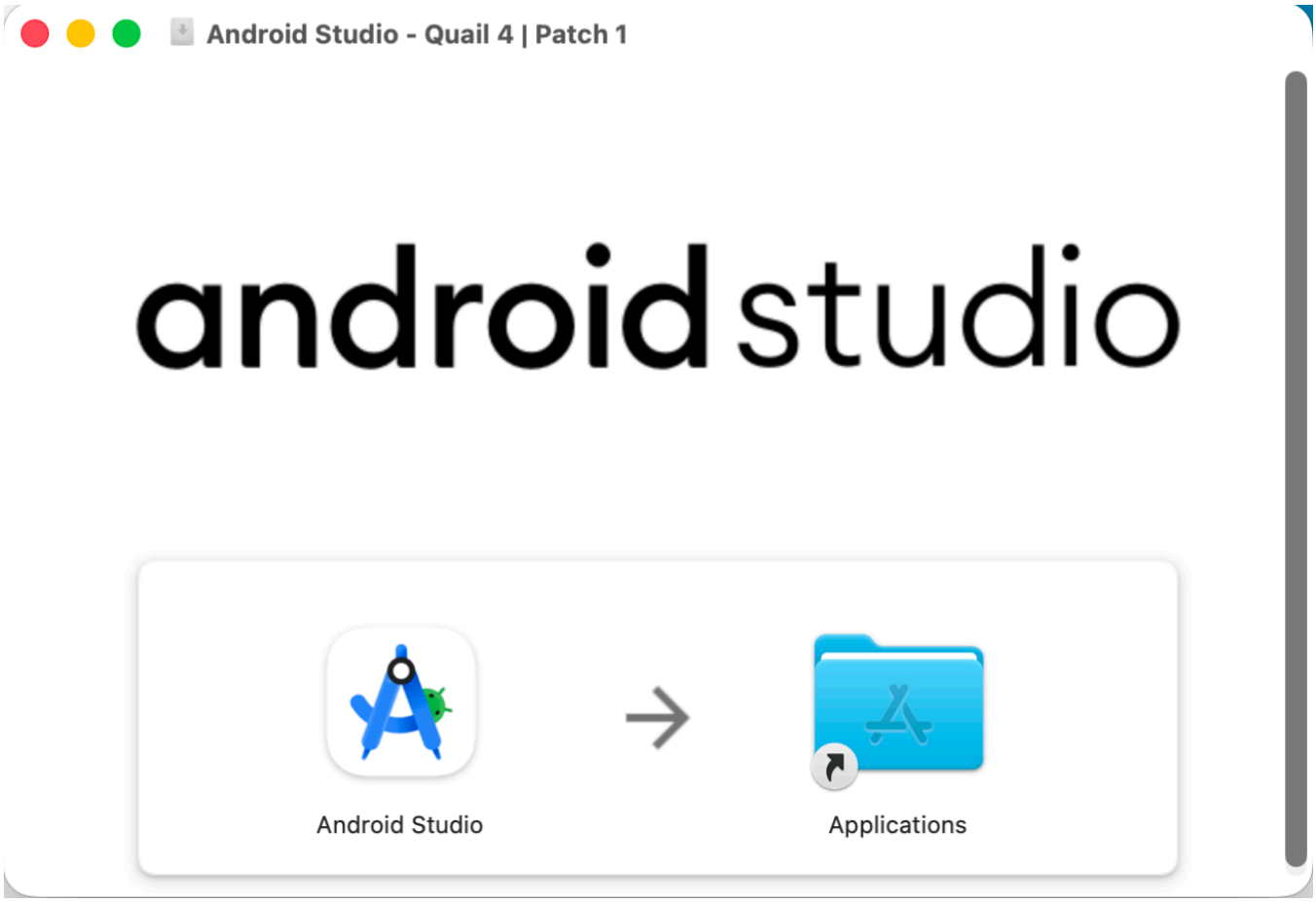
Download.

- Open https://developer.android.com/studio/archive and click **I agree** under the terms.
- Click **Android Studio Quail 4 | 2026.1.4 Patch 1** to open it.
- Click the file for your Mac: `android-studio-quail4-patch1-mac_arm.dmg` for an Apple chip, or `android-studio-quail4-patch1-mac.dmg` for an Intel chip. It is about 1.5 GB. If Safari asks to allow downloads, click **Allow**.



Install.

- Open the downloaded file.
- Drag Android Studio onto the Applications folder.
- Start Android Studio from Applications. If macOS says it came from the internet, click **Open**.



Check.

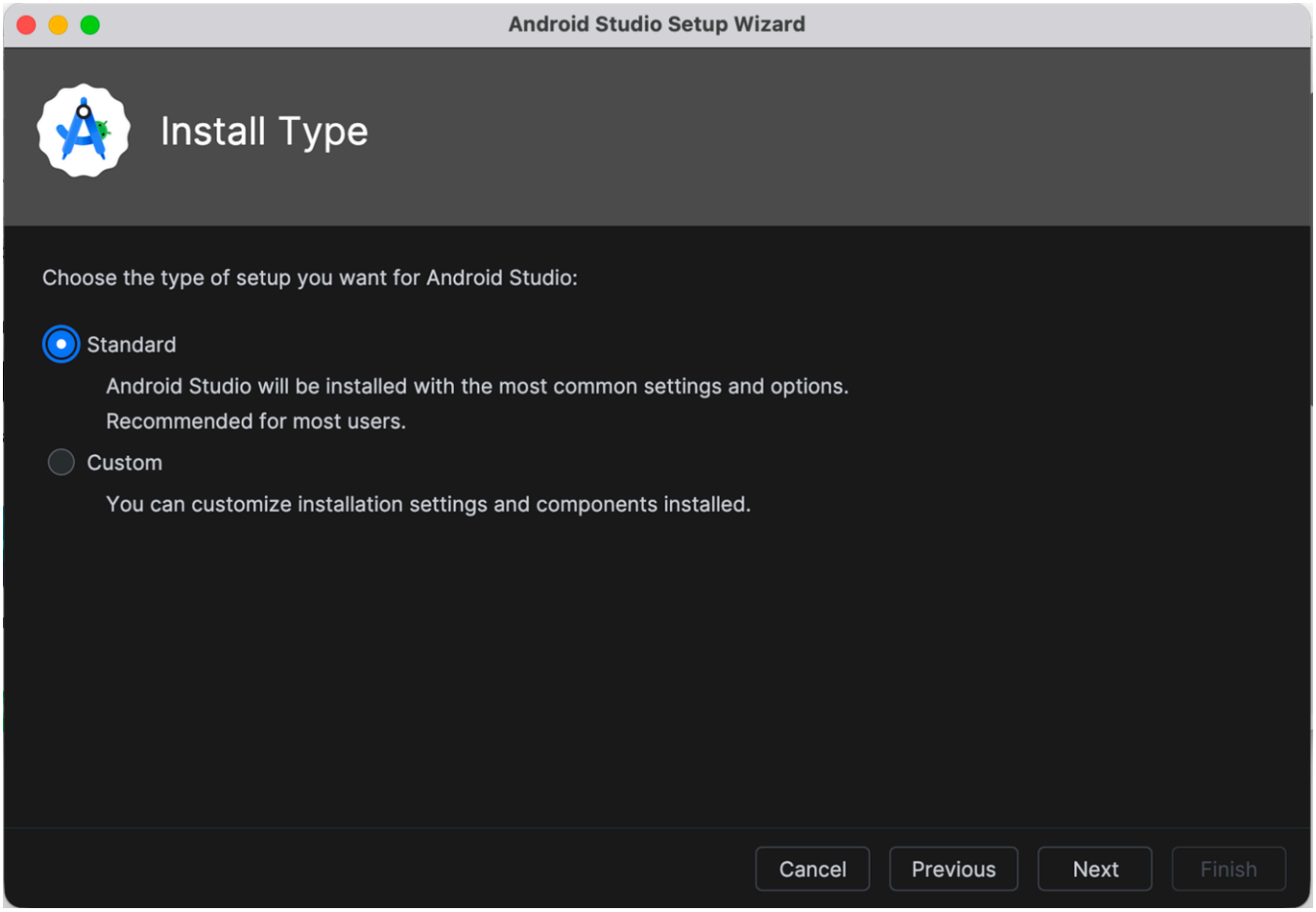
- Android Studio starts and asks to *Help improve Android Studio*. Click **Don't send**.

4.1.2 H012S020: let Android Studio fetch what it needs

The first time it starts, Android Studio's setup wizard downloads the Android tools the code is built with.

Setup.

- Click **Next** on *Welcome*.
- Pick **Standard** and click **Next**.
- Click **Next** on *Verify Settings*.
- Click each license on the left, pick **Accept**, and click **Next**.
- Wait while it downloads, then click **Finish**.



Check.

- The last page of the wizard reads *Android SDK is up to date*.
- *Welcome to Android Studio* opens, with **New Project**, **Open** and **Clone Repository**.

If a note says *Rabbit 1 available*, leave it. The team stays on Quail 4.

4.1.3 H012S030: install AdvantageScope

AdvantageScope draws what the robot is doing: where it is on the field, and graphs of its numbers.

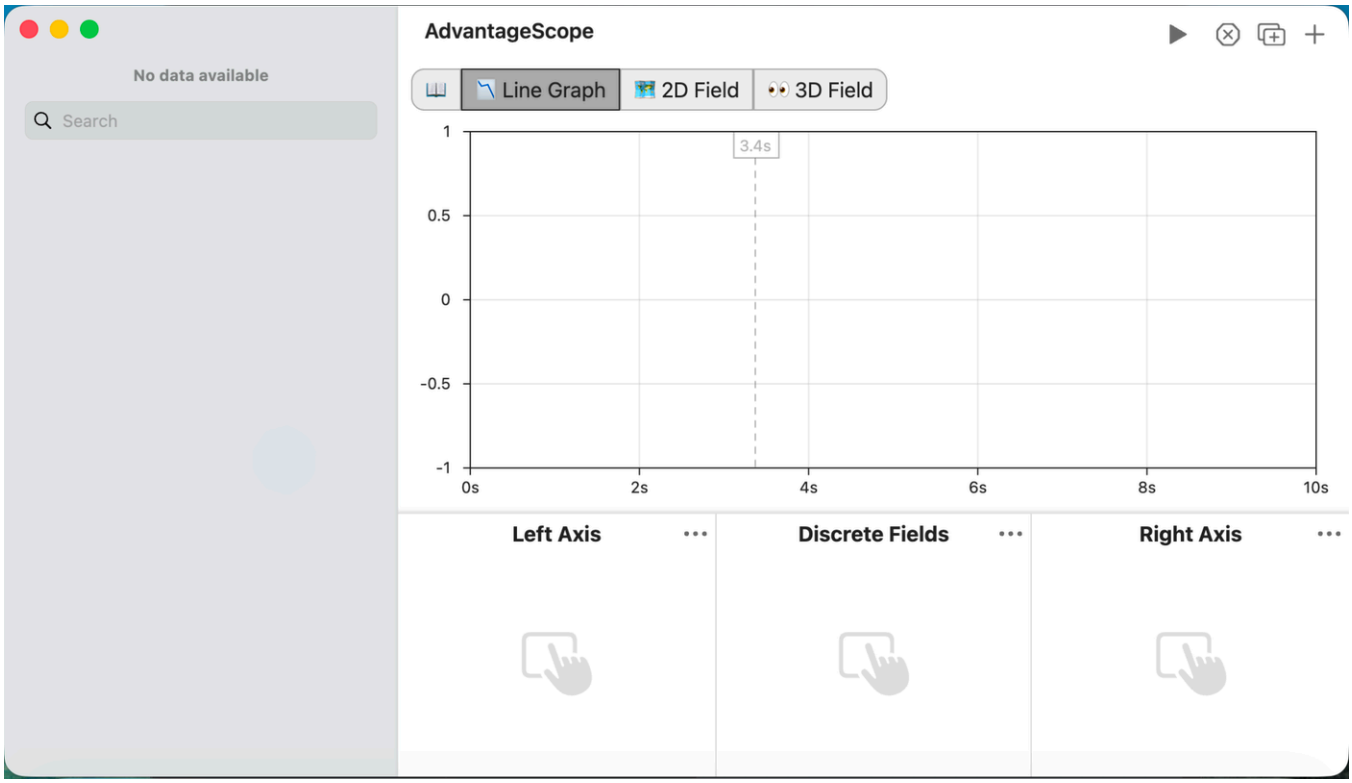
Download. Open https://github.com/Mechanical-Advantage/AdvantageScope/releases/tag/v26.0.2. Under **Assets**, click the file for your Mac:

- an Apple chip: `advantagescope-mac-arm64-v26.0.2.dmg`
- an Intel chip: `advantagescope-mac-x64-v26.0.2.dmg`

Install. Open the file. Drag AdvantageScope onto the Applications folder. Start it from there. If macOS asks, click **Open**.

Check.

- AdvantageScope opens on its **Line Graph** tab, with *No data available* on the left.

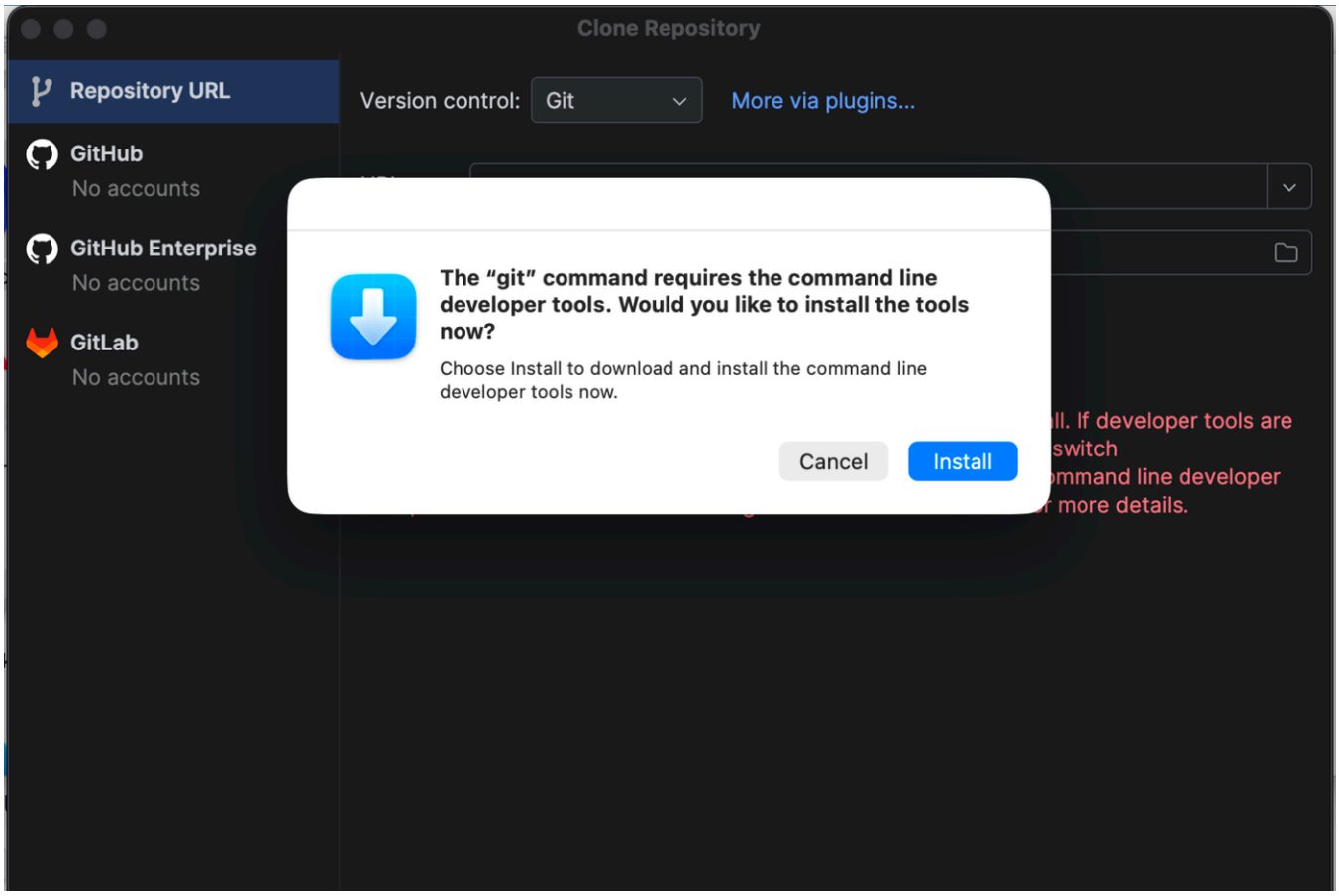


4.1.4 H012S040: get the project

The robot's code lives on GitHub. Android Studio copies it to the laptop with a tool called **git**.

Git.

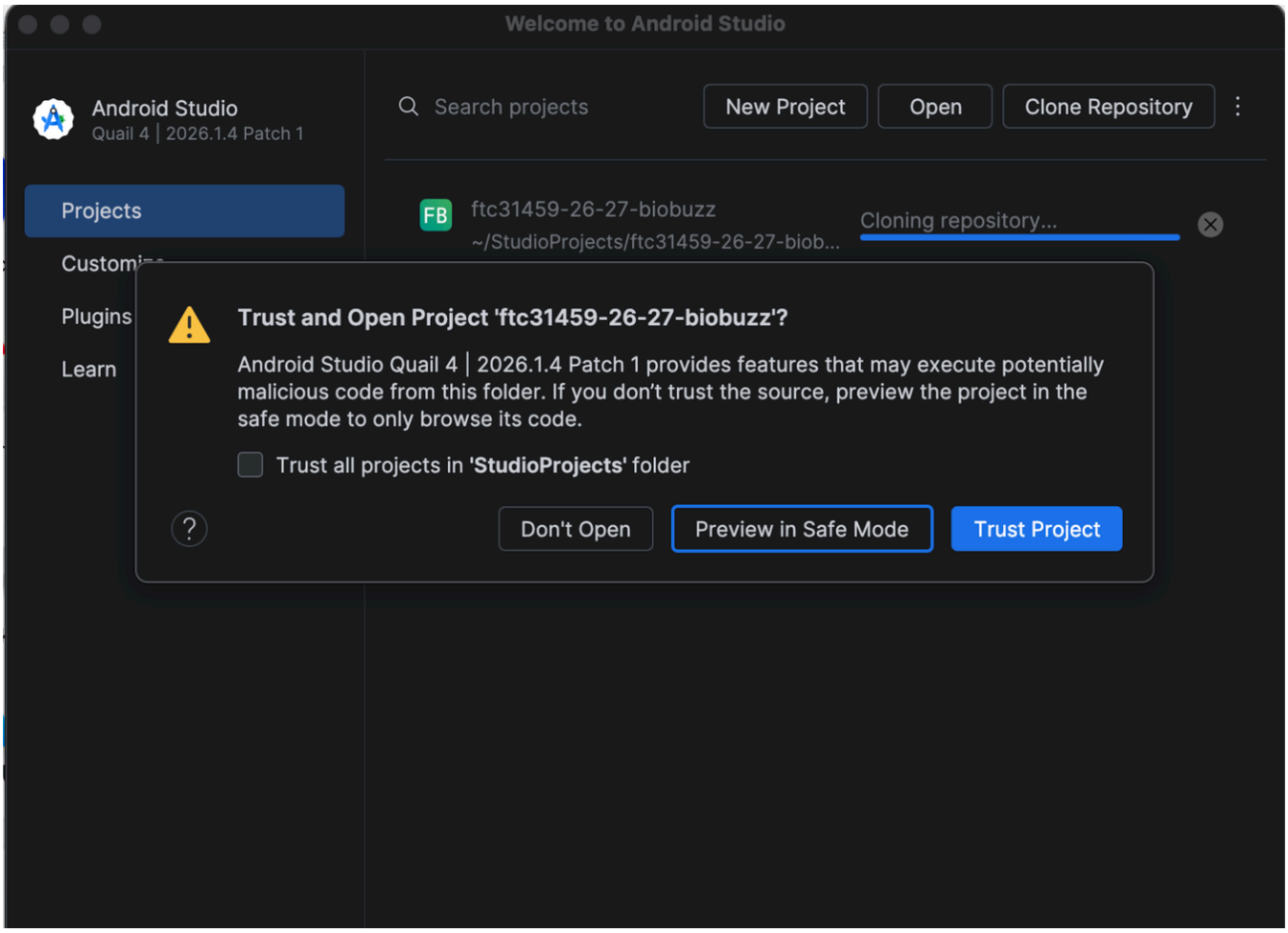
- On *Welcome to Android Studio*, click **Clone Repository**. It says *Git is not installed*.
- macOS asks whether to install the command line developer tools. Click **Install**, then **Agree**, and wait for it to finish.
- Quit Android Studio with **⌘Q**, and start it again from Applications.



Clone.

- On *Welcome to Android Studio*, click **Clone Repository**.
- Into **URL**, type `https://github.com/msb-stitt/ftc31459-26-27-biobuzz`. Leave **Directory** as it is.
- Click **Clone**.

Trust. When it asks *Trust and Open Project*, click **Trust Project**.



Check.

- The project opens, and the Project window lists `FtcRobotController` and `TeamCode`.

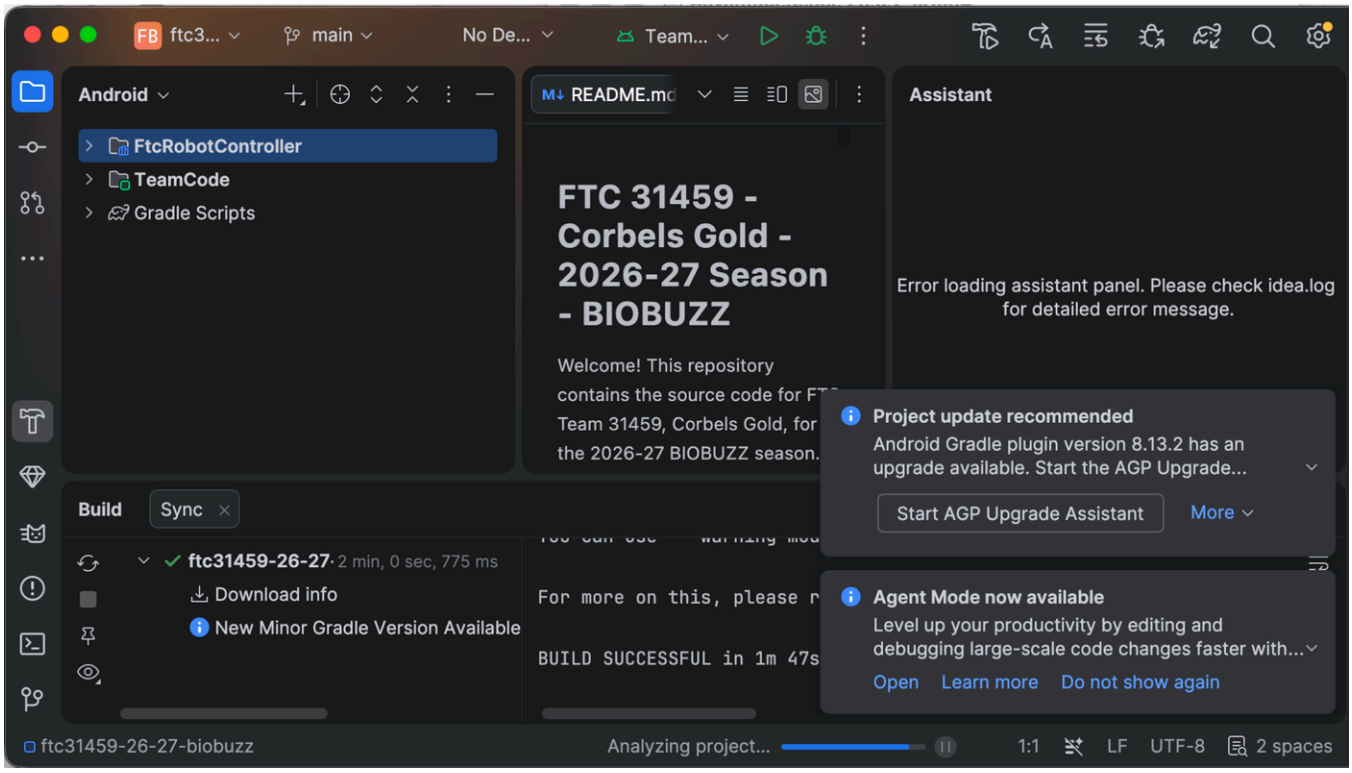
If it didn't, and *Git is not installed* stays after the tools finish, Android Studio has not looked again. Quit it with ⌘Q, not just its window, and start it again.

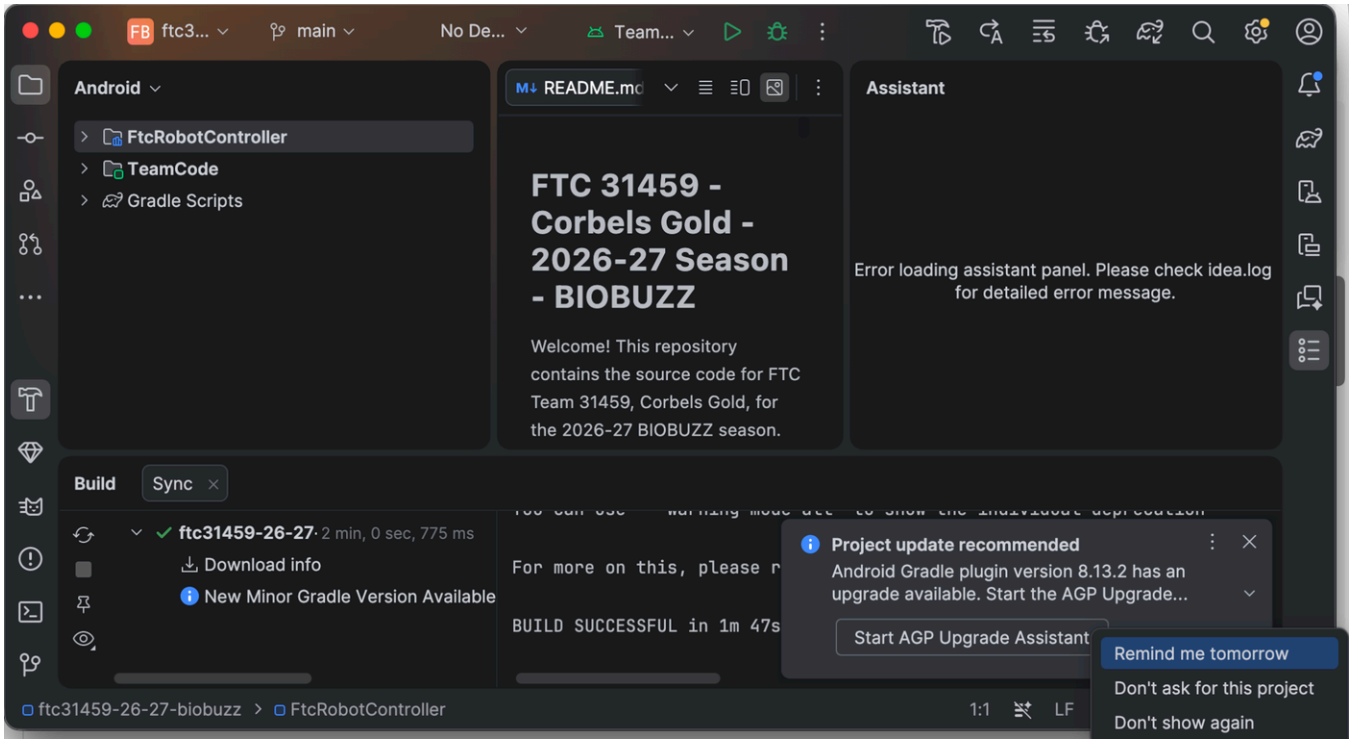
4.1.5 H012S050: answer what Android Studio offers

While the project loads, Android Studio reads it with **Gradle**, the tool that builds the code. This is called a *sync*. The first sync downloads Java 17 and more, and takes a few minutes.

Answer. Notes pop up in the bottom right while it works. Answer each one like this:

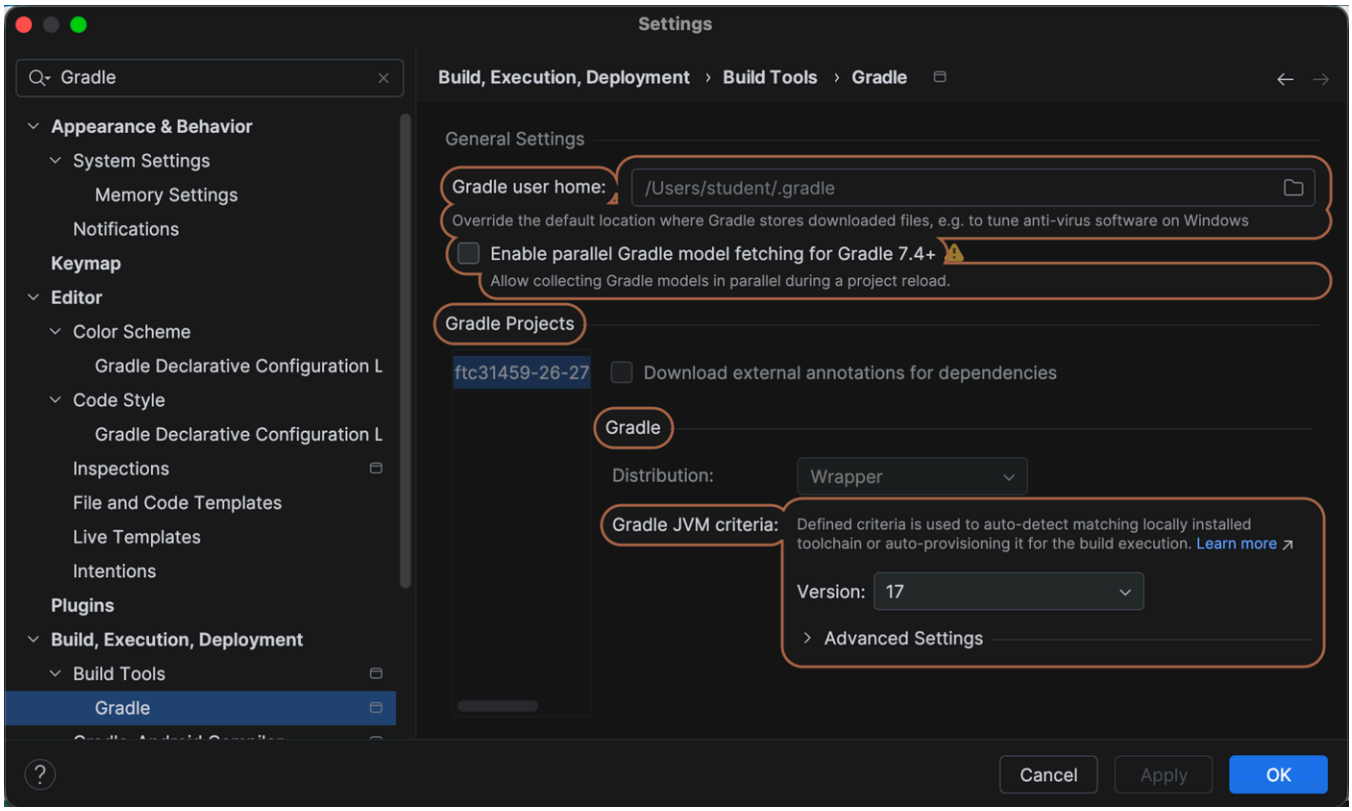
- **Agent Mode now available**: click **Do not show again**. The lessons do not use an AI helper.
- **Project update recommended**, *Android Gradle plugin version 8.13.2 has an upgrade available*: click **More**, then **Don't ask for this project**. The project needs version 8.13.2, so do not upgrade it.
- If macOS asks whether Android Studio may find devices on the local network, click **Allow**.


Android
FtcRobotController
TeamCode
Gradle Scripts
README.md
FTC 31459 - Corbels Gold - 2026-27 Season - BIOBUZZ
Welcome! This repository contains the source code for FTC Team 31459, Corbels Gold, for the 2026-27 BIOBUZZ season.
Assistant
Error loading assistant panel. Please check idea.log for detailed error message.
Project update recommended
Android Gradle plugin version 8.13.2 has an upgrade available. Start the AGP Upgrade...
Start AGP Upgrade Assistant
More
Agent Mode now available
Level up your productivity by editing and debugging large-scale code changes faster with...
Open
Learn more
Do not show again
Build
Sync
ftc31459-26-27 2 min, 0 sec, 775 ms
Download info
New Minor Gradle Version Available
For more on this, please r
BUILD SUCCESSFUL in 1m 47s
ftc31459-26-27-biobuzz
Analyzing project...
1:1
LF
UTF-8
2 spaces


Android
FtcRobotController
TeamCode
Gradle Scripts
README.md
FTC 31459 - Corbels Gold - 2026-27 Season - BIOBUZZ
Welcome! This repository contains the source code for FTC Team 31459, Corbels Gold, for the 2026-27 BIOBUZZ season.
Assistant
Error loading assistant panel. Please check idea.log for detailed error message.
Build
Sync
ftc31459-26-27 2 min, 0 sec, 775 ms
Download info
New Minor Gradle Version Available
For more on this, please r
BUILD SUCCESSFUL in 1m 47s
Project update recommended
Android Gradle plugin version 8.13.2 has an upgrade available. Start the AGP Upgrade...
Start AGP Upgrade Assistant
Remind me tomorrow
Don't ask for this project
Don't show again
ftc31459-26-27-biobuzz
FtcRobotController
1:1
LF

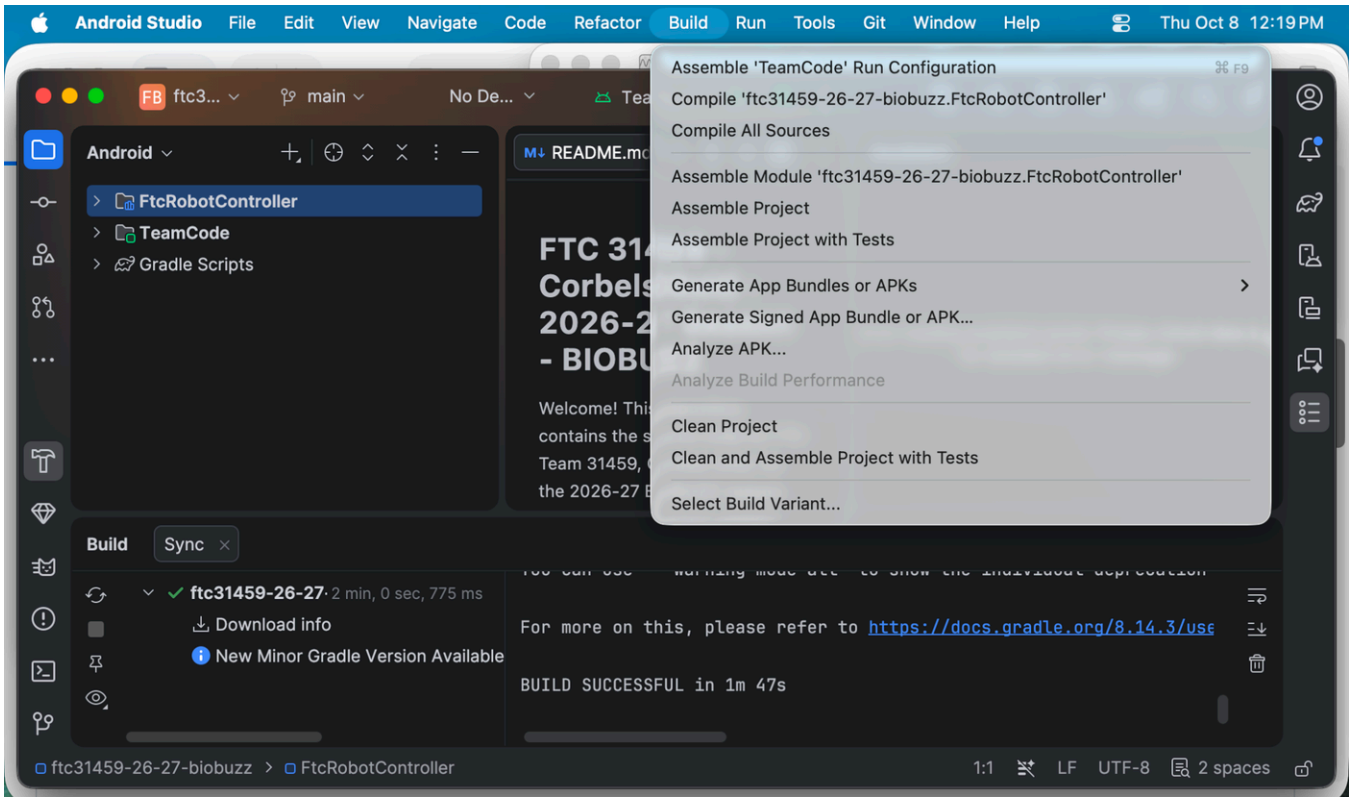
Check.

- The Build window's **Sync** tab has a green tick beside the project.
- Open **Android Studio → Settings**. Type `Gradle` in its search box and click **Gradle** under **Build Tools**. Under *Gradle JVM criteria,* **Version** reads `17`.



4.1.6 H012S060: build the code

Build. Click **Build → Assemble Project**. The first build takes a few minutes.



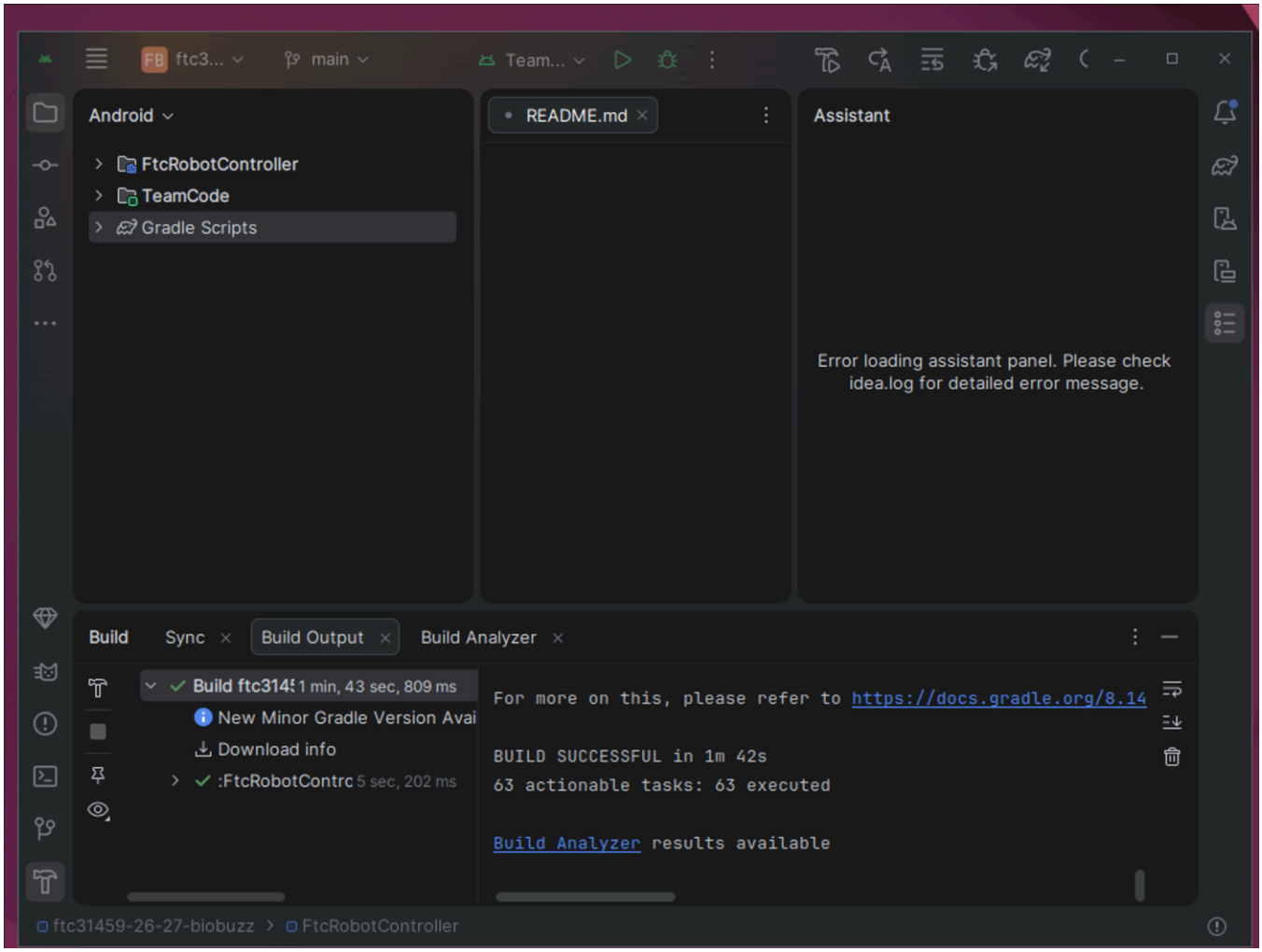
Check.

- The Build window's **Build Output** tab ends with *BUILD SUCCESSFUL* and *63 actionable tasks: 63 executed*.

What you just did. Android Studio is where the code is written, and Gradle turns the code into an app for the robot. Clone copied the team's repository from GitHub onto the laptop. The sync read the project's Gradle files and downloaded what they ask for: Gradle 8.14.3, the Android Gradle plugin 8.13.2 and Java 17 among them. Assemble Project built `FtcRobotController` and `TeamCode` into `TeamCode-debug.apk`, the app that is sent to the robot.

Where next. L020 copies the first starter into `mytry` and reads the gamepad's sticks.

013 how-to: get started on Ubuntu



Put the robot's code on an Ubuntu laptop and build it. After this, every lesson starts in Android Studio with the project open.

Before you start.

- An Ubuntu laptop, signed in to an account that may use `sudo`.
- The internet.
- No robot and no gamepad are needed.

Nothing else has to be installed first. Android Studio fetches Java 17 for the project by itself, and git is installed in L013S040.

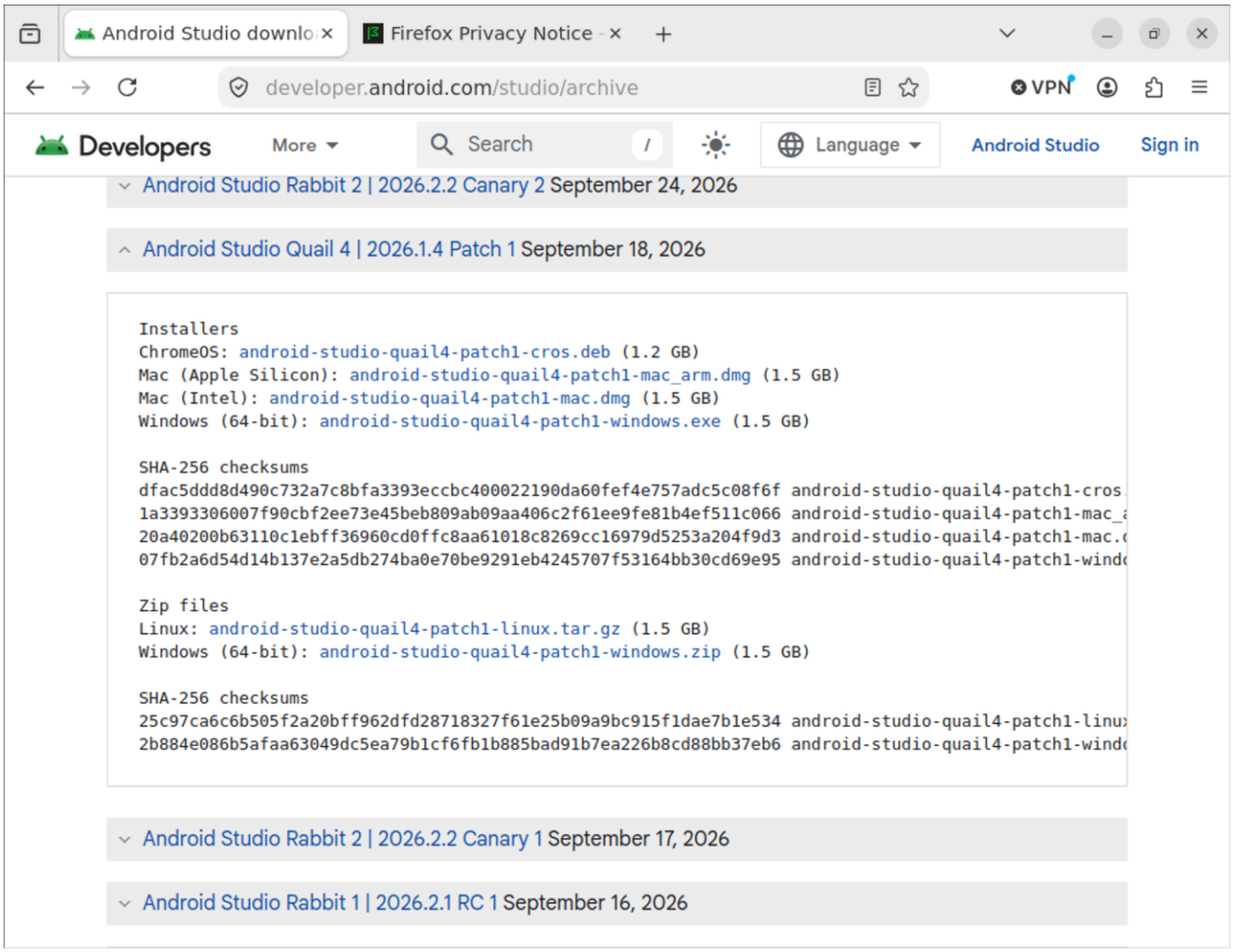
5.1 the steps

5.1.1 H013S010: install Android Studio

The robot's code is written in Android Studio. The team uses one version of it, **Quail 4**, so everyone sees the same screens. It is not the newest, so it comes from the page of older versions.

Download.

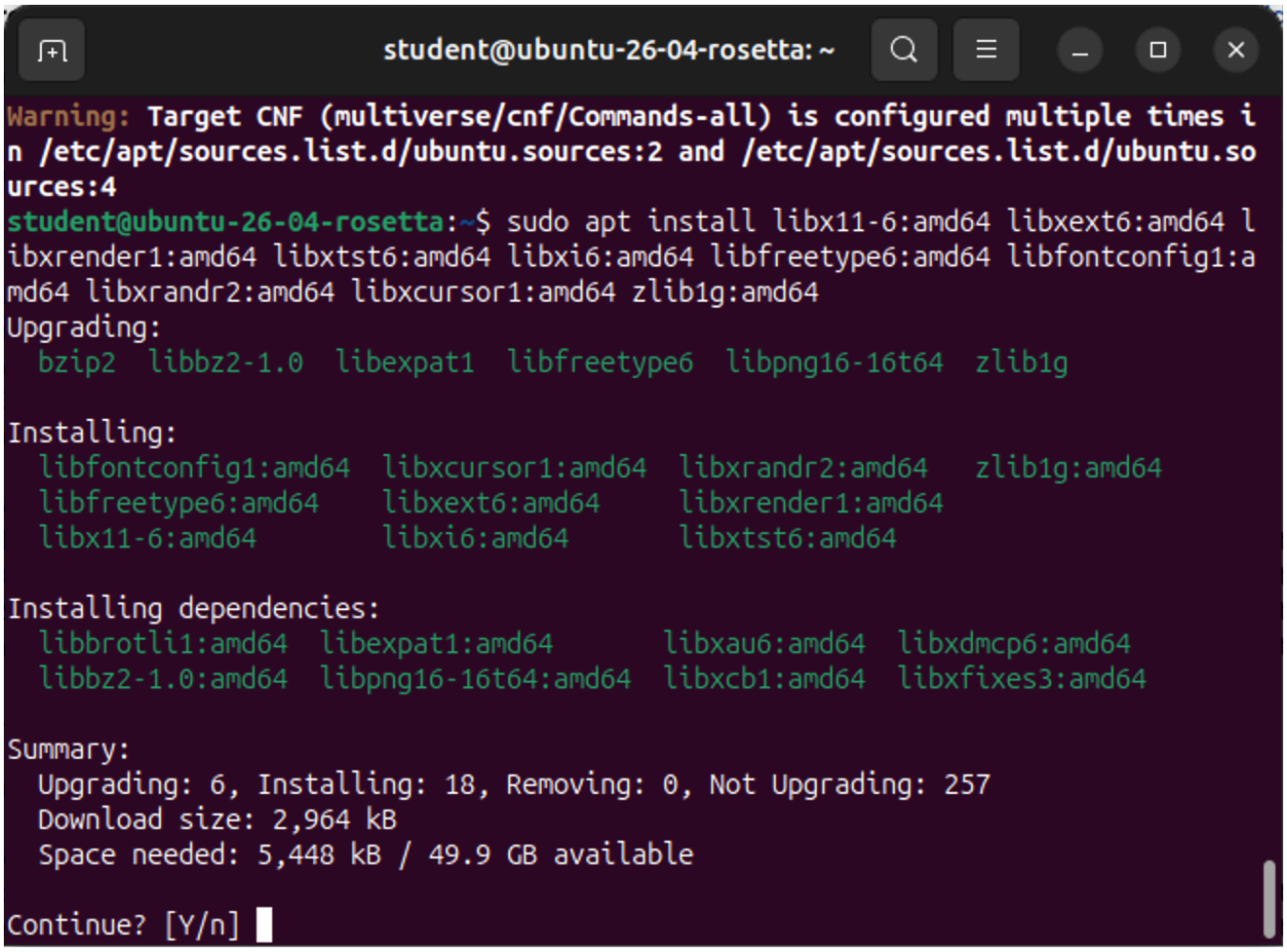
- Open https://developer.android.com/studio/archive and click **I agree** under the terms.
- Click **Android Studio Quail 4 | 2026.1.4 Patch 1** to open it.
- Click `android-studio-quail4-patch1-linux.tar.gz`. It is about 1.5 GB.



Install. Open Terminal and type these two lines, then your password when `sudo` asks:

```
sudo apt install libx11-6:amd64 libxext6:amd64 libxrender1:amd64 libxtst6:amd64 libxi6:amd64
libfreetype6:amd64 libfontconfig1:amd64 libxrandr2:amd64 libxcursor1:amd64 zlib1g:amd64
sudo tar -xzf ~/Downloads/android-studio-quail4-patch1-linux.tar.gz -C /opt
```

Then start Android Studio with `/opt/android-studio/bin/studio`. Leave that Terminal open while you work.



Check.

- Android Studio starts and asks to *Help improve Android Studio*. Click **Don't send**.

If it didn't, and Android Studio closes a few seconds after it starts with *Bus error (core dumped)* in Terminal, turn off its Gemini plugin and start it again:

```
mkdir -p ~/.config/Google/AndroidStudio2026.1.4
echo com.google.tools.ij.aiplugin >> ~/.config/Google/AndroidStudio2026.1.4
/disabled_plugins.txt
```

`bin/studio.sh` stops with *No JRE found*; start `bin/studio` instead.

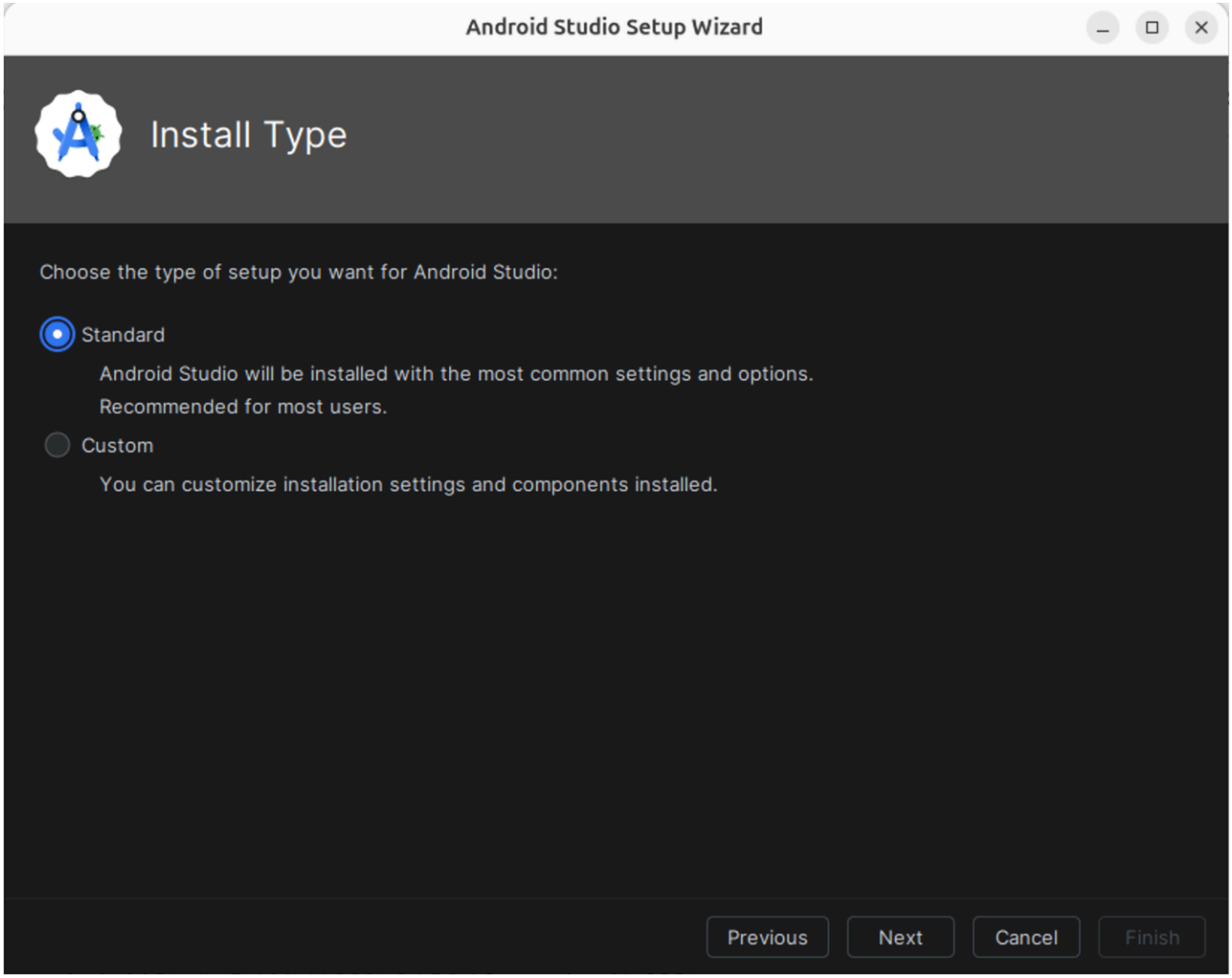
5.1.2 H013S020: let Android Studio fetch what it needs

The first time it starts, Android Studio's setup wizard downloads the Android tools the code is built with.

Setup.

- Click **Next** on *Welcome*.
- Pick **Standard** and click **Next**.

- Click **Next** on *Verify Settings*.
- Click each license on the left, pick **Accept**, and click **Next**.
- A page about KVM and the emulator comes next. Click **Next**.
- Wait while it downloads, then click **Finish**.



Check.

- The last page of the wizard reads *Android SDK is up to date*.
- *Welcome to Android Studio* opens, with **New Project**, **Open** and **Clone Repository**.

If a note says *Rabbit 1 available*, leave it. The team stays on Quail 4.

5.1.3 H013S030: install AdvantageScope

AdvantageScope draws what the robot is doing: where it is on the field, and graphs of its numbers.

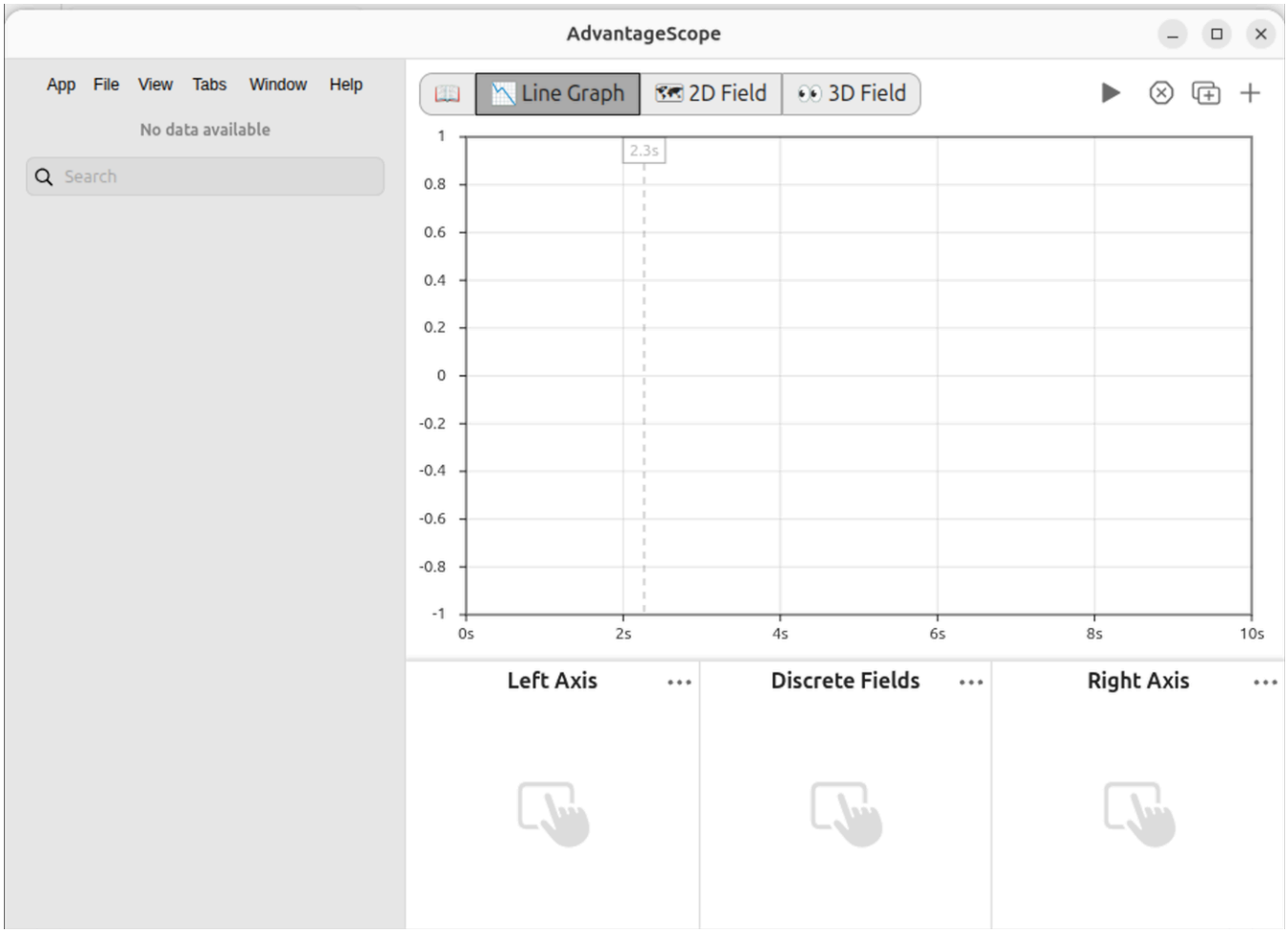
Download. Open https://github.com/Mechanical-Advantage/AdvantageScope/releases/tag/v26.0.2. Under

Assets, click `advantagescope-linux-x64-v26.0.2.deb`.

Install. In Terminal, type `sudo apt install ~/Downloads/advantagescope-linux-x64-v26.0.2.deb`. Then find AdvantageScope in the app grid and start it.

Check.

- AdvantageScope opens on its **Line Graph** tab, with *No data available* on the left.



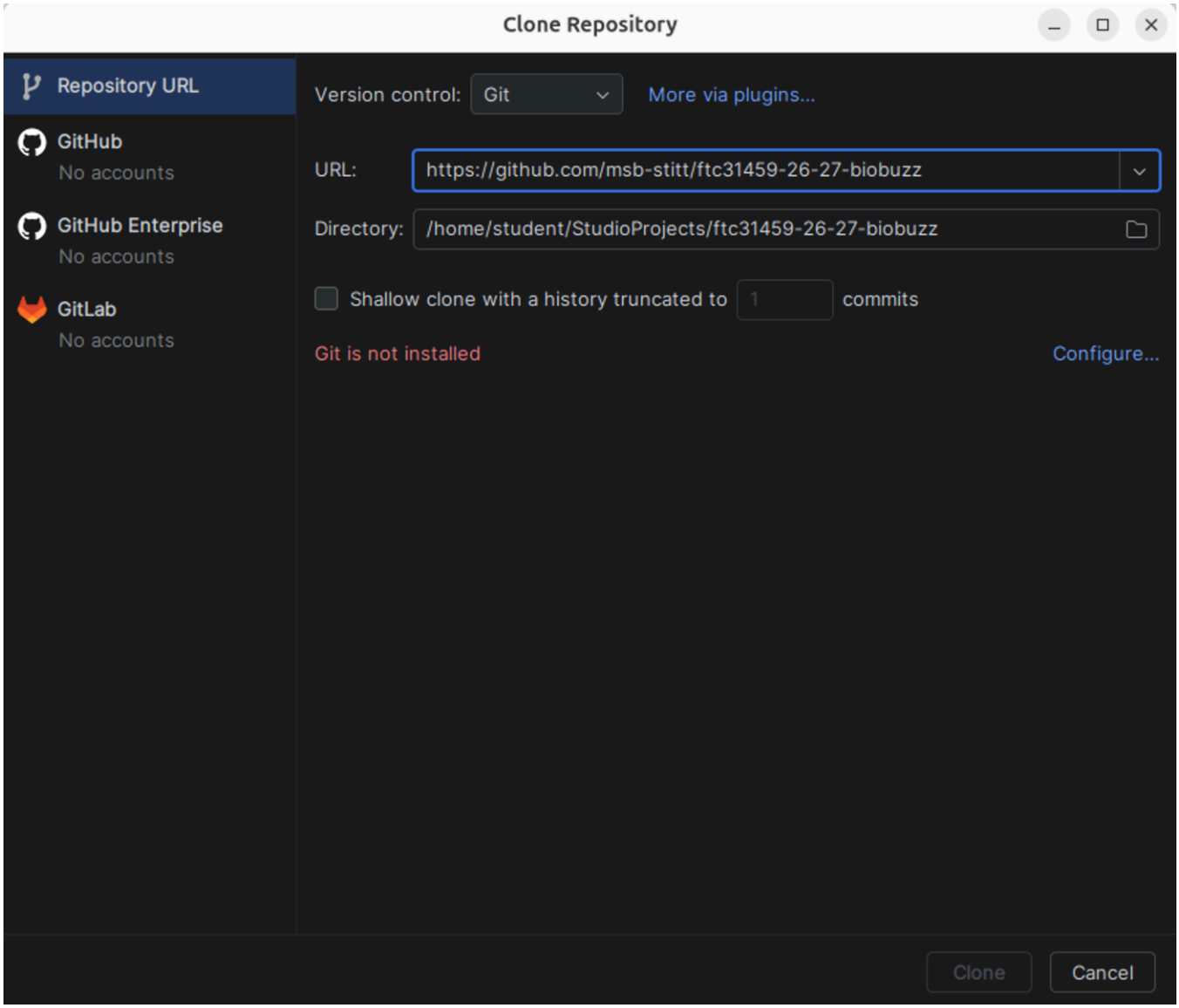
If it didn't, and the first start says *Failed to load assets*, click **Ok**, close AdvantageScope and start it again.

5.1.4 H013S040: get the project

The robot's code lives on GitHub. Android Studio copies it to the laptop with a tool called **git**.

Git.

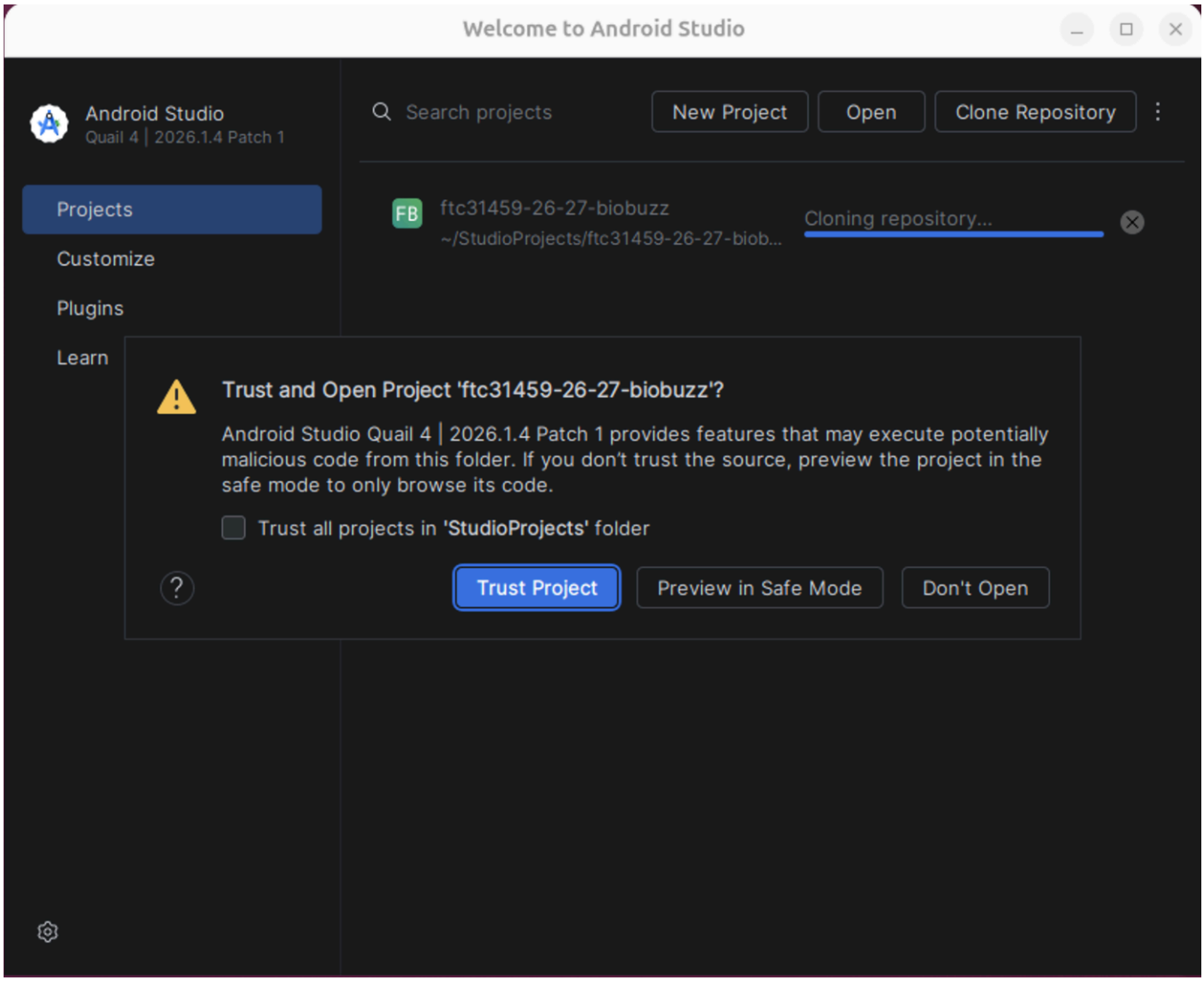
- On *Welcome to Android Studio*, click **Clone Repository**. It says *Git is not installed*. Close the box.
- In Terminal, type `sudo apt install git`.
- Close Android Studio, and start it again with `/opt/android-studio/bin/studio`.



Clone.

- On *Welcome to Android Studio*, click **Clone Repository**.
- Into **URL**, type `https://github.com/msb-stitt/ftc31459-26-27-biobuzz`. Leave **Directory** as it is.
- Click **Clone**.

Trust. When it asks *Trust and Open Project*, click **Trust Project**.



Check.

- The project opens, and the Project window lists `FtcRobotController` and `TeamCode`.

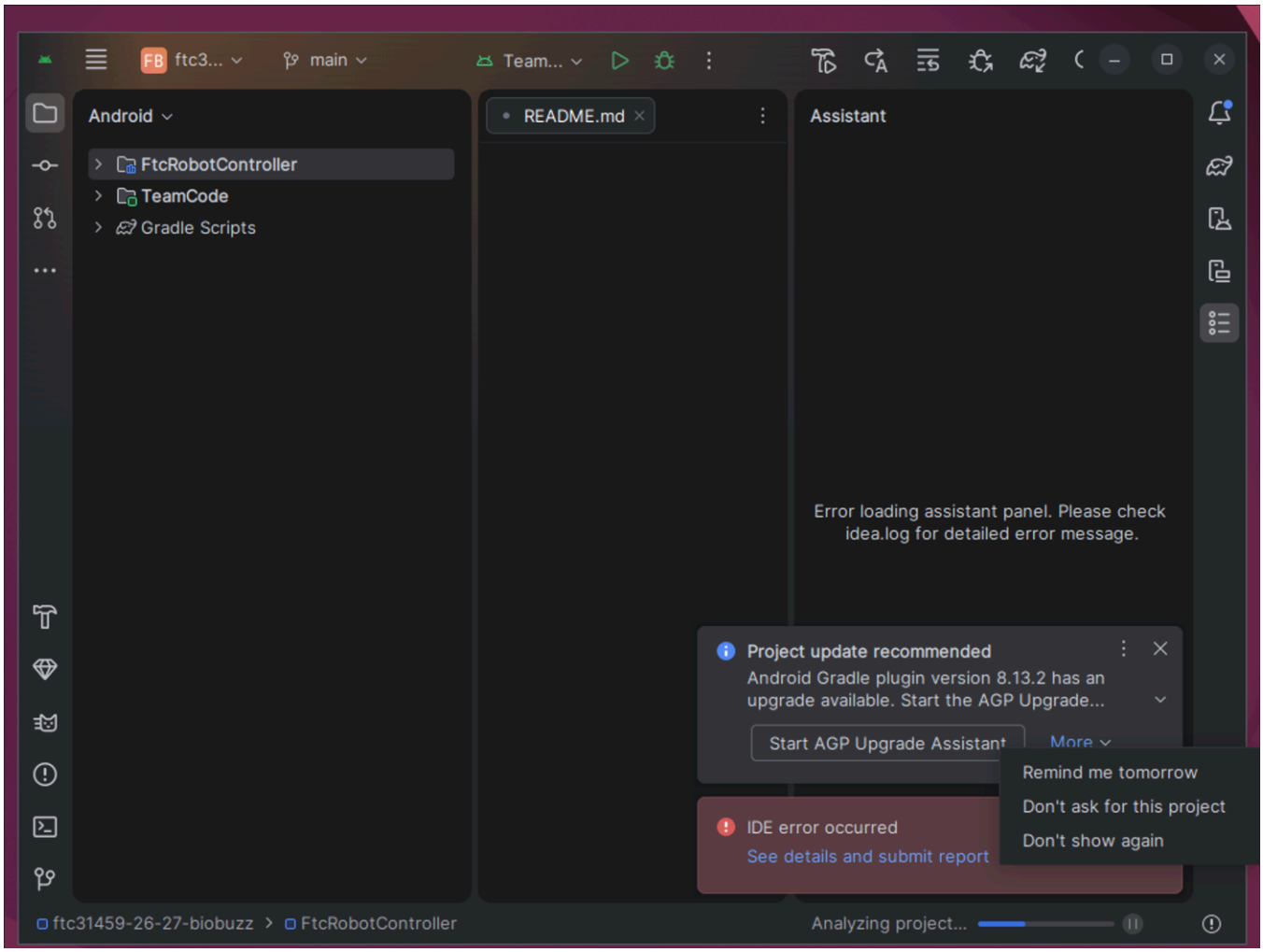
If it didn't, and Android Studio closes with *Bus error (core dumped)* while the project first loads, start it again, click **Open**, pick `~/StudioProjects/ftc31459-26-27-biobuzz`, and trust it again.

5.1.5 H013S050: answer what Android Studio offers

While the project loads, Android Studio reads it with **Gradle**, the tool that builds the code. This is called a *sync*. The first sync downloads Java 17 and more, and takes a few minutes.

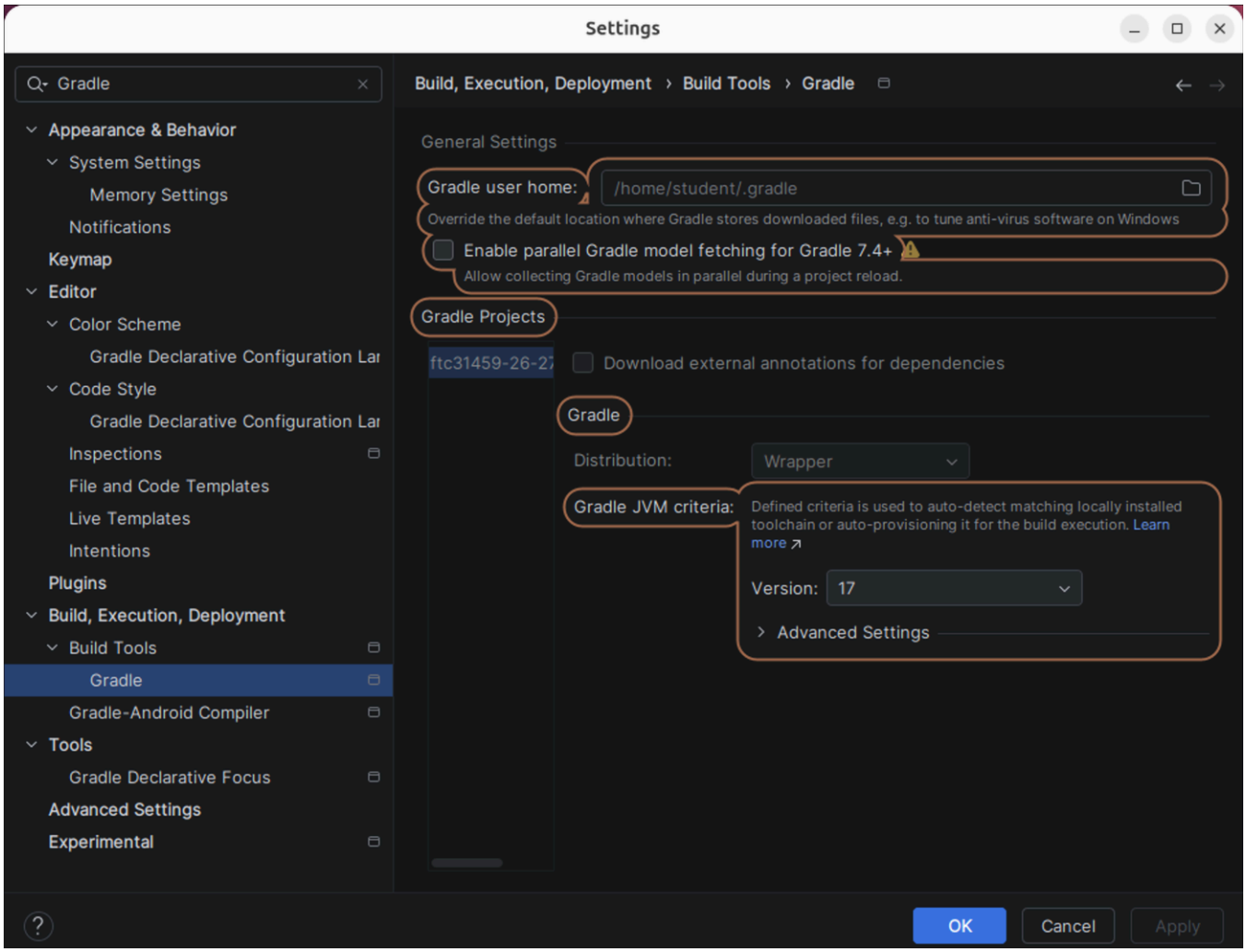
Answer. Notes pop up in the bottom right while it works. Answer each one like this:

- **Agent Mode now available**: click **Do not show again**. The lessons do not use an AI helper.
- **Project update recommended**, *Android Gradle plugin version 8.13.2 has an upgrade available*: click **More**, then **Don't ask for this project**. The project needs version 8.13.2, so do not upgrade it.



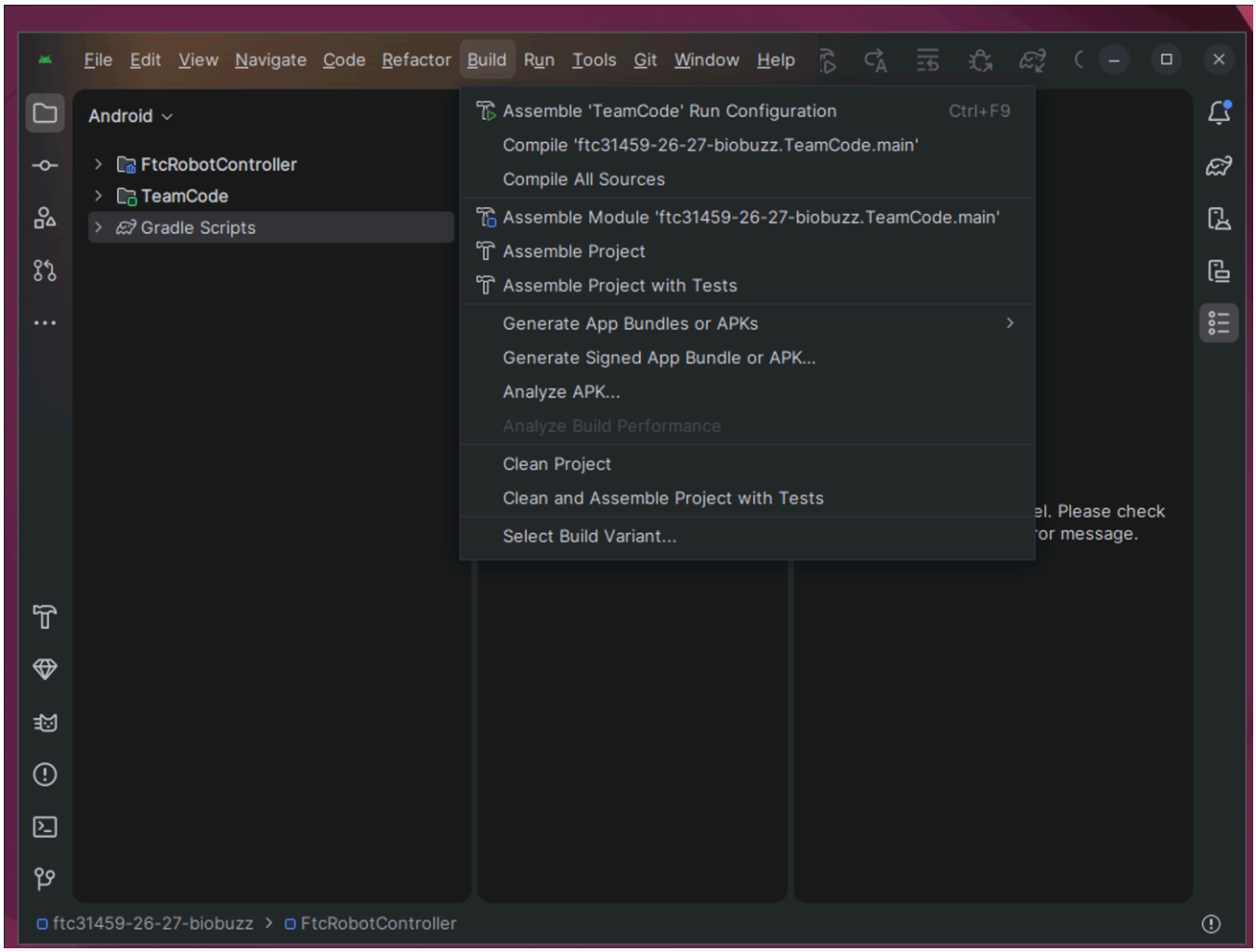
Check.

- The Build window's **Sync** tab has a green tick beside the project.
- Open **File → Settings**. Type `Gradle` in its search box and click **Gradle** under **Build Tools**. Under *Gradle JVM criteria*, **Version** reads `17`.



5.1.6 H013S060: build the code

Build. Click **Build → Assemble Project**. The first build takes a few minutes.



Check.

- The Build window's **Build Output** tab ends with *BUILD SUCCESSFUL* and *63 actionable tasks: 63 executed*.

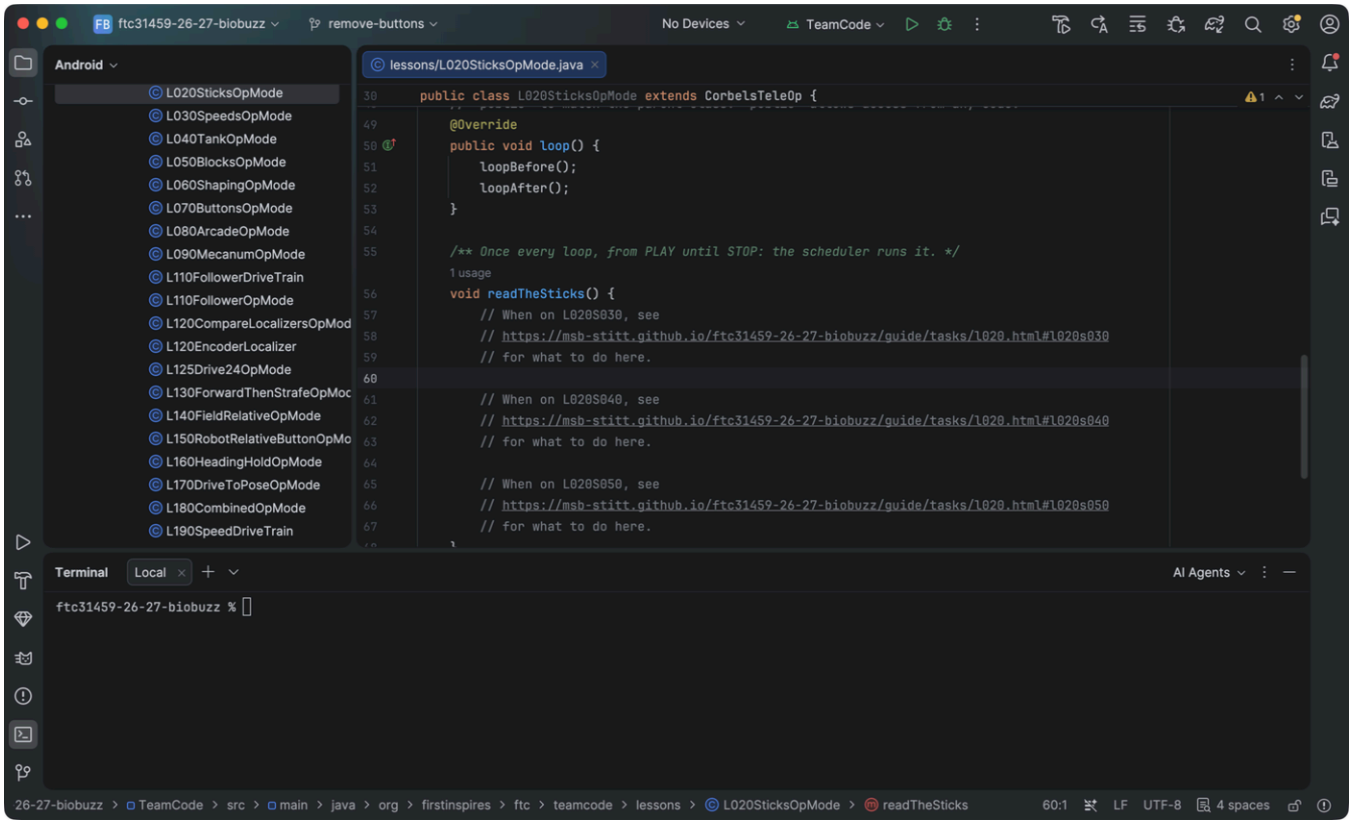
What you just did. Android Studio is where the code is written, and Gradle turns the code into an app for the robot. Clone copied the team's repository from GitHub onto the laptop. The sync read the project's Gradle files and downloaded what they ask for: Gradle 8.14.3, the Android Gradle plugin 8.13.2 and Java 17 among them. Assemble Project built `FtcRobotController` and `TeamCode` into `TeamCode-debug.apk`, the app that is sent to the robot.

Where next. L020 copies the first starter into `mytry` and reads the gamepad's sticks.

017 how-to: navigate Android Studio's panes

Before you start.

- B010 is done: Android Studio has the project open.



A video shows every part of this page: https://msb-stitt.github.io/ftc31459-26-27-biobuzz/dev/videos/h017.mp4.

Android Studio's window has these parts:

- **Toolbar**, along the top: runs the code and starts Android tools.
- **Editor**, in the middle: where you read and write the code.

- **Tool windows**, such as Project on the left and Terminal below: each is a pane that does one kind of job.
- **Tool window bar**, the buttons down each side: each one opens or closes a tool window.
- **Status bar**, along the bottom: what Android Studio is doing. Its left end is the path to the open file. Left-click a folder in the path to see what it holds.

Click.

- Left-click the folder button at the top of the left side.
- Left-click it again.
- Left-click the Terminal button, near the bottom of the left side.

Check.

- The first left-click closes the Project window, and the second opens it again.
- The Terminal opens below the editor, and its button is lit.

Drag.

- Left-click and drag the edge between Project and the editor to the right.
- Left-click and drag the top edge of the Terminal up or down.

Check.

- Each name in the Project window shows in full.
- The Terminal grows or shrinks.

Theme.

- Left-click the gear at the top right, then **Theme…**.
- Left-click **Islands Light**.

Check.

- Android Studio turns light. To go back, pick **Islands Dark** the same way.

The pictures in this book use Islands Dark.

View.

- Left-click the name at the top of the Project pane, then **Project**.
- Do it again and pick **Project Source Files**, then again and pick **Android**.

Check.

- **Project** shows every file in the project's folder, even the book.
- **Project Source Files** shows only the folders that hold code.
- **Android** shows three parts:
    - **FtcRobotController**, where the FTC robot platform code lives. It has features that make robots easy to program.
    - **TeamCode**, where our code lives.
    - **Gradle Scripts**, the scripts that build and test the code.

Find.

- In **Project**, open **TeamCode**, then **src**, **main** and **java**. **org.firstinspires.ftc.teamcode** may open with **java**. If it stays closed, open it too.
- In **Android**, open **TeamCode**, then **java** and **org.firstinspires.ftc.teamcode**.

Check.

- Each way ends at the same folders, among them **lessons** and **mytry**.

Sync.

- Left-click **File**, then **Sync Project with Gradle Files**.
- Left-click the **Build** button, the hammer near the bottom of the left side, then the **Sync** tab.

Check.

- The **Sync** tab shows a green check by the project's name.

Syncing makes Gradle, inside Android Studio, figure out the steps it takes to build the code. Sync again after you get new code from the internet.

Target.

- Left-click the list just left of the green triangle at the top, then **TeamCode**.

Check.

- The list reads **TeamCode**.

Build.

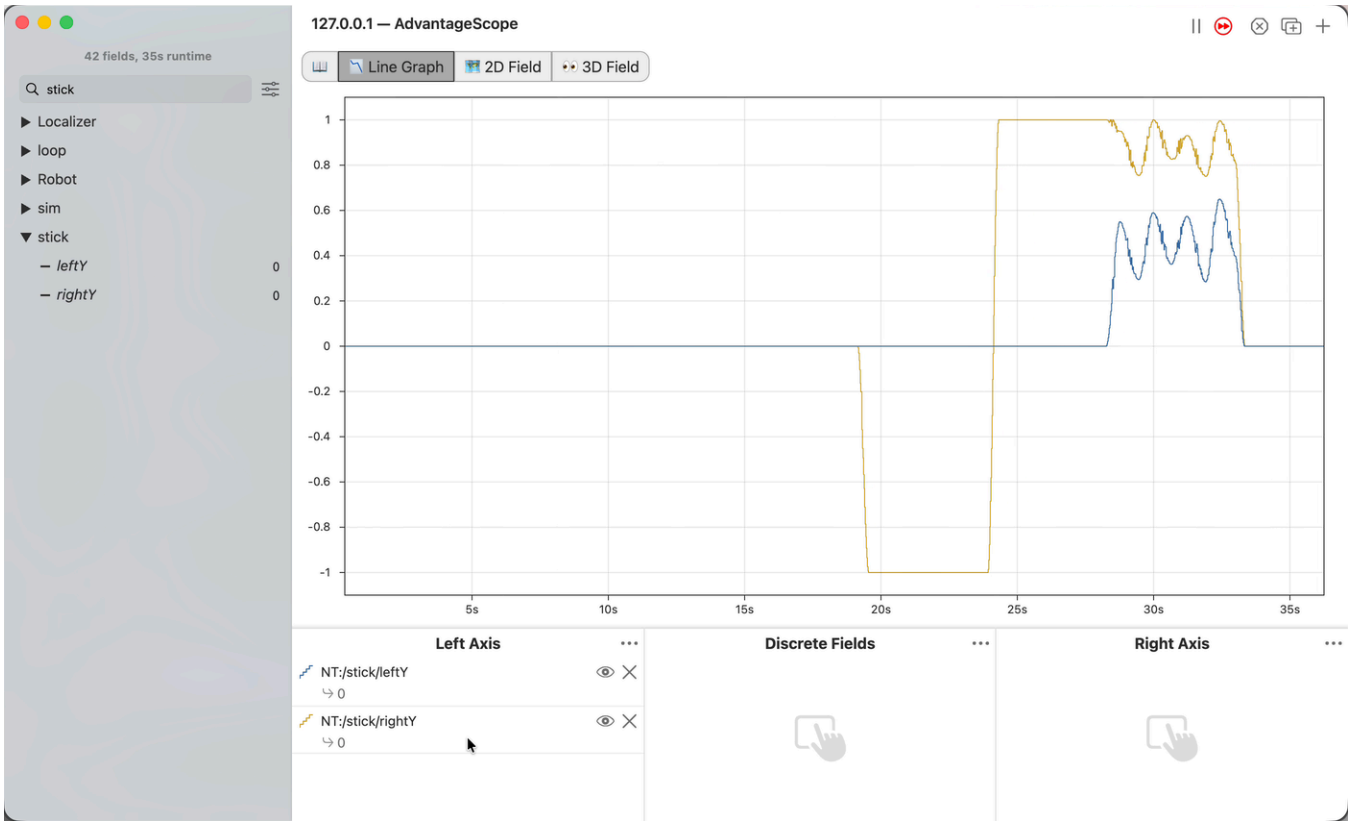
- Left-click the hammer at the top right of the toolbar.
- Left-click the **Build Output** tab in the **Build** pane.

Check.

- The tab shows a green check and `BUILD SUCCESSFUL`.

When the build fails, the mark is red. Open the list under it to find the file, what is wrong, and the line, as `';' expected :4`.

020 lesson: read the sticks



Push the gamepad's sticks and watch the numbers they send, live on the laptop. Nothing on the robot moves yet.

A video shows every step of this lesson: https://msb-stitt.github.io/ftc31459-26-27-biobuzz/dev/videos/l020.mp4.

Before you start.

- B010 is done: Android Studio has the project open.
- H017 is done: the student can open, close and resize Android Studio's panes.
- AdvantageScope is installed on the same laptop.
- A gamepad is plugged into the laptop.

- No robot is needed.

The student writes in one folder, `mytry`, inside `TeamCode/java/org.firstinspires.ftc.teamcode`. It holds no lesson until step L020S010 copies a starter into it. The `lessons` folder beside it holds the starting files.

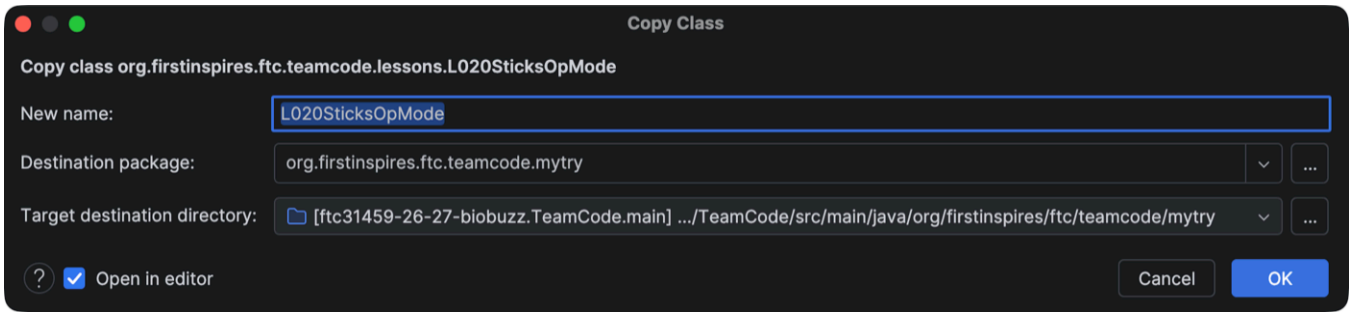
7.1 the steps

7.1.1 L020S010: copy the starter into mytry

The `lessons` folder holds the starters: the file each lesson begins from. You copy one into your own folder, `mytry`, and change the copy.

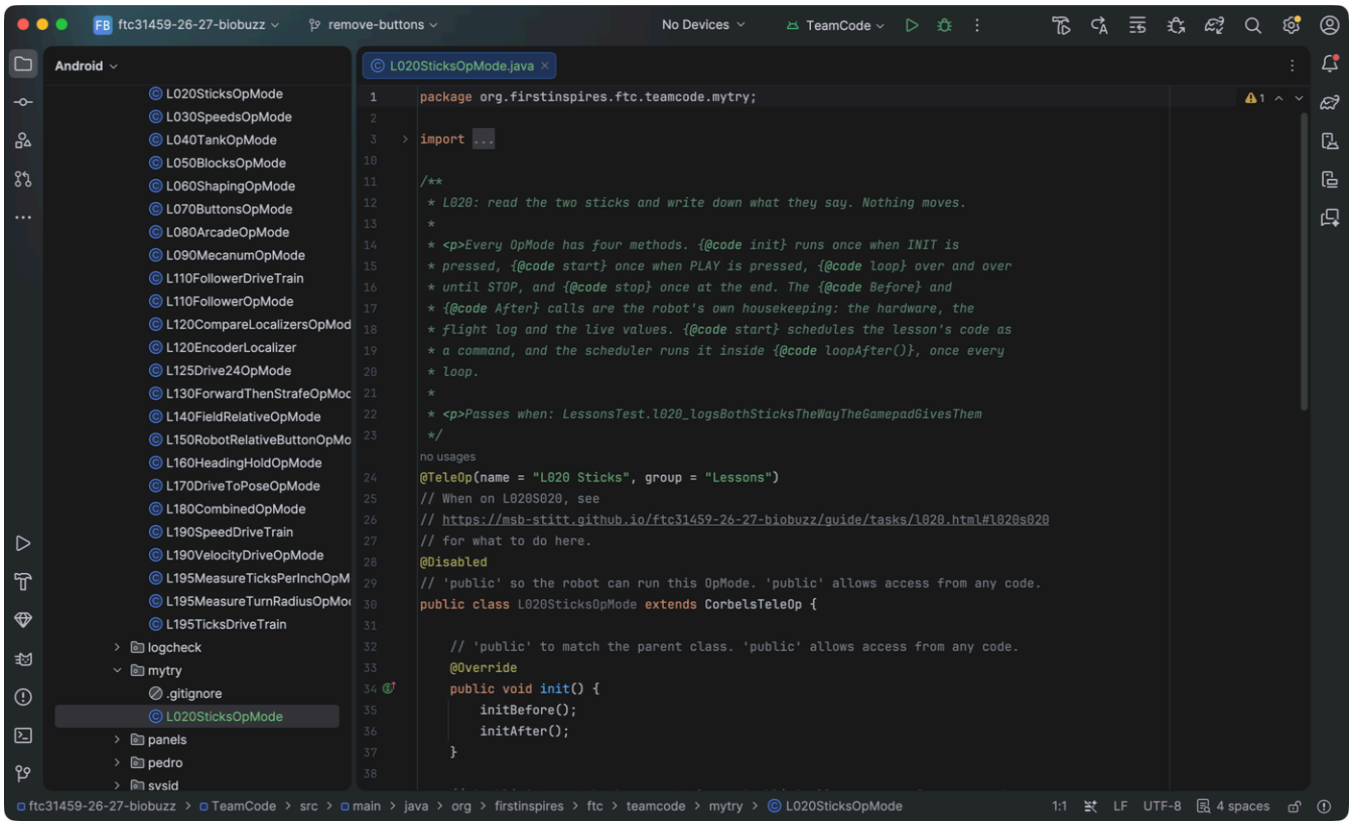
Copy.

- Open `TeamCode/java/org.firstinspires.ftc.teamcode/lessons`.
- Click `L020SticksOpMode` once, and copy it: ⌘C on a Mac, Ctrl+C on Windows.
- Click the `mytry` folder once, and paste.
- A box called **Copy Class** opens with **Destination package** already ending in `mytry`. Click **OK**.
- If Android Studio asks whether to add the file to Git, click **Add**.



Check.

- `mytry` holds `L020SticksOpMode`, the file is open, and its first line reads `package org.firstinspires.ftc.teamcode.mytry;`.



If it didn't, and the copy landed in `lessons` beside the starter, `lessons` was the folder clicked before pasting, so Destination package ended in `lessons`. Delete the copy, click `mytry`, and paste again.

7.1.2 L020S020: switch your copy on

Change. Find the line `@Disabled`, just under the comment that names this step. Click on it and delete the whole line: ⌘Delete on a Mac, Ctrl+Y on Windows and Ubuntu. The first time on Windows, Android Studio asks whether Ctrl+Y should Redo or Delete Line: click **Delete Line**.

`@Disabled` hides an OpMode from the robot's list. The starter in `lessons` keeps it, so the robot only ever lists your copy.

Check.

- Under the step's comment, the next lines read `// 'public' so the robot can run this OpMode. 'public' allows access from any code.` and `public class L020SticksOpMode`.

7.1.3 L020S030: read the left stick

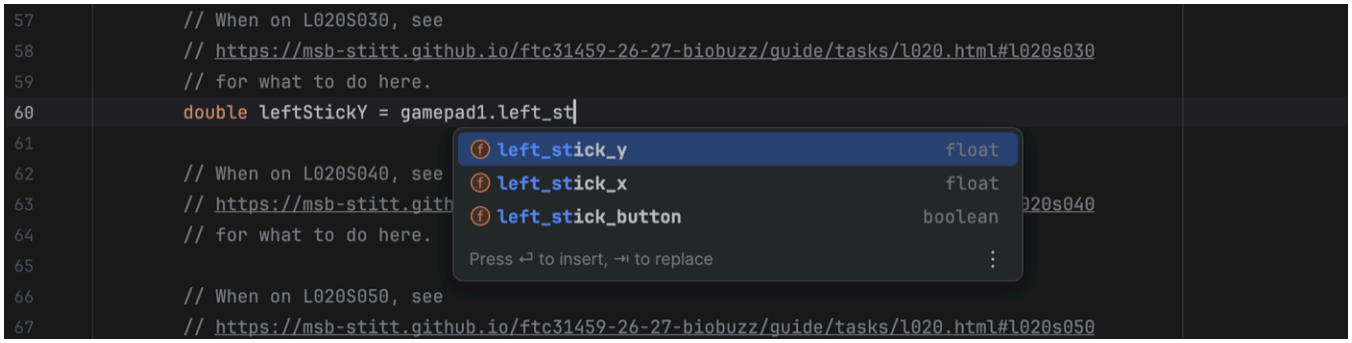
The file is an **OpMode**: one program the robot can run. Pressing INIT runs its `init()` once. Pressing PLAY runs its `start()` once, and `start()` hands `readTheSticks()` to the **scheduler**. From then until STOP, the scheduler runs `readTheSticks()` over and over, many times a second. The simulator presses all three for you.

Change.

- Scroll down to `readTheSticks()` and find the comment that names this step.
- Click at the end of its last line and press Enter. Type this much:

```
double leftStickY = gamepad1.
```

- A list opens with everything a gamepad has. Type `left_st` and the list shrinks to the left stick.
- `left_stick_y` is already highlighted. Press Tab. If it isn't, move the highlight to it with the arrow keys first.
- Type `;` to end the line.




```
        // When on L020S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l020.html#l020s030
        // for what to do here.
        double leftStickY = gamepad1.left_stick_y;
```

`double` is Java's word for a number with a decimal point, like 0.5 or -1.0. `leftStickY` is a name you chose for one. The list said `float`, which is a smaller kind of decimal number. A `double` holds any `float`.

Now hold the pointer still over `left_stick_y`. A box opens and says what it is: *left analog stick vertical axis*. You can do this with any name you are not sure about.



Check.

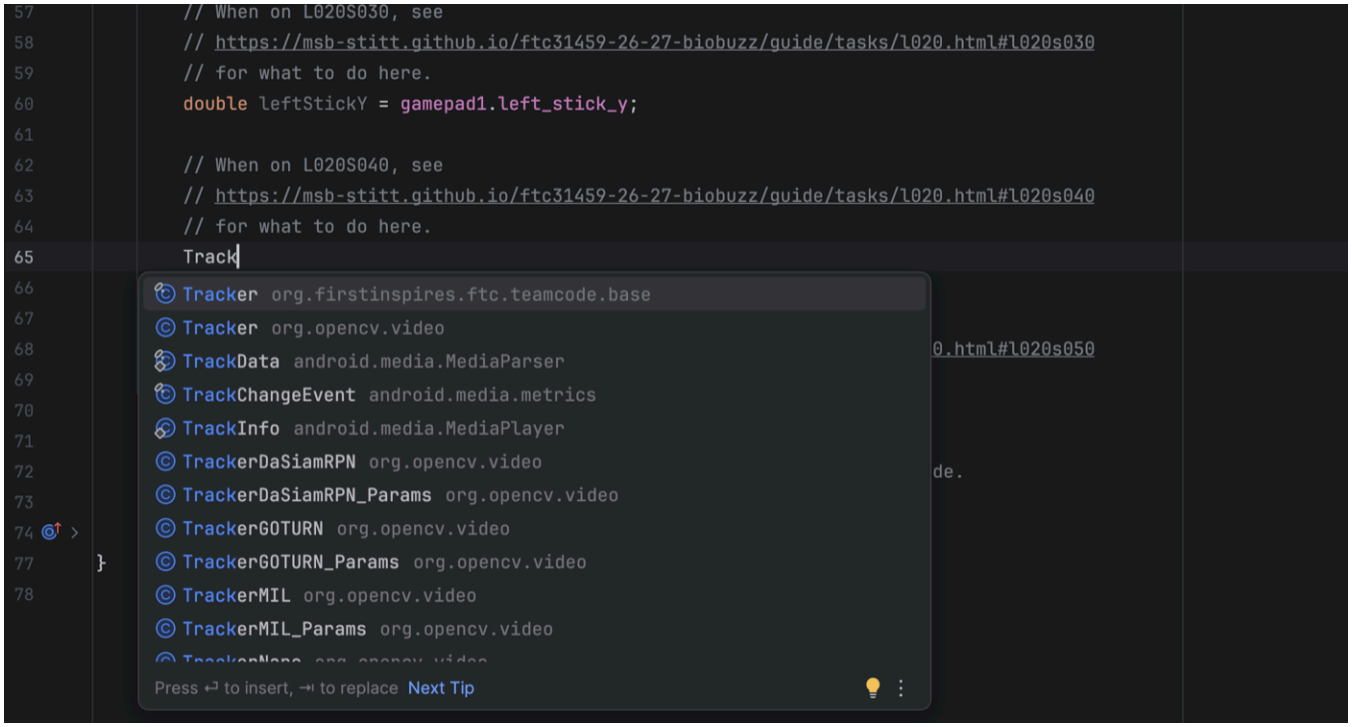
- The line has no red underline.

If it didn't, and the line reads `left_stick_x`, the highlight was on the wrong name when Tab was pressed. Change the letter by hand.

7.1.4 L020S040: write the left stick down

Change. Under the comment that names this step, type `Track`. The list shows more than one `Tracker`. Pick the one from `org.firstinspires.ftc.teamcode.base` and press Tab. Then finish the line:

```
        // When on L020S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l020.html#l020s040
        // for what to do here.
        Tracker.publish("stick/leftY", leftStickY);
```



`Tracker.publish` takes a name and a value. It sends both to AdvantageScope while the code runs, and writes them in the flight log. Android Studio added an `import` line for `Tracker` near the top of the file.

Check.

- `Tracker` is not red, and there is an `import org.firstinspires.ftc.teamcode.base.Tracker;` line near the top.

The simulator is a robot that exists only in the laptop. It runs your OpMode just as the real robot would.

Simulator. Click the **Terminal** button near the bottom of the left edge. Type this and press Enter:

```
./gradlew :TeamCode:simRun --args="mytry.L020SticksOpMode --pad"
```

Check.

- It lists any connected gamepads.
- Then it prints `Flight log:` and a folder, then `mytry.L020SticksOpMode running.`, and stays running.
- The folder is the project's top folder, the one that holds `README.md`.

If more than one gamepad is plugged in, hold Start and press A on yours, so it becomes `gamepad1`.

AdvantageScope draws the numbers your code publishes, as they change.

AdvantageScope.

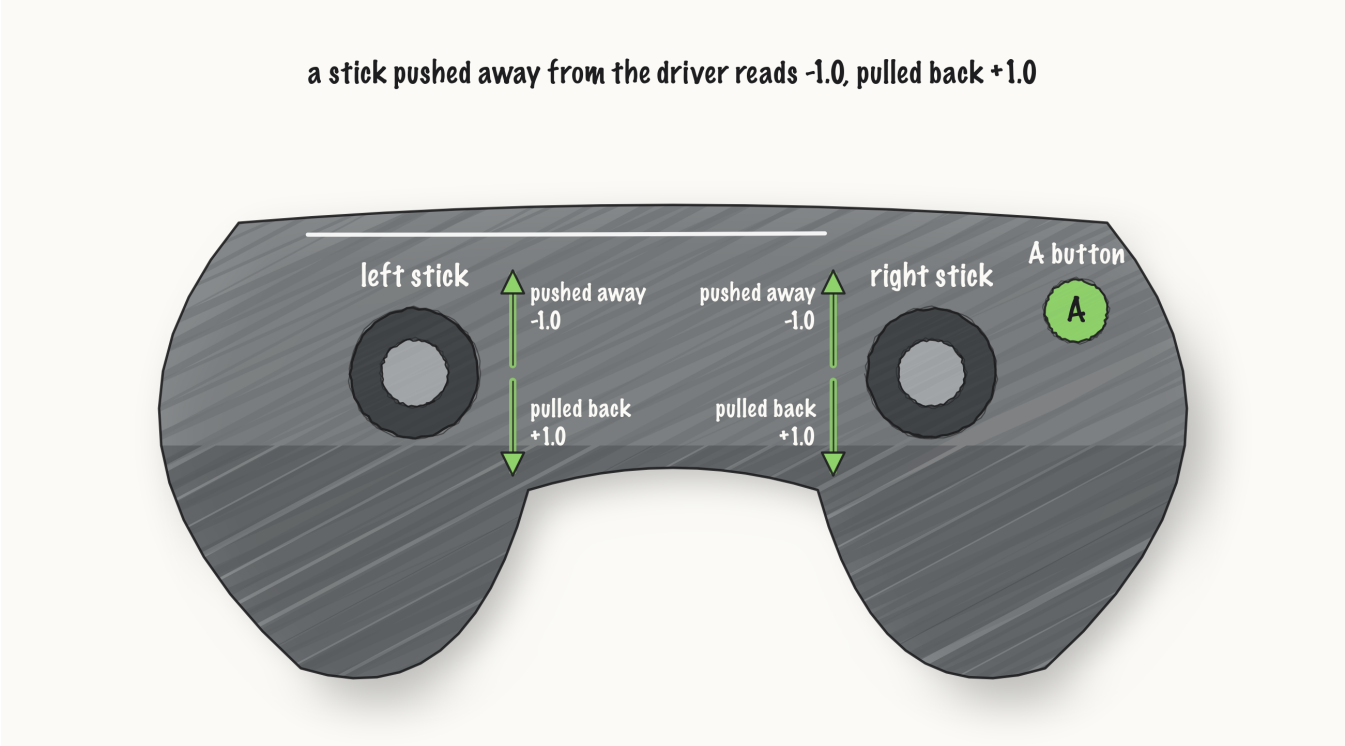
- Open AdvantageScope. It is an app of its own, not a part of Android Studio.
- Open **File > Connect to Simulator > NetworkTables 4**. The item marked **Default** works the same. Then click **Line Graph**.
- Type `stick` in the search box at the top left, and click `NT:/stick/leftY`.
- Drag `leftY` from the `stick` folder down into **Left Axis**.

Before you push the stick, guess. When you push the left stick all the way away from you, will the line go to 1.0 or to -1.0? Now push it and see!

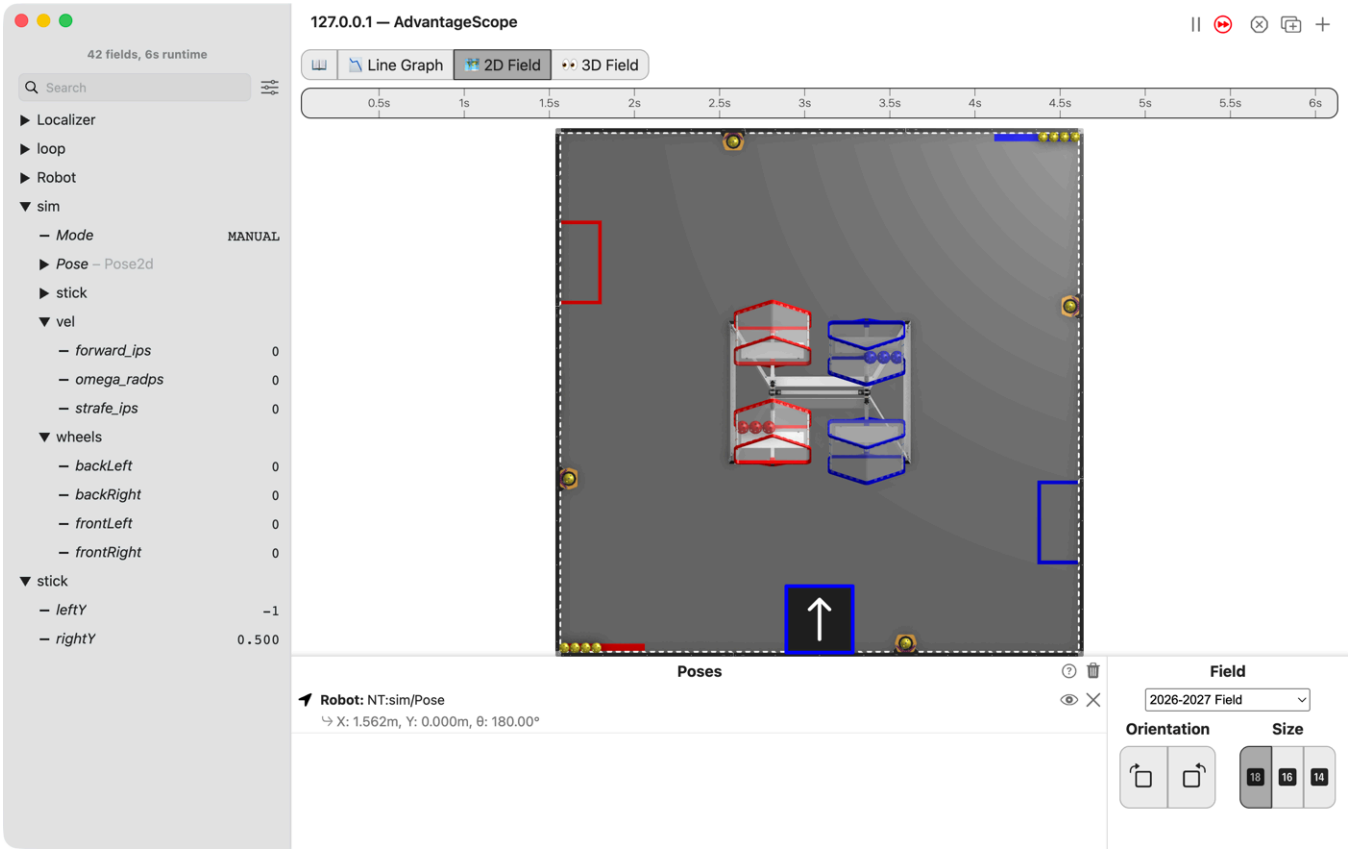
Check.

- The line moves when you push.
- All the way forward it reads -1.0, and all the way back it reads 1.0.

Forward is -1.0 on gamepads. That is how most gamepads are made, not a mistake in your code.



The list in AdvantageScope has two `stick` folders. The one inside `sim` is the gamepad as the simulator read it; the one on its own is what the student's code published. In L020 they read the same, because the code passes the stick on unchanged.



The simulator also sends where the robot is and the four wheel powers, as above. They stay still in L020, since nothing drives the wheels.

7.1.5 L020S050: the right stick too

Change. Click in the Terminal and press Ctrl+C to stop the simulator. That is Ctrl+C on a Mac too, not ⌘C. Under the comment that names this step, read and write down the right stick the same way:

```
        // When on L020S050, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l020.html#l020s050
        // for what to do here.
        double rightStickY = gamepad1.right_stick_y;
        Tracker.publish("stick/rightY", rightStickY);
```

Simulator. Run the same command again.

AdvantageScope. Drag `rightY` from the `stick` folder into **Left Axis** beside `leftY`.

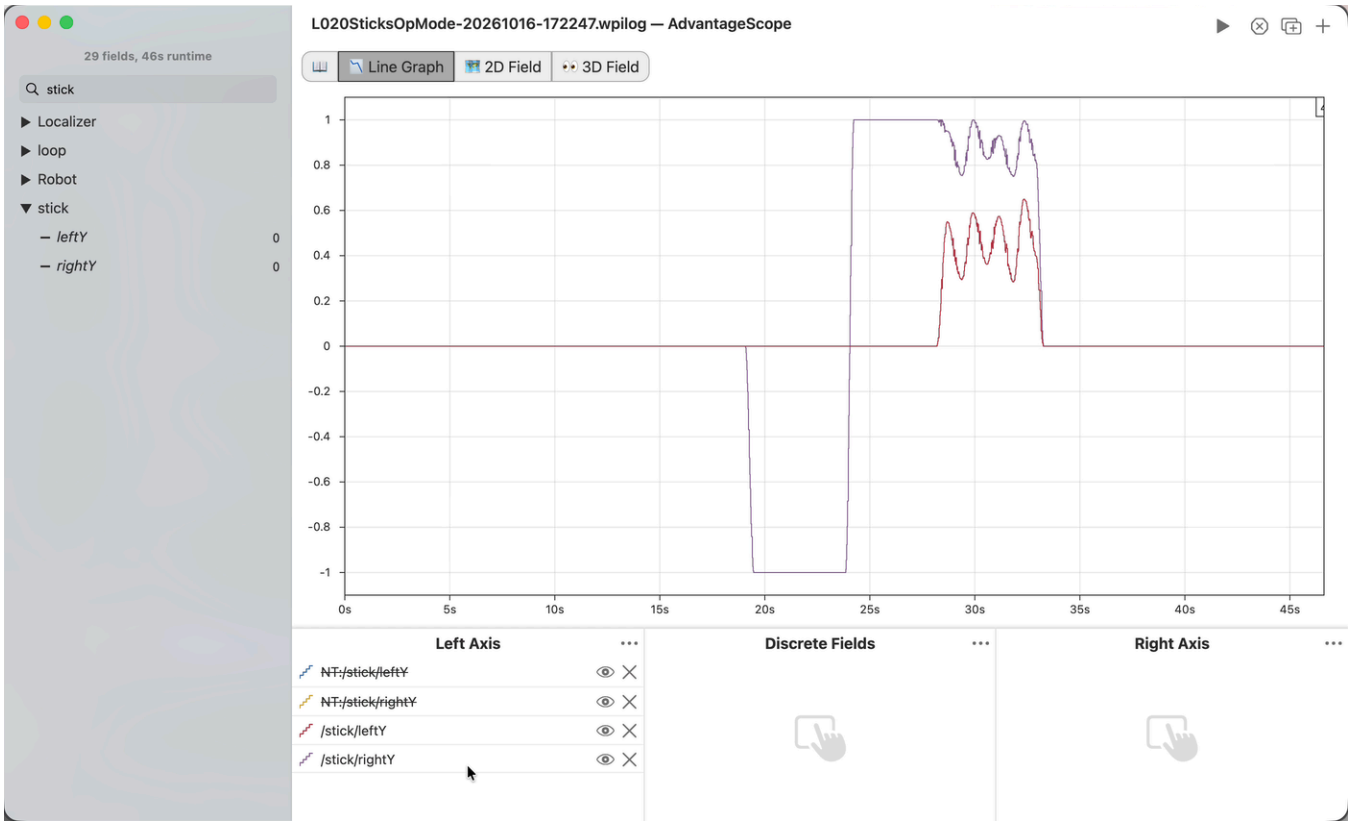
Check.

- Each line follows its own stick.
- Push each stick all the way both ways. The numbers never go past 1.0 or below -1.0, and forward is -1.0 on both.

Log.

- Press Ctrl+C in the Terminal to stop the simulator.

- In AdvantageScope, open **File > Open Log(s)…**. Go to the project's top folder, and click **Date Modified** so the newest file is on top. Open the top file whose name starts `L020SticksOpMode`.
- Drag `leftY` and `rightY` from `stick` into **Left Axis**.



Check.

- The lines show what the sticks did during the run.
- It is the same `stick/leftY` and `stick/rightY` you watched live, kept in a file you can open again later.

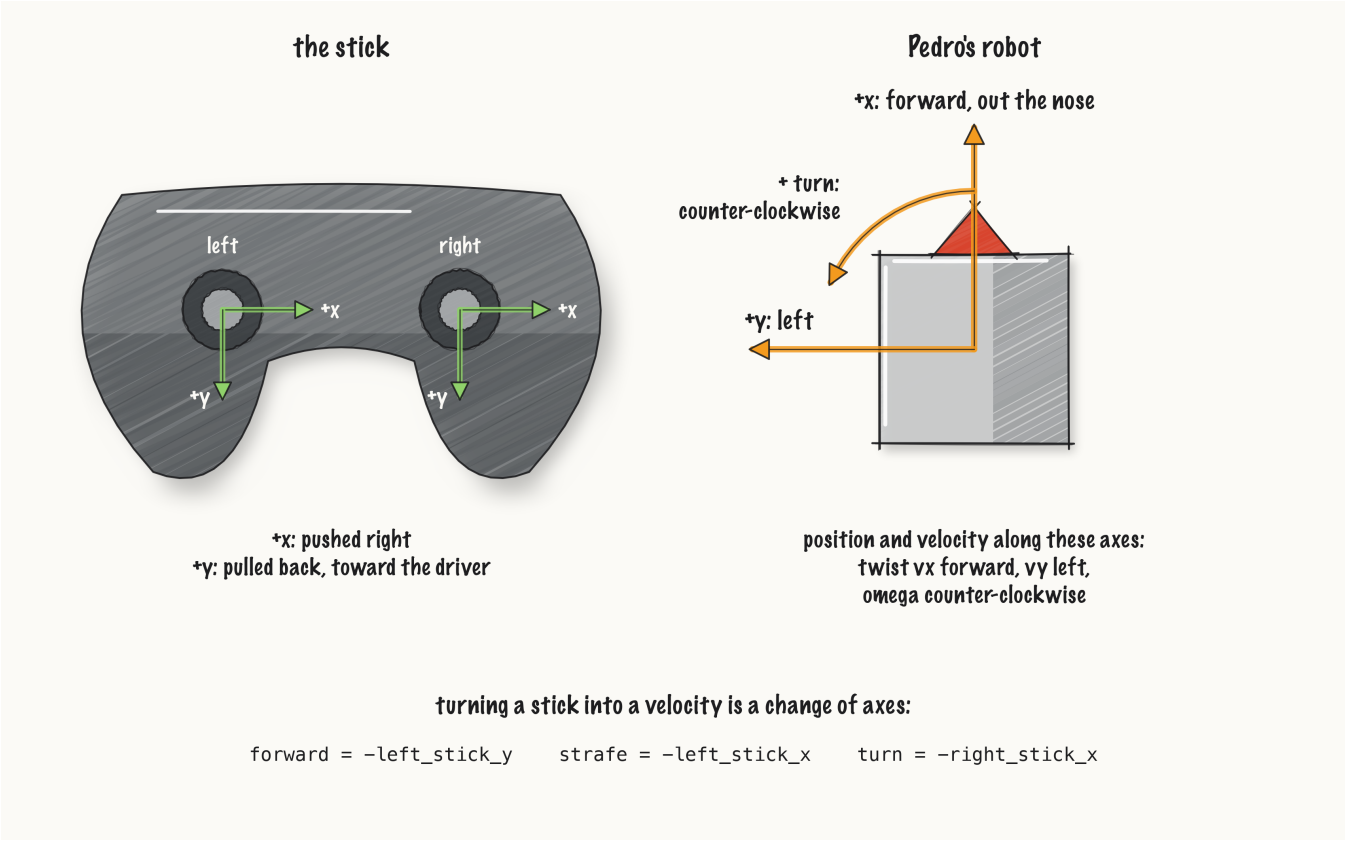
What's happening. E021 shows what your two lines do inside the computer.

What you just did. `readTheSticks` runs over and over, from PLAY until STOP, and each pass is one look at the gamepad. The code read both sticks into `doubles` and wrote each one down under a name. AdvantageScope showed the values live, and the flight log kept them to look at after the run.

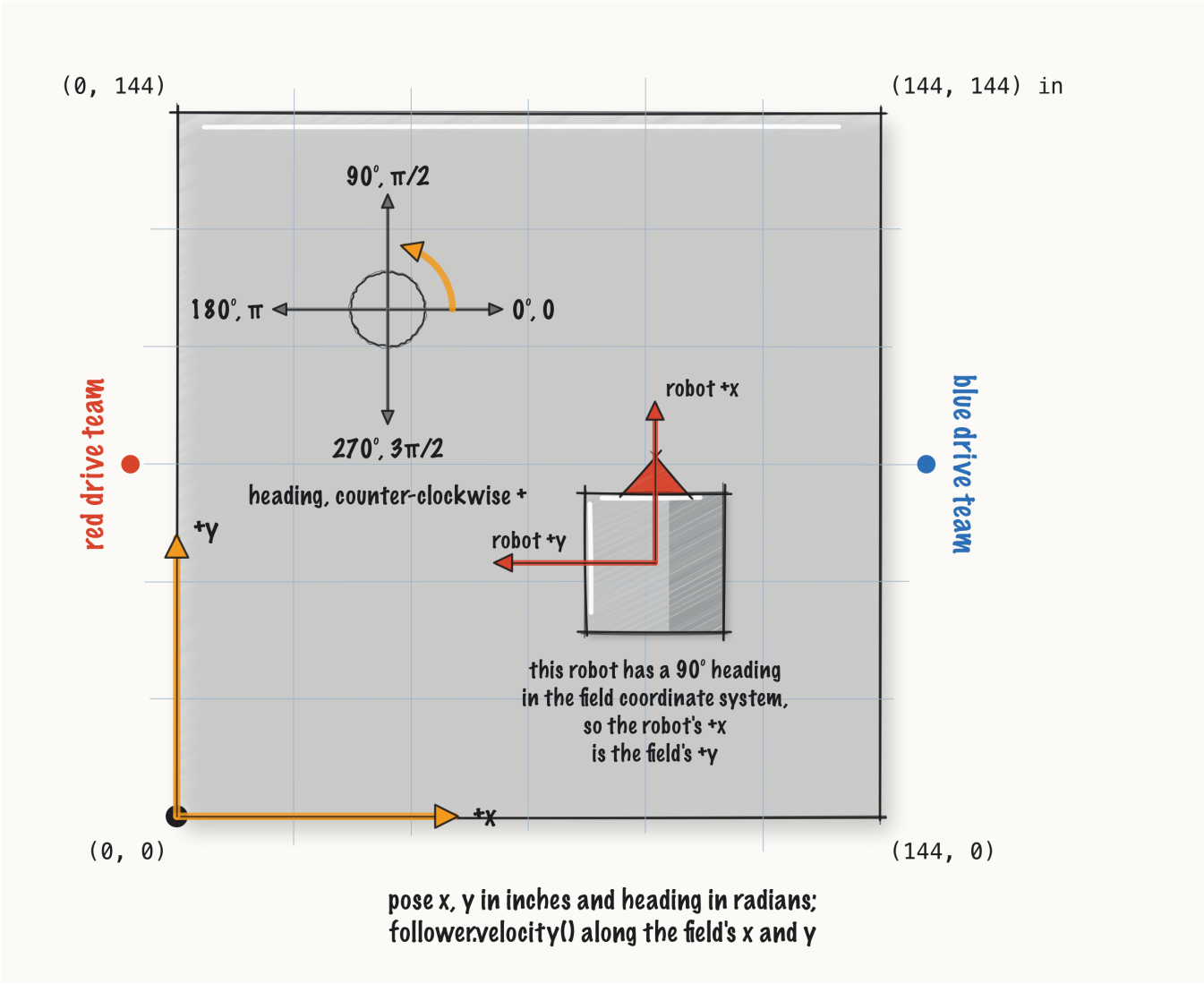
The scheduler. `start()` makes `readTheSticks` into a command that never finishes, with `Commands.infinite`, and schedules it. The scheduler runs every scheduled command once a loop, inside `loopAfter()`. Every lesson's code runs this way; L170 is the first to schedule a second command, on a button.

The stick's axes. A stick reads from -1.0 to +1.0 on each axis. +x is pushed to the driver's right, and +y is pulled back toward the driver, so forward on the stick is -y. From L030 on, a stick asks the robot for a velocity: the farther the push, the faster.

Pedro's axes. Pedro Pathing measures the robot along its own axes: +x is forward, out the nose, +y is to the robot's left, and a counter-clockwise turn, seen from above, is positive. `follower.twist()` gives the velocity on these axes: `vx` forward, `vy` left, `omega` counter-clockwise.



On the field, Pedro puts (0, 0) in a corner, with x and y each running 0 in to 144 in. Heading 0 points along the field's +x and $\pi/2$, 90°, along its +y; the code keeps it in radians. `follower.velocity()` gives the velocity along the field's x and y. Pedro's Coordinates page draws its field.



Field-relative driving. The red drive team stands at the field's left edge, x = 0 in, and the blue drive team at its right edge, x = 144 in. In field-relative driving, a red driver wants a push away on the left stick, the stick's -y, to drive the robot along the field's +x. A blue driver wants the same push to drive it along the field's -x. L140's push away drives along +x, as the red driver wants.

Turning a stick into a velocity changes axes. Forward is `-left_stick_y`, a slide to the left is `-left_stick_x`, and a counter-clockwise turn is `-right_stick_x`. The stick's axes are a mirror image of the robot's, so no turn of the gamepad lines them up, and each needs its minus sign. L030 makes the first, `-leftStickY`; L080 adds the turn and L090 the slide. L140 adds a second change, from the field's axes to the robot's.

Where next. L030 turns each stick into a speed, where forward is positive.

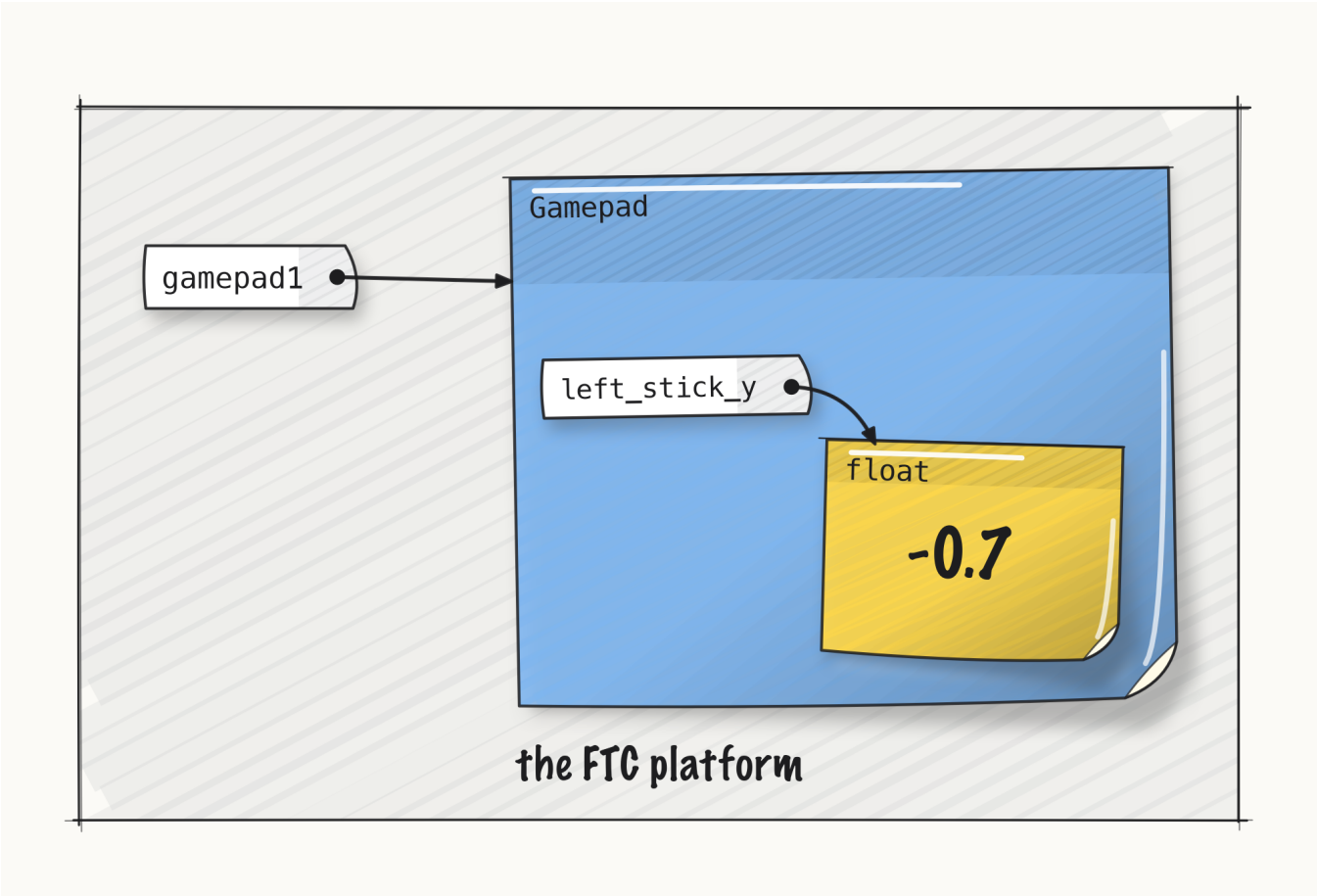
021 what's happening: assignments

When the computer runs your program, it reads one line of code at a time. It moves words and numbers around in its memory. It is like moving sticky notes around on your desk.

Here is what the two lines you wrote in L020 do. Each one is a **statement**: they tell the computer to do things.

```
double leftStickY = gamepad1.left_stick_y;
```

The FTC programming platform gives your program many **objects** to help program robots. `gamepad1` is one of them. So is `gamepad2`.



Think of `gamepad1` as a white label with an arrow to a sticky note. The note says what kind of thing it is: a `Gamepad`.

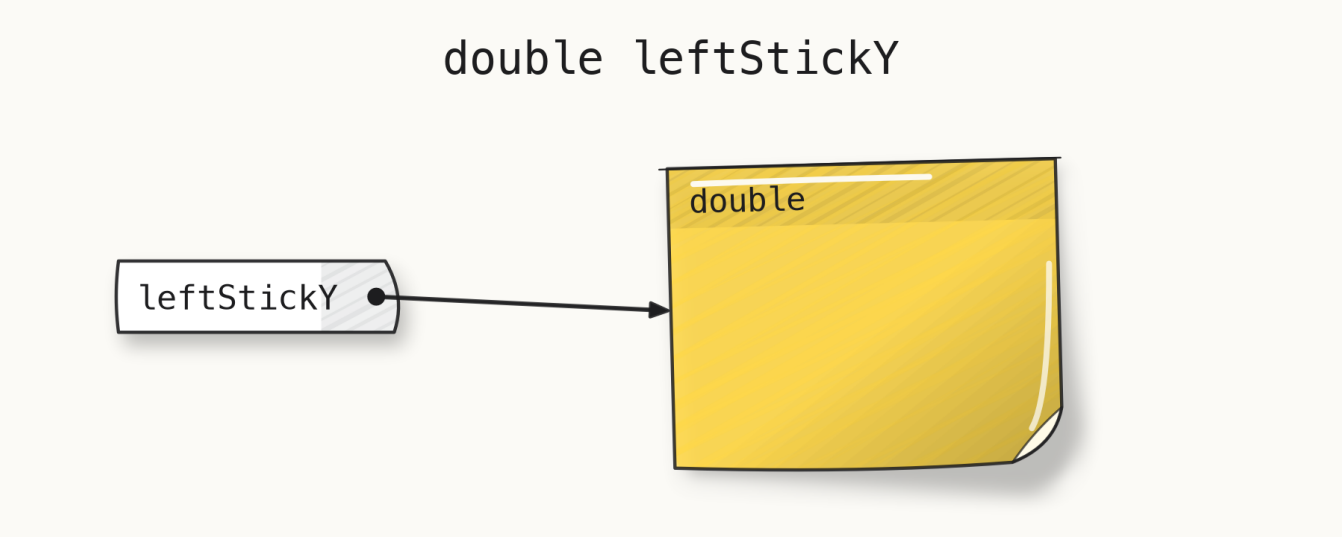
On the `Gamepad` note is the **field** `left_stick_y`, another label. Its arrow points to a small sticky note of type `float`.

The FTC platform does a lot of work for us. It runs `loop()` over and over, and in each pass the scheduler runs your `readTheSticks()`. Before each pass, it reads the left stick's y-axis on gamepad 1 and writes that number on the `float` note. The number sits there on the note, so your program can read it.

Our first statement started:

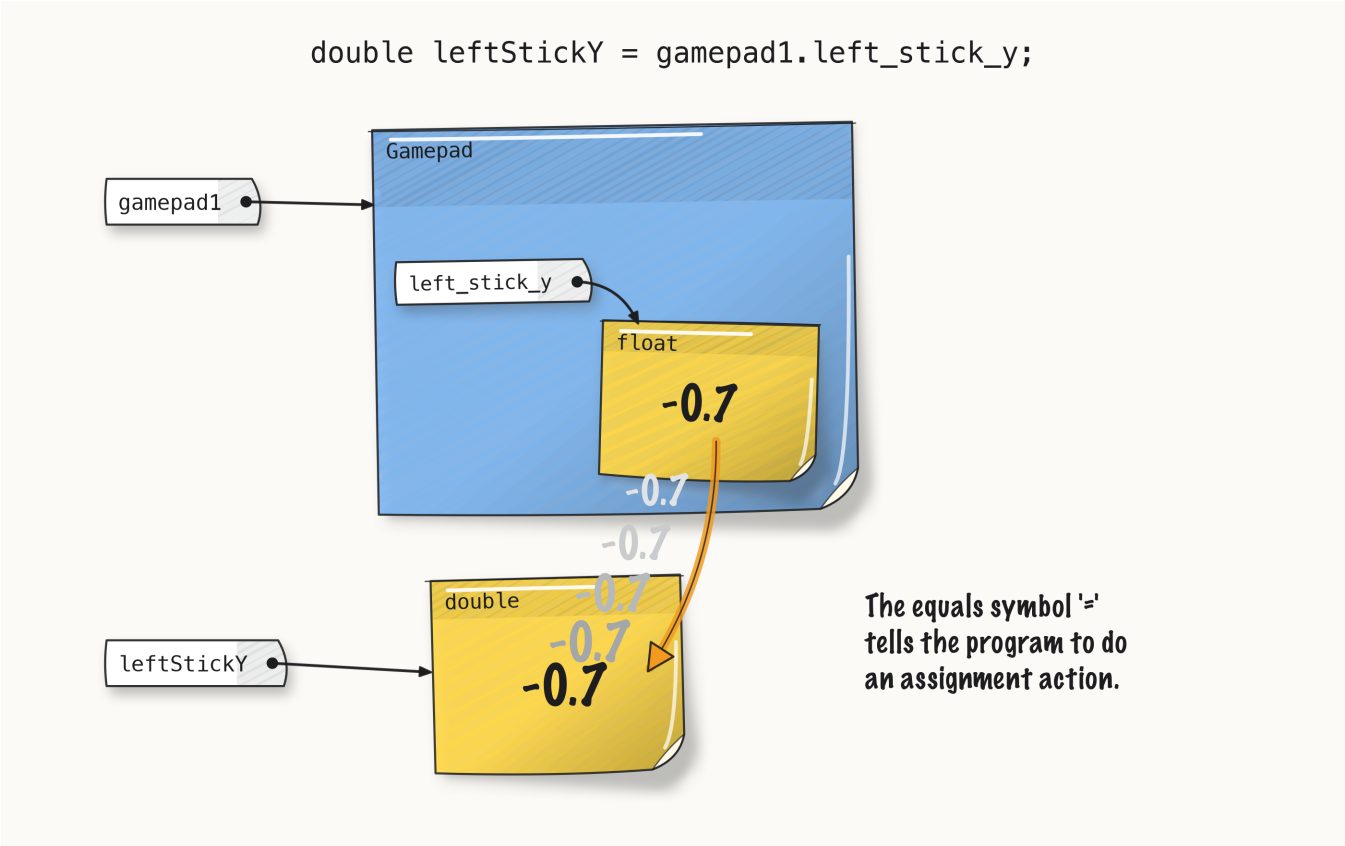
```
double leftStickY
```

That makes a **variable**. Also think of a variable as a name, drawn as a white label. Its arrow points to a sticky note of type `double`. But it starts out without a value.



In programming, the little equals sign, =, does a lot of work. It's a little different than the = in math. In math, = means two things are equal. In math we can say "2 = 2" is true, and "1 = 2" is false.

In programming, when we say something like `leftStickY = 1.0;`, we're not talking about something being true or false. We're telling the computer to take an action. The action is to forget about whatever was on the `leftStickY` sticky note, and write down a 1.0 on it. The = means "do the assignment action."

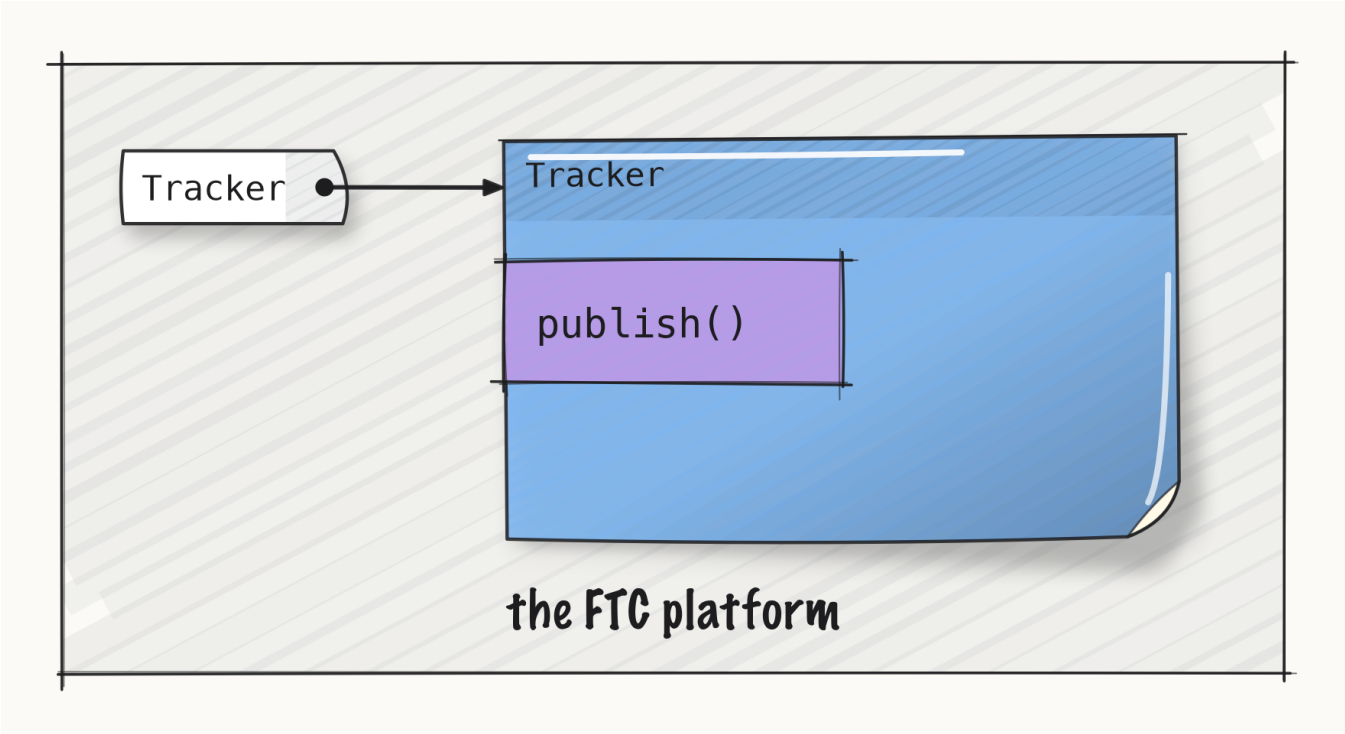


In your code, the `=` in `leftStickY = gamepad1.left_stick_y` copies the number on the `gamepad1` `left_stick_y` `float` note onto `leftStickY`'s note. `=` is an action: put this value here.

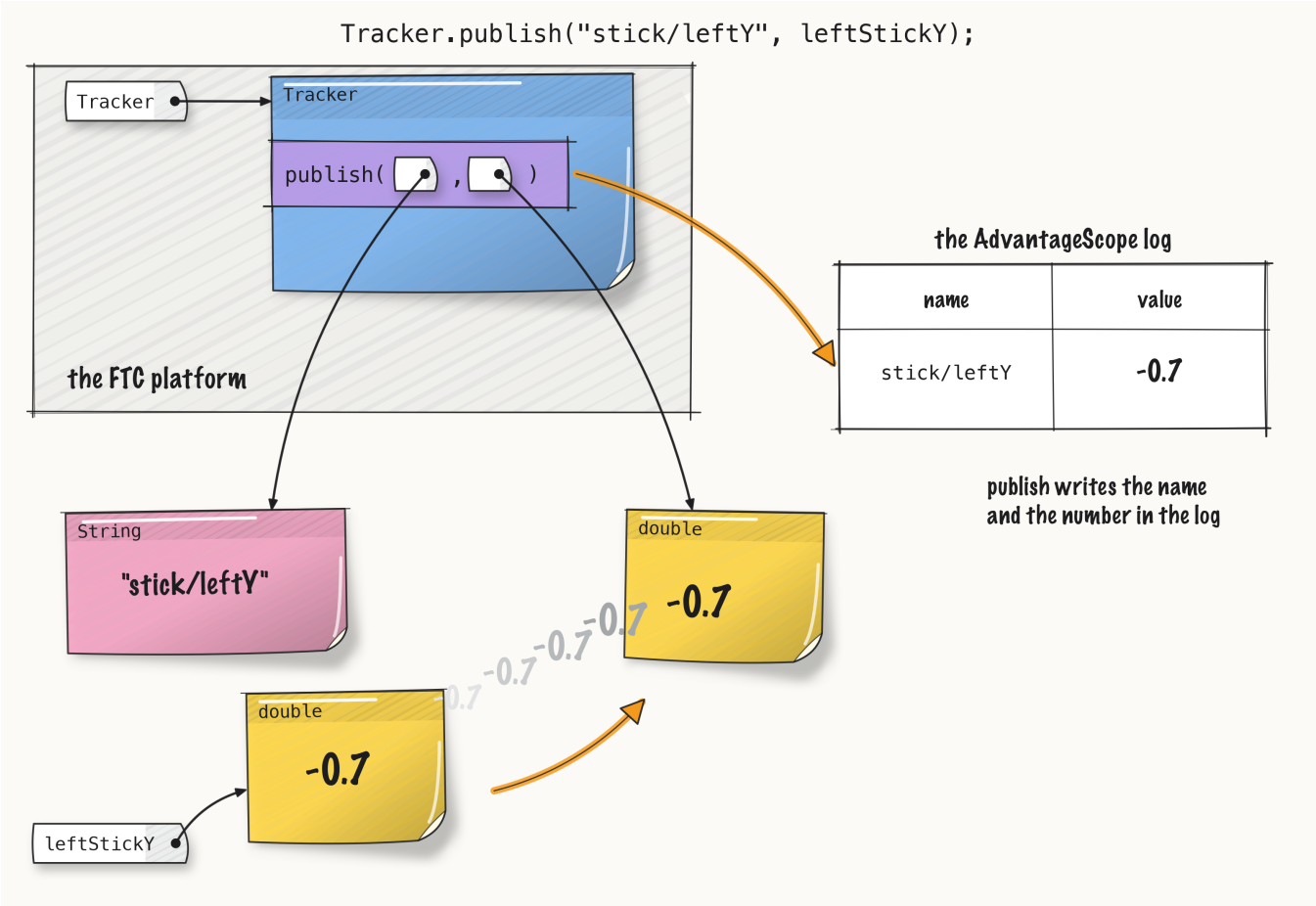
Your second line of code was:

```
Tracker.publish("stick/leftY", leftStickY);
```

`Tracker` is also provided by the platform. It lets you track things that you want to log to the **AdvantageScope** program.



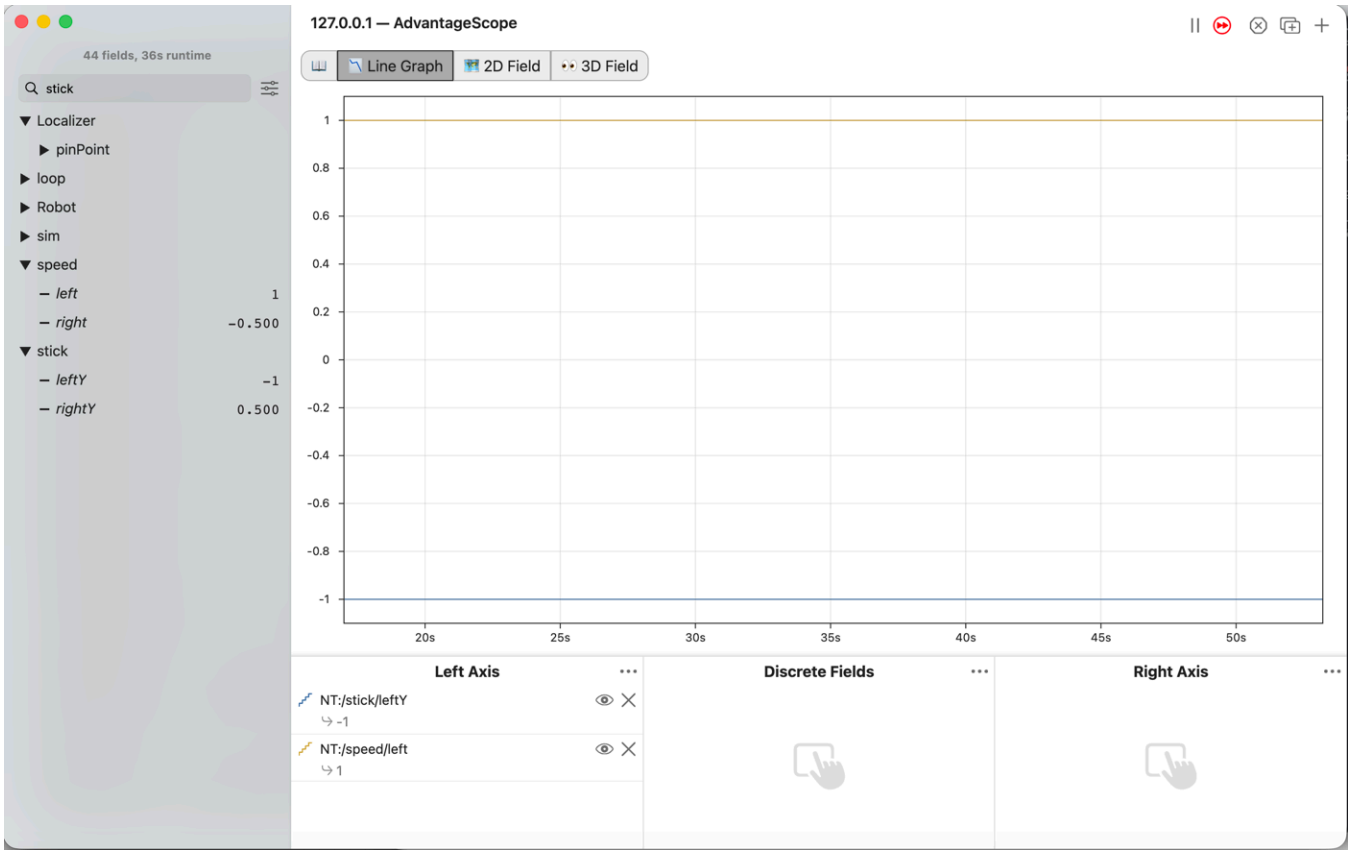
`Tracker` has a **method**, `publish`: a piece of code with a name. This line runs it, the way the scheduler runs your `readTheSticks()`.



The line `Tracker.publish("stick/leftY", leftStickY)` hands `publish` two things: a name for the value, in quotes, and a copy of `leftStickY`'s number.

`publish` sends the number on `leftStickY`'s note to AdvantageScope, and writes it in the flight log.

030 lesson: turn the sticks into speeds



Turn each stick into a wheel speed, where forward is positive. Nothing on the robot moves yet.

Before you start.

- L020 is done, so `mytry` holds `L020SticksOpMode`.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- No robot is needed.

9.1 the steps

9.1.1 L030S010: copy the starter into mytry

Copy.

- Open `TeamCode/java/org.firstinspires.ftc.teamcode/lessons`.
- Click `L030SpeedsOpMode` once, and copy it: ⌘C on a Mac, Ctrl+C on Windows.
- Click the `mytry` folder once, and paste.
- The **Copy Class** box opens with **Destination package** already ending in `mytry`. Click **OK**.
- If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L020SticksOpMode` and `L030SpeedsOpMode`, and the new file is open.

9.1.2 L030S020: switch your copy on

Change. Find the line `@Disabled`, just under the comment that names this step, and delete the whole line, the way you did in L020.

Check.

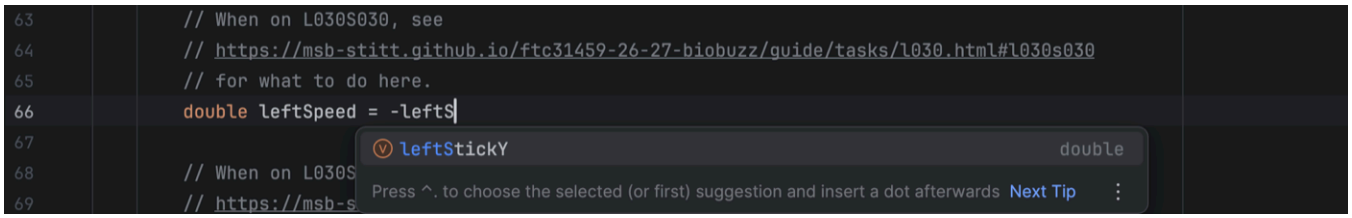
- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

9.1.3 L030S030: the left speed

We are working toward tank drive, where the left stick drives the left wheels and the right stick drives the right wheels. Push a stick away from you and its wheels should turn forward. We call forward speeds positive and backward speeds negative. But in L020 a stick pushed away read -1.0, so its number has to be turned around to make a speed.

The starter already has the four lines you wrote in L020. They read the sticks and write them down.

Change. Under the comment that names this step, type `double leftSpeed = -leftS`. The list offers `leftStickY`, the name from the lines above. Press Tab, then finish the two lines:




```
        // When on L030S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l030.html#l030s030
        // for what to do here.
        double leftSpeed = -leftStickY;
```

```
        Tracker.publish("speed/left", leftSpeed);
```

The minus sign turns the number around: -1.0 becomes 1.0, and 0.5 becomes -0.5. `leftStickY` is still what the stick says, and `leftSpeed` is how fast the wheels should go. Keeping two names for the two ideas means each name always says what it holds.

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L030SpeedsOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020. Type `stick` in the search box. Drag `leftY` from the `stick` folder, and `left` from the `speed` folder, into **Left Axis**.

Check.

- Push the left stick away from you. `stick/leftY` goes to -1.0, and `speed/left` goes to 1.0.
- The two lines are always mirror images of each other.

9.1.4 L030S040: the right speed

Change. Press Ctrl+C in the Terminal to stop the simulator. Under the comment that names this step, do the same for the right stick:

```
        // When on L030S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l030.html#l030s040
        // for what to do here.
        double rightSpeed = -rightStickY;
        Tracker.publish("speed/right", rightSpeed);
```

Simulator. Run the same command again.

AdvantageScope. Drag `rightY` from `stick` and `right` from `speed` into **Left Axis**.

Check.

- Push both sticks away from you. Both speeds read 1.0.
- Pull both back, and both read -1.0.

What's happening. E035 shows why each speed has a minus sign.

What you just did. Each speed is its stick with a minus sign in front, so forward is positive from here on. The stick and the speed have separate names because they are separate things: one is what the driver's hand does, the other is what the wheels should do. A name like `leftStick = -gamepad1.left_stick_y` would hide that.

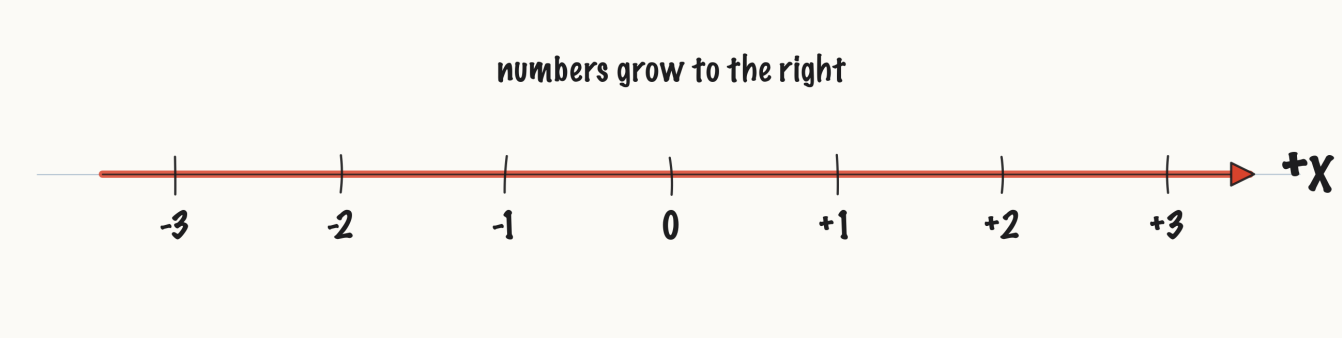
Where next. L040 sends `leftSpeed` and `rightSpeed` to the wheels and drives the robot in the simulator.

035 what's happening: sticks into speeds

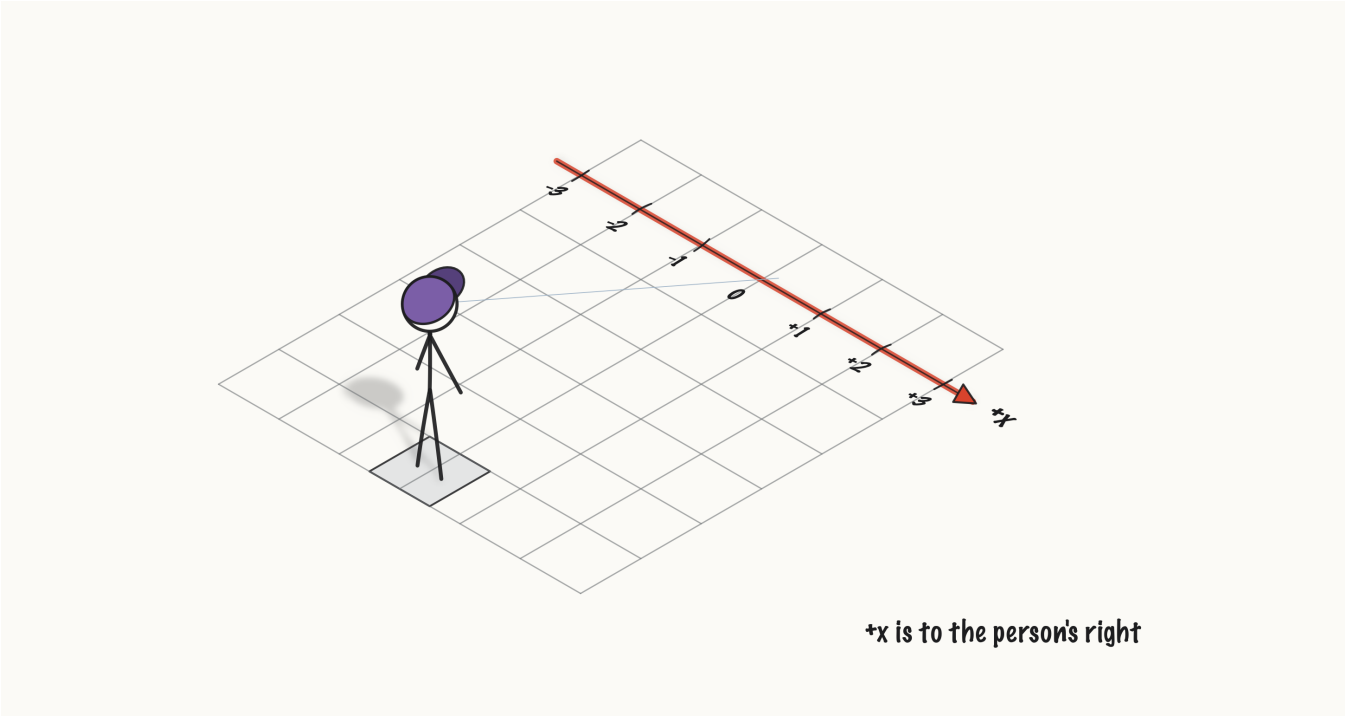
This page is about one line you wrote in L030, and why it has a minus sign.

```
double leftSpeed = -leftStickY;
```

We draw number lines like this. Zero is in the middle and numbers grow to the right. Numbers get smaller and more negative on the left.

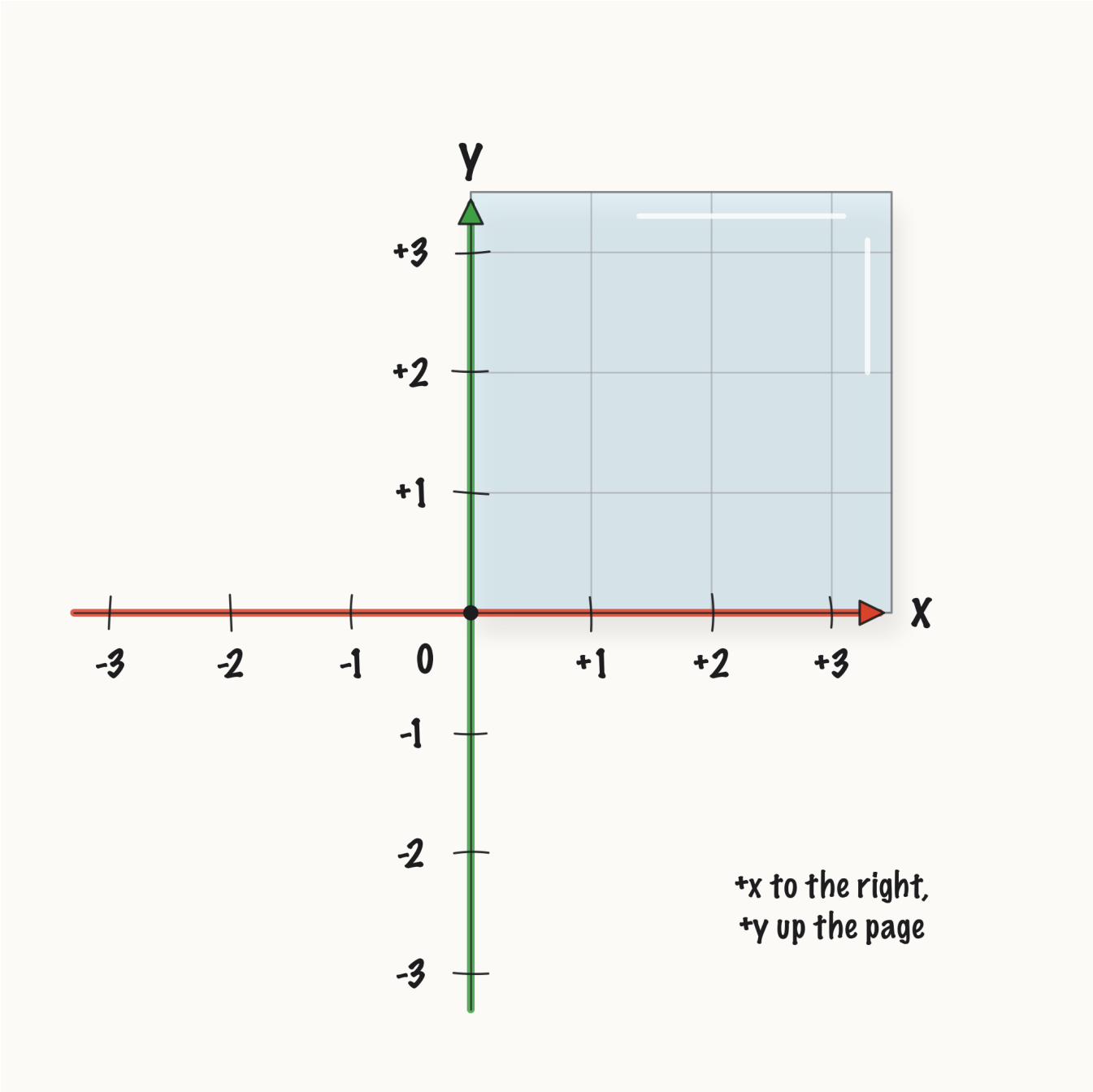


We look at number lines from our point of view.



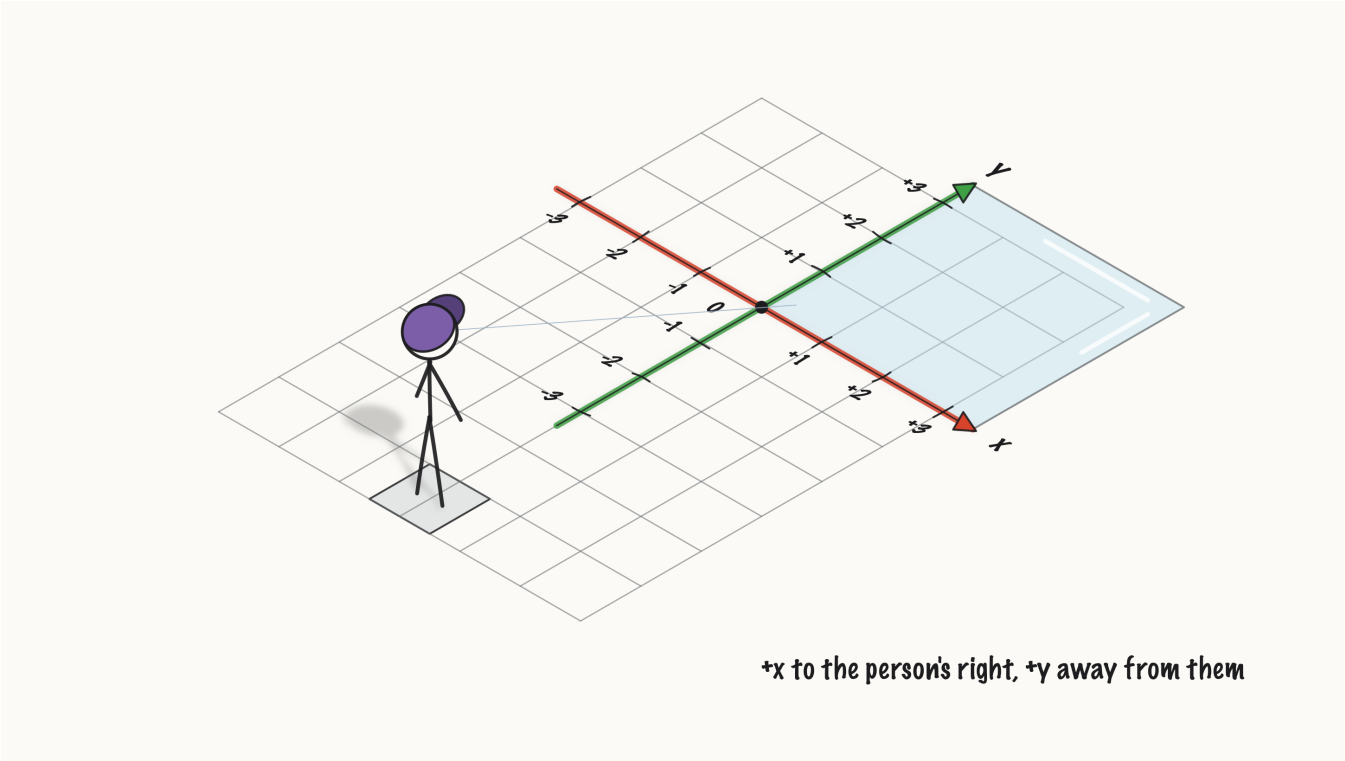
We use a number line to plot one quantity. It could be how many apples we have. On a robot, it could be how far to our right the robot is.

When we plot two quantities at the same time, we need a number grid.

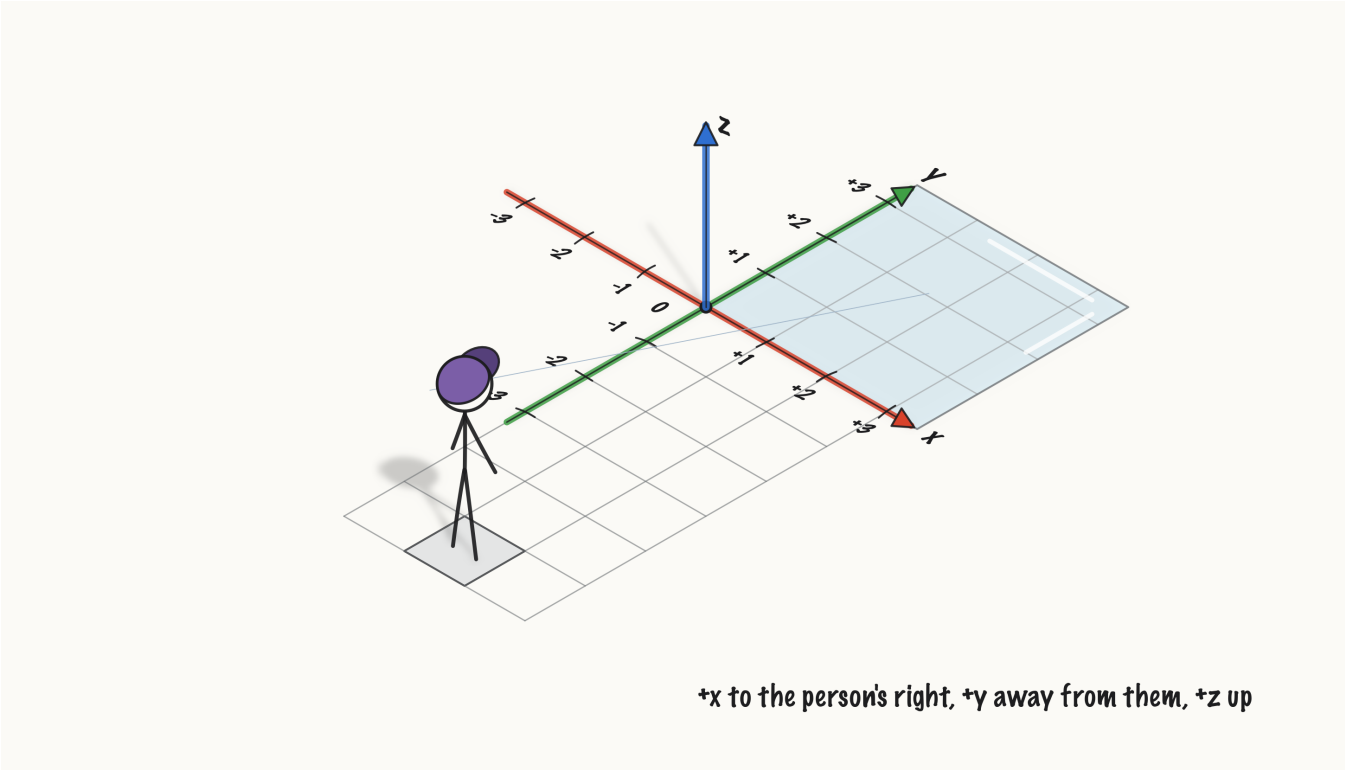


We call the axis that increases to the right the **x** axis. We call the axis that increases up the page the **y** axis. We're used to larger **+y** values going farther up the page.

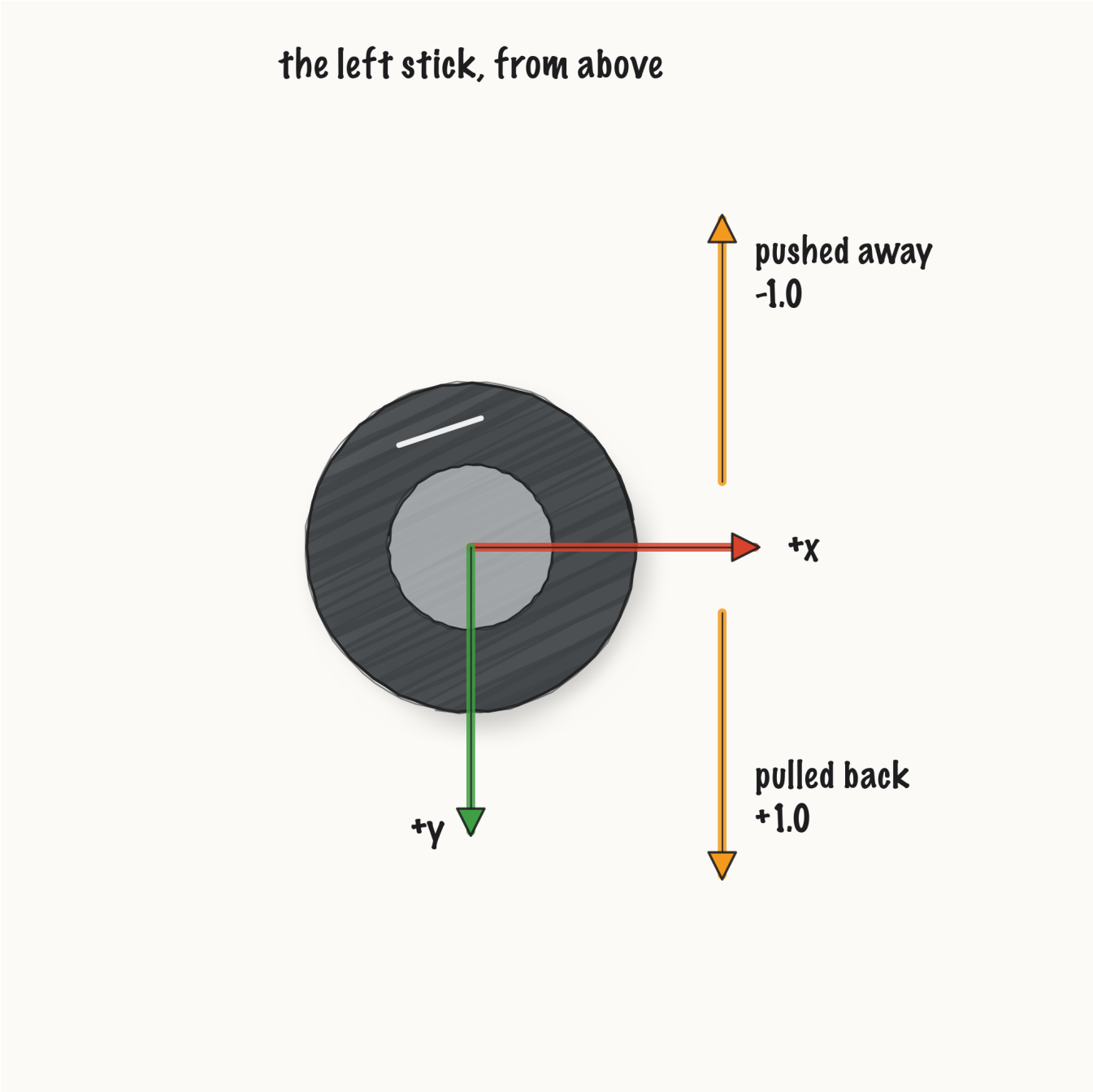
We call the **xy** grid a coordinate grid. We expect larger **+y** values to be farther away from us from our point of view.



When we plot three values at the same time, like how far to the right, how far away, and how far up something is, we add a third axis. A **z** axis. We expect **+z** to be up.



The gamepad joysticks surprise us because they don't follow our normal convention.



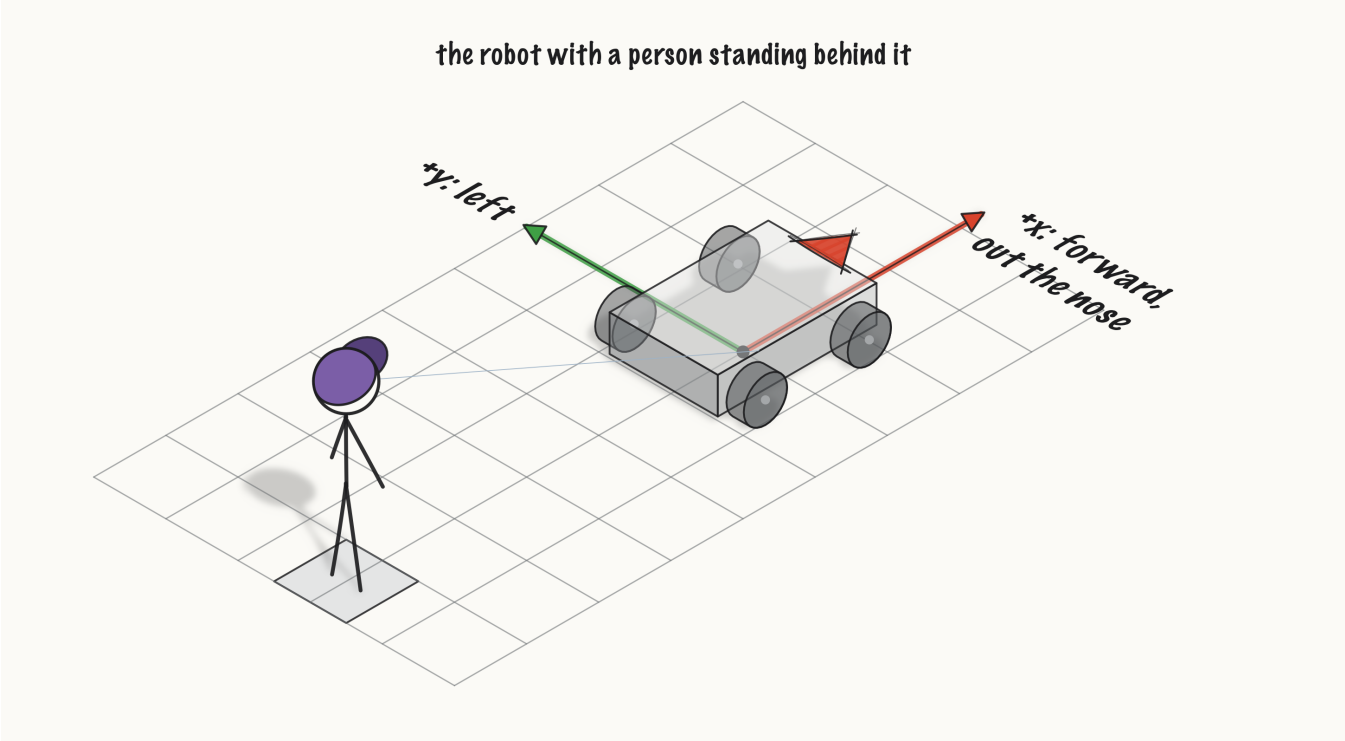
The **y** axis measures how far the stick is towards us or away from us. That's good. But **+1.0** is all the way towards us, and **-1.0** is all the way away from us. It's backwards!

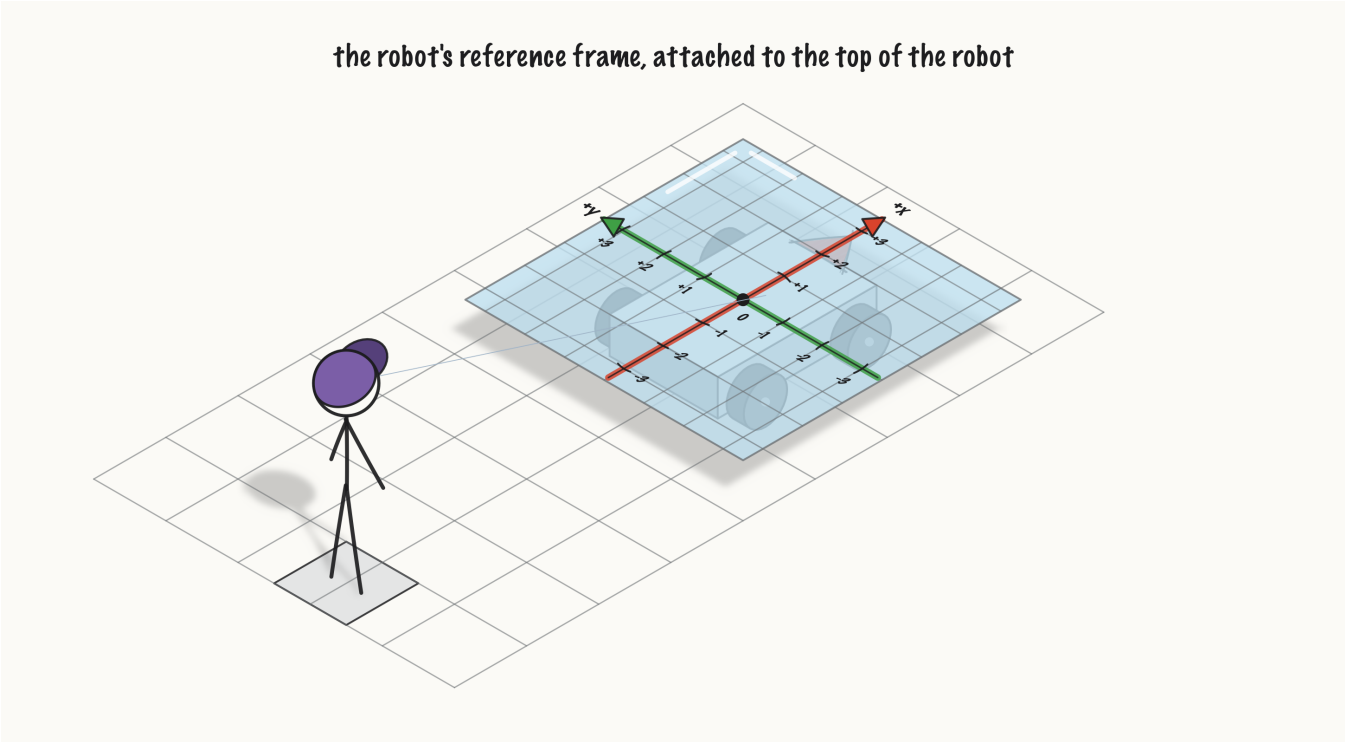
Coordinate grids can use an unexpected point of view or orientation. When we write robot programs, we need to know what each number means. Is **+** towards us or away from us? Is the **y** axis towards and away, or is it left and right?

We use a software library called Pedro Pathing as part of our software platform to steer our robots.

It uses a different coordinate grid orientation for steering our robot. When we program robots, we often attach

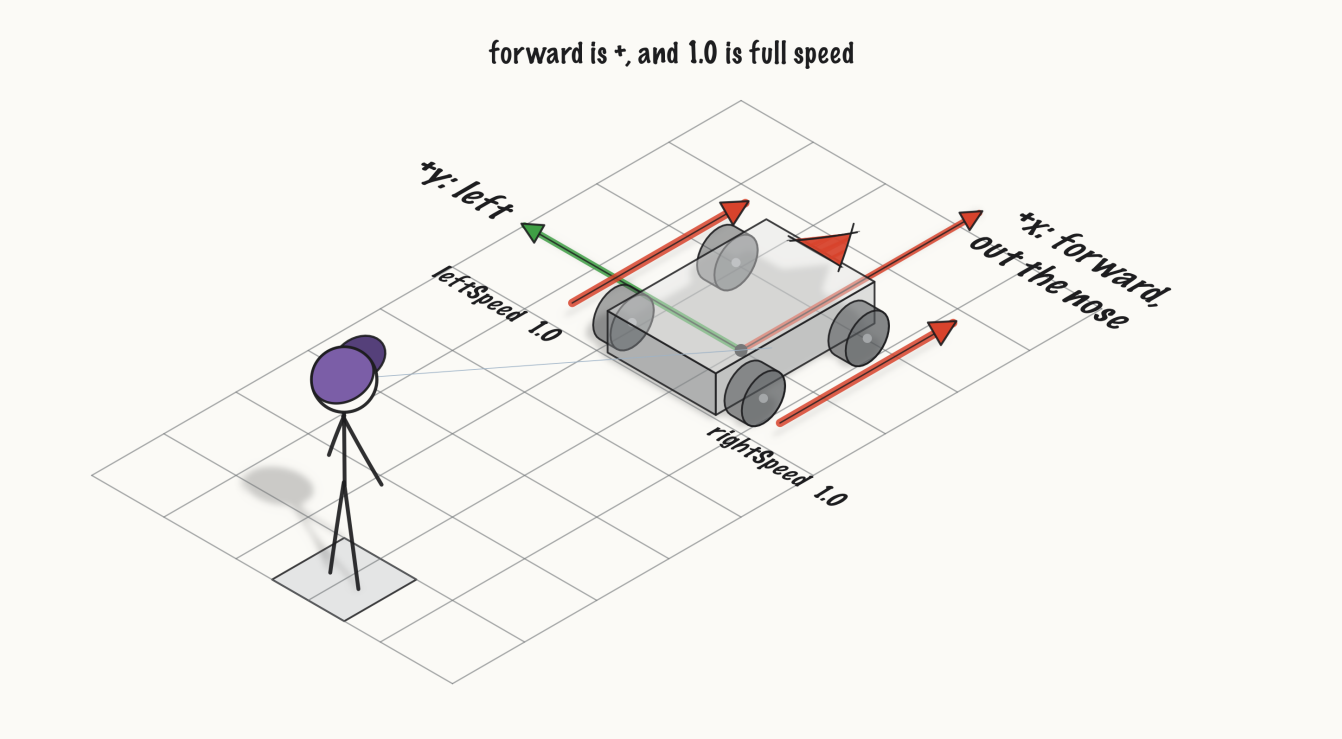
a coordinate grid to an object, such as the robot. A coordinate grid attached to an object is called a reference frame. When the robot moves or turns, its reference frame moves and turns with it.



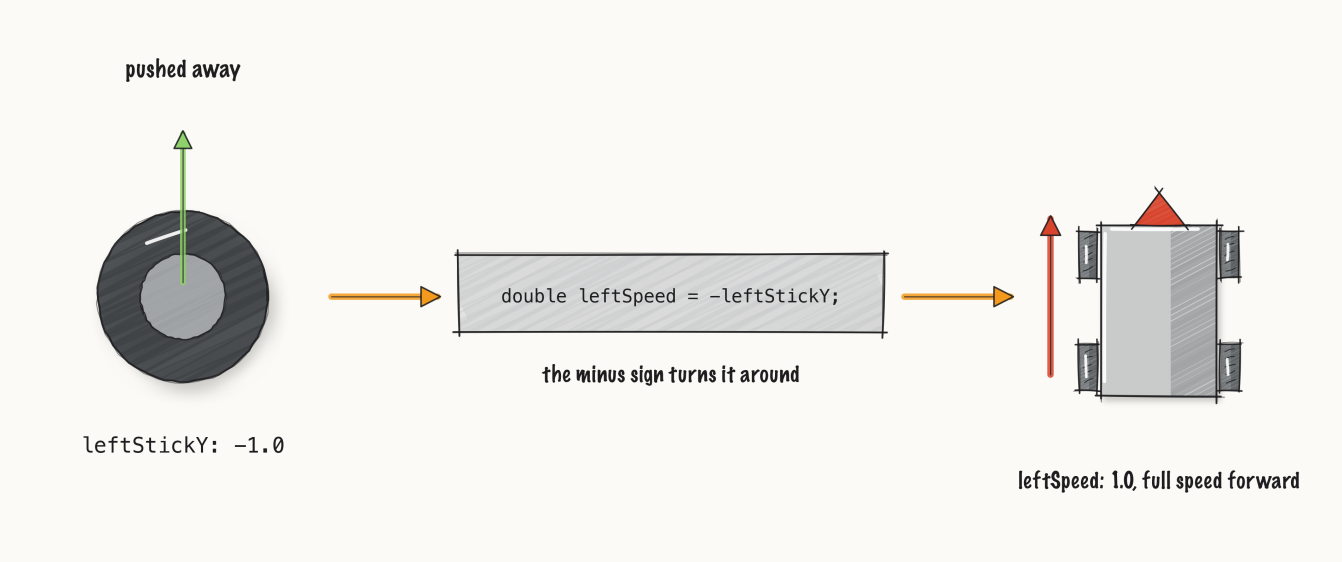


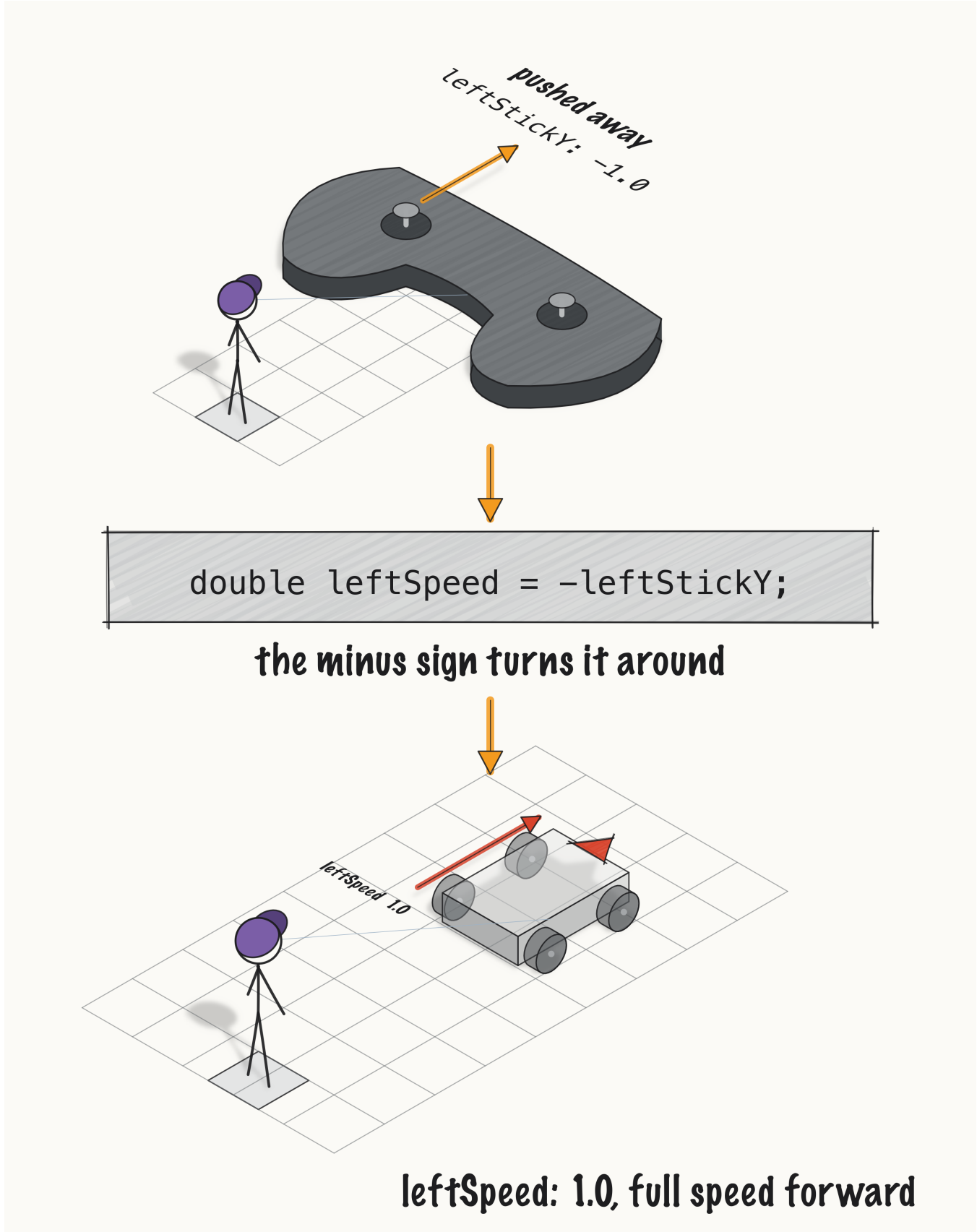
In the robot's reference frame in Pedro Pathing, **+x** is forward, out the nose of the robot. **+y** is to the left.

A speed tells the wheels how fast to go, and which way. We like forward to be positive. A speed of 1.0 is full speed forward, 0.0 is stopped, and -1.0 is full speed backward. In tank drive, `leftSpeed` is for the left wheels and `rightSpeed` is for the right wheels.



We expect a joystick to control the speed of the robot. Moving at a speed for some time changes the robot's position. When we let go of the sticks we get **zeros** from the sticks and we expect the robot to stop, to have a speed of **0**. The further we push the sticks, the faster we expect the robot to move.




pushed away
leftStickY: -1.0
double leftSpeed = -leftStickY;
the minus sign turns it around
leftSpeed 1.0
leftSpeed: 1.0, full speed forward

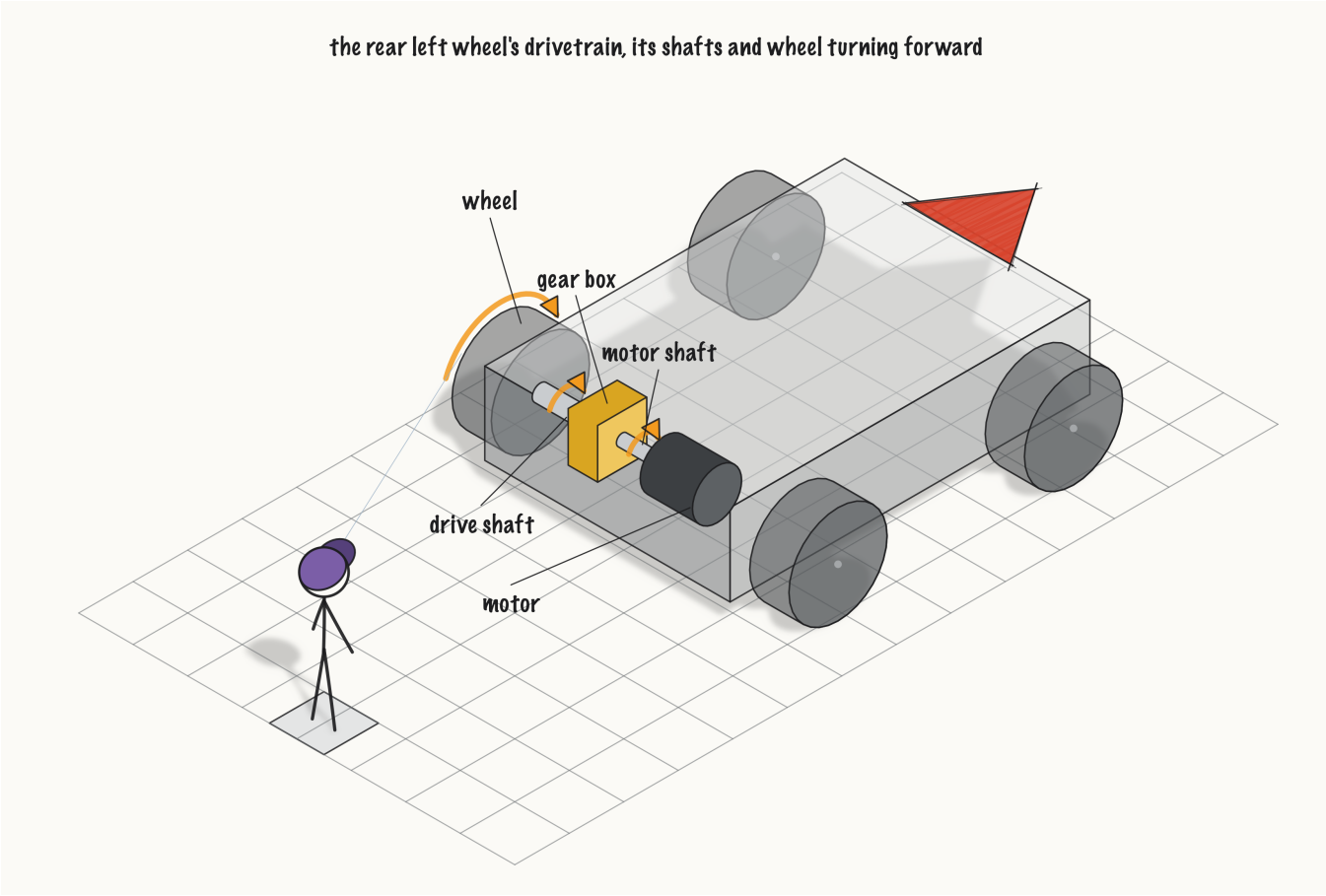
When you drive, pushing a stick away from you should drive the robot forward. Pushed away, the stick reads -1.0. Its wheels should go full speed forward, 1.0. The stick's +y points back at you, and the robot's forward points away from you, so the two numbers have opposite signs.

The minus sign switches from the stick's reference frame to the robot's reference frame. It turns -1.0 into 1.0, and 1.0 into -1.0. Forward is **+x** in the robot's reference frame:

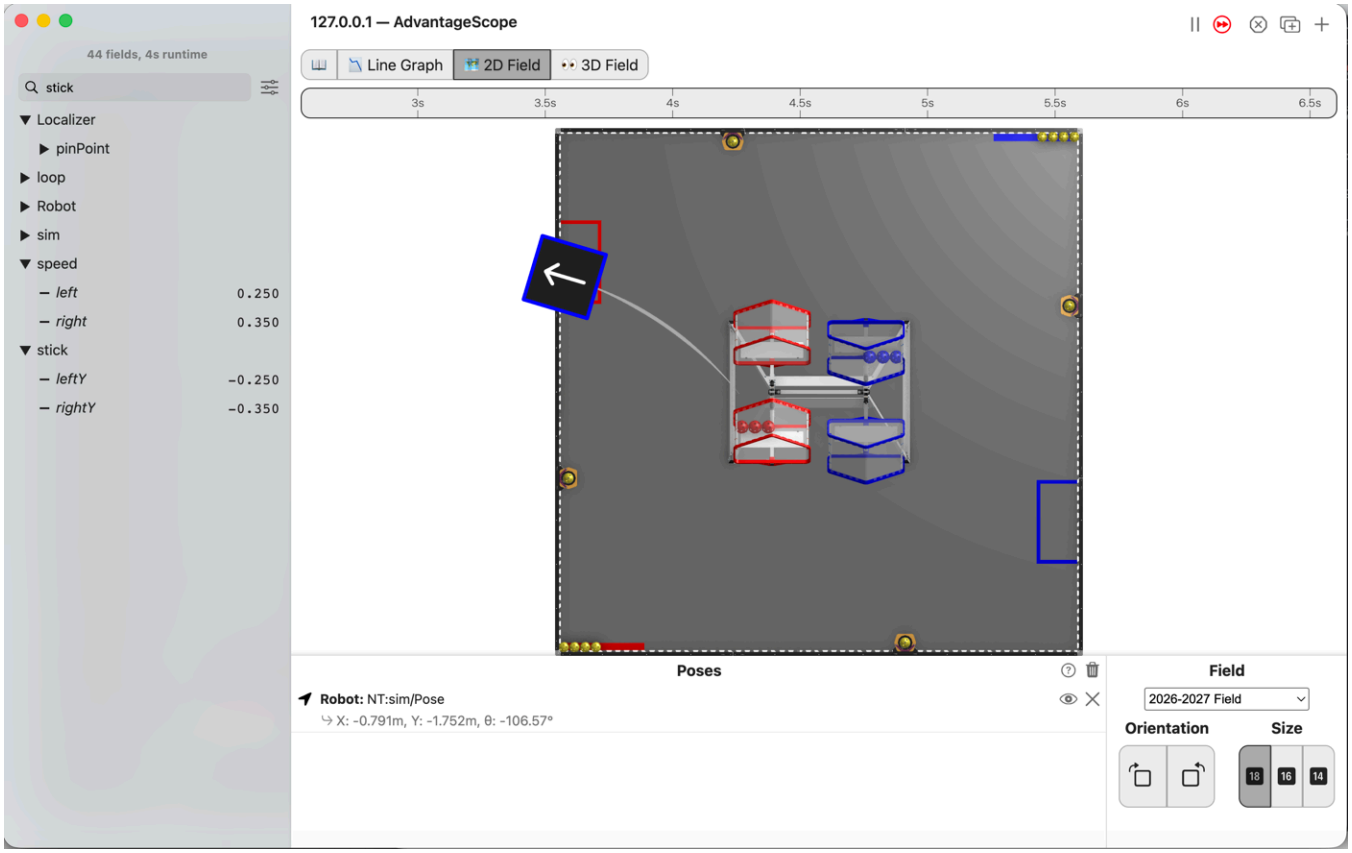
```
double leftSpeed = -leftStickY;
double rightSpeed = -rightStickY;
```

`leftStickY` is what the stick says. `leftSpeed` is how fast the wheels should go. They are numbers in two different reference frames, so they get two different names.

Motors turn gear boxes which turn wheels. When each part turns, it turns at a speed. Either turns per second, or revolutions per minute. Or for the wheel, inches per second at the outside of the wheel.



040 lesson: drive like a tank



The two speeds drive the wheels, and the robot drives in the simulator: tank drive!

Before you start.

- L030 is done, so `mytry` holds `L030SpeedsOpMode`.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- No robot is needed. The robot drives in the simulator only; L050 puts it on blocks.

11.1 the steps

11.1.1 L040S010: copy the starter into mytry

Copy. In `lessons`, click `L040TankOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L040TankOpMode`, and the new file is open.

11.1.2 L040S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

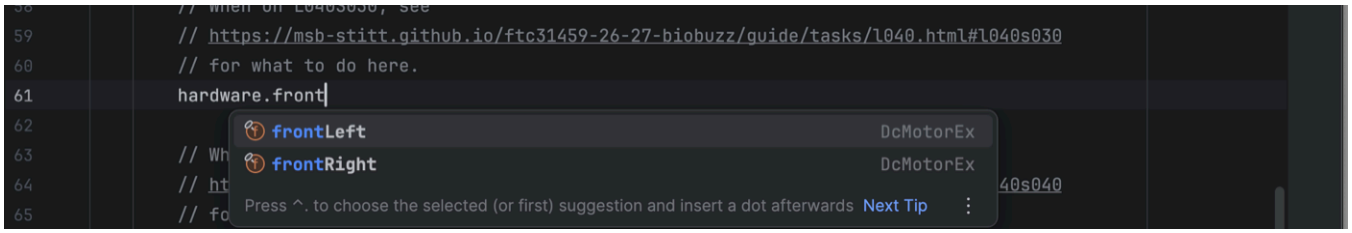
Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

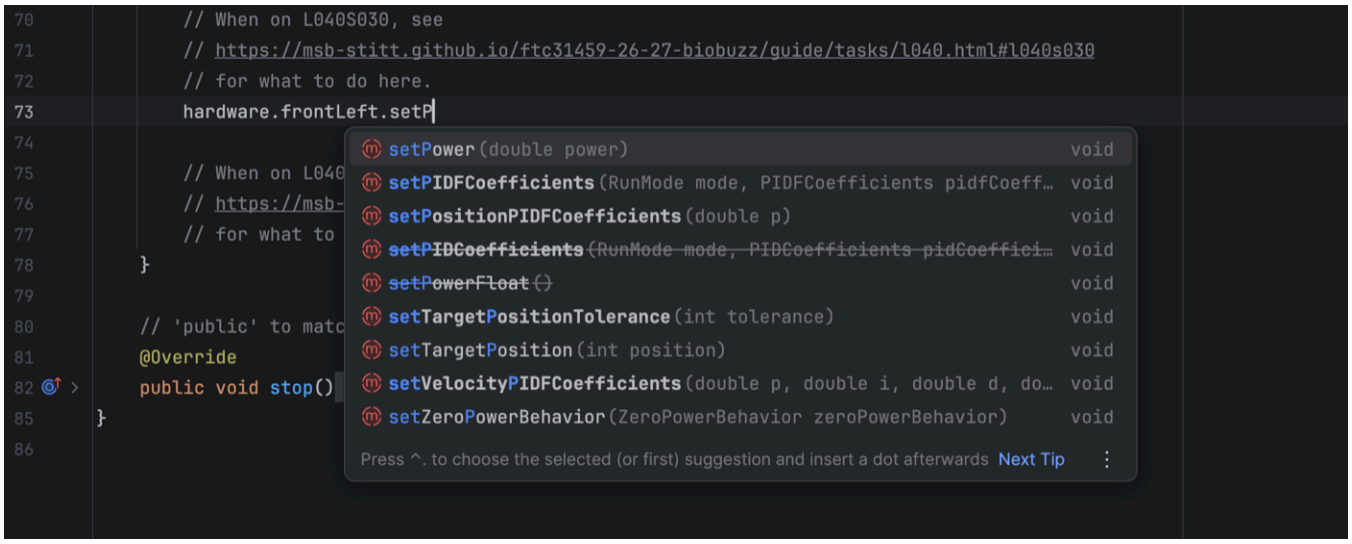
11.1.3 L040S030: the left wheels

The starter already reads the sticks and turns them into `leftSpeed` and `rightSpeed`, as you did in L030.

Change. The robot has four motors, one on each wheel. `hardware` holds all four. Under the comment that names this step, type `hardware.front`. The list offers `frontLeft` and `frontRight`. Pick `frontLeft` and press Tab.



Type `.setP`. The list shows what a motor can be told to do. The one we want is `setPower(double power)`.



Finish the line, and add the same for the back left wheel:

```
        // When on L040S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l040.html#l040s030
        // for what to do here.
        hardware.frontLeft.setPower(leftSpeed);
        hardware.backLeft.setPower(leftSpeed);
```

A power is a number from -1.0 to +1.0. It is how hard the motor pushes: 1.0 is full forward, -1.0 is full backward, and 0.0 is stop.

Check.

- Neither line has a red underline.

11.1.4 L040S040: the right wheels

Change. Under the comment that names this step, do the same for the right side:

```
        // When on L040S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l040.html#l040s040
        // for what to do here.
        hardware.frontRight.setPower(rightSpeed);
        hardware.backRight.setPower(rightSpeed);
```

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L040TankOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020, and click **2D Field**. Open the `sim` folder and drag `Pose` into **Poses**. The robot appears near the bottom edge of the field. A thin line runs to it from the middle of the field. That is where the simulator put the robot before the OpMode moved it to its start, so you can ignore it.

Check.

- Push both sticks gently forward, and the robot drives forward.
- Pull both back, and it comes back.

- Push one forward and pull the other back, and it spins in place.
- Push the right stick further than the left, and it curves to the left.

What's happening. E041 shows what a wheel line does inside the computer.

No gamepad. `--args="mytry.L040TankOpMode --left_stick_y=-0.25 --right_stick_y=-0.35"` holds both sticks gently forward, the right a little further. The picture at the top was taken that way, 4 s after connecting. The simulator has no walls, so a robot left running drives off the field.

If it didn't, and the robot creeps with both sticks let go, the sticks do not read exactly 0.0. Nothing in L040 tidies that up; L060 does.

What you just did. Each pass through `driveFromTheSticks` reads the sticks, turns them into speeds, and hands each speed to the two motors on its side. That is tank drive. In the simulator every motor turns its wheel forward for a positive power. On the robot, a motor mounted the other way around turns its wheel backward, and L050 is where that is found and fixed.

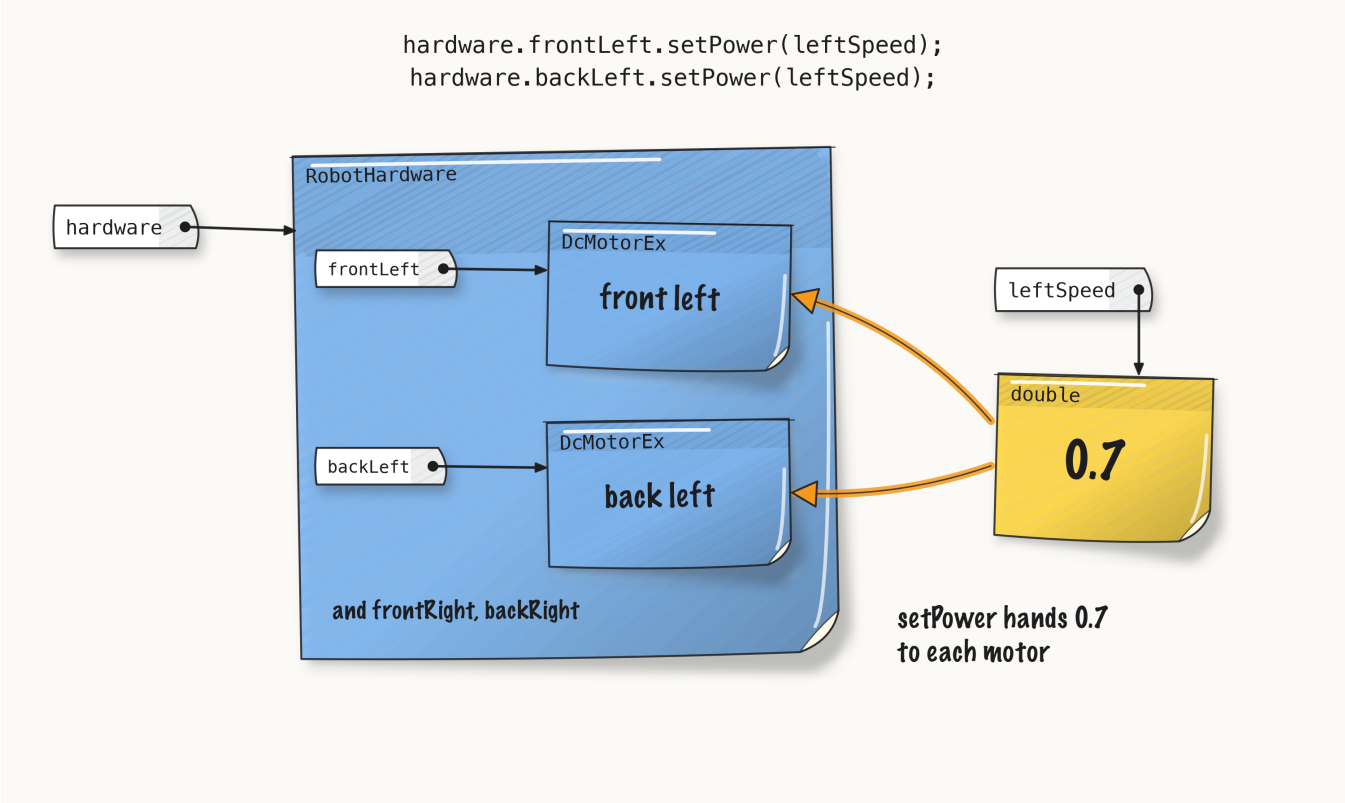
Where next. L050 runs this on the robot, on blocks.

041 what's happening: calling a method

When the computer runs your program, it reads one line of code at a time. It moves words and numbers around in its memory. It is like moving sticky notes around on your desk.

Here is what one of the wheel lines you wrote in L040 does.

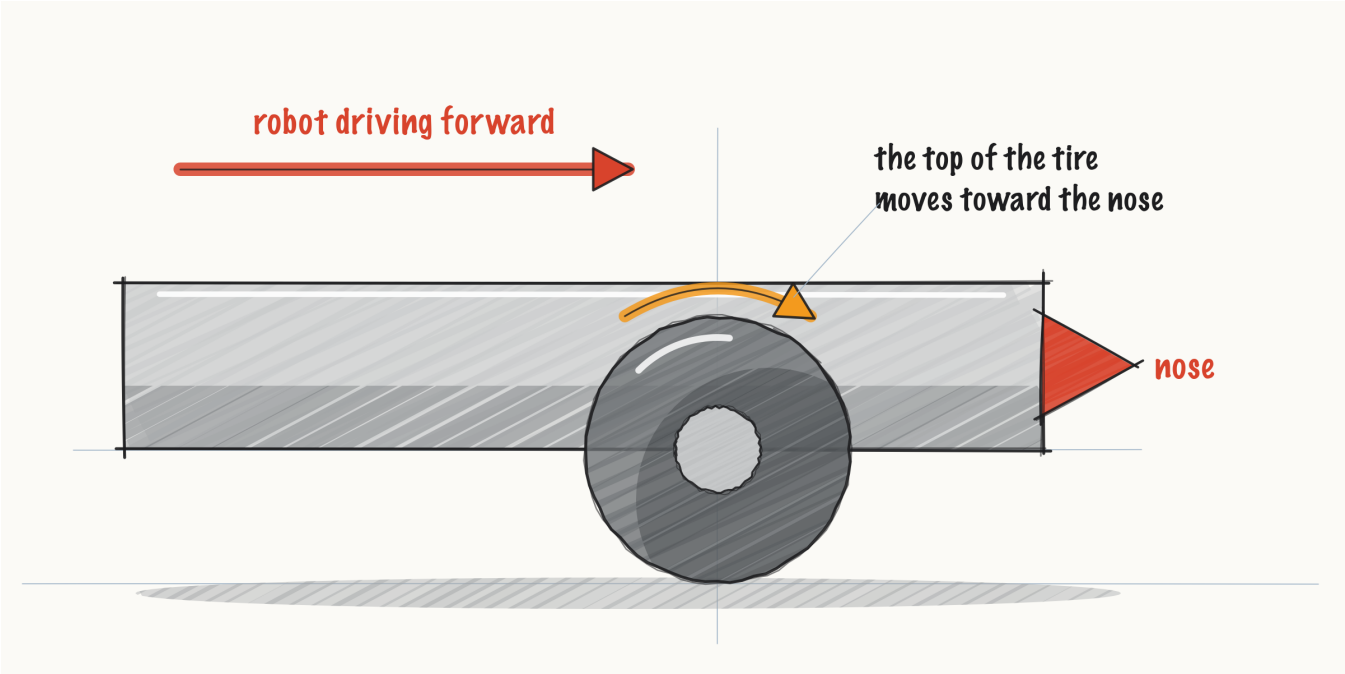
```
hardware.frontLeft.setPower(leftSpeed);
```



- The computer finds the variable `hardware`, and the big blue note its arrow points to.
- On that note is the field `frontLeft`. Its arrow points to a motor's note.
- A motor has a method, `setPower`, and the line hands it `leftSpeed`. That is the `double` on `leftSpeed`'s note, 0.7 in the picture.

- `setPower` is part of the FTC platform. It runs that motor full power forward at +1.0, not at all at 0.0, and full power backward at -1.0.
- The same happens for each of the four: `frontLeft`, `backLeft`, `frontRight` and `backRight`.

050 lesson: drive the robot on blocks



Tell each motor which way it is mounted. Then run tank drive on the robot, up on blocks, and check that every wheel turns the way its stick says.

Before you start.

- L040 is done, so `mytry` holds `L040TankOpMode`.
- A robot, a Driver Station with a gamepad, and a USB cable from the laptop to the Control Hub.
- Two blocks to rest the robot on, so no wheel touches the ground, and about 2 m of clear floor for the last step.

13.1 the steps

13.1.1 L050S010: copy the starter into mytry

Copy. In `lessons`, click `L050BlocksOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L050BlocksOpMode`, and the new file is open.

13.1.2 L050S020: switch your copy on

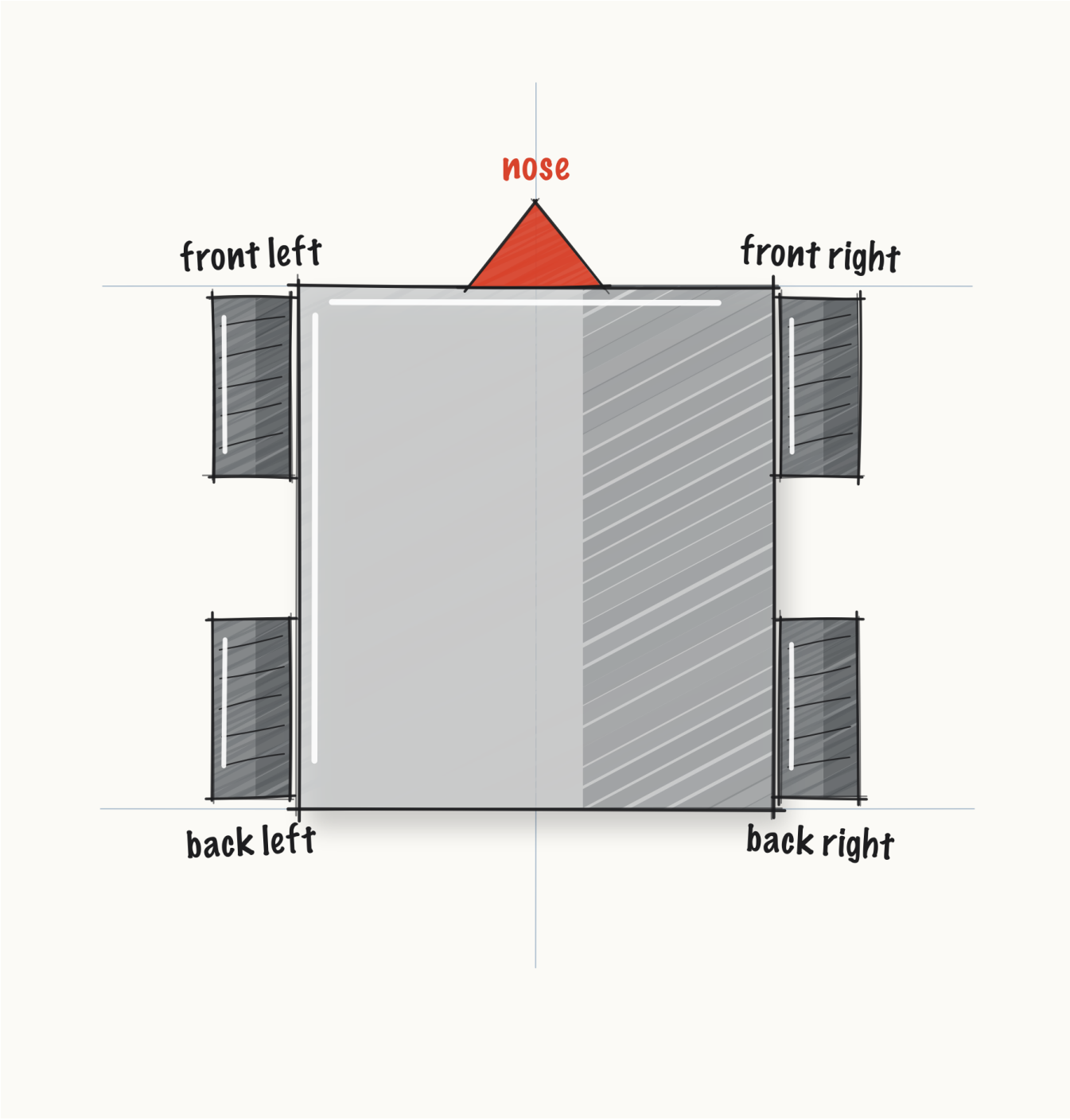
Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

13.1.3 L050S030: label the robot

Label. Put a strip of tape on the robot's front, and mark it FRONT. Mark the left side LEFT and the right side RIGHT, as the robot sees them when it drives forward.



Check.

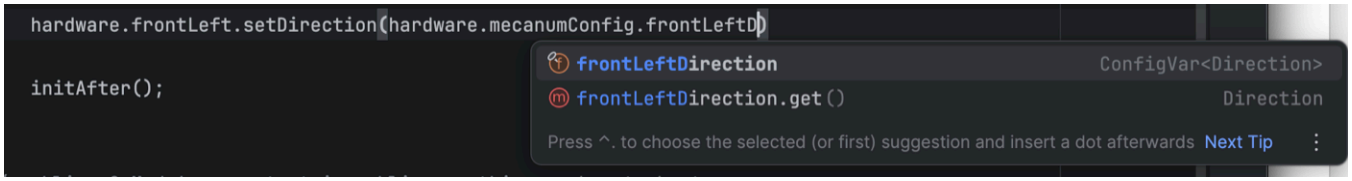
- Stand behind the robot, facing the way FRONT faces. LEFT is on your left.

13.1.4 L050S040: tell each motor which way it is mounted

The two sides of the robot face opposite ways. So the same power turns the left motors one way and the right motors the other. In L040 the simulator did not care. A real robot does.

The robot's own file in the `pedro` folder already says which way each motor is mounted, and `hardware.mecanumConfig` holds what it says.

Change. Scroll up to `init()` and find the comment that names this step. Type `hardware.frontLeft.setD`, and pick `setDirection`. Inside the brackets, type `hardware.mecanumConfig.frontLeftD`. The list offers `frontLeftDirection.get()`, marked `Direction`. Pick it.



Finish the line, and do the same for the other three motors:

```
        // When on L050S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l050.html#l050s040
        // for what to do here.
        hardware.frontLeft.setDirection(hardware.mecanumConfig.frontLeftDirection.get());
        hardware.frontRight.setDirection(hardware.mecanumConfig.frontRightDirection.get());
        hardware.backLeft.setDirection(hardware.mecanumConfig.backLeftDirection.get());
        hardware.backRight.setDirection(hardware.mecanumConfig.backRightDirection.get());
```

`init` runs once, when INIT is pressed, so the motors know their directions before PLAY.

Check.

- Each line ends `.get());`, and none has a red underline.

13.1.5 L050S050: send the code to the robot

The **Control Hub** is the robot's computer: the box the motors plug into. The **Driver Station** is the screen the drivers hold. It picks an OpMode, has the INIT, PLAY and STOP buttons, and passes the gamepad to the robot.

Robot.

- Rest the robot on the blocks, so all four wheels are off the ground.
- Turn it on, and plug the laptop into the Control Hub with the USB cable.

Send.

- In Android Studio, the device list left of **TeamCode** reads **No Devices** until the robot is plugged in. Pick the Control Hub from it.
- Click the green triangle right of **TeamCode**.

Check.

- Android Studio says the install worked, and the Driver Station shows the robot.

13.1.6 L050S060: watch the wheels

Gamepad. Plug the gamepad into the Driver Station. On the gamepad, hold Start and press A. That makes it gamepad 1.

Run. On the Driver Station, pick `L050 Blocks` from the `Lessons` group. Press INIT.

AdvantageScope.

- Join the laptop to the robot's Wi-Fi. Ask a mentor for its name and password the first time.
- Open **AdvantageScope > Settings…** on a Mac. On Windows it is **App > Show Preferences…**. Type `192.168.43.1` in **Robot Address**, and click the ?.
- Open **File > Connect to Robot > NetworkTables 4**.

The first time AdvantageScope talks to the robot, the operating system might ask for your permission. Please give it.

Check.

- The title bar reads `192.168.43.1`, and the list on the left shows `Localizer`.

Run. Press PLAY.

Check.

- Push the left stick a little way forward. In AdvantageScope, `speed/left` goes above 0.0.
- Both left wheels turn, with the top of each tire moving toward FRONT.
- Let go, and do the same with the right stick and the right wheels.

Write down any wheel that turns the other way, or that turns for the other side's stick.

13.1.7 L050S070: set each wheel right

Which robot. Open `Constants`, in the `pedro` folder. Near the top, `thisRobot` names the robot, such as `Robot.BOT_37523`. If it names a different robot from this one, change it to this one.

Turns backward. A wheel that turns backward needs its direction swapped in that robot's own file in the `pedro` folder, such as `Bot37523`. Find its line, such as `c.frontLeftDirection.set(DcMotorSimple.Direction.REVERSE);`, and swap `FORWARD` for `REVERSE`, or `REVERSE` for `FORWARD`.

Turns for the other stick. A wheel that turns for the other side's stick is plugged into a port the Robot Configuration gives a different name. Ask a mentor to check the names on the Driver Station.

Send. Send the code again, as in L050S050, and repeat L050S060.

Check.

- All four wheels pass L050S060.

13.1.8 L050S080: drive it on the floor

Drive. Press STOP. Lift the robot off the blocks and put it on the floor, with room in front of it. Press INIT, then PLAY.

Check.

- Push both sticks gently forward, and the robot drives forward!

- Pull both back, and it comes back.
- Push one forward and pull the other back, and it spins in place.

Live on the robot is for practice. An OpMode in `lessons` or `mytry` turns on the robot's NetworkTables server at INIT, which is what AdvantageScope connects to. FTC rule R704 forbids third-party telemetry over Wi-Fi at competitions, so a competition OpMode lives in another folder, where it stays off.

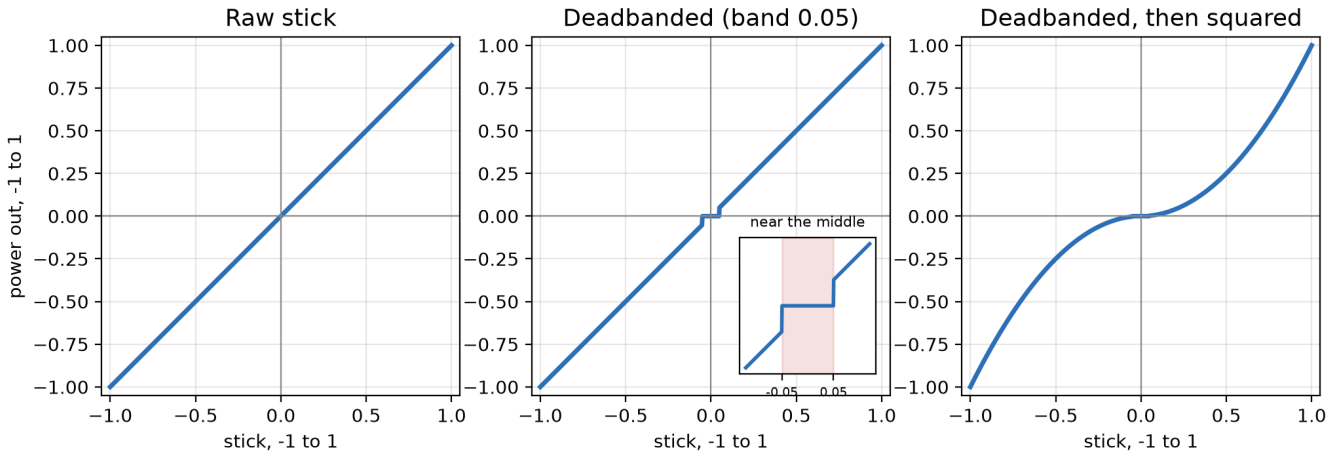
On a Mac, the **Settings…** window is titled **Preferences**.

If it didn't, and the robot creeps with both sticks let go, the sticks do not read exactly 0.0. L060 fixes that.

What you just did. Which way a motor turns is set by how it is bolted on and wired, not by the code. `setDirection` tells each motor which way it is mounted, so from then on a positive power turns every wheel forward. The directions live in one place, the robot's own file, which `thisRobot` in `Constants.java` picks. The code reads them from there, and so will the drivetrains in later lessons and AutoTune.

Where next. L060 stops the creep, and makes a gently pushed stick mean gently.

060 lesson: shape the sticks



Make the robot sit still when the sticks are let go, and make a gentle push mean gently.

Before you start.

- L050 is done, so `mytry` holds `L050BlocksOpMode`, and the robot's wheels all turn the right way.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- The robot and the Driver Station for the last step.

14.1 the steps

14.1.1 L060S010: copy the starter into mytry

Copy. In `lessons`, click `L060ShapingOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L060ShapingOpMode`, and the new file is open.

14.1.2 L060S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

14.1.3 L060S030: write a deadband

A stick that has been let go does not always read 0.0. It can read a few hundredths, and a few hundredths of power is enough to make a robot crawl across the floor on its own. A **deadband** fixes that: any reading closer to the middle than the band counts as let go.

Change. Scroll to the bottom of the file and find the comment that names this step. Under it, write a method of your own:

```
    // When on L060S030, see
    // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l060.html#l060s030
    // for what to do here.
    double deadband(double value, double band) {
        double alwaysPositiveMagnitude = Math.abs(value);
        boolean isCloseToZero = alwaysPositiveMagnitude < band;
        if (isCloseToZero) {
            return 0.0;
        }
        return value;
    }
```

A **method** is a named piece of code you can use again. This one takes two numbers, `value` and `band`, and gives back one number. `return` is the number it gives back.

Each line does one thing, and each result gets a name:

- `Math.abs` takes the minus sign off: `Math.abs(-0.03)` is 0.03. `alwaysPositiveMagnitude` is the size of the reading without its sign. A stick pulled halfway back gives -0.5, and -0.5 is less than 0.05. But it is far from the middle, so it has to be compared without its minus sign.
- `<` compares two numbers, and its answer is true or false. A **`boolean`** holds true or false, the way a `double` holds a number. `isCloseToZero` is true when the size is inside the band.
- `if` runs the lines inside its `{ }` only when what is in its brackets is true. Here, `return 0.0` runs only for a stick close to the middle.

Check.

- The method has no red underline.
- `deadband` is gray, with *no usages* above it. Nothing uses it yet.

14.1.4 L060S040: use the deadband

Change. In `driveFromTheSticks()`, under the comment that names this step, make a power for each side:

```
        // When on L060S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l060.html#l060s040
        // for what to do here.
```

```
        double band = 0.05;
        double leftPower = deadband(leftSpeed, band);
        Tracker.publish("power/left", leftPower);
        double rightPower = deadband(rightSpeed, band);
        Tracker.publish("power/right", rightPower);
```

Then change the four `setPower` lines below, so each one gets the power and not the speed:

```
        hardware.frontLeft.setPower(leftPower);
        hardware.backLeft.setPower(leftPower);

        hardware.frontRight.setPower(rightPower);
        hardware.backRight.setPower(rightPower);
```

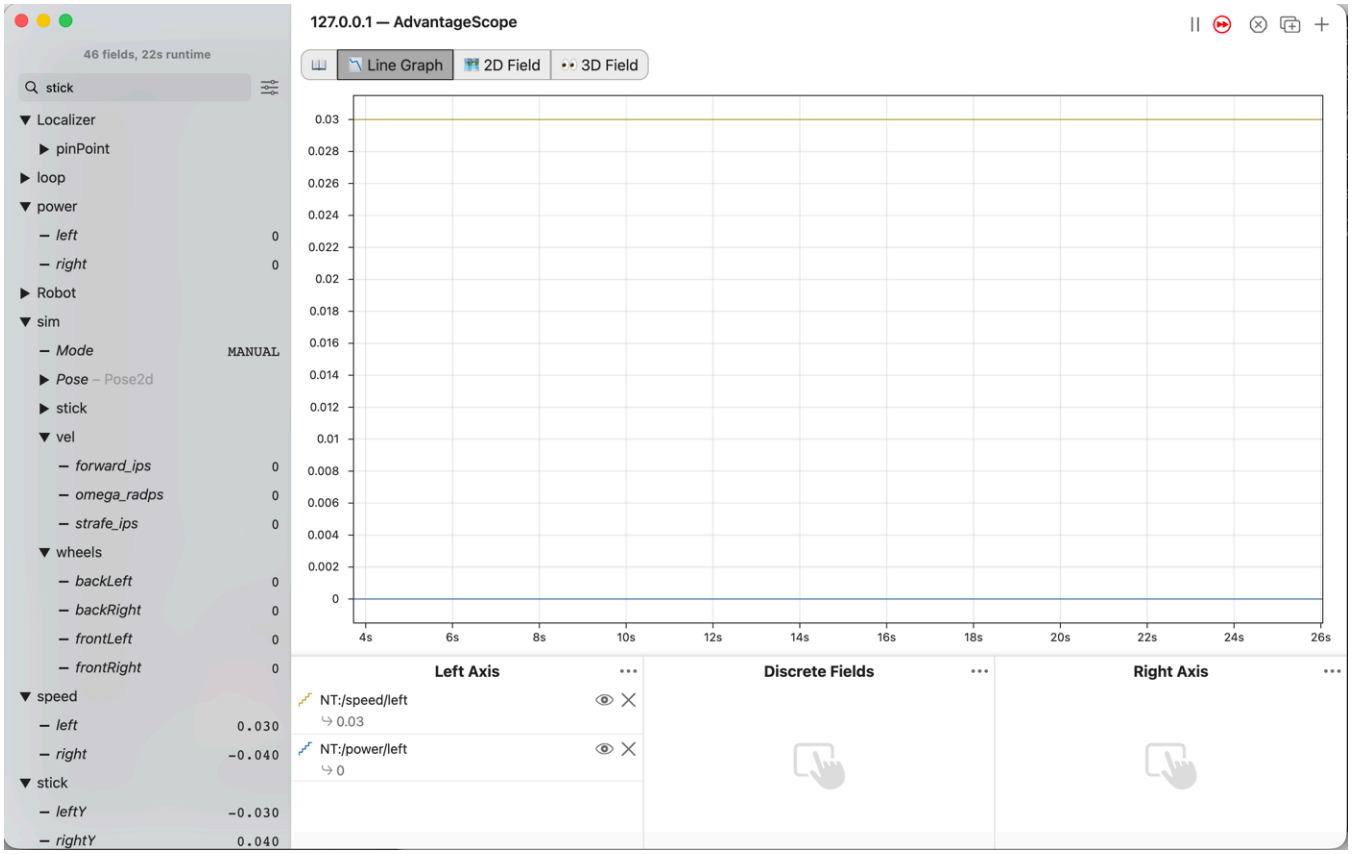
Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L060ShapingOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020. Drag `left` from the `speed` folder and `left` from the `power` folder into **Left Axis**.

Check.

- Let go of the sticks. `speed/left` may sit a little off 0.0, and `power/left` reads 0.0.
- The wheels in `sim/wheels` all read 0.0.



14.1.5 L060S050: write squaring

With the deadband in, a tenth of a push is still a tenth of the power. Near the middle that is more robot than a driver wants. **Squaring** fixes it: half a push becomes a quarter of the power, and a full push is still full power.

Change. Press Ctrl+C in the Terminal to stop the simulator. Find the comment that names this step. Under it, write a second method:

```
    // When on L060S050, see
    // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l060.html#l060s050
    // for what to do here.
    double signedSquared(double value) {
        return value * Math.abs(value);
    }
```

Squaring a number makes it positive. A plus times a plus is a plus. A minus times a minus is a plus too:

- `+0.5 * +0.5` is `+0.25`
- `-0.5 * -0.5` is `+0.25`

Backward would turn into forward. To keep the speed's sign, the method multiplies by the size of the number instead:

- `+0.5 * Math.abs(+0.5)` is `+0.5 * +0.5`, which is `+0.25`
- `-0.5 * Math.abs(-0.5)` is `-0.5 * +0.5`, which is `-0.25`, since a negative times a positive is negative

So it squares the size of the number, its magnitude, and keeps its sign. That is why it is called `signedSquared`.

Check.

- The method has no red underline.

14.1.6 L060S060: use squaring

Change. Go back to the two `deadband` lines from L060S040. Under each one, add a line that squares the power:

```
        double leftPower = deadband(leftSpeed, band);
        leftPower = signedSquared(leftPower);
```

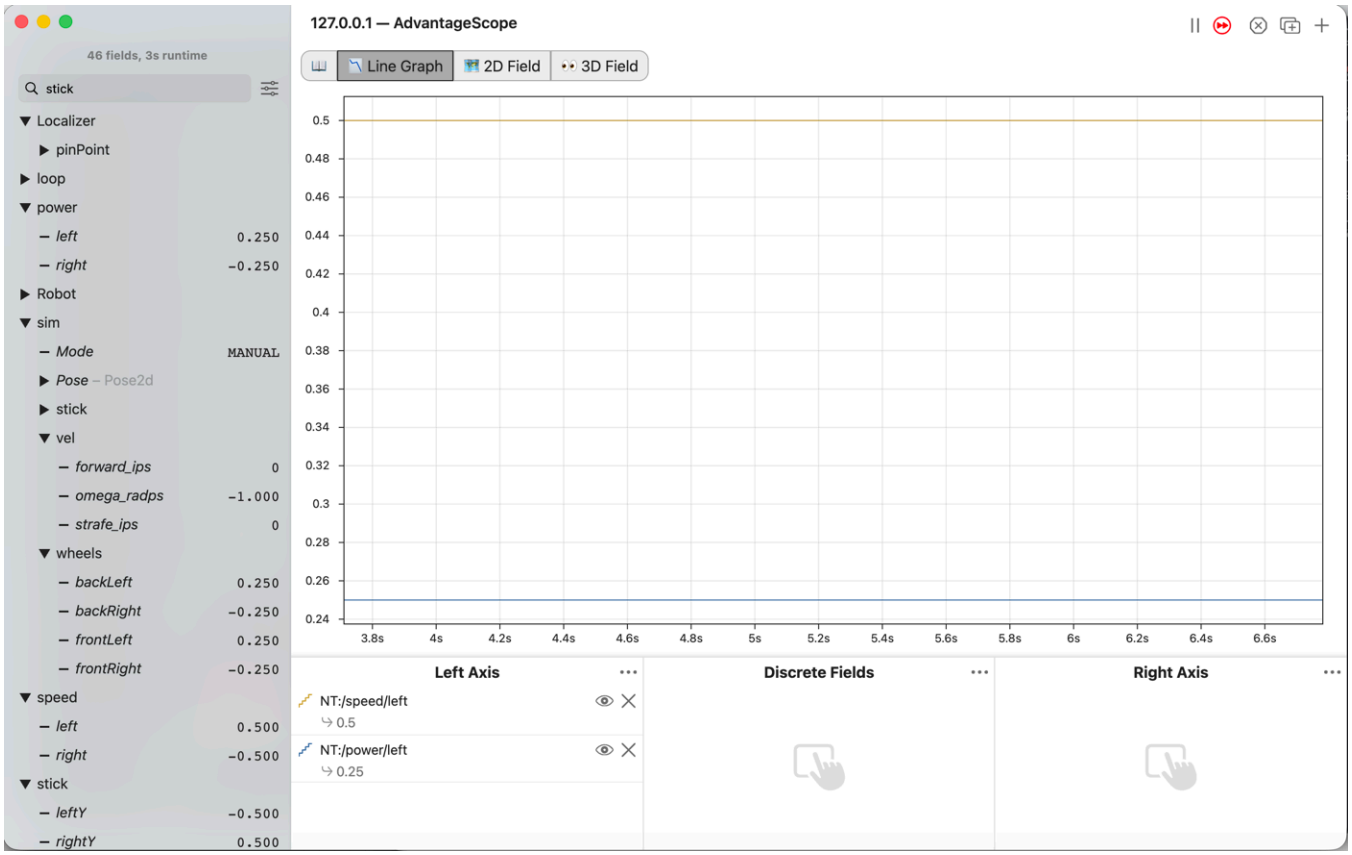
```
        double rightPower = deadband(rightSpeed, band);
        rightPower = signedSquared(rightPower);
```

`signedSquared` works on what `deadband` gave back, and puts its answer back in the same name.

Simulator. Run the same command again, and connect AdvantageScope.

Check.

- Push the left stick halfway forward. `speed/left` reads 0.5, and `power/left` reads 0.25.
- Push it all the way, and both read 1.0.
- Pull it back, and the power goes below 0.0.



14.1.7 L060S070: drive it on the robot

Send.

- Press Ctrl+C in the Terminal to stop the simulator.
- Plug the laptop into the Control Hub with the USB cable.
- Pick the Control Hub from the device list, and click the green triangle right of **TeamCode**.

Check.

- Android Studio says the install worked.

Run. On the Driver Station, pick `L060 Shaping` from the `Lessons` group. Press INIT.

AdvantageScope.

- Join the laptop to the robot's Wi-Fi.
- Open **File > Connect to Robot > NetworkTables 4**. AdvantageScope still has the robot's address from L050.

Check.

- The title bar reads `192.168.43.1`.

Run. Press PLAY. Drag `left` from the `speed` folder and `left` from the `power` folder into **Left Axis**.

Check.

- Let go of both sticks, and the robot sits still, with `power/left` at 0.0.
- Push a stick about a third of the way, and the robot creeps along under control.
- Push it all the way, and the robot goes full speed.

No gamepad. `--args="mytry.L060ShapingOpMode --left_stick_y=-0.03 --right_stick_y=0.04"` holds both sticks just off center; the first picture was taken that way. `--left_stick_y=-0.5 --right_stick_y=0.5` holds them half forward and half back for the second.

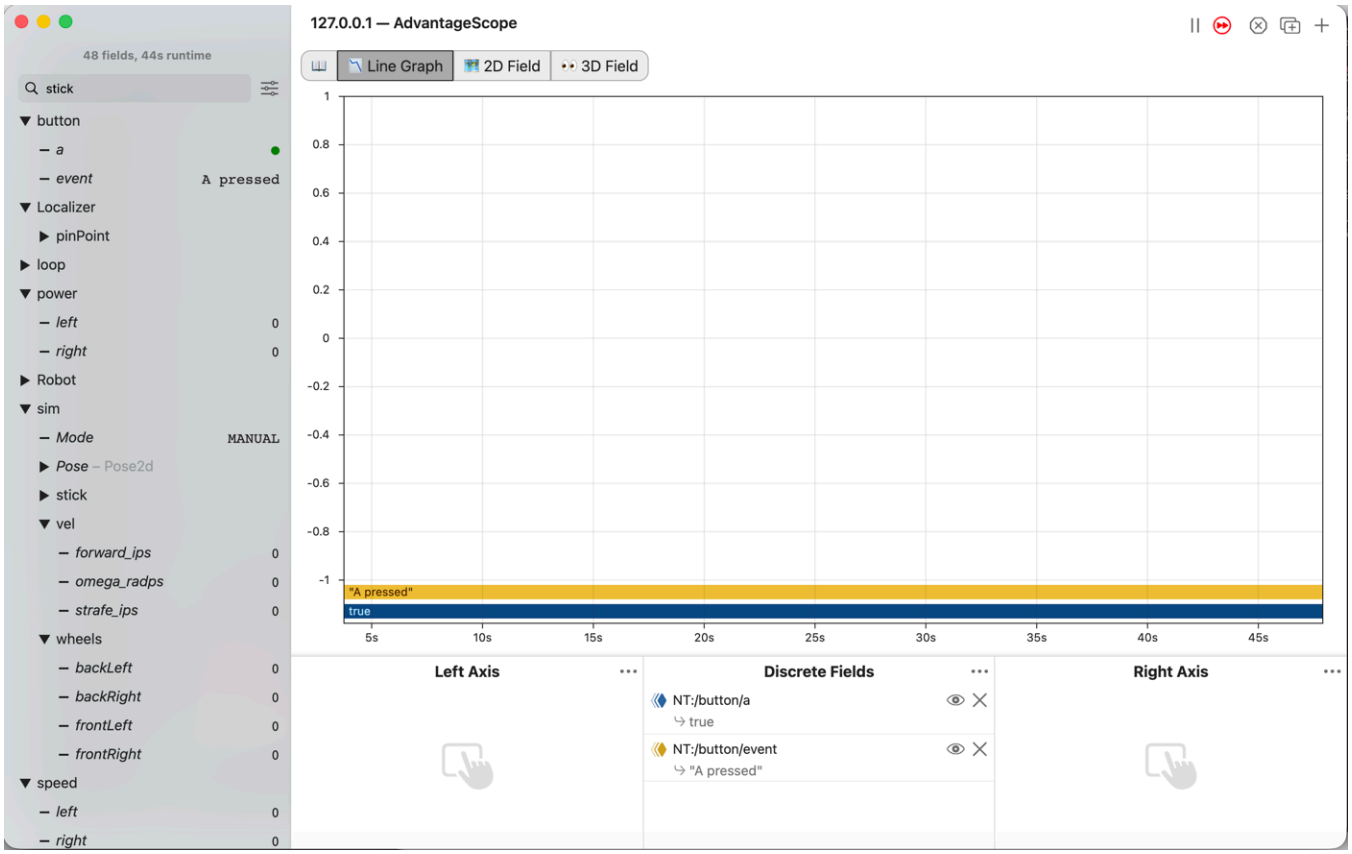
If it didn't, and the robot still creeps on the floor, the band is smaller than what this gamepad reads when it is let go. Raise it a little: 0.05 to 0.1 is the usual range. A band much bigger than that throws away real driving.

If it didn't, and the robot goes forward but will not back up, a sign was lost. Check `signedSquared` uses `Math.abs`, and `deadband` too.

What you just did. Both methods sit between the speed and the wheels. Each one takes a number and gives one back, and neither knows what a wheel is. `speed/left` is still what the stick asks for, and `power/left` is what the wheels get, so AdvantageScope shows both side by side.

Where next. L070 reads the buttons and the sideways sticks.

070 lesson: read the buttons



Read the A button, say the moment it goes down and the moment it comes up, and read both sticks sideways. The robot drives as it did in L060 the whole time.

Before you start.

- L060 is done, so `mytry` holds `L060ShapingOpMode`.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- The robot and the Driver Station for the last step.

15.1 the steps

15.1.1 L070S010: copy the starter into mytry

Copy. In `lessons`, click `L070ButtonsOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L070ButtonsOpMode`, and the new file is open.

15.1.2 L070S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

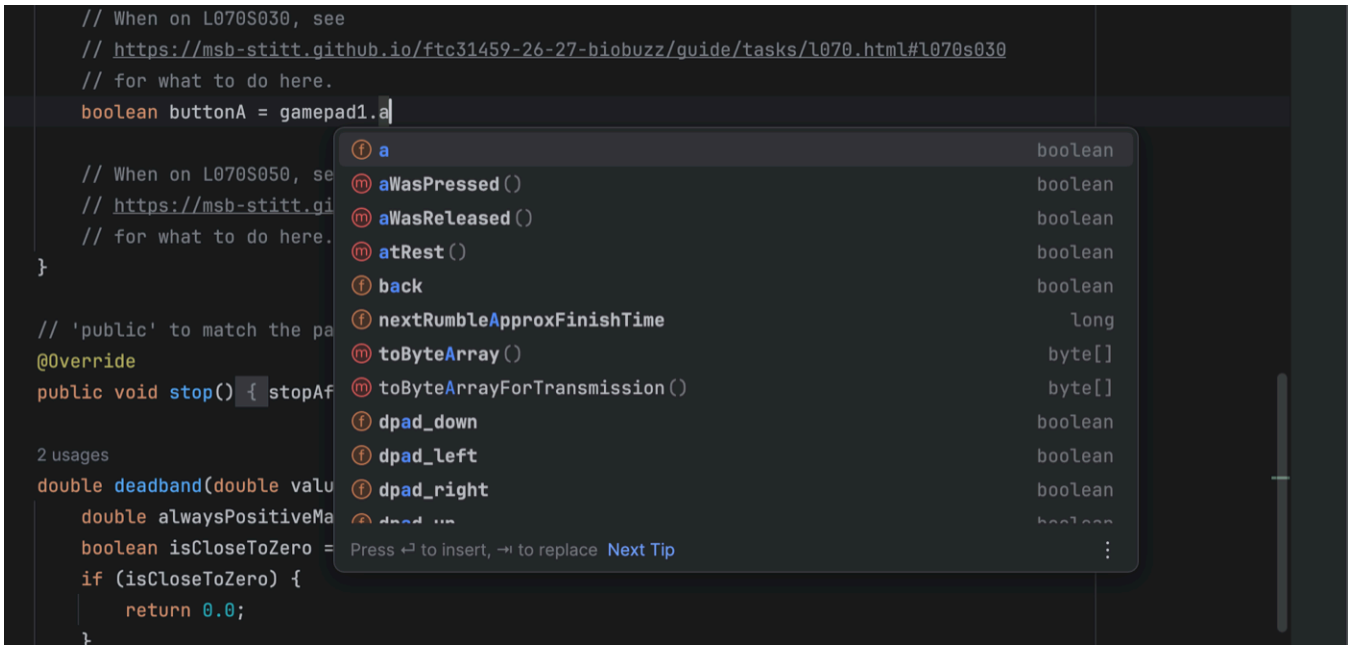
Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

15.1.3 L070S030: read the A button

A stick is a number. A button is only down or up, so Java keeps it as a **boolean**: `true` when it is down, `false` when it is up.

Change. In `driveFromTheSticks()`, find the comment that names this step, below the four `setPower` lines. Under it, type `boolean buttonA = gamepad1.a`. The list offers `a`, marked `boolean`.



Pick `a`, and finish the lines:

```
        // When on L070S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l070.html#l070s030
        // for what to do here.
        boolean buttonA = gamepad1.a;
        Tracker.publish("button/a", buttonA);
```

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L070ButtonsOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020. Open the `button` folder, and drag `a` into **Discrete Fields**.

Check.

- `button/a` is a red dot.
- Press A and hold it, and the dot turns green.
- Let go, and it turns red again.

15.1.4 L070S040: remember last loop's button

`buttonA` says whether A is down in this loop. To know the moment it changed, the code also needs to know whether it was down in the loop before. A variable made inside `driveFromTheSticks()` is gone when it ends, so this one goes outside it.

Change. Press Ctrl+C in the Terminal to stop the simulator. Scroll to the top of the class, and find the comment that names this step. Under it, type:

```
    // When on L070S040, see
    // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l070.html#l070s040
    // for what to do here.
    boolean previousButtonA;
```

A variable made here is a **field**. It keeps its value from one loop to the next. A `boolean` field starts out `false`.

Check.

- The line has no red underline.

15.1.5 L070S050: say when the button changes

Change. Back in `driveFromTheSticks()`, find the comment that names this step. Under it, type:

```
        // When on L070S050, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l070.html#l070s050
        // for what to do here.
        if (buttonA != previousButtonA) {
            if (buttonA) {
                Tracker.publish("button/event", "A pressed");
            } else {
                Tracker.publish("button/event", "A released");
            }
        }
        previousButtonA = buttonA;
```

`!=` means "is not the same as". The first `if` is true only on the loop where the button changed. The second one says which way it changed. `else` is what runs when the `if` is false. The last line saves this loop's button for the next loop.

The text in quotes is a **String**. `Tracker` writes it down the same way it writes a number.

Simulator. Run the same command again, and connect AdvantageScope.

AdvantageScope. Drag `event` from the `button` folder into **Discrete Fields**. Until A has been pressed once, `event` is crossed out, because nothing has been written to it yet.

Holding A down does not write `A pressed` again. `event` only changes on the loop the button changed.

Check.

- Press A and hold it. `button/a` turns into a green dot, and `event` reads `A pressed`.
- Let go. The dot turns red, and `event` reads `A released`.

15.1.6 L070S060: read the sticks sideways

Each stick also moves left and right. Pushed right, it reads above 0.0. Pushed left, it reads below 0.0. Tank drive does not use these yet. Later lessons drive sideways with them.

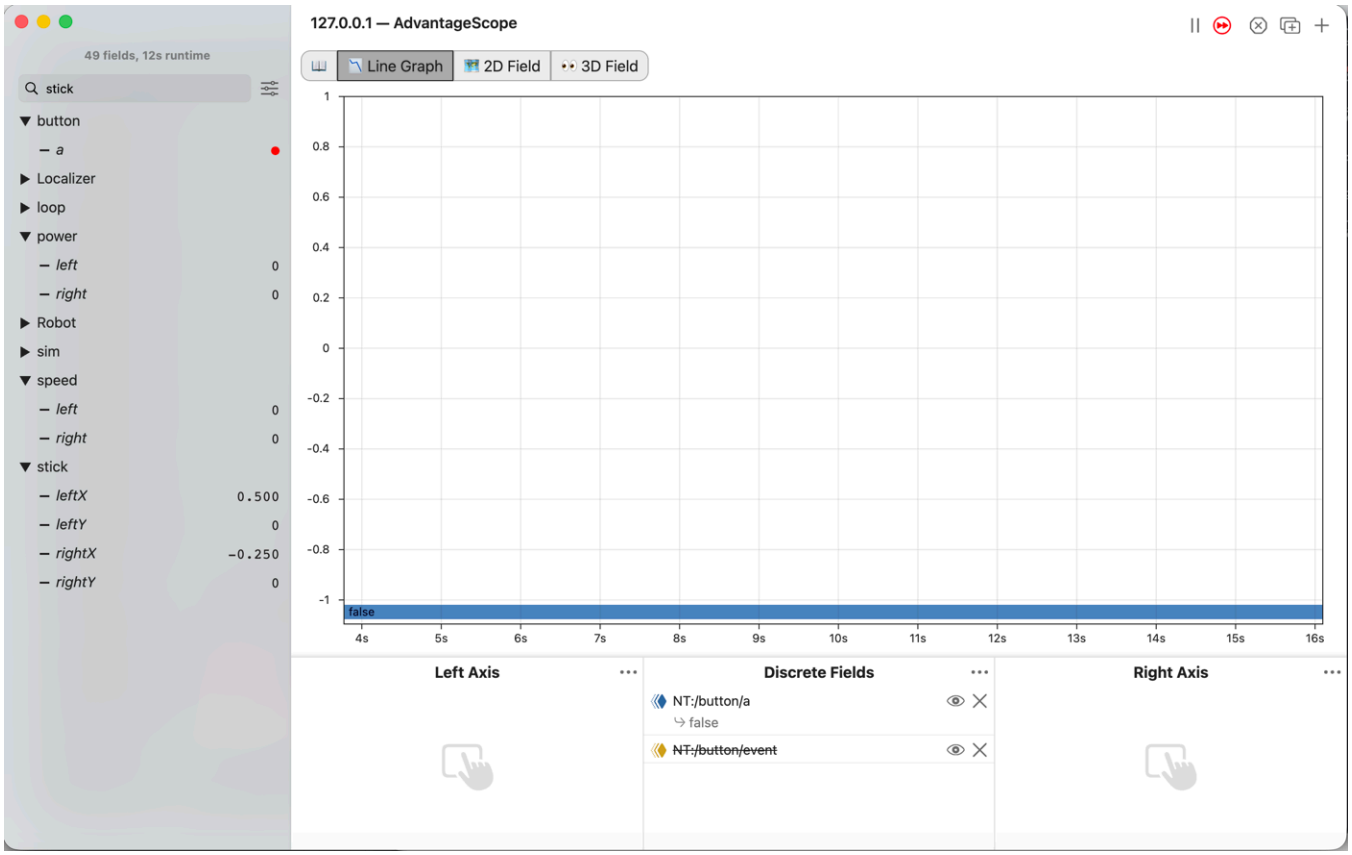
Change. Press Ctrl+C in the Terminal to stop the simulator. Near the top of `driveFromTheSticks()`, find the comment that names this step, under the lines that read the sticks up and down. Under it, type:

```
        // When on L070S060, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l070.html#l070s060
        // for what to do here.
        double leftStickX = gamepad1.left_stick_x;
        Tracker.publish("stick/leftX", leftStickX);
        double rightStickX = gamepad1.right_stick_x;
        Tracker.publish("stick/rightX", rightStickX);
```

Simulator. Run the same command again, and connect AdvantageScope.

Check.

- Push the left stick to the right, and `stick/leftX` goes above 0.0.
- Push the right stick to the left, and `stick/rightX` goes below 0.0.
- `speed` and `power` stay at 0.0, and the robot does not move.



15.1.7 L070S070: watch the buttons on the robot

Robot. Stop the simulator. Send the code, run `L070 Buttons` from the Driver Station, and watch it in AdvantageScope.

Check.

- Press A, and in AdvantageScope `button/event` reads `A pressed`.
- Let go, and it reads `A released`.
- The robot still drives with the sticks as it did in L060.

No gamepad. `--args="mytry.L070ButtonsOpMode --a=true"` holds A down for the whole run, so `event` reads `A pressed` and never changes; the picture at the top was taken that way. A release needs a gamepad. `--left_stick_x=0.5 --right_stick_x=-0.25` holds the sticks sideways for the last picture. The tests press and let go of A without a gamepad.

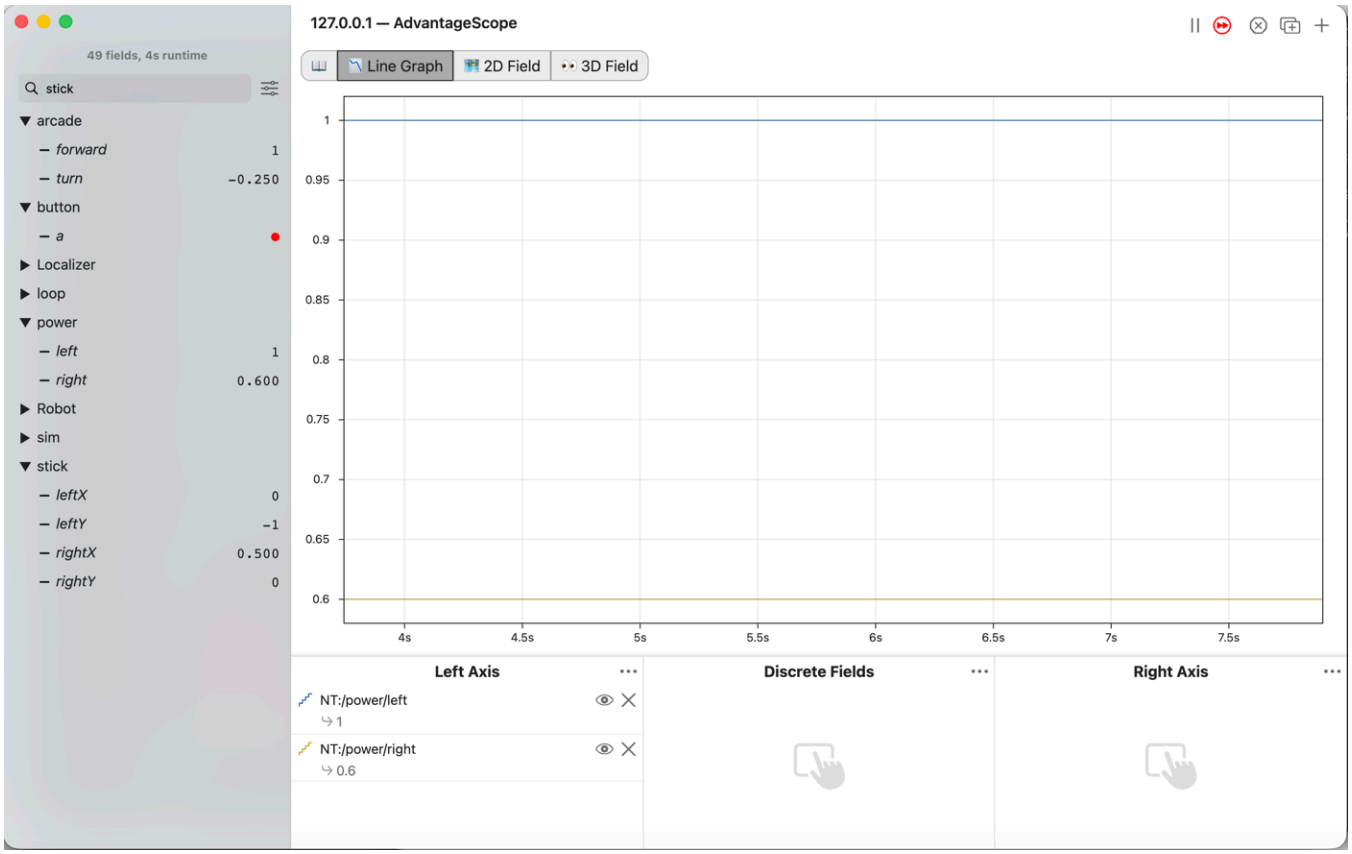
If it didn't, and `event` reads `A pressed` but never `A released`, the last line, `previousButtonA = buttonA;`, is missing. The test fails with `letting go says so`.

If it didn't, and `event` reads `A pressed` when A is let go, the two `Tracker.publish` lines are swapped.

What you just did. A boolean is either `true` or `false`. A field keeps a value between loops, and comparing this loop with the last one finds the moment something changed. That is how a button press can do one thing once, rather than once every loop it is held.

Where next. L080 drives the robot with one stick for going and one for turning.

080 lesson: drive like an arcade game



Drive with one stick for going and one for turning: arcade drive. When the sticks ask for more power than a motor has, scale every wheel down together, so the robot still drives the curve the driver asked for.

Before you start.

- L070 is done, so `mytry` holds `L070ButtonsOpMode`.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- The robot, the Driver Station, and about 3 m of clear floor for the last step.

16.1 the steps

16.1.1 L080S010: copy the starter into mytry

Copy. In `lessons`, click `L080ArcadeOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L080ArcadeOpMode`, and the new file is open.

16.1.2 L080S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

16.1.3 L080S030: forward and turn

Tank drive asks the driver to think in left wheels and right wheels. Arcade drive asks for forward and turn instead, which is how a driver already thinks. The code works out the two sides.

The starter has the sticks, the buttons, `deadband` and `squared` from the lessons before. It does not drive the wheels yet.

Change. In `driveFromTheSticks()`, find the comment that names this step. Under it, type:

```
        // When on L080S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l080.html#l080s030
        // for what to do here.
        double band = 0.05;
        double forward = deadband(-leftStickY, band);
        forward = signedSquared(forward);
        Tracker.publish("arcade/forward", forward);
        double turn = deadband(-rightStickX, band);
        turn = signedSquared(turn);
        Tracker.publish("arcade/turn", turn);
```

`forward` comes from the left stick, up and down. `turn` comes from the right stick, sideways. Both get a minus sign. For `forward`, that makes pushing away from you positive, as in L030. For `turn`, it makes a turn to the left positive. That is the way robots measure turning: left, or counter-clockwise, is positive.

Simulator. In the Terminal, run:

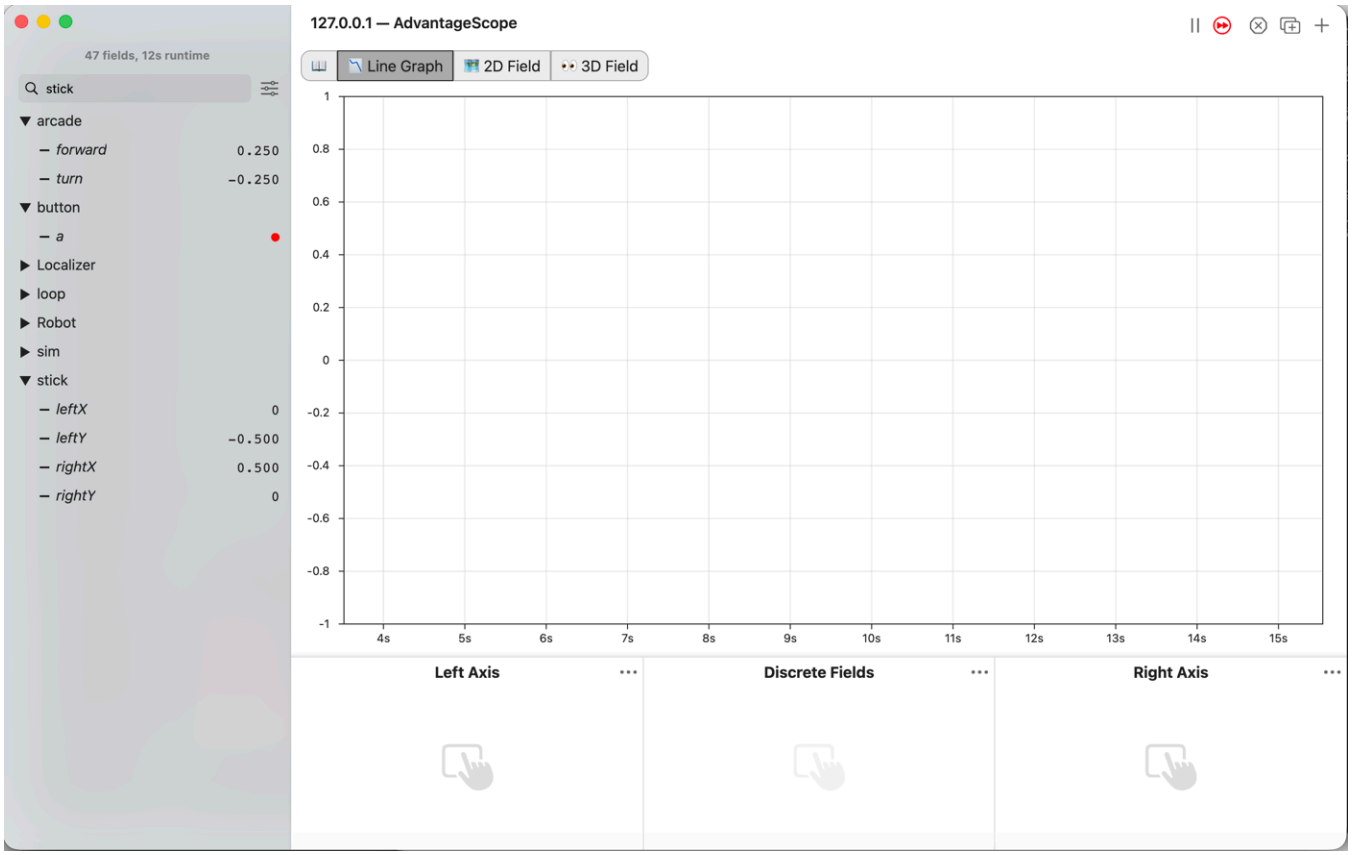
```
./gradlew :TeamCode:simRun --args="mytry.L080ArcadeOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020, and open the `arcade` folder.

Check.

- Push the left stick halfway forward, and `forward` reads 0.25.
- Push the right stick halfway right, and `turn` reads -0.25.

- The robot does not move yet.



16.1.4 L080S040: mix them into two sides

To turn right, the left side goes faster and the right side goes slower. A turn to the right is below 0.0, so taking `turn` away from the left side speeds it up, and adding it to the right side slows that side down.

Change. Press Ctrl+C in the Terminal to stop the simulator. Find the comment that names this step. Under it, type:

```
        // When on L080S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l080.html#l080s040
        // for what to do here.
        double leftPower = forward - turn;
        double rightPower = forward + turn;
        Tracker.publish("power/left", leftPower);
        Tracker.publish("power/right", rightPower);

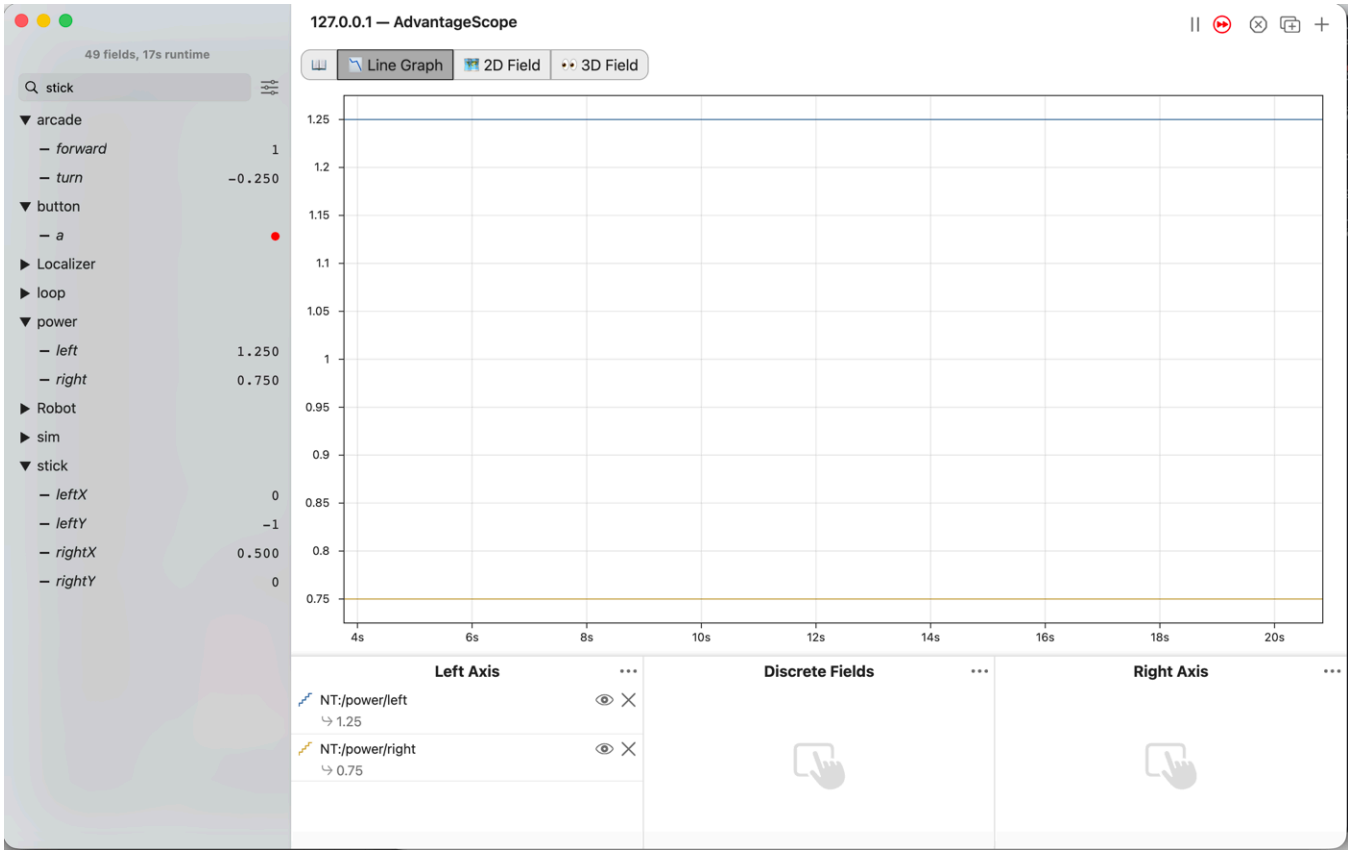
        hardware.frontLeft.setPower(leftPower);
        hardware.backLeft.setPower(leftPower);
        hardware.frontRight.setPower(rightPower);
        hardware.backRight.setPower(rightPower);
```

Simulator. Run the same command again, and connect AdvantageScope. Click **2D Field**. The robot from L040 is still in **Poses**.

Check.

- Push the left stick forward, and the robot drives forward.
- Push the right stick right, and it turns right.
- Push both, and it drives a curve.

Now push the left stick all the way forward and the right stick halfway right. Click **Line Graph**, and drag `left` and `right` from the `power` folder into **Left Axis**. `power/left` reads 1.25.



A motor's power only goes from -1.0 to +1.0. On the robot, the Control Hub cuts 1.25 down to 1.0 and leaves 0.75 alone. The left side loses some of what it asked for and the right side loses none, so the robot drives a wider curve than the sticks asked for. The simulator does not cut it, so the curve there looks right. The number gives it away.

16.1.5 L080S050: scale, don't cut

The fix is to divide both sides by the bigger one. 1.25 and 0.75 become 1.0 and 0.6. The robot goes a little slower, and the curve stays the one the driver asked for.

Change. Press Ctrl+C to stop the simulator. Go back to the two lines from L080S040 that make `leftPower` and `rightPower`. Under them, type:

```
        double leftPower = forward - turn;
        double rightPower = forward + turn;
        double leftSize = Math.abs(leftPower);
        double rightSize = Math.abs(rightPower);
        double biggest = Math.max(leftSize, rightSize);
        biggest = Math.max(1.0, biggest);
```

```
        leftPower = leftPower / biggest;
        rightPower = rightPower / biggest;
```

`leftSize` and `rightSize` are the powers with their minus signs taken off. `Math.max` gives back the bigger of two numbers, so the first `Math.max` finds the bigger size. The second makes sure `biggest` is never less than 1.0. Without it, a gentle 0.25 would be divided by 0.25 and become full power.

The last three lines have no `double` in front. `biggest` and `leftPower` already exist, so these lines change them rather than making new ones.

Simulator. Run the same command again, and connect AdvantageScope.

Check.

- Push the left stick all the way forward and the right stick halfway right. `power/left` reads 1.0, and `power/right` reads 0.6.
- Push the left stick halfway forward alone, and both read 0.25, the same as before this step.

16.1.6 L080S060: drive it on the robot

Robot. Stop the simulator. Send the code, and run `L080 Arcade` from the Driver Station.

Check.

- Push the left stick halfway forward and the right stick a quarter right, and the robot drives a circle to the right.
- Push both sticks twice as far, and it drives a circle the same size, only faster.

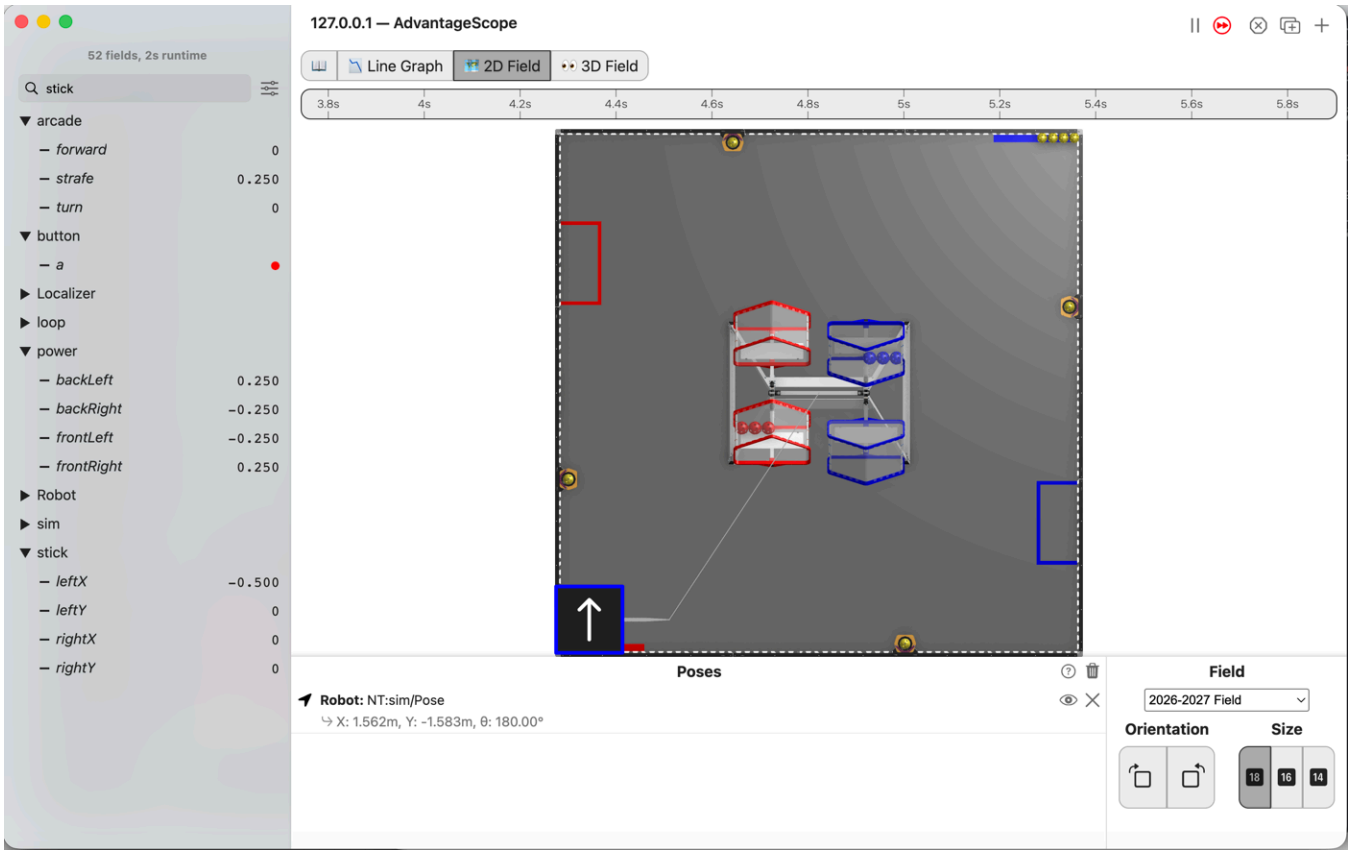
If it didn't, and the robot turns left when the right stick is pushed right, the minus sign is missing from `-rightStickX`, or − and + are swapped in the two mix lines.

If it didn't, and the robot goes full speed with a gentle push, `biggest` starts below 1.0. Check the `1` in the outer `Math.max`.

What you just did. Arcade drive mixes two things the driver wants into two things the wheels do. When the mix asks for more than a motor has, dividing keeps the shape of the request and gives up some speed. Cutting at 1.0 keeps the speed and changes the shape. The Control Hub cuts; it is the code's job to scale first.

Where next. L090 drives a mecanum robot, which adds a third thing to do with the sticks: going sideways.

090 lesson: slide it sideways



Drive a mecanum robot: the left stick drives and slides, the right stick turns, and all three can happen at once. Each wheel gets its own power.

Before you start.

- L080 is done, so `mytry` holds `L080ArcadeOpMode`.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- The robot on blocks, as in L050, the Driver Station, and about 2 m by 2 m of clear floor for the last step.

17.1 the steps

17.1.1 L090S010: copy the starter into mytry

Copy. In `lessons`, click `L090MecanumOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L090MecanumOpMode`, and the new file is open.

17.1.2 L090S020: switch your copy on

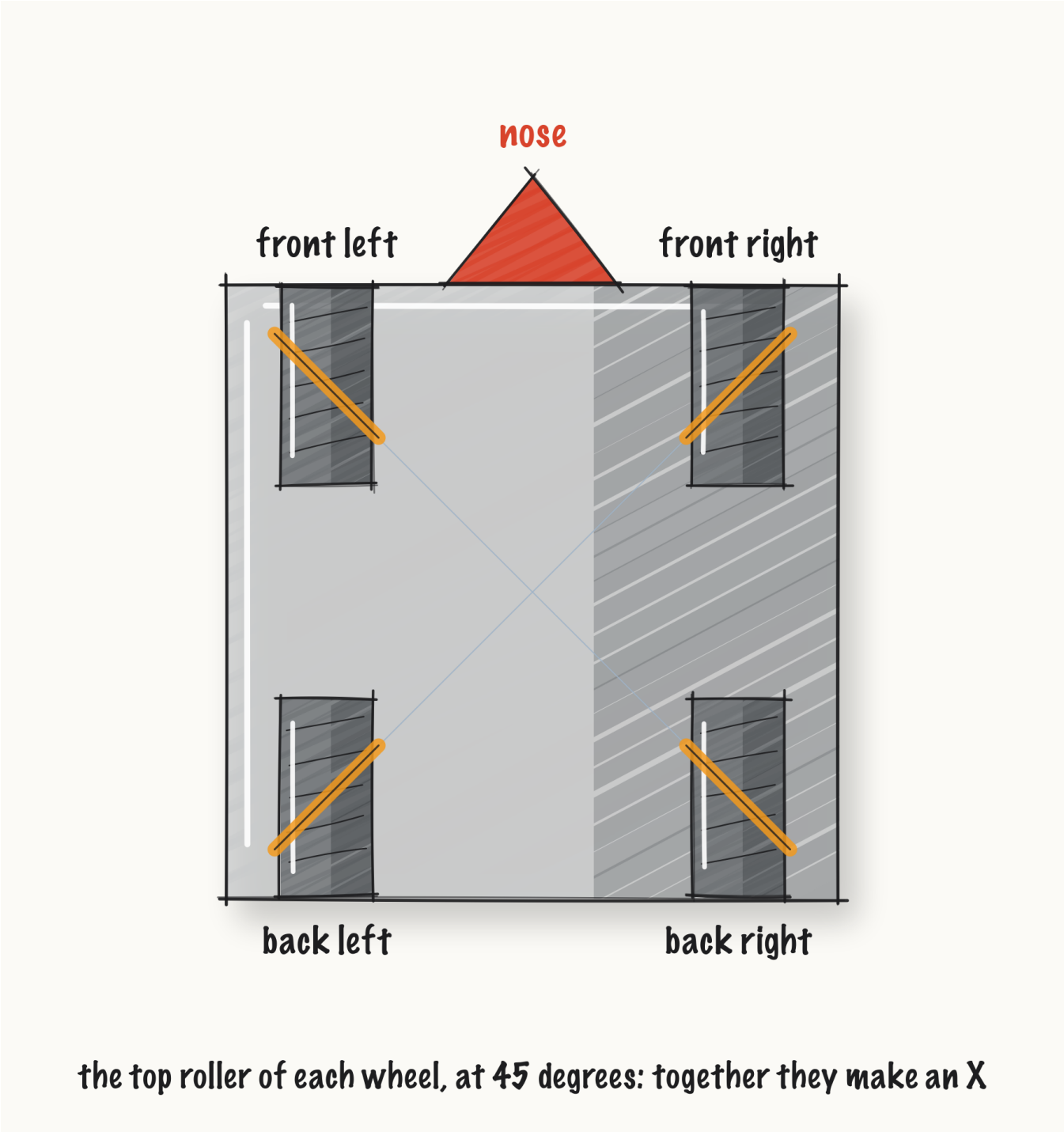
Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

17.1.3 L090S030: read the slide

A mecanum wheel has small rollers around it, set at 45 degrees. Spinning the wheel pushes the robot along and sideways at the same time. With four of them, the robot can drive, slide and turn.



Seen from above, the top rollers make an X. Game Manual 0 shows it on this page. A mecanum wheel is a left wheel or a right wheel, and the two are mirror images. goBILDA's page shows both: https://www.gobilda.com/96mm-mecanum-wheel-set-70a-durometer-bearing-supported-rollers/

The starter has `forward` and `turn` from L080. The slide is the third number, and it comes from the left stick, sideways.

Change. In `driveFromTheSticks()`, find the comment that names this step. Under it, type:

```
        // When on L090S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l090.html#l090s030
        // for what to do here.
        double strafe = deadband(-leftStickX, band);
        strafe = signedSquared(strafe);
        Tracker.publish("arcade/strafe", strafe);
```

Strafe is the robot word for sliding sideways. The stick reads above 0.0 to the right, and the minus sign makes a slide to the left positive, the same way a turn to the left is.

Check.

- Neither line has a red underline.

17.1.4 L090S040: give each wheel its power

Each wheel adds up the three numbers its own way:

Wheel	forward	strafe	turn
front left	+	-	-
front right	+	+	+
back left	+	+	-
back right	+	-	+

Change. Find the comment that names this step. Under it, type:

```
        // When on L090S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l090.html#l090s040
        // for what to do here.
        double frontLeftPower = forward - strafe - turn;
        double frontRightPower = forward + strafe + turn;
        double backLeftPower = forward + strafe - turn;
        double backRightPower = forward - strafe + turn;
        Tracker.publish("power/frontLeft", frontLeftPower);
        Tracker.publish("power/frontRight", frontRightPower);
        Tracker.publish("power/backLeft", backLeftPower);
        Tracker.publish("power/backRight", backRightPower);

        hardware.frontLeft.setPower(frontLeftPower);
        hardware.frontRight.setPower(frontRightPower);
        hardware.backLeft.setPower(backLeftPower);
        hardware.backRight.setPower(backRightPower);
```

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L090MecanumOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020, and click **2D Field**. Open the `arcade` and `power` folders.

Check.

- Push the left stick halfway left. `strafe` reads 0.25.
- The front right and back left wheels read 0.25, and the other two read -0.25.
- The robot slides to its own left, and its nose keeps pointing the same way!
- Forward and turning work as they did in L080.

17.1.5 L090S050: scale all four

Push the left stick into a corner, all the way forward and all the way left. `forward` and `strafe` are both 1.0, so the front right and back left wheels ask for 2.0. That is twice what a motor has.

Change. Press Ctrl+C in the Terminal to stop the simulator. Go back to the four lines that make the wheel powers. Under them, type:

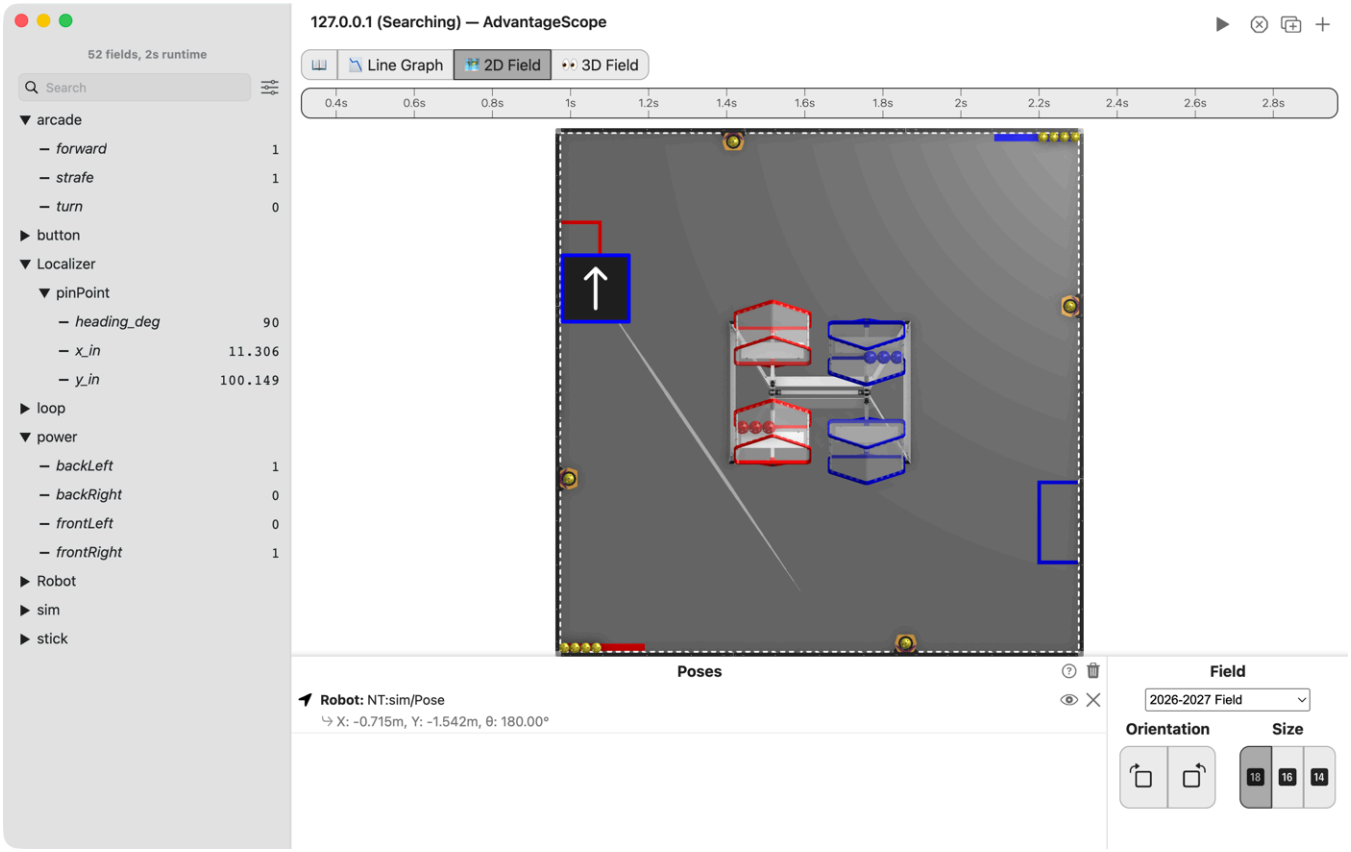
```
        double frontLeftPower = forward - strafe - turn;
        double frontRightPower = forward + strafe + turn;
        double backLeftPower = forward + strafe - turn;
        double backRightPower = forward - strafe + turn;
        double frontLeftSize = Math.abs(frontLeftPower);
        double frontRightSize = Math.abs(frontRightPower);
        double backLeftSize = Math.abs(backLeftPower);
        double backRightSize = Math.abs(backRightPower);
        double biggest = Math.max(frontLeftSize, frontRightSize);
        biggest = Math.max(biggest, backLeftSize);
        biggest = Math.max(biggest, backRightSize);
        biggest = Math.max(1.0, biggest);
        frontLeftPower = frontLeftPower / biggest;
        frontRightPower = frontRightPower / biggest;
        backLeftPower = backLeftPower / biggest;
        backRightPower = backRightPower / biggest;
```

This is L080's scaling, for four wheels. `Math.max` takes two numbers, so finding the biggest of four sizes takes three calls: the first compares the two front wheels, and each of the next two brings in one more wheel. The last `Math.max` keeps `biggest` from going below 1.0, as in L080.

Simulator. Run the same command again, and connect AdvantageScope.

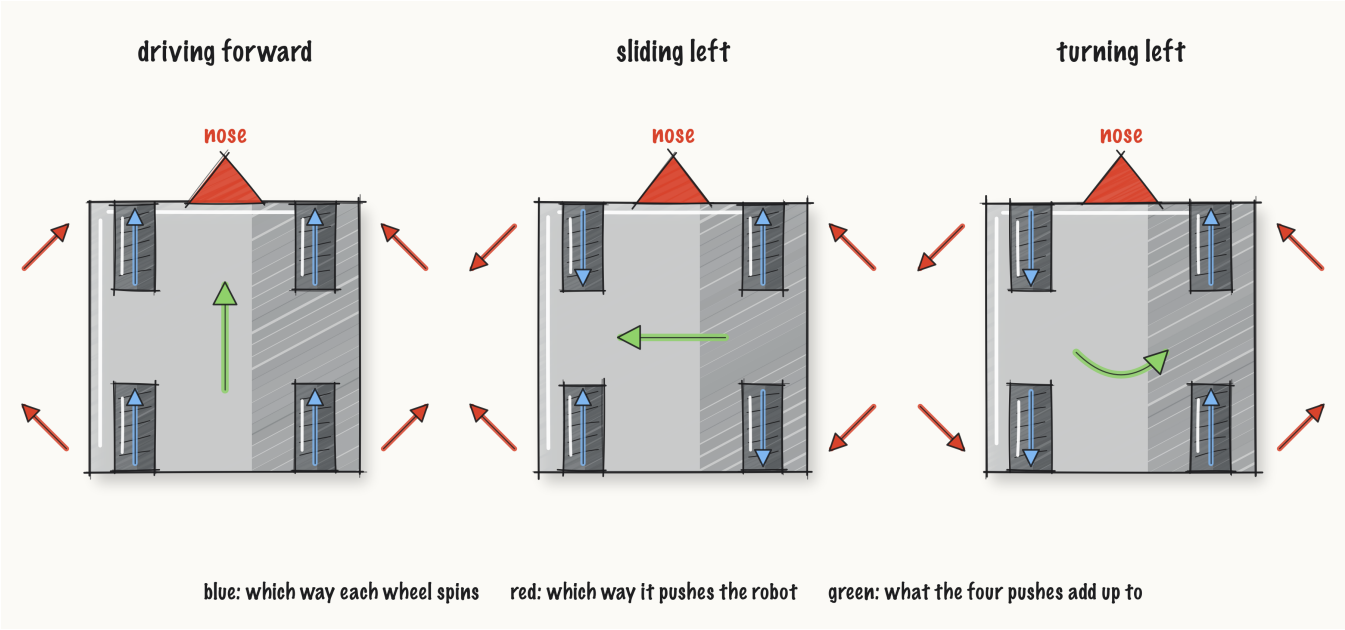
Check.

- Push the left stick into the forward-left corner. The front right and back left wheels read 1.0, and the other two read 0.0.
- The robot drives diagonally, forward and to its left, with its nose still pointing forward.



17.1.6 L090S060: check every wheel on blocks

A mecanum wheel spinning forward pushes the robot forward and sideways at once. The front left and back right wheels push forward and to the right. The front right and back left wheels push forward and to the left. Spinning backward, each pushes the opposite way. The robot moves the way the four pushes add up.



Robot. Stop the simulator. Rest the robot on the blocks, so all four wheels are off the ground. Send the code, and run `L090 Mecanum` from the Driver Station.

Check.

- Push each way gently and watch the top of each tire:

Push	front left	front right	back left	back right
left stick forward	forward	forward	forward	forward
left stick left	backward	forward	forward	backward
right stick left	backward	forward	backward	forward

Left stick forward: all four push forward, and their sideways pushes cancel. Left stick left: the forward and backward pushes cancel, and all four push left. Right stick left: the left side pushes backward and the right side forward, so the robot turns left.

Write down any wheel that turns the other way. A wheel that turns backward for every push needs its direction swapped, as in L050S070. A wheel that is right for forward but backward for a slide is on the other corner. Check the X from above.

17.1.7 L090S070: drive it on the floor

Drive. Press STOP. Lift the robot off the blocks and put it on the floor. Press INIT, then PLAY.

Check.

- Push the left stick left, and the robot slides to its own left with its nose still pointing the same way.
- Push it into a corner, and the robot drives diagonally.
- Push the right stick while sliding, and it turns as it slides.

If it didn't, and the robot slides right when the stick goes left, the minus sign is missing from `-leftStickX`, or the strafe signs are swapped in the four lines.

If it didn't, and the robot turns or grinds instead of sliding, on the floor, a wheel is on the wrong corner. Forward still works with a wheel in the wrong place, because all four still push forward. It is the sideways pushes that stop canceling. This is what the arithmetic says. No robot here has been driven with a wheel on the wrong corner.

What you just did. Three numbers, four wheels, one line each. Those four lines are the whole of mecanum drive, and the path follower in later lessons uses the same four. The scaling is L080's, for four wheels instead of two.

Where next. L110 hands the wheels to the path follower.

093 how-to: watch the numbers while it drives

18.1 what you will have when this is done

The robot's own numbers on a screen while somebody drives it. You get a table of every value the lesson sends, a graph of any two of them, and a picture of where the robot thinks it is. After this, "it went crooked" is a graph you can point at. It stops being an argument.

18.2 before you start

- L060 is finished, because step 3 graphs the two numbers L060's squaring makes.
- A robot with its battery in, running a lesson.
- A laptop or a phone that can join the robot's Wi-Fi.

18.3 the steps

18.3.1 step 1: open Panels

Join the robot's Wi-Fi, then open `http://192.168.43.1:8001`.

Panels is a web page the Robot Controller app serves. Nothing in the lesson starts it and nothing turns it on.

You'll know it worked when the page loads and shows its panels, even before an OpMode runs.

If it didn't, and the page never loads, the laptop is on the wrong Wi-Fi. Check which network it joined.

If it didn't, and you get a different page full of tuning buttons, that is AutoTune on port 10158. It is a second server in the same app, and it is not this.

18.3.2 step 2: find the numbers your lesson sends

Run `L060 Shaping`, the copy finished in L060, and press PLAY. The Telemetry panel fills with one line per value, each shaped `name: number`.

The Graph panel has a list to pick from, and a name only joins that list **after it has been sent once**. So start the OpMode first, then go looking. The list is flat, in the order the names first arrived.

You'll know it worked when `speed/left` and `power/left` are both in the list, and both numbers move when you move the left stick. The cheat sheet lists every name each lesson sends.

If it didn't, and a name you expected is missing, nothing sent it. Either that lesson does not publish it, or the line that publishes it is not typed yet.

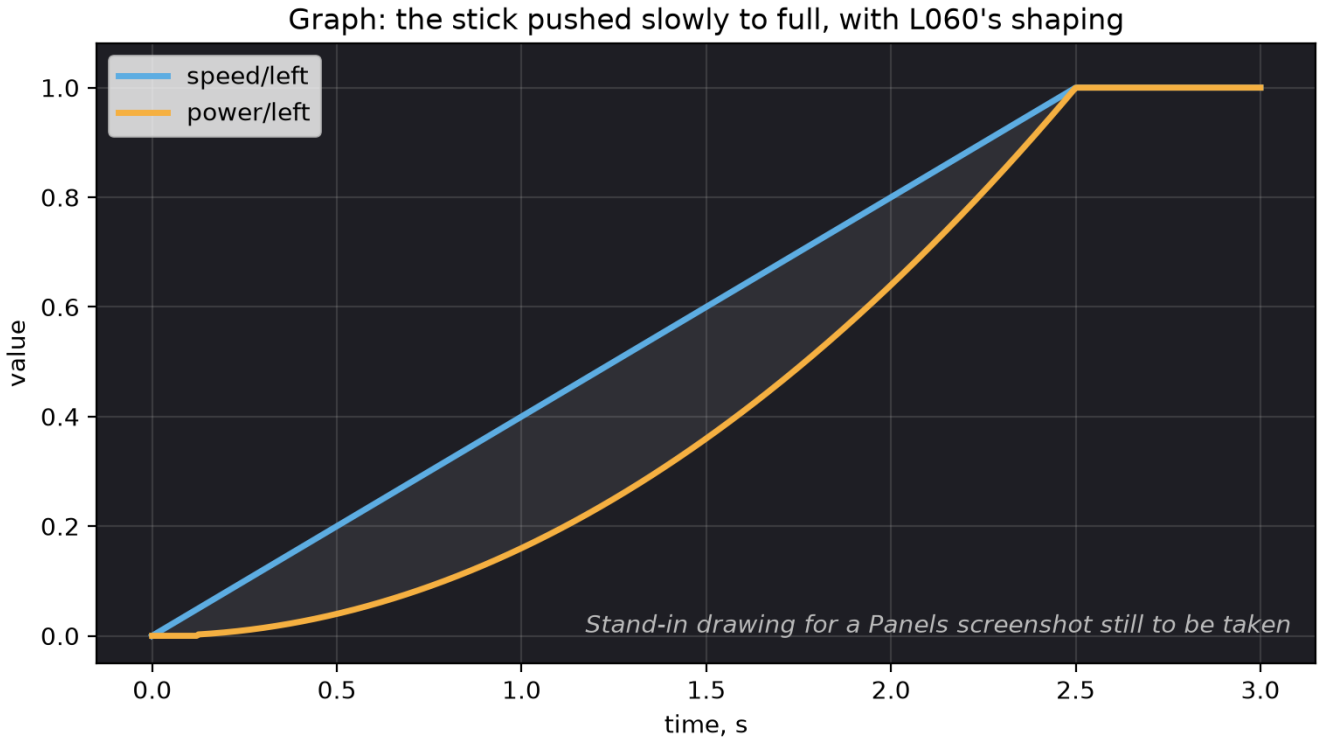
18.3.3 step 3: graph the two against each other

Pick `speed/left` and `power/left` and push the stick slowly from nothing up to full.

You'll know it worked when the two lines leave zero together and the shaped one stays under the raw one the whole way up, until they meet at full stick. That is L060 drawn: squaring a number below 1 makes it smaller.

If it didn't, and the two lines lie exactly on top of each other, nothing is being shaped. That is L060S060 not typed yet.

If it didn't, and both lines look like a staircase rather than a curve, that is the sampling and it is normal. The robot sends to Panels every 75 ms and drops the loops in between, so with a loop of about 5 ms, roughly one loop in 15 reaches the graph. For the same reason one slow loop will not show in `loop/period_ms`. Graph `loop/max_period_ms` instead: it only ever steps up, and a step means some loop blocked.



18.3.4 step 4: look at where it thinks it is

Open the Field panel. There is nothing to set up: the lesson sends the field's coordinates with every frame.

A blue square 18 in by 18 in is the robot, and the line out of it is the way the robot faces. A gray line behind it is where it has been since the OpMode started. In a lesson that follows a path you also get an orange line for the piece of path being followed, and a green square for the nearest point on that path at the heading the path wants there.

You'll know it worked when the blue square moves as the robot moves. Turn the robot, and the line out of the square swings around with it.

If it didn't, and the robot draws in a corner with most of the square off the field, nothing is wrong. Pedro puts (0, 0) in a corner and the field runs 0 in to 144 in, so a lesson that never sets a pose starts there.

If it didn't, and there is no green square or orange line, you are in a teleop lesson. There is no path to draw. You get the robot and its trail.

18.4 what you just did

Panels has no separate graph call. It reads the telemetry text, and any line shaped `name: number` becomes a table row and a graph series at once. That is why one `Tracker.publish` call gets you both, and why a name may hold anything except a colon.

The same call also writes the number into the flight log on the robot. Panels is the live view and the log is the recording, and they carry the same names, so a graph you found while driving is a graph you can find again afterwards.

The gap between the blue square and the green one is worth knowing. The green square is the nearest point on the path, not where the robot should have got to by now. So the gap says how far off the line the robot is and how crooked it is sitting. It does not say whether the robot is running late.

18.5 where next

- H095 is the other half of this. It runs the same lessons on a laptop, with no robot at all.

095 how-to: check your work without a robot

19.1 what you will have when this is done

Two ways to try a lesson with no robot in front of you. One runs every test and tells you which lessons are finished. The other runs a single lesson on the laptop and draws the robot moving, so you can see what your code does before it is your turn at the field.

19.2 before you start

- Android Studio opens the project, and you can type a Gradle command into its terminal. Ask a mentor to get a laptop that far.
- L040 is written, or step 2 has no lesson to show you.
- AdvantageScope installed on the same laptop, for step 2 and step 3.

19.3 the steps

19.3.1 step 1: run the tests and read the report

```
./gradlew :TeamCode:test
```

That runs the tests for everything the lessons are built on, and leaves the lesson tests out. It says so while it runs:

```
Skipping org.firstinspires.ftc.teamcode.lessons.*. Run them with :TeamCode:testLessons.
```

You'll know it worked when it ends with `BUILD SUCCESSFUL`. Nothing underneath the lessons is broken, so whatever fails next is yours.

If it didn't, and it says `SDK location not found`, the laptop has no `local.properties` file yet. That file is not kept in git, and Android Studio writes it the first time it opens the project. Open the project once, then run the command again.

Now the tests for your own work:

```
./gradlew :TeamCode:testLessons
```

On a fresh copy, where nobody has written a lesson yet, the report counts 44 tests and 44 failures. Forty-four failures is not a broken copy. Each one is a lesson nobody has written, and the count drops as you write them.

You'll know it worked when the last lines name a report:

```
> There were failing tests. See the report at: file:///.../index.html
```

Open that link in a browser, click the class and then the test, and you get the message the test wrote. The console does not give you that. All the console gives you is this:

```
L120EncoderLocalizerTest > countsStraightAhead FAILED
    java.lang.AssertionError at L120EncoderLocalizerTest.java:61
```

A file and a line, and no word about what went wrong. The report for that same run also holds `robot relative while held expected:<1.0> but was:<5.551115123125783E-17>`, which is L150's starter copied and not yet changed. Every *if it didn't* line in this guide quotes a message out of the report, so every one of them starts here.

19.3.2 step 2: watch one lesson move

A test says where the robot ended up. It does not show you the robot. For that, run one lesson on its own:

```
./gradlew :TeamCode:simRun --args="lessons.L040TankOpMode --left_stick_y=-1
--right_stick_y=-1"
```

The first word inside the quotes is the lesson's class name, with the package it sits in before it. Each option after it holds one gamepad control at one value for the whole run, named the way the gamepad names it. So `--left_stick_y=-1` is the left stick pushed fully forward, and `--a=true` is the A button held down. Ask for `--args="--help"` and it lists every control you can set.

To push the sticks yourself, plug a gamepad into the laptop and add `--pad`:

```
./gradlew :TeamCode:simRun --args="lessons.L040TankOpMode --pad"
```

One gamepad becomes `gamepad1` on its own. With two plugged in, hold Start and press A on the one you want to drive with.

Plug it in before the run or during it, whichever suits. Pull it out and the sticks go back to rest, so the robot stops. Plug it back in and it is yours again.

Use `--pad`, or set the controls yourself, but not both. Together they are an error, and the run names the setting that clashed.

It prints where the flight log goes, then runs:

```
Flight log: <the repository's top folder>
lessons.L040TankOpMode running. Connect AdvantageScope to 127.0.0.1 as NetworkTables 4, and
Ctrl-C to stop.
```

Open AdvantageScope and connect to `127.0.0.1` as NetworkTables 4. The topics arrive under `sim/`. Add a 2D field and give it `sim/Pose`.

You'll know it worked when the robot moves. `sim/wheels/frontLeft` and its three neighbors say what reached each motor, and `sim/vel/forward_ips` says how fast the robot is going.

If it didn't, and the robot sits still with all four wheels reading 0, that lesson is still blank. Watched with the old L2b lesson's blanks open and both sticks forward: the pose never changed for the whole run. Nothing is wrong with the simulator. It ran a lesson that writes nothing to the motors.

If it didn't, and no topics arrive at all, nothing connected. The address is `127.0.0.1`, and the kind is NetworkTables 4 rather than a log file.

Ctrl-C stops it. The flight log is in the project's top folder, which the first line it prints names.

Which way around the field gets drawn

The corner the arrow starts in and the way it points do not agree with the numbers going out. Trust that the robot moves, and how far it moves, rather than which way it faces on the screen.

19.3.3 step 3: open the log a failing test left

Some test classes hand you their flight log when they fail. Watched, by asking the simulator's own test for 400 in when the robot drives 54.75 in:

```
java.lang.AssertionError: drove forward, and got a fair way: 54.75097300967889
AdvantageScope can open what ran:
  /var/folders/.../Tank-20260928-065400.wpilog (37502 bytes)
```

You'll know it worked when that file opens in AdvantageScope as a log, not as a connection, and holds the run up to the moment the test gave up. There is more in it than `simRun` sends out: `/Robot/Pose`, `/Robot/Speeds`, `/Robot/Twist`, all four wheels, and the loop timing.

If it didn't, and the failure names no log, that class never asked for one. The lesson tests are all like that today. So `l2b_theSticksMoveTheSimulatedRobot` fails with `drove forward, and got a fair way: 0.0` and no path to open. A log does get written, and it holds 0 bytes, because nothing closed it. Watched. Use step 2 instead, and tell a mentor.

19.4 what you just did

Both halves run the same model of the robot, and it is a small one. Each loop, the four wheel powers are mixed back into forward, sideways and turn. Each of those three chases what was asked of it with a 0.15 s lag. The answer is added up into a pose.

Axis	Flat out
Forward	64.4 in/s
Sideways	43.6 in/s
Turn	4.0 rad/s

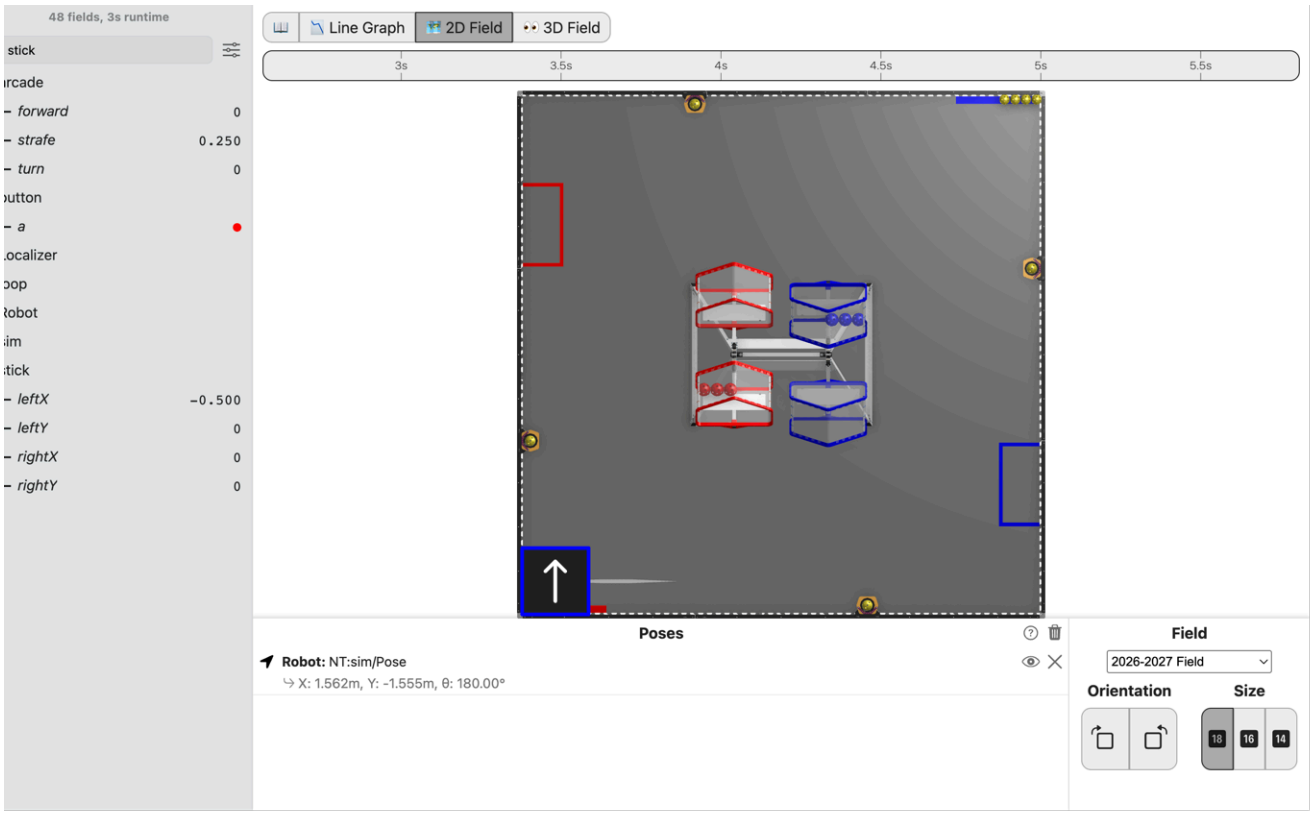
So the model has no slip, no scrub, no sagging battery, no floor, no field wall, no motor wired backwards and no Pinpoint. It will drive straight through the perimeter and never mention it, and it cannot tell you a path is too fast for the wheels. When the simulator and the robot disagree, the robot is right.

That is why a green test run is not a finished lesson. These two catch the mistakes that live in the arithmetic, which are most of them. What is left over is the kind of mistake only a robot on a floor will show you, and that is why every lesson's last step is a robot on a floor.

19.5 where next

- L110 hands the wheels to the path follower. From here on you can try each lesson on the laptop before you take it to the field.

110 lesson: hand the wheels to the path follower



The robot drives on the sticks as it did in L090, and the path follower is holding the drivetrain. The sticks still decide where the robot goes. The path follower is Pedro Pathing's code that drives the robot along a path; from L125 on, it drives the robot itself.

Before you start.

- L090 is done, so `mytry` holds `L090MecanumOpMode`.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- The robot, the Driver Station, and about 2 m by 2 m of clear floor for the last step.

A follower holding the drivetrain knows where the robot is. It can hold a heading, and it can drive a path. Up to L090 the follower was handed a stand-in that ignored it, so the lesson had the motors to itself. L120 onwards need the follower to hold the real one.

20.1 the steps

20.1.1 L110S010: copy the two starters into mytry

This lesson has two files. One is the OpMode. The other is the drivetrain: the code that turns three numbers into four wheel powers.

Copy. In `lessons`, click `L110FollowerDriveTrain` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**. Do the same for `L110FollowerOpMode`.

Check.

- `mytry` holds `L110FollowerDriveTrain` and `L110FollowerOpMode`.

20.1.2 L110S020: switch your copy on

Change. In `L110FollowerOpMode`, delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

20.1.3 L110S030: write the mix

Open `L110FollowerDriveTrain`. It `extends CorbelsDriveTrain`, which means it gets everything `CorbelsDriveTrain` already has. That class sets each motor's direction, scales the powers as L090 did, and sends them to the motors. The one part it leaves to you is `mix`.

`wheels` is an **array**: four numbers in a row, in one variable. `wheels[FL]` is the front left wheel's slot, and `FR`, `BL` and `BR` name the other three.

Change. In `mix`, find the comment that names this step. Under it, type L090's four lines, each into its own slot:

```
        // When on L110S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l110.html#l110s030
        // for what to do here.
        wheels[FL] = forward - strafe - turn;
        wheels[FR] = forward + strafe + turn;
        wheels[BL] = forward + strafe - turn;
        wheels[BR] = forward - strafe + turn;
```

Check.

- No line has a red underline, and `FL`, `FR`, `BL` and `BR` turn purple.

20.1.4 L110S040: make room for the drivetrain

Change. Open `L110FollowerOpMode`. Near the top of the class, find the comment that names this step. Under it, type:

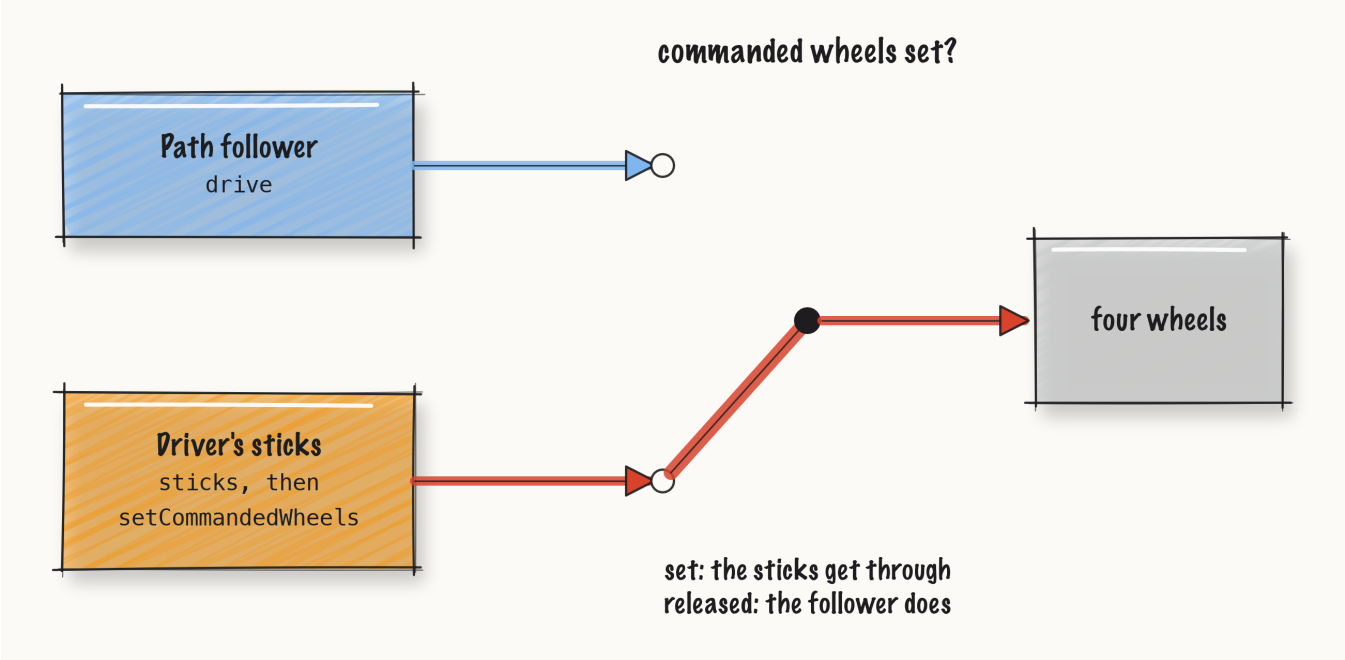
```
    // When on L110S040, see
    // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l110.html#l110s040
    // for what to do here.
    L110FollowerDriveTrain drivetrain;
```

This is a field, like `previousButtonA` in L070. It holds the drivetrain from `init` until the OpMode stops.

Check.

- The line has no red underline.

20.1.5 L110S050: hand the drivetrain to the follower



Change. In `init()`, find the comment that names this step. Under it, type the first line below. Then change `initAfter();` to `initAfter(drivetrain);`:

```
        // When on L110S050, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l110.html#l110s050
        // for what to do here.
        drivetrain = new L110FollowerDriveTrain(hardware);

        initAfter(drivetrain);
```

`new` makes the drivetrain. `initAfter` makes the path follower, and the follower holds what it is handed. Up to L090, `initAfter()` was handed nothing, because the lesson drove its own wheels.

The four `setDirection` lines from L050 are gone from `init`. The drivetrain sets each motor's direction itself, from the same config.

Check.

- Neither line has a red underline.

20.1.6 L110S060: drive with the sticks

Change. In `driveFromTheSticks()`, find the comment that names this step, under the `strafe` lines. Under it, type:

```
        // When on L110S060, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l110.html#l110s060
        // for what to do here.
        drivetrain.sticks(forward, strafe, turn);
```

`sticks` sends the three numbers through your `mix`. Then it scales them and saves them. At the end of the loop, `loopAfter()` updates the follower, and the follower drives the wheels.

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L110FollowerOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `arcade` folder.

Check.

- Push the left stick halfway left. `strafe` reads 0.25, and the robot slides to its own left with its nose pointing the same way.
- Forward, turning and the corners all work as they did in L090.

20.1.7 L110S070: drive it on the robot

Robot. Stop the simulator. Send the code, and run `L110 Follower` from the Driver Station.

Check.

- It drives the way L090 did: forward, sideways, turning, and the diagonals.

No gamepad. `--args="mytry.L110FollowerOpMode --left_stick_x=-0.5"` slides it left for the picture at the top. The simulator has no walls, so a robot left running slides off the field.

If it didn't, and the robot does not move at all, `initAfter` is still `initAfter()`. The sticks command the drivetrain, but nothing drives it, because the follower is holding a stand-in. `LessonsTest.l110_theFollowerDrivesTheSticks` fails with `the follower slides it left: front left expected:<-0.25> but was:<0.0>`.

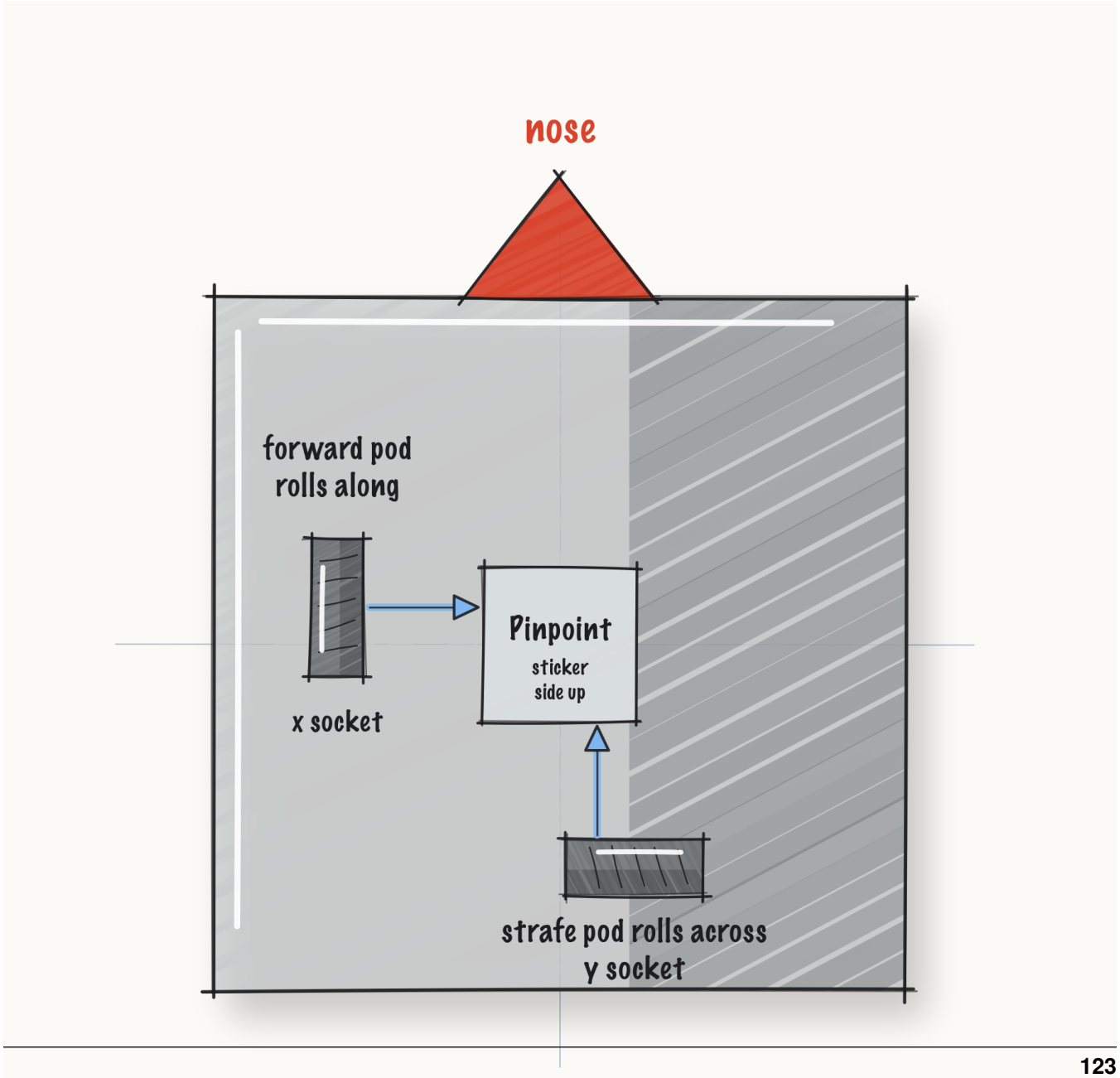
If it didn't, and it slides the wrong way or turns when it should slide, a sign in `mix` is wrong. `LessonsTest.l110_mixGivesEachWheelItsOwnSum` names the wheel and the move, such as `strafe left: back left`.

What you just did. The drivetrain asks one question every time the follower updates it: has the lesson commanded the wheels? If so, it uses what the lesson commanded. If not, it mixes what the follower asked for. Either way the three numbers go through the same `mix`, so the sticks and the follower agree about which wheel does what.

That door is ours, not Pedro's. Pedro's own mecanum drivetrain does only what the follower says. Ours lets the driver reach past it to the wheels, which the teleop lessons after L120 need. The rest of `CorbelsDriveTrain`, which answers everything else Pedro's follower asks a drivetrain, is written out with a comment above each method.

Where next. L115 mounts the Pinpoint, which the follower uses to know where the robot is.

115 lesson: mount the Pinpoint and plug it in



The Pinpoint is bolted on, its two pods are in the right sockets, and the Control Hub finds it when you press INIT. The robot does not yet know how far the pods sit from its middle. L117 measures that.

Before you start.

- L110 is done, so the follower holds the drivetrain. The follower is what builds the Pinpoint.
- The robot, the two odometry pods, and whatever driver the robot's own screws take.
- The I2C cable the Pinpoint came with, and the Driver Station.

The three rules in L115S010 and L115S020 are Pedro's, from its Pinpoint page.

21.1 the steps

21.1.1 L115S010: put one pod along the robot and one across it

The Pinpoint counts two wheels that nothing drives. One rolls along the length of the robot, and that one is the **forward** pod. The other rolls across it, and that one is the **strafe** pod.

Both have to touch the floor the whole time, and neither may be able to lift. A pod that skips loses count, and nothing after it can tell that it did.

Mount. Bolt the two pods on, one along the robot and one across it. Plug the forward pod into the Pinpoint's **x** socket and the strafe pod into its **y** socket.

Check.

- Push the robot forward by hand. The forward pod's wheel turns.
- The strafe pod's wheel drags sideways.

21.1.2 L115S020: plug it into an I2C port that is not port 0

The Pinpoint talks to the Control Hub over I2C. Port 0 already has the Control Hub's own IMU on it, so the Pinpoint goes on another port.

Mount. Bolt the Pinpoint on with its sticker side, the side the ports are on, facing **up**. Plug its cable into I2C port 1, 2 or 3 on the Control Hub.

Configure. On the Driver Station, open the robot's configuration. Add the Pinpoint on that port, and name it `pinpoint`, spelled exactly that way. That is the name the code asks for:

```
c.name.set("pinpoint");
```

Check.

- Pick `L110 Follower` from the `Lessons` group and press INIT.
- INIT finishes with no error.

INIT looks for the Pinpoint by its name, so if INIT finishes, it was found.

21.1.3 L115S030: hold the robot still while it inits

Pressing INIT builds the follower, and the follower builds the Pinpoint. The Pinpoint then resets its own gyro, which takes about a quarter of a second. The robot has to sit still for it.

Robot. Put the robot on the floor, step back, safety glasses on! Pick `L110 Follower` on the Driver Station and

press INIT, but not PLAY yet. Watch it in AdvantageScope.

Check.

- Before you press PLAY, `Localizer/pinPoint/heading_deg` holds steady while the robot sits still.

Drive. Press PLAY and drive a slow square.

Check.

- The heading comes back to where it started.

If it didn't, and a pod wheel does not turn at all in L115S010, it is not reaching the floor. Most mounts have a slot for that.

If it didn't, and the Driver Station shows this at INIT, the name in the configuration is not the name in the code:

```
Unable to find a hardware device with name "pinpoint" and type GoBildaPinpointDriver
```

A still robot cannot show whether the numbers are live; the square shows that.

If it didn't, and the heading crawls one way while the robot sits still, INIT happened while the robot was moving. Press STOP and INIT again without touching it.

If it didn't, and forward and sideways come out swapped once L117 has tuned the pods, the two pods are in each other's sockets. L115S010 is where that gets fixed, and no amount of tuning works around it.

Where the rest comes from. The missing-device message is the one `HardwareMap.get` throws in the SDK. The reset at INIT is `PinpointConfig`'s default `resetMode`, which `PinpointLocalizer`'s constructor acts on. The still robot and the quarter second are from the goBILDA driver's own javadoc.

What you just did. Every lesson from L120 on believes the Pinpoint. L120 puts a second way of tracking the robot beside it and shows the drive encoders drifting away from it over a run. L195 measures the robot against it. A pod in the wrong socket makes all of that quietly wrong instead of loudly broken.

The Pinpoint needs nothing from the drive motors to know where the robot is. Its two wheels are not driven, so a mecanum roller cannot scrub them sideways and a slipping wheel cannot spin them.

Where next. L117 lets AutoTune tune the robot.

Chapter 22

117 lesson: let AutoTune tune the robot

The robot measures itself. It finds which way each motor turns and where the pods sit. It finds how fast it can drive and how hard it can brake. Pedro Pathing's AutoTune does the measuring. You paste what it finds into `Constants.java`.

Before you start.

- L115 is done, so the Pinpoint and its two pods are mounted and plugged in, and L050's motor directions are set.
- The robot, the Driver Station, a laptop that can join the robot's Wi-Fi, and blocks to lift the robot's wheels off the floor.
- Clear floor for L117S040: at least 48 in ahead of the robot, 48 in to its left and 36 in to its right, plus a tile of room all around.

The step names and the messages quoted are from the tuner code in `pedro/procedures/`, and the address from Pedro's Mecanum page.

22.1 the steps

22.1.1 L117S010: open AutoTune

AutoTune is a web page the robot serves. It is not in the Driver Station's list of OpModes. It starts its own OpMode, called `Pedro Tuning`, when a step needs the robot to move.

Send. Send the code to the robot. Join the laptop to the robot's Wi-Fi. In a browser, open `http://192.168.43.1:10158`.

The first time the browser talks to the robot, the operating system might ask for your permission. Please give it.

Check.

- The page lists four tuners: `Mecanum Tuner`, `Tests`, `Pinpoint Tuner` and `Foresight Tuner`.

22.1.2 L117S020: find which way each motor turns

Put the robot on blocks, so its wheels spin freely.

Robot. Pick `Mecanum Tuner`. Type the four motor names as the Robot Configuration spells them: `Front Left`, `Front Right`, `Back Left` and `Back Right`. Each motor then spins on its own, at half power, beside a picture of which wheel it is. Say whether it spun, and whether it spun forward or reversed.

Change. At the end, AutoTune shows Java that starts `public static MecanumConfig drivetrainConfig`. Open `Constants`, in the `pedro` folder. Find the four lines that set the names, and put the four name lines AutoTune shows in their place. Keep `manualBrakeMode` and `powerThreshold`. AutoTune does not print them. Then open the file of the robot `thisRobot` names, such as `Bot37523`, and put the four direction lines AutoTune shows in place of its four.

Check.

- The directions match what L050 found on blocks.

22.1.3 L117S030: measure the pods

The Pinpoint needs four more numbers: which way each pod counts, and how far each one sits from the middle of the robot. The x offset is how far sideways the forward pod is from the point the robot turns about, and left counts up. The y offset is how far forward the strafe pod is, and forward counts up. This robot's are about 1.06 in and 3.36 in.

Put the robot on the floor.

Robot. Pick `Pinpoint Tuner`. Keep the name `pinpoint` and the pod type it offers. Then do the three pushes it asks for, stopping each one when you are done:

- Push the robot forward.
- Push the robot to the left.
- Spin the robot half a turn on the spot, counterclockwise.

Change. Replace `localizerConfig` in `Constants` with the `PinpointConfig` AutoTune shows.

Check.

- In `Tests`, pick `Localization Test` and drive with the gamepad.
- Driving forward raises x, and strafing left raises y.

22.1.4 L117S040: let the robot find its speed and its brakes

This one drives itself. Clear the floor first.

Robot. Pick `Foresight Tuner`. Keep the distances and the speed it offers: 48 in, 30 in/s and 36 in. It drives forward and left as fast as it can, coasts to a stop, turns back and forth, spins, brakes, and steps 12 in to 24 in each way.

Change. Replace `foresightConfig` in `Constants` with the `ForesightConfig` AutoTune shows. Keep the five lines it does not print, from `translationalConstraint` to `holdPointHeadingScaling`.

Three of its numbers are kept twice in `Constants`, for the lessons' own drivetrain. Copy each one up into its own line:

- the number in `c.coast.set(...)` goes in `powerPerInchPerSecond`;
- `maxAchievableForwardVelocity` goes in `maxForwardInchesPerSecond`;
- `maxAchievableStrafeVelocity` goes in `maxStrafeInchesPerSecond`.

Check.

- Each of the three pairs holds the same number.

22.1.5 L117S050: watch it drive a line

Send. Send the code again, so the new `Constants` is on the robot.

Robot. In `Tests`, pick `Line Test`, with the distance at 48.

Check.

- The robot drives 48 in and stops on the line.

If it didn't, and the Mecanum Tuner stops with this, a motor other than the one named spun, or none did:

```
The wrong motor spun. Check that your motors are plugged into the correct ports, and that
they are configured correctly. Then, try again.
```

It is the message for an answer of No to "Did the front left motor spin?", read from `MecanumTuner.java`. Swap two motor names and each of those motors is given the other's direction, so both wheels turn backwards. L050's checks on blocks cannot tell that from a backwards wheel. This one can.

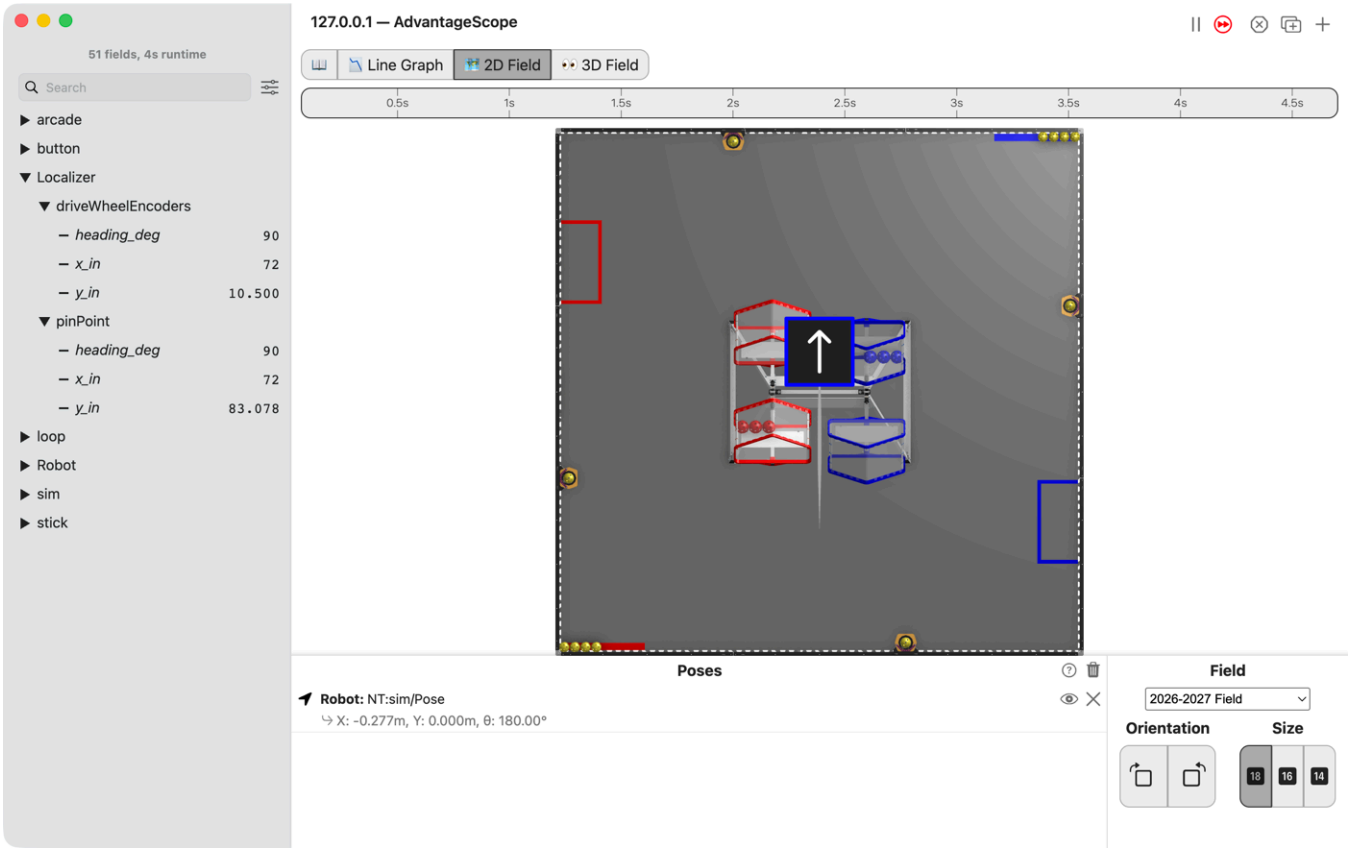
If it didn't, and x and y swap over in the Localization Test, the two pods are in each other's sockets. L115S010 is where that gets fixed, and no tuning works around it.

If it didn't, and the robot brakes late or never holds still after L117S040, check that the five hold lines are still in `foresightConfig`.

What you just did. Every number in `Constants.java` and the robot's own file either came off this robot or is waiting to. AutoTune measures the ones that need the robot driving itself. L195 measures the two it does not: ticks per inch and the turn radius.

Where next. L120 compares the Pinpoint with the drive encoders and logs both.

120 lesson: work out where the robot is from the drive wheels



Your own localizer runs beside the robot's own, and both are published so they can be graphed against each other. Yours counts how far the four drive wheels roll. The robot's own counts the Pinpoint's two pods. Neither is the truth. A tape measure is.

Before you start.

- L110 is done, so the follower holds a real drivetrain, and L115 is done, so the Pinpoint is on the robot.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- The robot on blocks, as in L050, the Driver Station, a tape measure, and about 3 m of clear floor for the last

step.

The simulator turns each wheel's encoder by how far the wheel rolls, at `Constants.ticksPerInch`, and turns the IMU with the robot. It has no slip, so a finished localizer follows the Pinpoint there more closely than it will on the robot. The tests check the arithmetic step by step.

23.1 the steps

23.1.1 L120S010: copy the two starters into mytry

This lesson has two files. One is the OpMode. The other is the **localizer**: the code that works out where the robot is.

Copy. In `lessons`, click `L120CompareLocalizersOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**. Do the same for `L120EncoderLocalizer`.

Check.

- `mytry` holds `L120CompareLocalizersOpMode` and `L120EncoderLocalizer`.

23.1.2 L120S020: switch your copy on

Change. In `L120CompareLocalizersOpMode`, delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

23.1.3 L120S030: run your localizer beside the robot's own

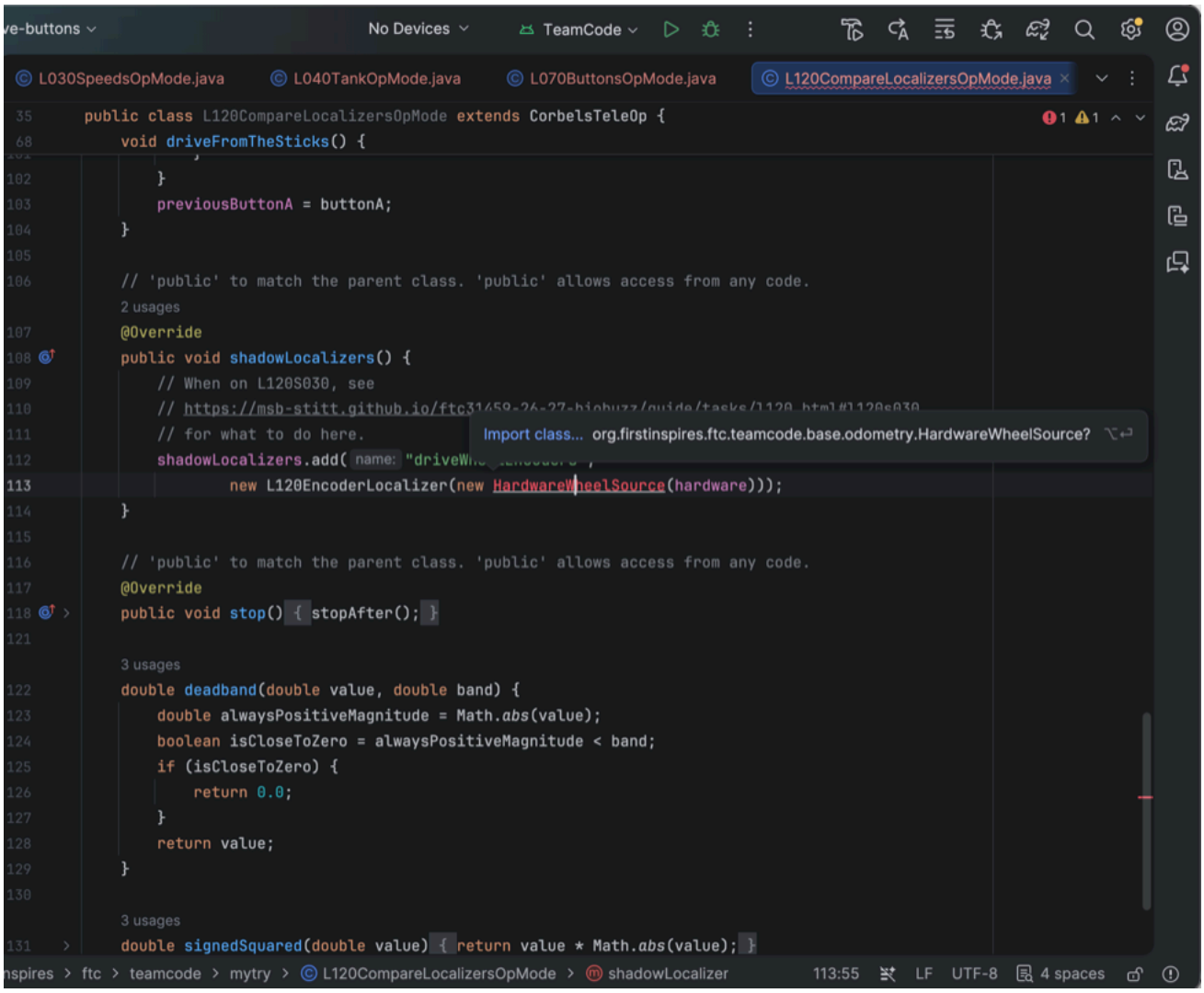
The robot's own localizer reads the Pinpoint. A **shadow** localizer runs beside it. It is updated and published every loop, but it never steers the robot.

Change. In `L120CompareLocalizersOpMode`, find `shadowLocalizers()` and the comment that names this step. Under it, type:

```
        // When on L120S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l120.html#l120s030
        // for what to do here.
        shadowLocalizers.add("driveWheelEncoders",
                new L120EncoderLocalizer(new HardwareWheelSource(hardware)));
```

`HardwareWheelSource` turns red, because this file does not know where it lives yet.

- Click `HardwareWheelSource`.
- Press Option+Enter on a Mac, Alt+Enter on Windows, and pick **Import class…**.
- Android Studio adds an `import` line at the top of the file, and the red goes away.



Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L120CompareLocalizersOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020. Open the `Localizer` folder, then `driveWheelEncoders` and `pinPoint` inside it.

Check.

- Both start at `x_in` 72, `y_in` 10.5 and `heading_deg` 90.
- Push the left stick forward. `pinPoint/y_in` climbs, and `driveWheelEncoders` stays where it started. Yours does not count anything yet.

`x_in` is how many inches the robot is from the left edge of the 2D Field, and `y_in` how many from its bottom edge. `heading_deg` is the way the robot's nose points, and 90 is straight up the field. Since L040, the **Poses** list has called that same direction 180°. Both mean up the field.

23.1.4 L120S040: add up how far the wheels rolled

`update` runs once a loop. Each time, it can only see how far each wheel has rolled in total, so it remembers last loop's numbers and takes them away.

The first loop has nothing to take away from, so it only remembers.

Change. Press Ctrl+C in the Terminal to stop the simulator. Open `L120EncoderLocalizer`. In `update`, find the comment that names this step. Under it, type:

```
        // When on L120S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l120.html#l120s040
        // for what to do here.
        if (lastWheels == null) {
            lastWheels = wheels.clone();
            lastRawHeading = rawHeading;
            return;
        }

        double frontLeft = wheels[0] - lastWheels[0];
        double frontRight = wheels[1] - lastWheels[1];
        double backLeft = wheels[2] - lastWheels[2];
        double backRight = wheels[3] - lastWheels[3];

        double forward = (frontLeft + frontRight + backLeft + backRight) / 4.0;
        double strafe = (-frontLeft + frontRight + backLeft - backRight) / 4.0;

        x += forward;
        y += strafe;
```

Android Studio types the `}` for you when you press Enter after the `{`. Type the rest below it.

`null` means "nothing yet". `.clone()` makes a copy, so next loop's numbers do not change it. `return` leaves `update` early, until next loop.

The two long lines are L090's table read backwards. There, three numbers made four wheel powers. Here, four wheel moves make `forward` and `strafe` again, in inches.

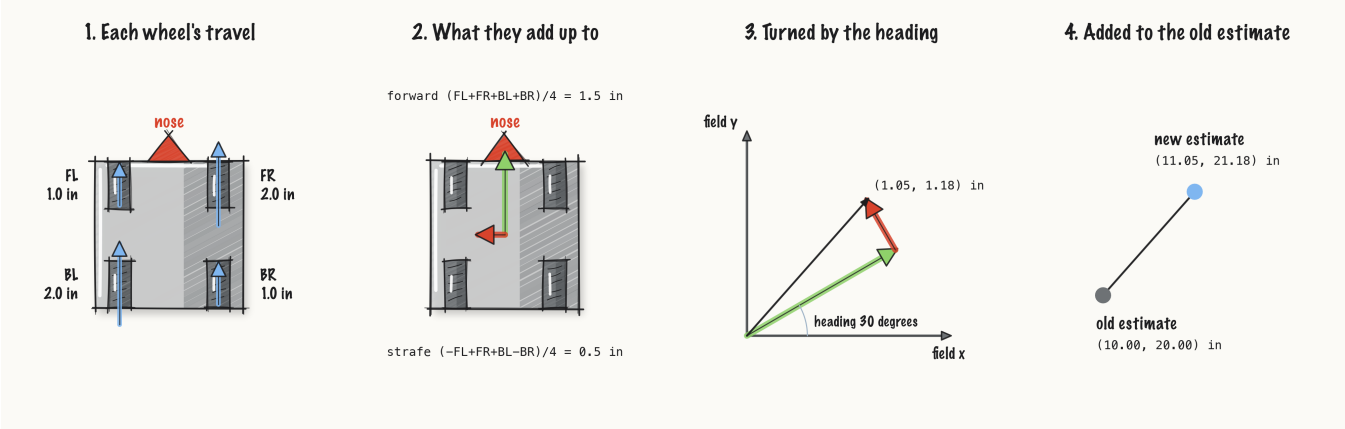
Check.

- In the Terminal, run this. It prints `8 tests completed, 2 failed`, then `BUILD FAILED` and more red lines. That is what failing tests look like here, and two are meant to fail now.

```
./gradlew :TeamCode:testDebugUnitTest --tests '*L120EncoderLocalizerTest*'
```

The two that fail are `appliesTheHeadingSoSidewaysIsRelativeToTheRobot` and `setPoseMovesTheEstimateWithoutMovingTheRobot`. Both are about which way the robot faces. That is the next step.

23.1.5 L120S050: turn the step to face the way the robot faces



`forward` is forward for the robot, not for the field. The robot starts facing along the field's y, so its forward is the field's y. L120S040 added it to x.

Change. Delete the two lines `x += forward;` and `y += strafe;`. Find the comment that names this step. Under it, type:

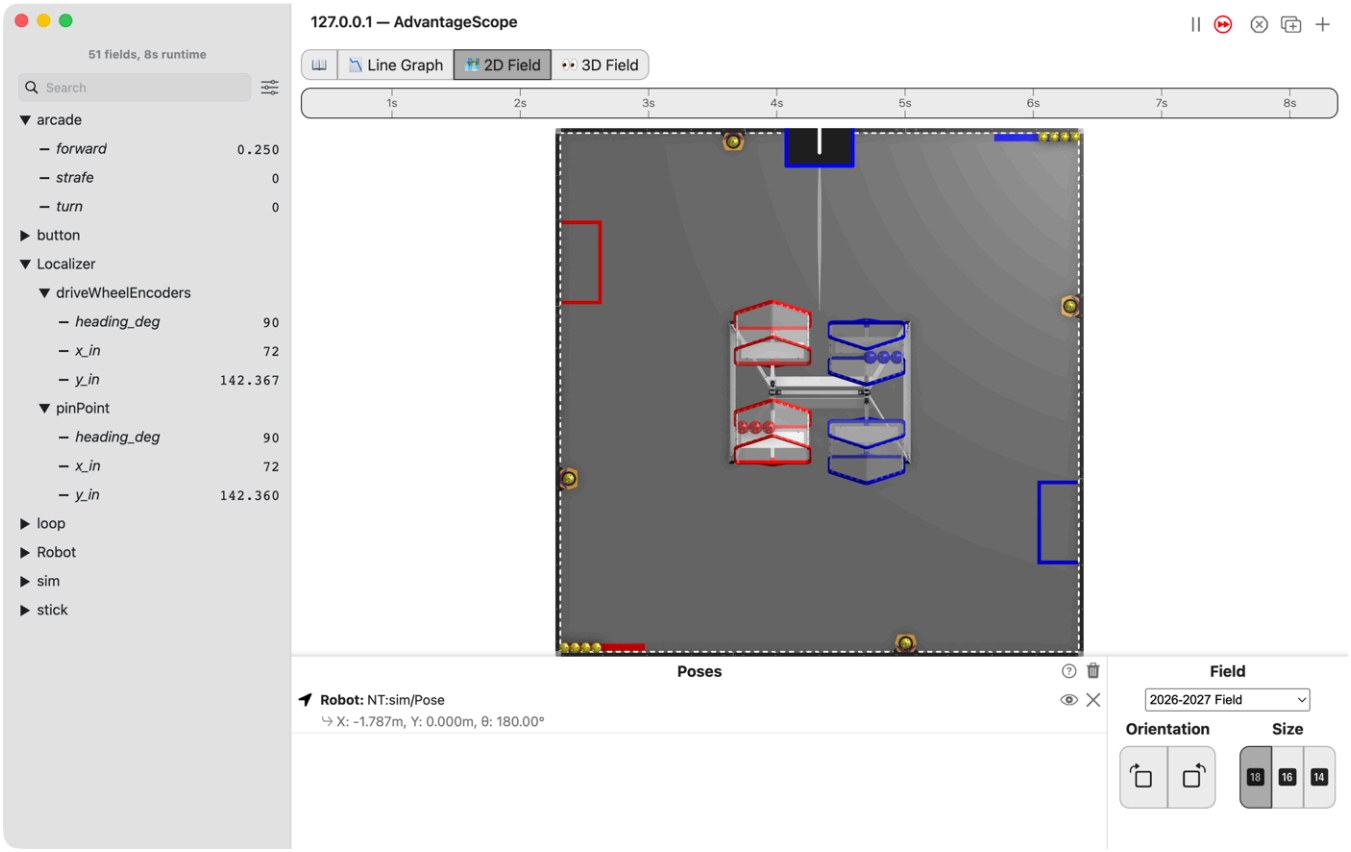
```
        // When on L120S050, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l120.html#l120s050
        // for what to do here.
        double heading = rawHeading + headingOffset;
        x += forward * Math.cos(heading) - strafe * Math.sin(heading);
        y += forward * Math.sin(heading) + strafe * Math.cos(heading);
```

`Math.cos` and `Math.sin` share the step out between x and y. Facing along x, `cos` is 1.0 and `sin` is 0.0, so forward all goes on x. Facing along y, `cos` is 0.0 and `sin` is 1.0, so forward all goes on y. In between, each gets a part.

`heading` counts in **radians**, not degrees. One whole turn is about 6.28 radians. `Math.cos` and `Math.sin` use radians.

Check.

- Run the same test command again. It ends with `BUILD SUCCESSFUL`.



Simulator. Run the command from L120S030 again, and connect AdvantageScope.

Check.

- Push the left stick forward. This time `driveWheelEncoders/y_in` climbs with `pinPoint/y_in`, and the two stay within a tenth of an inch.

The simulator turns each wheel's encoder by how far it rolls, so your localizer has something to count.

23.1.6 L120S060: drive it, then measure it

Robot. Stop the simulator. Send the code, run `L120 Compare Localizers` from the Driver Station, and watch it in AdvantageScope.

Check.

- Drive about 80 in straight ahead and stop. Both `y_in` numbers moved, and they do not quite agree.
- Measure how far the robot really went with the tape measure.

Then put the robot on the blocks and push the left stick forward. The wheels spin, and the robot does not go anywhere. `driveWheelEncoders` walks off across the field. `pinPoint` stays put, because its pods are off the floor too.

No gamepad. `--args="mytry.L120CompareLocalizersOpMode --left_stick_y=-0.5"` drives forward for the picture at the top, and for L120S050's.

If it didn't, and `driveWheelEncoders` is missing from the `Localizer` folder in L120S030, the two lines are not

in `shadowLocalizers()`. `LessonsTest.l120_theEncoderLocalizerRunsAlongsideAndIsLogged` fails with `the shadow localizer is logged:` and the list of names that were published.

If it didn't, and `countsStrafingLeft` fails with `left is +y when facing +x`, a sign in the `strafe` line is wrong. With the sign in front of `frontRight` flipped, it reads `expected:<12.0> but was:<6.0>`, and three other tests fail with it.

If it didn't, and all 8 tests fail with `Cannot load from double array because "this.lastWheels" is null`, the first-loop `if` is missing.

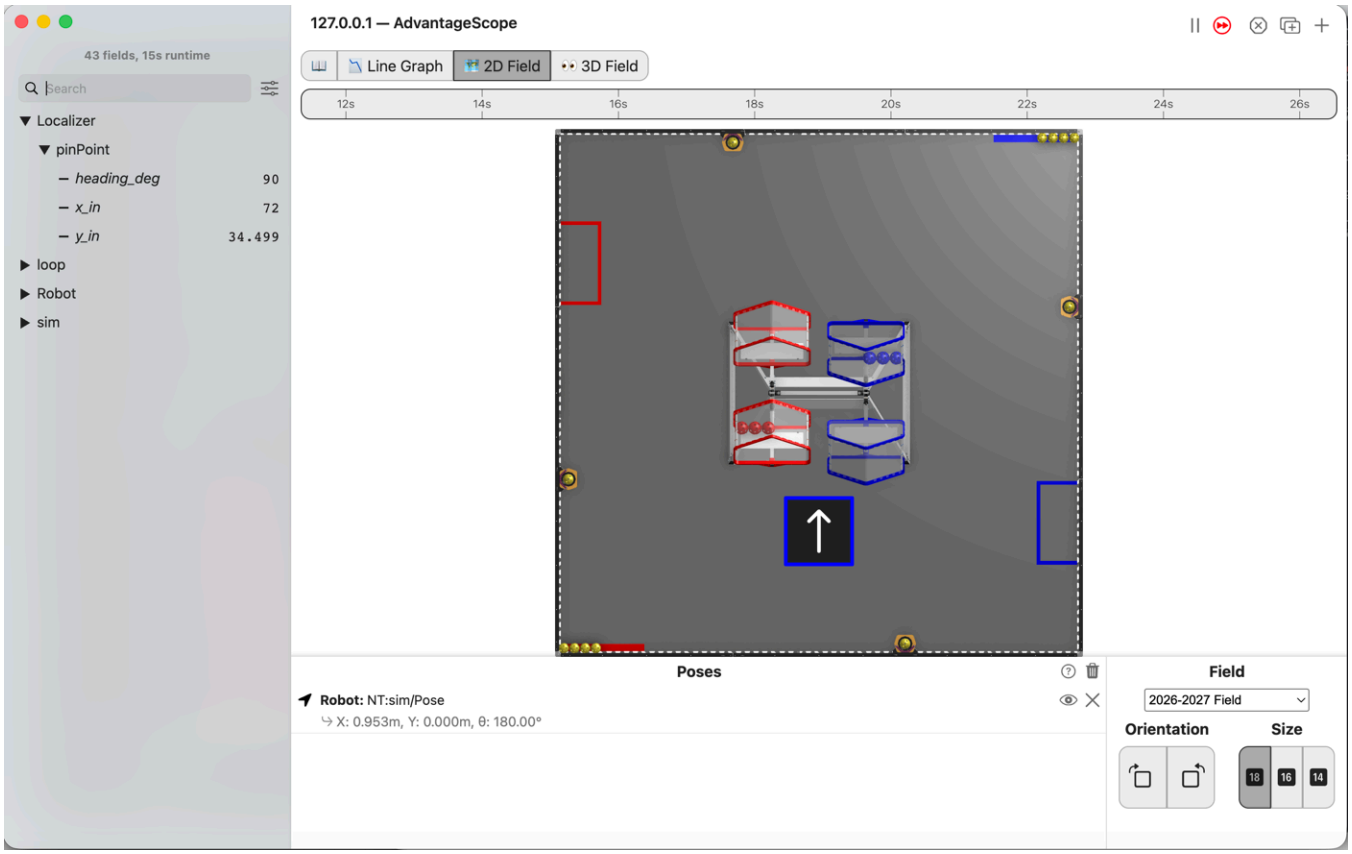
What you just did. That is **dead reckoning**: believe the wheels, and add up every step. It has no other source, so a wheel that spins without carrying the robot moves the estimate. The test `aWheelThatSlipsIsSimplyBelieved` shows it on a laptop: one wheel rolling 24 in on its own moves the estimate 6 in. The Pinpoint's pods only measure and never push, so they slip less. That is the case for having them.

The heading here is the one at the end of each step. While driving and turning at once, the middle of the step would be closer. Taking the end needs no subtraction of two headings, so a turn through the back of the circle needs no care.

The localizer reports a speed of zero. Only `x_in`, `y_in` and `heading_deg` are published for a shadow, so nothing reads its speed.

Where next. L125 drives 24 inches on its own.

125 lesson: the first autonomous



The robot drives 24 inches forward on its own and stops there. Nobody touches a stick. The path follower does the driving, and you write one line to tell it where to go.

Before you start.

- L120 is done. Nothing here uses L120's code; the follower uses the robot's own localizer.
- AdvantageScope is installed on the same laptop. No gamepad is needed.
- The robot, the Driver Station, a piece of tape, a tape measure, and about 1 m of clear floor in front of the robot for the last step.

24.1 the steps

24.1.1 L125S010: copy the starter into mytry

Copy. In `lessons`, click `L125Drive24OpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L125Drive24OpMode`, and the new file is open.

24.1.2 L125S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

24.1.3 L125S030: say where to go

An autonomous has no sticks to read. It has `driveForward24()`, which hands back a **command**: a list of things to do, in order. `start()` schedules that command, and the robot runs it after PLAY.

`start` and `end` are already near the top of the class, 24 inches apart along the field's y. The robot starts facing along y, so the line from one to the other is straight ahead. `startPose()` tells the follower where the robot is at the start.

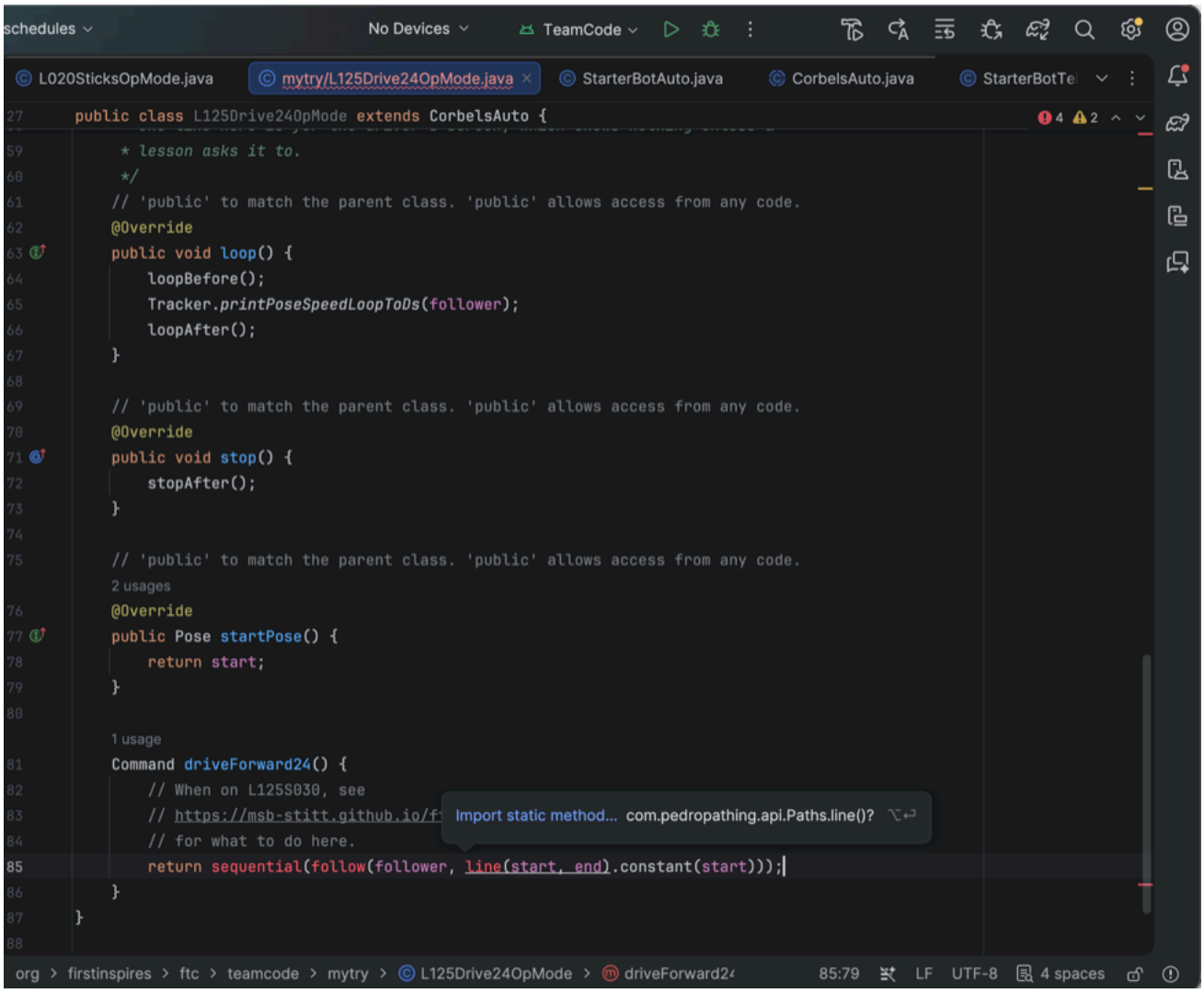
Change. In `driveForward24()`, find the comment that names this step. Delete the line `return Command.NOOP;` under it, and type:

```
        // When on L125S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l125.html#l125s030
        // for what to do here.
        return sequential(follow(follower, line(start, end).constant(start)));
```

Read it from the inside out. `line(start, end)` is the path. `.constant(start)` holds the heading `start` has the whole way. `follow` drives the path. `sequential` makes the list, and this list has one thing in it.

`sequential`, `follow` and `line` turn red. For each one:

- Click it.
- Press Option+Enter on a Mac, Alt+Enter on Windows. This time the popup says **Import static method…**, because each is a method, not a class. Pick it.
- Android Studio adds an `import static` line at the top of the file.



Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L125Drive24OpMode"
```

AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `Localizer` folder, then `pinPoint`.

Check.

- The robot has driven up the field and stopped.
- `y_in` reads 34.5, which is 24 more than the 10.5 it started at.
- `x_in` stays at 72 and `heading_deg` at 90.

24.1.4 L125S040: drive it on the floor

Robot. Stop the simulator. Put the robot on the floor with a piece of tape at its front edge. Send the code, and run `L125 Drive 24` from the Driver Station. It is in the autonomous list, not the TeleOp one. Keep your hands off

the sticks.

Check.

- The robot drives forward, slows down, and stops, all by itself!
- Measure from the tape to the front edge. It is close to 24 inches.

If it didn't, and nothing moves, `driveForward24()` still says `return Command.NOOP;`, the command that does nothing. `LessonsTest.l125_autoDrives24InchesForwardAndStops` fails with `the robot never started following a path`.

If it didn't, and nothing moves with the line typed, `start` and `end` are swapped. The robot is standing at that path's finish already, so it is done before it starts. The test fails with `ends 24 inches further along y expected:<34.5> but was:<10.5>`.

If it didn't, and the test fails with `and does not wander in x expected:<72.0> but was:<70.81415129095723>`, the heading is `.constant(90)`. The number form is radians, so 90 is not 90 degrees.

If it didn't, and on the floor it overshot or stopped short, that is the follower's tuning, not this line. L117 tunes it. If it curved, check the motor directions from L050S070 first.

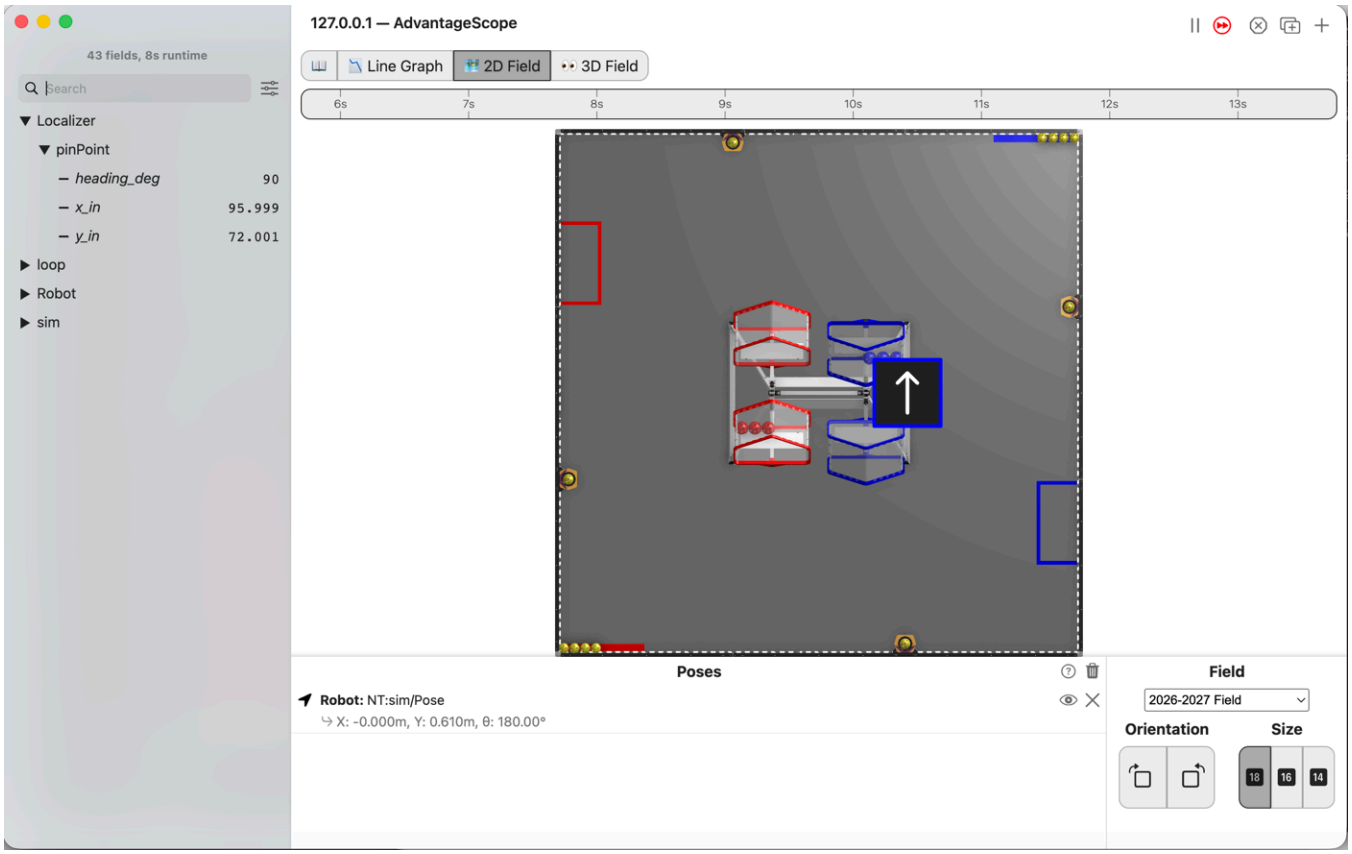
What you just did. The follower is a library, used the way a team uses one: say where to go, and it works out the wheel powers. Nobody here wrote the path code.

`.constant()` holds one heading for the whole line, which is what this path wants. Pedro also has `.linear()`, which turns from one heading to another. Before Pedro 3.0.1 it turned the wrong way on a line path, reported in its issues #176 and #181. This project uses 3.0.1, which fixes it.

`sequential` takes a list, and this one has a single item in it. L130 puts a second item after it, and that is the whole of what an autonomous routine is: a list of things to do in order.

Where next. L130 adds a second leg.

130 lesson: two legs, forward then sideways



The robot drives forward, then slides 24 inches to its right without turning, and holds where it ends up. It is the first autonomous that does more than one thing.

Before you start.

- L125 is done.
- AdvantageScope is installed on the same laptop. No gamepad is needed.
- The robot, the Driver Station, a piece of tape, a tape measure, and about 1 m by 2 m of clear floor for the last step.

25.1 the steps

25.1.1 L130S010: copy the starter into mytry

Copy. In `lessons`, click `L130ForwardThenStrafeOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L130ForwardThenStrafeOpMode`, and the new file is open.

25.1.2 L130S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

25.1.3 L130S030: add a second leg and a hold

A pose is a place on the field and the way the robot faces there. Near the top of the class are three poses: `start`, `corner` and `end`. All three face 90 degrees. `corner` is straight up the field from `start`, and `end` is 24 inches to the right of `corner`.

`driveForwardThenStrafe()` already holds the first leg, from `start` to `corner`. It is L125's line, with one item on each line so there is room for more.

Change. In `driveForwardThenStrafe()`, type a comma at the end of the first leg. The comment that names this step is under it. Under the comment, type:

```
        return sequential(
                follow(follower, line(start, corner).constant(start)),
                // When on L130S030, see
                // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks
/l130.html#l130s030
                // for what to do here.
                follow(follower, line(corner, end).constant(start)),
                hold(follower, end)
        );
```

Each item but the last ends with a comma. The second leg holds `start`'s heading too, so the robot never turns. It slides sideways instead, which is what mecanum wheels are for. `hold` keeps the robot at `end` once it gets there.

`hold` turns red. Click it and press Option+Enter, as in L125.

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L130ForwardThenStrafeOpMode"
```

AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `Localizer` folder, then `pinPoint`.

Check.

- The robot has stopped right of the middle of the field.

- `x_in` reads 96 and `y_in` 72.
- `heading_deg` is still 90.

25.1.4 L130S040: drive it on the floor

Robot. Stop the simulator. The robot drives about 62 in forward, then 24 in to its right, so clear that much floor, plus room for the robot. Put the robot on the floor with a piece of tape at its front edge. Send the code, and run `L130 Forward Then Strafe` from the autonomous list.

Check.

- The robot drives straight forward and stops, then slides to its right and stops again.
- Its nose points the same way the whole time.

If it didn't, and the robot cuts across in one diagonal, the first leg goes to `end` instead of `corner`. `LessonsTest.l130_autoDrivesForwardThenStrafesSideways` fails with `drove straight up to the corner first expected:<0.0> but was:<20.307354640929788>`.

If it didn't, and the test fails with `never turned, so it still faces +y expected:<90.0> but was:<116.04530086607369>`, the second leg is `.constant(90)`. The number form is radians, as in L125.

If it didn't, and on the floor it slid left, check which way +x runs from where the robot stands. Facing +y, the robot's right hand points at +x.

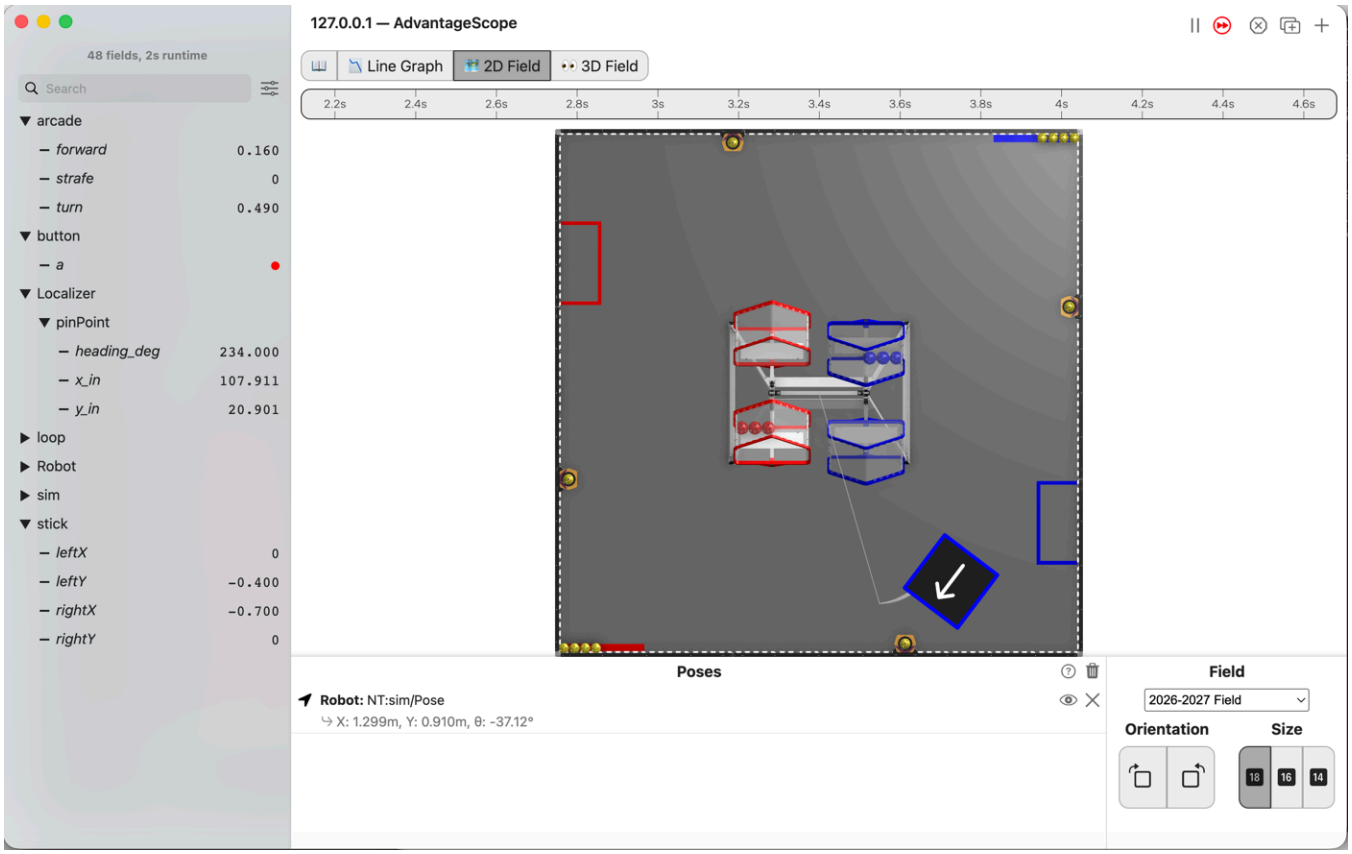
`hold` is not checked: the test passes with it left out. On a floor it keeps the wheels pushing back if something bumps the robot once the path is over.

What you just did. An autonomous routine is a list. Two legs and a hold here, and a match autonomous is the same list with more items, some of them moving something other than the wheels.

A leg holds one heading from one end to the other. Hold the same heading on both legs and the robot never turns. Where it goes and which way it faces are two separate things, and L140 spends that.

Where next. L140 goes back to the sticks, where a stick means a direction on the field.

140 lesson: make the sticks mean the field



Push the stick away from you and the robot goes away from you. It does that whichever way the robot happens to be facing, so a driver never has to think about which end of the robot is the front.

Before you start.

- L130 is done.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- The robot, the Driver Station, and about 2 m by 2 m of clear floor for the last step.

Where the driver stands. "Away from the driver" is the field's +x. The robot starts at heading 90°, facing +y, so at the start a push away from the driver slides it to its right.

26.1 the steps

26.1.1 L140S010: copy the starter into mytry

Copy. In `lessons`, click `L140FieldRelativeOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L140FieldRelativeOpMode`, and the new file is open.

26.1.2 L140S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

26.1.3 L140S030: drive and turn at once, the L110 way

The starter drives the way L110 did. Forward on the stick is forward for the robot.

Simulator. In the Terminal, run:

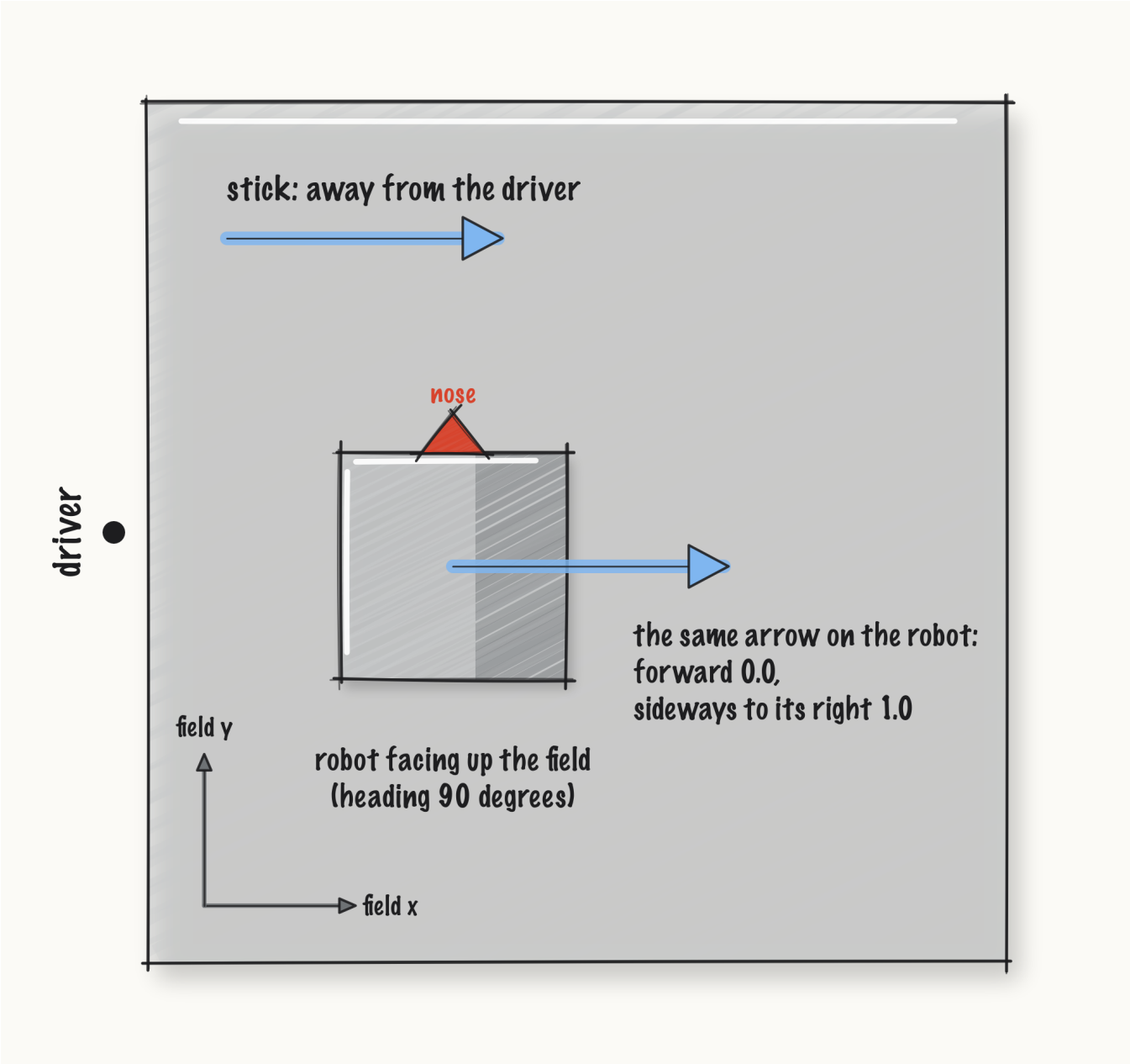
```
./gradlew :TeamCode:simRun --args="mytry.L140FieldRelativeOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `Localizer` folder, then `pinPoint`.

Check.

- Push the left stick a little forward and the right stick left, and hold both.
- The robot drives in a small circle, and `x_in` and `y_in` stay close to where they started.
- It goes where its nose points, and its nose keeps turning.

26.1.4 L140S040: turn the stick backwards through the heading



From now on, `forward` on the stick means away from the driver, along the field's x. `strafe` means to the driver's left, along the field's y. On AdvantageScope's 2D Field, +x runs to the right and +y runs up.

The wheels only know the robot's own forward and sideways. L120 turned a step on the robot into a step on the field, through the heading. This turns the other way, from the field to the robot, so the signs on `sin` swap.

Change. Press Ctrl+C in the Terminal to stop the simulator. In `driveFromTheSticks()`, find the comment that names this step. Delete the line `drivetrain.sticks(forward, strafe, turn);` under it, and type:

```
        // When on L140S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l140.html#l140s040
        // for what to do here.
        double heading = follower.pose().heading();
        double robotForward = forward * Math.cos(heading) + strafe * Math.sin(heading);
        double robotStrafe = -forward * Math.sin(heading) + strafe * Math.cos(heading);
        drivetrain.sticks(robotForward, robotStrafe, turn);
```

The follower knows which way the robot faces. The drivetrain does not. So the OpMode asks the follower, and does the turn itself.

Simulator. Run the same command again, and connect AdvantageScope.

Check.

- Hold the same two sticks. The robot spins, and at the same time it travels to the right on the 2D Field, along +x.
- `x_in` climbs while `heading_deg` goes around and around.

26.1.5 L140S050: drive it facing you

Robot. Stop the simulator. Send the code, and run `L140 Field Relative` from the Driver Station. Stand off the robot's left side, so that away from you is the field's +x.

Check.

- Push the right stick until the robot's nose points at you.
- Then push the left stick away from you. The robot drives away from you while it faces you.
- Turn it a quarter turn and try again, and the stick still means the same way on the field.

No gamepad. `--args="mytry.L140FieldRelativeOpMode --left_stick_y=-0.4 --right_stick_x=-0.7"` holds both sticks, for the circle in L140S030 and the picture at the top. Before L140S040, the run was watched at `x_in` 70.812 and `y_in` 7.230 after about 8 s.

After L140S040, the robot drifts toward +y as it goes: one run went from x 72 in and y 10.5 in to x 179 in and y 43.5 in, about 17 degrees off a straight line. The simulator has no walls, so a run left going drives off the field.

If it didn't, and `LessonsTest.l140_fieldRelativeIgnoresWhichWayTheRobotFaces` fails with `same stick, still moves away from the driver expected:<0.0> but was:<1.0>`, the line still reads `drivetrain.sticks(forward, strafe, turn);`.

If it didn't, and it fails with `expected:<-1.0> but was:<1.0>`, the turn runs the wrong way. The minus sign is missing from `-forward`, or both signs on `sin` are the other way around.

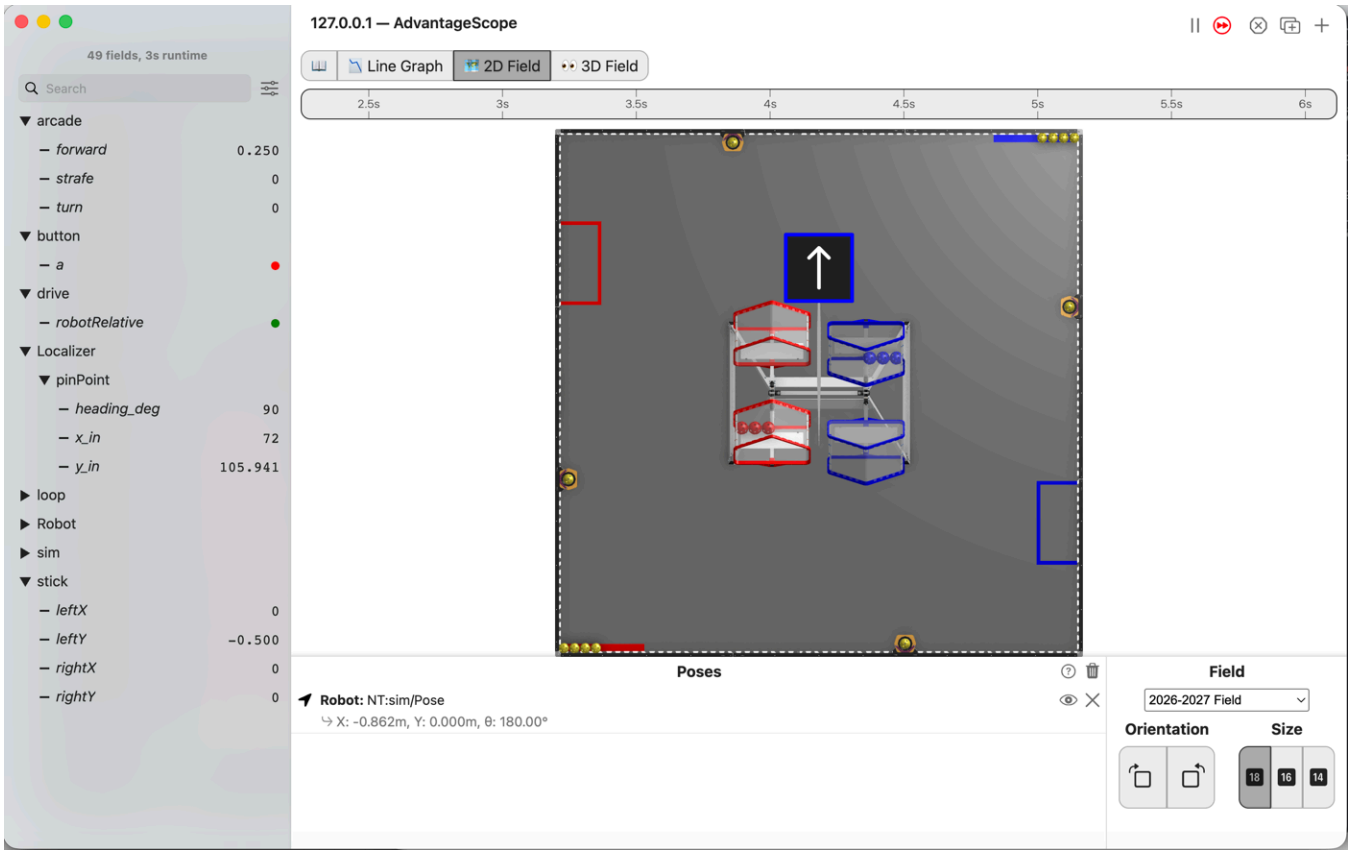
If it didn't, and it fails with `same stick, still moves away from the driver expected:<0.0> but was:<-0.3338674754196965>`, the heading went in as degrees. `Math.cos` wants radians.

What you just did. Field relative is one turn of two numbers and nothing else. The same `sticks`, the same mix, the same scaling. All that changed is which pair of numbers goes in.

It also depends on the follower knowing which way the robot faces. If the heading drifts 20 degrees, the sticks are 20 degrees crooked. This is the first lesson where a mistake somewhere else in the robot shows up in the driver's hands.

Where next. L150 puts the robot's own directions back on a bumper, so a driver can have both.

150 lesson: give the driver both



The sticks mean the field, the way L140 left them, until the right bumper is held. While it is held they mean the robot. Let go and they mean the field again.

Before you start.

- L140 is done.
- AdvantageScope is installed on the same laptop, and a gamepad is plugged into the laptop.
- The robot, the Driver Station, and about 2 m by 2 m of clear floor for the last step.

27.1 the steps

27.1.1 L150S010: copy the starter into mytry

Copy. In `lessons`, click `L150RobotRelativeButtonOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L150RobotRelativeButtonOpMode`, and the new file is open.

27.1.2 L150S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

27.1.3 L150S030: choose with the bumper

Field relative is the easier way to drive most of the time. It is not the easier way to line up against something. Backing straight out of a corner, or nudging along a wall, is easier when forward means the robot's own forward.

So the driver gets both, and an `if` chooses. The starter already works out `robotForward` and `robotStrafe`, L140's turn. `forward` and `strafe` are the sticks as they are, the robot's own directions.

Change. In `driveFromTheSticks()`, find the comment that names this step. Delete the line `drivetrain.sticks(robotForward, robotStrafe, turn);` under it, and type:

```
        double heading = follower.pose().heading();
        double robotForward = forward * Math.cos(heading) + strafe * Math.sin(heading);
        double robotStrafe = -forward * Math.sin(heading) + strafe * Math.cos(heading);
        // When on L150S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l150.html#l150s030
        // for what to do here.
        boolean robotRelative = gamepad1.right_bumper;
        Tracker.publish("drive/robotRelative", robotRelative);
        if (robotRelative) {
            drivetrain.sticks(forward, strafe, turn);
        } else {
            drivetrain.sticks(robotForward, robotStrafe, turn);
        }
```

When you type `{` and press Enter, Android Studio adds the `}` for you.

The bumper is read fresh every loop. Let go, and the next loop drives field relative again. The driver's thumb tells them which way the sticks work, so there is nothing to remember in a match.

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L150RobotRelativeButtonOpMode --pad"
```

AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `drive` folder, then `Localizer` and `pinPoint`.

Check.

- Hold the right bumper and push the left stick a little forward. `robotRelative` turns green, and the robot drives up the field, the way its nose points.
- `y_in` climbs and `x_in` stays at 72.
- Let go of the bumper and keep pushing. The robot slides to the right, along +x, and `x_in` climbs.

27.1.4 L150S040: drive both ways on the floor

Robot. Stop the simulator. Send the code, and run `L150 Robot Relative Button` from the Driver Station. Stand off the robot's left side, as in L140.

Check.

- Turn the robot so its nose points to one side, and push the left stick away from you.
- The robot drives away from you.
- Now hold the right bumper and push the same way. The robot drives along its own nose.
- Let go, and it drives away from you again.

No gamepad. This holds the bumper and the stick, for the picture at the top:

```
./gradlew :TeamCode:simRun --args="mytry.L150RobotRelativeButtonOpMode --left_stick_y=-0.5
--right_bumper=true"
```

Without `--right_bumper=true`, the same run was watched at `x_in` 126.674 and `y_in` 10.5. The simulator has no walls, so a run left going drives off the field.

If it didn't, and `LessonsTest.l150_theBumperSwitchesToRobotRelative` fails with `robot relative while held expected:<1.0> but was:<5.551115123125783E-17>`, the line still reads `drivetrain.sticks(robotForward, robotStrafe, turn);`. That long number is 0.0 with a rounding error from the turn.

If it didn't, and it fails with `field relative by default expected:<0.0> but was:<1.0>`, the two branches are swapped. The bumper is the exception, so the `if` reads the bumper, not `!` the bumper.

If it didn't, and it fails with `and back again on release expected:<0.0> but was:<1.0>`, the bumper flips a remembered flag instead of being read fresh every loop. That is a toggle, and this is the check that catches it.

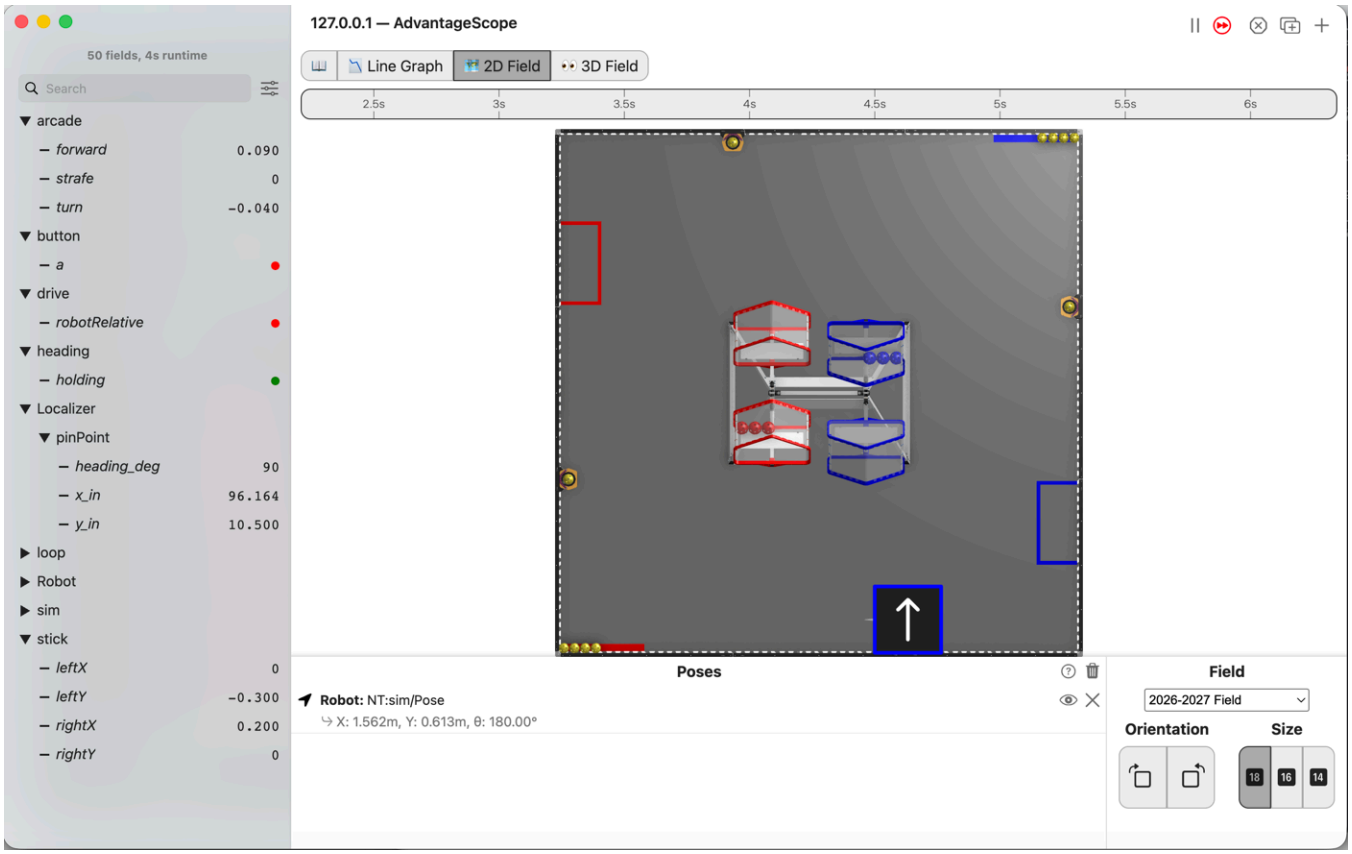
No test watches the `drive/robotRelative` line. It is there for AdvantageScope: a driver who says "it went crooked" and a graph that says which way the sticks worked settle the question between them.

What you just did. This is the first lesson that is a driver's choice rather than a robot's. Nothing here is new arithmetic. Both ways of driving existed already, and what you added is who chooses and when.

That is most of what a competition teleop turns out to be: a few ways to drive, and a gamepad laid out so a driver can reach them without looking.

Where next. L160 holds a heading for the driver, so the robot stops turning when nobody is turning it.

160 lesson: hold the heading the driver left



Let go of the turn stick and the robot keeps pointing where it was left. Push the turn stick and the driver has it back at once.

Before you start.

- L150 is done.
- AdvantageScope is installed on the same laptop. No gamepad is needed until the last step.
- The robot, the Driver Station, a gamepad, and about 2 m by 2 m of clear floor for the last step.

28.1 the steps

28.1.1 L160S010: copy the starter into mytry

Copy. In `lessons`, click `L160HeadingHoldOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L160HeadingHoldOpMode`, and the new file is open.

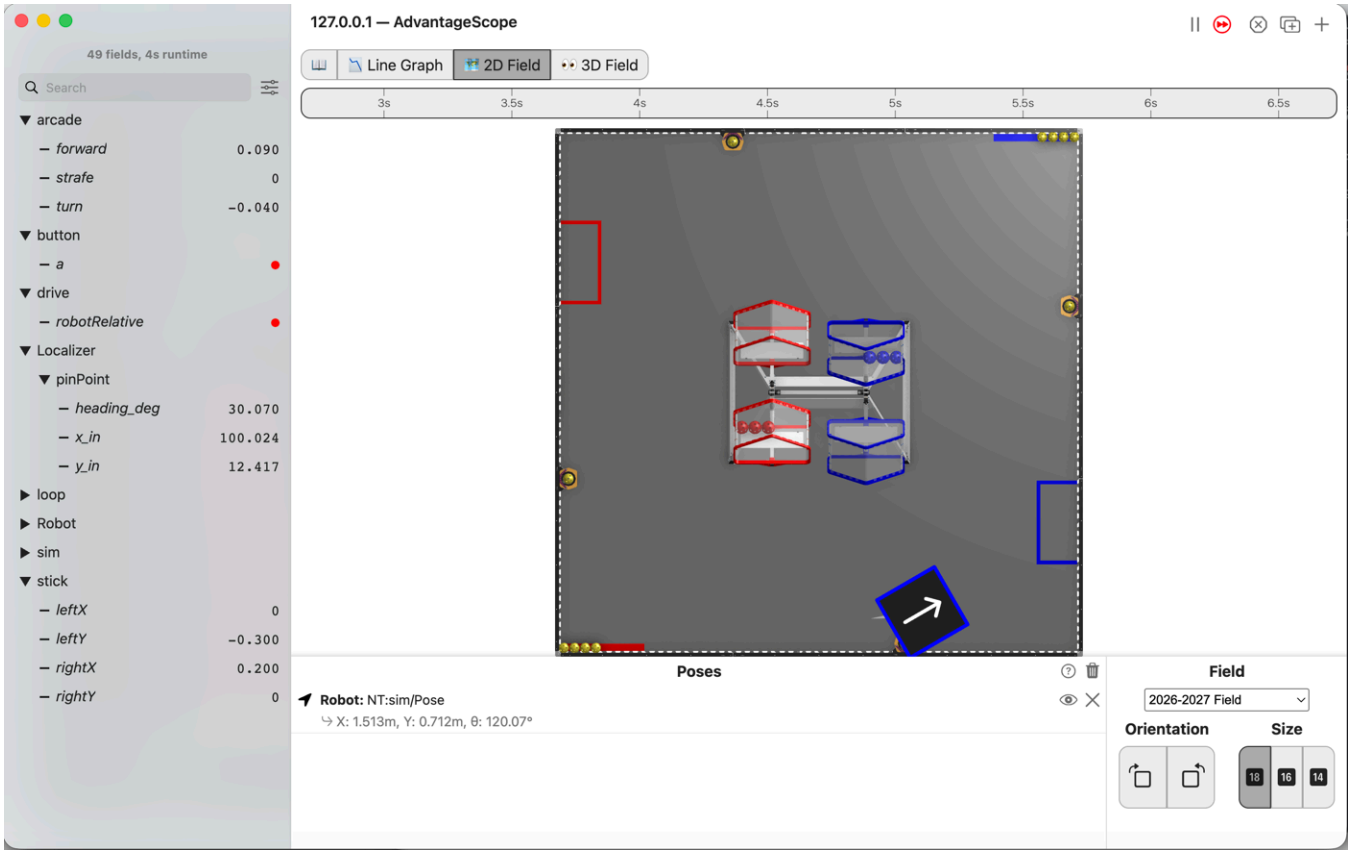
28.1.2 L160S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

28.1.3 L160S030: watch the nose creep



The starter is L150's finished OpMode. This run pretends the turn stick rests a little off center, the way a worn stick does.

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L160HeadingHoldOpMode --left_stick_y=-0.3
--right_stick_x=0.2"
```

AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `arcade` folder, then `Localizer` and `pinPoint`.

Check.

- `turn` reads -0.040, which is small.
- The robot drives to the right, and it turns as it goes.
- `heading_deg` started at 90 and keeps dropping.

28.1.4 L160S040: hold the heading

`HeadingHold` is written for you, in `base`. While the turn stick is pushed, it hands the stick straight back and the driver steers. On the loop the stick is let go, it remembers the heading. After that it hands back the turn that steers the robot back to that heading.

Its own deadband is 0.05, so a stick that is nearly centered counts as let go. `turn` at -0.040 is inside it.

Change. Press Ctrl+C in the Terminal to stop the simulator. Find the line `CorbelsMecanum drivetrain;` near the top of the class. Under it, type:

```
    CorbelsMecanum drivetrain;
    HeadingHold hold = new HeadingHold(Constants.foresightConfig.headingFeedback.get());
```

`Constants.foresightConfig` holds the follower's own settings. `headingFeedback` is the part that steers the heading in autonomous. Tune the follower, and this hold gets the same tuning.

`HeadingHold` and `Constants` turn red. Click each one and press Option+Enter, as in L120. The popup for `Constants` names `org.firstinspires.ftc.teamcode.pedro.Constants`.

Change. In `driveFromTheSticks()`, find the comment that names this step. Under it, type:

```
        double turn = deadband(-rightStickX, band);
        turn = signedSquared(turn);
        Tracker.publish("arcade/turn", turn);
        // When on L160S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l160.html#l160s040
        // for what to do here.
        turn = hold.turn(follower, turn);
        Tracker.publish("heading/holding", hold.target() != null);
```

The first new line swaps the stick's turn for the hold's. Both `sticks` calls below use `turn`, so both drive with it. Android Studio now underlines `turn` everywhere. That marks a variable that changes after it is made. It is not an error.

Simulator. Run the same command again, and connect AdvantageScope. Open the `heading` folder too.

Check.

- `holding` is green.
- The robot drives to the right, its nose still pointing up the field, and `heading_deg` stays at 90.

28.1.5 L160S050: drive it and then push it

Robot. Stop the simulator. Send the code, and run `L160 Heading Hold` from the Driver Station.

Check.

- Drive a long diagonal with the left stick, and do not touch the turn stick.
- The nose stays where it was.
- Then take hold of the robot and twist it a little. It turns back on its own.

No gamepad. The two runs above need none. With `--pad`, a driver can turn, let go, and watch the hold take over.

If it didn't, and `LessonsTest.l160_theRobotHoldsItsHeadingWhenTheStickIsReleased` fails with `it steers back`, nothing listens to the hold. Either the line is missing, or it reads `hold.turn(follower, turn);` without `turn =` in front, so its answer is thrown away.

If it didn't, and it fails with `while steering, the stick wins expected:<-1.0> but was:<1.0>`, the hold got `rightStickX` instead of `turn`. That skips the minus sign that makes a push to the right turn clockwise.

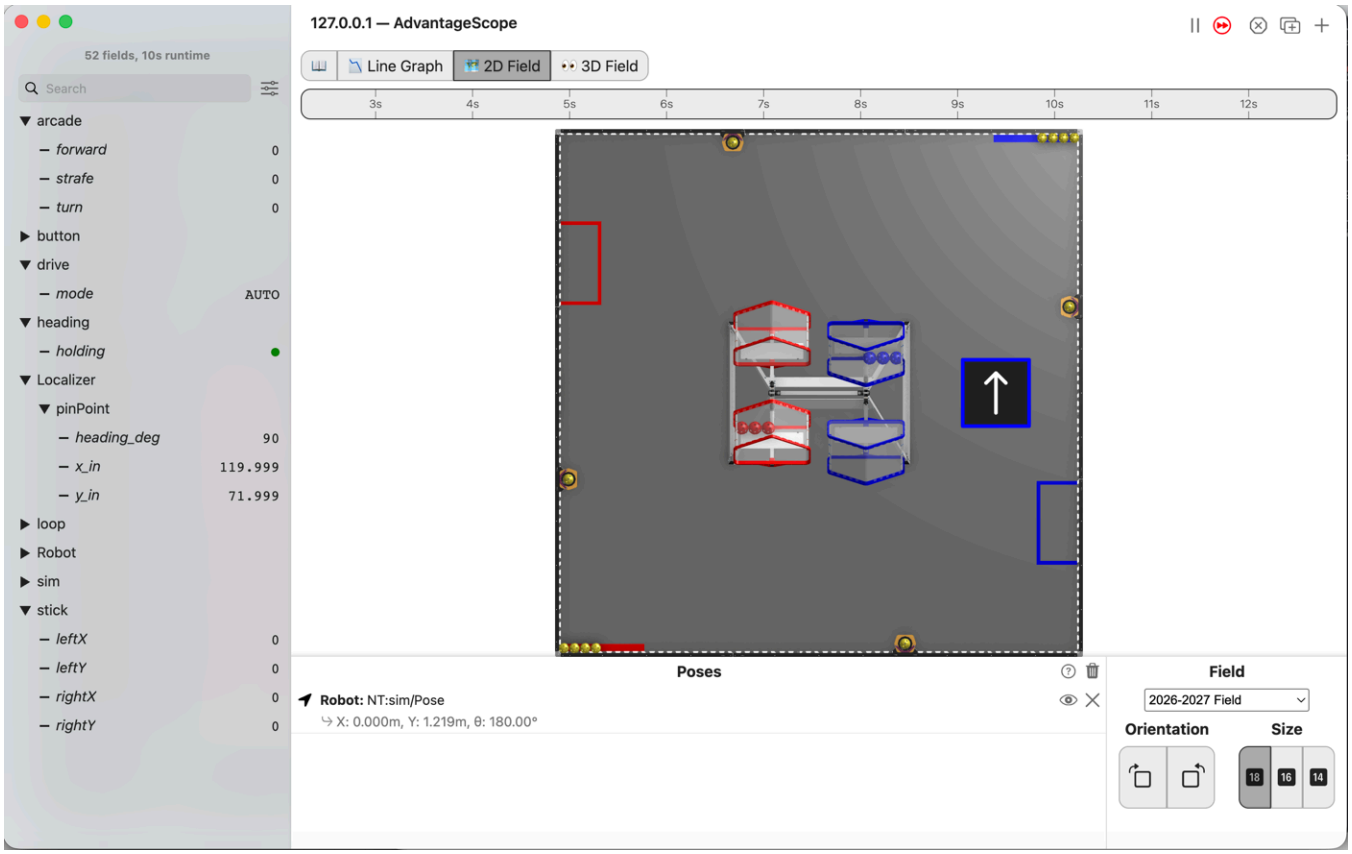
If it didn't, and on the floor the robot fights back and overshoots, that is the follower's tuning, not this lesson. L117 tunes it. If the nose drifts slowly anyway, the hold is working and the follower's heading is drifting with it.

The simulator turns the robot only when told to, so nothing there pushes it off its heading. The creep in L160S030 is a stick resting off center, not a bump.

What you just did. Three things want to turn the robot, and they take turns in a fixed order. The driver's stick beats the hold. The hold beats doing nothing. The person with their hands on the gamepad wins, and everything else fills in when they let go.

Where next. L170 gives a button a pose, and the robot drives there by itself while the driver watches.

170 lesson: press Y and the robot drives itself there



Press Y and the robot drives itself to a spot on the field. Push any stick and the driver has it back at once.

Before you start.

- L160 is done.
- AdvantageScope is installed on the same laptop. No gamepad is needed until the last step.
- The robot, the Driver Station, a gamepad, and about 2 m by 2 m of clear floor for the last step.

29.1 the steps

29.1.1 L170S010: copy the starter into mytry

Copy. In `lessons`, click `L170DriveToPoseOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L170DriveToPoseOpMode`, and the new file is open.

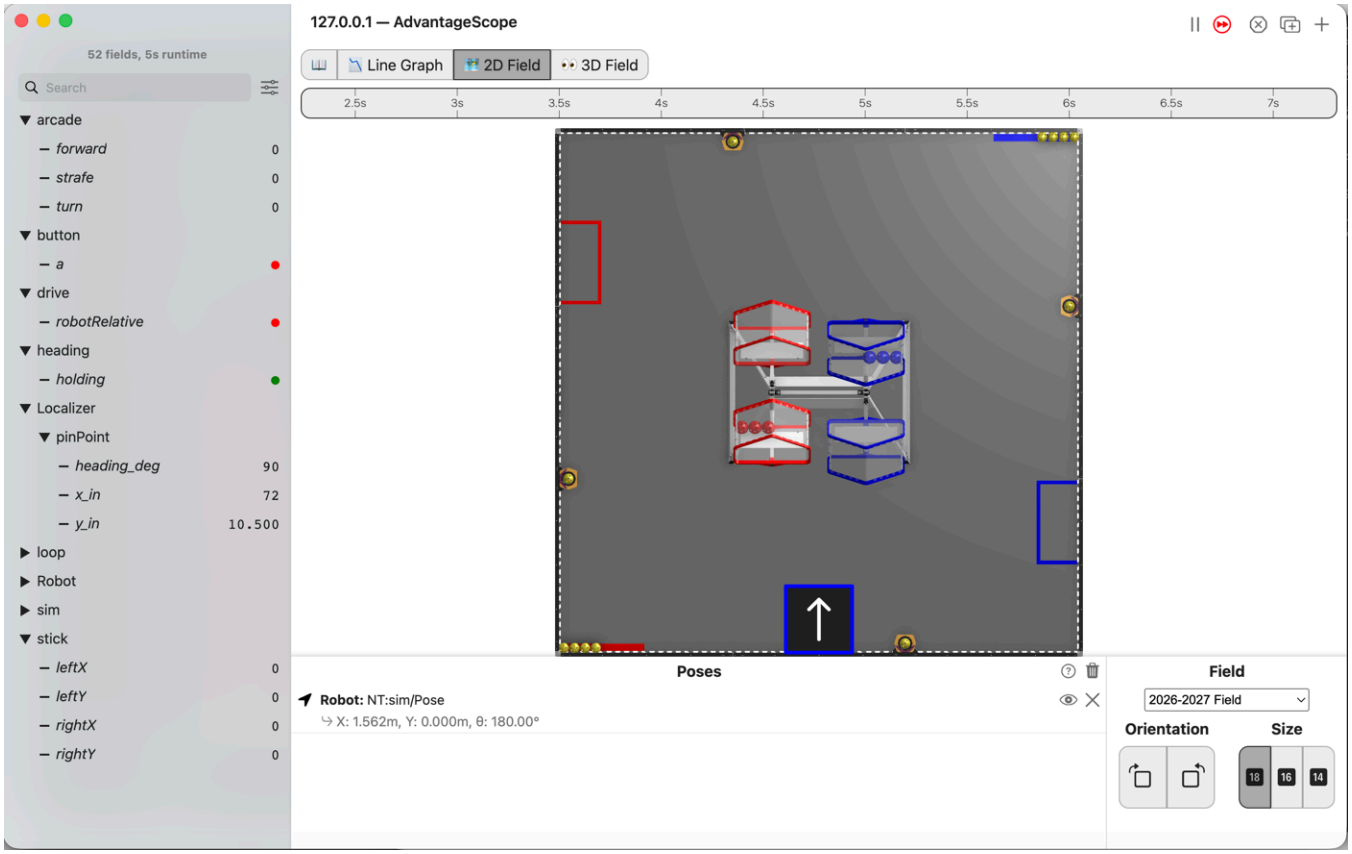
29.1.2 L170S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

29.1.3 L170S030: Y schedules the drive



The starter is L160's finished OpMode, moved around a little. Near the top are two new fields. `target` is the pose, at x 120 in, y 72 in and heading 90°. `driveToTheTarget` is a second **command**, besides the one that drives from the sticks.

`start()` builds both. The sticks' command now says `.requiring(drivetrain)`: it needs the drivetrain. `driveToTheTarget` needs it too. Only one command can have the drivetrain at a time, so the new one sets the sticks' command aside. `SUSPEND` means the sticks' command comes back when the new one ends.

`driveToTheTarget` is made of four methods near the bottom of the file. One runs when it starts, one every loop, one says when it is done, and one runs when it ends. It is done when `driverWantsControl()` says a stick is pushed.

Change. In `loop()`, find the comment that names this step. Under it, type:

```
        // When on L170S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l170.html#l170s030
        // for what to do here.
        boolean buttonY = gamepad1.y;
        boolean buttonYJustPressed = buttonY && !previousButtonY;
        if (buttonYJustPressed) {
            Scheduler.schedule(driveToTheTarget);
        }
        previousButtonY = buttonY;
```

This is L070's way of finding the loop a button goes down, for Y. `&&` means "and", and `!` means "not", so `buttonYJustPressed` is true only on the loop Y goes down. Then `Scheduler.schedule` starts `driveToTheTarget`. Holding Y does not start it again.

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L170DriveToPoseOpMode --y=true"
```

AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `Localizer` folder, then `pinPoint`. Open the `drive` folder too.

Check.

- `mode` reads AUTO.
- The robot does not move: `x_in` stays at 72 and `y_in` at 10.5.

The drive to the target has the drivetrain, but nothing has told the follower where to go yet.

29.1.4 L170S040: tell the follower where to go

Change. Press Ctrl+C in the Terminal to stop the simulator. In `startDrivingToTheTarget()`, find the comment that names this step. Under it, type:

```
        // When on L170S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l170.html#l170s040
        // for what to do here.
        follower.hold(target);
        drivetrain.releaseCommandedWheels();
```

`follower.hold` tells the follower to drive to `target` and stay there. Since L110 the sticks set the wheels. While they do, the follower is ignored. `releaseCommandedWheels()` lets go of the wheels, so the follower drives them.

When a stick is pushed, `driveToTheTarget` is done. `giveTheDriverTheWheels()` stops the follower, and the sticks' command comes back. It drives from the next loop on.

Simulator. Run the same command again, and connect AdvantageScope.

Check.

- `mode` reads AUTO.
- The robot drives right and up the field and stops at `x_in` 120 and `y_in` 72, its nose still pointing up the field. It may show 119.999 and 71.999.

29.1.5 L170S050: send it and then take it back

Robot. Stop the simulator. Send the code, and run `L170 Drive To Pose` from the Driver Station.

Check.

- Press Y once and let go. The robot drives itself toward the middle of the field, right of where it started.
- Press Y again and push the left stick while it is on its way. It stops going there, and drives the way the stick says.

No gamepad. Both runs above hold Y with `--y=true` and need none. With `--pad`, a driver can press Y and take over with a stick.

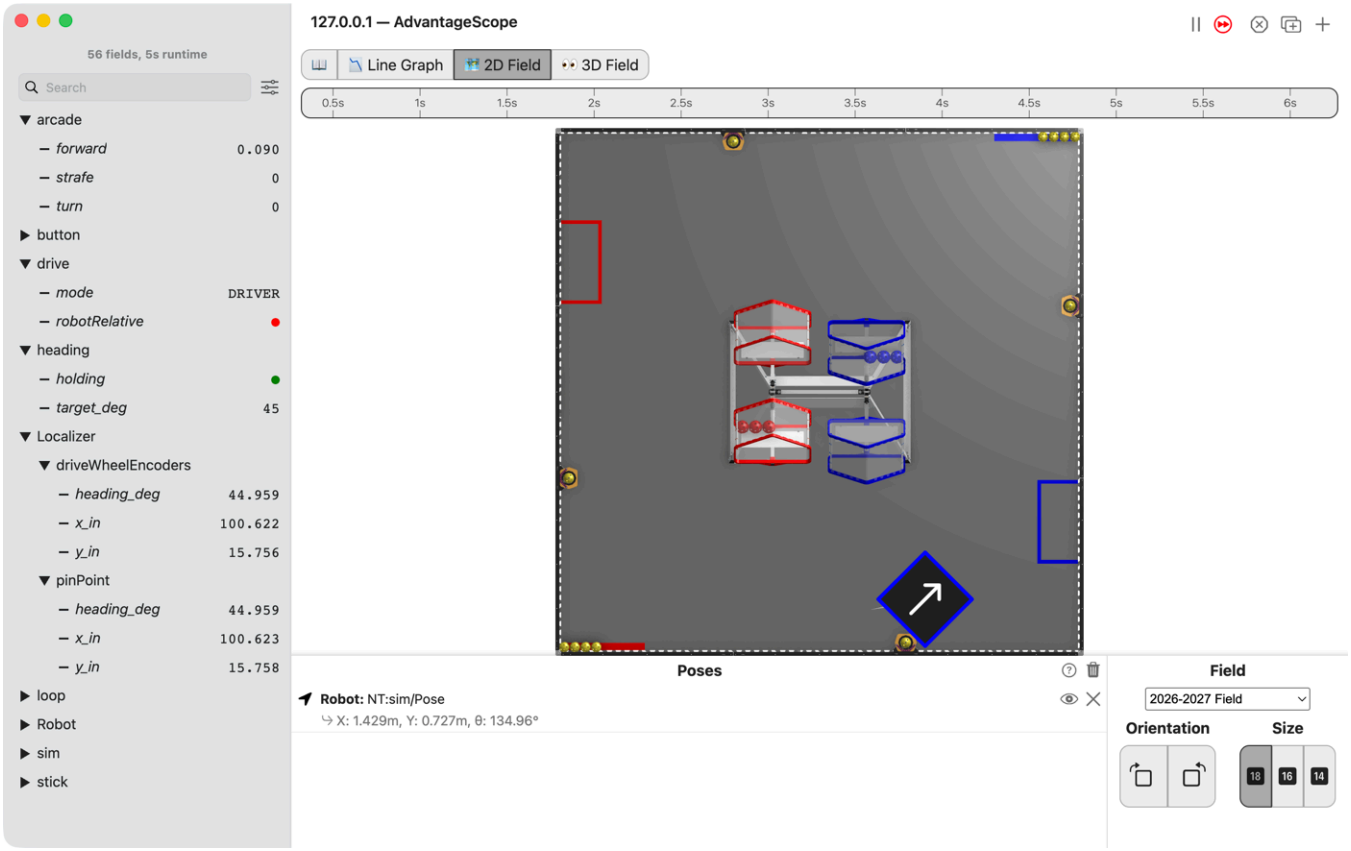
If it didn't, and `LessonsTest.l170_pressingYDrivesToAPoseAndTheDriverCanTakeOver` fails with `the command is driving, not the sticks expected:<HOLD> but was:<MANUAL>`, the follower was never told to hold. Either `follower.hold(target);` is missing, or the drive was never scheduled: the press check is missing, or reads `previousButtonY` without the `!`.

If it didn't, and the test passes but Y does nothing once the driver has driven, the `releaseCommandedWheels()` line is missing. The sticks' last wheel powers stay on the wheels, and the follower is ignored. The `--y=true` run still drives there, because Y goes down in the first loop, before the sticks have set the wheels.

What you just did. The robot has two drivers now, the person and the follower, and each is a command that needs the drivetrain. The scheduler gives the drivetrain to one at a time. The follower drives only when asked, and only until the person touches a stick. That is the same rule as L160, a step bigger: the person on the gamepad wins.

Where next. L180 puts every piece so far into one OpMode a driver could take to a match.

180 lesson: one OpMode a driver could take to a match



One teleop with all of it in. The sticks drive field relative and hold the heading they were left at. The right bumper means the robot's own directions. A points the robot at 45 degrees while the driver drives. Y sends the robot to its spot until a stick moves. And your own localizer from L120 runs beside the robot's.

Before you start.

- L170 is done, and so is L120, whose `L120EncoderLocalizer` this runs.
- AdvantageScope is installed on the same laptop. No gamepad is needed until the last step.
- The robot, the Driver Station, a gamepad, and about 3 m by 3 m of clear floor for the last step.

30.1 the steps

30.1.1 L180S010: copy the starter into mytry

Copy. In `lessons`, click `L180CombinedOpMode` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**.

Check.

- `mytry` holds `L180CombinedOpMode`, and the new file is open.

30.1.2 L180S020: switch your copy on

Change. Delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

30.1.3 L180S030: run your localizer again

Change. Find `shadowLocalizers()` and the comment that names this step. Under it, type the same two lines as in L120:

```
        // When on L180S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l180.html#l180s030
        // for what to do here.
        shadowLocalizers.add("driveWheelEncoders",
                new L120EncoderLocalizer(new HardwareWheelSource(hardware)));
```

`HardwareWheelSource` turns red. Click it and press Option+Enter, as in L120. `L120EncoderLocalizer` does not turn red. It is your own, from L120, and it sits in `mytry` too.

Keep the name `driveWheelEncoders`. AdvantageScope shows each localizer under its name, so a run of this OpMode and a run of L120 line up.

Check.

- Nothing in the file is red.

30.1.4 L180S040: read what you are about to drive

Nothing else in the starter is new. Read `loop()`, `driveFromTheSticks()` and the methods of the drive to the target, and find the lesson each part came from.

- Both drive sticks go through `deadband` and then `squared`, from L060.
- `forward` and `strafe` are turned through the heading so they mean the field, from L140.
- The right bumper picks the robot's own directions, from L150.
- `hold.turn` keeps the heading the turn stick was left at, from L160.
- Y schedules the drive to `target`, and any stick takes it back, from L170.

A is the one part that is new here, and it is `HeadingHold` again. A press of A schedules `aimAtFortyFive`,

a command that runs `aimAtFortyFiveDegrees()` once. It cancels any drive to the target, and `hold.aimAt` gives the hold a heading the driver did not leave, 45 degrees. The hold turns the robot there, and the left stick still drives. Push the turn stick and the driver has the turn back.

When a stick takes the robot back from Y, `hold.release()` in `giveTheDriverTheWheels()` forgets the heading the hold had before Y. Read the code without it: the hold would turn the robot back to that old heading.

Simulator. In the Terminal, run this. It holds A and pushes the left stick a little forward:

```
./gradlew :TeamCode:simRun --args="mytry.L180CombinedOpMode --a=true --left_stick_y=-0.3"
```

AdvantageScope. Connect to the simulator as in L020, and click **2D Field**. Open the `heading` folder. Then open `Localizer` and both folders in it.

Check.

- `target_deg` reads 45.
- The robot turns until its arrow points up and to the right, and `pinPoint/heading_deg` comes to about 45.
- All the while it drives to the right, and `pinPoint/x_in` climbs.
- `driveWheelEncoders` is there, and it follows `pinPoint`: the same heading, and x and y within a tenth of an inch.

30.1.5 L180S050: drive all four

Robot. Stop the simulator. Send the code, and run `L180 Combined` from the Driver Station.

Check.

- Try them in this order: field relative, the bumper held, A, and Y.
- Each one does what its own lesson did, and switching between them needs nothing but a thumb.
- On the robot, `driveWheelEncoders` follows `pinPoint` as the robot drives.

No gamepad. The run above needs none. With `--pad`, a driver can try all four in turn.

If it didn't, and `LessonsTest.l180_everythingTogether` fails with `the shadow localizer is still running`, nothing was published under `Localizer/driveWheelEncoders/`. The two lines are missing, or the name is spelled differently. With the name typed as `encoders`, this was the message, and the other two tests passed.

If it didn't, and one of the four misbehaves on its own, go back to that lesson's page. This page adds no arithmetic, so a fault here is a fault there.

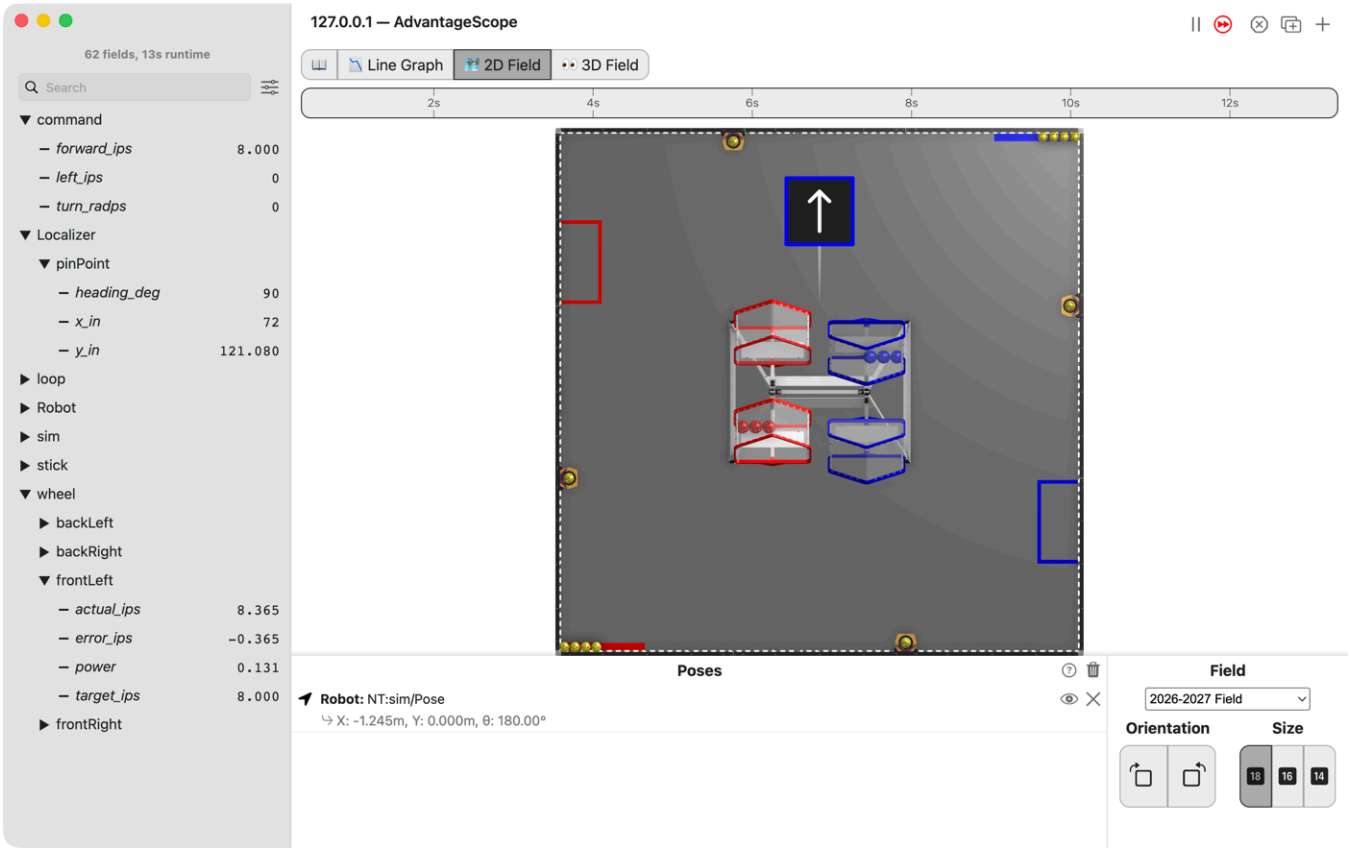
The other two tests, `l180_aPointsAt45DegreesWhileTheDriverKeepsDriving` and `l180_theTurnStickTakesAimingBack`, check the given A code. They pass on the starter.

What you just did. You wrote two lines and got a competition teleop. Every part lives in one place, so putting them together is choosing an order rather than writing code.

The order is the design. The driver's turn stick beats A's aim, and A's aim takes the place of the held heading. The follower drives only when nobody else has asked. The driver beats everything, and the robot fills in what the driver leaves.

Where next. L190 stops asking the wheels for power and starts asking them for a speed.

190 lesson: ask the wheels for a speed



Full stick means 40 inches per second, not full power. The drivetrain measures how fast each wheel turns and makes up the difference, on a fresh battery or a flat one, on the flat or up a slope.

Before you start.

- L180 is done.
- AdvantageScope is installed on the same laptop. No gamepad is needed until the last step.
- The robot, the Driver Station, a gamepad, about 3 m of clear floor, and something gentle to drive up, like a folded mat, for the last step.

31.1 the steps

31.1.1 L190S010: copy the two starters into mytry

This lesson has two files, as L110 did. One is the OpMode. The other is the drivetrain, which turns a speed into a power.

Copy. In `lessons`, click `L190SpeedDriveTrain` once and copy it. Click `mytry` once and paste. In the **Copy Class** box, click **OK**. If Android Studio asks whether to add the file to Git, click **Add**. Do the same for `L190VelocityDriveOpMode`.

Check.

- `mytry` holds `L190SpeedDriveTrain` and `L190VelocityDriveOpMode`.

31.1.2 L190S020: switch your copy on

Change. In `L190VelocityDriveOpMode`, delete the line `@Disabled`, under the comment that names this step.

Check.

- Under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

31.1.3 L190S030: make room for the drivetrain

Change. Near the top of the class, find the comment that names this step. Under it, type:

```
    // When on L190S030, see
    // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l190.html#l190s030
    // for what to do here.
    L190SpeedDriveTrain drivetrain;
```

Check.

- The line has no red underline.
- `L190SpeedDriveTrain` is in `mytry` too, so it needs no import.

31.1.4 L190S040: hand the drivetrain to the follower

Change. In `init()`, find the comment that names this step. Under it, type the first line below. Then change `initAfter();` to `initAfter(drivetrain);`:

```
        // When on L190S040, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l190.html#l190s040
        // for what to do here.
        drivetrain = new L190SpeedDriveTrain(hardware);

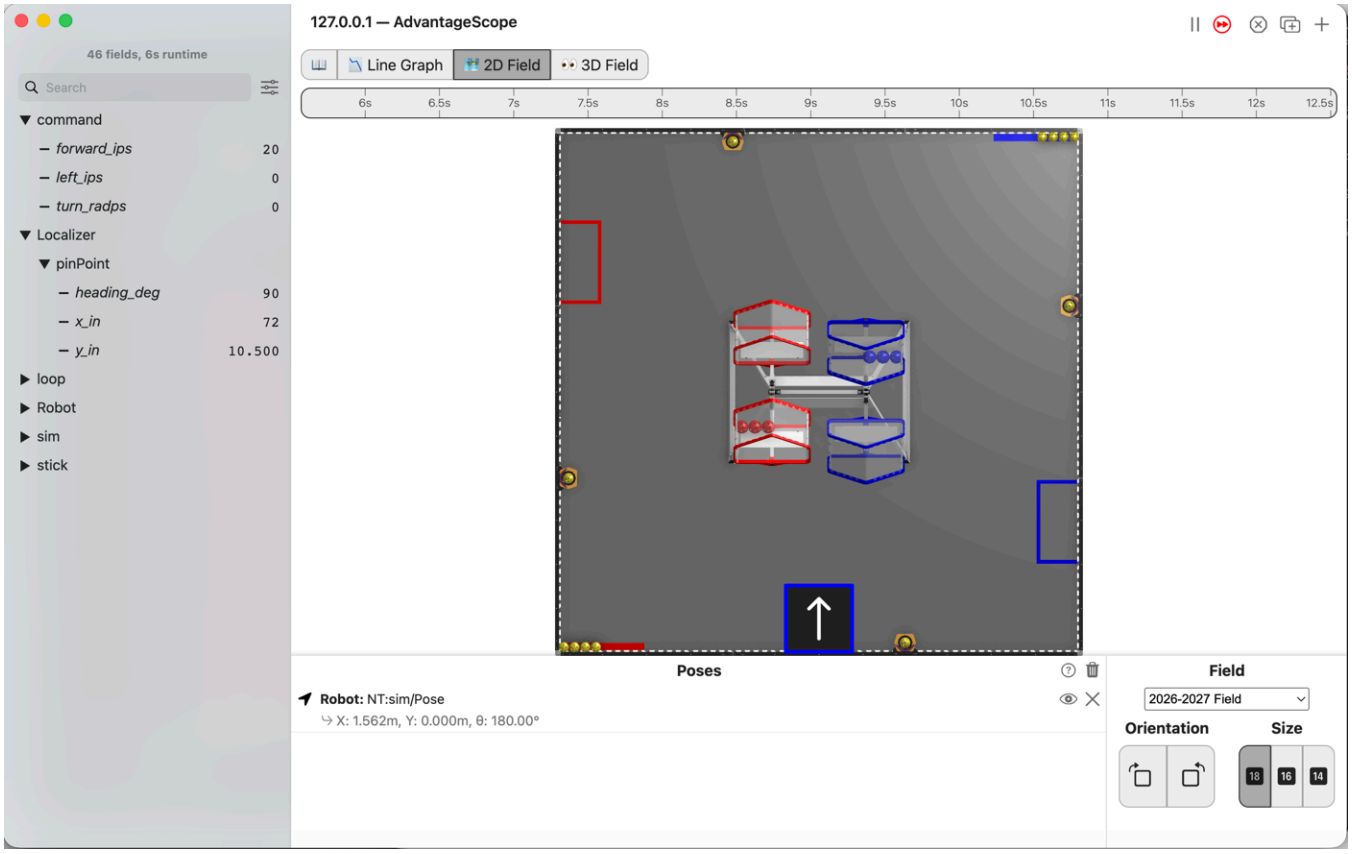
        initAfter(drivetrain);
```

This is L110S050 again, with the new drivetrain.

Check.

- Neither line has a red underline.

31.1.5 L190S050: turn the sticks into speeds



`MAX_IPS` is 40, in inches per second. `MAX_TURN_RADPS` is pi radians per second, half a turn each second. Both are near the top of the class.

Change. In `driveFromTheSticks()`, find the comment that names this step. Under it are three lines that end in `= 0;`. Change each `0` as below. Then, under the three `publish` lines, type the last two lines:

```
        // When on L190S050, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l190.html#l190s050
        // for what to do here.
        double forwardInPerS = deadband(-leftStickY, band) * MAX_IPS;
        double strafeInPerS = deadband(-leftStickX, band) * MAX_IPS;
        double turnRadPerS = deadband(-rightStickX, band) * MAX_TURN_RADPS;
        Tracker.publish("command/forward_ips", forwardInPerS);
        Tracker.publish("command/left_ips", strafeInPerS);
        Tracker.publish("command/turn_radps", turnRadPerS);
        double[] target = WheelTargets.forMecanum(forwardInPerS, strafeInPerS, turnRadPerS,
Constants.turnRadiusInches);
        drivetrain.setCommandedWheelSpeeds(target[0], target[1], target[2], target[3]);
```

The minus signs are the ones from L030. A stick pushed up reads less than 0.0.

`WheelTargets.forMecanum` works out how fast each wheel has to turn for the robot to move like that. It hands back an array, front left first, then front right, back left and back right. `turnRadiusInches` is how far each wheel sits from the middle of the robot.

`WheelTargets` and `Constants` turn red. Hold the mouse over each one until a box offers **Import class**, and click it. The box for `Constants` names `org.firstinspires.ftc.teamcode.pedro.Constants`.

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L190VelocityDriveOpMode --left_stick_y=-0.5"
```

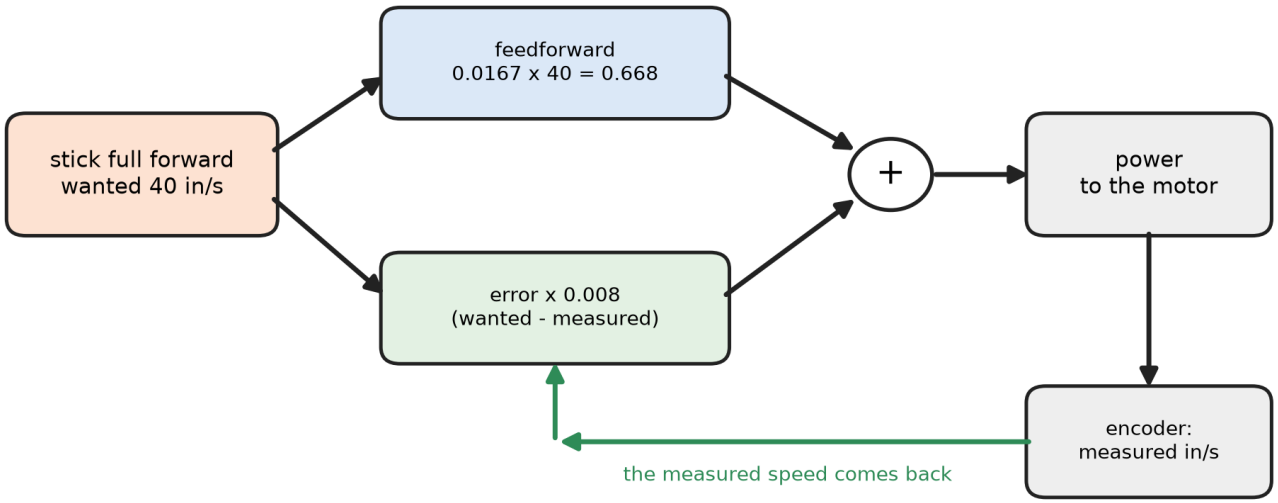
AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `command` folder. Then open `Localizer` and `pinPoint`.

Check.

- `forward_ips` reads 20, half of 40.
- The robot does not move, and `y_in` stays at 10.5.

The OpMode asks for a speed, but `setCommandedWheelSpeeds` in your drivetrain is still empty.

31.1.6 L190S060: turn a speed into a power



Open `L190SpeedDriveTrain`. It `extends CorbelsMecanum`, so it has everything L180's drivetrain had. `setCommandedWheelSpeeds` puts the four speeds into the array `wanted`. Each wheel gets its power from two parts added together.

- **Feedforward** is a guess, made before anything is measured. `Constants.powerPerInchPerSecond` is the power one inch per second needs, so the guess gets most of the way there at once.
- **Feedback** is the correction. `measuredSpeeds.all()` reads how fast each wheel really turns. The difference between wanted and measured, times `VELOCITY_KP`, is added on.

The guess is always a little off, and the correction only grows once the wheel is off its speed. Together they get there and stay there.

A `for` loop runs the lines inside its braces once for each wheel. `i` starts at 0 and goes up by 1 each time, while it is less than 4. So `wanted[i]` is `wanted[FL]` the first time around, then `FR`, `BL` and `BR`.

Change. Press Ctrl+C in the Terminal to stop the simulator. In `setCommandedWheelSpeeds`, find the comment

that names this step. Under it, type:

```
        // When on L190S060, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l190.html#l190s060
        // for what to do here.
        double[] measured = measuredSpeeds.all();
        double[] powers = new double[4];
        for (int i = 0; i < 4; i++) {
            double feedforward = Constants.powerPerInchPerSecond * wanted[i];
            double feedback = VELOCITY_KP * (wanted[i] - measured[i]);
            powers[i] = clampToPower(feedforward + feedback);
        }
        setCommandedWheels(powers[FL], powers[FR], powers[BL], powers[BR]);
        publishWheelSpeeds(wanted, measured);
```

Android Studio types the `}` of the `for` loop for you, when you press Enter after the `{`.

`clampToPower` keeps each power between -1.0 and +1.0. `setCommandedWheels` sends the four powers to the wheels, as in L110. `publishWheelSpeeds` puts each wheel's numbers in the `wheel` folder.

`Constants` turns red. Import it as in L190S050.

Simulator. Run the command again, with the stick pushed less so the robot stays on the field longer:

```
./gradlew :TeamCode:simRun --args="mytry.L190VelocityDriveOpMode --left_stick_y=-0.2"
```

AdvantageScope. Connect, and open `wheel`, then `frontLeft`.

Check.

- `forward_ips` and `target_ips` read 8.
- The robot drives up the field.
- On the first loop `power` reads 0.198: the guess, 0.134, plus the correction for a wheel not yet turning.

`actual_ips` climbs to about 8 in under half a second and settles a little above 8, so `error_ips` settles a little below 0.0. `power` drops to about 0.13 and stays there. The guess was a little too much for the simulated robot, and the correction takes off what it needs.

31.1.7 L190S070: drive it up something

Robot. Stop the simulator. Send the code, run `L190 Velocity Drive` from the Driver Station, and watch it in AdvantageScope.

Check.

- Drive on the flat first.
- Then drive straight up the mat, with the stick held in the same place. The robot slows a little at the foot of the slope and then picks back up.
- In AdvantageScope, `actual_ips` dips below `target_ips`, then climbs back, and `power` rises to do it.

No gamepad. The runs above need none. With `--pad`, a driver can push the stick and watch the numbers.

If it didn't, and every `LessonsTest.l190_` test fails with this, no wheel's speed was published under `wheel/`:

```
NullPointerException: Cannot invoke "java.lang.Double.doubleValue()" because the return
value of "java.util.Map.get(Object)" is null
```

L190S060 is not typed yet, or `publishWheelSpeeds` is missing. The test asks the log for a number nobody wrote. The same message comes when the `drivetrain.setCommandedWheelSpeeds` line in L190S050 is miss-

ing.

If it didn't, and a test fails with one of these, the line named is the one to look at. Each was planted and run.

- `full stick asks for a speed, in inches per second expected:<40.0> but was:<0.0>`: the three speeds are still `0`.
- `expected:<40.0> but was:<1.0>`: `* MAX_IPS` is missing. A full stick is 1.0, and 1 inch per second is very slow.
- `expected:<40.0> but was:<-40.0>`: the minus sign in front of `leftStickY` is missing. `l190_whenTheWheelsAreUpToSpeedOnlyTheFeedforwardRemains` fails with it, saying `expected:<0.0> but was:<-80.0>`.
- `turning counter-clockwise drives the left side backwards`, on its own: the minus sign in front of `rightStickX` is missing.
- `expected:<0.9878391425450086> but was:<0.6678391425450086>`: the feedback is not added. The difference is `0.008 * 40`.
- `expected:<0.9878391425450086> but was:<0.32>`: the feedforward is not added. A wheel already at speed then gets no power at all.
- `expected:<0.9878391425450086> but was:<0.34783914254500864>`: the error is the wrong way around, `measured[i] - wanted[i]`.
- `measured speed matches the command expected:<40.0> but was:<0.0>`: `measured` is not read from `measuredSpeeds.all()`.

Two mistakes pass every test. Both were run in the simulator with `--left_stick_y=-0.2`, and in both the `wheel` numbers were right while the robot stood still at x 72 in, y 10.5 in.

- `setCommandedWheels` is missing, so the powers are worked out and logged but never sent.
- `initAfter();` was left as it was, so the follower holds a stand-in and never sends the powers on, as in L110.

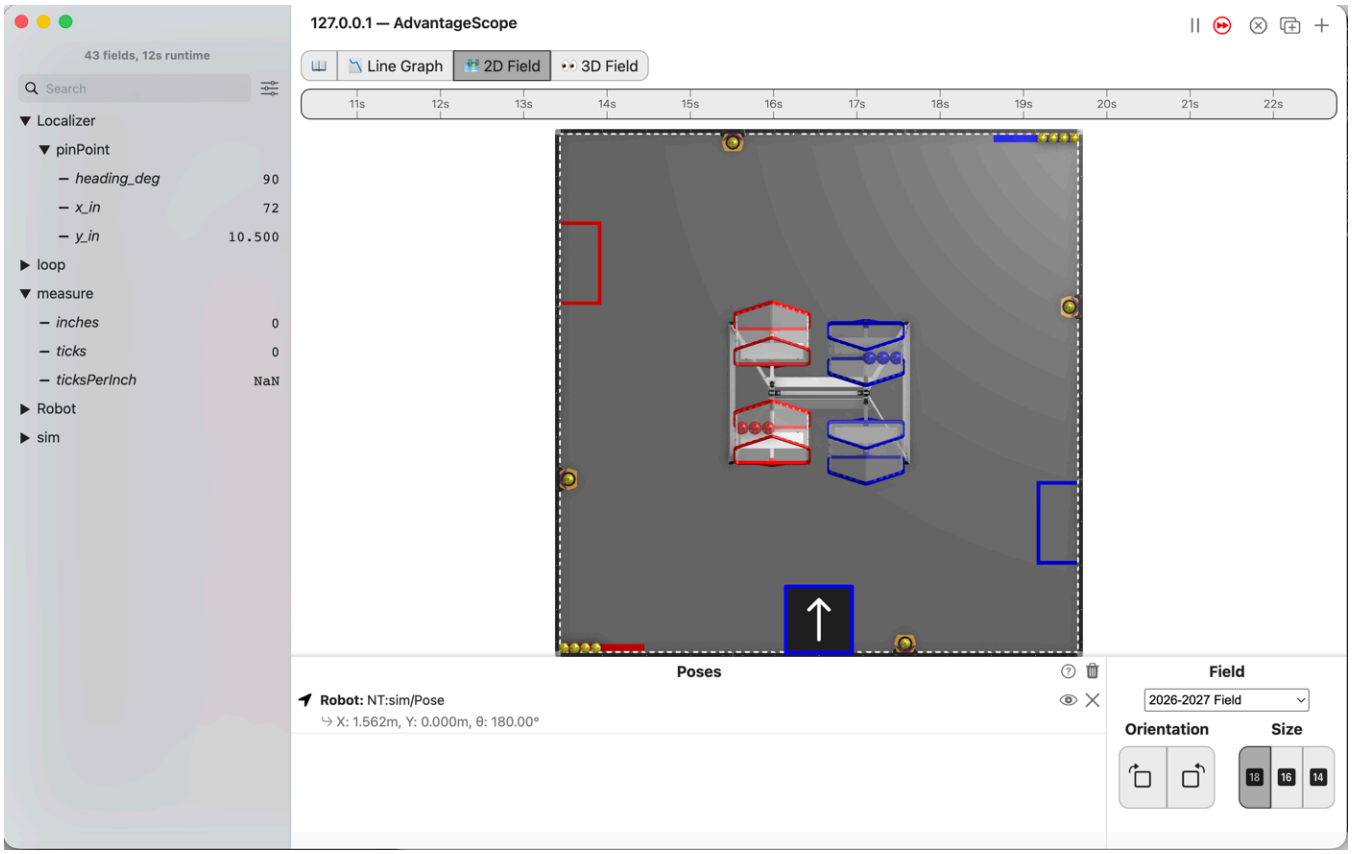
A number in the log is what the code decided, not what the robot did.

If it didn't, and on the robot it is slower than `target_ips` everywhere, `powerPerInchPerSecond` is off for this robot. L117 measures it. If turning is the wrong speed while driving straight is right, `turnRadiusInches` is 8 in, and this robot may not be. L195 measures it.

What you just did. A power is an instruction, and a speed is a result. Every lesson before this one sent instructions. This one states the result it wants, measures, and corrects.

Where next. L195 measures the numbers this lesson trusted: how far a wheel goes for each encoder count, and the turn radius.

195 lesson: measure the two numbers L190 trusted



Two numbers in `Constants.java` that came off this robot instead of somebody else's: how many encoder ticks make an inch, and how far a wheel sits from the middle. The robot measures them while you push it and spin it by hand.

Before you start.

- L190 is done. It is the lesson that uses both numbers.
- The Pinpoint is mounted, as in L115, and its offsets set, as in L117, because this measurement believes it.
- AdvantageScope is installed on the same laptop.

- The robot, the Driver Station, about 1.5 m of clear straight floor for the push, and room to spin the robot on the spot.

32.1 the steps

32.1.1 L195S010: copy the three starters into mytry

This lesson has three files. Two are OpModes, one for each measurement. The third is the drivetrain, which tells them what each wheel's encoder has counted.

Copy. In `lessons`, click `L195MeasureTicksPerInchOpMode` once. Hold Shift and click `L195TicksDriveTrain`, so all three are picked. Copy them, click `mytry` once, and paste. In the **Copy** box, click **OK**. If Android Studio asks whether to add the files to Git, click **Add**.

Copy all three at once. Copied one at a time, the OpModes still point at the drivetrain in `lessons`, and turn red.

Check.

- `mytry` holds `L195MeasureTicksPerInchOpMode`, `L195MeasureTurnRadiusOpMode` and `L195TicksDriveTrain`, and none of them shows red.

32.1.2 L195S020: switch both OpModes on

Change. In `L195MeasureTicksPerInchOpMode`, delete the line `@Disabled`, under the comment that names this step. Do the same in `L195MeasureTurnRadiusOpMode`.

Check.

- In both, under the step's comment, the next line reads `// 'public' so the robot can run this OpMode. 'public' allows access from any code.`

32.1.3 L195S030: open a door to the encoders

Open `L195TicksDriveTrain`. A drivetrain sends four powers out to the motors. `wheelTicks()` is the same shape pointing the other way: it brings four numbers back.

Change. In `wheelTicks()`, find the comment that names this step. Under it, type:

```
        // When on L195S030, see
        // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l195.html#l195s030
        // for what to do here.
        ticks[FL] = hardware.frontLeft.getCurrentPosition();
        ticks[FR] = hardware.frontRight.getCurrentPosition();
        ticks[BL] = hardware.backLeft.getCurrentPosition();
        ticks[BR] = hardware.backRight.getCurrentPosition();
```

`getCurrentPosition()` is how many ticks a motor's encoder has counted since the robot started. `FL`, `FR`, `BL` and `BR` name the slots, as in L190.

Simulator. In the Terminal, run:

```
./gradlew :TeamCode:simRun --args="mytry.L195MeasureTicksPerInchOpMode"
```

AdvantageScope. Connect to the simulator as in L020, click **2D Field**, and open the `measure` folder.

Check.

- `inches` and `ticks` read 0.0.
- `ticksPerInch` reads NaN, which means "not a number".

Nobody can push a robot in the simulator, so it stays at x 72 in and y 10.5 in. And 0.0 divided by 0.0 is not a number. The measuring is for the robot.

32.1.4 L195S040: push the robot and measure an inch

An encoder counts small steps, called **ticks**, as its motor turns. Many ticks make one turn of the wheel.

The robot keeps two accounts of the same trip. The Pinpoint says how far it went in inches, since its own wheels measure the floor. The drive encoders say how many ticks they counted. One divided by the other is the number.

Robot. Stop the simulator. Send the code, and run `L195 Measure ticks per inch` from the Driver Station.

Push. Push the robot slowly straight forward, at least 36 in, keeping it pointing the same way. Push slowly: a wheel that slips travels without turning, and the answer comes out short.

Check.

- The Driver Station shows `ticksPerInch` with a number after it.

Do it three times. Three answers a tick or two apart are a measurement.

Change. Open `Constants`, in the `pedro` folder, and put the number in `ticksPerInch`, which is 45.0 until you do. Send the code again and run it once more to check.

32.1.5 L195S050: spin the robot and measure the radius

A wheel `r` inches from the middle travels `r` inches for every radian the robot turns. So the radius is the wheel's inches divided by the radians. This OpMode turns ticks into inches with `ticksPerInch`, so L195S040 comes first.

Robot. Run `L195 Measure turn radius` from the Driver Station.

Spin. Spin the robot on the spot by hand, at least two full turns, keeping it in about the same place.

Check.

- The Driver Station shows a turn radius.
- Measure from the middle of the robot to the middle of one wheel, once across the robot and once along it. Add those two lengths. The sum is close to the turn radius the Driver Station shows.

Change. Put the number in `turnRadiusInches` in `Constants.java`, which is 8.0 until you do.

No gamepad. None of the runs above needs one. Pushing a robot by hand is not something the simulator does, so L195S040 and L195S050 are robot steps only.

If it didn't, and `LessonsTest.l195_theTicksDoorReportsWhatEachEncoderCounted` fails, the test puts 10, 20, 30 and 40 on the four encoders, so the numbers in the message name the wheel they came from. Each of these was planted and run.

- `front left, front right, back left, back right: arrays first differed at element [0]; expected:<10> but was:<0>`: L195S030 is not typed yet. Every measurement would say the robot never moved.
- `arrays first differed at element [1]; expected:<20> but was:<30>`: the front right and back left went into each other's slots. 30 is the back left's count, in the front right's place.

`getVelocity()` in place of `getCurrentPosition()` does not build. Android Studio underlines it red, and

the test run stops with:

```
error: incompatible types: possible lossy conversion from double to int
```

`getVelocity()` is how fast a wheel turns, as a decimal. `getCurrentPosition()` is the one that counts.

With `@Disabled` left in, every test passes. The OpMode is missing from the Driver Station's list.

If it didn't, on the robot, and it says `Encoders counted 0 ticks` while the inches climb, L195S030 is not on the robot you sent to.

If it didn't, and the turn radius is about half what the tape says, the robot was swung around the room rather than spun on the spot. The Pinpoint counts the turning, and driving in a circle adds turning the wheels never made.

The wheels have to roll. Both OpModes make the wheels coast in `start()` and put braking back in `stop()`. It is given code, and the other two `l195_` tests watch it. They pass on the starter. Each of these was planted in `L195MeasureTicksPerInchOpMode` and run.

- `allowConfiguredBrakeMode()` added at the end of `measureTicksPerInch()`: the coast lasts one loop, and the test says `L195MeasureTicksPerInchOpMode still coasts five loops in`. The wheels would brake while the student was still pushing.
- `allowConfiguredBrakeMode()` taken out of `stop()`: two messages, `L195MeasureTicksPerInchOpMode brakes again afterwards` and `braking is back although no loop ran`. The second is a run where STOP came before the first loop, which leaves the next OpMode coasting.

What you just did. Two constants became measurements. `Constants.java` came with 45.0 and 8.0 in it, and those are somebody else's robot. Nothing in L190 could tell they were wrong, because code cannot check a number it has no other way of knowing. Have the robot measure the same thing two ways, with two instruments that do not talk to each other, and compare.

Where next. L195 is the last lesson. L117 tunes the numbers that need the robot driving itself rather than being pushed.

900 reference: Pedro routes

Every way to say where the robot should go, and which ones to avoid. This project resolves `com.pedropathing:core:3.0.1`. Where the documentation and the jar disagree, the jar is what is written here.

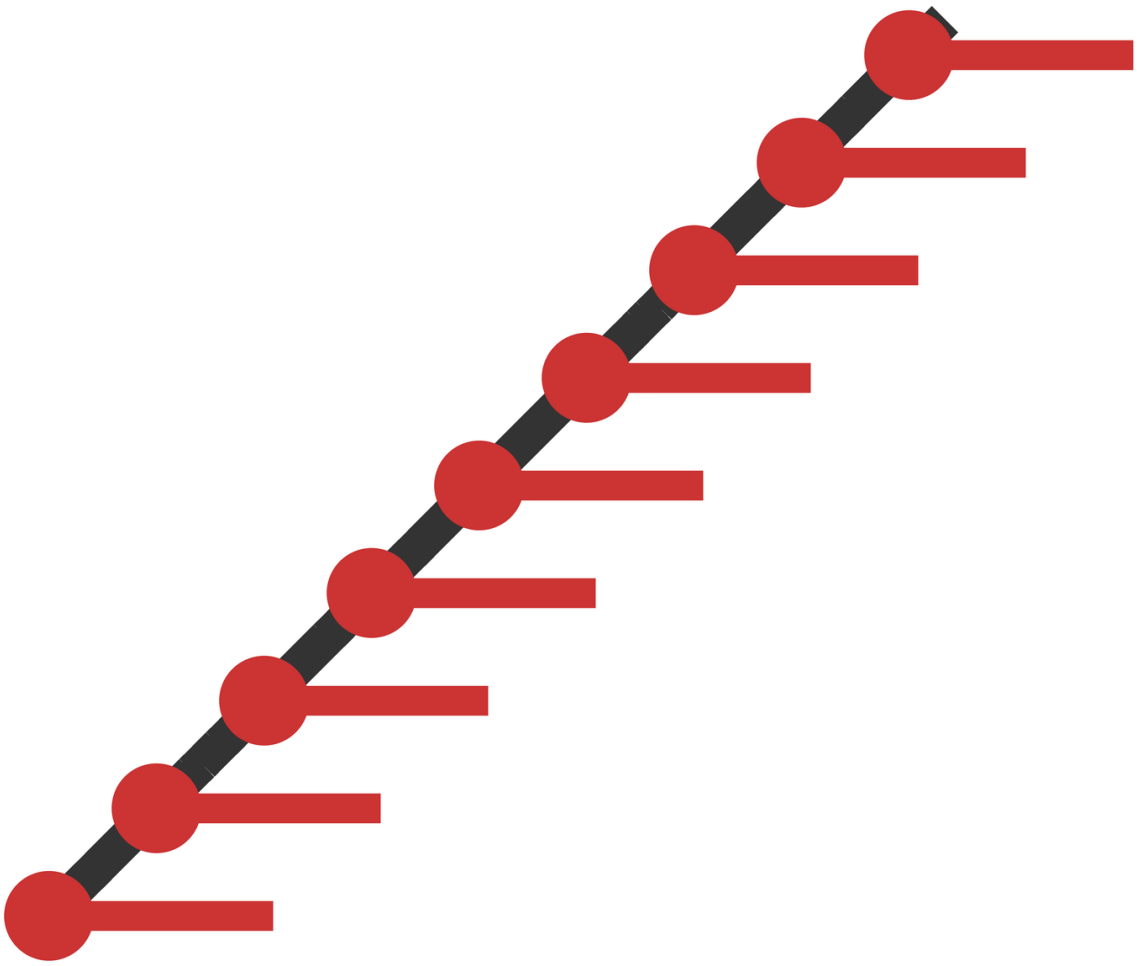
In the pictures the black line is where the robot drives and each red pin is its nose, sampled every eighth of the way along. Both come from `Path.get(t)` and `Path.heading(t)`, so they are the library's own output rather than a drawing of it.

33.1 the shapes

A route is a shape plus a heading rule. There are four shapes.

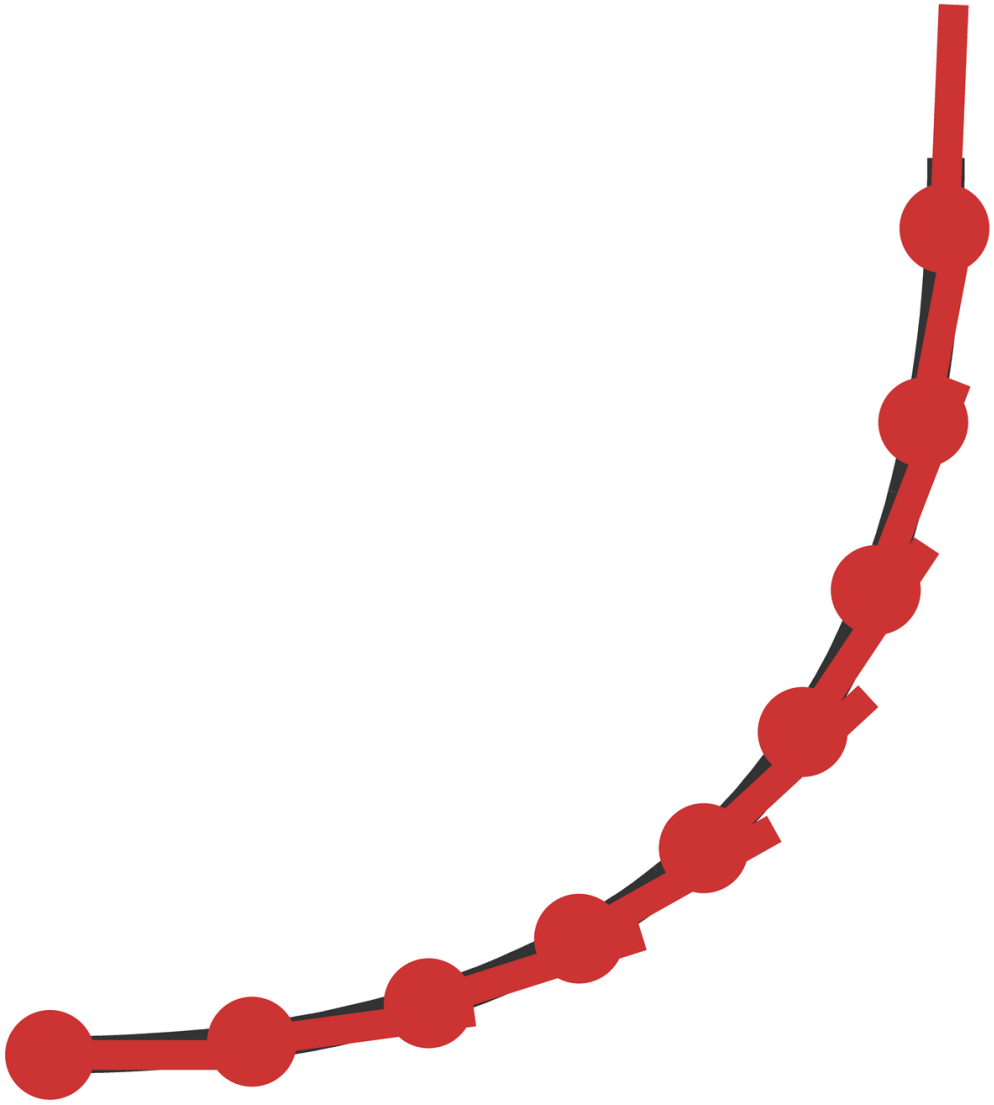
Shape	What it is
`Paths.line(start, end)`	A straight line between two poses.
`Paths.curve(start, control..., end)`	A Bézier curve. The first and last poses are the ends; every pose between them pulls the curve without being touched.
`Paths.through(poses...)`	A curve that passes through every pose given.
`Paths.path(paths...)`	Several paths run as one, in order.

33.1.1 `line`



Two poses, nothing between them. `line(start, end).constant(start)` is the plainest route there is and the one to reach for first.

33.1.2 `curve`



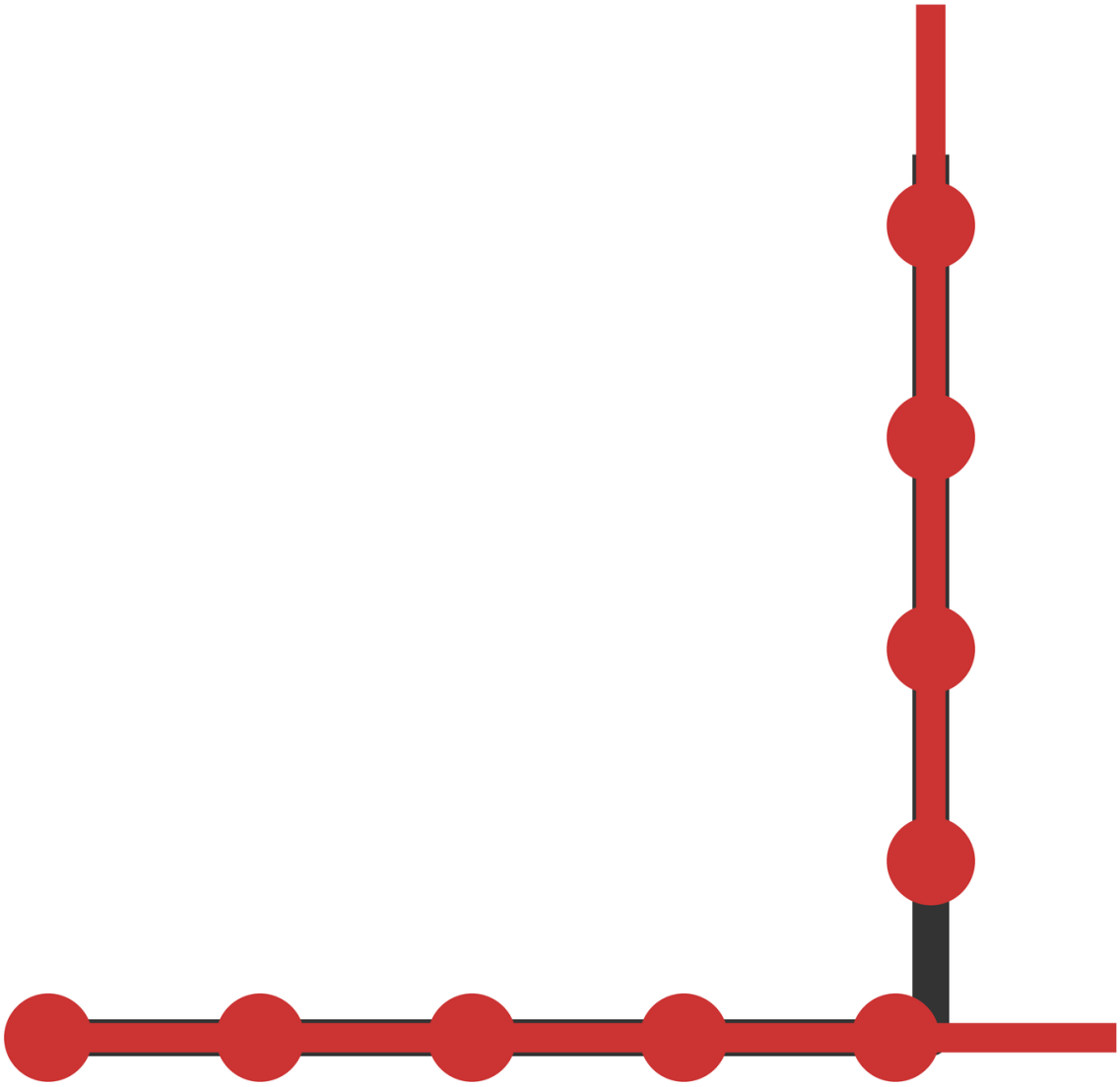
The control pose is a magnet, not a waypoint: the curve bends toward it and never reaches it. Its heading is ignored.

33.1.3 `through`



Every pose is on the route. The cost is at the ends: with `tangent()` the nose starts at -18.4 degrees and finishes at 108.4 degrees on the same three poses where `curve` runs 0 to 90, because the spline arrives at an angle. Check the end heading rather than assuming it.

33.1.4 `path`



Two legs of a route, joined. The nose snaps at the corner rather than easing round it: measured 0 degrees for the first leg and 90 for the second, with nothing in between.

33.2 the heading rules

Rule	What it does	Use it?
`constant(pose)`	Holds that pose's heading the whole way.	Yes.
`constant(radians)`	The same, from a number.	Careful: radians.
`tangent()`	Nose follows the direction of travel.	Yes.
`reverseTangent()`	Tangent plus 180 degrees, for driving backwards.	Yes.
`facingPoint(pose)`	Nose always aimed at one spot on the field.	Yes, with one edge.
`linear(a, b)`	Sweeps from one heading to another along the route.	Yes.
`longLinear(a, b)`	The same, taking the long way round.	Yes, undocumented.
`Interpolator.piecewise()`	A different rule on each stretch of the route.	Not covered here.

33.2.1 there is no default

The documentation says `tangent()` "is the default interpolation type when you create a path". It is not. A path with no heading rule throws as soon as anything asks for a heading:

```
java.lang.UnsupportedOperationException: No heading interpolator set.
```

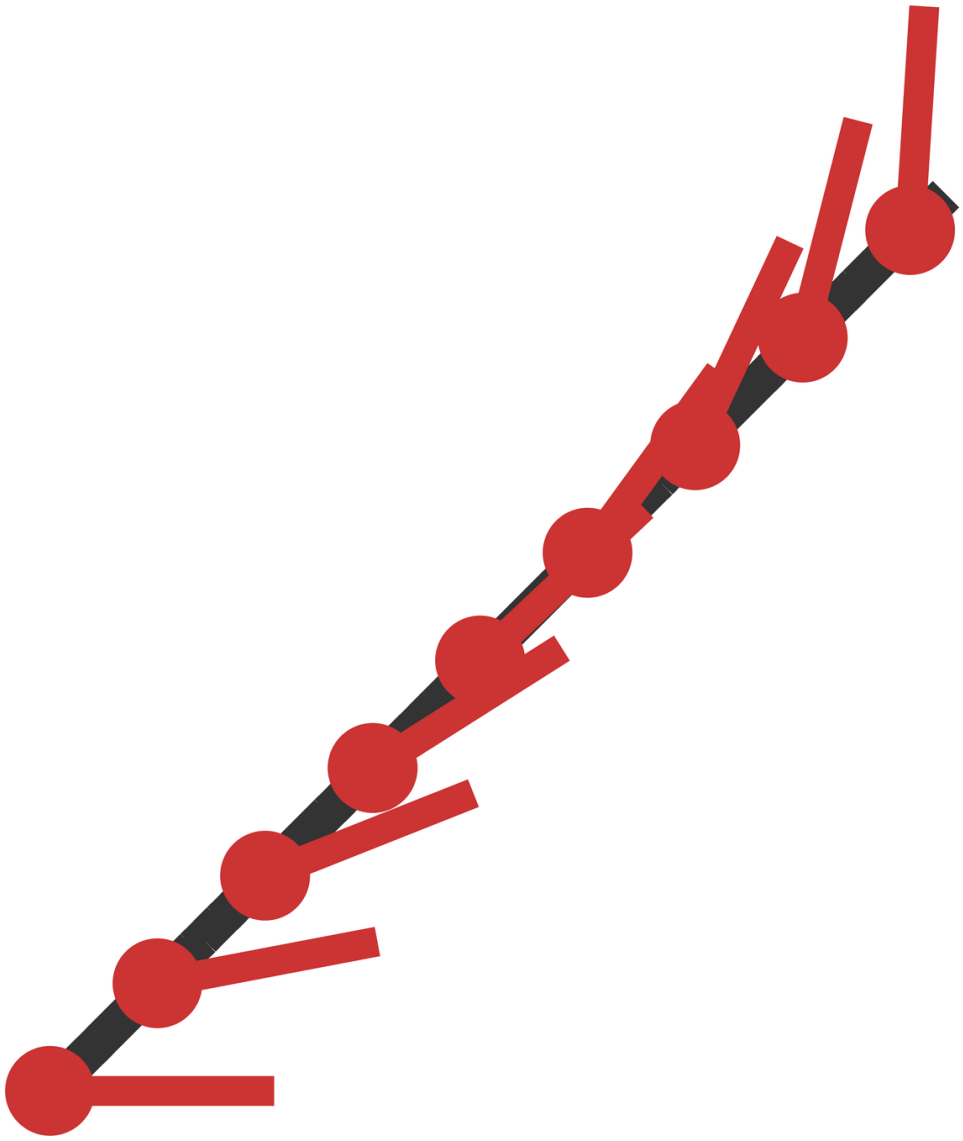
So every route needs a rule. It is not optional.

33.2.2 the number form is radians

`constant(90)` is not 90 degrees. It is 90 radians, which wraps to 116.6 degrees. `constant(0)` is safe only because zero is the same in both.

Two ways to avoid it. Pass a pose, `constant(start)`, and the heading comes from the pose. Or convert, `constant(Math.toRadians(90))`. A pose built by `PoseFactory.degrees()` converts for you; `constant` does not.

33.3 `linear()` on a line was reversed before 3.0.1



Asked to sweep from 0 to 90 degrees along a line, 3.0.1 does, and `endPose().heading()` reports the end heading. 3.0.0 swept from 90 to 0 and reported the start heading. On both:

on line(s, e)	version	t=0	t=0.5	t=1
.linear(0, toRadians(90))	3.0.1	0.0	45.0	90.0
.linear(0, toRadians(90))	3.0.0	90.0	45.0	0.0

```
.linear(e, s), swapped            3.0.1     90.0    45.0    0.0
```

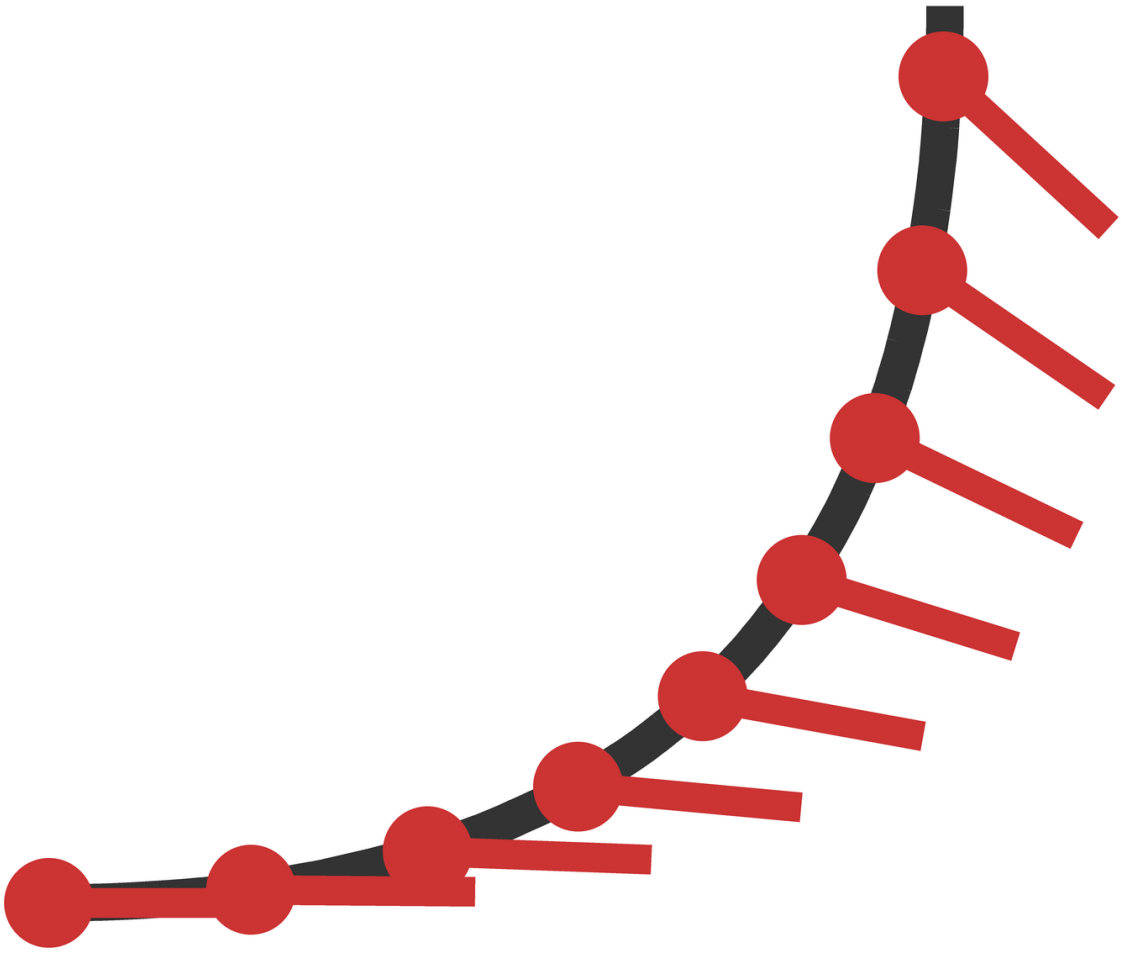
This was issue 176, with 180 and 181 reporting the same thing. The cause given there is `Curve.pathCompletion()` returning the fraction of the path remaining rather than the fraction completed.

The workaround in those issues was to swap the arguments. On 3.0.1 the swap turns the robot the wrong way, so a swapped `linear()` in code written for 3.0.0 is now a bug. On a curve, `linear()` is the same in both versions: `curve(s, m, e).linear(0, toRadians(90))` runs 0, 24.6, 45, 90.

33.4 `longLinear()` takes the long way round

`longLinear(0, toRadians(90))` runs 0, 292.5, 225, then 90 at the end: clockwise, the long way from 0 to 90. On 3.0.0 it was reversed the same way as `linear()`, running 90, 157.5, 225, then 0. It is undocumented: the documentation page lists tangent, linear, constant, facingPoint, piecewise and custom, and not this.

33.5 `facingPoint()` has one edge



It works: the nose tracks the point as the robot moves. The edge is a route that starts *at* the point it faces, where there is no direction to face. Measured on a line leaving the point, the heading reads 0 at `t=0` and the correct -135 degrees from the next sample on, so the first instant is a discontinuity. Issue 69 is about this interpolator; it is closed.

33.6 what this page does not cover

- `Interpolator.piecewise()`, which nothing here has run.
- Custom interpolators.
- Deceleration, path and end constraints, and foresight, each of which has its own reference page and changes how a route is driven rather than what it is.
- Whether any of this behaves on a floor. Everything above is the library's own arithmetic, read out of the jar. When the simulator and the robot disagree, the robot is right.

Chapter 34

910 reference: glossary

Every word here is one the task pages use, and each entry names the page that introduces it. Where the guide could have invented a friendlier word and did not, the reason is that the code, Pedro's documentation and the Driver Station all use this one.

OpMode
One program the Driver Station can run. Each lesson is an OpMode, and its name in the list is the one in its `@TeleOp` or `@Autonomous` line. L020

Simulator
A robot that exists only in the laptop. It runs the same OpMode, so code can be tried before the real robot is near. L020

AdvantageScope
The program that shows what the code published. It can show it live or from a flight log, as numbers, as graphs, or as a robot on the field. The cheat sheet lists every name the lessons publish. L020

Flight log
The file a run writes. It holds every number the code published, so you can look at the run later. L020

TeleOp and Autonomous
The two kinds of OpMode. A TeleOp reads the gamepads; an autonomous runs on its own. L125 and L130 are the two autonomous lessons. L125

Driver Station
The phone or tablet that holds the gamepads and picks the OpMode. Lines printed with `Tracker.printToDs` show up there. L050

Drivetrain
The four motors and the code that decides what power each one gets. L040 sets the four powers itself. From L110 on, a lesson's drivetrain is a class of its own. L090

Deadband
A band of stick readings near the middle that count as let go. Without one the robot creeps when nobody is touching it. L060

Method
A named piece of code you can use again. It takes numbers in and can give one back. L060

Squaring a stick
Making a small push mean a smaller power, by multiplying the stick by itself and putting the sign back. L060

Scaling the powers
Bringing two or four wheel powers down together when the biggest one is over 1. The robot still goes where it was asked. It only goes slower. L080

Strafe
Driving sideways without turning. L090

Mixing
Turning forward, sideways and turn into four wheel powers with four sums. L090

Mecanum
A wheel with rollers set at 45 degrees on its rim, which lets the robot drive sideways. The four wheels' rollers make an X seen from above. L090

Commanded wheels
Four powers a lesson has set itself. The path follower cannot overrule them. It gets the wheels back when the lesson hands them back. L110

Array
Several numbers in a row, in one variable. `wheels[FL]` is one of them. L110

Follower
Pedro's code that drives the robot along a path, or holds it at a pose. From L110 on it holds the drivetrain and the lessons reach past it. L110

Pose
Where the robot is and which way it faces: x, y and a heading. A pose is a place on the field, not a distance from the robot. L040

Heading
Which way the robot is facing, as an angle. Counter-clockwise counts up, and the code keeps it in radians even where a page says degrees. L090

Localizer
Whatever works out the pose. A robot can run more than one at a time. That is what L120 is for. L120

Shadow localizer
A second localizer that is read and logged but never steers anything. It is how two answers get compared on one run. L120

Encoder
The counter inside a drive motor that counts how far the motor has turned. L120

Dead reckoning
Working out where you are by adding up every step you took, with nothing to check it against. The drive encoders do it; the Pinpoint does it better. L120

Pinpoint
The odometry computer. It has its own two measuring wheels, and it works out the pose. The follower believes this one. L195 measures against it. L117

Path
A line or curve the follower drives along, with a heading to hold while it does. L125

Hold
Telling the follower to sit at one pose and stay there. If something pushes the robot, the follower puts it back. L130

Field relative
Sticks that mean the field. Push away from you and the robot moves away from you, whichever way its nose points. L140

Robot relative
Sticks that mean the robot's own directions. Better for lining up against something. L150

Heading hold
Keeping the nose where the driver left it, and giving it straight back the moment the turn stick moves. L160

Instant command
: A command that does its work and finishes in the same breath, rather than running for a while. A button press is usually one. L170

Feedforward
: A guess at the power a wanted speed needs, made before anything is measured. Gets most of the way there at once. L190

Feedback
: A correction worked out from two speeds: the one asked for and the one measured. It cleans up what the guess got wrong. L190

Ticks
: What an encoder counts as a motor turns. A tick is a fixed fraction of a turn, so ticks become inches once you know how many make one. L195

Ticks per inch
: How many ticks a wheel counts for an inch of travel. Measured, not looked up. L195

Turn radius
: How far a wheel rolls each time the robot spins one radian. Measure from the middle of the robot to a wheel, once across and once along, and add the two. It turns a spin speed into a wheel speed. L195

Coast and brake
: What a motor does at zero power. Braking holds the robot still; coasting lets the wheels roll, which is what a measurement you push by hand needs. L195

920 reference: keyboard shortcuts

The lessons use these keys in Android Studio. Click in the code first. Then the keys go to the code.

What it does	Windows and Ubuntu	Mac
Undo	Ctrl+Z	⌘Z
Redo	Ctrl+Shift+Z	⌘⇧Z
Cut	Ctrl+X	⌘X
Copy	Ctrl+C	⌘C
Paste	Ctrl+V	⌘V
Delete the line	Ctrl+Y	⌘Delete
Duplicate the line	Ctrl+D	⌘D
Comment out the line	Ctrl+/	⌘/
Show completions	Ctrl+Space	Control+Space

- **Delete the line, on Windows:** the first time you press Ctrl+Y, Android Studio asks whether it should Redo or Delete Line. Click **Delete Line**.
- **Comment out the line** puts `//` in front of it. Press the same keys again to take it off.
- **Show completions** lists the words that could finish what you typed. When only one fits, Android Studio types it for you.

Chapter 36

930 reference: cheat sheet

Every table here is read straight out of the lesson code, so it says what the code says.

36.1 what to run

One OpMode name for the Driver Station, one command for the laptop. Each command runs only that lesson's own tests, which is what makes a red one easy to read. A dash means no test names that lesson: its check is on the floor.

Lesson	On the robot	On the laptop
L020	`L020 Sticks` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l020*'`
L030	`L030 Speeds` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l030*'`
L040	`L040 Tank` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l040*'`
L050	`L050 Blocks` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l050*'`
L060	`L060 Shaping` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l060*'`
L070	`L070 Buttons` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l070*'`
L080	`L080 Arcade` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l080*'`
L090	`L090 Mecanum` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l090*'`
L110	`L110 Follower` (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l110*'`

L120	L120 Compare Localizers (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l120_theEncoderLocalizerRunsAlongsideAndIsLogged' --tests '*L120EncoderLocalizerTest*'`
L125	L125 Drive 24 (Autonomous)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l125*'`
L130	L130 Forward Then Strafe (Autonomous)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l130*'`
L140	L140 Field Relative (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l140*'`
L150	L150 Robot Relative Button (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l150*'`
L160	L160 Heading Hold (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l160*'`
L170	L170 Drive To Pose (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l170*'`
L180	L180 Combined (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l180*'`
L190	L190 Velocity Drive (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l190*'`
L195	L195 Measure ticks per inch (TeleOp) L195 Measure turn radius (TeleOp)	`./gradlew :TeamCode:testDebugUnitTest --tests '*LessonsTest.l195*'`

36.2 what shows up in AdvantageScope

A key ending in `...` has a name added to the end of it, one per wheel or per localizer.

Log key	Published by
Localizer/driveWheelEncoders/...	L120CompareLocalizersOpMode, L180CombinedOpMode
arcade/forward	L080ArcadeOpMode, L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode
arcade/strafe	L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode
arcade/turn	L080ArcadeOpMode, L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode
button/a	L070ButtonsOpMode, L080ArcadeOpMode, L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode
button/event	L070ButtonsOpMode, L080ArcadeOpMode, L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode
command/forward_ips	L190VelocityDriveOpMode
command/left_ips	L190VelocityDriveOpMode
command/turn_radps	L190VelocityDriveOpMode
drive/mode	L170DriveToPoseOpMode,L180CombinedOpMode
drive/robotRelative	L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode
heading/holding	L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode
heading/target_deg	L180CombinedOpMode
measure/inches	L195MeasureTicksPerInchOpMode
measure/radians	L195MeasureTurnRadiusOpMode
measure/ticks	L195MeasureTicksPerInchOpMode

measure/ticksPerInch	L195MeasureTicksPerInchOpMode
measure/turnRadius_in	L195MeasureTurnRadiusOpMode
measure/wheel_inches	L195MeasureTurnRadiusOpMode
power/backLeft	L090MecanumOpMode
power/backRight	L090MecanumOpMode
power/frontLeft	L090MecanumOpMode
power/frontRight	L090MecanumOpMode
power/left	L060ShapingOpMode, L070ButtonsOpMode, L080ArcadeOpMode
power/right	L060ShapingOpMode, L070ButtonsOpMode, L080ArcadeOpMode
speed/left	L030SpeedsOpMode, L040TankOpMode, L050BlocksOpMode, L060ShapingOpMode, L070ButtonsOpMode
speed/right	L030SpeedsOpMode, L040TankOpMode, L050BlocksOpMode, L060ShapingOpMode, L070ButtonsOpMode
stick/leftX	L070ButtonsOpMode, L080ArcadeOpMode, L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode,L190VelocityDriveOpMode
stick/leftY	L020SticksOpMode, L030SpeedsOpMode, L040TankOpMode, L050BlocksOpMode, L060ShapingOpMode, L070ButtonsOpMode, L080ArcadeOpMode, L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode,L190VelocityDriveOpMode
stick/rightX	L070ButtonsOpMode, L080ArcadeOpMode, L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode,L190VelocityDriveOpMode

stick/rightY	L020SticksOpMode, L030SpeedsOpMode, L040TankOpMode, L050BlocksOpMode, L060ShapingOpMode, L070ButtonsOpMode, L080ArcadeOpMode, L090MecanumOpMode, L110FollowerOpMode, L120CompareLocalizersOpMode, L140FieldRelativeOpMode, L150RobotRelativeButtonOpMode, L160HeadingHoldOpMode, L170DriveToPoseOpMode, L180CombinedOpMode

Generated by applying `solutions/` on `solutions-auto-schedules.1`, every lesson in order.

Chapter 37

950 reference: answers

What each lesson changes, file by file, out of the solutions line. Use it when you are stuck, not instead of being stuck.

37.1 L020SticksOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.1.1 L020

```
--- mytry/L020SticksOpMode.java  (before L020)
+++ mytry/L020SticksOpMode.java  (after L020)
@@ lines 7 to 13 @@
    7 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
    8
    9 import org.firstinspires.ftc.teamcode.base.CorbelsTeleOp;
+  10 import org.firstinspires.ftc.teamcode.base.Tracker;
   11
   12 /**
   13  * L020: read the two sticks and write down what they say. Nothing moves.
@@ lines 57 to 74 @@
   57         // When on L020S030, see
   58         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l020.html#l020s030
   59         // for what to do here.
+  60         double leftStickY = gamepad1.left_stick_y;
   61
   62         // When on L020S040, see
   63         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l020.html#l020s040
   64         // for what to do here.
+  65         Tracker.publish("stick/leftY", leftStickY);
   66
   67         // When on L020S050, see
   68         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l020.html#l020s050
   69         // for what to do here.
+  70         double rightStickY = gamepad1.right_stick_y;
+  71         Tracker.publish("stick/rightY", rightStickY);
   72     }
   73
   74     // 'public' to match the parent class. 'public' allows access from any code.
```

37.2 L030SpeedsOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on

`solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.2.1 L030

```
--- mytry/L030SpeedsOpMode.java  (before L030)
+++ mytry/L030SpeedsOpMode.java  (after L030)
@@ lines 62 to 75 @@
   62          // When on L030S030, see
   63          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l030.html#l030s030
   64          // for what to do here.
+  65          double leftSpeed = -leftStickY;
+  66          Tracker.publish("speed/left", leftSpeed);
   67
   68          // When on L030S040, see
   69          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l030.html#l030s040
   70          // for what to do here.
+  71          double rightSpeed = -rightStickY;
+  72          Tracker.publish("speed/right", rightSpeed);
   73      }
   74
   75      // 'public' to match the parent class. 'public' allows access from any code.
```

37.3 L040TankOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.3.1 L040

```
--- mytry/L040TankOpMode.java  (before L040)
+++ mytry/L040TankOpMode.java  (after L040)
@@ lines 69 to 82 @@
   69          // When on L040S030, see
   70          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l040.html#l040s030
   71          // for what to do here.
+  72          hardware.frontLeft.setPower(leftSpeed);
+  73          hardware.backLeft.setPower(leftSpeed);
   74
   75          // When on L040S040, see
   76          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l040.html#l040s040
   77          // for what to do here.
+  78          hardware.frontRight.setPower(rightSpeed);
+  79          hardware.backRight.setPower(rightSpeed);
   80      }
   81
   82      // 'public' to match the parent class. 'public' allows access from any code.
```

37.4 L050BlocksOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.4.1 L050

```
--- mytry/L050BlocksOpMode.java  (before L050)
+++ mytry/L050BlocksOpMode.java  (after L050)
@@ lines 38 to 47 @@
```

```
   38          // When on L050S040, see
   39          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l050.html#l050s040
   40          // for what to do here.
+  41          hardware.frontLeft.setDirection(hardware.mecanumConfig.frontLeftDirection.get());
+  42          hardware.frontRight.setDirection(hardware.mecanumConfig.frontRightDirection.get());
+  43          hardware.backLeft.setDirection(hardware.mecanumConfig.backLeftDirection.get());
+  44          hardware.backRight.setDirection(hardware.mecanumConfig.backRightDirection.get());
   45
   46          initAfter();
   47      }
```

37.5 L060ShapingOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.5.1 L060

```
--- mytry/L060ShapingOpMode.java  (before L060)
+++ mytry/L060ShapingOpMode.java  (after L060)
@@ lines 75 to 93 @@
   75          // When on L060S040, see
   76          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l060.html#l060s040
   77          // for what to do here.
+  78          double band = 0.05;
+  79          double leftPower = deadband(leftSpeed, band);
+  80          leftPower = signedSquared(leftPower);
+  81          Tracker.publish("power/left", leftPower);
+  82          double rightPower = deadband(rightSpeed, band);
+  83          rightPower = signedSquared(rightPower);
+  84          Tracker.publish("power/right", rightPower);
   85
-             hardware.frontLeft.setPower(leftSpeed);
-             hardware.backLeft.setPower(leftSpeed);
+  86          hardware.frontLeft.setPower(leftPower);
+  87          hardware.backLeft.setPower(leftPower);
   88
-             hardware.frontRight.setPower(rightSpeed);
-             hardware.backRight.setPower(rightSpeed);
+  89          hardware.frontRight.setPower(rightPower);
+  90          hardware.backRight.setPower(rightPower);
   91      }
   92
   93      // 'public' to match the parent class. 'public' allows access from any code.
@@ lines 99 to 117 @@
   99      // When on L060S030, see
  100      // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l060.html#l060s030
  101      // for what to do here.
+ 102      double deadband(double value, double band) {
+ 103          double alwaysPositiveMagnitude = Math.abs(value);
+ 104          boolean isCloseToZero = alwaysPositiveMagnitude < band;
+ 105          if (isCloseToZero) {
+ 106              return 0.0;
+ 107          }
+ 108          return value;
+ 109      }
  110
  111      // When on L060S050, see
  112      // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l060.html#l060s050
  113      // for what to do here.
+ 114      double signedSquared(double value) {
+ 115          return value * Math.abs(value);
+ 116      }
  117  }
```

37.6 L070ButtonsOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.6.1 L070

```
--- mytry/L070ButtonsOpMode.java  (before L070)
+++ mytry/L070ButtonsOpMode.java  (after L070)
@@ lines 35 to 41 @@
   35     // When on L070S040, see
   36     // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l070.html#l070s040
   37     // for what to do here.
+  38     boolean previousButtonA;
   39
   40     // 'public' to match the parent class. 'public' allows access from any code.
   41     @Override
@@ lines 76 to 85 @@
   76         // When on L070S060, see
   77         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l070.html#l070s060
   78         // for what to do here.
+  79         double leftStickX = gamepad1.left_stick_x;
+  80         Tracker.publish("stick/leftX", leftStickX);
+  81         double rightStickX = gamepad1.right_stick_x;
+  82         Tracker.publish("stick/rightX", rightStickX);
   83
   84         double leftSpeed = -leftStickY;
   85         Tracker.publish("speed/left", leftSpeed);
@@ lines 103 to 122 @@
  103         // When on L070S030, see
  104         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l070.html#l070s030
  105         // for what to do here.
+ 106         boolean buttonA = gamepad1.a;
+ 107         Tracker.publish("button/a", buttonA);
  108
  109         // When on L070S050, see
  110         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l070.html#l070s050
  111         // for what to do here.
+ 112         if (buttonA != previousButtonA) {
+ 113             if (buttonA) {
+ 114                 Tracker.publish("button/event", "A pressed");
+ 115             } else {
+ 116                 Tracker.publish("button/event", "A released");
+ 117             }
+ 118         }
+ 119         previousButtonA = buttonA;
  120     }
  121
  122     // 'public' to match the parent class. 'public' allows access from any code.
```

37.7 L080ArcadeOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.7.1 L080

```
--- mytry/L080ArcadeOpMode.java  (before L080)
+++ mytry/L080ArcadeOpMode.java  (after L080)
@@ lines 79 to 110 @@
```

```
   79         // When on L080S030, see
   80         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l080.html#l080s030
   81         // for what to do here.
+  82         double band = 0.05;
+  83         double forward = deadband(-leftStickY, band);
+  84         forward = signedSquared(forward);
+  85         Tracker.publish("arcade/forward", forward);
+  86         double turn = deadband(-rightStickX, band);
+  87         turn = signedSquared(turn);
+  88         Tracker.publish("arcade/turn", turn);
   89
   90         // When on L080S040, see
   91         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l080.html#l080s040
   92         // for what to do here.
+  93         double leftPower = forward - turn;
+  94         double rightPower = forward + turn;
+  95         double leftSize = Math.abs(leftPower);
+  96         double rightSize = Math.abs(rightPower);
+  97         double biggest = Math.max(leftSize, rightSize);
+  98         biggest = Math.max(1.0, biggest);
+  99         leftPower = leftPower / biggest;
+ 100         rightPower = rightPower / biggest;
+ 101         Tracker.publish("power/left", leftPower);
+ 102         Tracker.publish("power/right", rightPower);
+ 103
+ 104         hardware.frontLeft.setPower(leftPower);
+ 105         hardware.backLeft.setPower(leftPower);
+ 106         hardware.frontRight.setPower(rightPower);
+ 107         hardware.backRight.setPower(rightPower);
  108
  109         boolean buttonA = gamepad1.a;
  110         Tracker.publish("button/a", buttonA);
```

37.8 L090MecanumOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.8.1 L090

```
--- mytry/L090MecanumOpMode.java  (before L090)
+++ mytry/L090MecanumOpMode.java  (after L090)
@@ lines 86 to 123 @@
   86         // When on L090S030, see
   87         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l090.html#l090s030
   88         // for what to do here.
+  89         double strafe = deadband(-leftStickX, band);
+  90         strafe = signedSquared(strafe);
+  91         Tracker.publish("arcade/strafe", strafe);
   92
   93         // When on L090S040, see
   94         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l090.html#l090s040
   95         // for what to do here.
+  96         double frontLeftPower = forward - strafe - turn;
+  97         double frontRightPower = forward + strafe + turn;
+  98         double backLeftPower = forward + strafe - turn;
+  99         double backRightPower = forward - strafe + turn;
+ 100         double frontLeftSize = Math.abs(frontLeftPower);
+ 101         double frontRightSize = Math.abs(frontRightPower);
+ 102         double backLeftSize = Math.abs(backLeftPower);
+ 103         double backRightSize = Math.abs(backRightPower);
+ 104         double biggest = Math.max(frontLeftSize, frontRightSize);
+ 105         biggest = Math.max(biggest, backLeftSize);
+ 106         biggest = Math.max(biggest, backRightSize);
+ 107         biggest = Math.max(1.0, biggest);
+ 108         frontLeftPower = frontLeftPower / biggest;
+ 109         frontRightPower = frontRightPower / biggest;
```

```
+ 110         backLeftPower = backLeftPower / biggest;
+ 111         backRightPower = backRightPower / biggest;
+ 112         Tracker.publish("power/frontLeft", frontLeftPower);
+ 113         Tracker.publish("power/frontRight", frontRightPower);
+ 114         Tracker.publish("power/backLeft", backLeftPower);
+ 115         Tracker.publish("power/backRight", backRightPower);
+ 116
+ 117         hardware.frontLeft.setPower(frontLeftPower);
+ 118         hardware.frontRight.setPower(frontRightPower);
+ 119         hardware.backLeft.setPower(backLeftPower);
+ 120         hardware.backRight.setPower(backRightPower);
  121
  122         boolean buttonA = gamepad1.a;
  123         Tracker.publish("button/a", buttonA);
```

37.9 L110FollowerDriveTrain

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.9.1 L110

```
--- mytry/L110FollowerDriveTrain.java  (before L110)
+++ mytry/L110FollowerDriveTrain.java  (after L110)
@@ lines 44 to 53 @@
   44         // When on L110S030, see
   45         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l110.html#l110s030
   46         // for what to do here.
+  47         wheels[FL] = forward - strafe - turn;
+  48         wheels[FR] = forward + strafe + turn;
+  49         wheels[BL] = forward + strafe - turn;
+  50         wheels[BR] = forward - strafe + turn;
   51
   52         return wheels;
   53     }
```

37.10 L110FollowerOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.10.1 L110

```
--- mytry/L110FollowerOpMode.java  (before L110)
+++ mytry/L110FollowerOpMode.java  (after L110)
@@ lines 36 to 42 @@
   36     // When on L110S040, see
   37     // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l110.html#l110s040
   38     // for what to do here.
+  39     L110FollowerDriveTrain drivetrain;
   40
   41     // 'public' to match the parent class. 'public' allows access from any code.
   42     @Override
@@ lines 46 to 54 @@
   46         // When on L110S050, see
   47         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l110.html#l110s050
   48         // for what to do here.
+  49         drivetrain = new L110FollowerDriveTrain(hardware);
   50
```

```
-             initAfter();
+  51         initAfter(drivetrain);
   52     }
   53
   54     // 'public' to match the parent class. 'public' allows access from any code.
@@ lines 94 to 100 @@
   94         // When on L110S060, see
   95         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l110.html#l110s060
   96         // for what to do here.
+  97         drivetrain.sticks(forward, strafe, turn);
   98
   99         boolean buttonA = gamepad1.a;
  100         Tracker.publish("button/a", buttonA);
```

37.11 L120CompareLocalizersOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.11.1 L120

```
--- mytry/L120CompareLocalizersOpMode.java  (before L120)
+++ mytry/L120CompareLocalizersOpMode.java  (after L120)
@@ lines 9 to 15 @@
    9 import org.firstinspires.ftc.teamcode.base.CorbelsMecanum;
   10 import org.firstinspires.ftc.teamcode.base.CorbelsTeleOp;
   11 import org.firstinspires.ftc.teamcode.base.Tracker;
+  12 import org.firstinspires.ftc.teamcode.base.odometry.HardwareWheelSource;
   13
   14 /**
   15  * L120: drive around while two localizers disagree. Your localizer reads the
@@ lines 109 to 116 @@
  109         // When on L120S030, see
  110         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l120.html#l120s030
  111         // for what to do here.
+ 112         shadowLocalizers.add("driveWheelEncoders",
+ 113                 new L120EncoderLocalizer(new HardwareWheelSource(hardware)));
  114     }
  115
  116     // 'public' to match the parent class. 'public' allows access from any code.
```

37.12 L120EncoderLocalizer

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.12.1 L120

```
--- mytry/L120EncoderLocalizer.java  (before L120)
+++ mytry/L120EncoderLocalizer.java  (after L120)
@@ lines 51 to 76 @@
   51         // When on L120S040, see
   52         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l120.html#l120s040
   53         // for what to do here.
+  54         if (lastWheels == null) {
+  55             lastWheels = wheels.clone();
+  56             lastRawHeading = rawHeading;
+  57             return;
+  58         }
```

```
+  59
+  60          double frontLeft = wheels[0] - lastWheels[0];
+  61          double frontRight = wheels[1] - lastWheels[1];
+  62          double backLeft = wheels[2] - lastWheels[2];
+  63          double backRight = wheels[3] - lastWheels[3];
+  64
+  65          double forward = (frontLeft + frontRight + backLeft + backRight) / 4.0;
+  66          double strafe = (-frontLeft + frontRight + backLeft - backRight) / 4.0;
   67
   68          // When on L120S050, see
   69          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l120.html#l120s050
   70          // for what to do here.
+  71          double heading = rawHeading + headingOffset;
+  72          x += forward * Math.cos(heading) - strafe * Math.sin(heading);
+  73          y += forward * Math.sin(heading) + strafe * Math.cos(heading);
   74
   75          lastWheels = wheels.clone();
   76          lastRawHeading = rawHeading;
```

37.13 L125Drive24OpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.13.1 L125

```
--- mytry/L125Drive24OpMode.java  (before L125)
+++ mytry/L125Drive24OpMode.java  (after L125)
@@ lines 1 to 8 @@
    1 package org.firstinspires.ftc.teamcode.mytry;
+   2
+   3 import static com.pedropathing.api.Paths.line;
+   4 import static com.pedropathing.ivy.groups.Groups.sequential;
+   5 import static com.pedropathing.ivy.pedro.PedroCommands.follow;
    6
    7 import com.pedropathing.api.PoseFactory;
    8 import com.pedropathing.ivy.Command;
@@ lines 86 to 91 @@
   86          // When on L125S030, see
   87          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l125.html#l125s030
   88          // for what to do here.
-             return Command.NOOP;
+  89          return sequential(follow(follower, line(start, end).constant(start)));
   90      }
   91 }
```

37.14 L130ForwardThenStrafeOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.14.1 L130

```
--- mytry/L130ForwardThenStrafeOpMode.java  (before L130)
+++ mytry/L130ForwardThenStrafeOpMode.java  (after L130)
@@ lines 3 to 9 @@
    3 import static com.pedropathing.api.Paths.line;
    4 import static com.pedropathing.ivy.groups.Groups.sequential;
    5 import static com.pedropathing.ivy.pedro.PedroCommands.follow;
```

```
+   6 import static com.pedropathing.ivy.pedro.PedroCommands.hold;
    7
    8 import com.pedropathing.api.PoseFactory;
    9 import com.pedropathing.ivy.Command;
@@ lines 87 to 98 @@
   87
   88     Command driveForwardThenStrafe() {
   89         return sequential(
-                    follow(follower, line(start, corner).constant(start))
+  90                 follow(follower, line(start, corner).constant(start)),
   91                 // When on L130S030, see
   92                 // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l130.html#l130s030
   93                 // for what to do here.
+  94                 follow(follower, line(corner, end).constant(start)),
+  95                 hold(follower, end)
   96         );
   97     }
   98 }
```

37.15 L140FieldRelativeOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.15.1 L140

```
--- mytry/L140FieldRelativeOpMode.java  (before L140)
+++ mytry/L140FieldRelativeOpMode.java  (after L140)
@@ lines 88 to 97 @@
   88         // When on L140S040, see
   89         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l140.html#l140s040
   90         // for what to do here.
-            drivetrain.sticks(forward, strafe, turn);
+  91         double heading = follower.pose().heading();
+  92         double robotForward = forward * Math.cos(heading) + strafe * Math.sin(heading);
+  93         double robotStrafe = -forward * Math.sin(heading) + strafe * Math.cos(heading);
+  94         drivetrain.sticks(robotForward, robotStrafe, turn);
   95
   96         boolean buttonA = gamepad1.a;
   97         Tracker.publish("button/a", buttonA);
```

37.16 L150RobotRelativeButtonOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.16.1 L150

```
--- mytry/L150RobotRelativeButtonOpMode.java  (before L150)
+++ mytry/L150RobotRelativeButtonOpMode.java  (after L150)
@@ lines 91 to 103 @@
   91         // When on L150S030, see
   92         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l150.html#l150s030
   93         // for what to do here.
-            drivetrain.sticks(robotForward, robotStrafe, turn);
+  94         boolean robotRelative = gamepad1.right_bumper;
+  95         Tracker.publish("drive/robotRelative", robotRelative);
+  96         if (robotRelative) {
+  97             drivetrain.sticks(forward, strafe, turn);
```

```
+  98         } else {
+  99             drivetrain.sticks(robotForward, robotStrafe, turn);
+ 100         }
  101
  102         boolean buttonA = gamepad1.a;
  103         Tracker.publish("button/a", buttonA);
```

37.17 L160HeadingHoldOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.17.1 L160

```
--- mytry/L160HeadingHoldOpMode.java  (before L160)
+++ mytry/L160HeadingHoldOpMode.java  (after L160)
@@ lines 8 to 16 @@
    8
    9 import org.firstinspires.ftc.teamcode.base.CorbelsMecanum;
   10 import org.firstinspires.ftc.teamcode.base.CorbelsTeleOp;
+  11 import org.firstinspires.ftc.teamcode.base.HeadingHold;
   12 import org.firstinspires.ftc.teamcode.base.Tracker;
+  13 import org.firstinspires.ftc.teamcode.pedro.Constants;
   14
   15 /**
   16  * L160: hold the heading the driver left. Let go of the turn stick and the
@@ lines 36 to 42 @@
   36     boolean previousButtonA;
   37
   38     CorbelsMecanum drivetrain;
+  39     HeadingHold hold = new HeadingHold(Constants.foresightConfig.headingFeedback.get());
   40
   41     // 'public' to match the parent class. 'public' allows access from any code.
   42     @Override
@@ lines 86 to 93 @@
   86         // When on L160S040, see
   87         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l160.html#l160s040
   88         // for what to do here.
+  89         turn = hold.turn(follower, turn);
+  90         Tracker.publish("heading/holding", hold.target() != null);
   91
   92         double strafe = deadband(-leftStickX, band);
   93         strafe = signedSquared(strafe);
```

37.18 L170DriveToPoseOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.18.1 L170

```
--- mytry/L170DriveToPoseOpMode.java  (before L170)
+++ mytry/L170DriveToPoseOpMode.java  (after L170)
@@ lines 83 to 94 @@
   83         // When on L170S030, see
   84         // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l170.html#l170s030
   85         // for what to do here.
+  86         boolean buttonY = gamepad1.y;
+  87         boolean buttonYJustPressed = buttonY && !previousButtonY;
```

```
+  88          if (buttonYJustPressed) {
+  89              Scheduler.schedule(driveToTheTarget);
+  90          }
+  91          previousButtonY = buttonY;
   92
   93          loopAfter();
   94      }
@@ lines 158 to 165 @@
  158          // When on L170S040, see
  159          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l170.html#l170s040
  160          // for what to do here.
+ 161          follower.hold(target);
+ 162          drivetrain.releaseCommandedWheels();
  163      }
  164
  165      /** Each loop of the drive to the target: the follower drives, not the sticks. */
```

37.19 L180CombinedOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.19.1 L180

```
--- mytry/L180CombinedOpMode.java  (before L180)
+++ mytry/L180CombinedOpMode.java  (after L180)
@@ lines 14 to 20 @@
   14 import org.firstinspires.ftc.teamcode.base.CorbelsTeleOp;
   15 import org.firstinspires.ftc.teamcode.base.HeadingHold;
   16 import org.firstinspires.ftc.teamcode.base.Tracker;
+  17 import org.firstinspires.ftc.teamcode.base.odometry.HardwareWheelSource;
   18 import org.firstinspires.ftc.teamcode.pedro.Constants;
   19
   20 /**
@@ lines 110 to 117 @@
  110          // When on L180S030, see
  111          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l180.html#l180s030
  112          // for what to do here.
+ 113          shadowLocalizers.add("driveWheelEncoders",
+ 114                  new L120EncoderLocalizer(new HardwareWheelSource(hardware)));
  115      }
  116
  117      // 'public' to match the parent class. 'public' allows access from any code.
```

37.20 L190SpeedDriveTrain

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.20.1 L190

```
--- mytry/L190SpeedDriveTrain.java  (before L190)
+++ mytry/L190SpeedDriveTrain.java  (after L190)
@@ lines 2 to 8 @@
    2
    3 import org.firstinspires.ftc.teamcode.base.CorbelsMecanum;
    4 import org.firstinspires.ftc.teamcode.base.RobotHardware;
+   5 import org.firstinspires.ftc.teamcode.pedro.Constants;
    6
```

```
    7 /**
    8  * L190: the drivetrain that asks the wheels for a speed.
@@ lines 39 to 52 @@
   39          // When on L190S060, see
   40          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l190.html#l190s060
   41          // for what to do here.
+  42          double[] measured = measuredSpeeds.all();
+  43          double[] powers = new double[4];
+  44          for (int i = 0; i < 4; i++) {
+  45              double feedforward = Constants.powerPerInchPerSecond * wanted[i];
+  46              double feedback = VELOCITY_KP * (wanted[i] - measured[i]);
+  47              powers[i] = clampToPower(feedforward + feedback);
+  48          }
+  49          setCommandedWheels(powers[FL], powers[FR], powers[BL], powers[BR]);
+  50          publishWheelSpeeds(wanted, measured);
   51      }
   52 }
```

37.21 L190VelocityDriveOpMode

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.21.1 L190

```
--- mytry/L190VelocityDriveOpMode.java  (before L190)
+++ mytry/L190VelocityDriveOpMode.java  (after L190)
@@ lines 8 to 15 @@
    8
    9 import org.firstinspires.ftc.teamcode.base.CorbelsTeleOp;
   10 import org.firstinspires.ftc.teamcode.base.Tracker;
+  11 import org.firstinspires.ftc.teamcode.base.WheelTargets;
+  12 import org.firstinspires.ftc.teamcode.pedro.Constants;
   13
   14 /**
   15  * L190: ask the wheels for a speed. Full stick means 40 inches per second, not
@@ lines 44 to 50 @@
   44      // When on L190S030, see
   45      // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l190.html#l190s030
   46      // for what to do here.
+  47      L190SpeedDriveTrain drivetrain;
   48
   49      // 'public' to match the parent class. 'public' allows access from any code.
   50      @Override
@@ lines 54 to 62 @@
   54          // When on L190S040, see
   55          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l190.html#l190s040
   56          // for what to do here.
+  57          drivetrain = new L190SpeedDriveTrain(hardware);
   58
-              initAfter();
+  59          initAfter(drivetrain);
   60      }
   61
   62      // 'public' to match the parent class. 'public' allows access from any code.
@@ lines 88 to 101 @@
   88          // When on L190S050, see
   89          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l190.html#l190s050
   90          // for what to do here.
-             double forwardInPerS = 0.0;
-             double strafeInPerS = 0.0;
-             double turnRadPerS = 0.0;
+  91          double forwardInPerS = deadband(-leftStickY, band) * MAX_IPS;
+  92          double strafeInPerS = deadband(-leftStickX, band) * MAX_IPS;
+  93          double turnRadPerS = deadband(-rightStickX, band) * MAX_TURN_RADPS;
   94          Tracker.publish("command/forward_ips", forwardInPerS);
   95          Tracker.publish("command/left_ips", strafeInPerS);
```

```
   96          Tracker.publish("command/turn_radps", turnRadPerS);
+  97          double[] target = WheelTargets.forMecanum(forwardInPerS, strafeInPerS, turnRadPerS,
Constants.turnRadiusInches);
+  98          drivetrain.setCommandedWheelSpeeds(target[0], target[1], target[2], target[3]);
   99      }
  100
  101      // 'public' to match the parent class. 'public' allows access from any code.
```

37.22 L195TicksDriveTrain

What each lesson's patch does to this file, in the order the lessons come, from applying `solutions/` on `solutions-auto-schedules.1`. A line marked + is new, and a line marked - is gone; each number is the line's place in the file after the lesson.

37.22.1 L195

```
--- mytry/L195TicksDriveTrain.java  (before L195)
+++ mytry/L195TicksDriveTrain.java  (after L195)
@@ lines 30 to 39 @@
   30          // When on L195S030, see
   31          // https://msb-stitt.github.io/ftc31459-26-27-biobuzz/guide/tasks/l195.html#l195s030
   32          // for what to do here.
+  33          ticks[FL] = hardware.frontLeft.getCurrentPosition();
+  34          ticks[FR] = hardware.frontRight.getCurrentPosition();
+  35          ticks[BL] = hardware.backLeft.getCurrentPosition();
+  36          ticks[BR] = hardware.backRight.getCurrentPosition();
   37          return ticks;
   38      }
   39 }
```

Chapter 38

990 where this came from

38.1 the guide's own terms

These pages are licensed Creative Commons Attribution-NonCommercial 4.0 International, which is CC BY-NC 4.0 for short.

You may share these pages and build on them. Credit Mike Stitt, link the license, and do not sell what you make.

The repository's own `LICENSE` is FIRST's notice over the FTC SDK. It covers the code, not these pages.

38.2 Game Manual 0

Game Manual 0 is licensed CC BY-NC 2.0 Generic. It is a general FTC reference, written by FTC teams. This guide is not trying to replace it. These pages are about one robot and one set of lessons.

One page leans on it. L090 explains why the mecanum rollers make an X, and links GM0's page on holonomic drivetrains, which shows it. No page here copies GM0's words. The guide's license is chosen to keep every restriction GM0's asks for anyway.

38.3 Pedro Pathing

Pedro Pathing is the path following library the lessons hand the drivetrain to. It is licensed BSD 3-Clause, and its name and logo are its own. This guide links Pedro's documentation rather than restating it, and the code the lessons write is ours.

38.4 the FTC SDK

The **FTC SDK** is FIRST's, under the BSD-style notice in the repository's root `LICENSE`. The Driver Station, the Control Hub and every `@TeleOp` in this guide are its.

38.5 AdvantageScope

AdvantageScope, from the Mechanical Advantage team, is what opens a flight log after a run.

38.6 not affiliated with FIRST

FIRST and FIRST Tech Challenge are trademarks of For Inspiration and Recognition of Science and Technology.

This guide is written by one team for its own students. It is not produced by FIRST, endorsed by FIRST, or affiliated with FIRST.