Remotely Operated HyperRail Developer Guide

Team 11 HyperRail Developer Guide

Kyle Barton, Samuel Barton, Michael Guske, Orion Hollar

ECE 342: Winter 2021

Professor Matt Shuman

March 5th, 2021

Table of Contents

1 - System Overview

The HyperRail is a multi-axis robotic system that was built to allow remote access for the maintenance and sampling of agricultural crops. From the high level, the system consists of two aluminium rails and a control box mounted to the Y-Axis. The rails are broken up into two dimensional axes, there are two horizontal axes creating an X-Y coordinate plane. A third vertical axis (the Z-Axis) can be installed as well, but only two axes are required for minimal use of the HyperRail. Each rail is moved by a rubber belt connected to a stepper motor. The X-Axis has two stepper motors to move it, while only one stepper motor is needed to move the control box along the Y-Axis. From a lower level view, the control box consists of a Power Distribution Board (PDB), two microcontrollers, four stepper motor drivers and a grasping mechanism mounted on the chassis. Besides the physical hardware of the HyperRail, custom software had to be made in order to control the system. Three different software programs make up the backbone of the HyperRail, they are GRBL, MQTT Comms, and the Graphical User Interface (GUI).

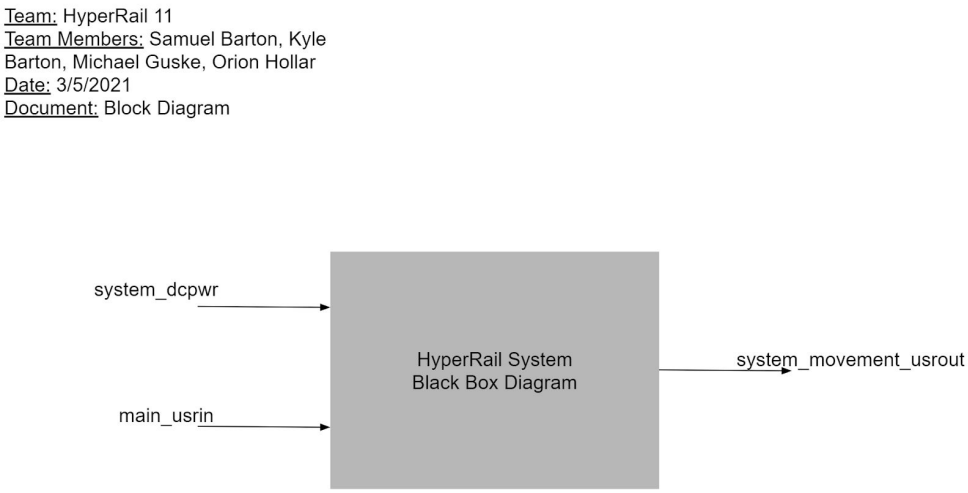

Figure 1: High-level black box diagram of the remotely operated HyperRail system(interface definitions in Appendix).

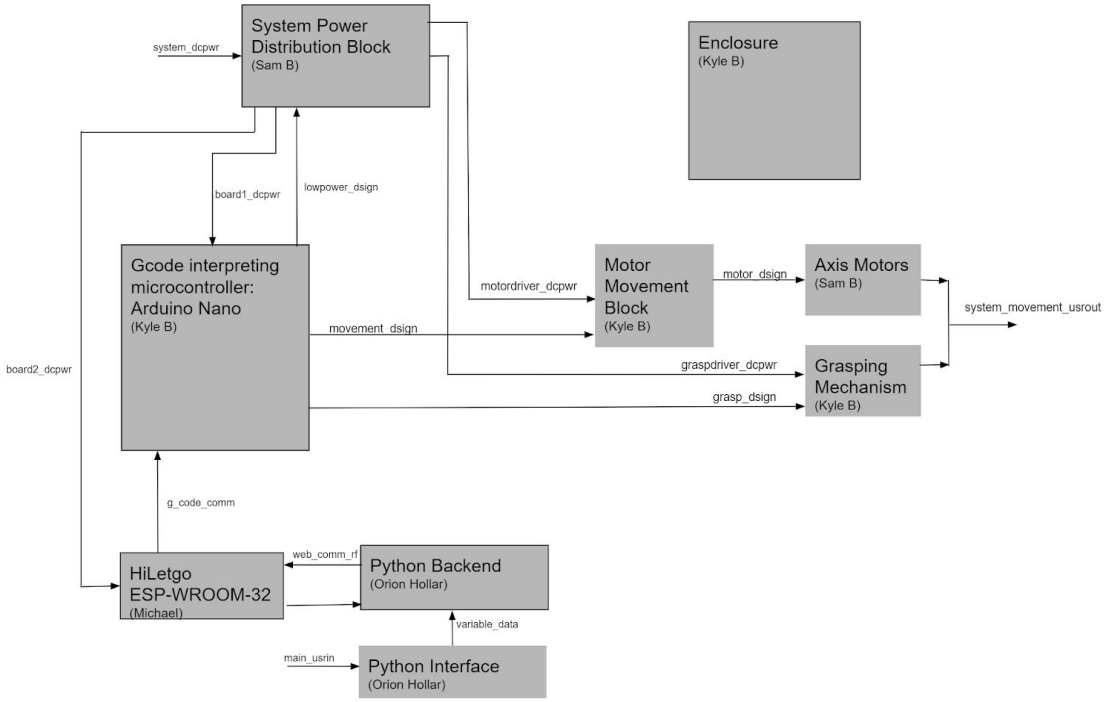

Figure 2: In-depth block diagram of the remotely operated HyperRail system (interface definitions in Appendix).

2 - Electrical Specifications

Table 1: Recommended Operating Conditions

Symbol	**Typ**	**Unit**
system_dcpwr	24	V
motordriver_dcpwr	24	V
graspdriver_dcpwr	4.97	V
lowpower_dsign	4.8	V
board1_dcpwr	4.97	V
board2_dcpwr	4.97	V
movement_dsign	4.7	V
motor_dsign	NEMA23: 2.6 NEMA17: 2.2	V V
grasp_dsign	4.7	V
g_code_comm	4.7	V

Table 2: Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Unit
system_dcpwr	Voltage applied by external power supply.	-0.3	30	V
motordriver_dcpwr	Stepper motor driver input voltage.	10	30	V
graspdriver_dcpwr	Servo motor input voltage	4.8	5.09	V
lowpower_dsign	Output from Arduino to relay switch in PDB	0	5.5	V
board1_dcpwr	Power supplied from PDB to Arduino.	4.5	5.5	V
board2_dcpwr	Power supplied from PDB to ESP32	4.7	10	V
movement_dsign	PWM control from Arduino to Motor drivers.	0	5.5	V
motor_dsign	Digital signal from DM332T stepper drivers to stepper motors		NEMA23: 2.6 NEMA17: 2.2	V
grasp_dsign	PWM used to control servo motor position	4.8	5	V
g_code_comm	Digital signal from Arduino Nano grbl pins	4.5	5	V

3 - User Guide

The HyperRail system has been designed to work off of 24V from an S-201-24 DC Power Supply, includes serial communication between the ESP32 wireless module and the G-code interpreting Arduino Nano, uses an MQTT network to allow for wireless communication between the ESP32 and the user's interface of choice, and results in desired motor movement using G-code commands. The following steps will ensure that future use of this system will work as intended.

3.1 - Setup:

- Hardware connections:
  - Microcontrollers:
    - The system must include an ESP32 wireless development board for connection to the MQTT wireless network to send the user's G-code commands from their desired user interface to the HyperRail system.
    - Either an Arduino Nano or Uno can be used to handle the grbl libraries for G-code interpretation. However, the designed enclosure has been created to fit an Arduino Nano which is substantially smaller than the Uno.
    - These two boards are currently established to communicate serially between each other using the Tx and Rx pins. Connecting a GND pin on each board to each other is also required for serial communication.
  - Wiring:
    - In reference to the schematic shown in the PCB Information section of this guide, the S-201-24 power supply that is the main source of power must be connected to J1 on the top left of the PCB. Note: The system **must** be powered using 24V to pin J1 for the entire system to work.
    - As the Power Distribution Board is used to power both the ESP32, Arduino Nano, and the SG90 servo motor, the 5 header pins located at J2 must be used to power each of the external electronics. As ground pins are limited, we suggest grounding the Arduino from the top pin on header J3 on the PCB to allow ease of grounding for the servo motor.
    - The high-power XT30 connectors J4-J7 on the bottom right of the PCB are used for powering each of the motors' corresponding stepper drivers. There's no specifics as to which driver/motor goes to which connector as they all supply 24V.
    - Each of the DM332T stepper drivers have pulse, direction, optocoupler, and enable pins which need to be connected to the corresponding pulse and direction pins for each axis on the grbl Arduino pinout shown in the Design Artifacts section of this document. The optocoupler needs 3.3V on each stepper driver (despite the datasheet calling for 5V) which can be supplied from the Arduino Nano 3.3V output pin.
    - Each of the stepper motors on the HyperRail system have 4 wires corresponding to the coils inside the motors that allow for the stepping to take place. On the DM332T drivers, there are 4 female connectors labeled A+, A-, B+, and B- which are the connectors corresponding to the stepper motor coils. The stepper motors have 4 wires, one for each of these connectors. The connection should be made as the following:

      A+: Black, A-: Green, B+: Red, B-: Blue

- On the stepper drivers, there are 6 switches between the power/coil connections and the Arduino control connections. Switches 1-3 correspond to the dynamic current flow that is allowed to the stepper motors, while switches 4-6 correspond to the microstepping resolution desired. Make sure that switches 1-3 correspond to the current rating of the connected stepper motors. On the top surface of the stepper drivers themselves, there is a guide as to which switch configuration corresponds to the amount of current and microstepping.
- Pin J3 is the pin to be connected to pin A3 on the Arduino Nano for Low-Power Mode. When pin A3/J3 is activated, the transistor allows current flow through the collector of the relay - disconnecting the stepper drivers and motors from the 24V power source.

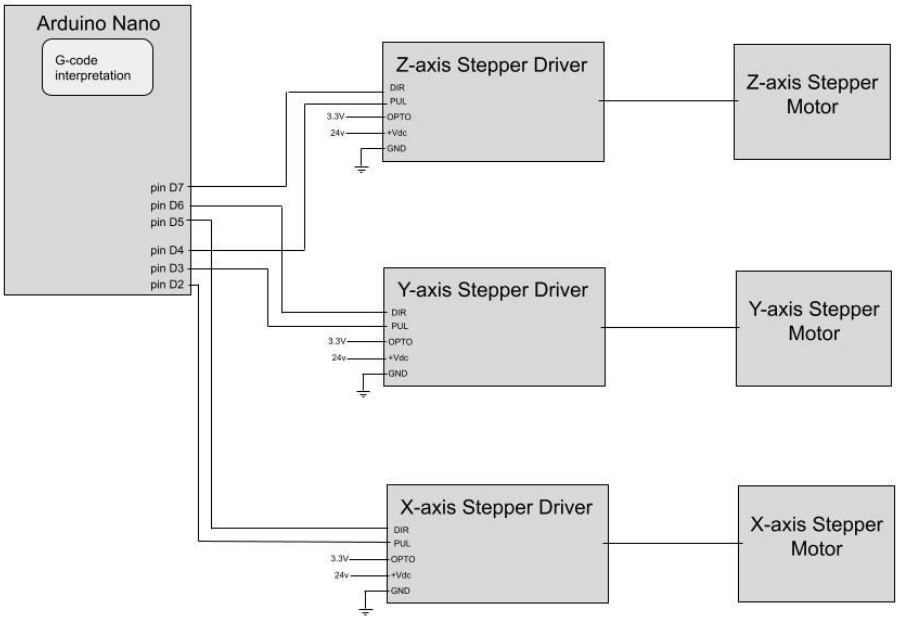

Figure 3: Wiring diagram for connection between the Arduino Nano and the DM332T Stepper Drivers.

- Grbl and configuration:
  - As servo control is not supported with the standard grbl v1.1 library, a different grbl library must be flashed to the Arduino to handle the grasping mechanism capabilities. This library can be found here: https://github.com/cprezzi/grbl-servo.
  - Grbl only needs to be installed on the Arduino, the ESP32 requires its own code to relay G-Code commands from the user interface to the Arduino
- ESP32 code:
  - Libraries:
    - The ESP32 code uses two libraries, the WiFi.h and the PubSubClient.h library. The WiFi.h library gives us access to a variety of different functions that set up and check Wi-Fi connectivity. The PubSubClient

library allows us to use MQTT functionality with the ESP32. This includes things such as establishing the ESP32 as a client and accessing a specific broker.

- Functions:
    - Callback(): This function accepts in a string, a byte, and an unsigned integer as it's parameters. The callback function collects whatever outgoing message is waiting at the broker for the subscribed topic. Using a for loop, the function iterates through each character in the payload byte (which pulled from the subscribed topic). These characters are then stored in a string, which after the for loop has ended goes through a series of continitional statements. Depending on the command, a different conditional will be executed. After the specific condition has been met, then the string is transmitted using serial to the Arduino Nano.
    - respond(): The respond() function does not have any parameters to it. The function is called when the ESP32 has sent a message to the Arduino Nano and now needs to relay the Arduino's response back to the user. The function essentially reads the serial ports on the ESP32 and collects the data into a string. The characters of the array are then stored into an array of characters, or chars. After this, the array of chars is published to the broker.
    - setup_wifi(): The setup() function is used to connect the ESP32 to a Wi-Fi network. This function then uses a predefined function from the WiFi to connect to a Wi-Fi network.

- GUI:
    - Libraries:
        - The tkinter python library comes pre-installed in most current python packages. Tkinter is used to create the interface itself as well as any other interaction with the user. The library contains classes to display buttons, labels, popups, group together other classes, and more. Near the beginning of the program the line 'top = Tk()' creates the primary window of the GUI which contains all other components of the GUI. A few lines down, firstgroup is made as a sub component of top, as seen in the first argument being 'top', a boarder of 5 pixels, seen with 'bd = 5' and a hexadecimal background color with 'bg = '#F0E9D5''. Next are the input fields. A label called L1 with text 'X, Y, F' is oriented on the left side of the firstgroup frame. Then the three input areas are created using Entry() oriented on the right side of the firstgroup frame. The entries are created in the opposite order as they display on the GUI because the earlier lines are given priority when being oriented to the right. Finally buttons are created. The first button is named Gzero, has a border of 5 pixels, displays the text 'G0' on it, and when pressed will run the function labeled 'G0' as seen with the 'command=G0' input. For every component created in tkinter they must be completed using the .pack() command, which also can take inputs such as the orientation of the component such as "L1.pack(side = LEFT)" will orient the component L1 to the left side of whatever group it is part of.
        - The paho-mqtt library must be installed. This library is used to wirelessly connect the interface with the rest of the system with an online broker as a middleman. To connect to a specific broker mqtt needs a specific address and a 'room' to send information to and to read. All systems that need to be connected will be looking at the same room. When one system

publishes information to a specific topic in a room, all other systems that are subscribed to that topic and connected to that room will be notified and be able to receive that information.

- Functions:
    - check_inputs: A simple command that is used to ensure that the information typed into the 3 data entry boxes are float values. It accepts one input which it will attempt to translate into a float value. If it is unable to translate it to a float value, an exception will be called which creates an error popup for the user and returns a value of 1 to indicate an error. If it can be translated to a float value then an exception will not be called and a value of 0 will be returned to indicate no error.
    - G0 & G1: The G0 and G1 commands construct an output string from the data input into the 3 entry boxes. First both functions start by creating a string called output with the values ‘G0 X’ and ‘G1 X’ respectively. The X is put in because the commands have a format of ‘G1 X# Y# F#’. Next values are taken in from the entry boxes with the .get() command. These values are then input to the check_inputs command. If any of the check_inputs commands return a 1, then it indicates an error and the function returns. But if they all pass the check then the program begins constructing the output string by appending the variable values to the end of the current string. Using client.publish the output string is then sent to the mqtt topic named “ESP32”.
    - G90-M6: All 6 of these commands are extremely simple. As the G Code for these commands have no variables, the functions just publish a static string to the mqtt topic named “ESP32”.
    - PSM: This function toggles the power saving mode functionality of the system. it uses a variable named ‘state’ to remember if the system is currently already in power saving mode or not. If the system is not in power saving mode then it will publish the string “PSMON” to the mqtt topic “ESP32” to be read by the system. It will then set the current state to 0 to indicate power saving mode and deactivate all of the other buttons on the interface. If the system is currently in power saving mode then it will publish “PSMOFF”, set the current state to 1 and restore functionality to the other buttons.
    - on_connect: This function runs as soon as the program connects to the mqtt room. It makes the interface subscribe to the topic named “resp”, which stands for response. When the rest of the system has something to send back to the user, it will write the information to the topic “resp” and the interface will run the on_message command when it notices the update.
    - on_message: This command takes the information sent by other users into the subscribed topics and then presents them to the user. First it stores the message into a string labeled command using str(msg.payload). Because the messages to and from the free broker can occasionally have noise, the function then uses a loop to only pull correct characters from command and store them in another string named temp. Finally a popup is made using tkinter titled ‘Feedback’ which will display the received message string to the user.

3.2 - Use:

- Commands:
    - Movement:
        - The G-code commands G90, G91, G20 and G21 are used to configure how movement commands are interpreted. G90 and G91 enable Absolute Mode and Incremental Mode respectively. If told to go to X 20 and Y 5, Absolute mode will make the head move to that position in relation to its default position and Incremental mode will make the head move 20 units in the x direction and 5 units in the y direction in relation to its current position. G20 and G21 enable Inches Mode and Millimeter Mode respectively. These commands interpret whether an input of X 20 means to move 20 inches or 20 millimeters. The system starts by default in Incremental Millimeter mode.
        - The G-code commands G0 and G1 move the head to the desired positions. G0 takes an X and Y value and goes at a specific speed. G1 takes in an X, Y, and F value, with the F value determining the speed that the head moves at. The G90, G91, G20 and G21 commands listed above determine how these variables are interpreted.
    - Change of Tool:
        - As the “Change of Tool” command in G-code (M6) is not currently supported on grbl v1.1, the ESP32 includes a work-around code for using the change of tool. The grasping mechanism (which includes the SG90 servo) is what will be used for the change of tool. When the M6 command has been passed from the user interface to the ESP32, the ESP sends two servo commands (included from the manipulated grbl library above) to open and close the grasping mechanism using the “S#” commands. Refer to the ESP code section of this document for insight on how this command works.
    - Low-Power Mode:
        - The Low-Power Mode feature of this HyperRail system is custom to this project. As there’s no coolant involved in this project, the “Coolant Enable pin (A3)” on the Arduino Nano is used to send the signal to the PCB to enable/disable Low-Power Mode. When the user selects the Low-Power Mode option on the user interface, the ESP sends a homing command to the Arduino, followed by the M8 (coolant enable) command to the Arduino which engages the Low-Power Mode. When the user decides to take the system out of low-power mode, the ESP32 sends the M9 (coolant disable) command to bring pin A3 on the Arduino back to its resting state of approximately 0V.

4 - Design Artifact Figures

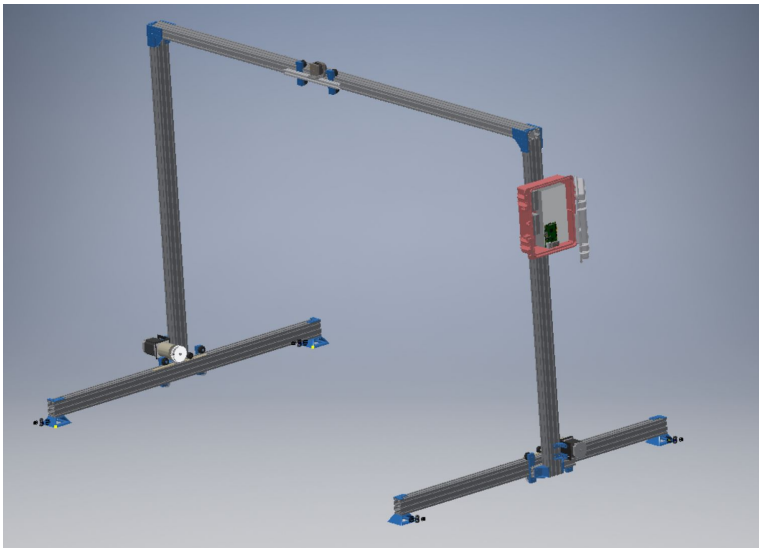

Figure 4: 3D model of the HyperRail mechanical system.

Figure _ above shows the rail system that our team's motion control system will be controlling. This design was created by the project's mentor Jorian Bruslind. This design currently only supports movement in the X and Y-axis, while progress in creating an extra arm in the Z-axis is currently underway. This structure is built with 80/20 T-slot framing, and the rail's movement is controlled by two NEMA17 stepper motors to handle X-axis movement and one NEMA17 stepper motor for Y-axis movement. The enclosure box on the right-side of Figure _ is to be designed by each of the HyperRail groups. For the purpose of our group, our enclosure will hold both the Arduino Nano, the ESP32, as well as the Power Distribution Board.

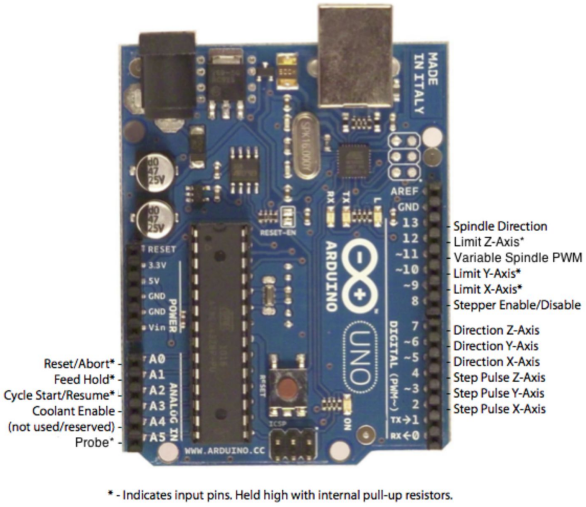

Figure 5: Grbl v1.1 Arduino Pinout.

Figure _ is a diagram of the grbl v1.1 pinout on an Arduino Uno. For the purposes of this project, an Arduino Nano was used with grbl as our G-code interpreting microcontroller. Though the model of the Arduino used is different than the one shown in Figure _ above, the corresponding

pins are the same on each of the boards. For implementation of our team's Low-Power Mode, pin A3 which is listed as a "Coolant Enable" pin was repurposed to send a 5V signal to the relay network used to disconnect the stepper motors from the power supply. For implementing a servo motor to control our system's grasping mechanism, a custom grbl library was used in which the PWM values from pin D11 are set from 255 (max PWM) to 0 for opening and closing the grasping mechanism. Other than these two changes, the rest of the pre-defined grbl pins shown in Figure _ are used for their intended purposes.

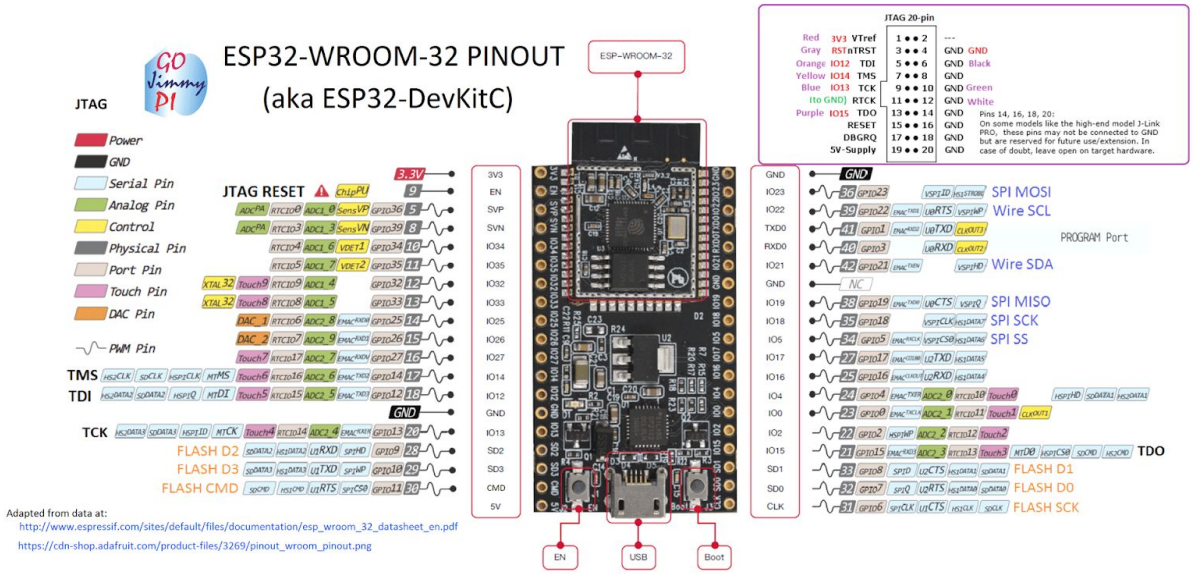

Figure 6: ESP32-WROOM-32 Pinout Diagram

The above figure shows the complete pinout for the ESP32, for the HyperRail though, only a few specific pins on the ESP32 are used. The Rx and Tx pins are used for serial communication with the Arduino Nano. Tx is connected to the Rx pin on the Arduino and Rx is connected to the Tx on the Arduino. The 5V and GND pin are used to supply power to the ESP32, since it is not being supplied power through the USB port. Both boards also need to be connected to each other using another GND pin, that's separate from the one being used for supplying power to the board.

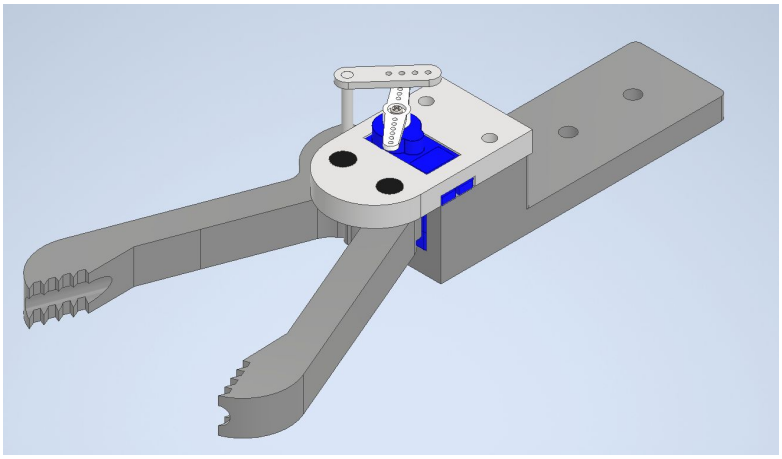

Figure 7: 3D model of the HyperRail grasping mechanism.

The grasping mechanism shown in Figure _ above was designed around the Tower Pro SG90 servo motor in which will be controlled and powered using the Arduino Nano. The rotation of the motor will be controlled by pulse-width modulation coming from pin D11 on the Arduino using the “S” G-code commands. Command S0 is used to open the grasping mechanism while command S255 sets the PWM to its maximum value to close the grasping mechanism. These values following the “S” command are still to be determined, as there was some delay in getting our grasping mechanism 3D printed for testing purposes.

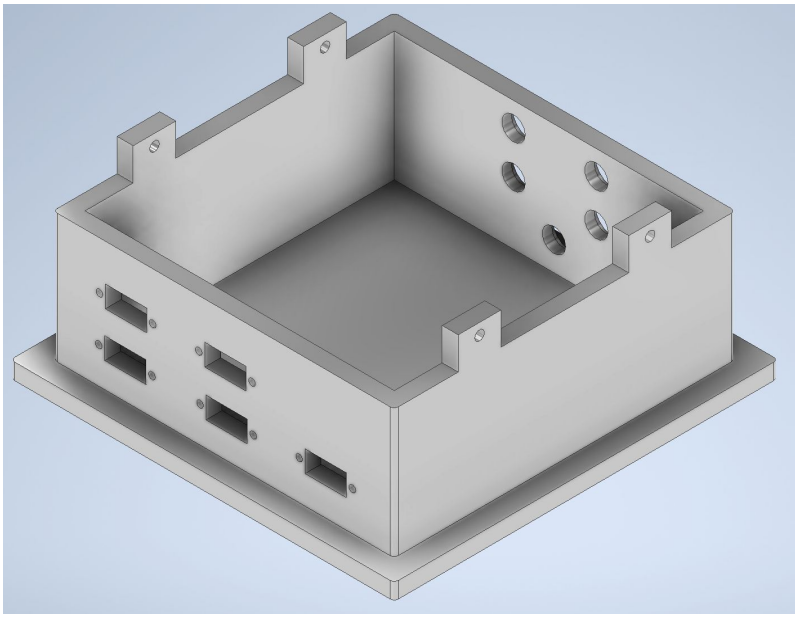

Figure 8: 3D model of the enclosure to be mounted to the side of the HyperRail System.

As stated earlier, our enclosure was designed to hold the two microcontrollers that our system includes, as well as the designed Power Distribution Board. As an engineering requirement, no wires must be leaving or entering the enclosure, so our design includes specific connectors for powering our stepper motors and drivers, as well as sending the controlling signals from the Arduino to the stepper drivers. The enclosure was sized to 180mm x 180mm x 82.5mm and

includes holes for 5 panel-mount db9 connectors for the signal wires exiting the enclosure as well as 52.1mm x 5.5mm DC Barrel Jacks used for powering the stepper drivers.

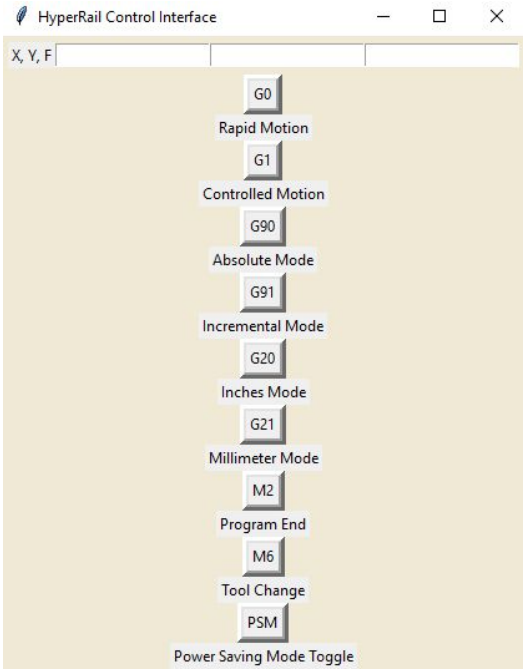

Figure 9: User interface for sending G-code commands to the ESP32.

The Graphical User Interface is the primary way to send commands to the rest of the system. Created using the tkinter library, it has a simple design with 3 entry boxes at the top to input variable values and a total of 9 buttons used to wirelessly send the associated command to the system. Each button has a label and a brief description of it's function underneath it. When the PSM button is pressed, the system is meant to go into power saving mode so the other buttons become greyed out and unresponsive until the button is pressed again.

5 - PCB Information

PCB design was completed using Altium CircuitMaker design software. The PCB was designed to hold the entirety of the Power Distribution Board, including a DC-DC buck converter used to power both of the microcontrollers included in the HyperRail system, as well as a relay network used to implement the Low-Power Mode command. The DC-DC buck converter was designed around Texas Instruments' TPS54232DR switching voltage regulator which was used in the JD Power Supply which was analyzed and used thoroughly during the previous Junior Design class last term. The PCB was designed to be 90mm x 60mm, allowing for plenty of space for the relatively large relay to be mounted to the PCB, as well as leave enough space for the large power-rail traces to not interfere with other components or traces.

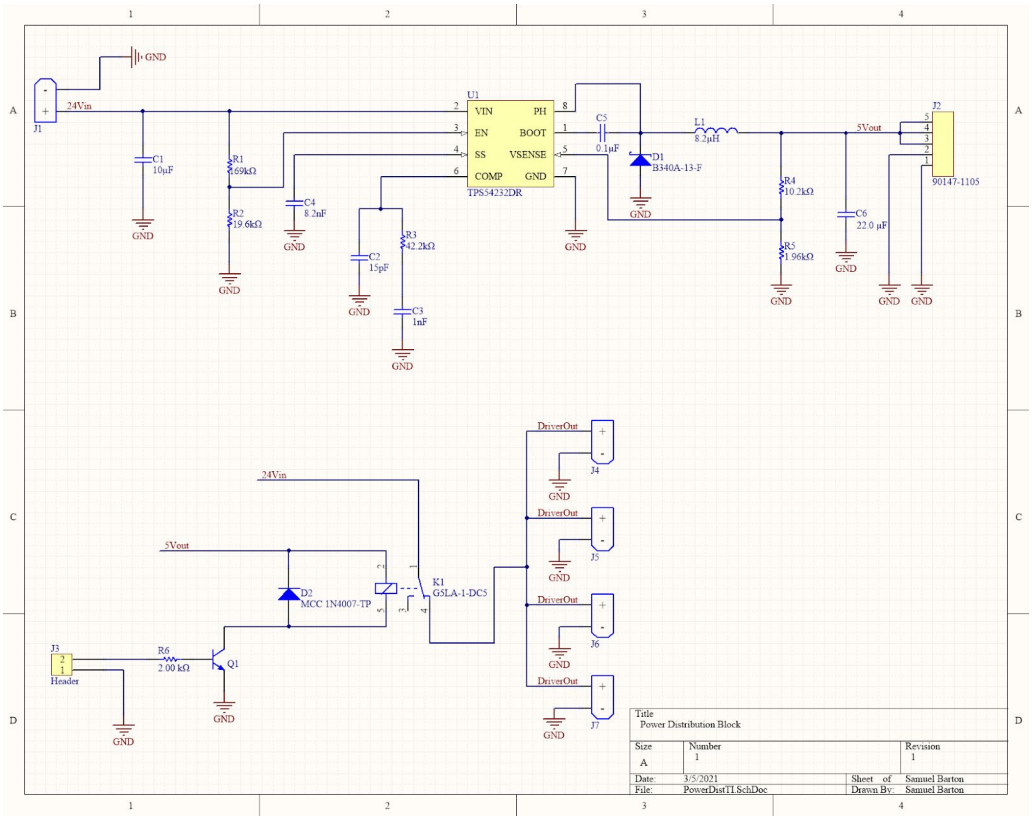

Figure 10: Schematic of the designed DC-DC buck converter as well as relay network for implementing Low-Power Mode, partially designed in Texas Instruments' [Webench Power Designer](#).

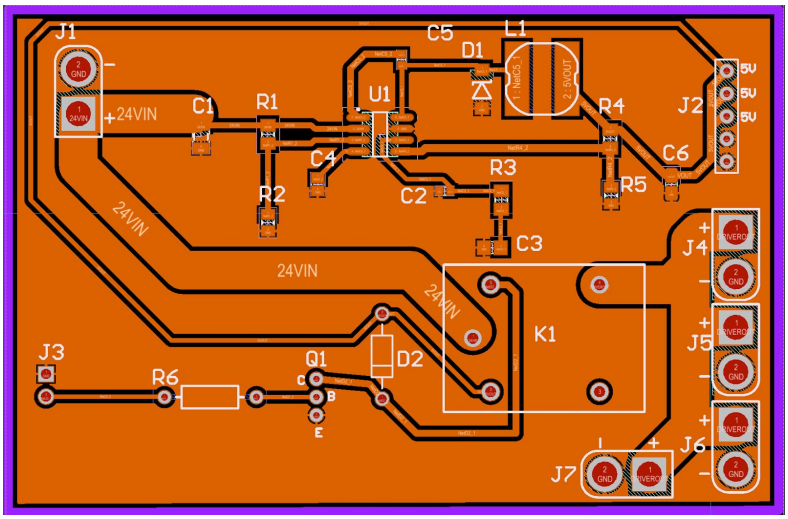

Figure 11: PCB traces of the Power Distribution board created in Altium CircuitMaker.

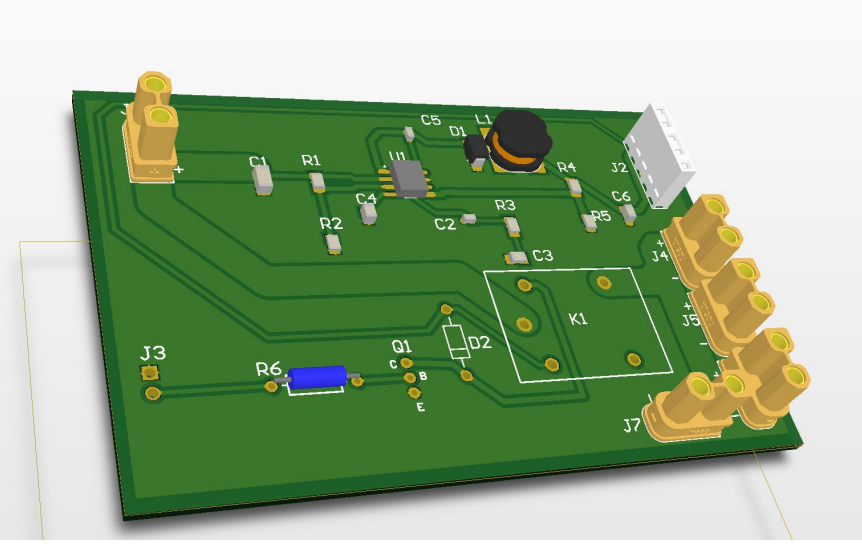

Figure 12: 3D rendering of the designed PCB.

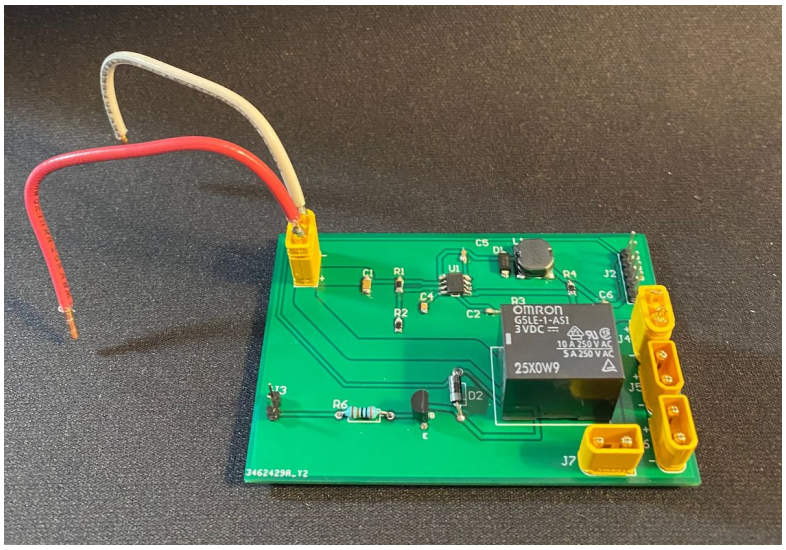

Figure 13: Physical PCB after all components had been added to the design and been tested for functionality.

6 - Part Information (BOM)

Table 3: Bill of Materials for components used for the creation of the framing and physical system of the HyperRail.

Type	Designator	Units	Unit Cost
Stepper Motor	NEMA17	1*	$10.48
	NEMA23	2	$53.50
Servo Motor	Tower Pro SG90	1	$3.99
Stepper Driver	DM332T	3**	$18.95
Microcontroller	Arduino NANO	1	$20.70
	HiLetgo ESP-WROOM-32	1	$10.99
		Total Components	**Total Cost**
		8	**$210.01**

* In the future, a z-axis will be established which will include another NEMA17 Motor
** If a z-axis is included, an extra DM332T must be added

Table 4: Bill of Materials for components used for the creation of the Power Distribution Board PCB.

Component Type	Designator	Value	Units	Cost
Capacitor	C1	10 μF	1	$0.32
	C2	15pF	1	$0.12
	C3	1nF	1	$0.17
	C4	8.2nF	1	$0.10
	C5	0.1 μF	1	$0.10
	C6	22 μF	1	$0.53
Resistor	R1	169k Ω	1	$0.10
	R2	19.6k Ω	1	$0.10

	R3	42.2kΩ	1	$0.10
	R4	10.2kΩ	1	$0.10
	R5	1.96kΩ	1	$0.10
	R6	1kΩ	1	$0.10
Diode	D1	B340A-13-F	1	$0.43
	D2	1N4007-TP	1	$0.10
Inductor	L1	SRN8040-8R2Y	1	$0.71
Relay	K1	G5LE-1-ASI DC3	1	$1.46
Switching Regulator	U1	TPS54232DR	1	$1.54
			Total Components	**Total Cost**
			17	$6.18

7 - Appendix

Table 3: System interface definitions.

Type	**Name**	**Characteristics**
DC Power (dcpwr)	system_dcpwr	$V_{nominal}$: +24VDC I_{peak}: 8.3A
	motordriver_dcpwr	V_{input}: +24VDC I_{input}: 6.9A
	board1_dcpwr	V_{input}: +5VDC I_{input}: <1A
	board2_dcpwr	V_{input}: +5VDC I_{input}: <1A
	graspdriver_dcpwr	V_{input}: +5VDC
User Input (usrin)	main_usrin	Type: typed float numbers and button presses Purpose: Accept user input
Data	variable_data	Type: Command calls and variable transfer Purpose: front-end communication with back-end
Wired Communication (comm)	g_code_comm	Type: Serial communication
Wireless Communication (RF)	web_comm_rf	Type: Wifi Purpose: Wireless communication to ESP32
Digital Signals (dsign)	movement_dsign	V_{max}: +5VDC V_{min}: 0VDC
	motor_dsign	Type: changing voltage for 4-coil stepper motor control
	grasp_dsign	Type: PWM
	lowpower_dsign	V_{max}: +5VDC V_{min}: 0VDC
User Output (usrout)	system_movement_usrout	Type: Desired movement of the HyperRail system