FRC Kitbot Programming prekickoff - 2026

Team 4533
Phoenix
10 JANUARY 2026

Presentation Overview

- This is based on 2025 Kitbot code and 2025 WPILIB
- Expect 2026 Kitbot code and 2026 WPILIB posted by FRC soon
- We'll go over:
  - Reference Sources
  - Setting up coding environment
  - Steps to compile
  - Steps to deploy
  - Some test/troubleshoot info

Important Resources

REFERENCE DOCS:

Search **”FRC Control System”** or visit
https://docs.wpilib.org/en/stable/docs/zero-to-robot/introduction.html
Notice left column there has so much info..

But go through steps 2 thru 4 on that page as they also are very helpful.

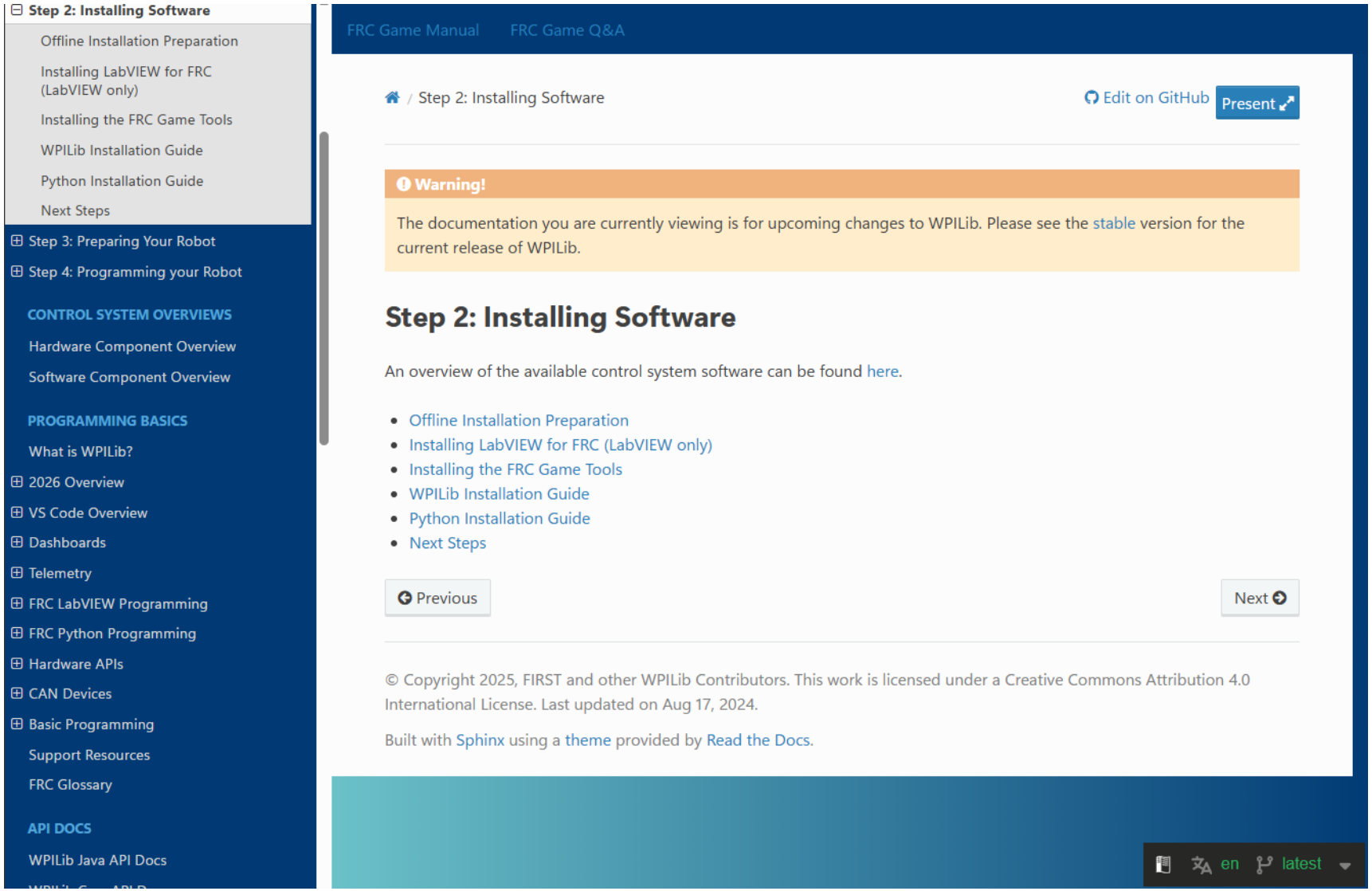
Kitbot programming guide PDF along with example code in zip file:
https://www.firstinspires.org/resources/library/frc/kitbot

We’ll refer to that PDF as the Kitbot Java PDF. There is also a Kitbot PDF related to building the actual robot hardware found on the Kitbot order page.

Development Environment

- Java is most popular Coding Language in FRC
  - Popular = examples are easier to find
  - Easier to get help with JAVA from other teams or FSA
  - some teams use Python or C++ or Labview instead of Java
- VScode is the usual editor
  - also known as an IDE (Integrated Development Environment)
- Gradle is behind the scene to do the build and deployment
- Git is used for revision control and coding sharing

Set up environment – follow suggestions



Search FRC Control System or visit

https://docs.wpilib.org/en/latest/docs/zero-to-robot/step-2/index.html

Make sure says Latest Here !

Install tips for Programming Environment

- Window Laptop for now (you will need drivestation, etc)
- First Install the FRC Game Tools
  - Brings along drivestation and Roborio image tool
- Second Install WPILib
  - Brings along VS Code
- Install Rev Client if using Sparks (not needed until time to deploy or setup CAN IDs)
- Install CTR Phoenix tool if using Krakens or other CTR devices
- Install Git for revision control

Install tips continued

- Decide where to keep your code on your computer
  BUT DO NOT store code on Onedrive, it will not compile correctly
  And will bite you at competition for sure
- example:
  C:\FRC might be good folder to create and put your projects in
- Once you've downloaded one time, you will understand how to save it all off to Flash drive for multiple computer's use
- Linux PC can be used for writing code, just Drivestation won't run there this year (maybe next year?)

Install Rev Hardware client

Follow instructions and allow to update: https://docs.revrobotics.com/rev-hardware-client/gs/install

This allows Spark firmware and more to get updated, and also to assign CAN IDs.

If using CTR hardware instead of REVrobotics, you would use CTR Phoenix Tuner, similar procedure

Installation

⚠ Those using REV ION products on REVLib 2026 or newer must use REV Hardware Client 2 ↗.

Before starting download the latest version of the REV Hardware Client.

REV Hardware Client - Version 1.7.5 ↗

System Requirements

- Operating System: Windows 10 (64-bit) or newer
- Processor: 64-bit

ⓘ As of April 12, 2024 Windows 10 or later is required for the latest version of the REV Hardware Client. Please use 1.6.4 if you are on an older version of Windows. ↗

Installation Instructions

- Download the REV Hardware Client Installer
- Run the Installer
- Run the REV Hardware Client from the Windows Start Menu or a desktop shortcut

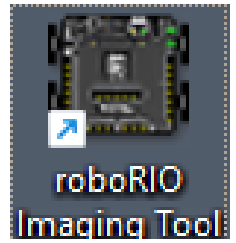
After setting up Development Environment, you then “Image the RoboRio”

Visit https://docs.wpilib.org/en/stable/docs/zero-to-robot/step-3/index.html

Walks you through Imaging your RoboRio and setting up your radio.

Programmers need to know how to do this.

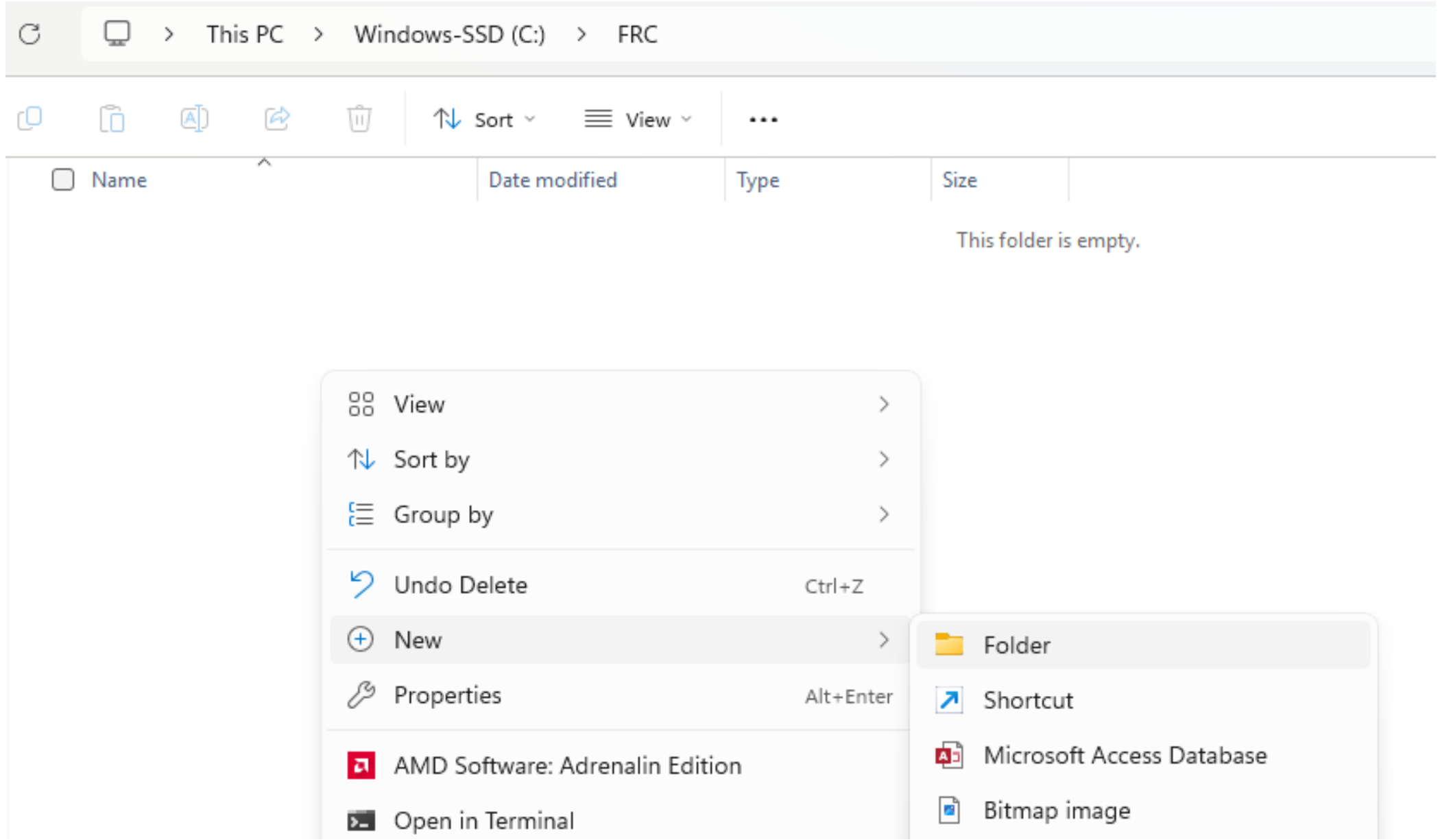
Roborio Imaging Tool provides a way to update the Roborio firmware if needed,

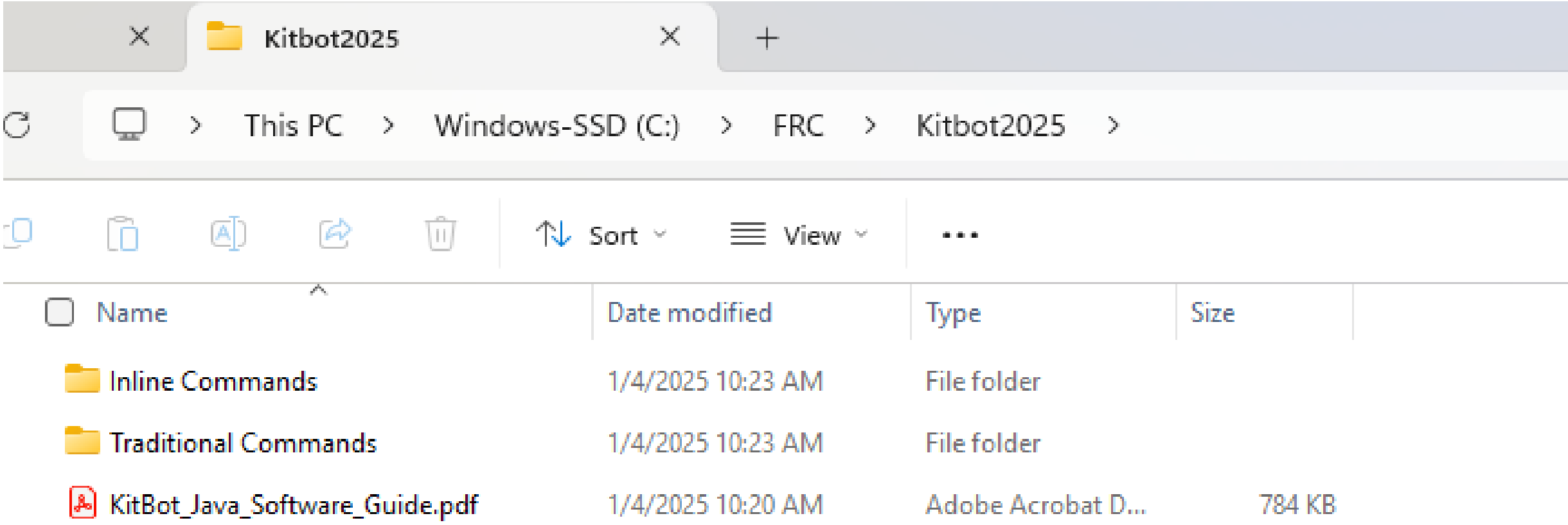


But almost always you will need to update the Roborio Image from year to year, and **at competition it will slow you down** if you did not do this at it is required to be up to date, as is all firmwares on all robot devices cabaple of firmware updates.

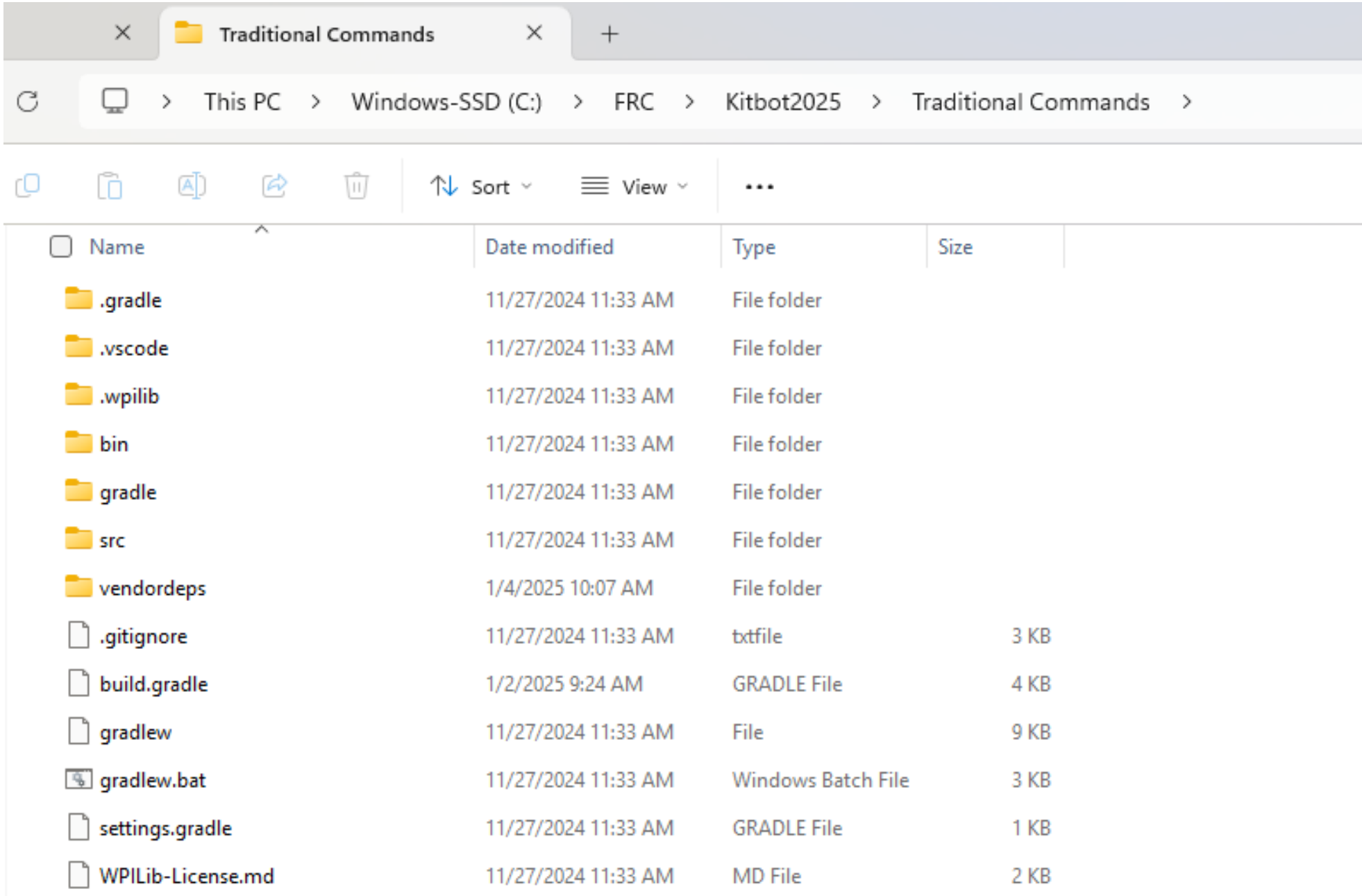
Kitbot Programming starting point

- visit https://www.firstinspires.org/resources/library/frc/kitbot
- download the zip file of code for that bot (might also be on github)
- open up file explorer , find that C:\FRC folder or make one
- and again right click and create folder for Kitbot
- Drag in the content of the Zip file
- Launch VS


This PC
Windows-SSD (C:)
FRC
Sort
View
Name
Date modified
Type
Size
This folder is empty.
View
Sort by
Group by
Undo Delete
Ctrl+Z
New
Properties
Alt+Enter
AMD Software: Adrenalin Edition
Open in Terminal
Folder
Shortcut
Microsoft Access Database
Bitmap image



Zipfile had 2 separate Code directories, you can rename or use as is
and PDF file is Guide, good reference
Start with Traditional Commands folder, probably best to copy directory
and rename the copied directory as your TestBot for starting out

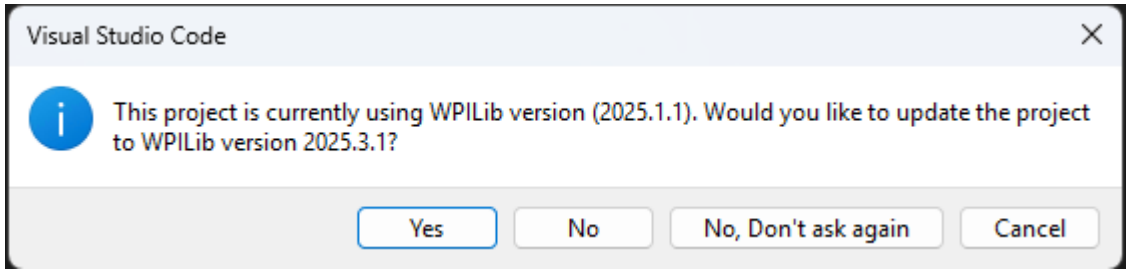


Next open up vscode

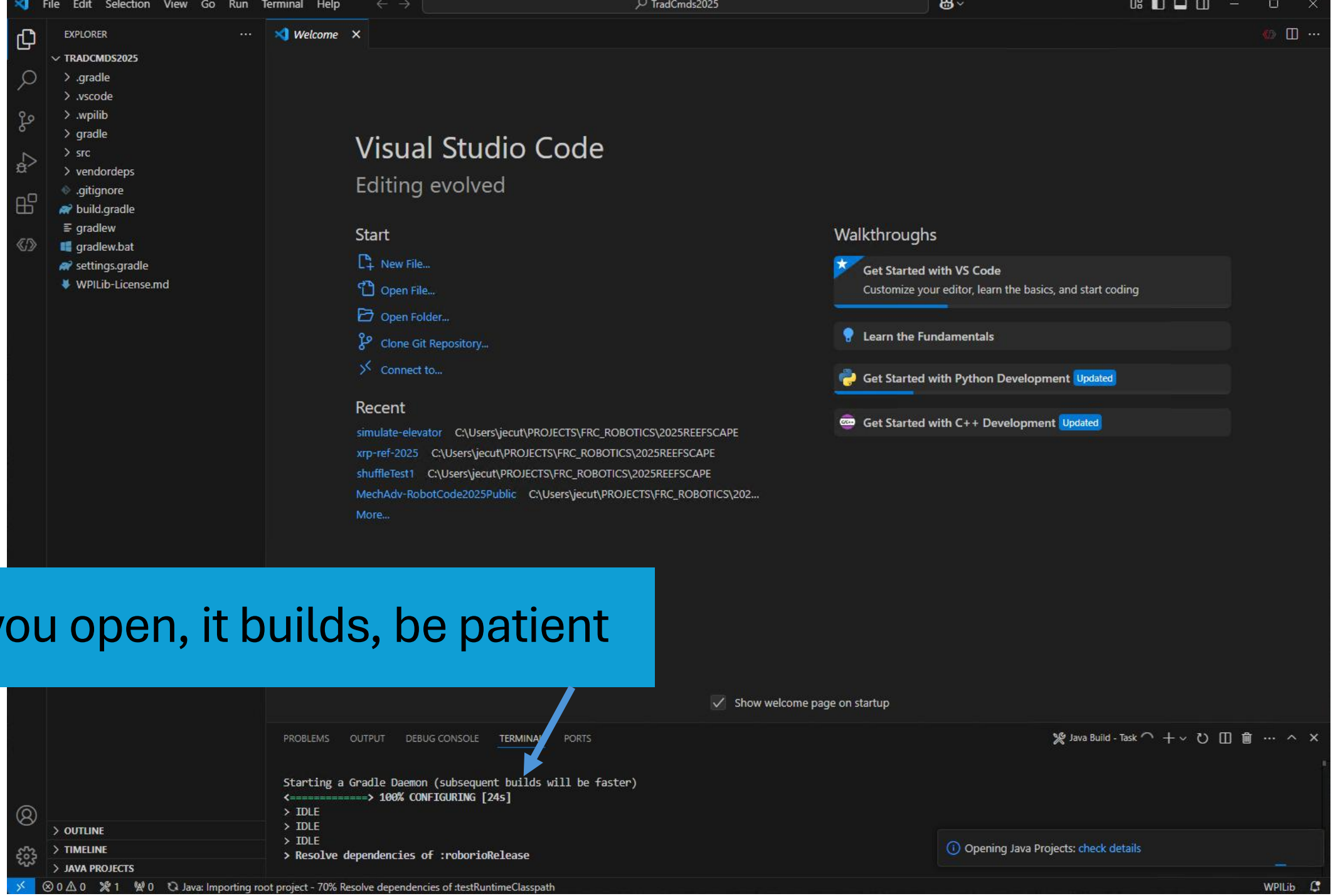
Launch VS Code

Icons from previous installs may appear along with Documentation links, use the Icon with year that makes sense

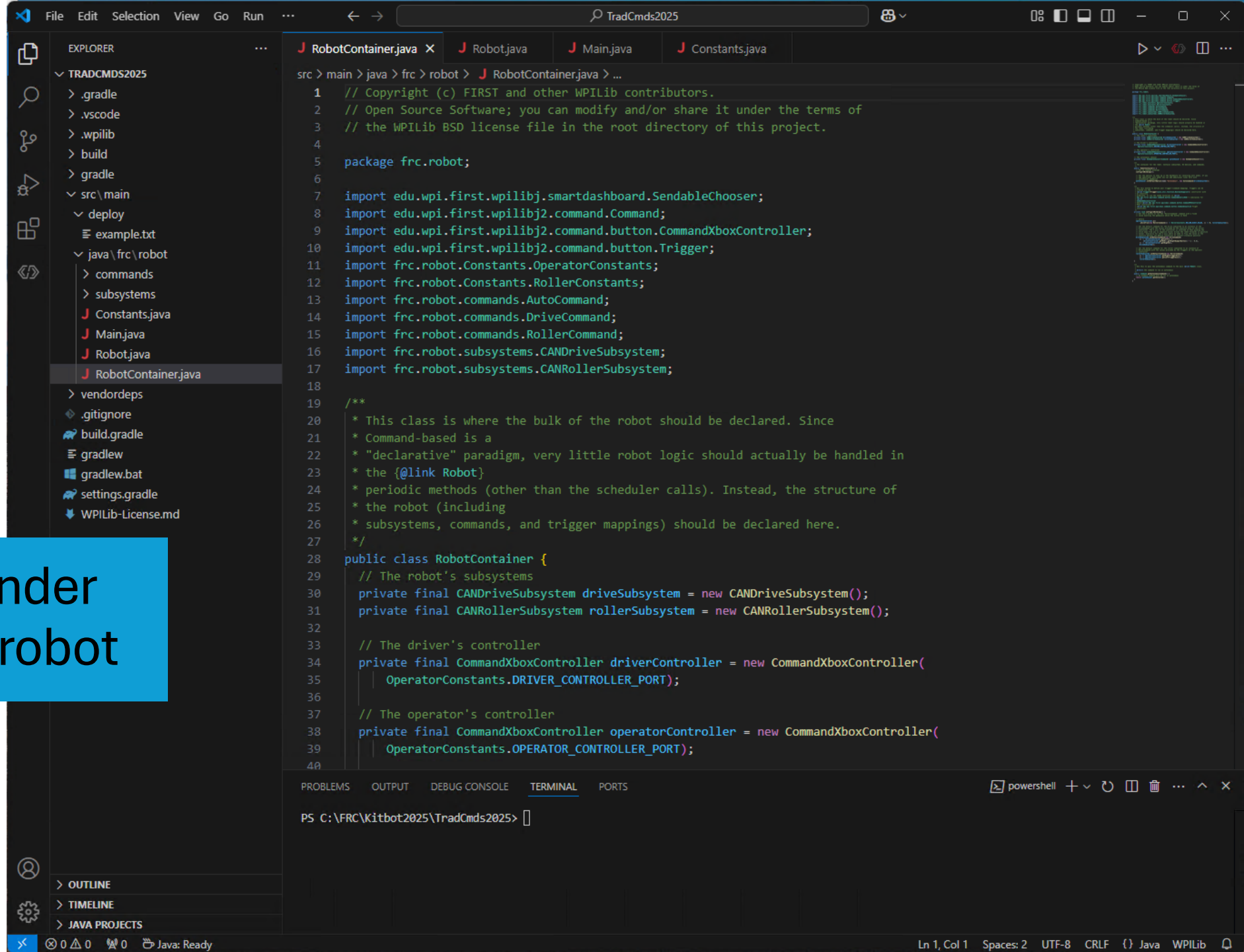
First Time you open existing or Old Project



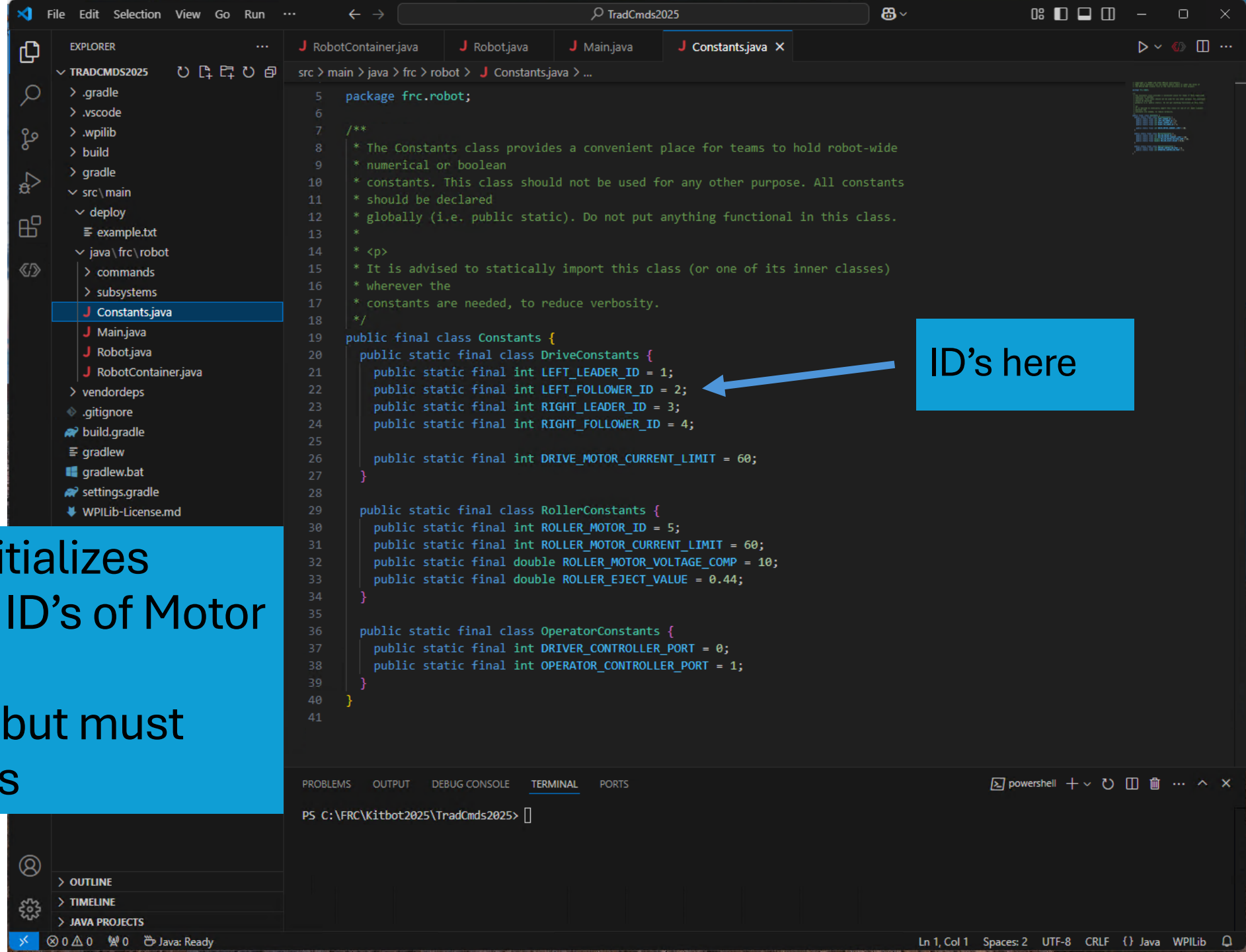
Feel free to say no and see if it still builds.
You can always close it and it will prompt you again if you said no.



First time you open, it builds, be patient



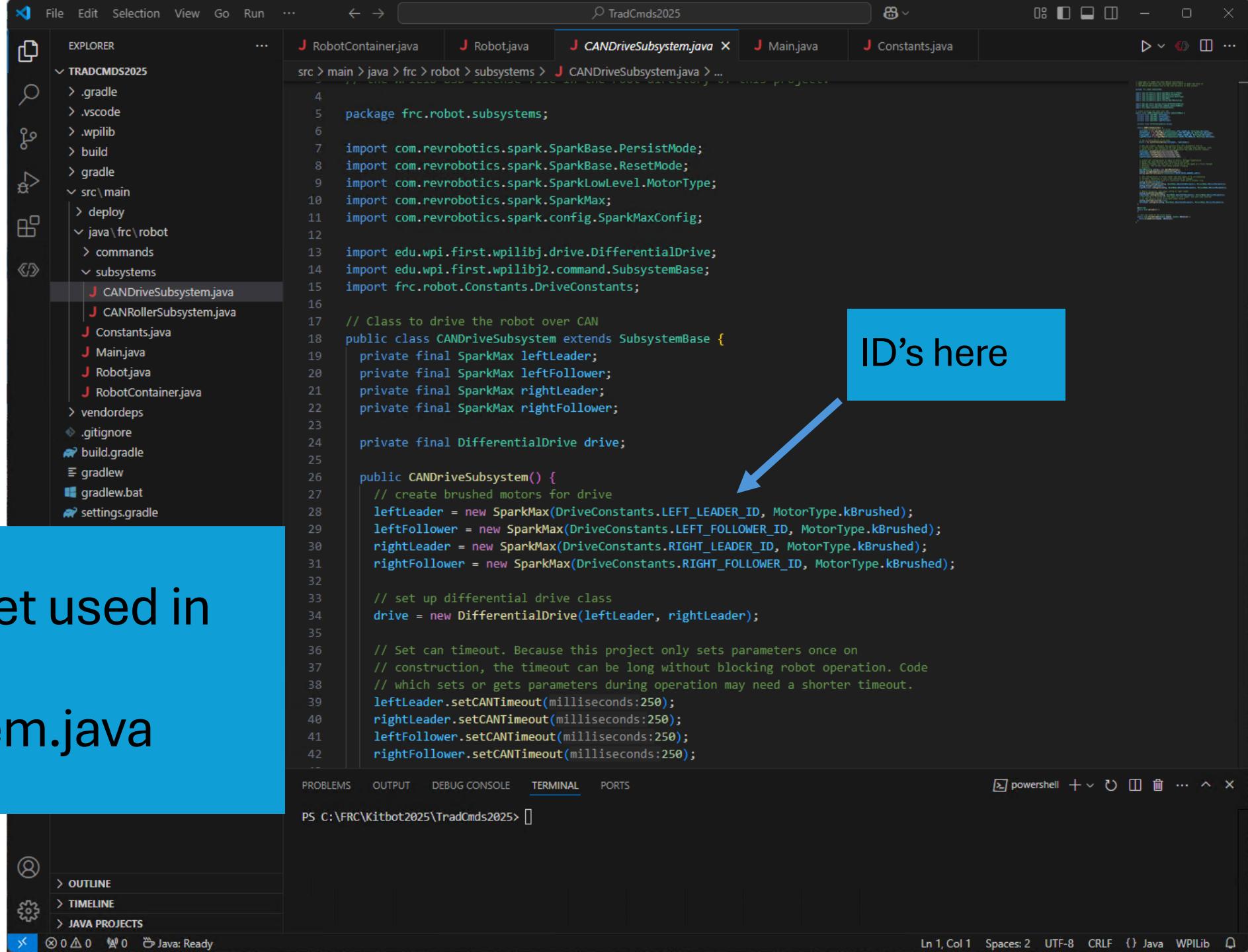
Java code found under src\main\java\frc\robot



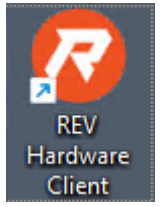
Constants.java file initializes ID #’s related to CAN ID’s of Motor Controllers

Values are up to you, but must agree with Controllers

Notice those ID's get used in subsystems CANDriveSubsystem.java



Set hardware to agree with Software



First time you run REV Client,

You will see list of CAN Devices, And you can ‘blink’ an individual device to find it on the Robot.

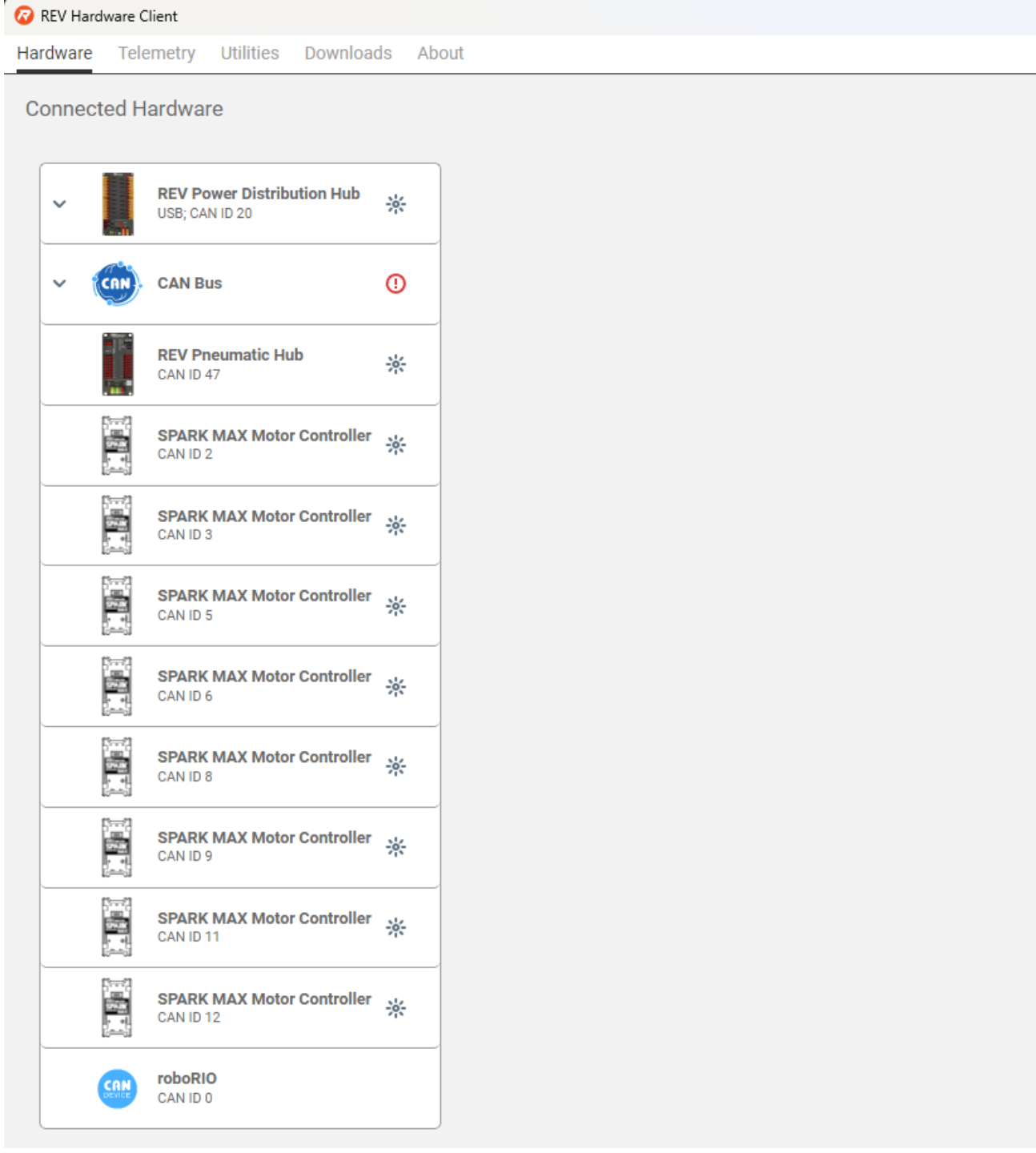
You should make sure all CAN ID’s are unique.

Set the ID’s inside REV client to agree with actual device’s function on the robot.

If they would conflict or be duplicate, move some to higher numbers. Keep in the range 1 to 62 unless you know otherwise.
Try not to use ID 0 or 1 as it may conflict with Power Distribution or Pneumatics controller.

See also section 3.4 of Kitbot Java PDF

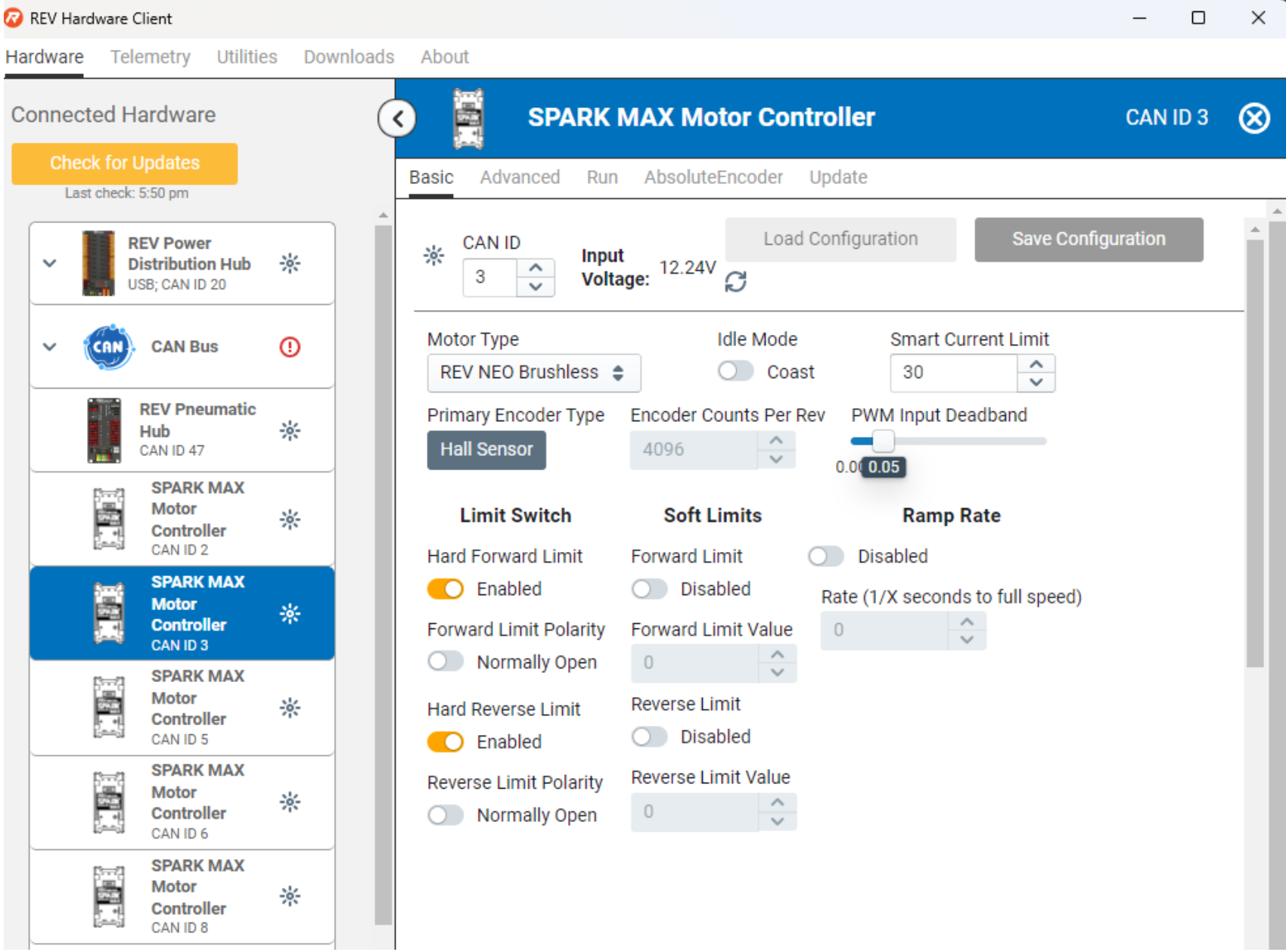
Also recommend Max Current 30A not 60A.



Here is where you actually set the CAN ID

AND there is Update tab to update firmware

You see we set Current limit to 30A, but that can be overridden in software, set both places to 30A.

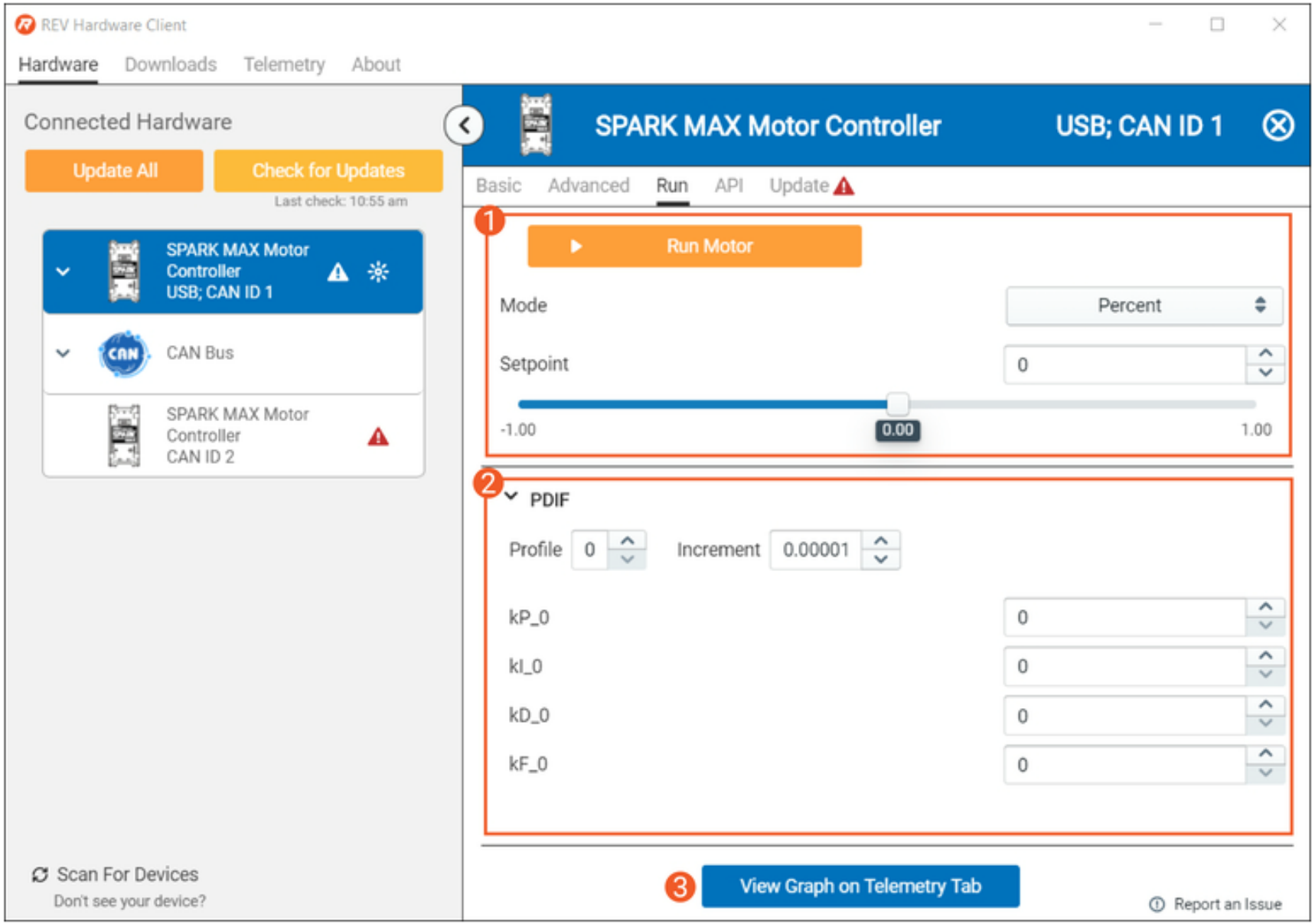


Notice Red Triangle near the Update tab, it wants Firmware update.

When you select a Spark Max you can choose Run and move the slider/click run to have motor spin. BUT keep hands clear and warn people.

Always have robot on blocks off floor in case you are spinning the wrong motor.

Also keep bumpers or poolnoodle on perimeter of robot to guard against running it into objects.



CAN BUS ISSUES:

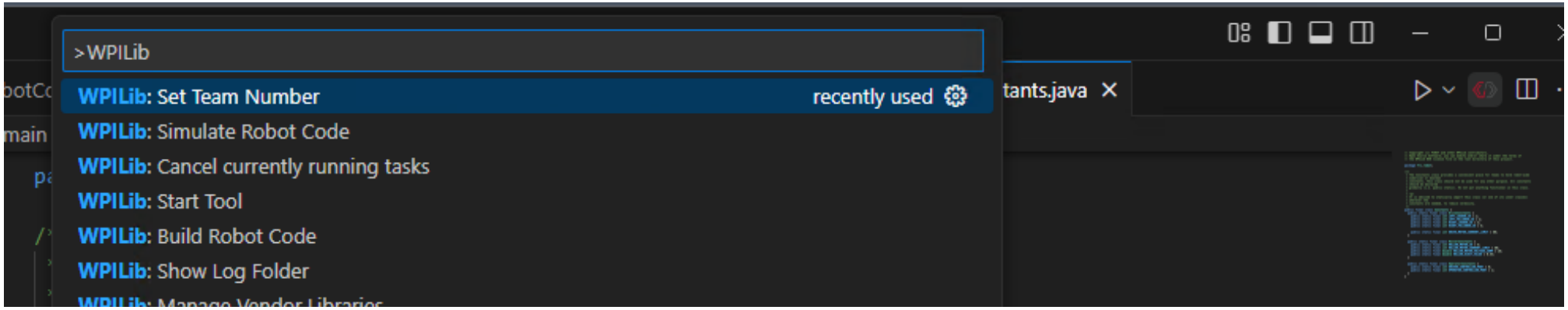
If you don’t see all of the devices, you likely have one or more issues with your CAN bus wiring:

1. Verify that your CAN bus starts with the roboRIO and ends the built in terminator of a Power Distribution Hub or Power Distribution Panel (with the termination set to On using the appropriate jumper or switch).
2. Check that your CAN bus connections all match yellow-yellow and green-green.
3. Check that all CAN wire connections are secure to each other and that the connectors are securely installed in each SPARK Max.
4. If you’re still having trouble, moving the USB connection around to different devices and seeing what each device can “see” on the bus can help pinpoint the location of an issue.
5. You can also disconnect a single SPARK from CAN bus and after a power cycle, use the USB cable to ‘talk’ to just that one SPARK by plugging into it directly.

Build the Code

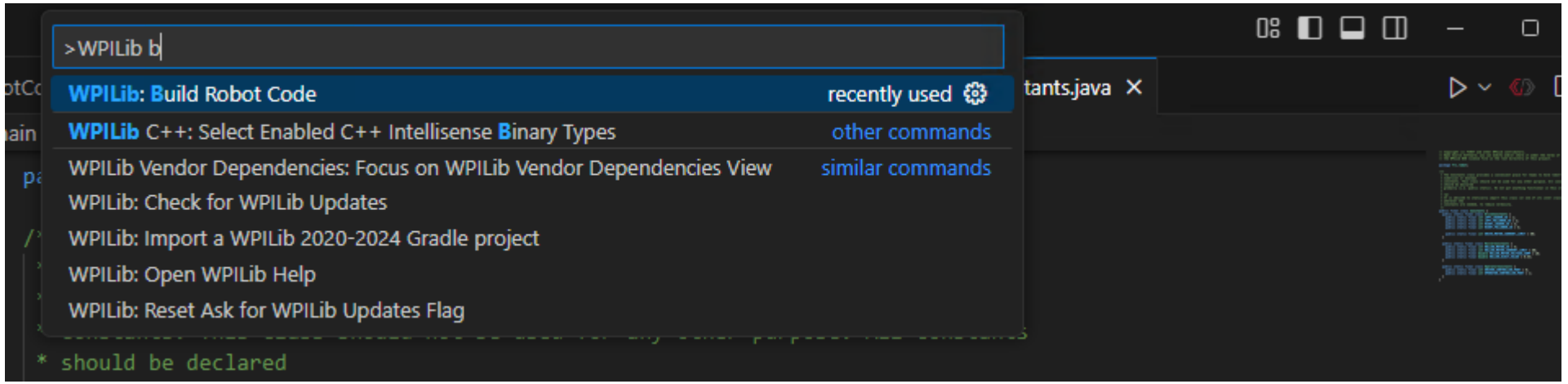
First set the team # (normally you do this when you build a new project), click the Red/Gray <<>> at top right and start typing “Set Team Number”, you should get option to set team to your team#.

Leading Zeros are not allowed



Build Continued 1

Click the Red Gray <<>> again and choose Build Robot Code



Time to Deploy

But first words of safety: WARNING

Make sure area safe to operate the robot.

Have bumpers or pool noodle attached to edges of robot to lessen any impacts.

It will go the wrong way at first.

It will need to probably be slowed down, do that in code.

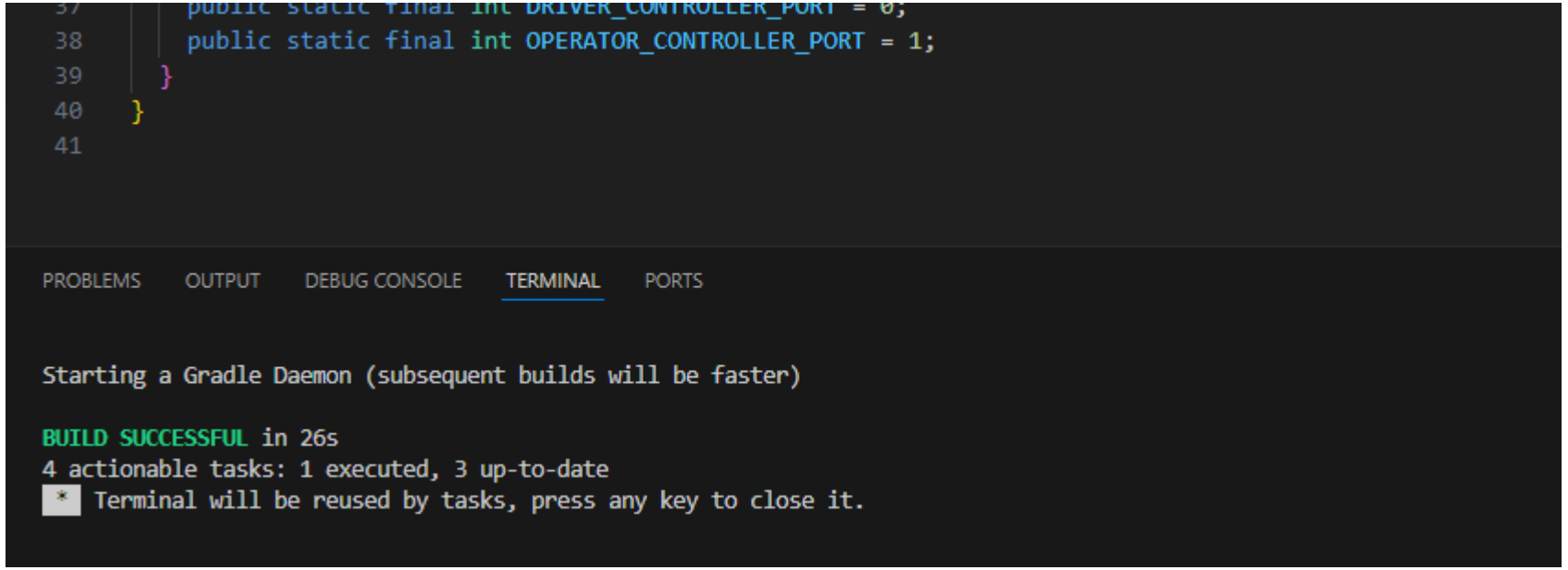
Remember the spacebar is E-stop (Emergency Stop) and the Enter is enable/disable. After an E-stop you may have to power cycle robot.

Deploy Robot Code

Click the Red / Gray <<>> again and choose Deploy to Robot

You should see Build Successful as below after about 30 seconds,

Subsequent builds happen faster.



Game Controllers

Xbox Controllers are the most popular, possibly followed by Logitech F310 type controllers.
The code is set up to use the Xbox controller class.

There are 2 controllers expected, one for Driver and one for the Operator.

Driver will move the robot, and the Operator will operate the roller.

If a team wants to use the Logitech F310 game controllers, they will appear like Xbox controllers to the WPILib software if they are configured in the correct mode. To set up the Logitech controllers, check that the switch on the back of the controller is set the ‘X’ setting. Then when using the controller, make sure the LED next to the Mode button is off; if it is on press the Mode button to toggle it. When the Mode light is on, the controller swaps the function of the left Analog stick and the D-pad.

Those extra buttons can cause robot downtime during competition so train the drive team on them or find a way for them to not be pressed.

More details on Code Layout

Subsystems

For the 2025 KitBot we have 5 motors that make up 2 subsystems, the Drivetrain, and the Roller. The 4 motors in the drivetrain always need to be working together to move the robot around the field and the Roller motor must spin to manipulate Pipes.

Two subsystem files: Drive and Roller in subsystems folder

The general rule of thumb on how to break up subsystems is to think about what actions, or commands, you might have to control the subsystems.

Perhaps you will run the intake while moving the arm or wrist?
Err towards more smaller subsystems; you can always make commands that require multiple subsystems but if you end up wanting separate commands to control a single subsystem at the same time, you'll have to refactor the subsystem to split it up.
See the Kitbot Java PDF as some of this mentioned there.

Imports

```
import com.revrobotics.spark.SparkBase.PersistMode;
import com.revrobotics.spark.SparkBase.ResetMode;
import com.revrobotics.spark.SparkLowLevel.MotorType;
import com.revrobotics.spark.SparkMax;
import com.revrobotics.spark.config.SparkMaxConfig;

import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import edu.wpi.first.wpilibj2.command.SubsystemBase;
import frc.robot.Constants.DriveConstants;
```

Import section of code declares what other classes or packages we need to reference within this code (imports). A common practice is to add imports as you go.

The first group of imports in this subsystem is from the REV Robotics library for the Spark MAXs.

The second group is WPILib classes (one for the type of drivetrain on the robot and one for subsystems) and the DriveConstants section of the Constants file from this project.

Constructors

```java
/** Class to drive the robot over CAN */
public CANDriveSubsystem() {
  // create brushed motors for drive
  leftLeader = new SparkMax(DriveConstants.LEFT_LEADER_ID, MotorType.kBrushed);
  leftFollower = new SparkMax(DriveConstants.LEFT_FOLLOWER_ID, MotorType.kBrushed);
  rightLeader = new SparkMax(DriveConstants.RIGHT_LEADER_ID, MotorType.kBrushed);
  rightFollower = new SparkMax(DriveConstants.RIGHT_FOLLOWER_ID, MotorType.kBrushed);

  // set up differential drive class
  drive = new DifferentialDrive(leftLeader, rightLeader);
```

The next section is the constructor. In the first part of the constructor we initialize any variables contained in the subsystem.

In the case of our drivetrain, we initialize the 4 motor controllers and add one controller for each side into a WPILib DifferentialDrive object which describes the whole drivetrain.

The remainder of the constructor (not pictured) sets up the SparkMAXs for the drivetrain. One motor on each side is set as a follower and directed to follow the leader, then the leader motors are configured with some additional details.

Methods

```
38    public void driveArcade(double xSpeed, double zRotation) {
39      drive.arcadeDrive(xSpeed, zRotation);
40    }
```

The remainder of the subsystem class is methods. Here we define any methods that commands may need to call to get status from or perform actions on the subsystem. For our simple drivetrain, the only method we need is the driveArcade method which simply passes the parameters through to the same method on the DifferentialDrive object.

Actual Drive Control

RobotContainer.java's ConfigureBindings sets up what the Xbox controller does

```
private void configureBindings() {
  // Set the A button to run the "RollerCommand" command with a fixed
  // value ejecting the gamepiece while the button is held

  // before
  operatorController.a()
      .whileTrue(new RollerCommand(() -> RollerConstants.ROLLER_EJECT_VALUE, (

  // Set the default command for the drive subsystem to an instance of the
  // DriveCommand with the values provided by the joystick axes on the driver
  // controller. The Y axis of the controller is inverted so that pushing the
  // stick away from you (a negative value) drives the robot forwards (a posit
  // value). Similarly for the X axis where we need to flip the value so the
  // joystick matches the WPILib convention of counter-clockwise positive
  driveSubsystem.setDefaultCommand(new DriveCommand(
      () -> -driverController.getLeftY() *
          (driverController.getHID().getRightBumperButton() ? 1 : 0.5),
      () -> -driverController.getRightX(),
      driveSubsystem));
```

Command DriveCommand does the driving

Constructor stores parameters into Class variables for later.

The addRequirements will make sure subsystem is available

```java
package frc.robot.commands;

import edu.wpi.first.wpilibj2.command.Command;
import frc.robot.subsystems.CANDriveSubsystem;
import java.util.function.DoubleSupplier;

// Command to drive the robot with joystick inputs
public class DriveCommand extends Command {
  private final DoubleSupplier xSpeed;
  private final DoubleSupplier zRotation;
  private final CANDriveSubsystem driveSubsystem;

  // Constructor. Runs only once when the command is first created.
  public DriveCommand(
      DoubleSupplier xSpeed, DoubleSupplier zRotation, CANDriveSubsystem driveSubsystem) {
    // Save parameters to local variables for use later
    this.xSpeed = xSpeed;
    this.zRotation = zRotation;
    this.driveSubsystem = driveSubsystem;

    // Declare subsystems required by this command. This should include any
    // subsystem this command sets and output of
    addRequirements(this.driveSubsystem);
  }
```

Command DriveCommand does the driving

In the execute() method, we call the driveArcade method from the drive subsystem which tells the motors to drive such that the robot moves according to the joystick inputs.

```
38    @Override
39    public void initialize() {}
40
41    @Override
42    public void execute() {
43      driveSubsystem.driveArcade(xSpeed.getAsDouble(), zRotation.getAsDouble());
44    }
45
46    @Override
47    public void end(boolean isInterrupted) {}
48
49    @Override
50    public boolean isFinished() {
51      return false;
52    }
```

Autonomous – drive forward 1 second

Imports and Constructor

Class sets up two variables and a method

```java
package frc.robot.commands;

import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj2.command.Command;
import frc.robot.subsystems.CANDriveSubsystem;

// Command to run the robot at 1/2 power for 1 second in autonomous
public class AutoCommand extends Command {
  CANDriveSubsystem driveSubsystem;
  private Timer timer;
  private double seconds = 1.0;

  // Constructor. Runs only once when the command is first created.
  public AutoCommand(CANDriveSubsystem driveSubsystem) {
    // Save parameter for use later and initialize timer object.
    this.driveSubsystem = driveSubsystem;
    timer = new Timer();

    // Declare subsystems required by this command. This should include any
    // subsystem this command sets and output of
    addRequirements(driveSubsystem);
  }
```

AutoCommand Methods

- Initialize: Start the timer.
- Execute: Sets the drive motors to drive forward.
- End: Stop the timer and stop the drive subsystem.
- IsFinished: Returns true when the timer exceeds 1 second.

```java
// Runs each time the command is scheduled. For this command, we handle starting
// the timer.
@Override
public void initialize() {
  // start timer, uses restart to clear the timer as well in case this command has
  // already been run before
  timer.restart();
}

// Runs every cycle while the command is scheduled (~50 times per second)
@Override
public void execute() {
  // drive at 1/2 speed
  driveSubsystem.driveArcade(xSpeed:0.5, zRotation:0.0);
}

// Runs each time the command ends via isFinished or being interrupted.
@Override
public void end(boolean isInterrupted) {
  // stop drive motors
  driveSubsystem.driveArcade(xSpeed:0.0, zRotation:0.0);
}

// Runs every cycle while the command is scheduled to check if the command is
// finished
@Override
public boolean isFinished() {
  // check if timer exceeds seconds, when it has this will return true indicating
  // this command is finished
  return timer.get() >= seconds;
```

Enable/Disable AutoCommand

RobotContainer.java

```java
/**
 * The container for the robot. Contains subsystems, OI devices, and commands.
 */
public RobotContainer() {
  // Set up command bindings
  configureBindings();

  // Set the options to show up in the Dashboard for selecting auto modes. If you
  // add additional auto modes you can add additional lines here with
  // autoChooser.addOption
  autoChooser.setDefaultOption(name:"Autonomous", new AutoCommand(driveSubsystem));
}
```

autoChooser is a pulldown that can appear on your drivestation's dashboard such as Shuffleboard.

Read about Dashboards

More on Autonomy

Refer to the Kitbot JAVA PDF to learn more about Autocommands

And also the Command Based Programming section of Wpilib docs.

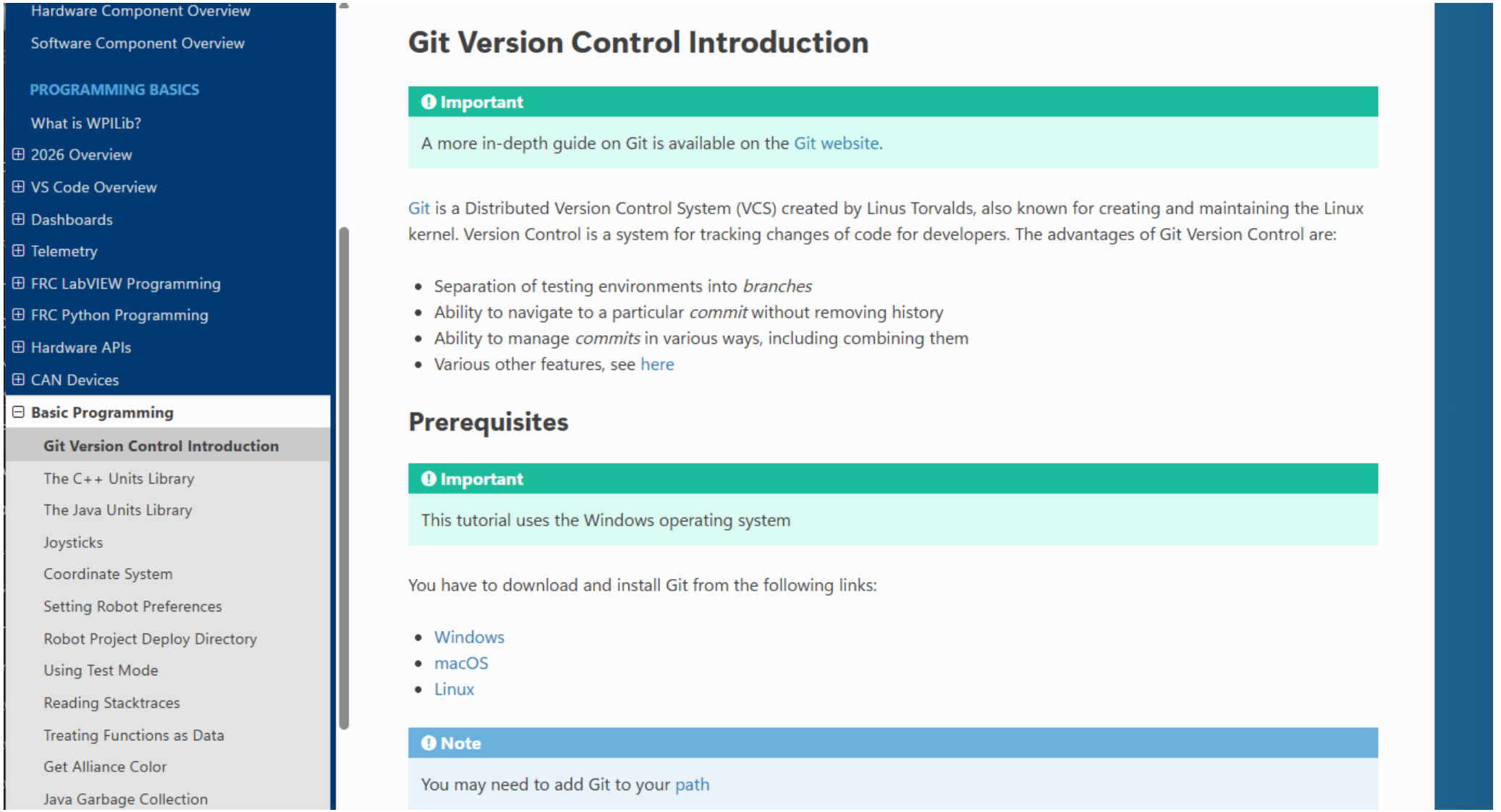
Why Git ?

For Revision Control

- Once you add and commit, code is on github forever
- Allows you to revert back before that accidental wrong edit
- Allows ‘branching’ for adding a feature while main baseline still available
- Allows other team to look at your code to assist
- Allows you to share your code to assist other teams

Install GIT for version Control

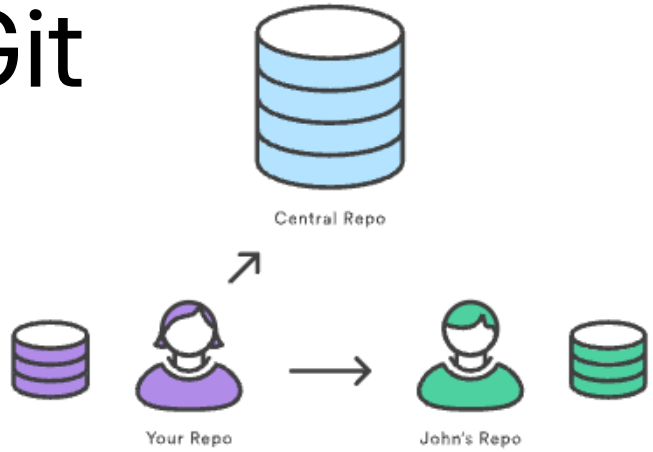
https://docs.wpilib.org/en/stable/docs/software/basic-programming/git-getting-started.html
Lots of info on the FRC page, follow those instructions



Git Vocabulary

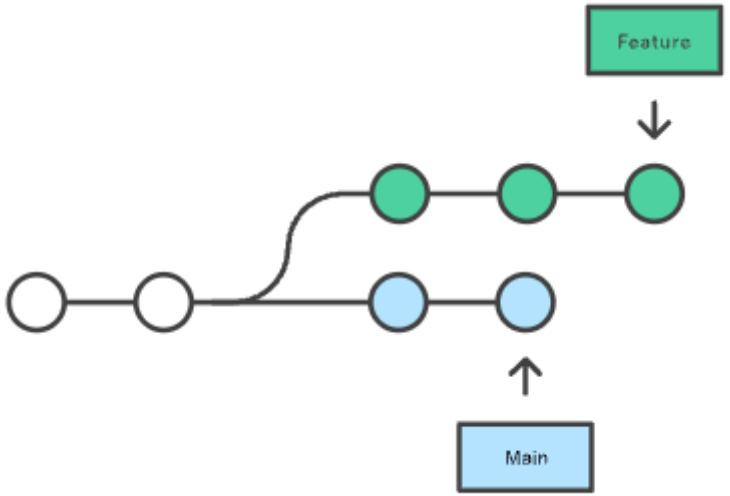
- Git concepts and commands:
- **Repository:** the data structure of your code, including a .git folder in the root directory
- **Clone:** retrieve a local copy of a repository to modify
- **Fork:** duplicate a pre-existing repository to modify, and to compare against the original
- **Branch:** a means of grouping a set of commits. Each branch has a unique history. This is primarily used for separating development and stable branches
- **Add:** Queues up some changes
- **Commit:** a particular saved state of the repository, which includes all files and additions
- **Push:** update the remote repository with your local changes
- **Pull:** update your local repository with the remote changes
- **Merge:** combine various changes from different branches/commits/forks into a single history
- VS Code extension so it’s built into VS code

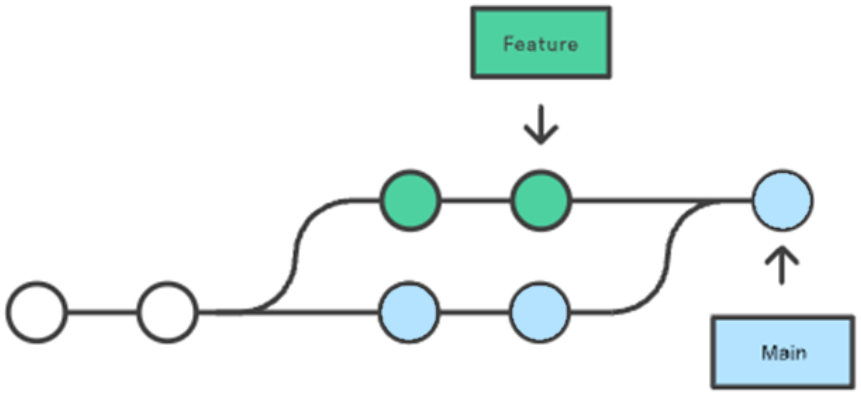
More Git



Tutorials on Git

- https://git-scm.com/learn
- https://www.atlassian.com/git/tutorials
- https://docs.github.com/en/get-started/start-your-journey/hello-world





Vision – very helpful for automous period

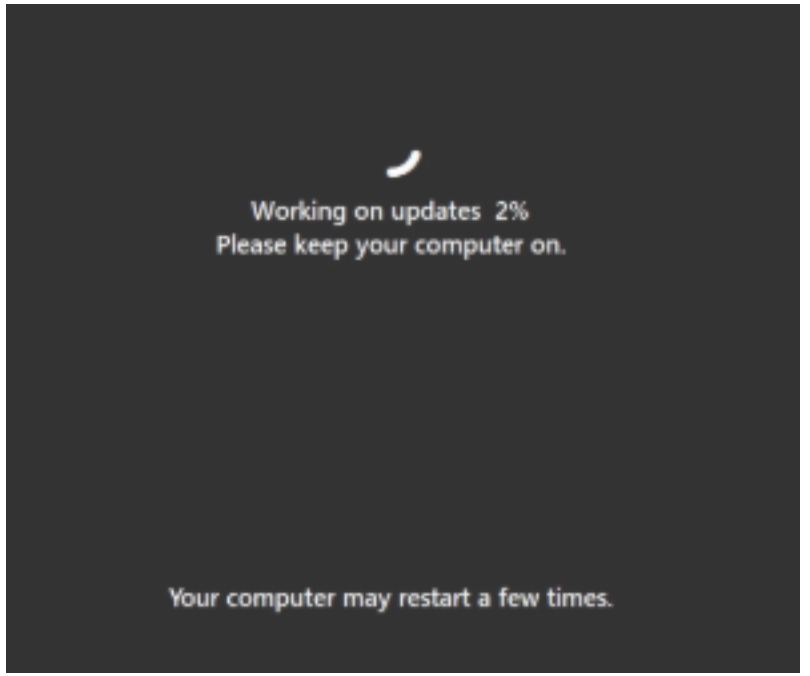
- Photonvision runs on Raspberry Pi with webcam to help locate your robot on field https://photonvision.org/
- Limelight is all in one solution for locating robot on field.
- https://docs.wpilib.org/en/stable/docs/software/vision-processing/apriltag/index.html

- Once your robot knows where it is, autonomy becomes easier
- https://www.youtube.com/@0ToAuto

Learn Java and more

- https://learn.microsoft.com/en-us/shows/java-for-beginners/
- Codeacademy
- Coders need to know the hardware they control: https://docs.wpilib.org/en/stable/docs/controls-overviews/control-system-hardware.html

Competition gotchas

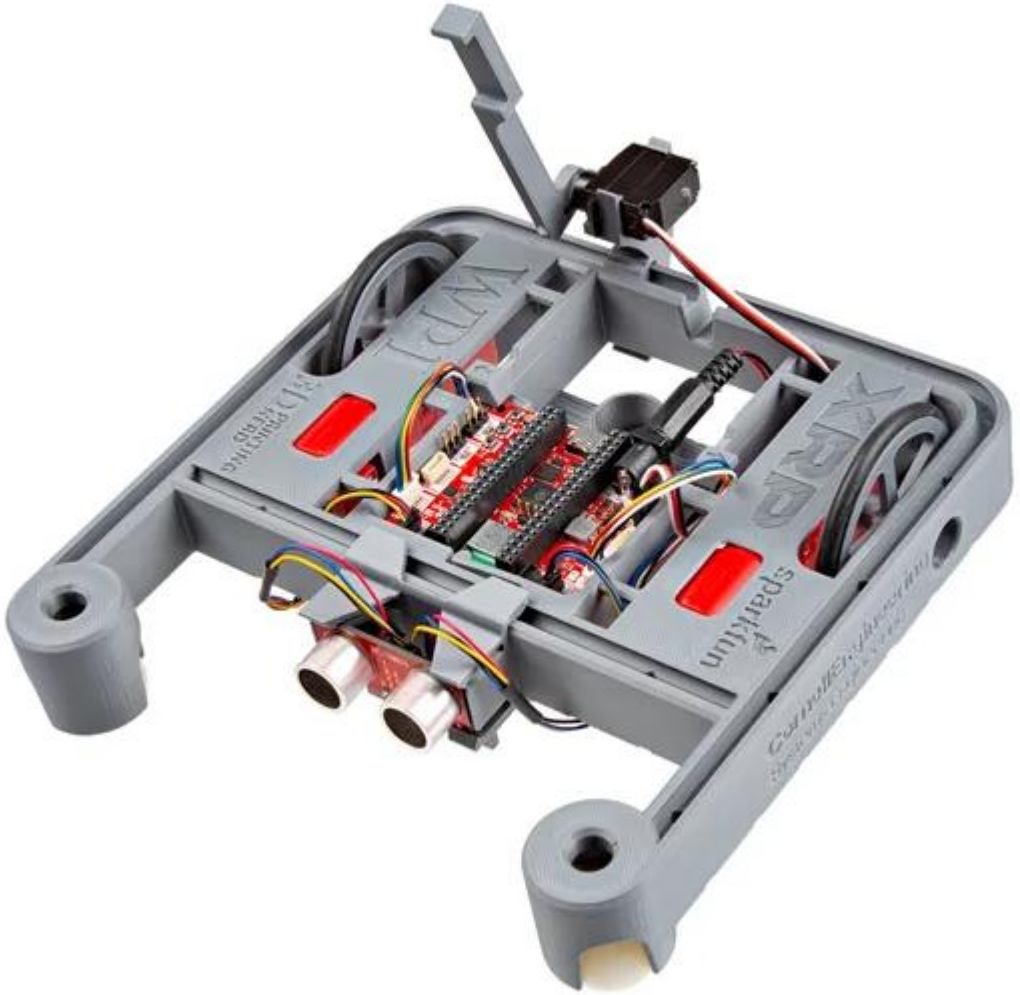


BEFORE Leaving for competition:

- Make sure firmware updated for all motor controllers (use REV client or CTR phoenix tuner)
- Make sure your code compiled with latest WPIlib
- Make sure Roborio has latest firmware and Image
- Visit here https://github.com/wpilibsuite/allwpilib/releases
- **Update Laptop software, but once updated, turn off updates for duration of competition**, as that ‘blank screen updating’ during your scheduled competition does happen.

XRP robot is tiny robot using same code

- Uses same WPIlib code
- Great intro to code for new coders
- Helps understand game controller and motors
- https://docs.wpilib.org/en/stable/docs/xrp-robot/index.html



Other References

- Team 254 Slides https://drive.google.com/drive/folders/1tZ6xdn-xzFpcAzW3VB5ttzQwuhUmvPJB?usp=sharing
- Kitbot 2025 https://firstfrc.blob.core.windows.net/frc2025/KitBot/KitBotBuildInstructions.pdf
- And the Kitbot Java PDF from the kitbot java code zipfile
- https://frcteam3255.github.io/FRC-Java-Tutorial/
- FirstSC slack for updates on any South Carolina specific coding collaborations

More

There is so much at the docs.wpilib.org site