**CPEN 291   
Project 1 Report**

**A. Group info**

Lab section: L2A (*L2A or L2B)* Group #: A-G4 (*Example: A-G1*)   
Group’s Lab Bench #s: 4 and 9 (*Example: 1 and 2*)   
Student names:

|  |  |
| --- | --- |
| Andrea Shao | Matthew Chow |
| Brielle Law | Matthew Stefansson |
| Sebastian Gonzalez | Matthew Yen |

**B. Technical documentation for the main functionality**

1. Hardware

We considered three configurations for the reflective optical sensors:

* **Option A**: Three reflective optical sensors aligned side-by-side, located on the lower platform at the front of the robot. The two outer sensors are separated by the distance of the black electrical tape (approx. 2cm) and the third sensor is centered in between.
* **Option B**: Five reflective optical sensors aligned side-by-side, located on the lower platform at the front of the robot. Each sensor is separated evenly by half of the distance of the electrical tape (approx. 1cm). The two additional sensors were considered to maximize the smoothness.
* **Option C:** Four reflective optical sensors aligned in a triangular formation. A single sensor is placed 3cm in front of the remaining three sensors aligned side-by-side separated by the distance of the black electrical tape (approx. 2cm). The positional offset of the sensor is to account for the expected 3cm gaps in the track. This would prevent us from having to track the time required to travel 3cm.

We initially decided to implement Option B, making use of the 5 reflective optical sensors. We made this decision to maximize accuracy and to allow for greater error when searching for the black line. We were concerned that using only 3 sensors would result in continuous back and forth swerving.

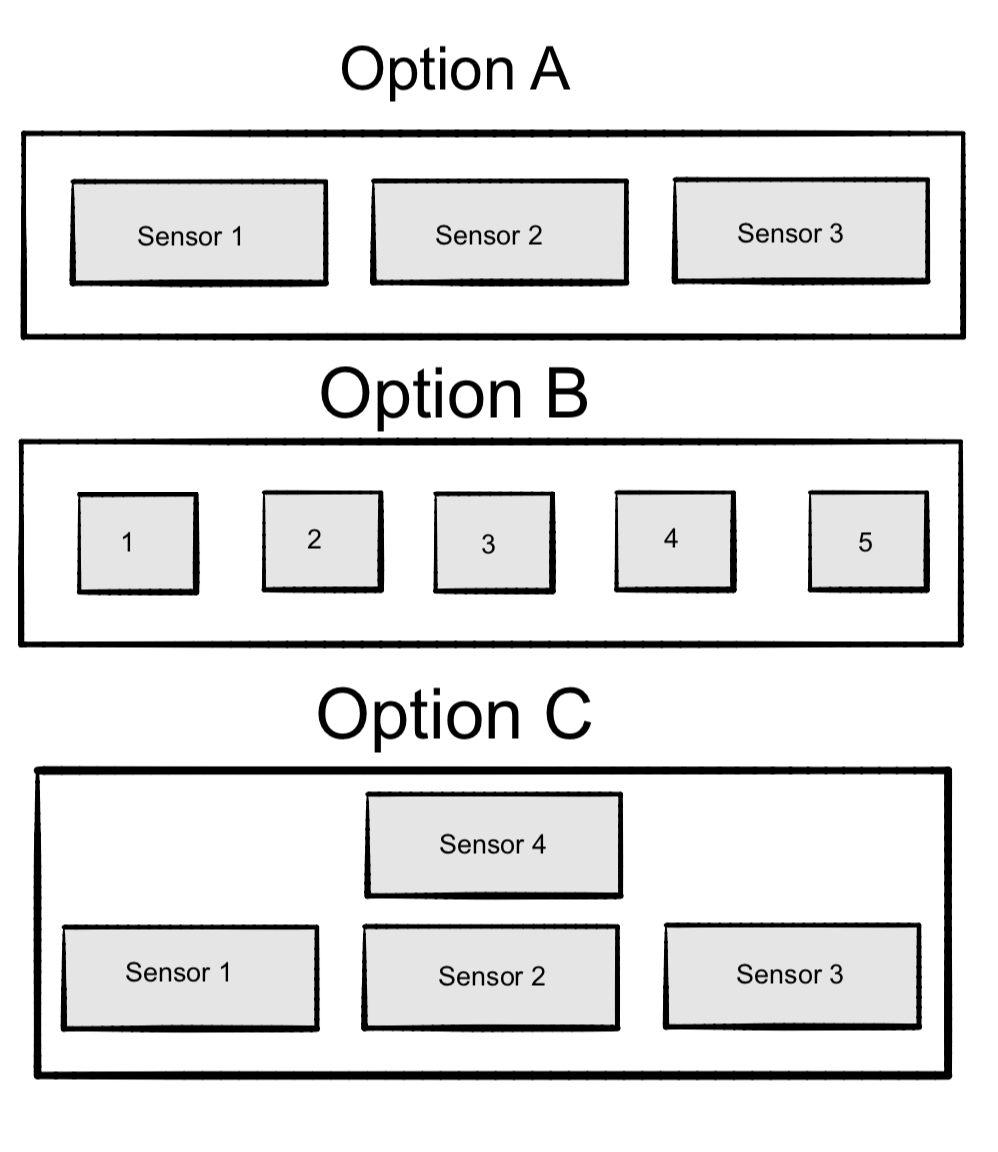
Challenges:

After implementing Option B one of the optical sensors was giving inconsistent results. This single sensor would unreliably return the expected value (1 for black and 0 for white). Thus, to ensure we could achieve the main functionality of following the line, we decided to simplify our design to Option A. Using three optical sensors simplified the PID controller which made it easier to fine tune and it is why our robot could follow the line well. Unfortunately because we used 3 sensors and not 5 our robot could not do 90 degree turns. We concluded that this was the optimal decision, because the implementation of the PID controller would provide more reliable performance compared to the the 5 sensor option.

When adding the additional features, getting the stream to work on the app was very time-consuming as finding out exactly how to get a bitmap image to appear on imageview proved troublesome. The integration of the remote and socket components turnout out to be more difficult than we expected and we faced the challenge of a slow stream which caused the whole project to work improperly, which can be attributed to the hardware but also to our own inexperience with the software as well.

1. Line Following Algorithm

When considering which hardware configuration to choose, we also took into account their corresponding line following algorithms.

* **Option A:** (Corresponding to 3 Reflective Optical Sensors) If the middle sensor reads 1, the robot should continue on a forward path. If the left sensor reads 1, the robot needs to correct its path by veering left. If the right sensor reads 1, the robot needs to correct its path by veering right.
* **Option B:** (Corresponding to 5 Reflective Optical Sensors) If the middle sensor reads 1, the robot should continue on a forward path. If the two center right sensors read 1, the robot needs to correct its path by veering a *slight* left. If the outer left sensor reads 1, the robot needs to correct its path with a larger left turn. The equivalent conditions for the right sensors hold true.
* **Option C:** (Corresponding to 4 Reflective Optical Sensors) For option C sensors one, two, and three are positioned the same in option A. Similarly to Option A if sensor one reads on the robot should veer left. If Sensor 2 reads one it should continue on the forward path. If sensor 3 reads one it should veer right. The reason why option C is different from option A is the 4th sensor. Adding the fourth sensor allows us to know when the robot should stop. Sensor 4 is 3cm in front of sensor 2. If sensor 2 and 4 read zero the robot should stop.

Implementation of the PID controller

* Both Option A and Option B require similar PID controller implementations. When calculating the PID we first read from the optical sensors to determine the error. Each possible combination of triggered sensors correlate to an error value. As we are using 3 sensors, we created an array of size 3 where each element corresponds to a specific sensor. If only the middle sensor sees the electrical tape, it will read 1, meaning the robot is travelling on the desired path, and hence has an error of 0. For all possible combinations (000, 001, 011, 010, 110, 100, 111), an appropriate error value was calculated. After calculating the error value the PID is calculated based upon the formula below. Then, the current error is set to the variable prevError. If the PID is negative it turns the robot left and if the PID is positive it turns the robot right. All of the PID functions (getOptics, calculateError, calculatePID) were written in a class named Error.

The different error combinations were calculated as follows:

if (errorTotal == 1): *# right sensor triggered*

self.error = -1.7

elif (errorTotal == 11): *# middle and right sensors triggered*

self.error = -1

elif (errorTotal == 10): *# middle sensor triggered*

self.error = 0

elif (errorTotal == 110): *# middle and left sensors triggered*

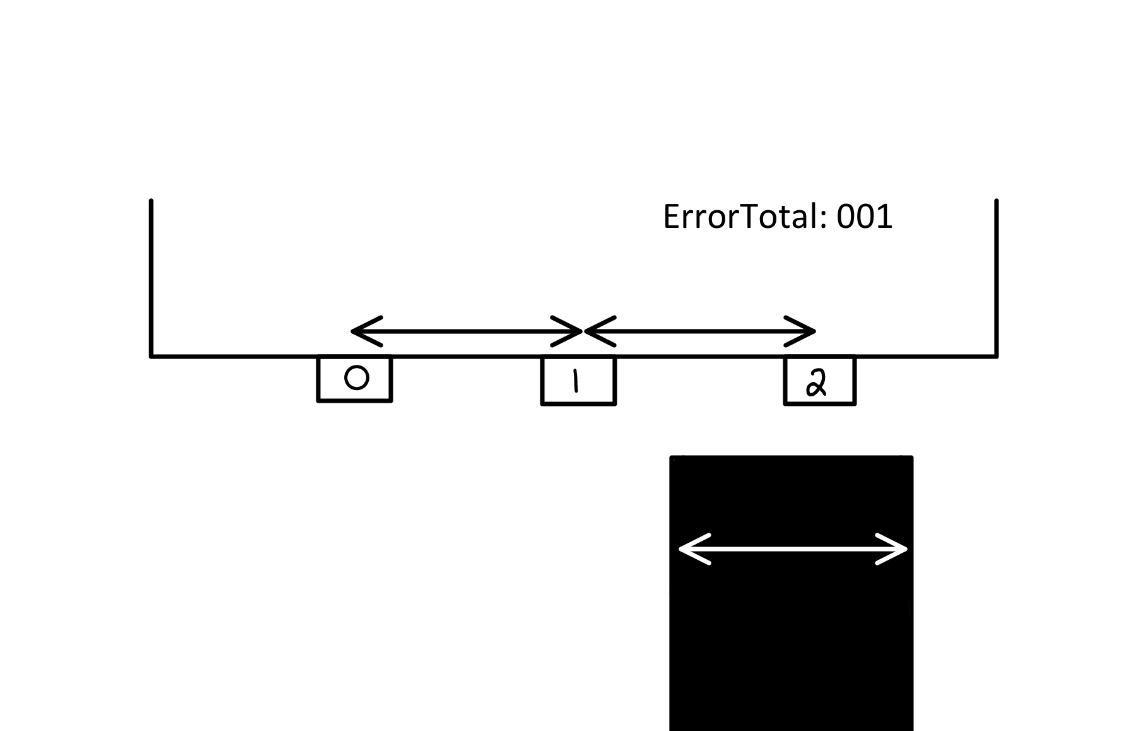
self.error = 1

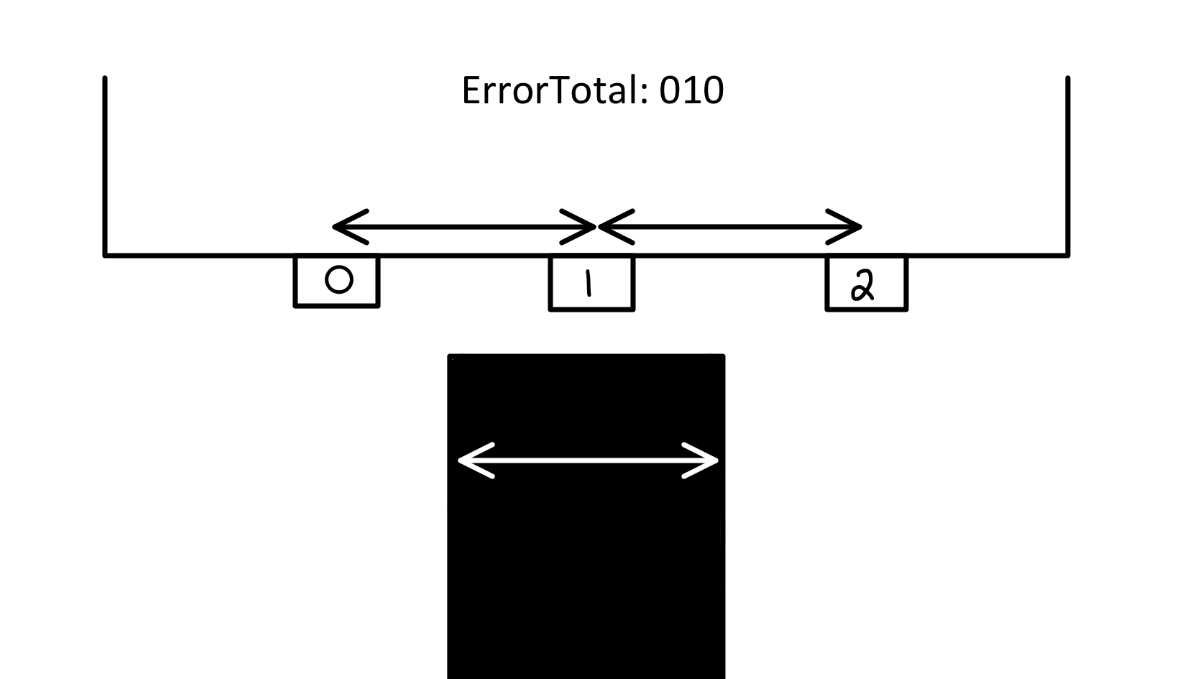
elif (errorTotal == 100): *# left sensor triggered*

self.error = 1.7

elif (errorTotal == 111): *# all sensors triggered, most likely crossover*

self.error = 0

ErrorTotal visualized below: (black strip represents electrical tape, three sensors labeled 0-2)



The motor’s speed is set in the code below:

kit.motor1.throttle = 0.25 + PID *#Left motor*

kit.motor2.throttle = 0.25 - PID *#Right motor*

pidValue = self.Kp \* self.error + self.Kd \* (self.error - self.prevError) + self.Ki \* self.integral

In addition to keeping the robot on track, we had to consider how our robot would approach 3cm gaps and ensuring that it would stop at the end of the track. To do this, we considered a number of options like timing how long it took for the robot to go for 3cm and using the time function to check whether or not a certain time had passed before stopping the robot. We also considered using a count to In the end, we decided to add a count to our Error class and if the error was 000, we would increment the counter. In our control loop, we check if the error has reached a certain number before stopping both motors and writing our stop image to the LCD. The number 25 was determined through experimentation as we had to ensure that we passed through 3cm gaps, but also gave us enough time to make relatively sharp turns without the robot stopping.

The gap algorithm can be seen in the two code snippets below:

elif (errorTotal == 0):

self.count = self.count + 1

if (error.count == 25):

writeImages("stopGear.jpg")

kit.motor1.throttle = 0.0

kit.motor2.throttle = 0.0

1. Autonomous Robot
   1. **Headless Pi**

The goal of the robot was to have an autonomous robot that could be controlled remotely. To solve this problem we set up VNC on our raspberry pi and connected to the raspberry pi using VNC viewer from RealVNC.

* 1. **Battery-operated Robot Implementation and Challenges**

As the robot needed to run wirelessly we need to find a power source to power the raspberry Pi. To solve this problem we bought a battery pack that could supply power to devices that had USB-C connections like the raspberry Pi. After buying the battery pack we placed it on the back of the robot's chassis as the robot was very front heavy. Adding the battery pack allowed our robot to run autonomously and more smoothly as the robot was now balanced.

For both the use of VNC and battery pack were the first options we used. As both VNC and the battery pack worked well we did not need to use alternative solutions.

**C. Technical documentation for the additional functionality**

1. Android Application

**Sending**

1. Mode - Autonomous vs Remote Control

There are two possible modes for the robot - Autonomous and Remote Control. In Autonomous mode, the PID controller function is invoked such that the robot autonomously follows a black line. In Remote Control mode, the user has motor control over the robot through the app.

Upon launching the app, the robot is set to default in a waiting mode such that when the ‘Start’ button is pushed, the robot will first take a picture and display it in the app. However, the user has access to a toggle switch that will switch the robot to Remote Control mode. The mode can be switched at any point through the use of this toggle switch. To access Autonomous mode, we have to first click start, then hit the toggle twice.

1. Remote Control

When in Remote Control mode, the user is able to utilize the ‘Forward,’ ‘Left,’ ‘Right,’ and ‘Stop’ buttons to direct the robot.

**Implementation:**

A connection between the Raspberry Pi and Android Application is established using sockets. The Pi acts as the server, creates a server port and listens for a client connection. Once a connection is established, commands are sent to the Pi from the Android App in the form of a JSONObject converted to a byte stream containing a ‘Mode’ and ‘Type’ field. The ‘Mode’ field will either contain the string ‘Autonomous’ or ‘Remote’ to indicate the desired mode. The ‘Type’ field will either contain the string ‘Forward’, ‘Left’, ‘Right’, or ‘Stop’ to indicate the desired motor control command. A new JSON Object is created and sent in response to either the toggle switch or buttons being pushed on the app.

Once the Pi receives the input stream, we call a parseJson function that restores the JSON Object from the stream. If the mode field indicates Remote Control, we check the ‘Type’ field to determine whether the Straight(), Left(), Right(), or Stop() functions should be called. These functions change the values of the throttle to move the robot in the desired direction.

Below are examples of how the request is sent through event listeners:

*//Add an event listener for the forwardsButton to call moveForward onclick*

forwardButton.setOnClickListener(new View.OnClickListener() {

**public** **void** onClick(**View** v) {

moveForward();

}

});

*//Send a request*

**public** **void** sendRequest(**String** movement, **String** mode) {

*//Create a new JSONObject*

**JSONObject** request = new JSONObject();

*//Try adding fields to the JSON*

try {

request.put("Type", movement);

request.put("Mode", mode);

*//Send the request out over the socket*

out.print(request);

*//Flush the socket*

out.flush();

}

catch (**Exception** e) {

*//If exception, print stack trace*

e.printStackTrace();

}

}

*//Tell the pi to change modes*

**public** **void** ChangeMode(**boolean** mode) {

*//If toggle is set to remote control send mode for remote*

if(mode) {

sendRequest("NULL", "Remote");

}

*//If toggle is set to autonomous send mode for autonomous*

else {

sendRequest("NULL", "Autonomous");

}

}

*//send a request to turn left*

**public** **void** turnLeft() {

*//if the mode is remote control then send request*

if(modeStatus.getText() == "Mode:Remote Control") {

sendRequest("Left", "Remote");

}

}

**Receiving**

1. Picture

With the use of the PiCamera, we send a picture to be displayed on the Android App. The image is sent from the Pi in the form of byte arrays which are received by the Android App. The byte array are reverted back to images using BitMapFactory class that decodes the input stream and the image is scaled down to reduce loading time. The images are then sent to be displayed on the App interface. Below is how the input stream is decoded into a bitmap to be displayed on the app.

*//Start to listen and serve requests*

**public** **void** mainFunctionality() {

*//Create a bitmapfactory options object*

**BitmapFactory**.**Options** options = new BitmapFactory.Options();

*//Set the scaling to value of 8*

options.inSampleSize = 8;

*//create a new bitmap*

**Bitmap** image = BitmapFactory.decodeStream(in, null, options);

*//set the image as the newly converted bitmap if the image is not null*

if (image != null) {

robotCamera.setImageBitmap(image);

}

}

1. PiCamera
   1. **Livestream Software Information**

While the robot is running regardless of it being in autonomous mode or remote control mode it is live streaming a video from the camera to the Android app. Refer to the above section for greater detail. It is not really a video, but a series of images.

* 1. **Camera hardware implementation**

The camera is attached directly to the raspberry pi. You can see this in the fritzing diagram and the robot images. We use the picamera library and socket to connect to Android app.

1. Component List
   1. Android phone to display Android Application
   2. PiCamera

**D. Test and evaluations**

Throughout mini project one, we mainly focused on integration testing as it was essential for us to be able to synchronize the robot with the buzzer music. While this form of testing proved to be a great choice in assessing the overall capabilities of our project, it also showed us how difficult it was to find small bugs when testing so much code at once. For that reason, we decided to focus on unit testing this time around.

We started off by testing the performance of our various hardware components. We first built our robot from the parts kit and began to separately test that the motors and wheels functioned as expected. Similarly, we followed the same process for our other components such as the optical sensors and pi camera. All hardware components were first assessed through the most basic python test we could implement before we continued to increase complexity.

After correctly wiring our sensors to our robot, we created a separate file to test their functionality. The file simply printed the value of each sensor: 1 if it sensed the electrical tape and 0 otherwise. While the test was very simple, it proved to be extremely useful when the robot had to be taken apart and put back together as we were able to quickly run the sensor test to check whether our wiring was correct. Additionally, when the robot moved unexpectedly, we could run the sensor test to help us narrow down if our issue was with the code or hardware. By focusing on unit testing, not only were we able to check the expected behavior of our sensors, but it also allowed us to debug other portions of our project.

Similarly, we created separate test files for displaying images on the LCD. We first began by displaying a single image before moving on to rotating between multiple images to create an animation like display. After we had rigorously tested this component, we then integrated it into the main functionality and performed integration testing by making sure the robot still moved as expected after adding the additional LCD functionality. We then found that our robot was unable to turn smoothly anymore. As we had tested all components separately and had been performing integration testing after every modification to the main file, we were able to easily pinpoint our bug to the LCD. We realized that while the LCD image code was running, the optic sensor would stop sensing the tape and the robot would veer off course. We were able to quickly correct our code and change how often we changed images, allowing our robot to turn smoother.

We also modularized our code by creating an Error class that executed all PID and error calculations needed to control the movement of the robot. The error class included an array for our sensors and Ki, Kd, and Kp values that were determined through experimentation. It also included three functions: calculateError(), getOptics(), and calculatePID(). These functions were all tested separately and, once we were confident they behaved appropriately, were integrated into our main file. Hiding the internal details of error and PID calculations not only made the main file easier to read, but also helped in organization of our code. By modularizing this component of our project, we were able to add a layer of abstraction and simplicity to our main functionality that allowed us to debug and review our code easier.

**E. Conclusions and Reflections**

Throughout the course of this project, we developed a greater appreciation for the interactions between hardware and software. For instance, in the process of developing our Android Application and establishing communication with the Raspberry Pi, we learned a tremendous amount about socket communication. Because we were so unfamiliar with sending and receiving bytestreams, we found ourselves resorting to various websites, initially intimidated by our overwhelming lack of knowledge. This type of open ended learning reinforced our ability to problem solve through searching and parsing various online solutions which we believe to be a valuable skill for the future. However, we approached each of our challenges in smaller modules. Rather than focussing on our end goal of creating a fully functional app with motor control and livestream input, we focussed first on establishing the connection, sending over JsonObjects, parsing received JsonObjects, getting motor control, sending over a single image, and finally sending over a livestream. We found this to be the most effective for debugging as we ensured smaller preliminary steps were functional before moving on.

This project also helped emphasize the importance of adaptability. We found ourselves continuously facing unexpected obstacles that forced us to reconsider our initial plans. For example, despite getting the livestream working within a smaller test file, integration of this element into the final working project introduced greater challenges. The delay involved with uploading continuous streams of images interfered with the performance of our main line-following functionality. Thus, we had to make the decision to implement a single image capturing function to ensure the integrity of our required functions was maintained.

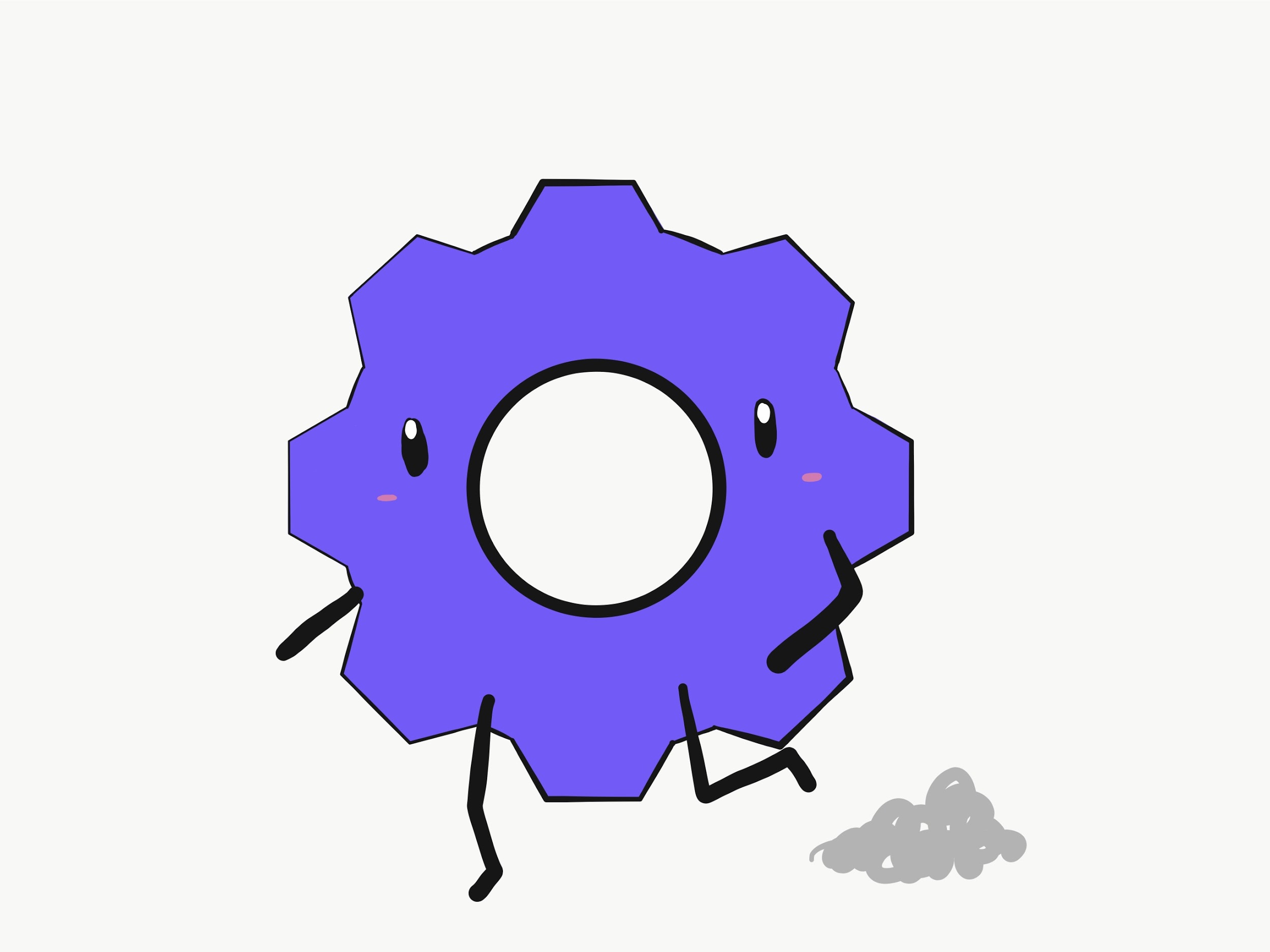
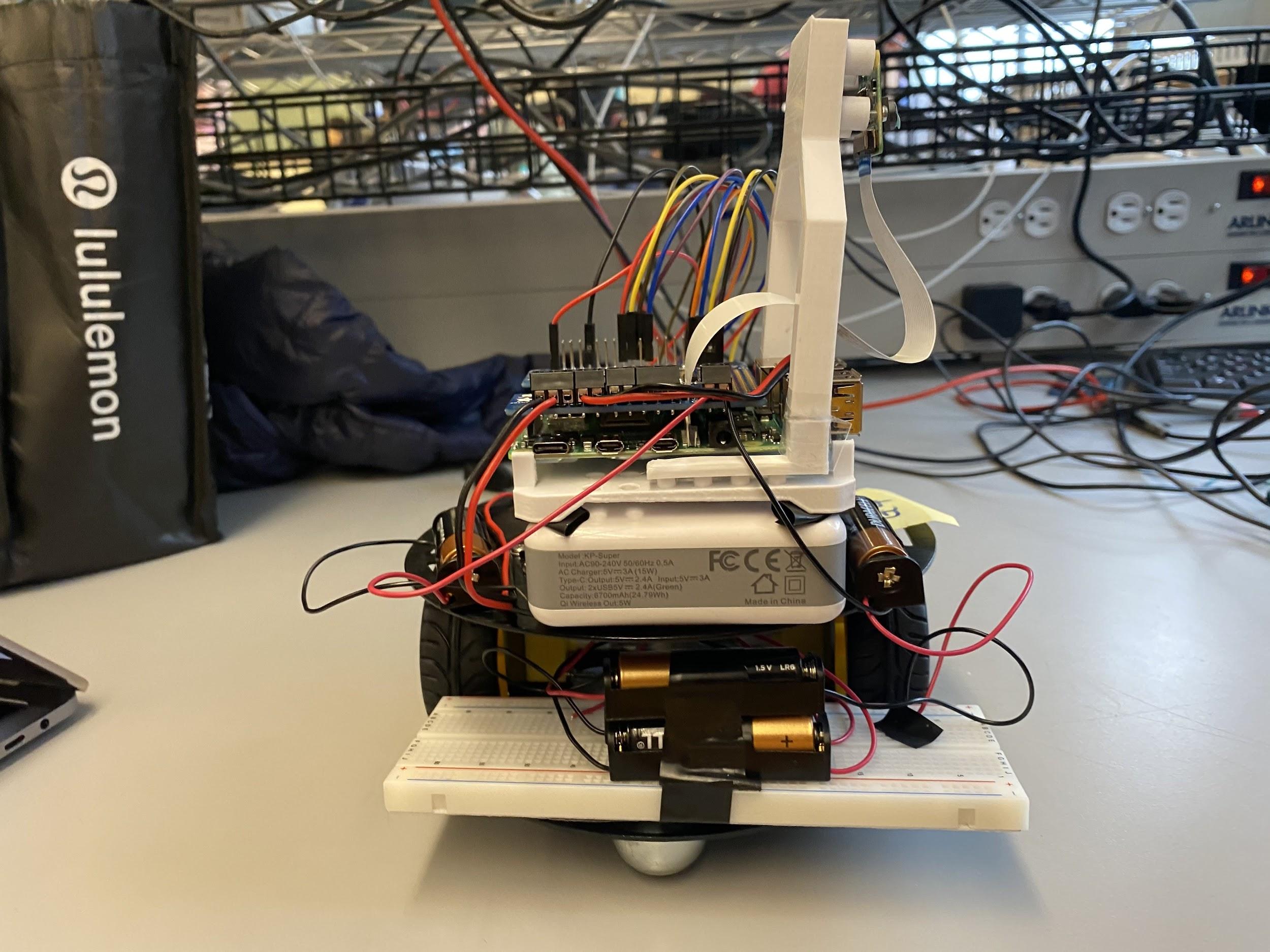
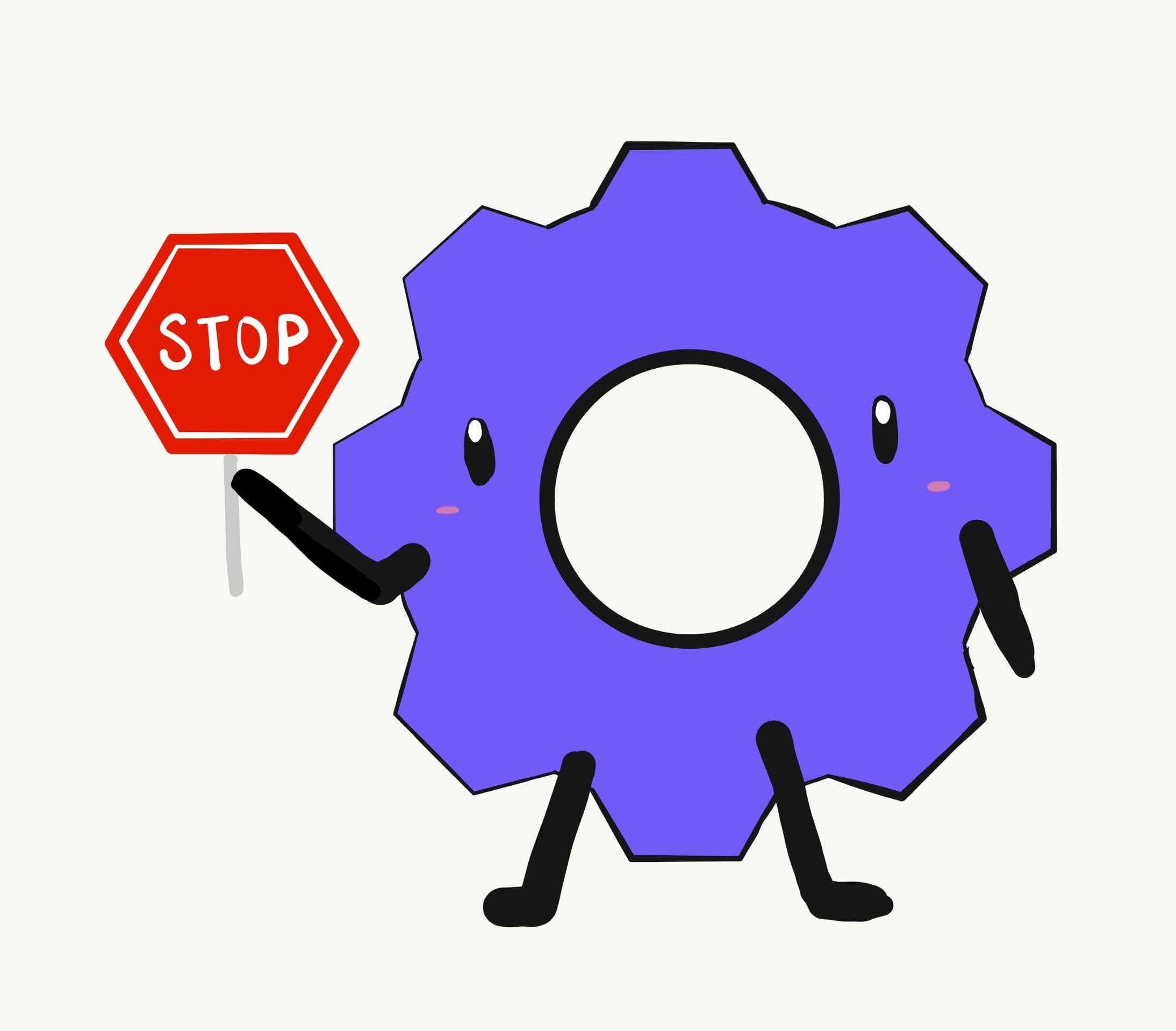
As we continue working with one another, the roles of leaders, mediators, and members continue to develop naturally. We are able to act as a fully functioning team where, although there are natural leaders, everyone is able to communicate and contribute to group discussions fully. We can easily divide different tasks amongst ourselves and split off during lab hours to work on our responsibilities. The management did not fall on just one person’s shoulders, but rather the tasks were delegated and taken up by almost everyone. As for time management, we had a bit of trouble with this in the beginning as we all have the same upcoming deadlines and exam schedule. But we were able to still find days between exams to meet after class to work on our project.

Overall, this was a challenging, but rewarding experience that taught us technical, communication, and teamwork skills.

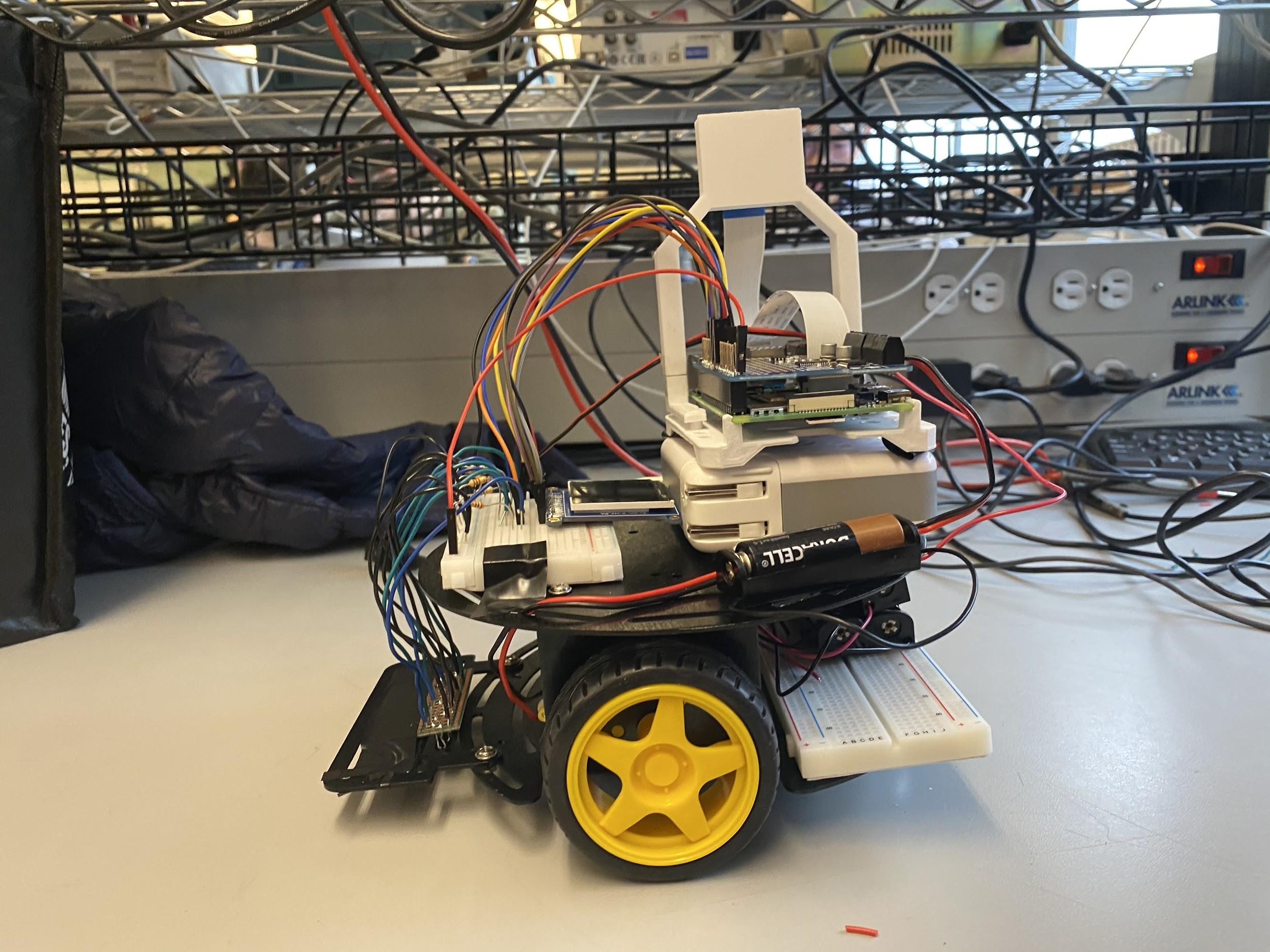
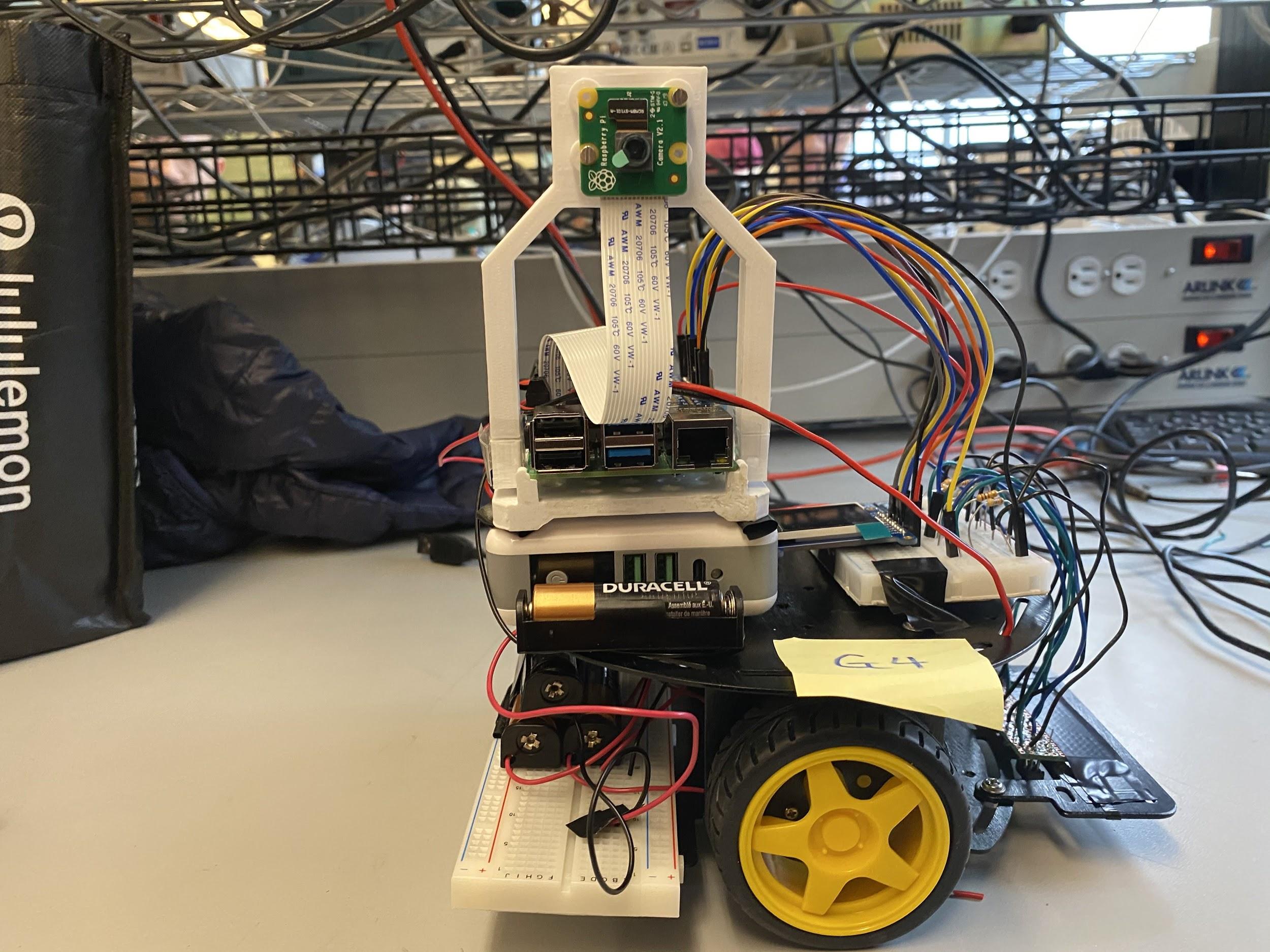
**F. References and bibliography**

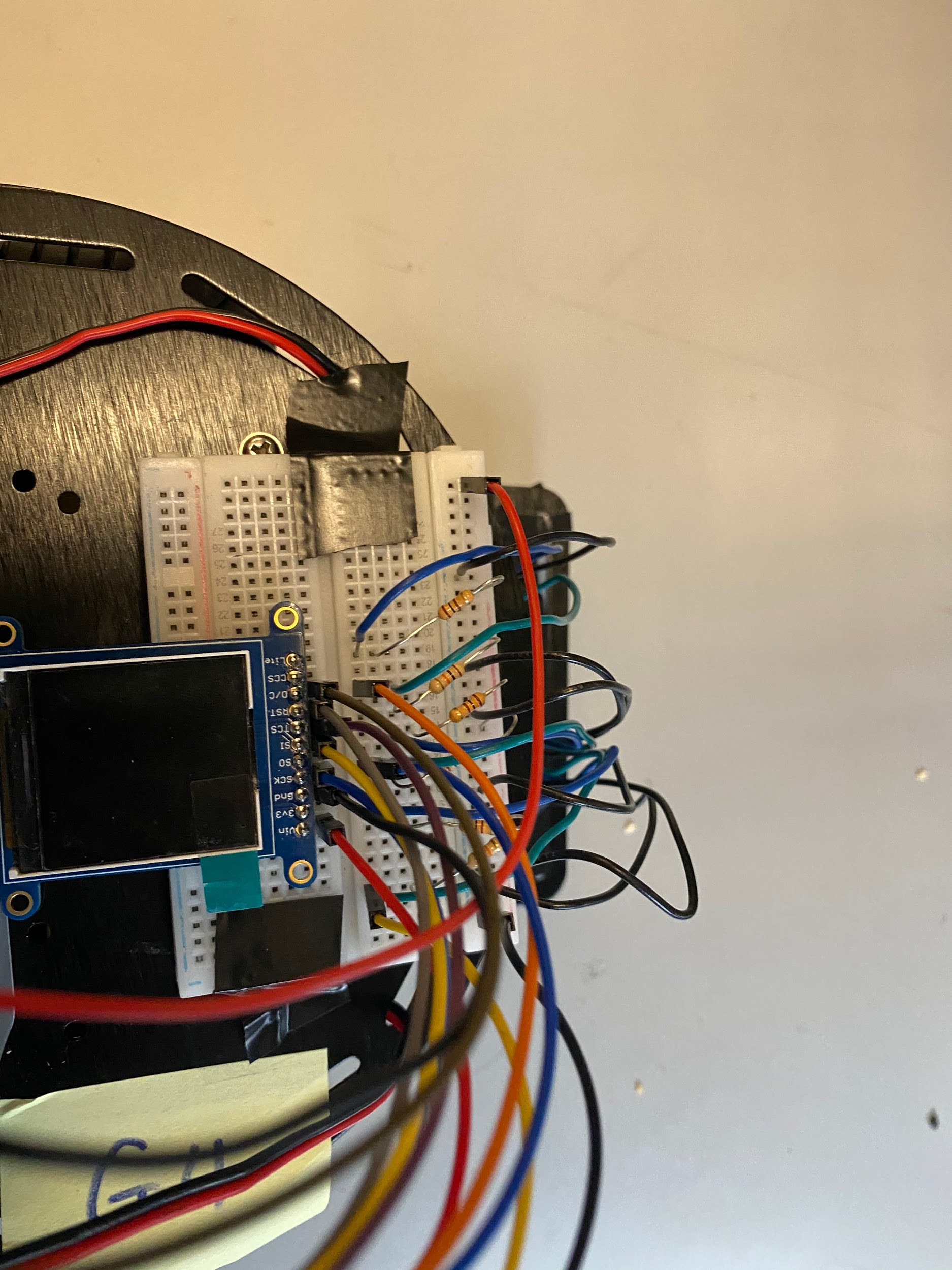
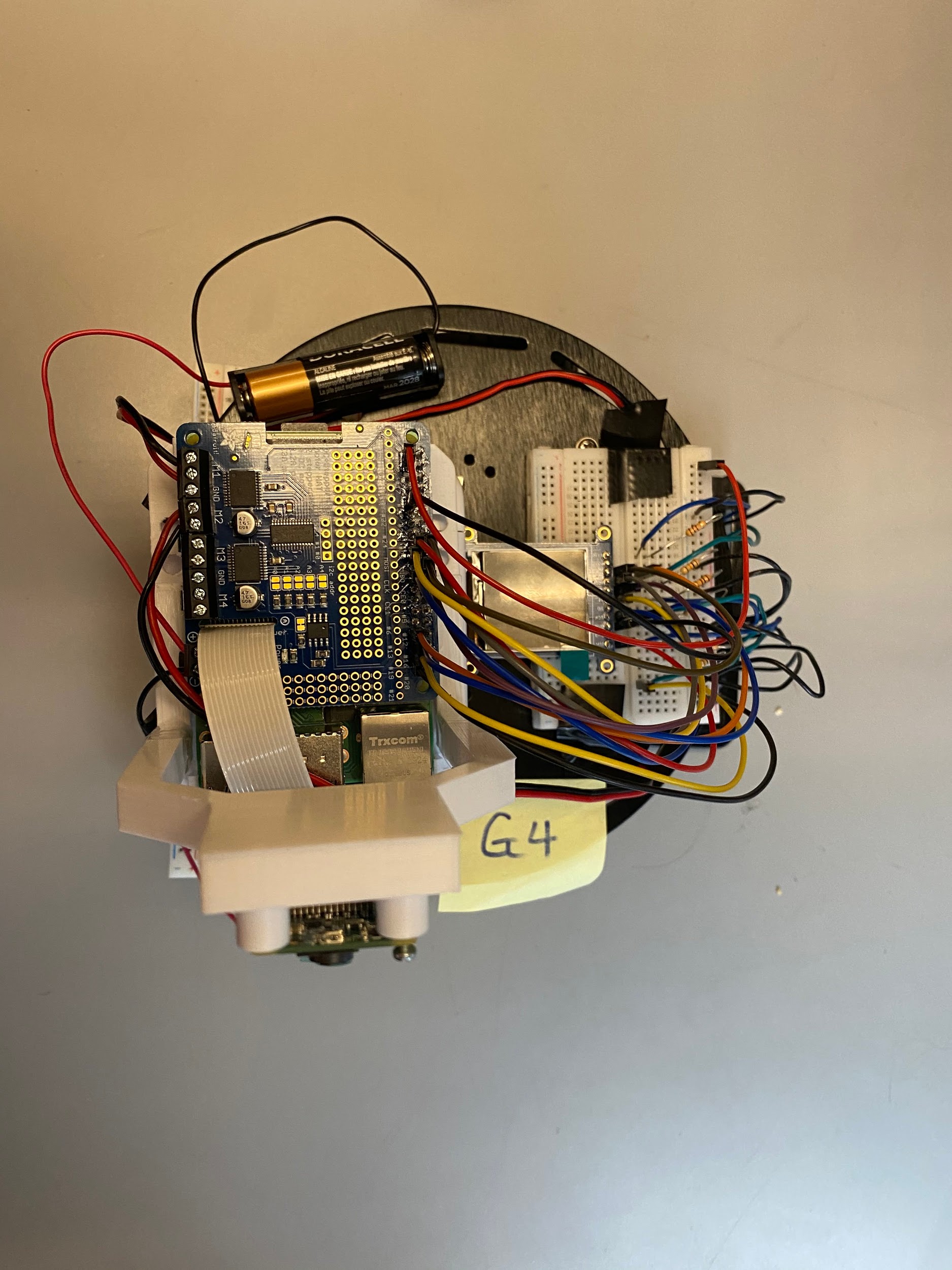
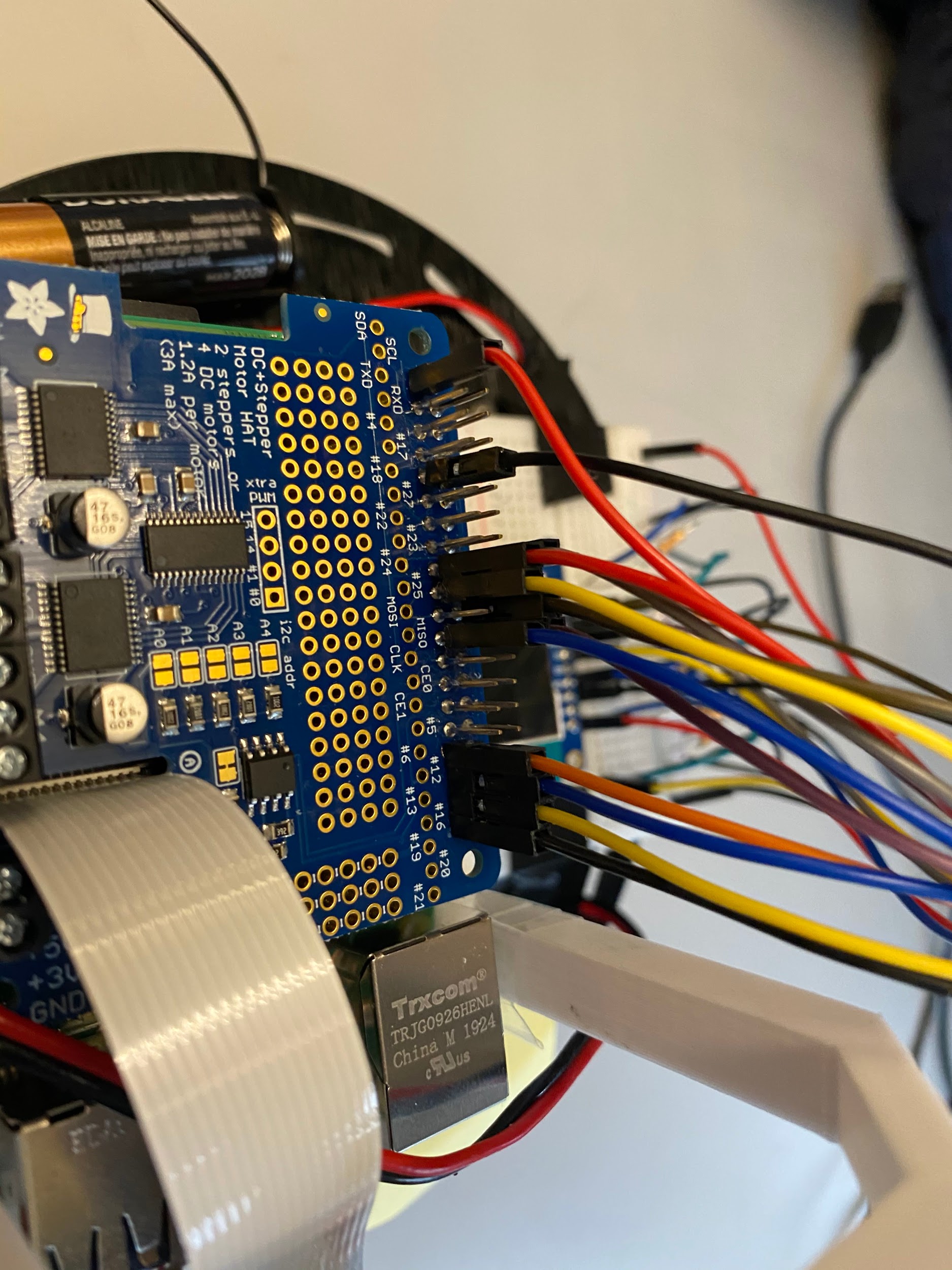
Submission:

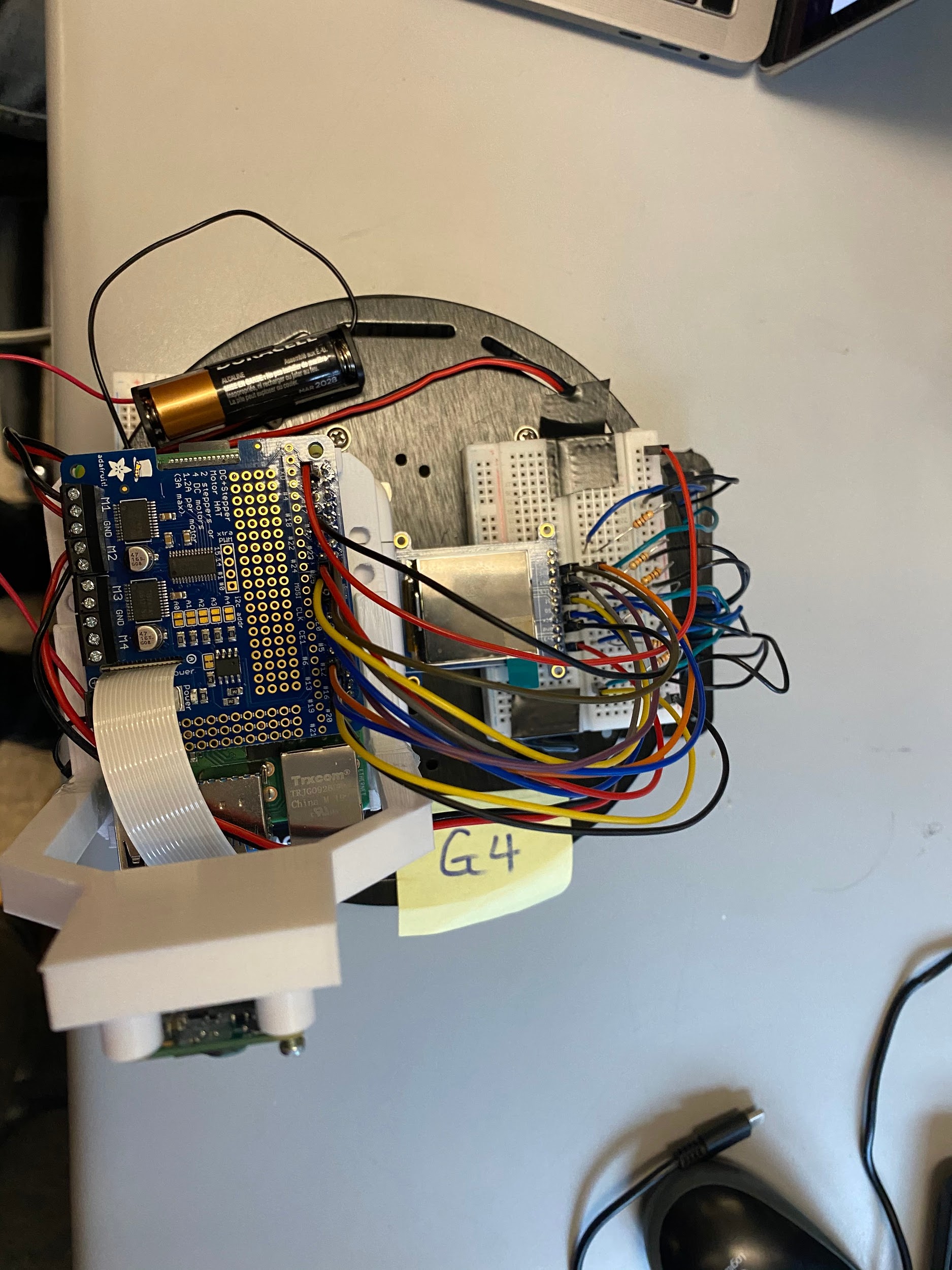
* 291 Project Report (this file)
* Fritzing file (RPi, LCD, both breadboards)
* Python file (main functionality)
* Java file (additional app functionality)

**Appendix A – Robot pictures**

LCD Image 1 - Displayed while robot moving LCD Image 2 - Displayed while robot stopped

****





**Appendix B - Code**

**Main functionality (Python):**

from RPi import GPIO

from adafruit\_motorkit import MotorKit

import time

import digitalio

import board

import io

from picamera import PiCamera

import picamera

from PIL import Image, ImageDraw, ImageFont

import adafruit\_rgb\_display.ili9341 as ili9341

import adafruit\_rgb\_display.st7789 as st7789 *# pylint: disable=unused-import*

import adafruit\_rgb\_display.hx8357 as hx8357 *# pylint: disable=unused-import*

import adafruit\_rgb\_display.st7735 as st7735 *# pylint: disable=unused-import*

import adafruit\_rgb\_display.ssd1351 as ssd1351 *# pylint: disable=unused-import*

import adafruit\_rgb\_display.ssd1331 as ssd1331 *# pylint: disable=unused-import*

*# Import libraries to allow us to communicate with android app*

import json

import socket

PORT = 5042 *# Port to listen on (non-privileged ports are > 1023)*

HOST = ''

GPIO.setmode(GPIO.BCM)

*# Assign sensor pins*

sensor1 = 26

sensor2 = 19

sensor3 = 13

*# Setup GPIO inputs*

GPIO.setup(sensor1, GPIO.IN)

GPIO.setup(sensor2, GPIO.IN)

GPIO.setup(sensor3, GPIO.IN)

*# Configuration for CS and DC pins (these are PiTFT defaults):*

cs\_pin = digitalio.DigitalInOut(board.CE0)

dc\_pin = digitalio.DigitalInOut(board.D25)

reset\_pin = digitalio.DigitalInOut(board.D24)

*# Config for display baudrate (default max is 24mhz):*

BAUDRATE = 24000000

*# Setup SPI bus using hardware SPI:*

spi = board.SPI()

*# 1.44" ST7735R*

disp = st7735.ST7735R(spi, rotation=270, height=128, x\_offset=2, y\_offset=3, cs=cs\_pin, dc=dc\_pin, rst=reset\_pin, baudrate=BAUDRATE)

camera = PiCamera()

**class** Error:

**def** \_\_init\_\_(self):

*# sensorVal[0] = left, sensorVal[1] = middle, sensorVal[2] = right*

self.sensorVal = [0, 0, 0]

self.error = 0

self.prevError = 0

self.integral = 0

self.Kp = 0.08

self.Kd = 0.13

self.Ki = 0

*# count for stopping motors*

self.count = 0

**def** calculateError(self):

*# 0th index for left sensor, 4th is the rightmost sensor from a topdown view*

*# Shift array elements to create one sum*

errorTotal = (self.sensorVal[0] \* 100)

errorTotal += (self.sensorVal[1] \* 10)

errorTotal += self.sensorVal[2]

if (errorTotal == 1): *# right sensor triggered*

self.count = 0

self.error = -1.7

elif (errorTotal == 11): *# middle and right sensors triggered*

self.count = 0

self.error = -1

elif (errorTotal == 10): *# middle sensor triggered*

self.count = 0

self.error = 0

elif (errorTotal == 110): *# middle and left sensors triggered*

self.count = 0

self.error = 1

elif (errorTotal == 100): *# left sensor triggered*

self.count = 0

self.error = 1.7

elif (errorTotal == 111): *# all sensors triggered, most likely crossover*

self.count = 0

self.error = 0

elif (errorTotal == 0):

self.count = self.count + 1

**def** getOptics(self):

*# sens1 = left sensor, sens2 = middle sensor, sens3 = right sensor*

sens1 = GPIO.input(sensor1)

sens2 = GPIO.input(sensor2)

sens3 = GPIO.input(sensor3)

self.sensorVal = [sens1, sens2, sens3]

**def** calculatePID(self):

*# Calculates PID value based on new sensor inputs and past values (prevError, integral)*

self.calculateError() *# adjust error values based on sensor readings*

self.integral += self.error

pidValue = self.Kp \* self.error + self.Kd \* (self.error - self.prevError) + self.Ki \* self.integral

self.prevError = self.error *# set prevError to new error for next iteration*

return pidValue

*#Instantiate the error class to calculate PID for us*

error = Error()

*# Function to parse json*

**def** parseJson(byteStream):

*# Decode UTF-8 bytes to unicode*

*# To make valid JSON, replace single quotes with double quotes*

jsonStream = byteStream.decode('utf8').replace("'", '"')

*# Load JSON to Python list*

jsonObject = json.loads(jsonStream)

print(jsonObject)

print(jsonObject.get("Type")) *# Type tells us if we are moving forward, left, right, or stopping*

return jsonObject

*# Function for robot to go straight*

**def** straight(kit):

*# both motors same speed*

kit.motor1.throttle = 0.35

kit.motor2.throttle = 0.40

*# Function for robot to turn left*

**def** turnLeft(kit):

*# left motor slower than right*

kit.motor1.throttle = 0.15

kit.motor2.throttle = 0.5

*# Function for robot to turn right*

**def** turnRight(kit):

*# right motor slower than left*

kit.motor1.throttle = 0.5

kit.motor2.throttle = 0.15

*# Function for robot to turn off*

**def** off(kit):

*# both motors turned off*

kit.motor1.throttle = 0.0

kit.motor2.throttle = 0.0

*#parameter is the imageName to write to board*

**def** writeImages(imageName):

*# pylint: enable=line-too-long*

*# Create blank image for drawing.*

*# Make sure to create image with mode 'RGB' for full color.*

if disp.rotation % 180 == 90:

height = disp.width *# we swap height/width to rotate it to landscape!*

width = disp.height

else:

width = disp.width *# we swap height/width to rotate it to landscape!*

height = disp.height

image = Image.new('RGB', (width, height))

*# Get drawing object to draw on image.*

draw = ImageDraw.Draw(image)

*# Draw a black filled box to clear the image.*

draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0)) *# create a rectangle that is blank to put on the LCD*

disp.image(image) *# put the image on the LCD*

image = Image.open(imageName) *# open the image given by the string imageName*

*# Scale the image to the smaller screen dimension*

image\_ratio = image.width / image.height

screen\_ratio = width / height

if screen\_ratio < image\_ratio:

scaled\_width = image.width \* height // image.height

scaled\_height = height

else:

scaled\_width = width

scaled\_height = image.height \* width // image.width

image = image.resize((scaled\_width, scaled\_height), Image.BICUBIC)

*# Crop and center the image*

x = scaled\_width // 2 - width // 2

y = scaled\_height // 2 - height // 2

image = image.crop((x, y, x + width, y + height))

*# Display image.*

disp.image(image)

*# Reference for sending stream: https://picamera.readthedocs.io/en/release-1.10/recipes1.html*

**def** captureStreamPIL():

stream = io.BytesIO()

camera.capture(stream, format='bmp') *# Take picture and store in stream*

stream.seek(0) *# Go to beginning of stream*

image = Image.open(stream) *# Store as PIL Image object*

*# Convert to byte array*

imgByteArr = io.BytesIO()

image.save(imgByteArr, format='bmp')

imgByteArrToReturn = imgByteArr.getvalue()

*#Return to byte array*

return imgByteArrToReturn

**def** controller(kit):

**global** error

print('entering controller')

*#Loop for the feedback loop*

try:

*#Calculate the PID value*

error.getOptics()

PID = error.calculatePID()

time.sleep(0.02)

*# Print stop image if counted to 25*

if (error.count == 25):

writeImages("stopGear.jpg")

kit.motor1.throttle = 0.0

kit.motor2.throttle = 0.0

elif (PID == 0.0):

kit.motor1.throttle = 0.40

kit.motor2.throttle = 0.40

*#sum the pid value with the base throttle of 0.75 to turn left or right based on imbalances in the throttle values*

else :

kit.motor1.throttle = 0.25 + PID *#Assuming this is the left motor*

kit.motor2.throttle = 0.25 - PID *#Assuming this is the right motor*

except KeyboardInterrupt:

kit.motor1.throttle = 0.0

kit.motor2.throttle = 0.0

*#Instantiate the motorkit instance*

kit = MotorKit()

camera.start\_preview()

*# Camera warm-up time*

time.sleep(2)

*# Wait for client connection*

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

off(kit)

s.bind((HOST, PORT))

s.listen()

(conn, addr) = s.accept()

print("Connected")

with conn:

img = captureStreamPIL() *#capture the image*

conn.send(img) *# send the image through socket*

time.sleep(1)

camera.stop\_preview() *# stop the preview*

input = conn.recv(1024) *# receive input from client*

noValue = input *# store initial input value - may be garbage*

conn.setblocking(0) *# set connection to be blocking*

mode = "Stop" *# default mode to stop*

writeImages("firstGear.jpg")

s.close()

try:

*# Once connected, continuously check for input from app*

while True:

try:

*# Reading next input*

input = conn.recv(1024)

except:

print("Error!")

print(input)

if input != noValue:

print("Trying to print")

jsonObj = parseJson(input)

mode = jsonObj.get('Mode') *# Get 'Mode' from parsed Json*

if (mode == 'Autonomous'): *# If Autonomous mode, run main functionality code*

print('entering auto')

controller(kit)

elif (mode == 'Remote'): *# If Remote mode, check if forward, left, right, or stop button clicked*

Type = jsonObj.get("Type")

time.sleep(0.02)

if (Type == 'Forward'): *# Move forward*

straight(kit)

elif (Type == 'Left'): *# Move left*

turnLeft(kit)

elif (Type == 'Right'): *# Move right*

turnRight(kit)

else:

off(kit) *# Stop moving*

else:

off(kit)

camera.stop\_preview()

s.close()

except KeyboardInterrupt:

kit.motor1.throttle = 0.0

kit.motor2.throttle = 0.0

**Additional Functionality (Java):**

**package com.example.robotcontrol;**

**import android.graphics.Bitmap;**

**import android.graphics.BitmapFactory;**

**import android.os.Bundle;**

**import android.os.Handler;**

**import android.os.StrictMode;**

**import android.view.Menu;**

**import android.view.MenuItem;**

**import android.view.View;**

**import android.widget.Button;**

**import android.widget.CompoundButton;**

**import android.widget.ImageView;**

**import android.widget.Switch;**

**import android.widget.TextView;**

**import androidx.appcompat.app.AppCompatActivity;**

**import org.json.JSONObject;**

**import java.io.IOException;**

**import java.io.InputStream;**

**import java.io.OutputStreamWriter;**

**import java.io.PrintWriter;**

**import java.net.InetAddress;**

**import java.net.Socket;**

**import java.net.UnknownHostException;**

**import java.util.Timer;**

**import java.util.TimerTask;**

**public class MainActivity extends AppCompatActivity {**

**private Socket socket; *//Create the socket instance***

**private InputStream in; *//Create the inputstream***

**private PrintWriter out; *//output stream instance***

**private InetAddress address; *//Put address of the raspberry pi here***

**private int port; *//Put the port number of the raspberry pi here***

**private ImageView robotCamera; *//create the imageView instance***

**private Switch modeToggle; *//Create the switch***

**private TextView modeStatus; *//Create text to display the mode status***

**private Handler mHandler = new Handler();**

**@Override**

**protected void onCreate(Bundle savedInstanceState) {**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.activity\_main);**

***//Instantiate the port***

**port = 5035;**

***//Change the android policy to allow us to use network operations on the main thread***

**StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();**

**StrictMode.setThreadPolicy(policy);**

***//Try creating an ip address***

**try {**

***// address = InetAddress.getByName("137.82.226.227");***

**address = InetAddress.getByName("137.82.226.222");**

**} catch (UnknownHostException e) {**

**e.printStackTrace();**

**}**

***//Instantiate the mode toggle***

**modeToggle = (Switch) findViewById(R.id.modeToggle);**

***//Instantiate the mode status***

**modeStatus = (TextView) findViewById(R.id.ModeStatus);**

***//Get the forwardButton***

**final Button forwardButton = (Button) findViewById(R.id.forward);**

***//Get the leftButton***

**final Button leftButton = (Button) findViewById(R.id.leftwards);**

***//Get the rightbutton***

**final Button rightButton = (Button) findViewById(R.id.rightwards);**

***//Get the stopButton***

**final Button stopButton = (Button) findViewById(R.id.stop);**

***//Get the startButton***

**final Button startButton = (Button) findViewById(R.id.Start);**

***//Instantiate the imageView and set settings***

**robotCamera = findViewById(R.id.imageView);**

**robotCamera.setLayerType(View.LAYER\_TYPE\_SOFTWARE, null);**

**robotCamera.setVisibility(View.VISIBLE);**

***//Create onclick listener for the start button***

**startButton.setOnClickListener(new View.OnClickListener() {**

**@Override**

**public void onClick(View v) {**

***//Send json to pi that we want to start in stop mode***

**sendRequest("Null", "Stop");**

***//Retrieve the photo to display***

**mainFunctionality();**

**}**

**});**

***//Add an event listener for the forwardsButton to call moveForward onclick***

**forwardButton.setOnClickListener(new View.OnClickListener() {**

**public void onClick(View v) {**

**moveForward();**

**}**

**});**

***//Add an event listener for stopButton to call stop***

**stopButton.setOnClickListener(new View.OnClickListener() {**

**@Override**

**public void onClick(View v) {**

**stop();**

**}**

**});**

***//Add an event listener for the leftButton to call turnLeft onclick***

**leftButton.setOnClickListener(new View.OnClickListener() {**

**public void onClick(View v) {**

**turnLeft();**

**}**

**});**

***//Add an event listener for the rightButton to call turnRight onclick***

**rightButton.setOnClickListener(new View.OnClickListener() {**

**public void onClick(View v) {**

**turnRight();**

**}**

**});**

***//Add an event listener for the toggle***

**modeToggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {**

**@Override**

**public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {**

***//If the switch is on set the text to remote control***

**if(isChecked) {**

**modeStatus.setText("Mode:Remote Control");**

**}**

***//If the switch is off set the text to autonomous***

**else {**

**modeStatus.setText("Mode:Autonomous");**

**}**

***//Make sure we update the pi with the mode***

**ChangeMode(isChecked);**

**}**

**});**

***//Call serve to connect to the pi***

**try {**

**serve();**

**} catch (Exception e) {**

***//If exception, print stack trace***

**e.printStackTrace();**

**}**

**}**

***//Start the sever***

**public void serve() throws IOException{**

***//Set the onscreen message to say that there is a good connection***

**final TextView connectedStatus = (TextView) findViewById(R.id.textView);**

***//While loop for handling until we close sockets***

**try {**

***//Connect it to the specified port and ipaddress***

**socket = new Socket(address, port);**

**System.out.println("CONNECTED");**

***//Setup the input and output streams***

**in = socket.getInputStream();**

**out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));**

***//If no exception, display a message so we know that it is connected***

**connectedStatus.setText("Connected");**

**} catch (Exception e) {**

**System.out.println("Not connected");**

***//If execption, print stack trace***

**e.printStackTrace();**

***//Once the try catch finishes***

**}**

**}**

***//Start to listen and serve requests***

**public void mainFunctionality() {**

***//Create a bitmapfactory options object***

**BitmapFactory.Options options = new BitmapFactory.Options();**

***//Set the scaling to value of 8***

**options.inSampleSize = 8;**

***//create a new bitmap***

**Bitmap image = BitmapFactory.decodeStream(in, null, options);**

***//set the image as the newly converted bitmap if the image is not null***

**if (image != null) {**

**robotCamera.setImageBitmap(image);**

**}**

**}**

***//Send a request***

**public void sendRequest(String movement, String mode) {**

***//Create a new JSONObject***

**JSONObject request = new JSONObject();**

***//Try adding fields to the JSON***

**try {**

**request.put("Type", movement);**

**request.put("Mode", mode);**

***//Send the request out over the socket***

**out.print(request);**

***//Flush the socket***

**out.flush();**

**}**

**catch (Exception e) {**

***//If exception, print stack trace***

**e.printStackTrace();**

**}**

**}**

***//Tell the pi to change modes***

**public void ChangeMode(boolean mode) {**

***//If toggle is set to remote control send mode for remote***

**if(mode) {**

**sendRequest("NULL", "Remote");**

**}**

***//If toggle is set to autonomous send mode for autonomous***

**else {**

**sendRequest("NULL", "Autonomous");**

**}**

**}**

***//send a request to turn left***

**public void turnLeft() {**

***//if the mode is remote control then send request***

**if(modeStatus.getText() == "Mode:Remote Control") {**

**sendRequest("Left", "Remote");**

**}**

**}**

***//send a request to turn right***

**public void turnRight() {**

***//if the mode is remote control then send request***

**if(modeStatus.getText() == "Mode:Remote Control") {**

**sendRequest("Right", "Remote");**

**}**

**}**

***//send a request to move forwards***

**public void moveForward() {**

***//if the mode is remote control then send request***

**if(modeStatus.getText() == "Mode:Remote Control") {**

**sendRequest("Forward", "Remote");**

**}**

**}**

***//send a request to stop***

**public void stop() {**

***//if the mode is remote control then send request***

**if(modeStatus.getText() == "Mode:Remote Control") {**

**sendRequest("Stop", "Remote");**

**}**

**}**

**@Override**

**public boolean onCreateOptionsMenu(Menu menu) {**

***// Inflate the menu; this adds items to the action bar if it is present.***

**getMenuInflater().inflate(R.menu.menu\_main, menu);**

**return true;**

**}**

**@Override**

**public boolean onOptionsItemSelected(MenuItem item) {**

***// Handle action bar item clicks here. The action bar will***

***// automatically handle clicks on the Home/Up button, so long***

***// as you specify a parent activity in AndroidManifest.xml.***

**int id = item.getItemId();**

***//noinspection SimplifiableIfStatement***

**if (id == R.id.action\_settings) {**

**return true;**

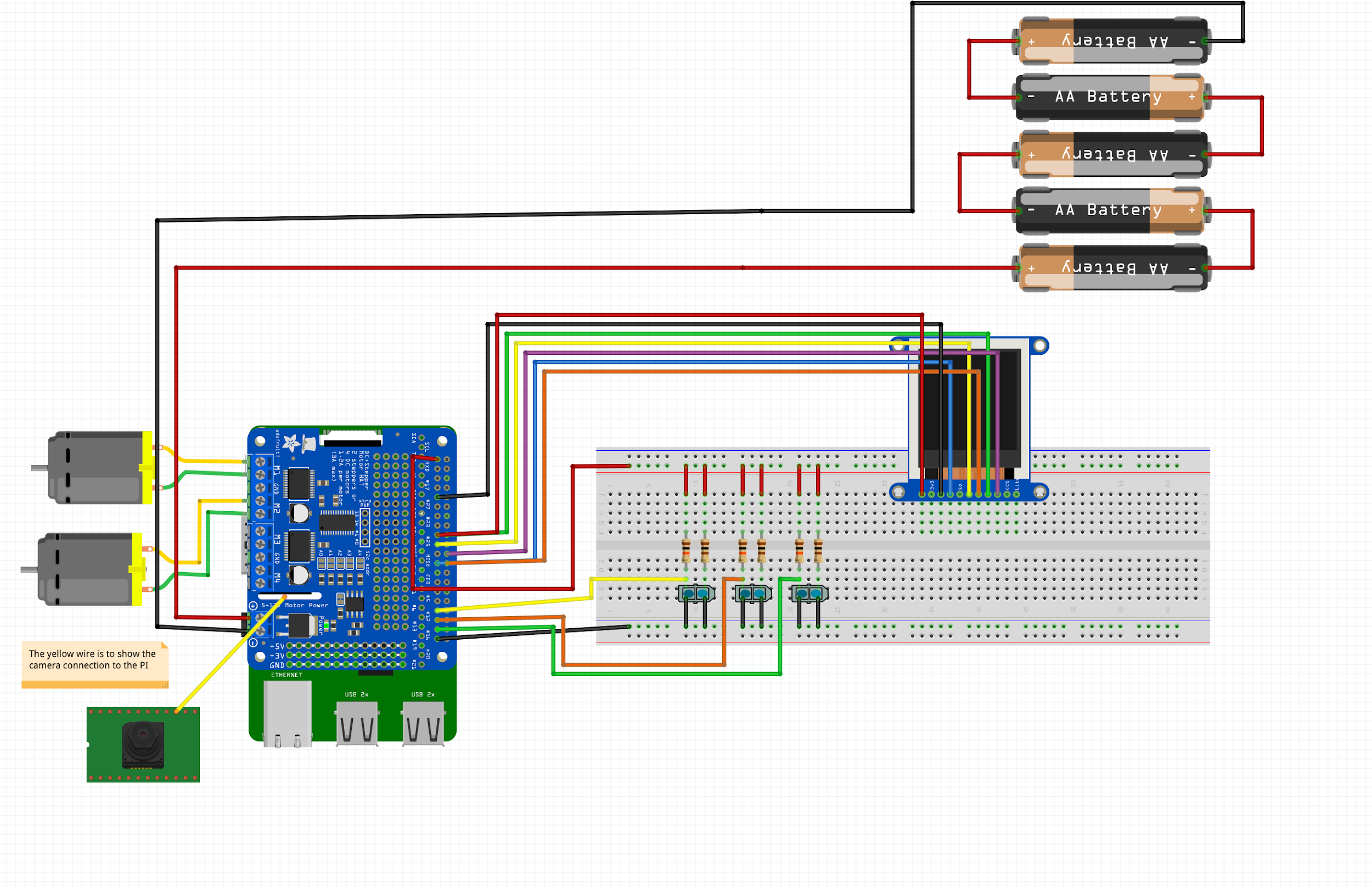
**}**

**return super.onOptionsItemSelected(item);**

**}**

**}**

**Appendix C - Fritzing**



**Appendix D - GitHub**

**Appendix E – Complete Component list**

* 1 Raspberry Pi 4 Model B
* 1 Adafruit 1.44 inch TFT LCD
* 1 Robot kit
  + Fuselage upper and lower plate
  + 2 Geared DC motors
  + Multiple screws, washers, spring washers, nuts
  + 2 wheels
  + 1 Caster
  + 1 Sensor mounting grill
  + 1 Battery holder (burned, required multiple single battery holders)
  + Power switch and jack
  + 1 IR sensor mounting bracket
* 5 AA batteries
* 1 Motor hat
* 3 TCRT5000 reflective optical sensors
* 1 Raspberry Pi Camera Module V2
* 1 Android phone (Used for app for additional functionality)
* 2 Small breadboards
* 1 Large breadboard
* Multiple wires
* Electrical tape

**Appendix F – Answer the following questions:**

Q1 – Teamwork: Explain in detail the methods your group has used to communicate effectively among team members.

After the last mini project, we found that a Facebook chat proved to be useful in connecting with one another while not at lab. We continued using the same chat for this project as well. It allowed us to easily ask questions, send links to google docs, and keep progress regularly updated.

On top of this, we also had a short team meeting at the beginning of the first lab. We broke our project into smaller modules and assigned one or two people to be the leaders of each section. This allowed us to avoid an error we ran into in the last project where different portions of the project would be almost done, but nothing was ever fully complete. By assigning leaders to each section, there was always someone continuously checking and ensuring each portion was up to date. This being said, each group member still worked on all parts of the project to ensure maximum exposure and experience. The leaders per section was just a strategy to keep progress flowing.

Additionally, our team also implemented a lot more partner programming in this project. As our main components were the Android app, sensor and PID controls, and actual movement of the robot, there were always around 2 people working on a portion of the project at a time. We were able to continuously rotate between one person coding and another reviewing. As the person coding was able to talk through their thought process with their partner, we were able to catch a portion of bugs just through verbal communication. This also eased us into an environment that welcomed constructive criticism and overall allowed for more effective communication and collaboration.

Q2 – Design Process for the additional functionalities: Describe clearly the process you used for the following design aspects of your own additional functionalities. Please spend time to carefully answer each of them.

1. **Use of process**: *Describe your approach to adapt and apply a general design process for any additional feature. What was your approach?*

When conducting this project, we decided on what we wanted to do and divided the work equally. For more difficult aspects of the project, two or more people would meet to work together to develop solutions and implement solutions. This included building the robot, programming the LCD screen, optical sensors, Android app, and the video stream from the camera. After doing our individual parts, we would review one another’s code. This helped us quickly fix bugs, as it gave a new perspective on the problem at hand. This allowed us to quickly integrate everything into the final product.

1. **Constraint identification**: *Explain the constraints that you must consider in the design of the additional functionalities.*
2. Constraint 1: We must consider the abilities of the components we used. For example, the ability of the sensors to detect whether the ground is black (is covered in tape) and what distance from the ground they must be placed. We must also consider the power and torque that motors exert. If not enough power is being fed to the motor, it might not exert enough torque to move the robot.
3. Constraint 2: When introducing additional features, we found that it would sometimes slow down the main functionality of our program. We have to ensure that the features can be made to work in sync with the other components of the project. A problem we faced was that extensive use of the LCD caused the robot to not change direction as fast as necessary. This reflects the limits of the Raspberry Pi’s processing speed. Another problem was working with the restrictions of sockets and learning how to properly pass data through a socket between the Pi and the Android app.
4. **Solution generation**: *Explain at least two possible alternative additional features that your group rejected due to technical reasons and explain why.*

**Solution 1**

We considered including more complex computer vision functionalities. Such functionalities would have required the installation of libraries that would have been impractical in comparison to implementing other additional features due to the memory space and processing speed of the Pi. These computer vision components would have involved having the robot react to recognition of certain objects and images and would have likely involved the use of Tensorflow and OpenCV to process the image and work with the image data. The installation of these packages and their functioning might not have led to optimal functionality.

**Solution 2**

We considered controlling the robot with hand gestures in front of a gyroscopic sensor. After thinking about this idea we realized that we already worked with the gyroscopic sensor extensively and we wanted to learn new things like app development and sockets.

1. **Solution Assessment**: *Explain how you tested and assessed the viability and then correctness of your group’s additional features.*

To be successful in this project, testing, assessing viability and overall correctness was vital. Before integrating any part into the project, we tested the code and its implication extensively. This is shown in our GitHub repository. Each aspect of the robot’s functionality and its additional features has its own file, and each one was tested rigorously. This reduced errors in the overall project and code and allowed our results to be more correct. This ideology was applied in all aspects of creating the robot and the additional features. Repetitive testing was required to get the “stream” to work.

**Appendix G - Other**

Below we have included photos of the interface for our app (additional functionality):