Modular Robotics: Molucube

Cole Butler
Ali Helmi
Andrew Hunt
Carlos Torres

SUBSYSTEM REPORTS

REVISION – Draft
4 December 2022

SUBSYSTEM REPORTS
FOR
Modular Robotics: Molucube

TEAM 10

APPROVED BY:

Project Leader	Date
Prof. Kalafatis	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	12/04/2022			Draft Release

Table of Contents

List of Tables

List of Figures

1. Introduction

The modular robot will be able to autonomously navigate a course to perform a desired task, in this case, collect ping pong balls and deliver them to a basket. The robot is broken down into four subsystems: Power Delivery and Motor Driver, Raspberry Pi and Microcontrollers, Housing and PCBs, and Object Detection and Gripper Arm.

2. Power Delivery and Motor Driver

2.1. Subsystem Introduction

Power Delivery Subsystem

- This subsystem is split into two further systems, the Battery Management System (BMS) and the Buck Converters. To ensure safe battery external charging and regular use discharging the BMS was needed. It would connect to an external Lipo battery charger and then the balancing cables from the Lipo battery packs would connect to the BMS as well. This ensures the battery cells all charge and discharge at the same rate. The BMS was tested and during testing a mosfet on the PCB blew. So, it was not able to be tested for battery charging and discharging validation as this was a PCB design flaw and couldn't be fixed in time. This would be further discussed in the failure analysis portion section below.

- As for the buck converters I was only able to test the 3.3v and 5v power outputs as the 12v power output rail was shorted and was narrowed down to a design flaw in the PCB where a diode was shorted. However, I was only able to get voltage outputs from 3.3v and 5v lines via multimeter since it was tested at my house during thanksgiving break. Once I got back to lab, I did some adjustments to get the 12v line working and ended up frying the board and it's useless now. This too will be further discussed in the failure analysis section.

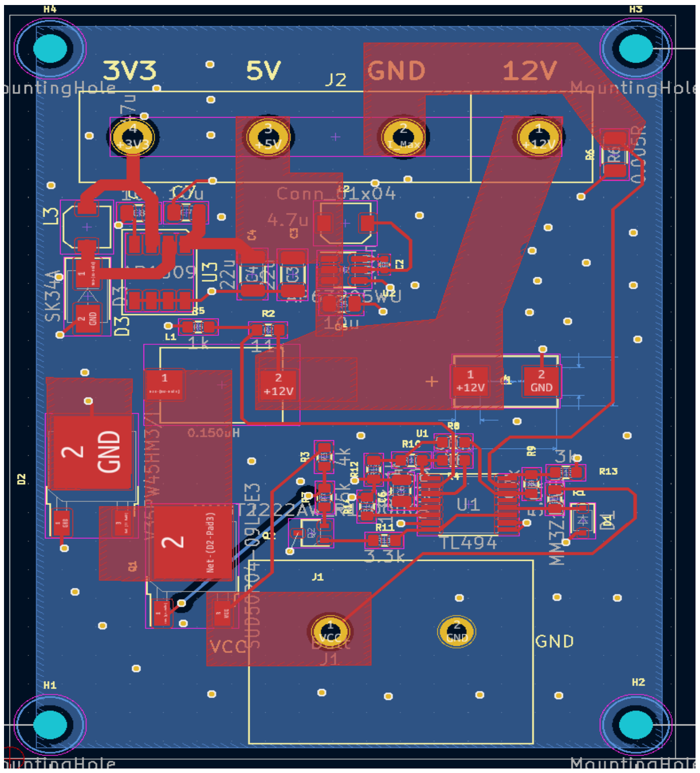


Figure 1 Buck Converters PCB Design

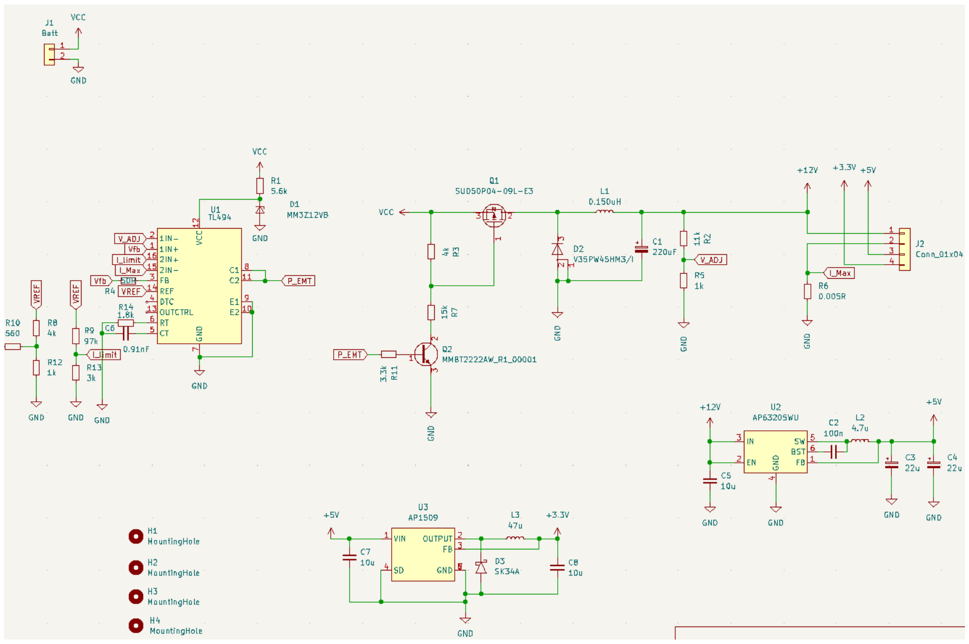


Figure 2 Buck Converters Circuit

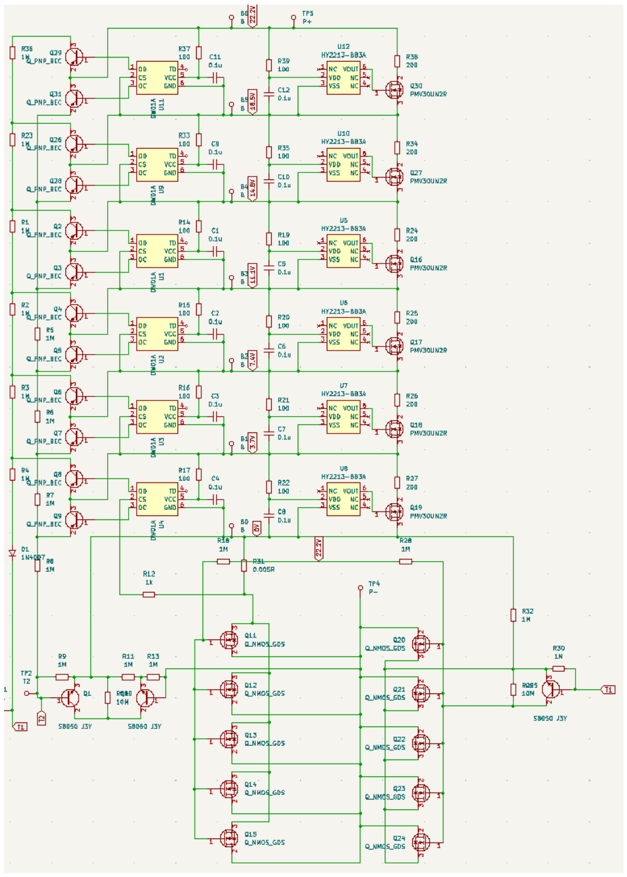


Figure 3 BMS Circuit

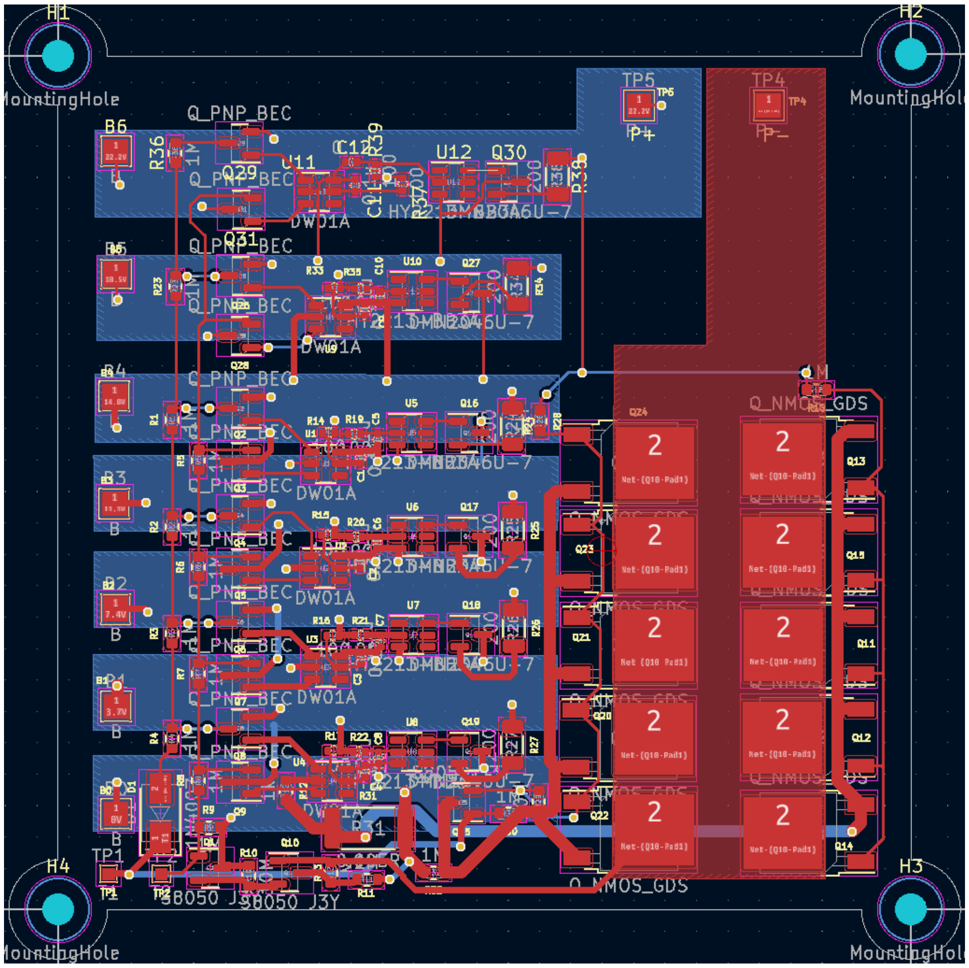


Figure 4 BMS PCB Design

Motor Driver Subsystem

- The Motor Driver subsystem would be taking in the power from the 12v buck converter as well as the 3.3v for the microcontroller and then using the logic from the microcontroller would allow for the H-bridge to control the direction of spin of the motor. This was tested and only forward direction of the motor was achieved, and it was later found that the NMOSs were not being turned on due to gate voltage not being high enough. This will be discussed further in the failure analysis section.

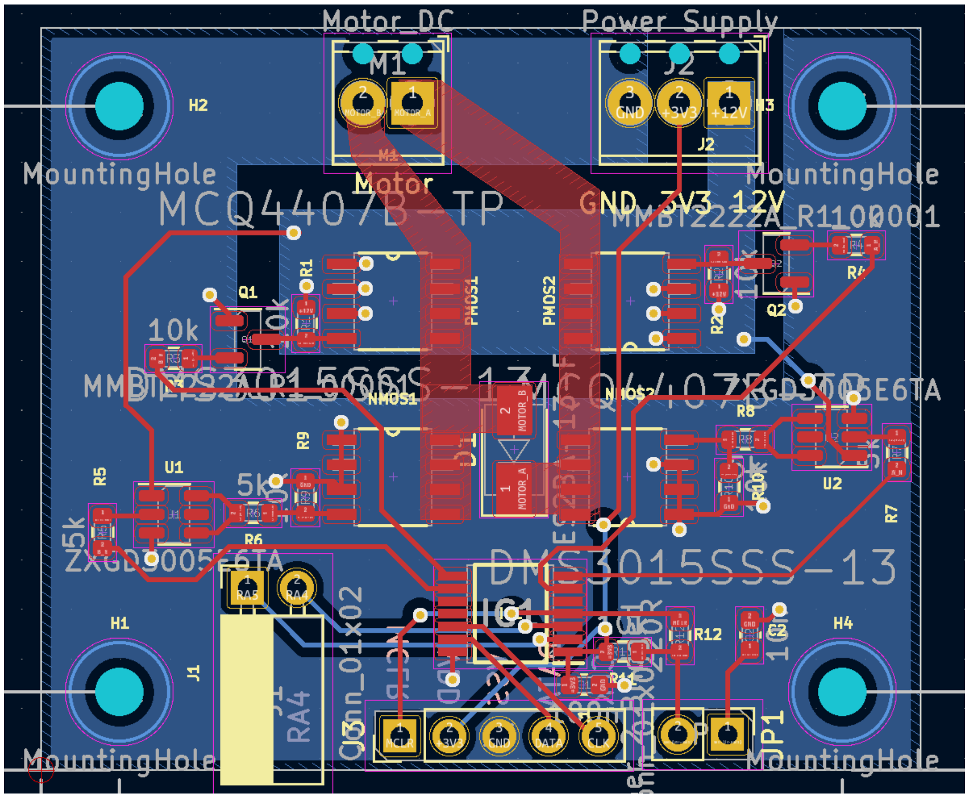


Figure 5 H-Bridge PCB Design

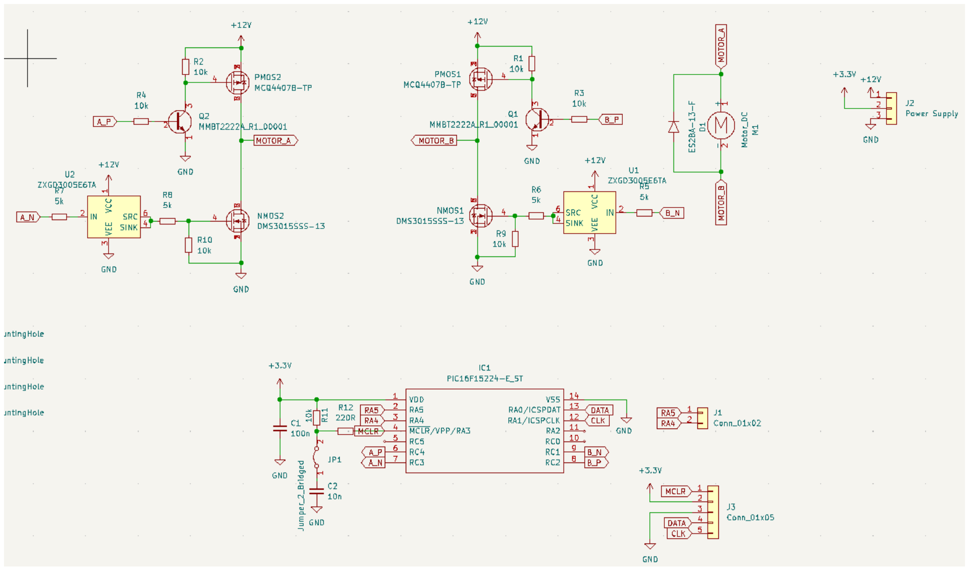


Figure 6 H-Bridge Circuit

2.2. *Subsystem Details*

Power Delivery Subsystem

- Two 22.2V Lipo batteries placed in parallel would be fed into the buck converters where then the voltage would be dropped or “bucked” down to 12V with a full load current of 30A then 5v and then 3.3V both with full load currents of 2A each. The DC wheel motors and gripper arm linear actuator both need 12v to operate, the raspberry pi and gripper arm servo need the 5v and the various microcontrollers need the 3.3v to operate. A major challenge was being completely down on the 12v line. As the 5v and 3.3v outputs relied on the 12v line to work the entire system was down. I was forced to bypass the 12v line and force 12v on to the 5v line that would be passed to the 3.3v line to work.

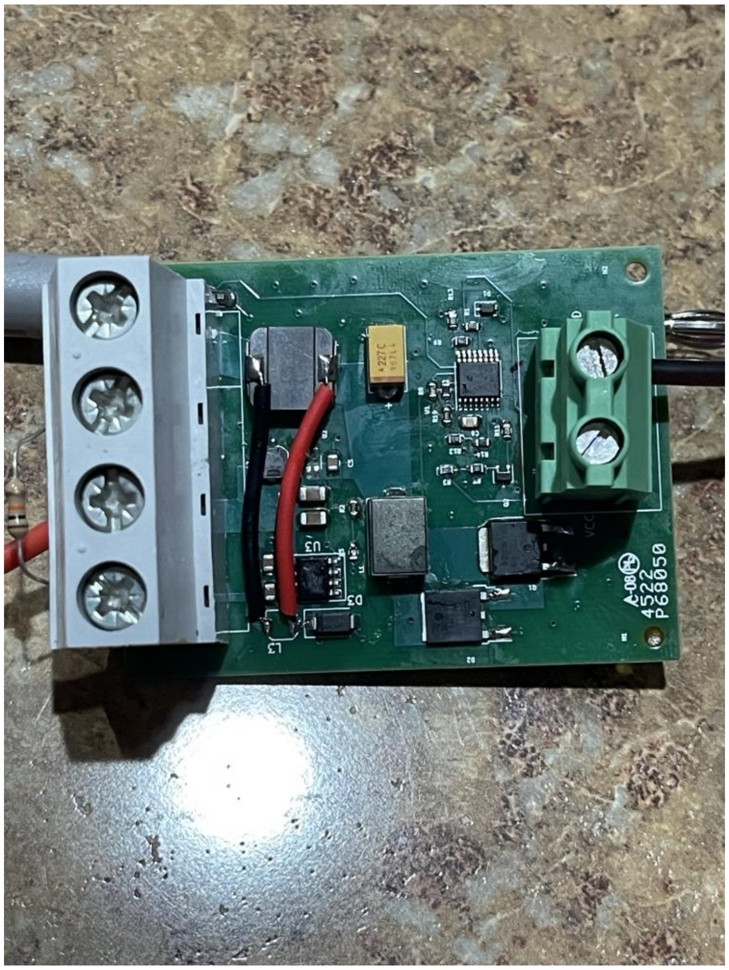


Figure 7 Physical Buck Converters PCB

- The BMS would take in the range of .1A-10A and 150W charging from the external charger and then the balance wires from the Lipo batteries themselves would be connected to the BMS PCB where the BMS would make sure all cells charge and discharge at the same rate ensuring battery health and safety. Each cell contains 3.7V which totals to 22.2v for 6 cells.

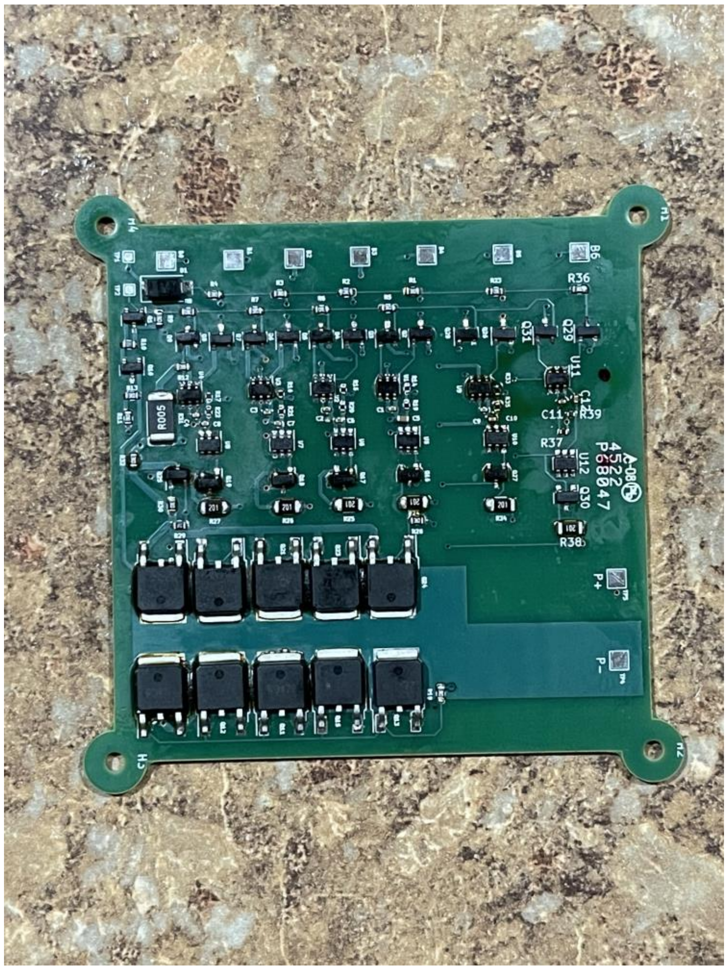


Figure 8 Physical BMS PCB

Motor Driver Subsystem

- The Motor Driver subsystem receives power for the wheel motors from the 12v output of the buck converter and then uses the 3.3v output line for the microcontroller. Logic from the microcontroller is used to control the “gates” of the H-bridge to then control which direction the DC motor spins. The motors are rated at a stall current of 9A but since our robot has four motors and the robot itself isn't heavy, we wouldn’t see the motor reach that 9A, so I designed the motor driver to operate at a safe and reasonable 5A for the motor.

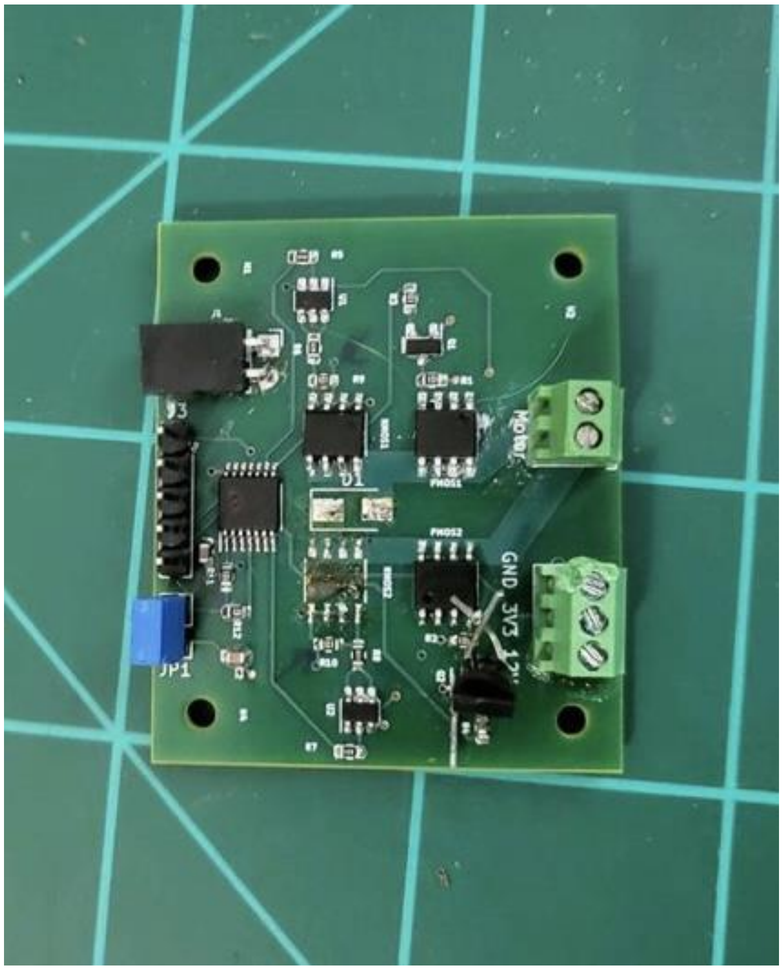


Figure 9 Physical H-Bridge PCB

2.3. Subsystem Validation

Power Delivery Subsystem

- In the introduction of my subsystems, I mentioned that I was able to get the 3.3v and 5v outputs of the buck converters to work but only was able to measure the output voltage using my handheld multimeter back at home during thanksgiving break I was unable to test for actual output current, voltage noise and efficiency until I was back in lab. Once I was back in lab, I attempted to fix the 12v line and in the process blew the board, so I was unable to salvage what was left. A proper buck converter should be able to produce the same voltage across a range of load hence why Dr.Lusher wanted us to use the eLoad in lab to test this. As well as operating up to a certain current. Since the 3.3V buck converter runs off of the output of the 5v line I couldn't really vary the input voltage as it would always be 5v give or take a few mV.

Table 1 3.3V buck converter voltage data

INPUT VOLTAGE	OUTPUT VOLTAGE	INPUT CURRENT
4.98V	3.31V	.03A

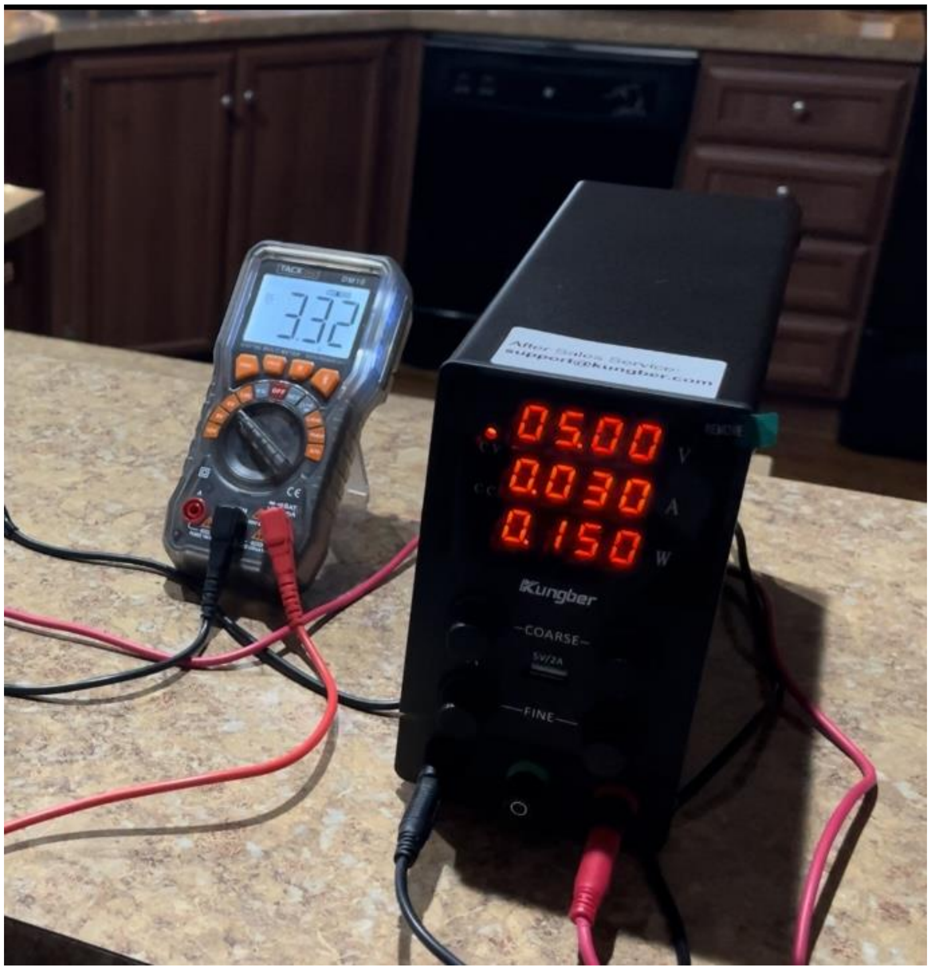


Figure 10 3.3V Buck Converter Output Voltage Validation

- The BMS would

Motor Driver Subsystem

- The Motor Driver subsystem was able to be semi validated. I was able to get the motor to turn but only in one direction. It was found that both NMOSs weren't being turned on so bypassing those and solely using the two PMOSs to turn the motor for demo was done.

2.4. Failure Analysis of Subsystems

To start, the buck converter PCB was semi functional at one point but out of desperation to get the 12v line fixed in time for demo I fried the board. After further and extensive analysis of the PCB design it was found that D2, which is a diode shown below, was “wired” wrong. The output/large pad of the diode was supposed to be connected to the 12v polygon line but instead I connected that large pad to ground and the 12v polygon line was connected to pin 1 which is connected to pin2 which was connected to ground which is why I was getting conductivity between the 12v output and the ground input thus a short. This will require me to redesign that small portion of the PCB design and reorder the board over Christmas break along with putting the correct footprint of the inductor for the 3.3v output as I ordered an inductor that was 3 times the size of the pad and so I had to solder wires from the pads to the inductor to make it work for demo.

For the BMS some small minor adjustments need to be made. When trying to connect my BMS to the external battery charger the charger told me “Reverse Polarity” at first, I didn’t know what this meant exactly then when I disconnected my circuit from the charger to diagnose, my U11(battery management IC) component for cell 6 sparked then smoked. It had burnt the traces inside the board and I couldn’t test/validate. It was found that I had a mosfet flipped and the drain was connected to BATT+ when it was supposed to be to the source. U11 oversees overcurrent protection and voltage. And I was missing a 1k ohm resistor on pin 2 of U11. Both problems were minor but I made this mistake on all 6 cells so I must go back and implement these fixes on the BMS PCB design for all 6 cells. Which would be done over Christmas break and then validated as soon as possible for 404.

Lastly, for the motor driver like I mentioned before, only the forward direction was able to work but not reverse direction. This is because the NMOSs of the H-bridge weren't being enabled so I modified the circuit to only run off of the PMOSs and got the motor going in one direction. After trouble shooting the issue, I found that the gate threshold voltage of the NMOSs was 2.5v and after viewing the voltage being measured at the gate, I found that it was about 1.7v. So im going to recalculate my values over Christmas break and have a working final PCB completed by 404.

3. Microcontrollers and Raspberry Pi

3.1. Subsystem Introduction

This subsystem consists of a Raspberry Pi, which is the main controller of the robot, as well as several microcontrollers, which control and serve as an interface between the Raspberry Pi and the motor, sensor, and gripper arm modules.

3.2. Subsystem Details

3.2.1. Raspberry Pi

A Raspberry Pi 4 serves as the main controller for the entire robot. While the computing power of the Raspberry Pi 4 is not necessarily needed for this subsystem, it is necessary in order to have enough performance for the Object detection subsystem. It does though have several features needed for this subsystem. It supports Wi-Fi as well as Bluetooth which will be necessary for communicating with the robot remotely. When the robot is able to connect to a known Wi-Fi network, it is possible to transfer files and update the software for the robot. If a known Wi-Fi network is unavailable, the Raspberry Pi is set up to support serial over Bluetooth which allows terminal access to run or edit programs as necessary.

The Raspberry Pi will eventually serve a larger purpose of running all of the control logic for the robot, which will allow it to navigate, locate objects, pick them up, and place the objects in a desired location. For 403, the Raspberry Pi mostly serves as an interface between the microcontrollers and a laptop in order to test the desired functionality before integration. It currently runs python scripts which talk to the microcontrollers via UART to get them to perform the desired functions.

For now, the Raspberry Pi consists of several test scripts that use a communication class that allows for easy communication with the microcontrollers. The class takes the desired address of the microcontroller as input, as well as an array of bytes intended to be sent as data. The class takes these inputs and constructs a data frame to be sent via UART. The class is also able to receive data from the buffer and return the data it receives.

3.2.2. Microcontrollers

The microcontrollers used are the PIC16F15224. These microcontrollers have the features needed for the various modules, including UART and PWM while also being a low cost to meet the goal of making a more affordable robot. The firmware of each microcontroller is mostly the same. They all run the same main program, and the type of module and communication address is chosen at compile time, which determines which other functions will be included by the compiler.

3.2.2.1. Communication

The microcontrollers communicate with the Raspberry Pi via UART. It was originally planned to use the RS-485 protocol for communication, but this requires the ability to send 9 bit words, which the Raspberry Pi is unable to do. Instead, 8 bit words are used with the most significant bit being reserved to indicate an address byte. Data is sent in frames consisting of an address byte, various numbers of data bytes, a check byte, and then an end byte, consisting of the number 126 in decimal as shown in the table below.

Table 2 Microcontroller data frame

Address Byte	Data Bytes	Check Byte	Final Byte
Value > 128, MSB + MCU Address	Data Values < 128	XOR of data bytes	126

While UART is normally point to point communication, the robot will require a bus with multiple microcontrollers. This is accomplished by having the microcontrollers send their transmit pins to a tristate mode so they will not fight with each other when one of them is transmitting back to the Raspberry Pi. Each Microcontroller will listen to each data frame, but only take action and respond if the frame is being sent to the microcontroller's address.

The Raspberry Pi and Microcontrollers communicate at a baud rate of 115200 bps to ensure fast transmission of data frames so there will not be a large gap between commands sent to motor modules.

3.2.2.2. Motor Drivers

Pulse Width Modulation is required to control the speed of the wheel motors, as well as select the angle for the servo motors in the gripper arm module. The PIC16F15224 has two PWM channels which allow for setting various pulse frequencies as well as the duty cycle. This allows for setting the correct frequency to match the required switching speed for the transistors of the motor driver.

The motor driver controller waits for a data frame from the Raspberry Pi which will tell it which direction to move and what speed to use. When this information is received, the microcontroller sets the desired values and sends a message back to the Raspberry Pi to verify it received the command. The different modes for the motor driver are: Forward, Reverse, Coast, and Brake. Forward and reverse also require a speed from 1-100 which is a percentage of how much power to use. Coast allows the motors to spin freely. Brake will cause the motors to resist motion.

When changing speeds, the motor driver is able to instantly change to the desired speed. When changing direction, the motor driver firmware delays several microseconds between stopping one direction and starting the other in order to allow all of the transistors to switch to the new configuration to prevent shorting directly to ground.

3.2.2.3. Sensors

The sensor module requires the use of a timer as well as an interrupt. The microcontroller must send a 10μs pulse to the ultrasonic sensor. The sensor measures the distance to a nearby object and then returns a pulse to the Microcontroller, the length of which indicates how far away an object has been detected. The length of this pulse in microseconds can be converted to inches or centimeters by dividing by a constant based on the speed of sound. The rising and falling edges of the pulse is detected by the microcontroller with a pin interrupt, and the length of the pulse is measured by a timer.

The sensor firmware currently constantly polls the ultrasonic sensor every 10ms and sends the distance data back to the Raspberry Pi. For integration with the entire robot, where there will be multiple sensors, this will likely have to be switched to the Raspberry pi requesting the sensor do a measurement and return the distance, in order to avoid any potential interference between different sensors.

3.3. Subsystem Validation

3.3.1. UART Communication

UART Communication was validated with a logic analyzer to determine the correct data was being sent and received. As shown in the figure below, the Raspberry Pi was able to send a data frame telling the microcontroller to set the motor mode to forward, and to set the speed to 55%. The Microcontroller received this frame, changed the percentage of the PWM, and sent back the original check byte to verify to the Raspberry Pi that it received the correct command. The entire process of sending the command, changing state and sending the reply takes around 850 µs.

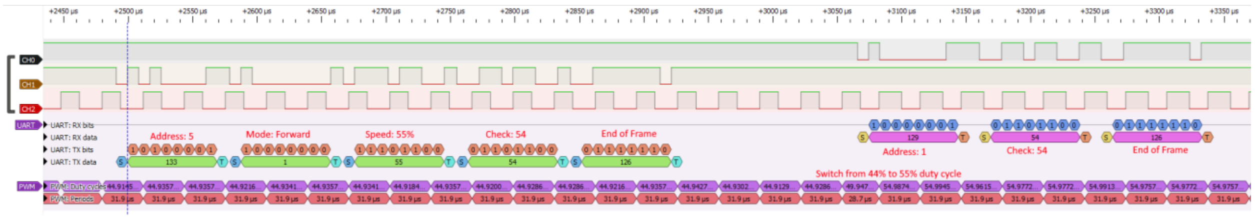


Figure 11 Data frame send and receive test

The validation for communicating with multiple microcontrollers on a bus was also validated with a logic analyzer as shown below. The Raspberry Pi sent out a data frame to each microcontroller's address, telling each to toggle an LED connected to one of the I/O pins. Each microcontroller only responded when its address was sent. This entire process took around 2.2 ms. Extrapolating out to multiple motor drivers, the process of updating each wheel motor could be expected to take around 3.5 to 4.5 ms, which should not cause a noticeable gap between the motors changing modes.

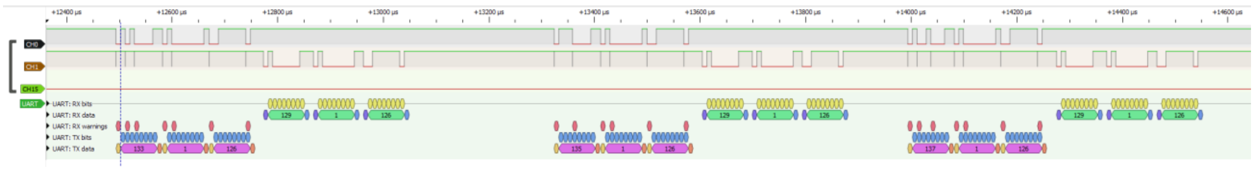
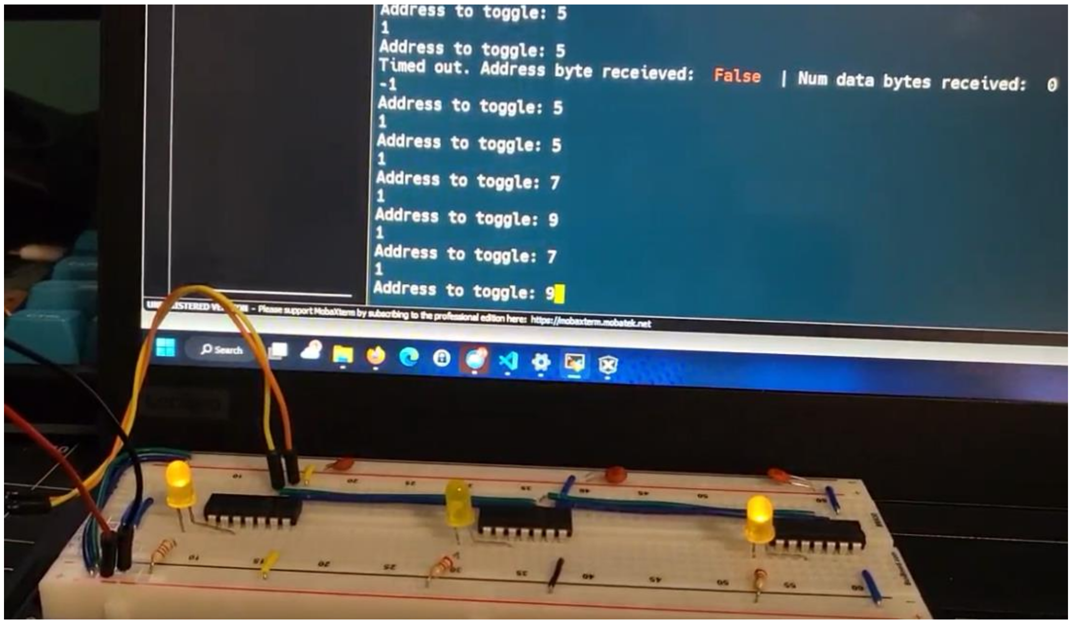


Figure 12 MCU network test

Validation for individual addressing of each microcontroller was done visually with a python script running on the Raspberry Pi. The script asks for an address as input. It then takes this address and sends the command to the desired microcontroller with the communication class.



```
Address to toggle: 5
1
Address to toggle: 5
Timed out. Address byte receieved:  False  | Num data bytes received:  0
-1
Address to toggle: 5
1
Address to toggle: 5
1
Address to toggle: 7
1
Address to toggle: 9
1
Address to toggle: 7
1
Address to toggle: 9
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
Search

Figure 13 Microcontroller network test with LEDs

The only issues remaining with the UART communication is an occasional error. Sometimes bytes are lost or malformed, so error handling will need to be added in 404 to gracefully handle communication issues.

3.3.2. Sensor

The sensor firmware was validated via logic analyzer and visually with a python script to determine that the microcontroller was receiving correct data and that it matched up with real world measurements. The logic analyzer confirmed that a 10 µs pulse was being sent as required by the data sheet, and that realistic pulse was sent back. The sensor module prototype was then validated visually by placing an object a known distance from the sensor and running a python script on the Raspberry Pi to constantly poll the microcontroller to measure the distance.

The measurement was not extremely accurate, as the floating point operations take too much memory on the microcontroller, forcing the use of integer math. This was compounded by needing to use inches, as that was the only unit available for larger measurements. Measurements can be made more accurate by switching to centimeters or millimeters which will give more detail with integers.

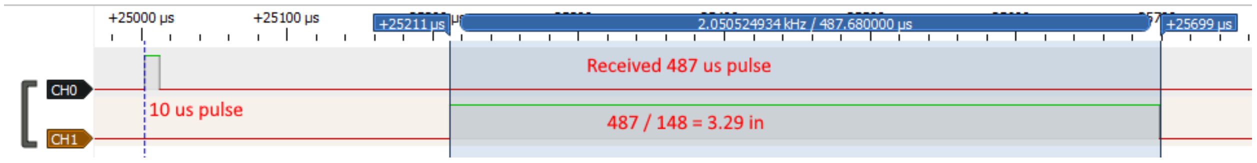
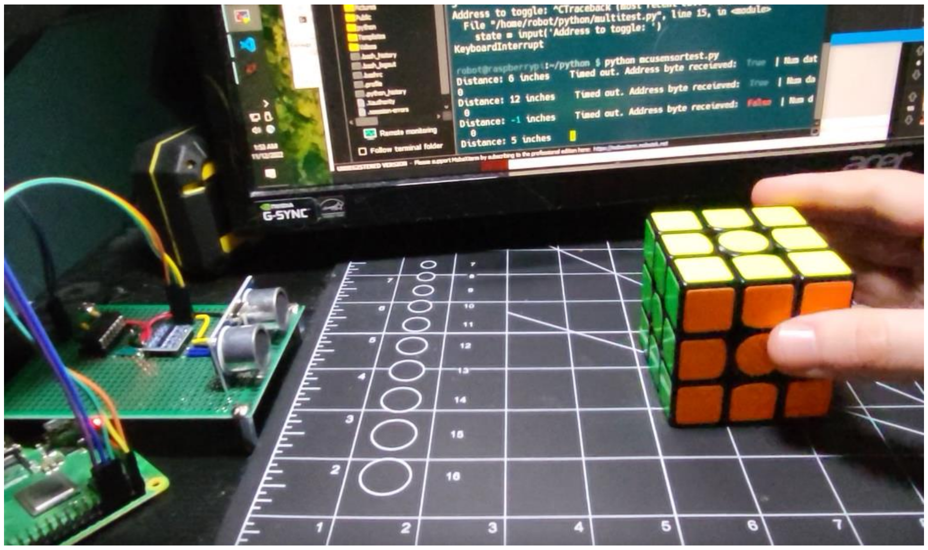


Figure 14 Analysis of ultrasonic sensor signal



```
Address to toggle: ^CTraceback (most recent call last):
  File "/home/robot/python/multitest.py", line 15, in <module>
    state = input('Address to toggle: ')
KeyboardInterrupt
robot@raspberrypi:~/python $ python mcusensortest.py
Distance: 6 inches      Timed out. Address byte recieved:  True  | Num dat
0
Distance: 12 inches     Timed out. Address byte recieved:  True  | Num da
0
Distance: -1 inches     Timed out. Address byte recieved:  False | Num d
0
Distance: 5 inches
```

Figure 15 Visual confirmation of sensor module

3.3.3. Motor Driver

The motor driver was validated via a logic analyzer as well as visually with the motor driver PCB. It was not fully able to be tested, as there were some errors with the PCB that prevented full functionality. The PCB was able to be modified in a way to allow the microcontroller to turn on the motor and control the speed. The motor controller firmware was initially tested without the motor driver to ensure the outputs were correct, to prevent possible errors that would lead to a short. The figure below shows a change in direction where the PWM has a 45% duty cycle, the firmware turns off all outputs to allow any transistors to fully turn off, and then turns on the other side at 65% duty cycle.

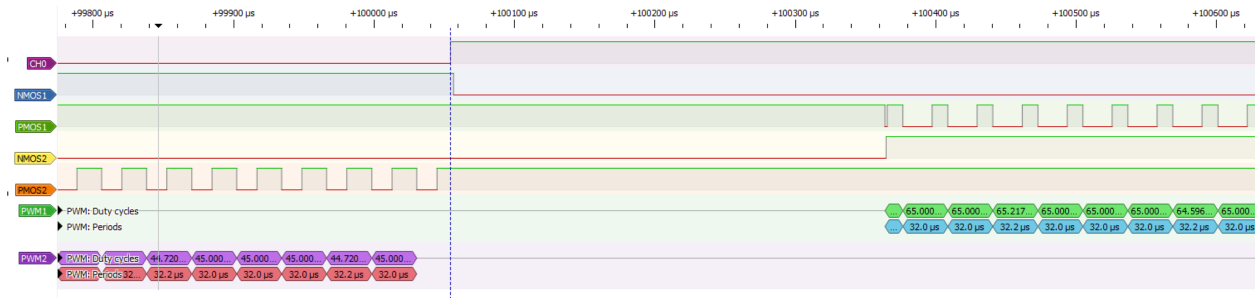


Figure 16 Motor driver initial tests

Due to the prototype motor driver PCB not functioning, not every aspect of the motor driver was able to be validated. It was possible to use a multimeter to validate that the correct outputs were being turned on for each direction in a similar way to the previous figure. The circuit was unfortunately not configured correctly to allow these control signals to do anything. After modifying the circuit to allow one direction, it was possible to validate the ability to turn the motor on and off, as well as modify the speed. Originally, the PWM signal was being sent at 31Khz, but this proved way too fast to allow the BJT transistor that enabled the PMOS side of the H-bridge to fully switch. The frequency was dropped to around 500hz which allowed for full speed control of the motor. Once the motor driver design errors are corrected, it will need to be retested with the firmware to confirm functionality.

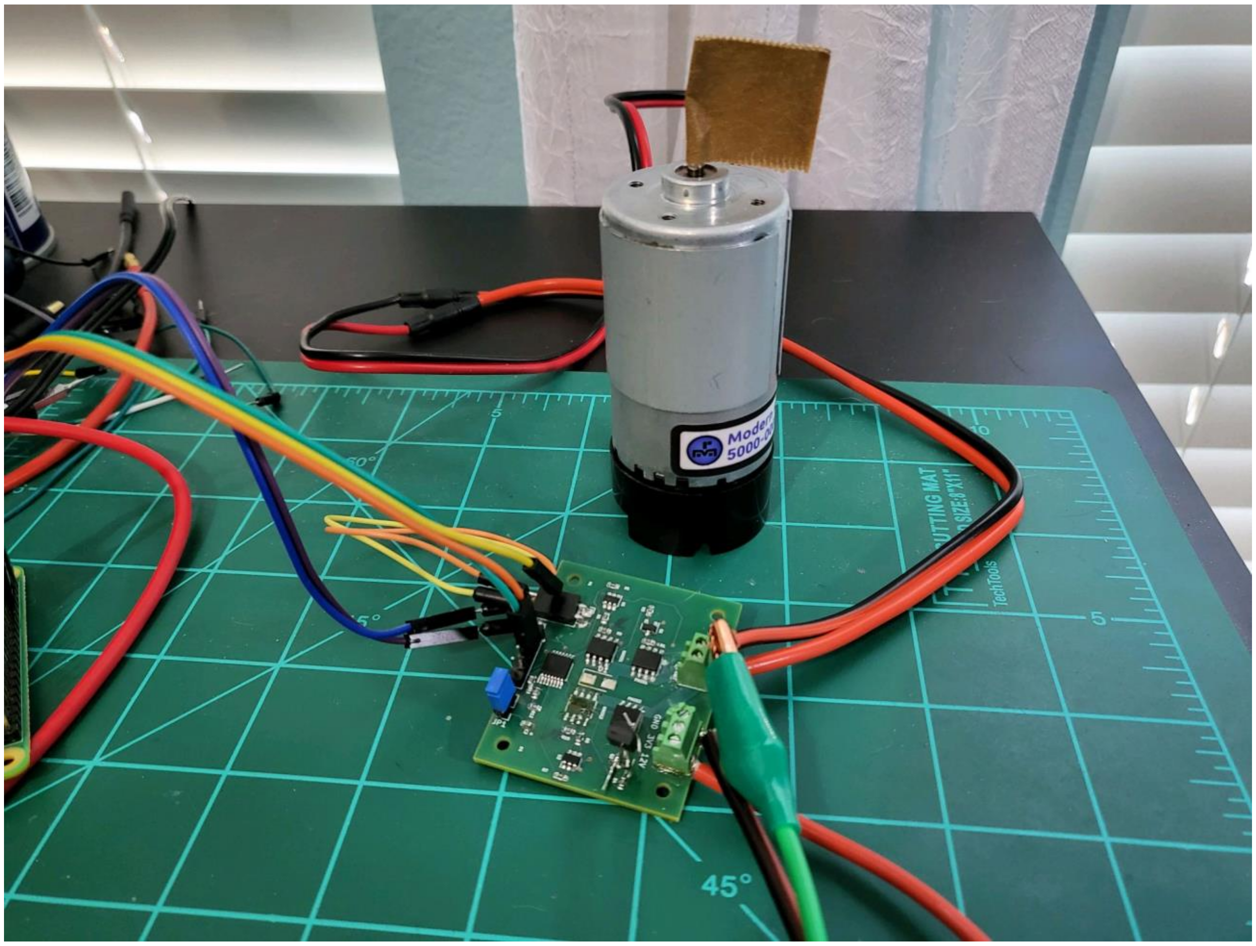


Figure 17 Motor driver test setup

3.4. Subsystem Conclusion

The Raspberry Pi and Microcontroller subsystem was able to successfully complete nearly all required tasks. UART communication between multiple microcontrollers at once was successfully tested, as well as the ability to send and receive commands and data, which will be vital for the final integration. The sensor module is able to successfully measure distance to an object and then send this data to the Raspberry Pi when requested. The motor driver firmware is able send the correct control signals to control the H-bridge of the motor driver, unfortunately, the motor driver design does not currently work, so full functionality can not be confirmed.

4. Housing and PCBs

4.1. Subsystem Introduction

4.1.1. Housing Subsystem

The purpose of the housing is to contain and provide mounting for all hardware of each module. It also provides a means of connecting modules together and preventing separation when in use. There is a housing design for each module.

4.1.2. Housing Subsystem

This subsystem includes PCBs for the sensor module, arm module, and connectors.

4.2. Subsystem Details

4.2.1. Housing Subsystem:

The basic design for all housing is a cube with an empty cube-shaped interior for storing and mounting any hardware. All housing designs are equipped with at least on face for connecting to another module. The connecting face is an octagonal pattern of small stubs protruding out and holes. When connecting two modules, these stubs slot into the holes of the opposing face to prevent any lateral movement. Magnets are used on the interior side of a connecting face to keep the modules together when in use. One side is screwed on separately to the housing, so that the interior is accessible.

Table 3 Housing General Dimensions

Exterior Side Length	Interior Side Length	Wall Thickness	Exterior Edge Fillet Radius
4.5 inches	4 inches	0.25 inches	0.25 inches

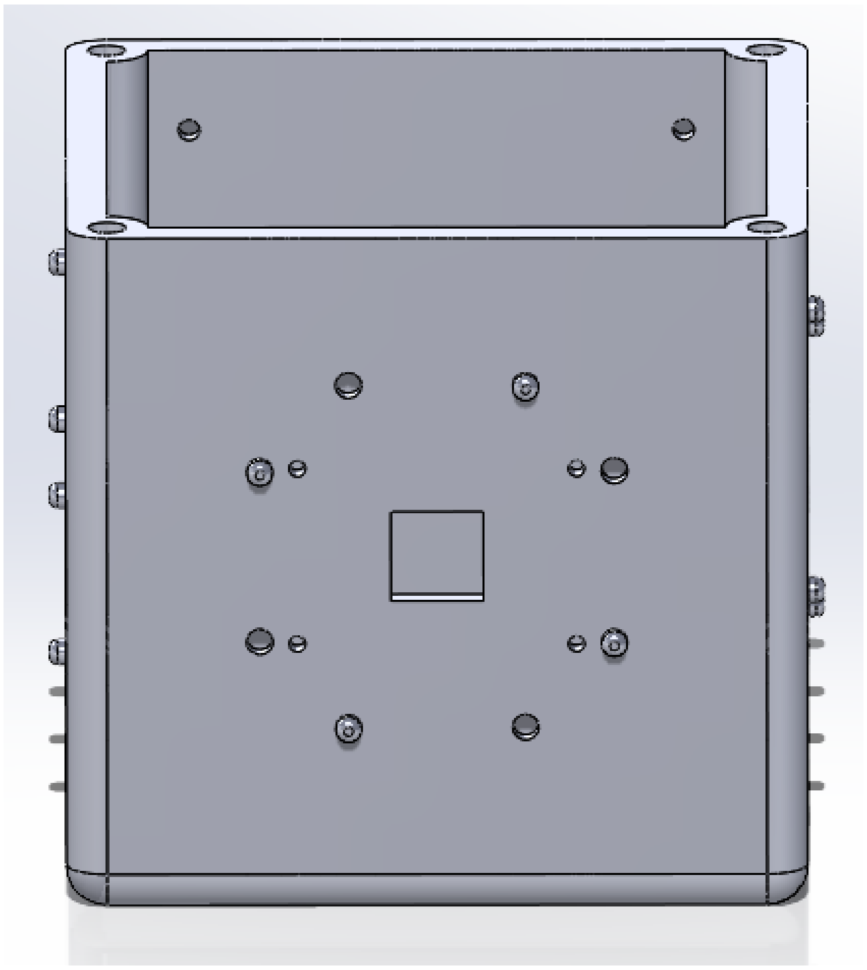


Figure 18 Control Module

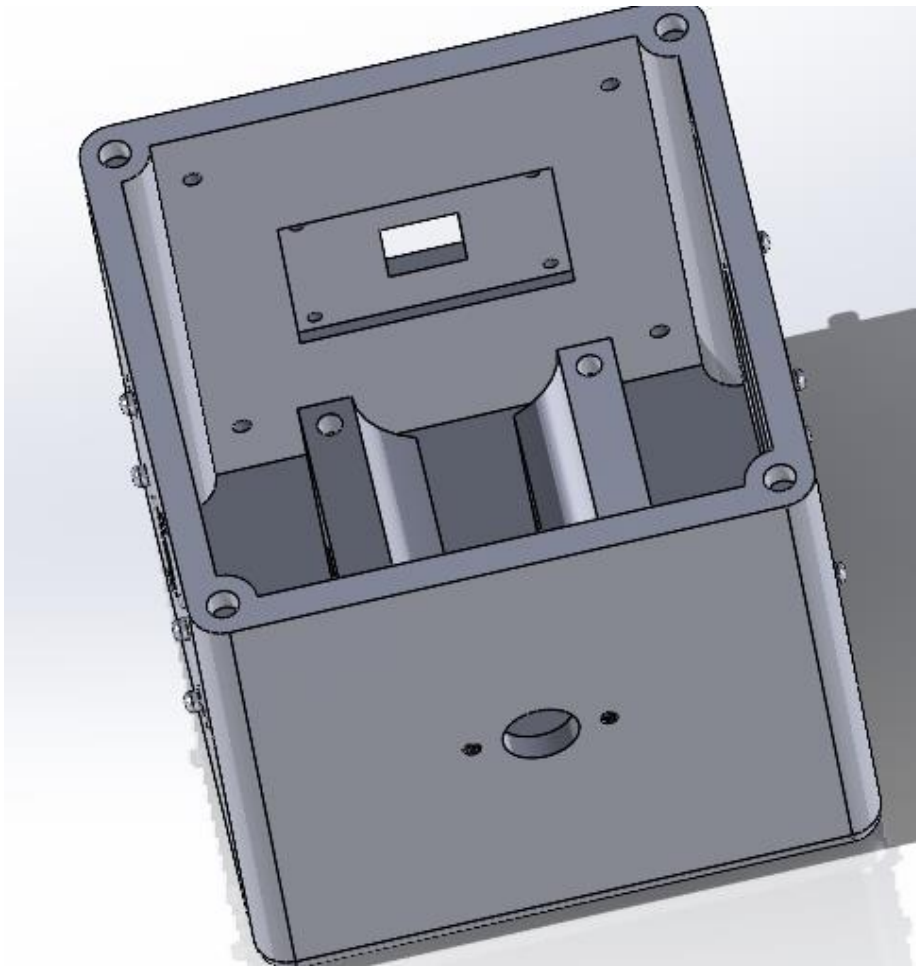


Figure 19 Motor Module

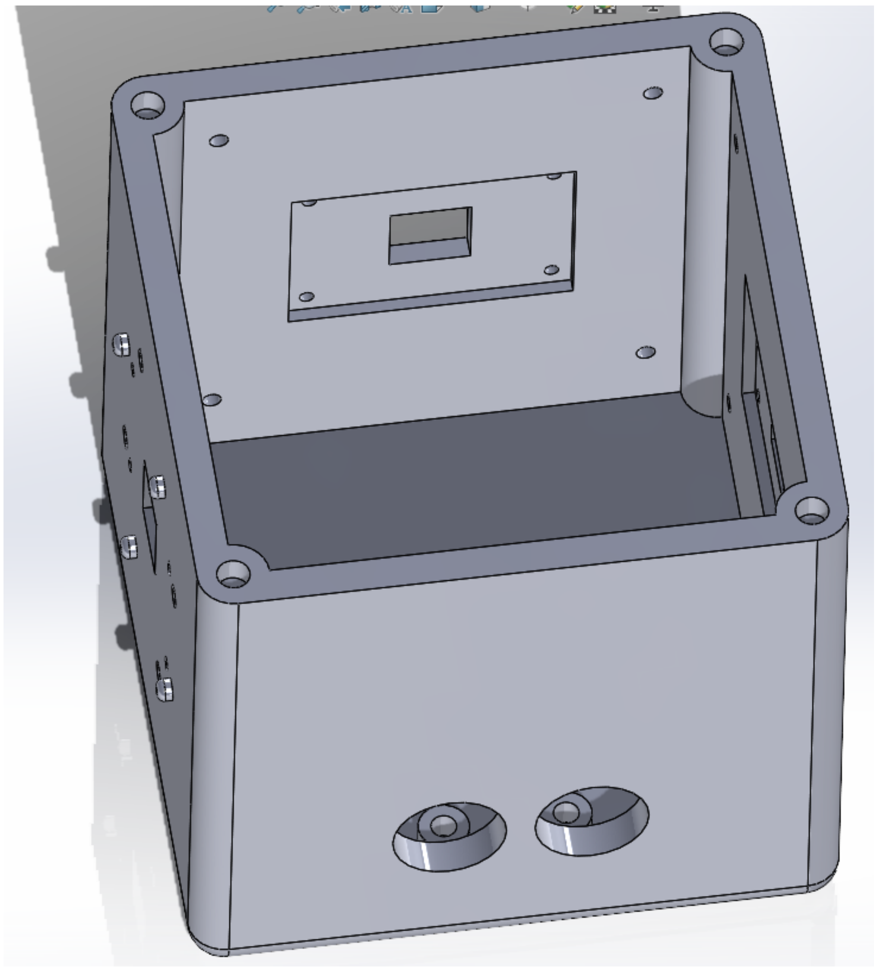


Figure 20 Sensor Module

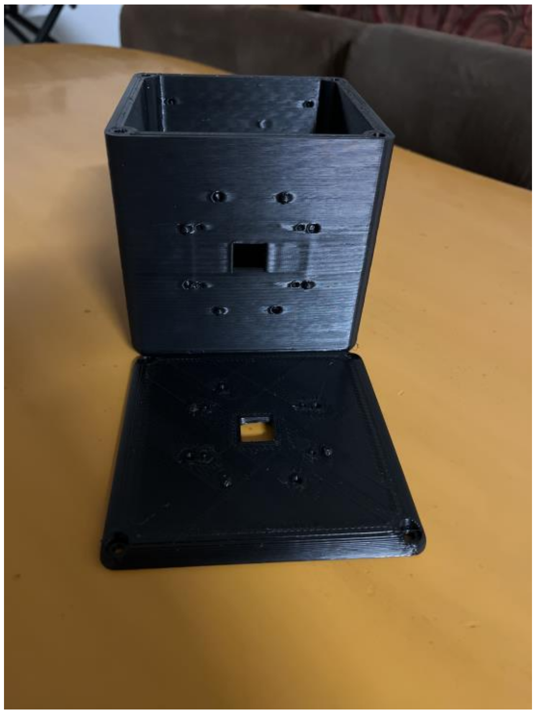


Figure 21 Printed Housing 1

4.2.2. PCB Subsystem:

4.2.2.1. Sensor

The sensor PCB primarily consists of a microcontroller, 2 level shifters, and a 4-pin connector, J4, for mounting an ultrasonic sensor. The microcontroller, PIC16F15224-E/ST, requires 3.3V power, and the sensor, HC-SR04, requires 5V power. The level shifter is used for shifting logic levels from 3.3V to 5V for good communication between the microcontroller and the sensor.

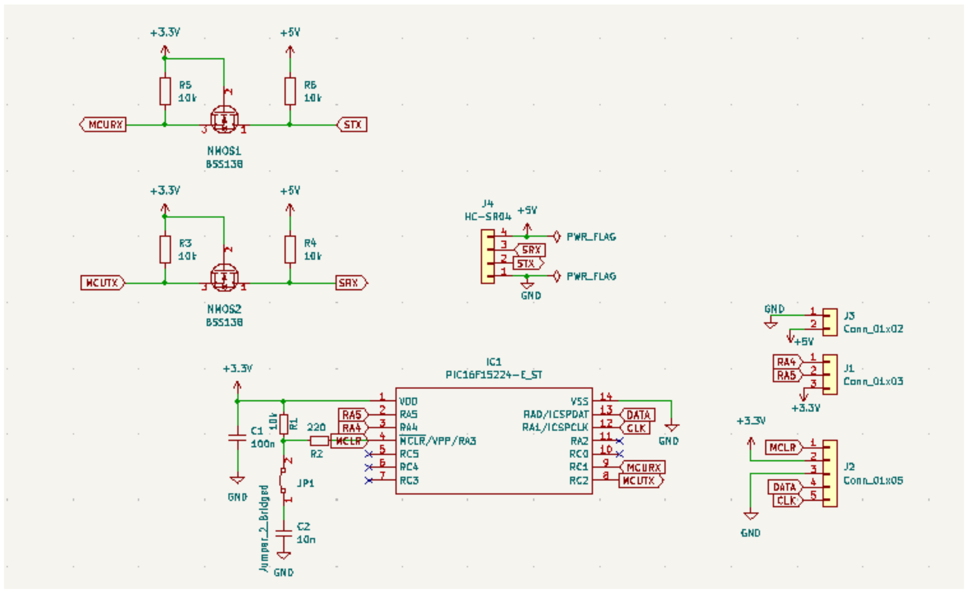


Figure 22 Sensor Schematic

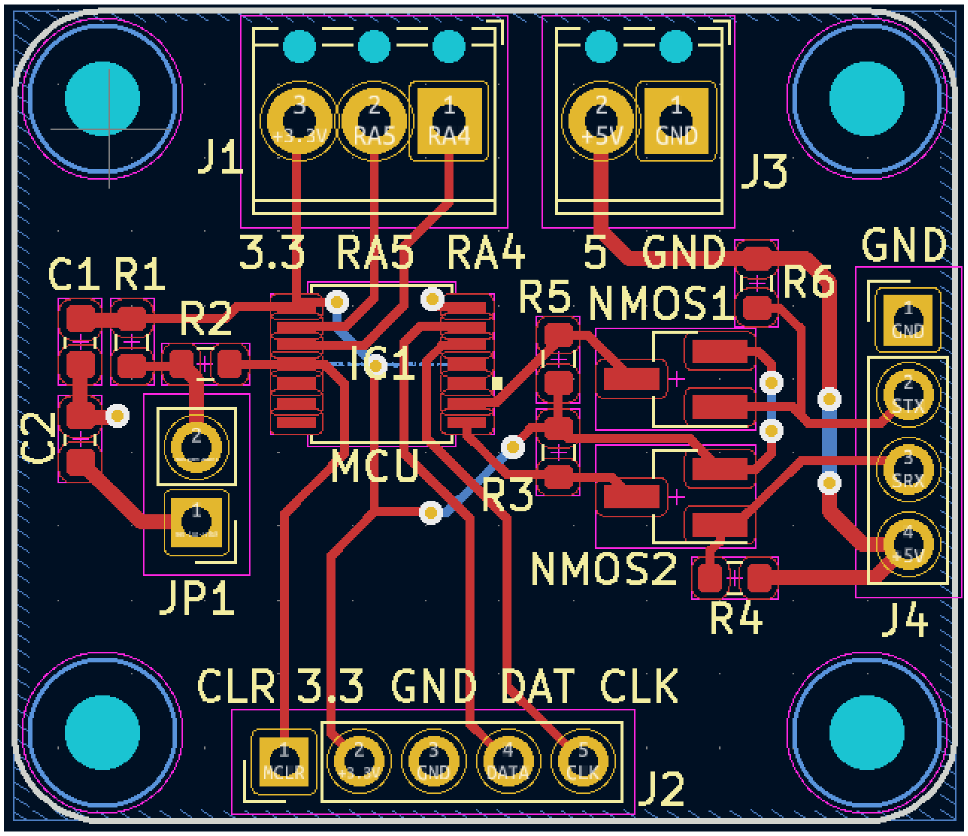


Figure 23 Sensor PCB Design

The 5-pin connector, J2, is used only for programming the microcontroller when not in use. When programming, jumper JP1 must first be disconnected. When in use, J1, J3, and JP1 will be connected.

Table 4 Servo Circuit Components

Component	Value
IC1	
NMOS1, NMOS2	
R1, R3, R4, R5, R6	10 kΩ
R2	220 Ω
C1	100 nF
C2	10 nF

Arm

After the demonstration, it was determined that a PCB design would be necessary for the arm subsystem. This design I very similar to that of the sensor and consists of a microcontroller, 3 level shifters, and 3 3-pin connectors, for connecting servomotors. The microcontroller, PIC16F15224-E/ST, requires 3.3V power, and the servos require 5V power. The level shifter is used for shifting logic levels from 3.3V to 5V for accurate control of the servos by the microcontroller.

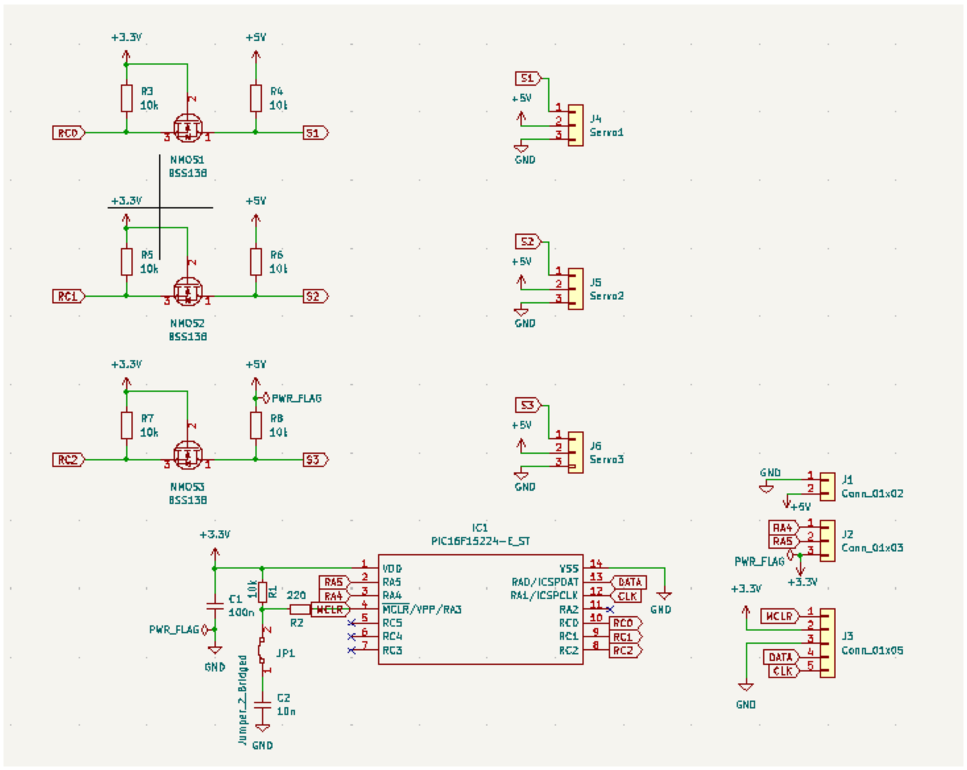


Figure 24 Arm Schematic

Table 5 Arm Circuit Components

Component	Value
IC1	
NMOS1, NMOS2	
R1, R3, R4, R5, R6, R7, R8	10 kΩ
R2	220 Ω
C1	100 nF
C2	10 nF

The 5-pin connector, J3, is used only for programming the microcontroller when not in use. When programming, jumper JP1 must first be disconnected. When in use, J1, J2, and JP1 will be connected

Connector

The connector is a very simple design with the sole purpose of providing electrical connections between modules, and for sending signals to the correct components. These signals include 3.3V, 5V, 12V, GND, UART RX & TX, and USB 2.0 D+ & D-.

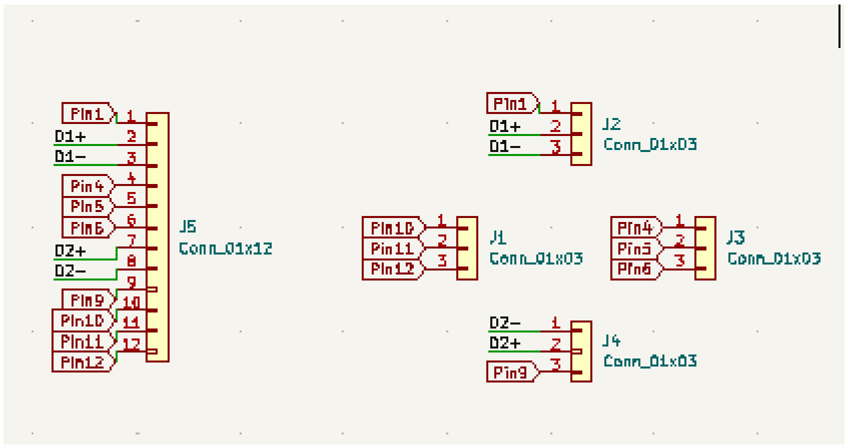


Figure 25 Connector Schematic

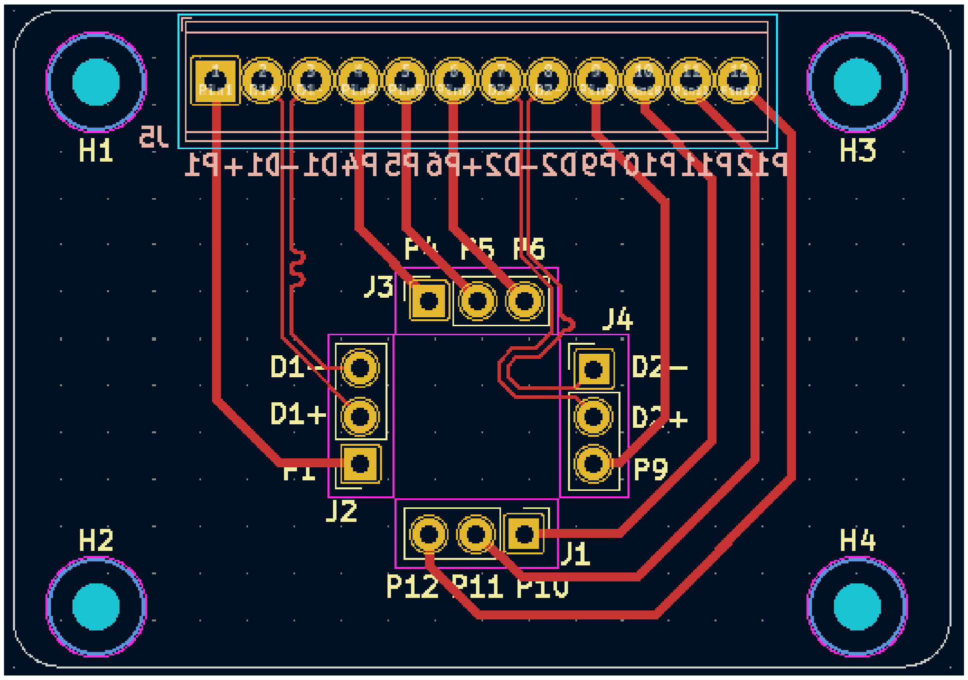


Figure 26 Connector PCB Design

The current layout of this design is 4 3-pin connectors arrayed in a square with alternating male and female pattern. The intent of this is to allow for modules to be connected in different orientations. These pins connect to a 12-pin connector on the opposite side of the PCB, to then be routed to parts inside a module.

4.3. Subsystem Validation

4.3.1. Housing/Connector

The housing subsystem could only be partially validated. The initial print did not turn out very well and there were several issues. Most holes were not large enough for screws or heat-set threaded inserts, and the stubs on the connecting faces were deformed.
I put 2 of the connector designs on perfboard and attached them to connecting faces on the housing. I was able to successfully connect them together in that way, however it was not flush due to the issue with the deformed stubs.

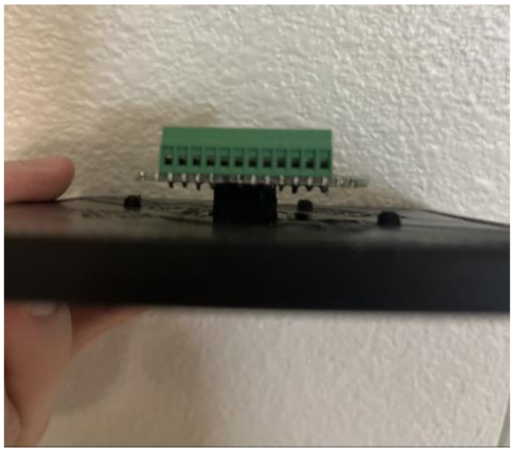


Figure 27 Housing/Connector 1

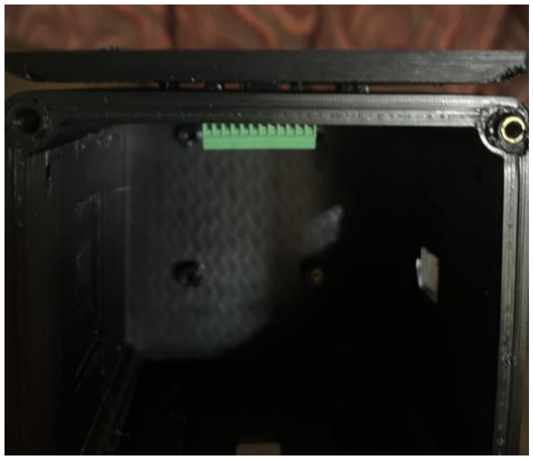


Figure 28 Housing/Connector 2

4.3.2. Sensor

The sensor PCB design was initially tested on perfboard by probing for continuity and checking correct voltage levels. Once the microcontroller was programmed by Andrew, and equipped with the HC-SR04 sensor, it was tested and shown to work correctly and accurately during the demonstration. This was done by placing your hand in front of the sensor next to a measuring tape and reading the distance readout on the computer. Unfortunately, I do not have an image from the demo, and Andrew has the sensor. One thing that Dr. Lusher pointed out was that I was not using a crystal oscillator in the design and was instead using the one built into the microcontroller. This could be a potential issue in the future and may be an improvement to the design.

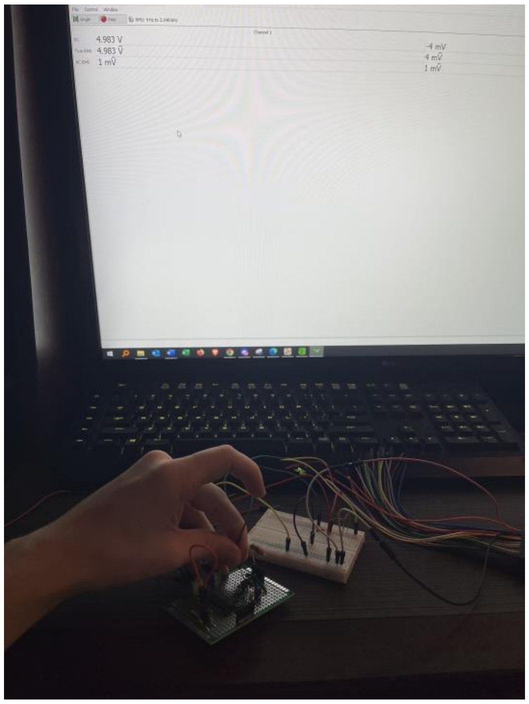


Figure 29 Sensor Design Probing

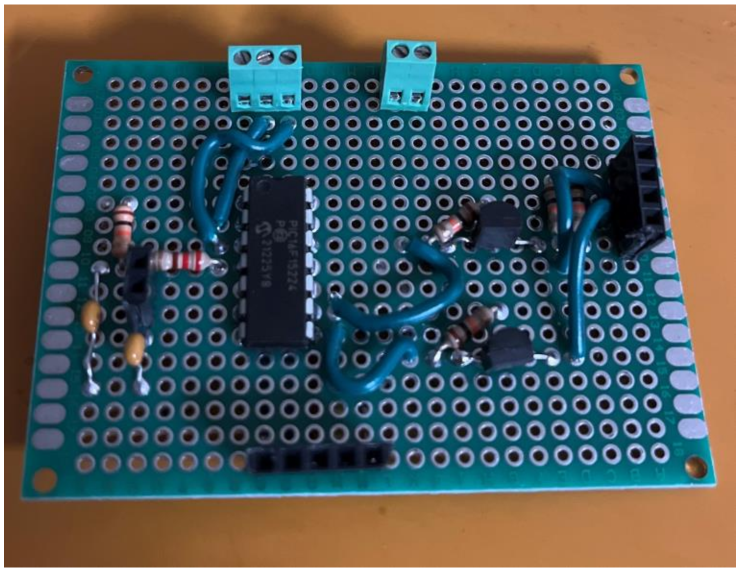


Figure 30 Sensor Design on Perfboard

4.4. Failure Analysis of Subsystem

4.4.1. Housing/Connector

As was stated in the validation, the print of the housing had some deformation issues preventing flush connections. Another part was printed, and it came out much more accurate, however it still had the issue with hole sizes being too small. The stubs held shape more consistently, however about 2 came out slightly warped. With this 2nd print the connecting faces were almost flush but not quite. This could be corrected by replacing the stubs with holes for metal rods to be inserted and heat set, however given that there was only slight deformation of the stubs on the second print it may be better to simply increase the hole size for the stubs altogether. The insert holes for the magnets were too small as well, so they were increased by heating the plastic. The magnets I had were quite small and ended up being too weak, so I will be using larger and possibly more magnets.

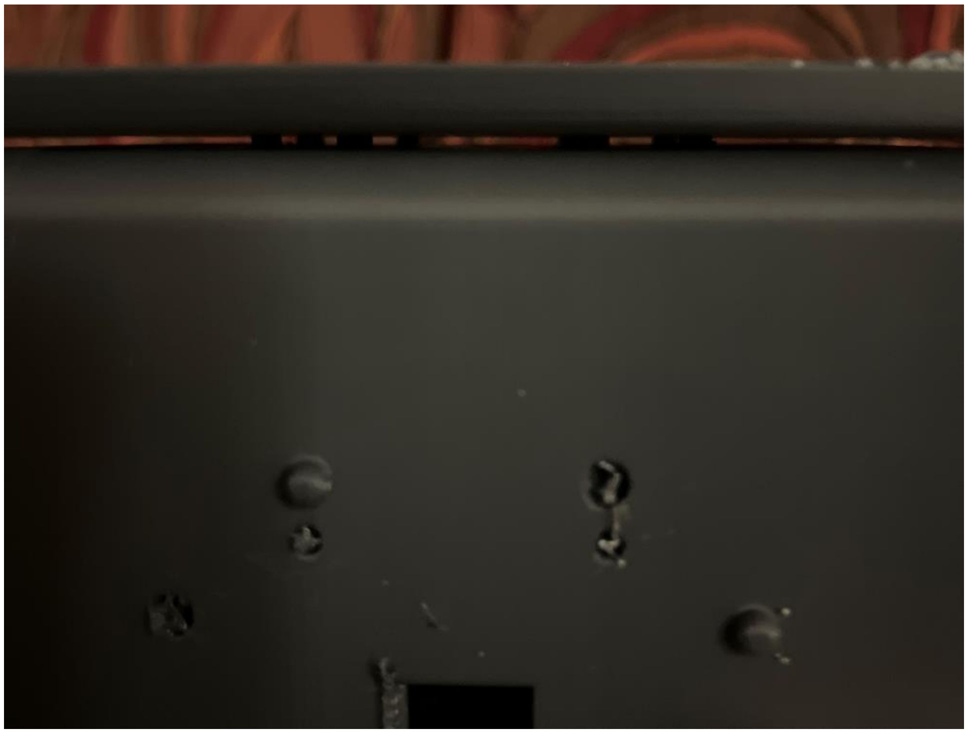


Figure 31 Printed Housing 2

5. Object Detection and Gripper Arm

5.1. Subsystem Introductions

Here lies a detailed report and analysis of the subsystems of which this section consists. Those subsystems are the Object Detection subsystem and Gripper Arm subsystem which are explained as follows and all files relating to the following subsystems are included in the Github repository inside their respective folders.

5.1.1. Object Detection Subsystem Introduction

The purpose of this subsystem is to utilize the webcam shown in the figure below to input video data of what is essentially in front of the robot and process it. The video is going to feed the raspberry pi through a USB connection. The object detection subsystem then identifies all objects inside the frame of the webcam's line of sight and sends those data to determine the next action.

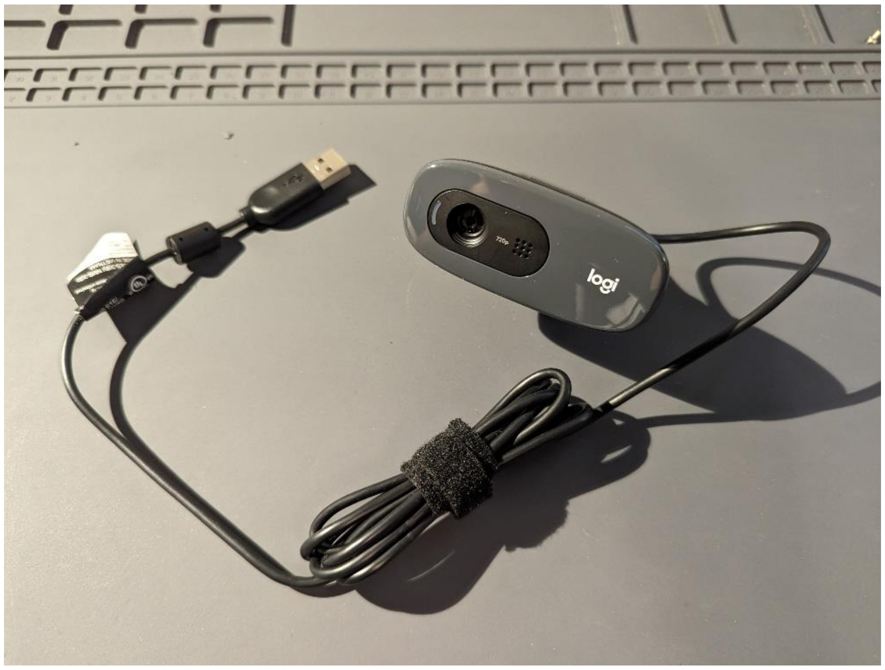


Figure 32: Webcam Interface Used.

5.1.2. Gripper Arm Subsystem Introduction

The purpose of this subsystem is to reach over the robot and pick up the desired object once identified by the object detection subsystem. The arm contains three main servo motors that control its movements. Two servo motors at the base of the arm for multiple reasons are discussed in the later sections, and one servo at the gripper head.

5.2. Subsystem Details

5.2.1. Object Detection Subsystem Details

The program which processes the live feed from the webcam and identifies objects in frame real-time is written using python and utilizes the libraries of OpenCV and YOLO (You Only Look Once). Initially, the program at early stages used to only able to process saved videos and output the results in another video with overlays. Then through multiple updates and modifications now the program is capable of processing while the video data is being transmitted.

The program's object detection accuracy is tied to its processing speed, and the ratio is inversely proportional meaning that if the accuracy is increased the program ends up being slower and vice versa. Thus, two versions of this subsystem were created each specialized in one aspect, one had superior accuracy, and the other outputted result as fast as the webcam could capture and stream.

During presentations and demonstrations, the program was made to present its results and show what it can identify for visual purposes, as shown in the figure below, which also proved detrimental to its processing speed. Since the robot should be able to coordinate and move accordingly on its own, the robot does not need a video stream of what it sees. Rather the robot requires only information and said coordinates to act on. Having said that, the script was modified once again at this point to be able to process everything without needing to show, and the script was tested heedlessly. The program also is capable of outputting its result in a fully customizable manner that suits the main controller of the robot and determining which form is a task for 404. As of now, the program can output to the console all the data it can extract including but not limited to the name of the object, the center coordinate of the object, and the accuracy of the object's identification.

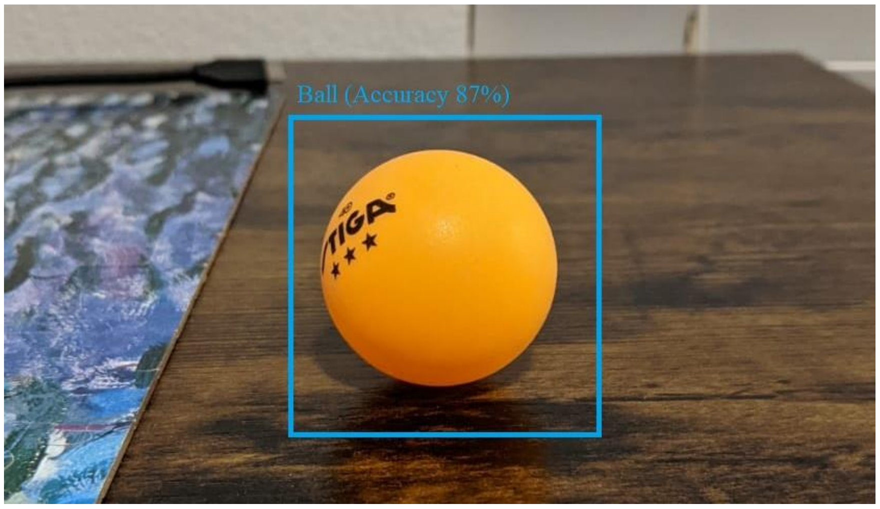


Figure 33: Representation of How Object Detection Looks Like.

5.2.2. Gripper Arm Subsystem Details

The arm is made to be lightweight to minimize the tension on the base the arm extends over the robot while having a sturdy base to support such extension. The first reason for having two heavy servos at the base is to add weight and support the base so it can withstand the full extension of the arm during task completion. The second reason for having two servos is to be able to fully control the arm’s movement and customize where exactly should the arm go. One servo controls the vertical movements and points the gripper toward the desired altitude, and the second servo pushes the gripper forward and backward controlling the horizontal movements. On the other hand, the gripper head’s servo is specifically chosen to be a small light servo to reduce the tension and the weight on the arm. Since no heavy tasks like lifting the entire arm are required from this servo, choosing a smaller one works perfectly.

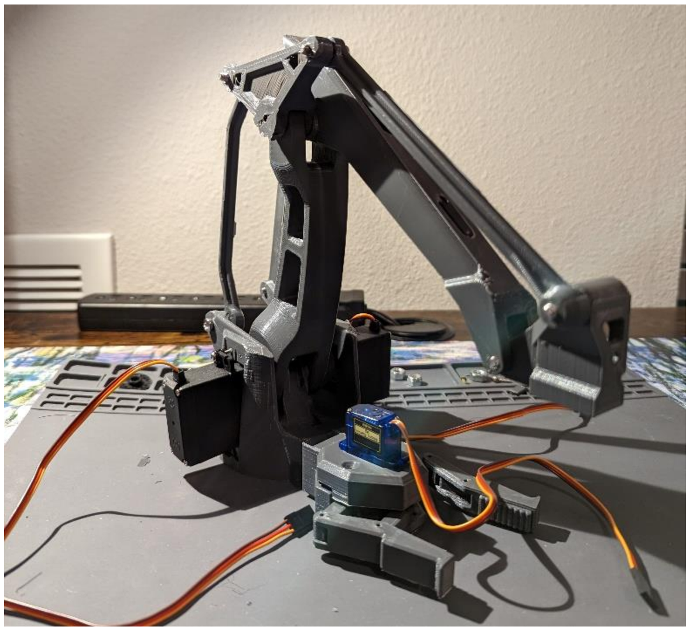


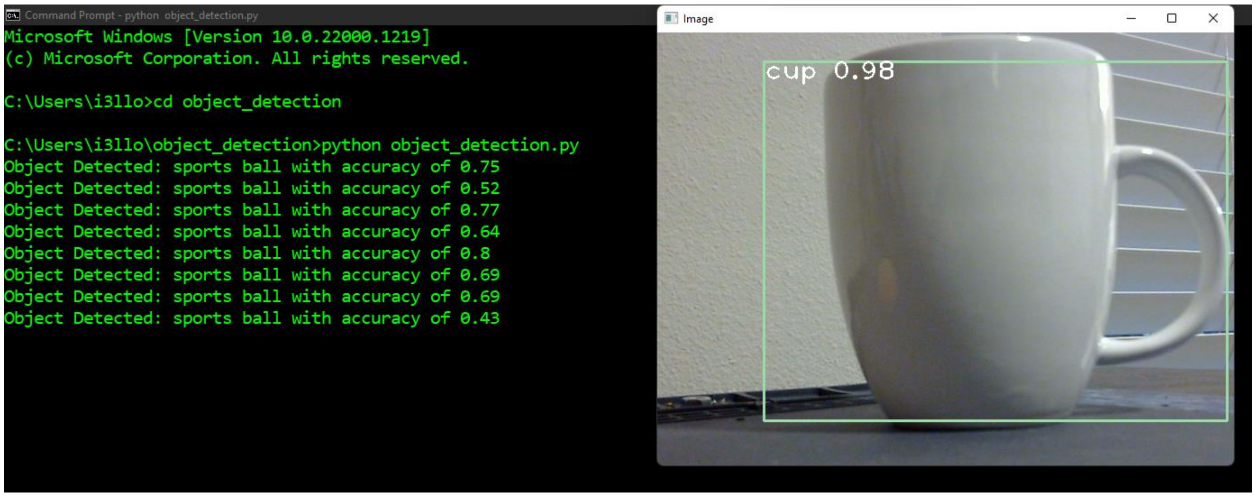
Figure 34: Gripper Arm Demonstration.

As shown in the figure above there are two supporting links in the back of the main arm and each has a purpose. The straight backlink is used to control the vertical position of the second arm, in addition, to support. The curved backlink however is, in contrast to the first backlink, connected to the base itself instead of the moving parts of the arm. This link’s purpose is to limit and control the orientation of the triangular joint which is responsible for stabilizing the gripper head and ensuring that it always points to a certain direction regardless of the arm’s positioning. The gripper head is connected to the main arm to control its position and is connected to the triangular joint through a straight link to determine where it points. Technically there are two parallel arms, however only one carries the weight and the other only consists of thin links whose sole purpose is to stabilize and support the main arm.

5.3. Subsystem Validation

5.3.1. Object Detection Subsystem Validation

The goal of this subsystem is to be able to provide the main controller with sufficient information to determine where the robot should go and what objects it should pursue, and for that, we do not need the object detection program to show anything. Once the program was not needed to present and eliminated the show feature, its processing speed was upped by almost double, and it did not use as much memory which is to be expected. In addition, the program proved to be capable of outputting all sorts of data in various forms, some of which are visual and console print as shown in the figure below. The figure also demonstrates the capabilities of the program where it can focus on a few objects and ignore all others, which is a feature that will be used in 404 as well.



```
Microsoft Windows [Version 10.0.22000.1219]
(c) Microsoft Corporation. All rights reserved.

C:\Users\i3llo>cd object_detection

C:\Users\i3llo\object_detection>python object_detection.py
Object Detected: sports ball with accuracy of 0.75
Object Detected: sports ball with accuracy of 0.52
Object Detected: sports ball with accuracy of 0.77
Object Detected: sports ball with accuracy of 0.64
Object Detected: sports ball with accuracy of 0.8
Object Detected: sports ball with accuracy of 0.69
Object Detected: sports ball with accuracy of 0.69
Object Detected: sports ball with accuracy of 0.43
```

Figure 35: Both Forms of Results Output.

5.3.2. Gripper Arm Subsystem Validation

For this subsystem to be able to move as intended, it needs to go to 404 and be integrated with the other subsystems for power, control signals, and housing. However, as of now the gripper arm, fortunately, has a PCB tailored for its specific usages made by the PCBs subsystem. The arm's PCB will allow for the power the arm needs to flow and provide the connection so that the main controller can send signals and control its movements. Nonetheless, for validation purposes, the arm proved that it could move through its full range of motion, in addition, simple movements controlled by an Arduino were shown to the professor during demonstrations. Two example positions of the arm were captured and presented in the figures below for clarification. One note regarding the base servo motors is that one of them is 1-2mm too far out. The servo still connects to the gear and moves properly, however, just to be safe and more secure the servo should go deeper and fully locks in place.

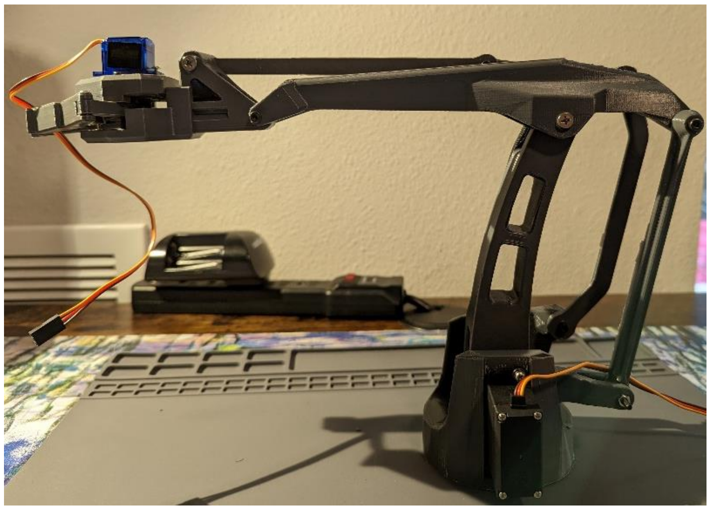


Figure 36: Gripper Arm Example Position 1.

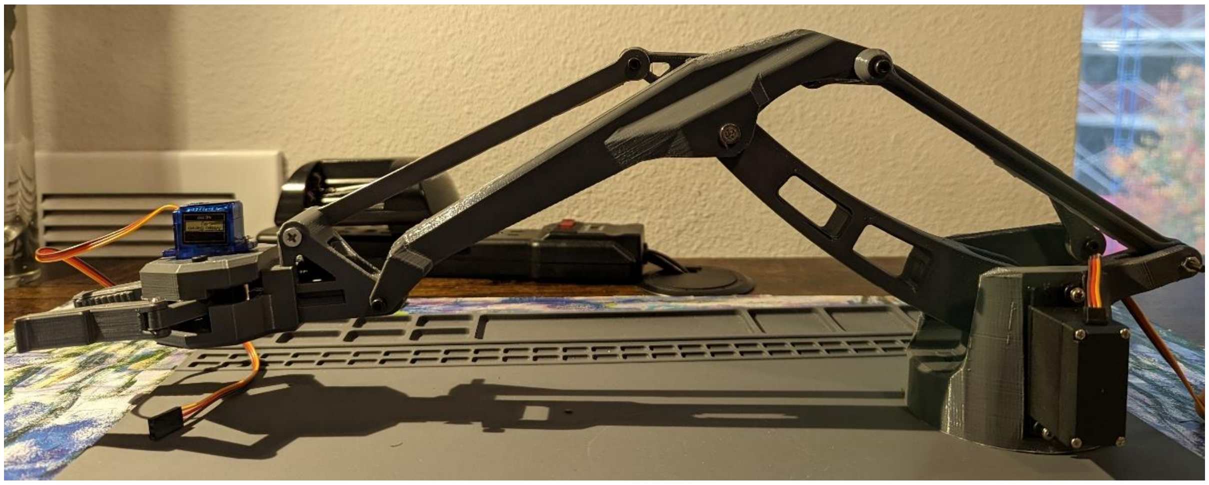


Figure 37: Gripper Arm Example Position 2.

5.4. Subsystem Conclusions

Both parts of this subsystems have been completed and validated accordingly, both subsystems produced appropriate results. As of this stage and the state of both subsystems, both of them proved readiness to go into 404 with minor expected modifications. Modifications such as for the object detection subsystem would be tailoring the output results to however fits the main controller. Additionally, the base of the gripper arm subsystem will be integrated with the housing subsystem and is expected to be adjusted to fit with the module, on top of the servo position adjustments.