Simulink integration into First Robotics Competition WPILib VS Code

Dunlap Eagles Robotics Team (Dunlap, Illinois, USA)
First Robotics Team #2040
Randy Anderson (mentor) and Janelyn Anderson (student)
December 2023

This tutorial will bring you from zero to working software on an FRC RoboRio by using the WPILib software architecture, Matlab/Simulink for the algorithm portion between the input ports and output ports, and FRC Game Tools (Driver Station, Shuffleboard). Through an example you will simulate an algorithm in Simulink, generate Simulink code, modify WPILib code, deploy integrated WPILib/Simulink software to the RoboRio, tune parameters, and measure signals using Shuffleboard.

Advantages of using Matlab/Simulink

- Learn how to use an industry standard graphical coding and simulation tool.
- Students without much coding experience can pick this up quicker than object-oriented coding languages.
- Simulate your algorithms controlling the robot in a closed loop virtual environment. This allows you to test algorithm content (small unit tests up to an entire robot) before the physical robot is available. Multiple students can develop/simulate code for different parts of the robot without tying up a physical robot. This can greatly reduce the time it takes to get working software.
- The code generation process from models to C-code is robust. Once the model updates and simulates with the desired behavior then there aren't any errors when translating to C-code.

Disadvantages of using Matlab/Simulink (for FRC)

- Instead of using many of the WPILib modules you will need to reverse engineer some of what is in WPILib (or invent it for yourself) and code it in Simulink. In the long run this might not be a disadvantage because your team will have a deeper understanding of the algorithms and therefore know how to change/optimize them for improved performance.
- You need to use C++ for the RoboRio software architecture. Matlab/Simulink is not able to generate code for Java.
- You must use timed-robot to execute the Simulink code at a periodic rate. This doesn't work at all for command-based robot.

Software Installation

1. Install FRC Game Tools (this document is based on 2023 version)
2. Install WPILib (this document is based on 2023 version)
3. Install Matlab (this document is based on Matlab 2023b)
    a. One mentor from the team needs to visit this site and fill out the "Request software".
    b. The mentor will receive an email from The Mathworks with a license number and setup instructions.

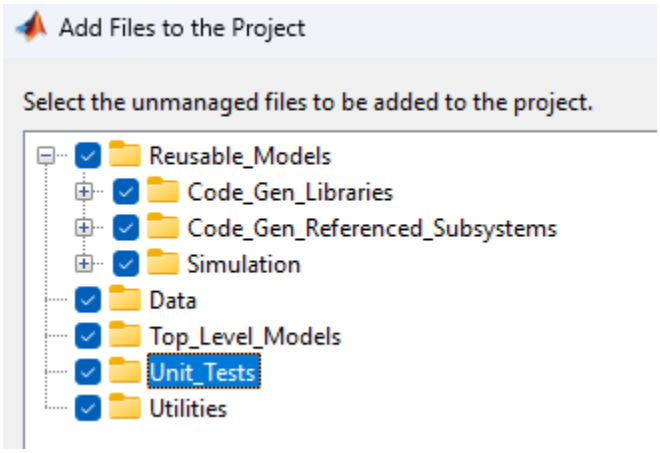
Github Repository

- https://github.com/DERT-2040/WPILib-Simulink_Tutorial.git

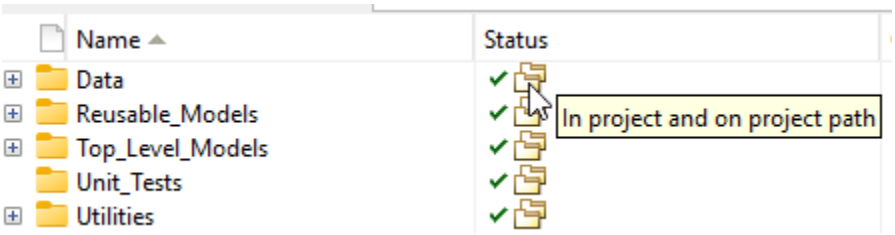
Procedure

1. Create a new WPILib project
    a. Open WPILib VS Code
    b. Ctrl+Shift+P (View > Command Palette …)
    c. Choose from the dropdown “WPILib: Create a new project”
    d. Select a project type (Example or Template): “Template”
        i. Note: Choosing “Template” removes extra code that we would end up deleting anyway.
    e. Select a language: “cpp”
        i. Note: Simulink cannot generate code for Java, so we must choose C++.
    f. Select a project base: “Timed Skeleton (Advanced)”
        i. Note: “Timed” will run the code on fixed time step whereas Command executes upon request.
        ii. Note: Choosing “(Advanced)” removes extra code that we would end up deleting anyway.
    g. Select a folder on your hard drive where a new project folder will be created.
        i. “Select a new project folder” to select the folder.
    h. Enter a Project Name (keep the checkbox checked)
    i. Enter the Team Number
    j. Click “Generate Project”
2. Create a new “simulink” folder at the project top level (same level as “src”) with subfolders.
    a. simulink
        i. Root folder for a Matlab project to be created in the next step.
    b. simulink\Data
        i. M-files containing parameters used for code generation and simulation.
    c. simulink\Reusable_Models
        i. model files that are reusable (libraries and referenced subsystems)
    d. simulink\Reusable_Models\Code_Gen_Libraries
        i. Simulink library model files used for code generation.
    e. simulink\Reusable_Models\Code_Gen_Referenced_Subsystems
        i. Simulink reference subsystem model files used for code generation.
    f. simulink\Reusable_Models\Simulation
        i. Simulink library files use used in simulation models, for example, representing the plant.
    g. simulink\Top_Level_Models
        i. Simulink top level code generation and simulation model files.
    h. simulink\Unit_Tests
        i. Simulink model files used for unit testing, for example, testing code gen libraries or referenced subsystems.
    i. simulink\Utilities
        i. Miscellaneous files that are used in the code generation process, plotting simulation results, etc.

3. Create a Matlab project in the new “simulink” folder
    a. Open Matlab
    b. Change the working directory to the “simulink” folder
        i. 

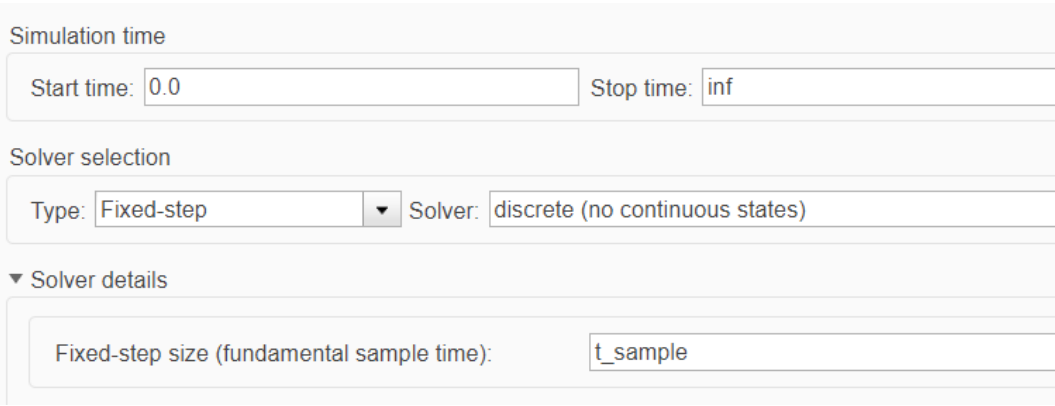
    c. Create a new Matlab project
        i. HOME (ribbon) > New > Project > Blank Project
            - Project name: simulink
            - Browse to select the new “simulink” folder created above
            - Click on “Create”
        ii. When creating the project a “resources” folder is automatically created. This is only used by Matlab and contains files to manage the project folder structure.
        iii. Close the “Welcome to your project” window that popped up.
        iv. Add folders to the project
            - PROJECT (ribbon) > TOOLS > PROJECT FILES > Add Files
                a. 

            - Another method is to drag the folders from the left pane “Current Folder” into the project window in the center of the screen.
        v. Highlight all of the project folders, right click, add to project path with subfolders. It should the following status symbol for each.



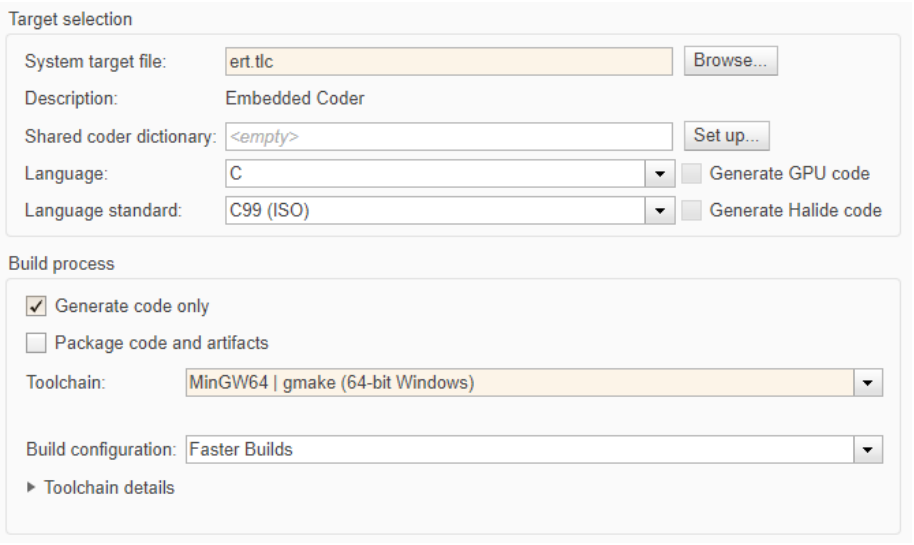
4. Copy files from the repository to the desired locations.
    a. Folders and files:
        - \simulink\Data
            - Code_Gen_Model_data.m
        - \simulink\Reusable_Models\Code_Gen_Referenced_Subsystems
            - Robot_sub.slx
        - \simulink\Top_Level_Models
            - Simulation_Model.slx
        - \simulink\Utilities
            - Build_Extern.m
            - Build_Intern.m
            - generate_controller_code.m
    b. Add each of the files to the Matlab project using one of the methods described in the previous step for folders.
5. Simulate the algorithm.
    a. Open “Simulation_Model.slx” which already includes a referenced subsystem link to “Robot_sub.slx”.
    b. Run the “Code_Gen_Model_data.m” file to load data into the workspace.
    c. Explore the model and notice the following:
        i. Dashboard elements for Enable/Disable and operating mode when enabled which mimic the FRC Driver Station.
        ii. Slider bar for the amplitude of a waveform.
        iii. Inside the subsystem there are four game states and each calculates a different waveform.
        iv. There are tunable parameters for “Frequency” and “Offset” which are set in the “Code_Gen_Model_data.m” file along with the discrete sample time.
        v. The subsystem for Autonomous operation contains a signal test point named “sine_wave_raw”,
    d. Spend some time running the simulation.
        i. While the simulation is running change the Enable/Disable slider.
        ii. Change the operating mode.
        iii. Change the amplitude slider.
        iv. Stop the simulation and open the “Code_Gen_Model_data.m” file where you can change the “Frequency” and “Offset” values. Run the script file to load the new parameter values into the workspace. Run the simulation with these changes in place.

6. Create a new Simulink model and set it up for code generation.
    a. Starting from the Matlab window create the Simulink model
        i. HOME (ribbon) > New > Simulink Model
            - Blank Model
        ii. Save the new model into the “Top_Level_Models” folder with the .slx extension type. In this example walkthrough, the model name is “Code_Gen_Model.slx” which will show up in the hand code examples.
        iii. Add this file to the project (same method as in previous steps).
    b. From the Simulink model (not the Matlab window) go to MODELING (ribbon) > Model Settings (click on the gear, not the drop down menu).
        i. 

        ii. Solver (match these settings)
            - 

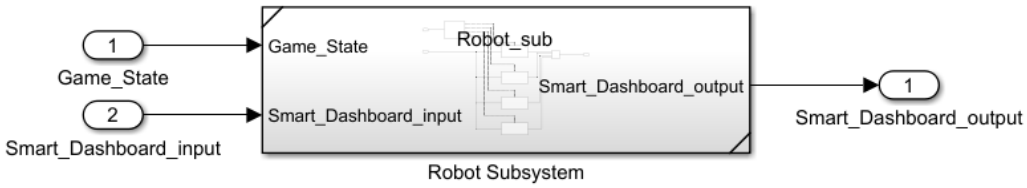
            - Note: The fixed step size is set to parameter “t_sample” which is the update rate of the model and should match the periodic update rate of the WPILib VS Code project. The WPILib VS Code default is set to 0.020 seconds and therefore the Matlab workspace needs to be populated with “t_sample = 0.020”. This parameter will be added to an m-file located in the “Data” folder in a later step.

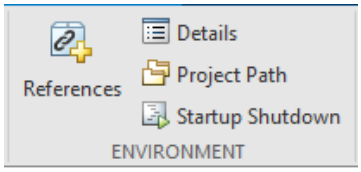
iii. Code Generation (match these settings)



-
- Note: Choosing Language “C” (instead of “C++”) will allow us to access internal model data externally in WPILib VS Code. For example, tuning parameters and test points (measurement variables). The WPILIb VS Code C++ code can include Simulink generated C code. As mentioned earlier, Simulink cannot generate code for Java.

c. Add the referenced subsystem, input ports, and output ports exactly as shown in the image below.
   i. From the Simulink model: MODELING (ribbon) > COMPONENT > INSERT COMPONENT > Subsystem Reference
   ii. Using the dialog browse to locate the “Robot_sub.slx” reference subsystem file. This is the same subsystem used in the simulation model.
   iii. Add input ports (double click the white space and type “Input”) and output port (double click the white space and type “output”). Name the ports as shown below and make the connections.



7. Set the Simulink project code generation folder
    a. Open the Matlab window (not the Simulink model)
    b. PROJECT (ribbon) > Details
        i. 

    c. In the Project Details window find the “Generated Simulink Files” section (scroll down) and set “Code generation folder” to “[project root]/../src/main”.
        i. 

8. Add shortcuts for code generate workflow.
    a. From the Matlab project highlight “Build_Extern.m”.
    b. Find “PROJECT SHORTCUTS” on the ribbon and click “New Shortcut”. Click OK on the dialog that pops up.
    c. Repeat for “Build_Intern.m”.
    d. The “Extern” option makes all Simulink parameters tunable from Smart Dashboard or Shuffleboard unless they are included in the excepts list found in “Code_Gen_Model_data.m”.
    e. The “Intern” option keeps all Simulink parameters local to the generated code which converts them to numeric values and allows for optimized generated code.
    f. When executed during the code generation process these scripts auto generate files “SimulinkSmartDashboard.h” and “SimulinkSmartDashboard.cpp” into the “include” and “cpp” folders.
9. Generate C code from the Simulink code generation model.
    a. Find the PROJECT SHORTCUTS ribbon and click on the “Build_Extern.m” shortcut.
    b. Check the command window to ensure the build was successful.

10. When using Git for version control add the following to ".gitignore" in the Git repository.

```
# BELOW COPIED FROM SIMULINK PROJECT

# Autosave files
*.asv
*.m~
*.autosave
*.slx.r*
*.mdl.r*

# Derived content-obscured files
*.p

# Compiled MEX files
*.mex*

# Packaged app and toolbox files
*.mlappinstall
*.mltbx

# Deployable archives
*.ctf

# Generated helpsearch folders
helpsearch*/

# Code generation folders
slprj/
sccprj/
codegen/

# Cache files
*.slxc

# Cloud based storage dotfile
.MATLABDriveTag
```

11. Modify the WPILib files to pull in Simulink generated code.
    a. Open the WPILib VS Code project folder.
    b. Make changes to the “build.gradle” file.
        i. Change the source and include directories to be one level higher so they include in the Simulink generated code.
        ii. Add .c and .h files since Simulink will be generating these file types (C, not C++).
        iii. Exclude “ert_main.c” since the WPILib project already includes a main function. We can’t have two main functions.
        iv. Below is a diff screen capture:
        v.

```
diff --git a/build.gradle b/build.gradle
index 3e9b217..ba622b5 100644
--- a/build.gradle
+++ b/build.gradle
@@ -56,11 +56,13 @@ model {
56 56
57 57             sources.cpp {
58 58                 source {
59    -                 srcDir 'src/main/cpp'
60    -                 include '**/*.cpp', '**/*.cc'
   59 +                 srcDir 'src/main'
   60 +                 include '**/*.cpp', '**/*.cc', '**/*.c'
   61 +                 exclude '**/ert_main.c'
61 62                  }
62 63                  exportedHeaders {
63    -                 srcDir 'src/main/include'
   64 +                 srcDir 'src/main'
   65 +                 include '**/*.h'
64 66                  }
65 67              }
66 68
```

    c. Important concepts for Simulink integration:
        i. RobotInit() needs to call Code_Gen_Model_initialize()
        ii. RobotPeriodic() needs to call Code_Gen_Model_step()
        iii. Input ports: Code_Gen_Model_U.*input_port_name*
        iv. Output ports: Code_Gen_Model_Y.*output_port_name*
        v. Internal model test points: Code_Gen_Model_B.*test_point_name*
        vi. Tunable parameters are made available to SmartDashboard by SimulinkSmartDashboard.h and SimulinkSmartDashboard.cpp files generated by the Simulink code generation scripts.
    d. Make changes to the “Robot.h” and “Robot.cpp” files. These files are in the “src/main/include” and “src/main/cpp” folders. See file difference reports below (changes are commented). These files may be copied from the repository instead of typing everything out (to speed things up).

Robot.h

```
6  6
7  7   #include <frc/TimedRobot.h>
   8  +#include "Code_Gen_Model_ert_rtw/Code_Gen_Model.h"  // This line is for the Simulink code
   9  +#include <frc/SmartDashboard/smartdashboard.h>  // This line is for the Smart Dashboard to Simulink input port
   10 +#include <networktables/NetworkTable.h>         // This line is for the Smart Dashboard to Simulink input port
   11 +#include <networktables/NetworkTableInstance.h> // This line is for the Smart Dashboard to Simulink input port
   12 +#include "include/SimulinkSmartDashboard.h"     // This line is to support tunable parameters from Simulink
8  13
```

```
25 30
26 31    void SimulationInit() override;
27 32    void SimulationPeriodic() override;
28      -};
   33 +
   34 +private:
   35 +  void PreStep();
   36 +  void PostStep();
   37 +  nt::NetworkTableEntry Entry;
   38 +  SimulinkSmartDashboard m_TunableSmartDashboard;   // This line is to support tunable parameters from Simulink
   39 +};
      \ No newline at end of file
```

Robot.cpp

```
4  4
5      -#include "Robot.h"
       5  +#include "include/Robot.h"
6  6
7      -void Robot::RobotInit() {}
8      -void Robot::RobotPeriodic() {}
       7  +void Robot::RobotInit() {Code_Gen_Model_U.Game_State = 0;
       8  +  Code_Gen_Model_initialize(); //code gen model init
9  9
```

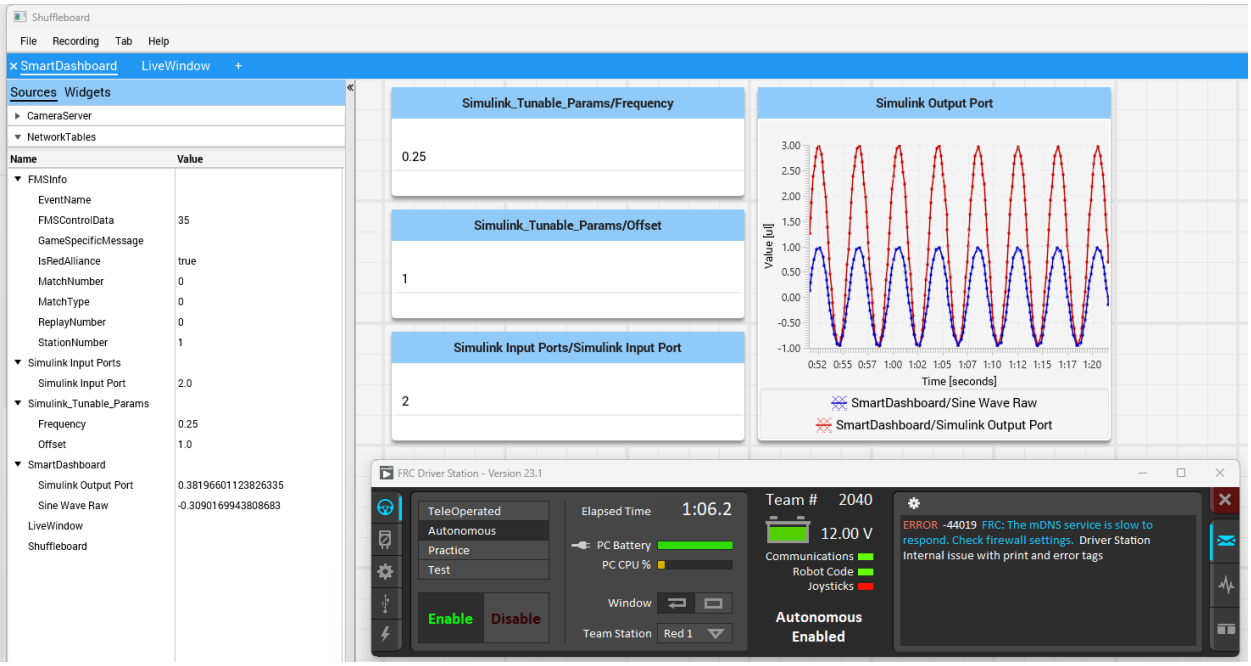
```
9  9
10     -void Robot::AutonomousInit() {}
       10 + // This block of code is just to get a Smart Dashboard parameter passed to a Simulink input port
       11 + // Normally the Simulink input ports would be connected to Human Input Devices or Sensors
       12 + nt::NetworkTableInstance NTinst = nt::NetworkTableInstance::GetDefault();
       13 + auto NTtable = NTinst.GetTable("Simulink Input Ports");  // section in Network tables
       14 + Entry = NTtable->GetEntry("Simulink Input Port");       // parameter name in Network tables
       15 + NTinst.AddListener(Entry, nt::EventFlags::kValueAll, [] (const nt::Event& event) {Code_Gen_Model_U.Smart_Dashboard_input = event.GetValueEventData()->value.GetDouble();});
       16 + Entry.SetDouble(1);  // Initial value
       17 +
       18 + // This line is to support tunable parameters from Simulink
       19 + m_TunableSmartDashboard.InitTunableSmartDashboard();
       20 +}
```

```
       20 +}
       21 +void Robot::RobotPeriodic() {
       22 +  PreStep(); //Robot wide PreStep
       23 +  Code_Gen_Model_step(); //Step the model
       24 +  PostStep(); //Robot wide PostStep
       25 +}
       26 +
```

```
       26 +
       27 +void Robot::AutonomousInit() {Code_Gen_Model_U.Game_State = 1;}    // Input ports are prepended with "Code_Gen_Model_U."
11 28  void Robot::AutonomousPeriodic() {}
12 29
13     -void Robot::TeleopInit() {}
       30 +void Robot::TeleopInit() {Code_Gen_Model_U.Game_State = 2;}    // Input ports are prepended with "Code_Gen_Model_U."
14 31  void Robot::TeleopPeriodic() {}
15 32
16     -void Robot::DisabledInit() {}
       33 +void Robot::DisabledInit() {Code_Gen_Model_U.Game_State = 0;}    // Input ports are prepended with "Code_Gen_Model_U."
17 34  void Robot::DisabledPeriodic() {}
18 35
19     -void Robot::TestInit() {}
       36 +void Robot::TestInit() {Code_Gen_Model_U.Game_State = 3;}    // Input ports are prepended with "Code_Gen_Model_U."
20 37  void Robot::TestPeriodic() {}
21 38
```

```
24 41
       42 +void Robot::PreStep() {}
       43 +void Robot::PostStep() {
       44 +  // This line sends the Simulink output port data to Smart Dashboard
       45 +  // Output ports are prepended with "Code_Gen_Model_Y."
       46 +  // In this case it is only going to the dashboard, but normally it would be going to drive a motor output
       47 +  frc::SmartDashboard::PutNumber("Simulink Output Port",Code_Gen_Model_Y.Smart_Dashboard_output);
       48 +
       49 +  // This line sends the Simulink test point data to Smart Dashboard
       50 +  // Test points are prepended with "Code_Gen_Model_B."
       51 +  frc::SmartDashboard::PutNumber("Sine Wave Raw",Code_Gen_Model_B.sine_wave_raw);
       52 +}
       53 +
25 54  #ifndef RUNNING_FRC_TESTS
26 55  int main() {
27 56    return frc::StartRobot<Robot>();
```

12. Build and Deploy code to the RoboRio.
    a. Connect your computer to the robot Wi-Fi radio.
    b. Ctrl+Shift+P (View > Command Palette …)
    c. Choose from the dropdown “WPILib: Deploy Robot Code” (Shift+F5)
    d. Check the VS Code terminal to see if the build is successful.
13. Open the FRC Driver Station
    a. Choose which game state to run (TeleOperated, Autonomous, Test).
    b. Enable
14. Open Shuffleboard
    a. Open the “shuffleboard.json” file in the “shuffleboard_layouts” folder.
    b. Change the frequency, offset, and input port amplitude adjustment.



- Note: The “Sine Wave Raw” signal only runs when the Autonomous game state is selected and enabled. Otherwise, the sine wave signal will be static because that subsystem is not active from the switch case in Simulink.