**CPEN 291   
Project 1 Report**

**A. Group info**

Lab section: *L2B* Group #: B\_G11 Group’s Lab Bench #s: 11 & 12

Student names:

|  |  |
| --- | --- |
| **Sanjeev Krishnan** | **Parsa Riahi** |
| **Arnold Ying** | **Amir Ali Barkam** |
| **Stella Wang** | **Rain Zhang** |
| **Manek Gujral** |  |

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

* The hardware including the circuit for the reflective optical sensors, and why you chose this configuration (i.e. the number of sensors you used and the way you arranged them relative to each other)

For the reflective optical sensors, we used a 100 Ohm resistor connected to the Digital in pin and a 4100 Ohm resistor connected to the 5V pin. This ratio was chosen to maintain the recommended circuit but to increase the power to the circuit at a safe level that will operate well under our otherwise low power drawing conditions. In total, we used 4 reflective optical sensors for our robot. They are used as sensors to support our PID controller that detects whether the robot is aligned with the track.

The four sensors are configured in two pairs that are situated tape-width apart - approximately 1 cm (See Appendix G-1 for diagram). This way, if the inner sensors detect black while the outer detect white, the robot is following the line. If both sensors in the right pair detect black while both sensors in the left pair detect white, then the robot is straying left, and vice versa for straying right.

Used in conjunction with a PID controller, we are able to generate a stream of error values based on the robot’s alignment with the track to output adjustment values to the motor, so the robot will turn smoothly and without significant oscillation, having used all three terms of the PID.

* The algorithm for the line tracking functionality

The algorithm for the line tracking functionality is a Proportional Derivative Integral (PID) controller that takes into account the current error value, its derivative and its integral over time according to binary input from the four sensors. Each sensor outputs a bit, with 0 signifying white and 1 signifying black. Therefore, we get a 4-bit input from four sensors with the most significant bit from the leftmost sensor and least significant bit from the rightmost sensor.

Using the input, we calculate an error value that is either positive, negative, or zero which signifies an adjustment to the right, left, or none, respectively. For example, if the input is 4b’0110, then our PID outputs a value of 0, namely no adjustment is required and the robot continues to move forward.

Sensors are sampled at a rate of 3000 Hz, namely approximately every 1/3000 seconds. The sample rate was decided upon after trial and error and fine tuning to allow for sufficient time to see a sharp turn and react to it.

* The headless Pi use, implementation, and challenges

The headless Pi is used as the main controller for the PID algorithm and the motor speeds. It is attached to a portable battery pack and the Motor Hat and is situated on the 2WD Mobile Platform robot to allow autonomous function and can be controlled via ssh if needed after powering on.

It is at times difficult to control the RPi headless, as it can only be communicated via terminal. In the beginning, it was difficult to get used to coding through the terminal, but this issue was solved by the team’s desire to always learn more and now each member can use the terminal to start the Pi. We also purchased a touchscreen LCD for our demo so this will allow easier use of the Pi going forward.

* Battery-operated robot implementation and challenges

The main body of our robot is the 2WD Mobile Platform, which consists of 5 AAA batteries and 2 DC motors. The two DC motors connect to pins on the Motor Hat to allow control of voltage supply via the MotorKit (software). The battery source is attached to a power switch on the 2WD Mobile Platform before connecting to the Motor Hat. The motors can independently go forwards and backwards due to the individual H-Bridges existing in the motor hat, removing the need for our team to use relays or another large hardware component that would hinder the robot’s overall speed or else draw more power to move due to the added weight.

The greatest challenge that we ran into with the robot implementation was during fine tuning of the motor and PID control. Many times, we will fine tune our code to working condition and come back the next day to have the robot not following the line properly again. This is due to a number of factors: battery pack drains slowly, ambient lighting and shadows in testing area, different track shape, etc. The general method we used to determine accurate Kd, Kp, and Ki values (in that order) was similar to the Ziegler-Nichols method described here:

1. Set all gains to 0.
2. Increase Kd until the system oscillates.
3. Reduce Kd by a factor of 2-4.
4. Set Kp to about 1% of Kd.
5. Increase Kp until oscillations start.
6. Decrease Kp by a factor of 2-4.
7. Set Ki to about 1% of Kp.
8. Increase Ki until oscillations start.
9. Decrease Ki by a factor of 2-4 and the PID values are approximately set.
10. Do some small fine tuning.

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

* What the additional functionalities are
  + Mobile app control of the robot
  + LCD Display
  + picture taking at the finish line
  + twitter bot
  + object detection
  + full path travelled graphing on LCD
* Include the list of the additional components you used
* ultrasonic sonar sensor
* 3.5” Touchscreen TFT LCD
* Pi camera module
* android phone for robot control
* How camera is used as a part of an additional feature
* The camera is used to take snapshots of the robot’s environment to be posted to Twitter as an update tweet. We decided against complicating its implementation in terms of adding filters and adjusting its exposure levels and modes, so as to speed up the photo taking process, as the camera already needs a minimum time of 5 seconds between shots to adjust for the lighting of the shot.

* The hardware implementation
  + Infrared Sensor protoboard
    - Specific angle chosen to minimize the effect of shadows cast into the infrared sensors, averaging around 45 degrees for best performance, also allows for extra time before the turn to handle the data read from the sensors since the distance between the wheels and the part of the tape detected is maximized.
  + Vehicle weight balancing
    - To maximize the robot’s control in turns, the hardware we installed was centered around the robot’s center of mass to make sure that both wheels experience the same friction and torque in their respective sides’ turns so that we do not have to adjust for weight imbalance experimentally in the software. This was something of an issue before we switched to the protoboard and fixed all our hardware in place.
* The software implementation
  + LCD Path Graphing
    - Since the robot always determines its own direction and speed, we can plot the path it travels, and essentially draw a picture of the tape line that it sees.
    - We assume an initial orientation of moving up in the y direction, and use the last known point to calculate the coordinates of the next point through some trigonometric identities and formulae
    - This image is shown on the LCD dynamically after the robot traverses its path, so as to not interfere with the PID readings at our set rate of 3000 readings/second.
  + Mobile App to control Robot
    - The App was developed in Android Studio. It connects to the robot via bluetooth. It controls the robot through a joystick. The App sends the data in a string format to the pi. The main.py code has a decoder for the input and calls a function depending on what the user calls.
    - The App code has two parts. One part is the bluetooth code that connects to a bluetooth device. The other part checks for the coordinates the user moves the joystick and then casts the value to a string so it can be sent to the robot. There were several challenges we faced while implementing the code on android studio. We realised we missed a few parameters but the code still compiled and showed no errors, this gave us a hard time to find the bug. The bluetooth connection with the pi was dicey and disconnected very often.
    - The Python code takes the input as x,y coordinates of the joystick. The x,y coordinates are then converted into vectors and angle is computed. The angle is then used to find the direction the robot has to move in. There is also a factor, calculated depending on the distance of the joystick from the center, which determines the speed of the robot. The main challenge we faced in creating this code was conversion from the vectors to angle because we had a duplicate variable constant which was causing a bug.
    - Another important aspect to discuss about the app was the choice to use a joystick rather than buttons, because we felt the moving robot should not be constrained by 4 or even 8 directions. We first implemented an app with buttons to move forward, backward, right, left, stop and demo but decided to create a joystick to make the app more interesting and controlling the robot fun.
  + Object detection
    - The robot looks for obstacles before and after it starts running and will print the distance of the closest object as a tweet
  + Twitter bot
    - Will tweet the photo the raspberry pi captures along with relevant information about the distance of obstacles in front of it and the speed at which it travels to the twitter API, which posts it when the robot detects a distance between 10 and 100 cm in front of it (See Appendix G-3).

**D. Test and evaluations**

Since each of us started off by writing separate code in separate files for each different component, we did initial testing for each component separately. For example, one person wrote ‘motor.py’ for software controlling the DC motors, while another person wrote ‘camera.py’ for testing and controlling the camera. After ensuring hardware and software for each component was working, we started integrating two components at a time and ensured the integration was functioning before integrating additional components. For example, the first two components that we integrated together was the motor and PID controller.

To test the motors, we tried both stepper motor and motor.throttle from the MotorKit library. We started by connecting one motor to the MotorHat on the Raspberry Pi and ensured that the motor moved according to the code. We chose to use motor.throttle over stepper motor because the output movement on the motor was more consistent. Then, we attached both motors to the Motor Hat and ensured both motors moved simultaneously and according to code.

After ensuring motor control via software, we tested the integration of our PID controller. This part required lots of iteration and fine tuning from trial and error, as various factors affected the output on our robot. For example,

Finally, after ensuring the basic functionality of robot following a line, we moved on to implementing and integrating additional functionalities, starting with the camera and LCD display. Most of our issues here were in the integration of all processes together, and since the PID controls needed around a 3000 Hz frequency of input signal to be accurate, we didn’t want to sacrifice our robot’s performance for the additional features, thus most additional features except the mobile app control take place at the beginning and end of the robot’s movement, since the time to read the sonar or update the graph severely hinders the performance of the line tracking in our single thread implementation.

The last thing we had to test was the mobile app, We had 2 basic functionalities to test for the app to work. One was the bluetooth connection to the robot. We decided to have a hard coded connection to the robot so we do not have errors while connecting. The second functionality was evaluating the angle of the joystick to decide the direction the robot will move in. We had to find the center coordinates of the joystick and then compute the vectors from origin to compute the angle using the arctan formula. We also added a factor to change the speed of the robot depending on the magnitude of the vector, because we thought it was a good feature while testing. To ensure the joystick moves within its constraints, we hard coded the center and its boundaries. This did affect the functionality of the joystick but ensured it worked within the constraints. The negative effects of this was when the joystick was moved out of the constraint region by force, it tried to reverse the direction. Another issue was the constraint region was an approximate of the desired constraint region, so this caused a bug when the user moved out of the constraint region, even though he was well within the desired one. We decided it was a trade-off we needed to take for better functionality of the joystick, because if we did not hardcode the constraints the joystick did not respond accurately.

**E. Conclusions and Reflections**

This was a very open-ended project, which allowed more freedom and creativity, but also required more responsibility and communication. In earlier stages, it was difficult to figure out equal tasks for each team member, especially because we were not fully set on all of our ideas yet. The process was an iterative one and there were no formulas or guidelines for us to follow closely, and this lead to lots of communication - for large or small steps - during the design process, to ensure that everybody is on the same page and being productive with their time.

In the end, each member on our team rotated taking leadership depending on the component that they focused mainly on and where that fit into the design process. For example, teammates in charge of the motor and PID took charge in the earlier stages of the process, while the teammates in charge of the LCD display displayed more leadership towards the later stages in the process.

We were able to develop all of our technical skills immensely, with the development of the mobile app, PID controls, and dc motors controls being new to all of us, as well as giving some team member’s the opportunity to better understand concepts our groups implemented in the past, like twitter, graphing, hardware, and sonar, where they may not have had the chance before.

We also were somewhat limited in our timeline to fully implement the features we desired, and our team feels that we would have greatly benefited from learning how to add multithreaded streams to our implementation to allow for reading from the sonar, PID updates, and track plotting all at the same time. Another feature we would have learned a lot from was the implementation of a switch which could safely power on and power off the pi through a pure hardware “on” process, and the running of a shell script to turn the pi off and then disconnect the power. Albeit our team is at least satisfied that we can understand the theory behind such features even if we did not get to see them.

In short, the most important things we take away from this project are the soft skills we needed to work efficiently, the technical skills and tasks we were able to accomplish, skills we were able to research and understand. These skills will not only carry through to Project 2, but also future projects and jobs we undertake, and in this we find this project's true value.

**F. References and bibliography**

**References**

<http://developer.android.com> - main reference used for app development  
<https://github.com/ivmech/ivPID> - main reference used for PID development

<https://twitter.com/cpen291team11> - our team’s twitter account where tweets & photos are sent

**Files**

main.py - Python file containing complete code

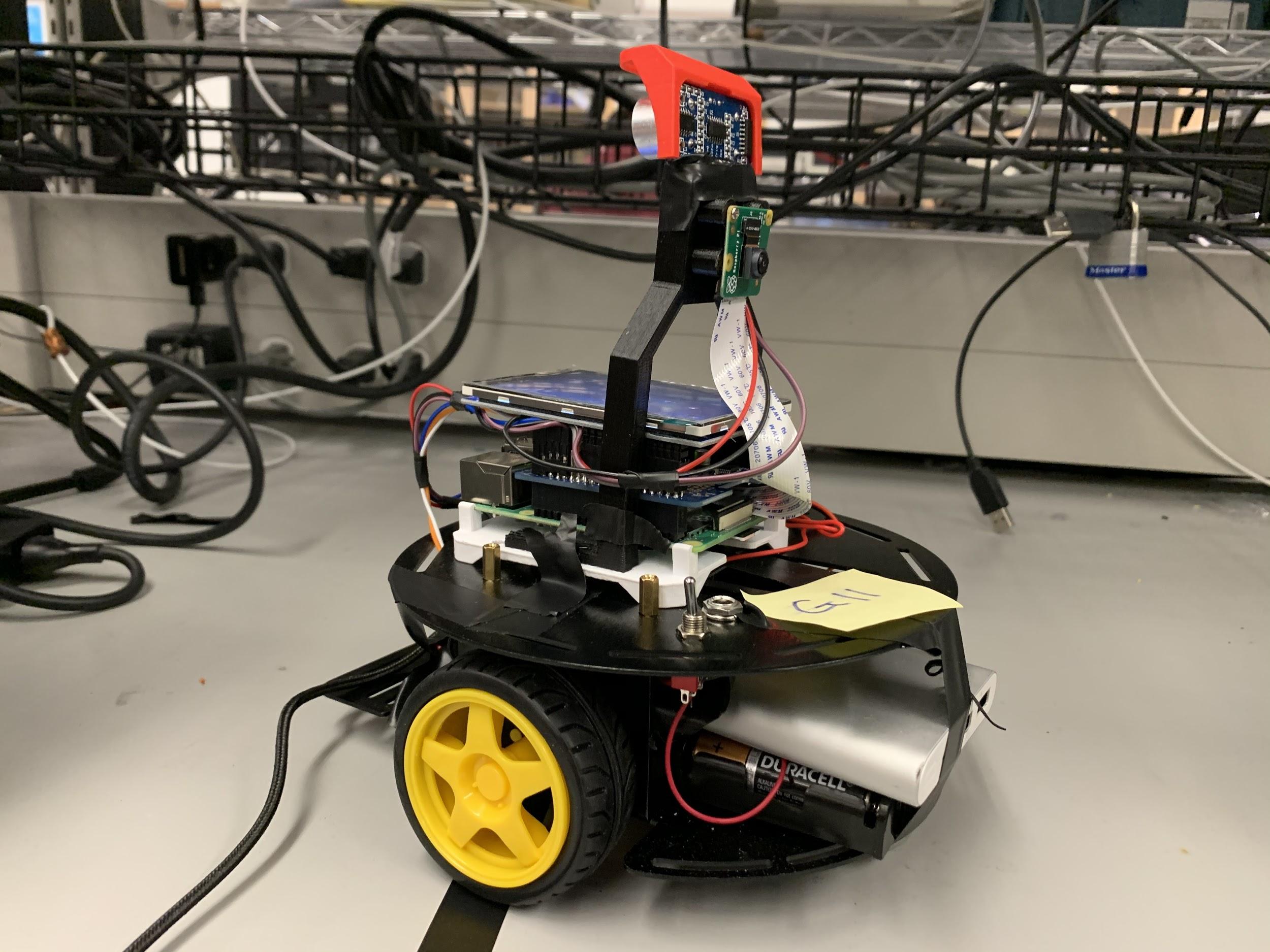
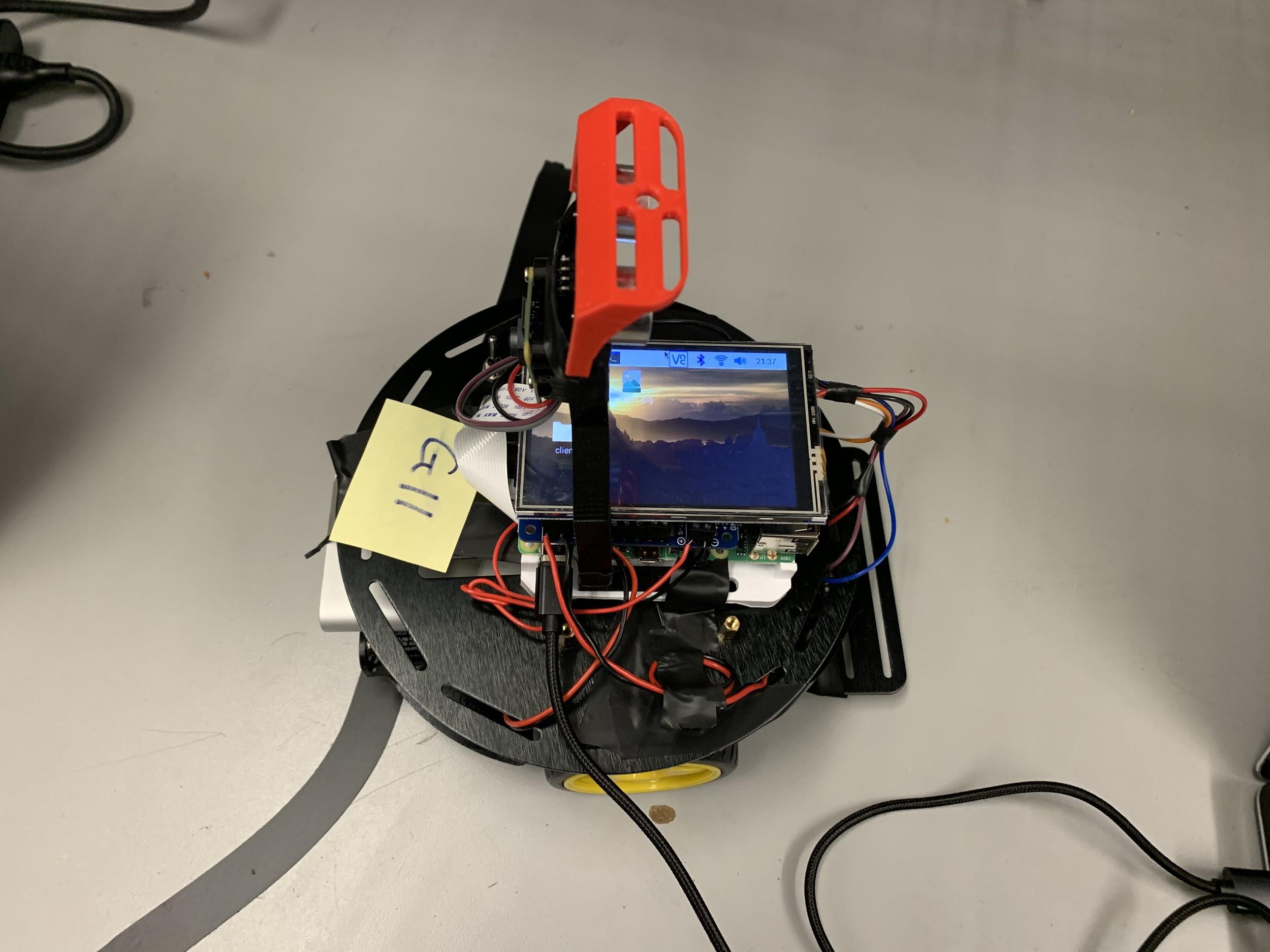
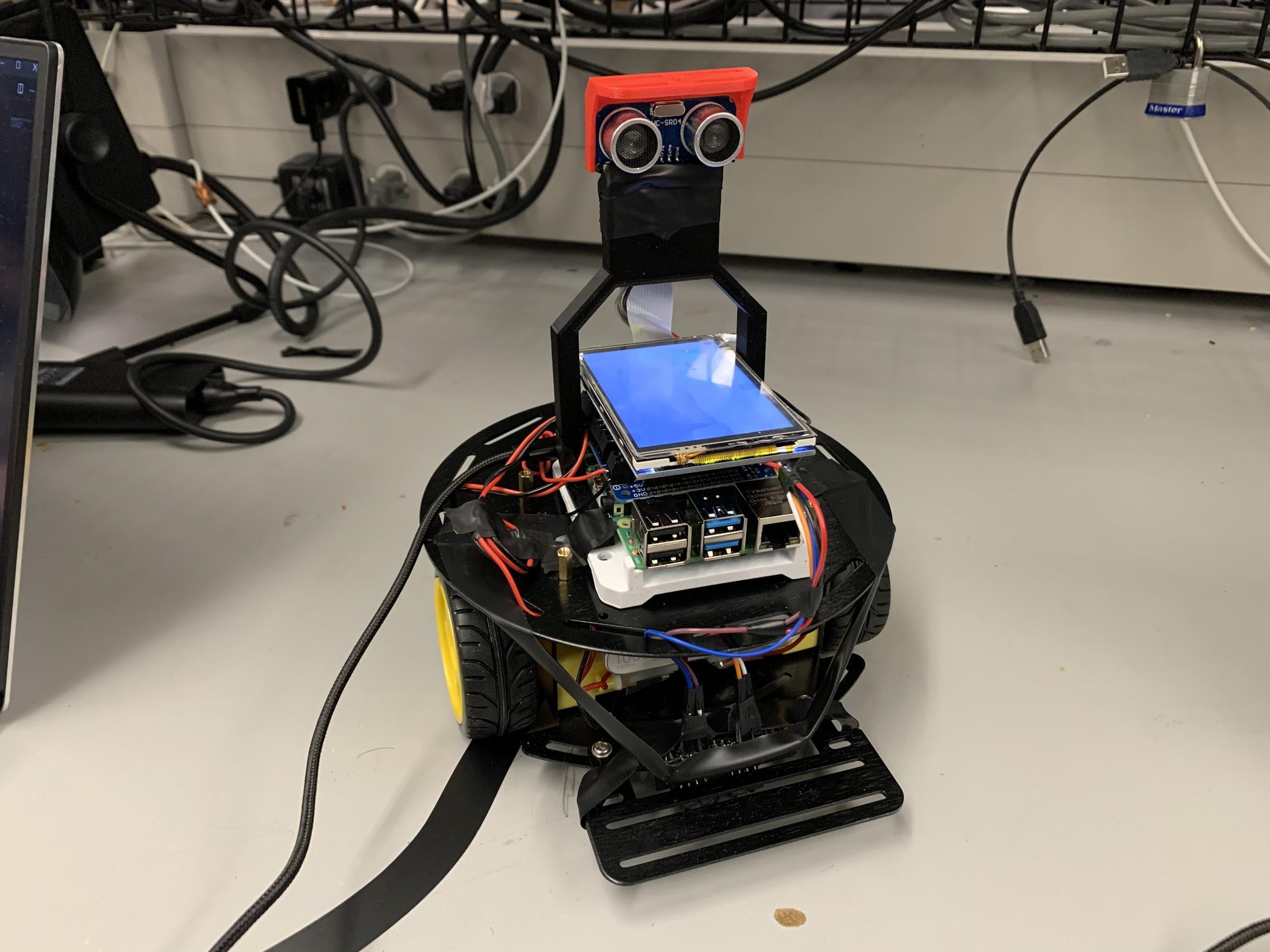
project2Fritzing.fzz - fritzing file of the experiments

291\_Project1\_Report.docx - project report document

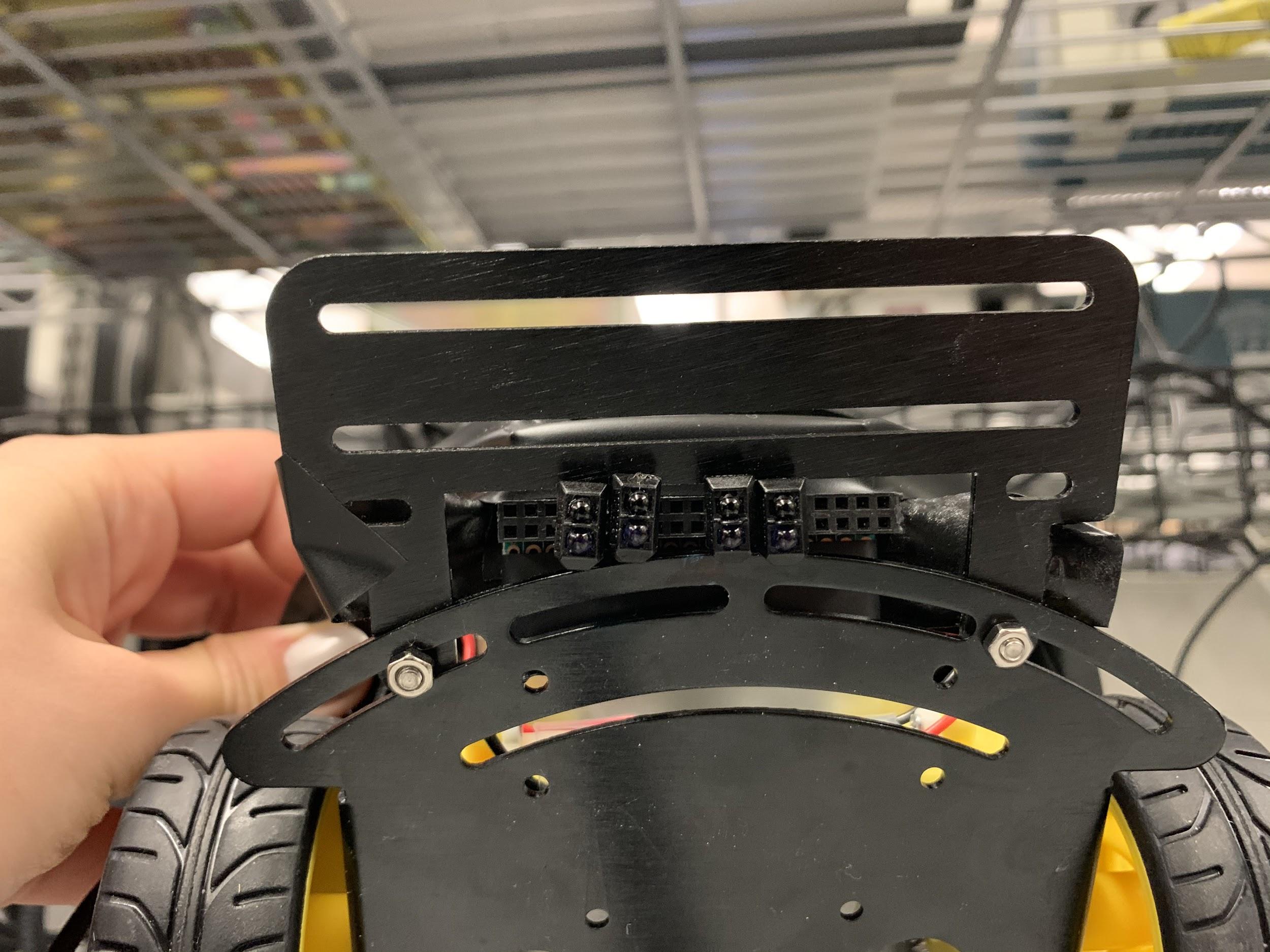
RobotController - folder containing project files for the mobile application

**Appendix A – Robot pictures**

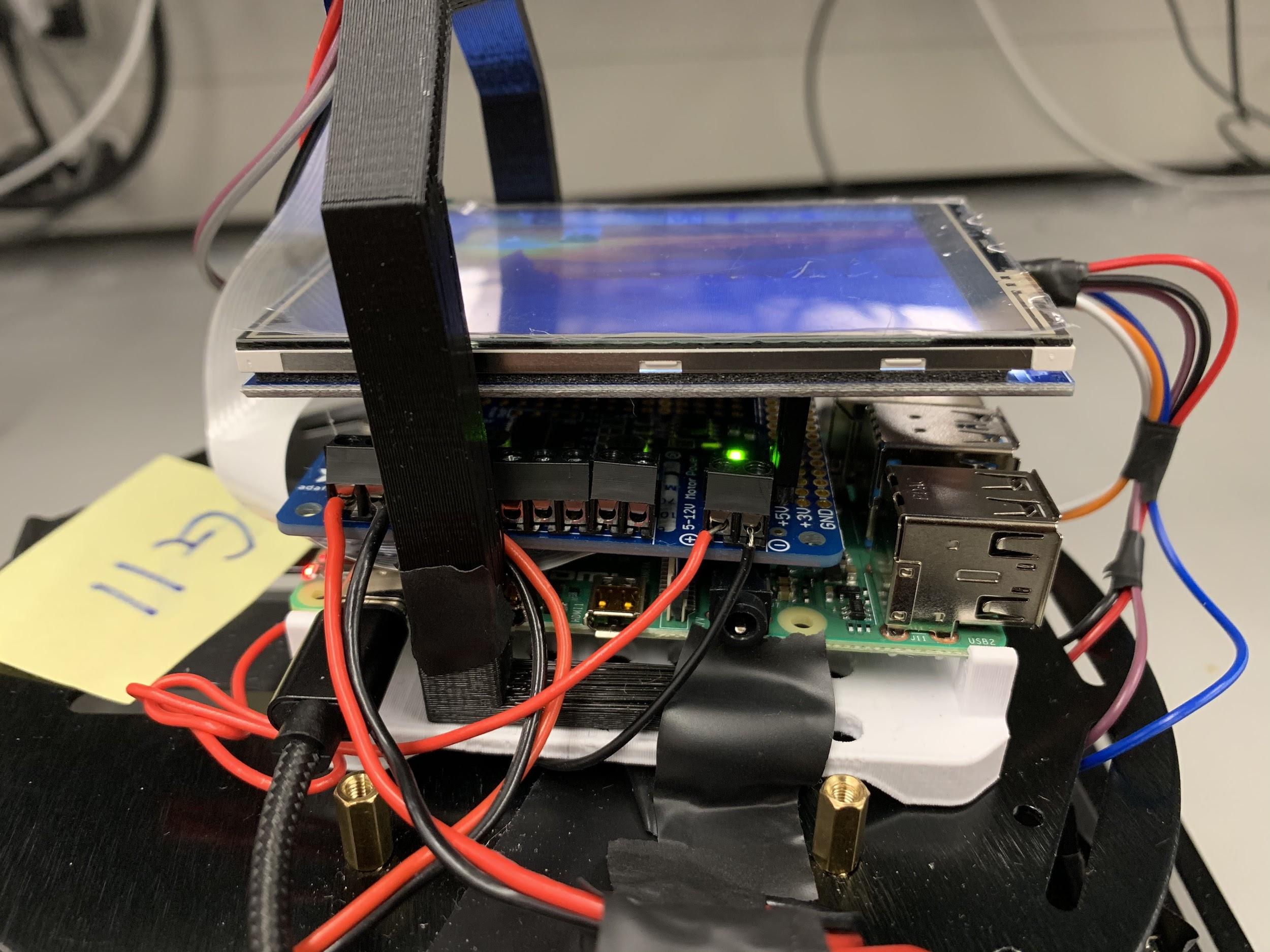
A-1) Front, top, and side view of our robot



A-2) wiring and position of our optical sensors



A-3) wiring and placement of the Motor Hat and Raspberry Pi on the robot



A-4) Back view of our robot with camera and sonar sensor wiring



**Appendix B-a - Python Code for Robot**

import RPi.GPIO as GPIO

import time

#-----------------------------------------------------------------#

# LCD Plot code

from tkinter import \*

import matplotlib

import matplotlib.pyplot as plt

matplotlib.use('TkAgg')

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from matplotlib.figure import Figure

import math

import time

class Plotter:

def \_\_init\_\_(self, frame):

self.fig = Figure(figsize=(5,4))

self.figCanvas = FigureCanvasTkAgg(self.fig, master=frame)

self.p = self.fig.add\_subplot(1,1,1)

# method for setting the labels of the current plot

def set\_labels(self, title=None, xlabel=None, ylabel=None):

self.p.set\_title(title)

self.p.set\_xlabel(xlabel)

self.p.set\_ylabel(ylabel)

# method for showing the plot at a specific location on the grid

def show\_plot(self, xloc=0, yloc=0, padx=0, pady=0):

self.figCanvas.get\_tk\_widget().grid(row=xloc,column=yloc,padx=padx,pady=pady)

# method for making the plot disappear off of the current window

def hide\_plot(self):

self.figCanvas.get\_tk\_widget().grid\_remove()

# method for clearing the plot and displaying the specified data

def graph(self, xdata, ydata, color, title, xtitle, ytitle):

self.p.clear()

self.set\_labels(title, xtitle, ytitle)

self.p.plot(xdata, ydata, color)

self.figCanvas.draw()

plt.xticks(rotation=45, ha='right')

plt.autoscale(enable=True, axis='both', tight=None)

# robotGUI class to display the graph

class RobotGUI:

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

# declare Tkinter window and variables to calculate the path travelled

self.root = Tk()

# make fullscreen

self.root.overrideredirect(True)

self.root.geometry("{0}x{1}+0+0".format(self.root.winfo\_screenwidth(), self.root.winfo\_screenheight()))

self.turnVal = 0

self.speed = 1

self.x = 0

self.y = 0

self.yDir = 1

self.xDir = 0

self.hyp = 0

self.theta1 = math.pi / 2

self.phi1 = 0

self.theta2 = 0

self.phi2 = math.pi - self.theta2

self.xVals = []

self.yVals = []

self.flag = 1

self.plotLength = 300

self.running = False

self.message = StringVar()

self.message.set("Welcome to PathPlotter!")

button = Button(self.root, text="Quit", command=self.stop\_program)

button.grid(row=0, column=0)

self.plot = Plotter(self.root)

self.plot.show\_plot(1,0)

self.begin\_measure()

self.root.mainloop()

def begin\_measure(self):

for i in range(0, self.plotLength):

self.running = True

# getting the turnvals from the track array

self.turnVal = track[round(i/self.plotLength \* len(track))]

self.speed = 0.4

# calculating angles and direction of travel

self.hyp = math.sqrt(math.pow(self.x, 2) + math.pow(self.y, 2) + self.speed

- 2 \* math.sqrt(math.pow(self.x, 2) + math.pow(self.y, 2))

\* math.cos(math.pi \* (1 - self.turnVal / 2)))

if self.x != 0:

self.theta1 = math.atan(self.y / self.x)

else:

self.theta1 = math.pi / 2

self.theta2 = self.turnVal \* math.pi / 2

self.phi2 = math.pi - self.theta2

self.phi1 = math.asin(math.sin(self.phi2) / self.hyp)

self.y = self.hyp \* math.sin(self.phi1 + self.theta1)

self.x = self.hyp \* math.cos(self.phi1 + self.theta1)

self.x = round(self.x, 2)

self.y = round(self.y, 2)

self.xVals.append(self.x)

self.yVals.append(self.y)

self.plot.graph(self.xVals, self.yVals, 'b', "Path Travelled So Far", "x", "y")

# self.loop = self.root.after(1, lambda: self.begin\_measure())

# method for terminating the program

def stop\_program(self):

# try:

# if self.running:

# self.root.after\_cancel(self.loop)

self.root.destroy()

self.root.quit()

# except:

# print("Program terminated")

# method for running the current program

def run\_program(self):

self.message.set("Program running...")

if not self.running:

self.begin\_measure()

#-----------------------------------------------------------------#

# PID controller code

class pid:

"""PID controller."""

def init(self, Kp, Ki, Kd):

# set origin time as the current time

origin\_time = time.time()

# Gains for each term

self.Kp = Kp

self.Ki = Ki

self.Kd = Kd

# Corrections calculated in update (outputs)

self.Cp = 0.0

self.Ci = 0.0

self.Cd = 0.0

self.previous\_time = origin\_time

self.previous\_error = 0.0

def Update(self, error):

# dt is change in time (time interval between calls to update)

# if dt is negative, return 0

current\_time = time.time()

dt = current\_time - self.previous\_time

if dt <= 0.0:

return 0

# de is change in error (from previous)

de = error - self.previous\_error

self.Cp = error

self.Ci += error \* dt

self.Cd = de / dt

self.previous\_time = current\_time

self.previous\_error = error

return (

(self.Kp \* self.Cp) # proportional term

+ (self.Ki \* self.Ci) # integral term

+ (self.Kd \* self.Cd) # derivative term

)

#-----------------------------------------------------------------#

# Sensors code

# setup the sensors according to their ports

GPIO.setwarnings(False)

rightIRTrackingPinL = 12

rightIRTrackingPinR = 16

leftIRTrackingPinL = 20

leftIRTrackingPinR = 21

# setup left and right pins array to store the left and right sensor ports

leftPins = [leftIRTrackingPinL, leftIRTrackingPinR]

rightPins = [rightIRTrackingPinL, rightIRTrackingPinR]

# define optical sensor setup function that calls GPIO.setup to setup the

# sensors as input

GPIO.setmode(GPIO.BCM) # Set the GPIO pins as BCM

GPIO.setup(leftIRTrackingPinL, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

GPIO.setup(leftIRTrackingPinR, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

GPIO.setup(rightIRTrackingPinL, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

GPIO.setup(rightIRTrackingPinR, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

# define get optical value function that retuns the values read from the

# optical sensor in binary format

def getOptiValues(pins):

value = 0

for pin in pins:

value = value << 1

value = value | GPIO.input(pin)

return value

# define a cleanup destroy function that calls GPIO.cleanup

def destroy():

GPIO.cleanup() # Release resource

# call the setup functions to setup the optical sensor

# setupOptiSensor()

#-----------------------------------------------------------------#

# Sonar code

# setting the ports for ultrasonic sensor

TRIG = 26

ECHO = 19

def setupSonar():

# setting the input and output

GPIO.setmode(GPIO.BCM)

GPIO.setup(TRIG, GPIO.OUT)

GPIO.setup(ECHO, GPIO.IN)

def getSonar():

GPIO.output(TRIG, False)

time.sleep(0.1)

# time between the pulse 10uS

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

# starting the pulse

pulse\_start = time.time()

while GPIO.input(ECHO) == 0:

pulse\_start = time.time()

# ending the pulse

pulse\_end = time.time()

while GPIO.input(ECHO) == 1:

pulse\_end = time.time()

# calculatGPIOn of distance

sound\_speed = 331.5 + (0.6\*21)

pulse\_duratGPIOn = pulse\_end - pulse\_start

distance = pulse\_duratGPIOn \* sound\_speed \* 50

distance = round(distance, 2)

return distance

#-----------------------------------------------------------------#

# Twitter code

from twython import Twython

# define the keys and tokens for twitter

consumer\_key = 'IbZYLMhINCxuxRLd4OyrM2Ph2'

consumer\_secret = 'pBfsBvgDYBTjcXuYNgJXl5DhwXNEosZStpjCu4az7SgTvgyMcx'

access\_token = '1220402110610604032-0Eca0tLBOjE2TKd1fPbh6BfZzMZ4u2'

access\_token\_secret = 'GH4fuU3riSkWVh0UGODAasDCI2gN8Qqpc7AkGlQnwzHQZ'

# initialize a twitter object by calling Twython initializer

twitter = Twython(

consumer\_key,

consumer\_secret,

access\_token,

access\_token\_secret

)

# define a postTweet function that posts different tweets to @cpen291team11

def postTweet(distance, speed, state, imageFile):

if state == "end":

message = "I have finished running the track!"

if distance <= 10:

message += "There is an object " + str(distance) + " cm away and we are approaching at " \

+ str(speed) + " m/s. Brace for evasive maneuvers!!"

elif distance >= 100:

message += "There are no obstacles in sight! The closest barrier is " \

+ str(distance) + "cm away and we are approaching at " + str(speed) + " m/s."

image = open(imageFile, 'rb')

response = twitter.upload\_media(media=image)

media\_id = [response['media\_id']]

twitter.update\_status(status=message, media\_ids=media\_id)

print("Tweeted: " + message)

#-----------------------------------------------------------------#

# Camera code

from picamera import PiCamera

# initialize a PiCamera object by calling PiCamera

camera = PiCamera()

# define a takePhoto function that takes a picture and stores it to a

# certain directory

def takePhoto():

pic = '/home/pi/Desktop/image1.jpg'

camera.capture('/home/pi/Desktop/image%s.jpg' % 1)

return pic

# setup the Sonar sensors

setupSonar()

# if the sonar reading is less than 10 cm, take a picture and tweet it

if getSonar() <= 10:

postTweet(getSonar(), 3, "end", takePhoto())

#-----------------------------------------------------------------#

# Motors code

from adafruit\_motorkit import MotorKit

from adafruit\_motor import stepper

# define a MotorKit object to control the motors

kit = MotorKit()

track = []

# robot\_stop function that stops the motors (throttle = 0)

def robot\_stop():

kit.motor1.throttle = 0.0

kit.motor2.throttle = 0.0

#

global count

count = 0

global entry

entry = 0

# robot\_move function that change the motors to the input throttle

# for delay amoung of time (in seconds)

def robot\_move(left, right, delay):

kit.motor2.throttle = left

kit.motor1.throttle = right

global entry

global count

entry += (right-left)/2

count += 1

if count > 100:

track.append(entry/count)

time.sleep(delay)

# robot\_run function that simply sets the left and right motors to

# the input speeds indefinitely

def robot\_run(left, right):

kit.motor2.throttle = left

kit.motor1.throttle = right

# a factor variable that controls how much faster the robot moves when

# going straight than turning

factor = 1.1

# define the robot\_ir function that moves with the optical sensor

# and PID controller output

# adjuster is a value from 1 to -1

def robot\_ir(speed, adjuster, times, flag, blockade):

# we set additional varible left and right that is equal to speed to

# control the left and right motors

left = speed

right= speed

# flag is the variable that controls

if flag == 1 and blockade == 0:

# according to the adjuster value, the robot either turns or goes forward

# if adjuster is 0, the robot goes forward

if adjuster == 0:

robot\_move(left\*factor, right\*factor, times)

# if adjuster is positive, the robot turns right by throttling the right

# motor with speed - adjuster

elif adjuster > 0:

robot\_move(left, right-adjuster, times)

# if adjuster is positive, the robot turns left by throttling the left

# motor with speed - adjuster

elif adjuster < 0:

robot\_move(left+adjuster,right, times)

# else if flag is 0 and blockade is 0 stop the robot

elif flag == 0 and blockade == 0:

robot\_stop()

# else if blockade is 1, move the robot back

elif blockade == 1:

robot\_stop()

time.sleep(1)

robot\_move(-left, -right, time)

time.sleep(1)

robot\_stop()

#-----------------------------------------------------------------#

# Line tracking code

import math

# dictionaries for the error value according to the optical sensor readings

dictLeftErrors = {0b00: 0.7, 0b01: 0, 0b11: -0.7, 0b10: -2}

dictRightErrors = {0b00: -0.7, 0b10: 0, 0b11: 0.7, 0b01: 2}

# gap count variable that keeps track of how many consecutive all-white detections

# the optical sensors have

global gap\_count

gap\_count = 0

# define getError functions that obtain error value is the dictionary

# declared above and combines the left and right optical sensors

def getErrorRight():

dataR = getOptiValues(rightPins)

print(dataR)

error = dictRightErrors[dataR]

return error, dataR

def getErrorLeft():

dataL = getOptiValues(leftPins)

print(dataL)

error = dictLeftErrors[dataL]

return error, dataL

def getErrorOverall():

errorL, dataL = getErrorLeft()

errorR, dataR = getErrorRight()

# increments the gap\_count variable if the readings of all optical sensors

# are 0

global gap\_count

if (dataL is 0b00 and dataR is 0b00):

gap\_count += 1

# otherwise clear the gap\_count variable (set to 0)

else:

gap\_count = 0

return errorL + errorR

# flag indicates whether we want the robot to move or not

flag = 1

# calls a robot\_stop at the beginning of every program start

robot\_stop()

# gap variable for how long the robot can move when it doesn't detect anything

# we obtained this value through calibration and calculation (~3cm gap)

gap = 100

# define a demo function that runs the robot autonomously to track line

def demo():

# loops indefinitely for autonomous line tracking

while True:

try:

# set sampling rate and speed

sampling\_rate = 2000

speed = 0.

# call pid to update and get the output values that the robot is suppose to turn

pid.init(pid, Kp=0.1, Ki=0, Kd=7)

# output is between 1 and -1 and the robot turns right if output is positive and

# left if its negative, goes forward if 0

output = pid.Update(pid, getErrorOverall())

# if gap\_count is bigger than the 3cm, stops the robot and tweets and graphs path travelled

if (gap\_count >= gap/factor):

robot\_stop()

postTweet(getSonar(), 3, "end", takePhoto())

rg = RobotGUI()

break

# uncomment this to print the output

# print(output)

# calls robot\_ir

robot\_ir(speed, 2\*math.atan(output)/math.pi\*speed, 1/sampling\_rate, flag, 0)

# catches a keyboardInterrupt and stops the robot, tweets and graphs path travelled

except KeyboardInterrupt:

robot\_stop()

postTweet(getSonar(), 3, "end", takePhoto())

rg = RobotGUI()

return

# catches IO error and continues to run

except IOError:

print("IO error")

destroy()

#-----------------------------------------------------------------#

# Bluetooth handling code

import glob

from bluetooth import \*

import re

# Maximum speeds for the motorhat motors

MAX\_FORWARD = 1

MAX\_BACKWARDS = -1

# converts input string in the format "x,y" into a tuple of integers to use for

# motor speed calculation

def get\_data(data):

tup = tuple(filter(None, data.split(',')))

return (int(tup[0]), int(tup[1]))

# gets the speed of the left and right motors based on x and y coordinates of

# joystick on app

def get\_speeds(x, y):

# centre coordinates of the joystick in the app

cX = 290

cY = 590

#radius of joystick outer circle

radius = 220

# calculting current displacement of joystick from centre

radX = x - cX

radY = cY - y

left\_speed = 0

right\_speed = 0

if (x != 0 and y != 0):

# angle calculated using simple cartesian coordinates

angle = math.degrees(math.atan2(radY, radX))

# for angles > 180 degrees, atan calculates the negative angle, so it is

# readjusted to compensate

if angle < 0:

angle += 360

# right and left motor speeds are adjusted based on the current quadrant

# the joystick is in

# when the joystick is aligned towards the right of the joystick area,

# the left motor is set to maximum speed and the right motor speed is

# increased to cause robot to turn right

# the same applies for left turns

if angle <= 90:

left\_speed = MAX\_FORWARD

right\_speed = (angle % 91) / 90 \* MAX\_FORWARD

elif angle <= 180:

right\_speed = MAX\_FORWARD

left\_speed = ((180 - angle) % 91) / 90 \* MAX\_FORWARD

elif angle <= 270:

right\_speed = MAX\_BACKWARDS

left\_speed = ((angle - 180) % 91) / 90 \* MAX\_BACKWARDS

else:

left\_speed = MAX\_BACKWARDS

right\_speed = ((360 - angle) % 91) / 90 \* MAX\_BACKWARDS

else:

return (0, 0)

# displacement calculated based on joystick distance from centre

displacement = math.sqrt(radX \* radX + radY \* radY)

# speeds are adjusted relative to the displacement, ie. further from the

# centre causes faster speeds

left = left\_speed \* displacement / radius

right = right\_speed \* displacement / radius

# if the input goes out of bounds and causes the speed to go outside of the

# indicated range, it is capped at max speed

if left > MAX\_FORWARD:

left = MAX\_FORWARD

if right > MAX\_FORWARD:

right = MAX\_FORWARD

if left < MAX\_BACKWARDS:

left = MAX\_BACKWARDS

if right < MAX\_BACKWARDS:

right = MAX\_BACKWARDS

return (left, right)

# creating a new bluetooth server socket using rfcomm bluetooth protocols

server\_sock=BluetoothSocket( RFCOMM )

server\_sock.bind(("",PORT\_ANY))

server\_sock.listen(1)

port = server\_sock.getsockname()[1]

# setting a uuid that both client and user can use to connect to the service

# this uuid is a standard one that is used for rpi bluetooth communication

uuid = "94f39d29-7d6d-437d-973b-fba39e49d4ee"

# advertising the service to allow for client connections

advertise\_service( server\_sock, "LineTrackerServer",

service\_id = uuid,

service\_classes = [ uuid, SERIAL\_PORT\_CLASS ],

profiles = [ SERIAL\_PORT\_PROFILE ],

)

while True:

print("Waiting for connection on RFCOMM channel ", port)

# blocking call that waits for a client to connect to the server before

# proceeding

client\_sock, client\_info = server\_sock.accept()

print ("Accepted connection from ", client\_info)

while True:

try:

# receieves data from the client

data = client\_sock.recv(1024)

if len(data) == 0:

print("no data")

break

# data from client will be in string format so it is first decoded

direction = data.decode(encoding='UTF-8')

# checks which type of data the client has sent and acts accordingly

if direction == 'Demo':

demo()

elif (re.search('[a-zA-Z]', direction)):

robot\_stop()

else:

motor\_vals = get\_data(direction)

speeds = get\_speeds(motor\_vals[0], motor\_vals[1])

left\_speed = speeds[0]

right\_speed = speeds[1]

robot\_run(left\_speed, right\_speed)

except IOError:

print("IOError")

continue

# on a KeyboardInterrupt, the connection is cancelled and the socket is

# closed

except KeyboardInterrupt:

print("disconnected")

client\_sock.close()

server\_sock.close()

break

**Appendix B-b: Java Code for Android Application**

**package** com.example.robotcontroller;

**import** android.annotation.SuppressLint;

**import** android.bluetooth.BluetoothAdapter;

**import** android.bluetooth.BluetoothDevice;

**import** android.bluetooth.BluetoothSocket;

**import** android.content.Intent;

**import** android.graphics.Rect;

**import** android.os.Build;

**import** android.os.Handler;

**import** android.os.Message;

**import** android.util.Log;

**import** android.view.MotionEvent;

**import** android.view.View;

**import** android.widget.Button;

**import** android.widget.ImageView;

**import** android.widget.Toast;

**import** androidx.annotation.Nullable;

**import** androidx.annotation.RequiresApi;

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

**import** android.os.Bundle;

**import** java.io.IOException;

**import** java.util.Set;

**import** java.util.UUID;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**public class** MainActivity **extends** AppCompatActivity {

**private static final int *REQUEST\_ENABLE\_BT*** = 1;

**private final** BluetoothAdapter **bluetoothAdapter** = BluetoothAdapter.*getDefaultAdapter*();

**private static final** String ***PI\_ADDRESS*** = **"DC:A6:32:30:25:A9"**;

**private** ClientThread **clientThread**;

**private float dX** = 0;

**private float dY** = 0;

**private** Rect **rect**;

**private float joystickX**;

**private float joystickY**;

@SuppressLint(**"ClickableViewAccessibility"**)

@RequiresApi(api = Build.VERSION\_CODES.***O***)

@Override

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

**super**.onCreate(savedInstanceState);

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

*// setting a listener that checks when the connect button is pressed*

Button connect\_bt = findViewById(R.id.***connect***);

connect\_bt.setOnClickListener(**new** View.OnClickListener() {

@Override

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

*// checks if device has bluetooth*

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

*// checks if bluetooth is on and if not, request user to turn on bluetooth*

**if** (!**bluetoothAdapter**.isEnabled()) {

Intent enableBtIntent = **new** Intent(BluetoothAdapter.***ACTION\_REQUEST\_ENABLE***);

startActivityForResult(enableBtIntent, ***REQUEST\_ENABLE\_BT***);

} **else** {

*// if bluetooth is enabled, connect to the robot*

connectBluetooth();

}

}

}

});

*// listener for demo button that sends demo direction to raspberry pi if connection exists*

Button demo = findViewById(R.id.***demo***);

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

@Override

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

**if** (**clientThread** != **null**)

**clientThread**.**connectedThread**.write(**"Demo"**.getBytes());

}

});

**final** ImageView joystick = findViewById(R.id.***joystick***);

**final** ImageView joystick\_out = findViewById(R.id.***joystick\_out***);

*// main touch listener for the joystick*

joystick\_out.setOnTouchListener(**new** View.OnTouchListener() {

@Override

**public boolean** onTouch(View v, MotionEvent event) {

*// checks if the user just touched the joystick, if it is moving the joystick or*

*// if the user has just let go*

**switch** (event.getAction()) {

**case** MotionEvent.***ACTION\_DOWN***:

*// stores the current location of the joystick, corresponding to the centre*

**joystickX** = joystick.getLeft();

**joystickY** = joystick.getTop();

*// checks the displacement of the joystick and the location of the screen*

*// touch*

**dX** = joystick.getX() - event.getRawX();

**dY** = joystick.getY() - event.getRawY();

**return true**;

*// case for when finger is being dragged across screen*

**case** MotionEvent.***ACTION\_MOVE***:

*// if there is an active connection with the server, send the x and y*

*// coordinates of the joystick to the robot*

**if** (**clientThread** != **null**)

**clientThread**.**connectedThread**.write(((**int**) joystick.getX() + **","** + (**int**) joystick.getY()).getBytes());

*// checks if the joystick is within the bounds of the outer joystick circle*

*// if outside, joystick is centered and the Stop directive is sent to the*

*// robot to cause it to stop*

**if** (event.getRawX() > 575 || event.getRawX() < 175 ||

event.getRawY() > 1000 || event.getRawY() < 600) {

joystick.setX(**joystickX**);

joystick.setY(**joystickY**);

**if** (**clientThread** != **null**)

**clientThread**.**connectedThread**.write(**"Stop"**.getBytes());

} **else** {

*// if the joystick is within the bounds of the outer circle, the joystick*

*// image is moved to the current location of the screen touch*

joystick.animate()

.x(event.getRawX() + **dX**)

.y(event.getRawY() + **dY**)

.setDuration(0)

.start();

}

**return true**;

*// case for when the screen is let go*

**case** MotionEvent.***ACTION\_UP***:

*// centres the joystick again*

joystick.animate()

.x(**joystickX**)

.y(**joystickY**)

.setDuration(0)

.start();

*// if there is an active connection, direct robot to stop*

**if** (**clientThread** != **null**)

**clientThread**.**connectedThread**.write(**"Stop"**.getBytes());

**break**;

}

**return true**;

}

});

}

*// method for checking if bluetooth has been enabled upon a request*

@Override

**protected void** onActivityResult(**int** requestCode, **int** resultCode, @Nullable Intent data) {

**super**.onActivityResult(requestCode, resultCode, data);

**if** (requestCode == ***REQUEST\_ENABLE\_BT***) {

**if** (resultCode == ***RESULT\_OK***) {

connectBluetooth();

} **else** {

Toast.*makeText*(**this**, **"Bluetooth not approved"**, Toast.***LENGTH\_SHORT***);

}

}

}

*// method for searching through paired devices and getting the bluetooth device corresponding*

*// to the raspberry pi*

**private void** connectBluetooth() {

**boolean** found = **false**;

BluetoothDevice pi = **null**;

*// gets all devices that have been paired to the current android device*

Set<BluetoothDevice> paired\_devices = **bluetoothAdapter**.getBondedDevices();

**if** (paired\_devices.size() > 0) {

*// loops through all previously paired devices until pi is found*

**for** (BluetoothDevice bd : paired\_devices) {

**if** (bd.getAddress().equals(***PI\_ADDRESS***) || bd.getName().equals(**"raspberrypi"**)) {

found = **true**;

pi = bd;

}

}

}

**if** (!found) {

Toast.*makeText*(**this**, **"Device not found"**, Toast.***LENGTH\_SHORT***).show();

} **else** {

**clientThread** = **new** ClientThread(pi);

**clientThread**.run();

}

}

**private class** ClientThread **extends** Thread {

**private final** String **TAG** = ClientThread.**class**.getName();

**private final** BluetoothSocket **mmSocket**;

**private final** BluetoothDevice **mmDevice**;

**private boolean connected**;

**public** ConnectedThread **connectedThread**;

**private final** UUID **MY\_UUID** = UUID.*fromString*(**"94f39d29-7d6d-437d-973b-fba39e49d4ee"**);

**public** ClientThread(BluetoothDevice device) {

*// Use a temporary object that is later assigned to mmSocket*

*// because mmSocket is final.*

BluetoothSocket tmp = **null**;

**mmDevice** = device;

**connected** = **false**;

**try** {

*// Get a BluetoothSocket to connect with the given BluetoothDevice.*

*// MY\_UUID is the app's UUID string, also used in the server code.*

tmp = device.createRfcommSocketToServiceRecord(**MY\_UUID**);

} **catch** (IOException e) {

Log.*e*(**TAG**, **"Socket's create() method failed"**, e);

}

**mmSocket** = tmp;

}

**public void** run() {

*// Cancel discovery because it otherwise slows down the connection.*

**bluetoothAdapter**.cancelDiscovery();

**try** {

*// Connect to the remote device through the socket. This call blocks*

*// until it succeeds or throws an exception.*

**mmSocket**.connect();

} **catch** (IOException connectException) {

*// Unable to connect; close the socket and return.*

**try** {

**mmSocket**.close();

} **catch** (IOException closeException) {

Log.*e*(**TAG**, **"Could not close the client socket"**, closeException);

}

**return**;

}

*// The connection attempt succeeded. Perform work associated with*

*// the connection in a separate thread.*

manageMyConnectedSocket(**mmSocket**);

}

*// Closes the client socket and causes the thread to finish.*

**public void** cancel() {

**try** {

**mmSocket**.close();

} **catch** (IOException e) {

Log.*e*(**TAG**, **"Could not close the client socket"**, e);

}

}

**private void** manageMyConnectedSocket(BluetoothSocket socket) {

**connected** = **true**;

Toast.*makeText*(MainActivity.**this**,

**"Successfully connected to device with address: "** + **mmDevice**.getAddress(),

Toast.***LENGTH\_LONG***).show();

**connectedThread** = **new** ConnectedThread(socket);

*//connectedThread.run();*

}

}

**private static final** String ***TAG*** = **"MY\_APP\_DEBUG\_TAG"**;

**private** Handler **handler**; *// handler that gets info from Bluetooth service*

*// Defines several constants used when transmitting messages between the*

*// service and the UI.*

**private interface** MessageConstants {

**public static final int *MESSAGE\_READ*** = 0;

**public static final int *MESSAGE\_WRITE*** = 1;

**public static final int *MESSAGE\_TOAST*** = 2;

*// ... (Add other message types here as needed.)*

}

**private class** ConnectedThread **extends** Thread {

**private final** BluetoothSocket **mmSocket**;

**private final** InputStream **mmInStream**;

**private final** OutputStream **mmOutStream**;

**private byte**[] **mmBuffer**; *// mmBuffer store for the stream*

**public** ConnectedThread(BluetoothSocket socket) {

**mmSocket** = socket;

InputStream tmpIn = **null**;

OutputStream tmpOut = **null**;

**handler** = **new** Handler();

*// Get the input and output streams; using temp objects because*

*// member streams are final.*

**try** {

tmpIn = socket.getInputStream();

} **catch** (IOException e) {

Log.*e*(***TAG***, **"Error occurred when creating input stream"**, e);

}

**try** {

tmpOut = socket.getOutputStream();

} **catch** (IOException e) {

Log.*e*(***TAG***, **"Error occurred when creating output stream"**, e);

}

**mmInStream** = tmpIn;

**mmOutStream** = tmpOut;

}

**public void** run() {

**mmBuffer** = **new byte**[1024];

**int** numBytes; *// bytes returned from read()*

*// Keep listening to the InputStream until an exception occurs.*

**while** (**true**) {

**try** {

*// Read from the InputStream.*

numBytes = **mmInStream**.read(**mmBuffer**);

*// Send the obtained bytes to the UI activity.*

Message readMsg = **handler**.obtainMessage(

MessageConstants.***MESSAGE\_READ***, numBytes, -1,

**mmBuffer**);

readMsg.sendToTarget();

} **catch** (IOException e) {

Log.*d*(***TAG***, **"Input stream was disconnected"**, e);

**break**;

}

}

}

*// Call this from the main activity to send data to the remote device.*

**public void** write(**byte**[] bytes) {

**try** {

**mmOutStream**.write(bytes);

*// Share the sent message with the UI activity.*

Message writtenMsg = **handler**.obtainMessage(

MessageConstants.***MESSAGE\_WRITE***, -1, -1, **mmBuffer**);

writtenMsg.sendToTarget();

} **catch** (IOException e) {

Toast.*makeText*(MainActivity.**this**, **"Error occurred when sending data"**, Toast.***LENGTH\_LONG***);

Log.*e*(***TAG***, **"Error occurred when sending data"**, e);

*// Send a failure message back to the activity.*

Message writeErrorMsg =

**handler**.obtainMessage(MessageConstants.***MESSAGE\_TOAST***);

Bundle bundle = **new** Bundle();

bundle.putString(**"toast"**,

**"Couldn't send data to the other device"**);

writeErrorMsg.setData(bundle);

**handler**.sendMessage(writeErrorMsg);

}

}

*// Call this method from the main activity to shut down the connection.*

**public void** cancel() {

**try** {

**mmSocket**.close();

} **catch** (IOException e) {

Log.*e*(***TAG***, **"Could not close the connect socket"**, e);

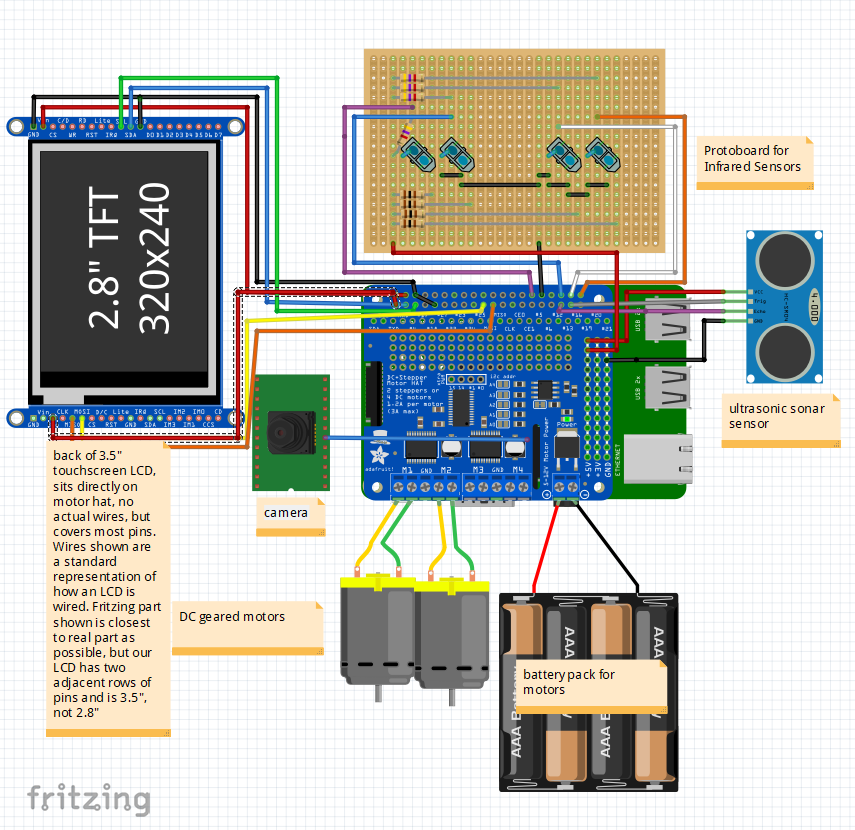
}

}

}

}

**Appendix C - Fritzing**

****

**Appendix D - GitHub**

There was an even spread of the workload amongst our group members, with some of the work being done together on one computer while debugging and testing directly with the Pi, and a good segment of the work, especially during the portion of the project where we used separate files, that the group members all worked on their own computers. This process was not perfect, as sometimes all the commits would be from a single account for a period of time, and this also doesn’t account for the time spent on the hardware and fritzing implementation.

**Appendix E – Complete Component list**

* Raspberry Pi
* 2WD Mobile Platform robot kit
* Motor Hat
* 2x DC Motors
* 3.5” TFT LCD Display for final demo, 1” TFT LCD for first demo.
  + <http://www.kumantech.com/kuman-35-inch-touch-screen-tft-lcd-display-spi-with-touch-pen-for-raspberry-pi-3b-pi-2b-pi-zero-w-pi-a-b_p0442.html?fbclid=IwAR3yYOxRVp23q_tMWj9pfTLT0T5kVnOPfuhFs4RfwVZ2mXSb20nyRVzxNJ0>
  + <http://www.lcdwiki.com/MHS-3.5inch_RPi_Display>
  + Approximately CAD $30.00
* RPi Camera attachment
* Ultrasonic Sensor
* 5x AAA Batteries
* Android phone

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

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

To streamline the development process, we clearly delegated tasks to each teammate both through a README on Github as well as in person, wherever possible. We developed a group culture of openness and honesty so that every team member was able to contribute and test their ideas effectively. This also allowed for extensive constructive criticism to occur so that we could rapidly prototype effectively and develop and polish a large breadth of ideas fast into our core additional functionalities and core features.

To maintain a consistent and efficient level of communication, our team used a messenger group chat to stay in touch while not meeting face to face. This was especially practical during the reading break since many of our team members were off-campus at this time, and yet we were still able to coordinate meetings, delegate tasks, and to complete the project during this time.

Another way we ensure that everyone is working equitably for the project is through the analysis of our Github commits in terms of lines added and removed, past peer reviews, and qualitative milestones reached for the hardware and for the report. This allows for a simple diagnosis of everyone’s workload, and gives the opportunity to each teammate to see where they may be falling behind, and thus incentivizing consistent self-improvement for the teammate as well as ensuring we meet deadlines.

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.

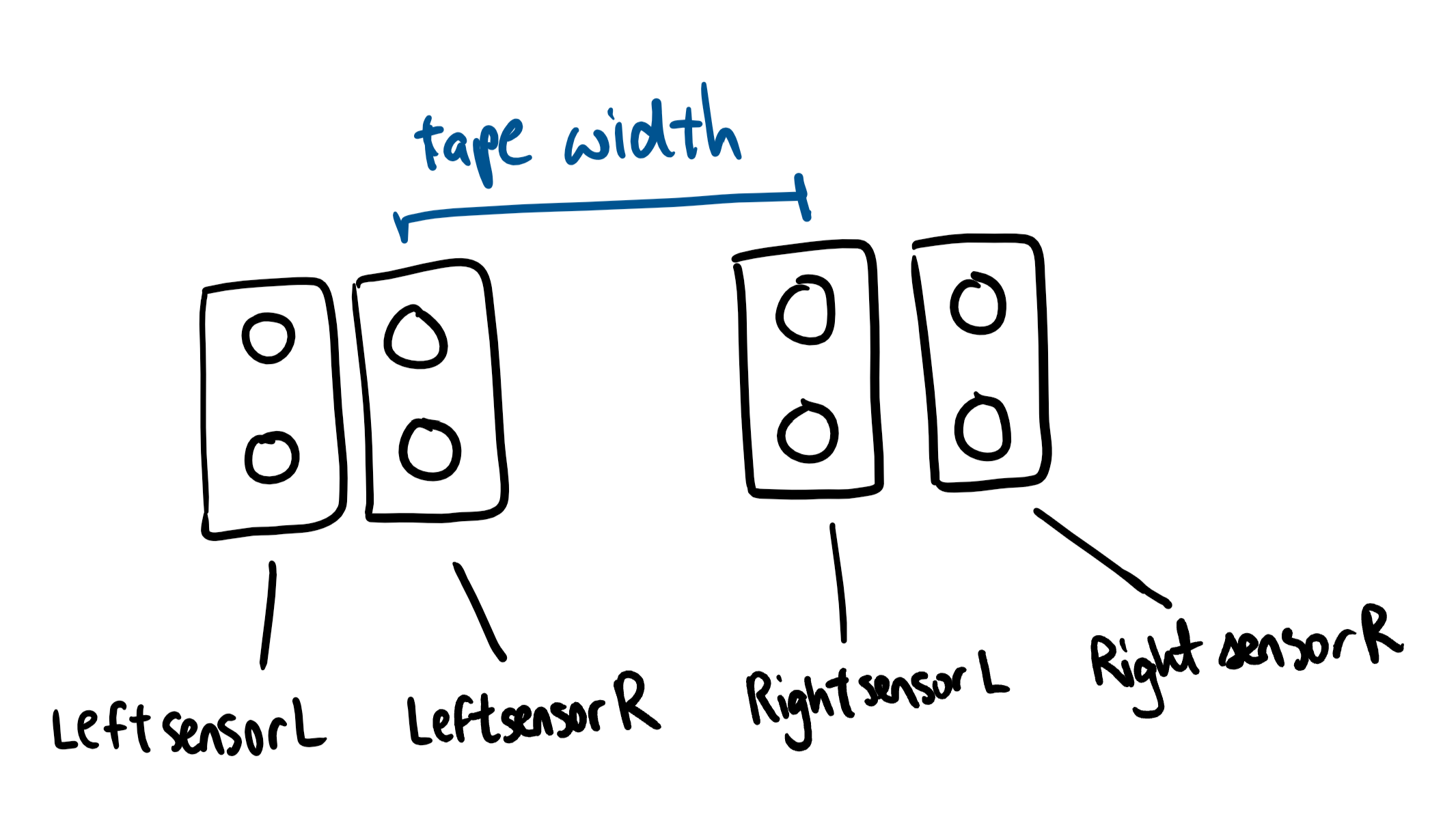
1. **Use of process**: Describe your approach to adapt and apply a general design process for any additional feature. What was your approach?

We used an iterative rapid prototyping process to incrementally add new features to our line tracking robot. We began with the base requirements given to us by the lab description, then went on to address the open-ended problems we still saw with the system, adding multiple new features to improve client engagement and to be able to fully visually display the data we received from the RPi.

1. **Constraint identification**: Explain the constraints that you must consider in the design of the additional functionalities.
   1. Hardware constraints: The main hardware constraint with this project was limiting the pins used by all of our smaller components since the LCD uses so many pins. Another factor that added to this was how to design a circuit that is clean and safe in a dense space, and which will not short with the metal casing of the robot or within itself. This was made more critical as we switched away from the breadboard to a more permanent protoboard solution.
   2. Software constraints: There were three very large software constraints we faced, getting reliable PID values, developing an app which was compatible with the RPi and our other hardware/software, and integrating additional features into the robot while maintaining the relative timing of the robot.
      1. The PID values were determined according to the Ziegler-Nichols method as described earlier, but since the battery level, ambient light, angle of the sensors, shadows, and other power drawing components were added and removed as we tested each day, the values we determined for the PID were effectively obsolete whenever we made a change. This allowed us to become more efficient in determining new PID values which can turn quickly, smoothly, without oscillation, while still being able to stop fast and make sharp turns.
      2. We decided to make an android app which controls the robot via bluetooth. We began developing on android studio but realised MIT App Inventor had a better interface for new app developers. We made an app which connects to a bluetooth server and sends text when a button is pressed. The text was decoded on the server side. Due to some technical anomalies, when the app client connects to a bluetooth server, the app stops responding and closes.
      3. Since the reliability of the robot movement is heavily dependant on getting very accurate readings as quick as possible, we had issue adding features like dynamic track plotting and wall detection, as those increased the latency of our robot either directly in the case of the sonar, while has to have sleep function calls to align its readings, or in the case of the tkinter graph, which requires too much time to virtually add points to the plot and make complex calculations at the rate of 3000Hz we were using for the IR sensors.
   3. Quality vs Quantity Tradeoff consideration: A consideration our group always keeps in mind is the feasibility of our ambitions for a project given its deadline and requirements. One limitation our team sometimes has, and one which we are consistently fixing, is that we try to add to many additional features into a project before we can get the base functionality off the ground. While this allows for a more holistic view of a project even from its early stages, it can overshadow the main goal of the project until the last portion of the project.
2. **Solution generation**: Explain at least two possible alternative additional features that your group rejected due to technical reasons and explain why.
3. Bluetooth App with multiple features like live stream of pi camera, default mode for line tracking, joystick control for moving the robot and controlling the sensors connected to the raspberry pi. The live streaming was rejected because the feature required web development which was not recommended for project 1. The other features of the app were not tested because we could not get the basic functionality of sending information from the client to the server in bluetooth due to technical anomalies.
4. Using servos to control the Ultrasonic Sensor angle for wall detection. This feature was supposed to be implemented for the wall detection mode of our line following robot but it was rejected because of space limitations on that robot and code conflicts while trying to move, detecting wall and torating the sensor. A similar design to change the angle of the ultrasonic sensors for maximum accuracy of readings was also rejected due to the time cost during runtime as well as the hardware cost.
5. A switch which can control the powering on and off of the pi through a pure hardware “on” process, and the running of a shell script to turn the pi off and then disconnect the power. This required a lot of additional hardware and posed the risk of damaging the power supply to the pi, and since we didn’t have the time resources to devote to this optional, yet highly useful mechanism, nor many pins on the pi to utilize for this, the feature was revoked.
6. Multithreaded synchronous processes for the PID controls, sonar reading, and dynamic track plotting were rejected due to the high overhead associated with the undertaking of a threaded implementation, due to the need to synchronize group data with locks, which would likely cause the program to not run much faster than it already does in its single threaded approach, since almost every function call between these three features requires accurate data about the state of the others and the ability to interrupt other processes when required, ie to add a new data point to the plot or to stop the robot quickly when it detects an obstacle. Our team worked around this issue, making some compromises around the implementation of our additional features’ integration so that the core functionality stayed resolute, although we were still able to research about and appreciate the implementation of such a multithreaded solution.
7. **Solution Assessment**: Explain how you tested and assessed the viability and then correctness of your group’s additional features.
   1. The correctness of the sonar, twitter, and track plotter were all assessed by the accuracy of their response to their predefined function conditions. We can easily measure if the sonar is measuring X cm with a ruler or by eye, and the track plotter final image has to look like the tape below it or else it would need refining. The twitter bot can easily be verified with print statements and checking our team’s twitter account for updates.
   2. The mobile app control’s correctness was verified when it was able to connect to and turn the robot at the user’s will.
   3. The robot’s ability to make turns, speedup, and stop was tested by how well it adhered to the track at its defined speed (which speeds up on straight patches), how little oscillation and jittering there is in its adjustments, and how sharp of a turn it can handle. This correctness was wholly dependent on the accuracy of the PID value determination process outlined before.

**Appendix G - Other**

G-1) Reflective optical sensor configuration diagram

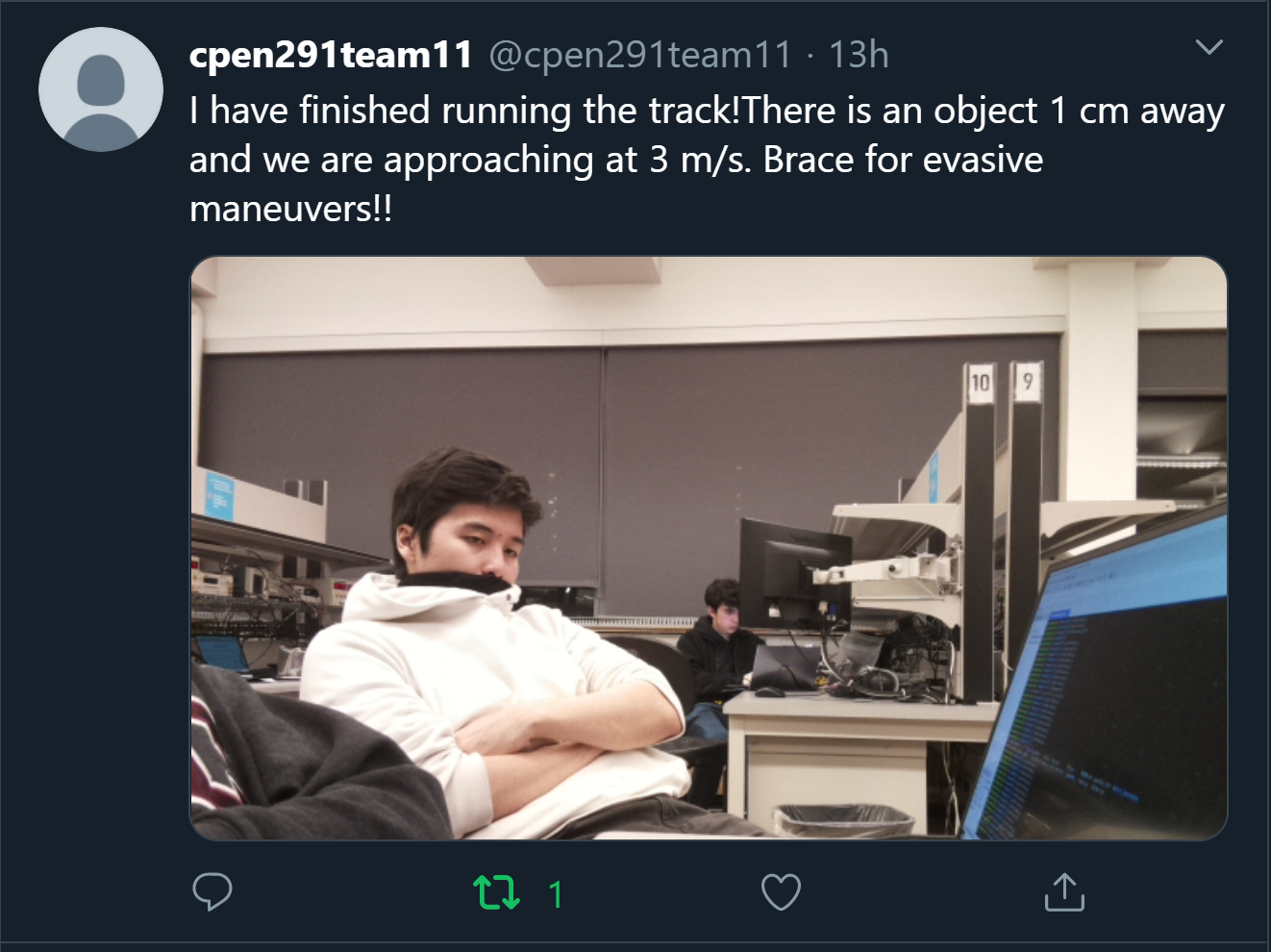


G-2) Graphical representation of GUI graph algorithm



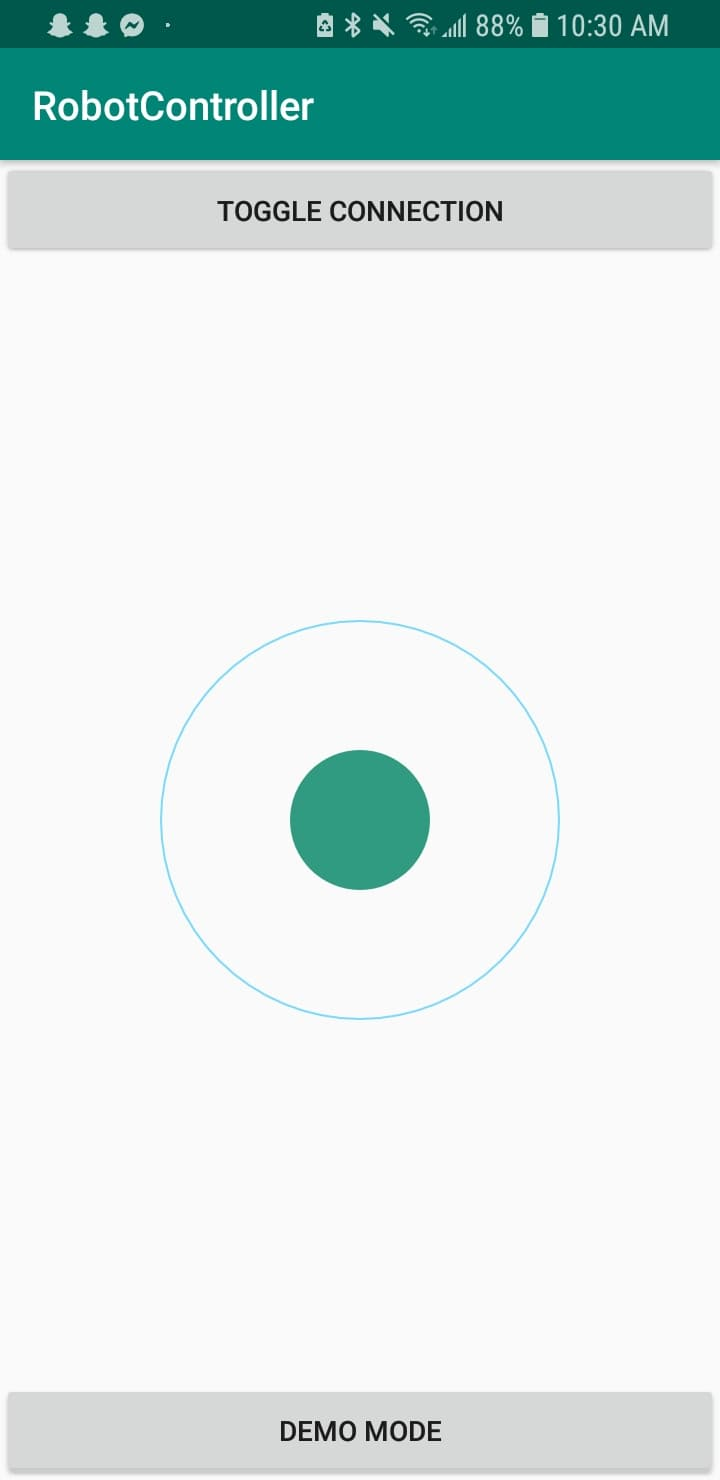
This figure illustrates the angles and coordinates we calculated in our implementation of the path plotting, taking in a speed and a turnvalue, and assuming an initial direction of (x,y) = (0,1), we calculate the robot’s location and are able to roughly draw the path it travels.

G-3) screenshot of a tweets that our robot sends at the end of running a track





G-4) Screenshot of the interface of our mobile app and logo



**Appendix H - Explanation of desired implementation for Demo**

This section will be used to elaborate on the issues faced in the demo which were not occuring the night before the demo. Approximately 45 minutes before the demo, we identified that the hardware connection to the Infrared sensors was malfunctional, and was not performing the intended autonomous line tracking correctly, if at all. This was in stark contrast to the night before, where our additional implementation code, which exists all in the single main.py file, executed in the following manner. First the Pi connects through bluetooth to the android phone app, and gives the user the option to run an enhanced autonomous demo mode, or to use the joystick to control the robot manually. Once in this “enhanced demo mode”, The robot first checks for obstacles within 50 cm before initiating on its track and turning based on the PID reading. This is where our actual demo fell short as the reading were inconsistent with the performance and data we received not only in the demo of the main functionality, but also with the data we received with the same python code just 8 hours earlier. Nonetheless, the robot was supposed to take an average of its speeds and turn values over each 1 second interval, or 3000 readings, and to store these values in an array of pairs for later use in the plotting of the graph. Once the path terminates and the robot halts, it would check for a waving hand with the sonar to indicate that it should tweet the image it took with the camera at that time, along with some diagnostics about speed and distance data. Then it would read each value of speed and turn value from the array of pairs, and dynamically calculate the x and y coordinates of the path it travels, so that a user can analyze what path their robot has travelled for their own record. This would then return control back to the joystick in the app and the cycle starts again.

After the demo we realized that the reason the sensors were not working was due to a mistake made around 8am, namely that the infrared sensors had fallen out of their sockets, and when we plugged them back in, we made the mistake of orienting them the wrong way, which caused the sensor to read opposite value if any values at all. This was an unfortunate mistake, but one we solved and re-demoed for on the same day after reaching an agreement with our instructor. :)