How to program and drive a MiniFRC robot with the NoU-WPILib software tool

The NoU-WPILib software tool lets you easily program and run a MiniFRC robot using the same programming environment as regular FRC robots. With this tool, you will program your robot in the WPILib Integrated Development Environment (IDE) and drive it using the WPILib Robot Simulator. This tool specifically works for robots built with an ESP32 microcontroller and NoU motor shield from Alfredo Systems.

Sections:

When should I use the NoU-WPILib software tool?

About this guide

Required setup

Creating your robot program

Customizing your code

Example code

Driving your robot

Setting up Robot Simulator

When should I use the NoU-WPILib software tool?

You'll probably want to use this tool if you're signed up for MiniFRC and

- You've never used the Arduino IDE and have no time to learn it, despite its many applications to small-scale robotics and hobby electronics
- You already know how to program a robot in WPILib
- You don't know how to program a robot in WPILib, but you want to learn
- You have a Mac. Sadly, the AlfredoConnect drive station software is only made to be compatible with Windows. Using it on a Mac without Windows emulation requires advanced workarounds, a lot of patience, and possibly divine intervention—although it is certainly possible. Sometimes.

Note: The NoU-WPILib software tool currently only supports Java programming.

About this guide

This guide provides basic instructions for how to get started programming your MiniFRC robot, even if you have no prior experience with WPILib. If you want a more detailed tutorial for WPILib programming, see their "Zero to Robot" tutorial.

Required setup

1. Build your robot. If you need help, refer to the NoU2 Robot Build Tutorial.

The code in this tutorial is for a robot that has left motors wired together into NoU2 port 1 and right motors wired together into NoU2 port 2.

2. Install the NoU-WPILib software tool and all required software.

If you've followed all these instructions, this is the software you should have now:

- Arduino IDE, with Alfredo libraries installed (Instructions **here**)
- NoU-WPILib, with proxy server set up (Instructions **here**)
- WPILib (Instructions **here**)
- VSCode for WPILib (Included in the WPILib installation).

Once you have a robot and all the software set up, it's time to start programming!

Creating your robot program

The first thing you need to do is create a new project for your robot in VSCode and set it up for NoU-WPILib integration:

1. Open VSCode
2. Generate a new project and select "TimedRobotSkeleton (Advanced)" as your base
3. Add the required lines of code to the build.gradle file, as described in furseiry's "Setting up WPILib" instructions. This code sets up the Robot Simulator for NoU-WPILib integration.
4. Install the NoU-WPILib vendor library, as described in furseiry's "Programming Setup" instructions.

Customizing your code

1. In the sidebar, expand "src"; this is your source code.
2. Select the Robot.java file. This is the only file you need to mess with.

   Tip: For extra help, keep the WPILib programming tutorial handy. The following instructions are based on the programming tutorial for "TimedRobotSkeleton (Advanced)", which has skeleton methods for each of the timed periods in an FRC match (autonomous and teleoperated) and is the easiest to adapt for MiniFRC.
3. Below the existing "package frc.robot" line, add the following lines of code to import needed libraries:

```
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
```

```
import edu.wpi.first.wpilibj.XboxController;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
```

4. Import NoU libraries for motors, servos, and General Purpose In/Out (GPIO) pins:

```
import com.NoULib.lib.NoUMotor; // import for all robots
import com.NoULib.lib.NoUServo; // import if using servos
import com.NoULib.lib.NoUGPIO;  // import if using GPIO pins
```

5. Define variables in the existing Robot class:

```
public class Robot extends TimedRobot {
```

You'll use the existing WPILib classes for Joystick and Timer. You're going to use the NoUMotor and NoUServo classes to define all your motors and servos.

Here is example code for variables to set up a differential drive for left motors and right motors:

```
private final NoUMotor m_leftDrive = new NoUMotor(1);
private final NoUMotor m_rightDrive = new NoUMotor(2);
private final DifferentialDrive m_robotDrive = new DifferentialDrive(m_leftDrive,
    m_rightDrive);
private final Joystick m_controller = new Joystick(0);
private final Timer m_timer = new Timer();
```

Notes: This code is for a robot that has left motors wired together into NoU2 port 1, and right motors wired together into NoU2 port 2. If your robot has each motor in its own port, you'll need to use the WPILib MotorControllerGroup class, as documented in "Using the WPILib Classes to Drive your Robot".

6. Fill in methods to control your robot. There are a bunch of public methods already listed, but these are empty (i.e. skeletons); you'll need to add code for the ones you want to use.

There are methods for:

- the robot in general
- the autonomous time period
- the teleoperated time period

For each of these there are initial methods (run once) and periodic methods (run the whole time).

Here's an example of a method for the autonomous period that drives the robot forward for two seconds at half speed:

```
@Override
 public void autonomousPeriodic() {
   // Drive for 2 seconds
   if (m_timer.get() < 2.0) {
    // Drive forwards half speed, make sure to turn input squaring off
    m_robotDrive.arcadeDrive(1, 0.0, false);
  } else {
    m_robotDrive.stopMotor(); // stop robot
  }
 }
```

Here's an example of basic code just to move your robot, which you'll put in teleopPeriodic():

```
@Override
 public void teleopPeriodic() {
   m_robotDrive.arcadeDrive(m_controller.getRawAxis(1),
m_controller.getRawAxis(0));
   //if robot is not behaving, switch axis labels or reconfigure your joystick
 }
```

You will not need *test*, *disabled* or *simulation* methods for a basic MiniFRC robot, but you can just leave them there and empty.

7. Save your code!

After you finish writing your code, you don't need to Deploy. We'll be doing everything in the Robot Simulator.

Example code

Here's an example program for a basic drive train controlled by a joystick:

```
// Copyright (c) FIRST and other WPILib contributors.
```

```
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

package frc.robot;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.XboxController;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import com.NoULib.lib.NoUMotor;

/**
 * The VM is configured to automatically run this class, and to call the functions
corresponding to
 * each mode, as described in the TimedRobot documentation. If you change the
name of this class or
 * the package after creating this project, you must also update the build.gradle file
in the
 * project.
 */
public class Robot extends TimedRobot {
  /**
   * This function is run when the robot is first started up and should be used for any
   * initialization code.
   */
```

```
  private final NoUMotor m_leftDrive = new NoUMotor(1);
  private final NoUMotor m_rightDrive = new NoUMotor(2);
  private final DifferentialDrive m_robotDrive = new DifferentialDrive(m_leftDrive,
m_rightDrive);
  private final Joystick m_controller = new Joystick(0);
  private final Timer m_timer = new Timer();

  @Override
  public void robotInit() {
    //m_rightDrive.setInverted(true);
    m_leftDrive.setInverted(true);
  }

  @Override
  public void robotPeriodic() {} //you very rarely need to put anything here

  @Override
  public void autonomousInit() {
    m_timer.restart(); //restarts count to zero
  }

  @Override
  public void autonomousPeriodic() {
     // Drive for 2 seconds
     if (m_timer.get() < 2.0) {
      // Drive forwards half speed, make sure to turn input squaring off
      m_robotDrive.arcadeDrive(1, 0.0, false);
```

```
    } else {
      m_robotDrive.stopMotor(); // stop robot
    }
  }

  @Override
  public void teleopInit() {}

  @Override
  public void teleopPeriodic() {
    m_robotDrive.arcadeDrive(m_controller.getRawAxis(1),
m_controller.getRawAxis(0));
    //if robot is not behaving, switch axis labels maybe or reconfigure your joystick
  }
}
```

Driving your robot

Always perform these steps in this order:

1. Power on the robot
2. In Terminal (Mac) or Command Prompt (Windows), start your proxy server with your robot name
3. In VSCode, start the robot simulator (shortcut F5):
    a. Open the Command Palette and select WPILib: Simulate Robot Code.
    b. When prompted, run with Sim GUI and Websocket Client extensions.

Setting up Robot Simulator

The first time you drive your robot, you'll need to set up controller or keyboard mappings in Robot Simulator.

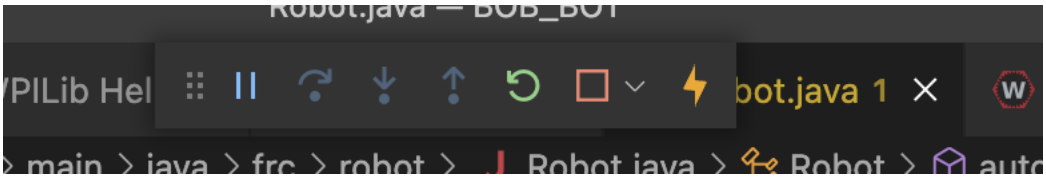
1. If you have a Joystick or XBox Controller connected, the mappings are set up automatically to reflect what you set up in your code
2. Alternatively, set up keyboard mappings:
    a. Use the keyboard by dragging keyboard 0 to Joystick 0.

b. Go to the DS menu→Keyboard 0 settings. If the robot doesn't do what you expect (e.g. goes forward when you press the key for back), the easiest fix is to swap the keys for the affected axis.

c. If both sides of your robot aren't going in the same direction, invert one of the sides in the code:

```
public void robotInit() {
   //m_rightDrive.setInverted(true);
   m_leftDrive.setInverted(true);
 }
```

Every time you make changes to your code, restart the Robot Simulator in VSCode (the green circular arrow):



3. Save your Workspace in your WPLlib project folder, and save after any changes.