**CPEN 291   
Project 1 Report**

**A. Group info**

Lab section: L2A (*L2A or L2B)* Group #: A-G10 (*Example: A-G1*)   
Group’s Lab Bench #s: 10 and 15 (*Example: 1 and 2*)   
Student names:

|  |  |
| --- | --- |
| Justin Hua | Steven Huang |
| Eric Lyu | Nicholas Ng |
| Angus Wu | Winston Wu |

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

**B.1 Hardware:**

**B.1.1 3 reflective optical sensors:**

Optical sensors are arranged in a row, 1 cm from each other. The first sensor was used to determine if the robot should turn left, the middle sensor for going straight and the third sensor for going right. We arranged the sensors in this configuration to account for turns and curves in the track.

**B.1.2 Raspberry Pi Camera:**

The camera is mounted on the front of the robot, giving us an optical view of where the robot was headed. Also used for one of our additional functionalities: streaming the footage to a website/server. Besides that, we also use camera to capture multiple pictures at a time whose resolutions are set 128x128 in order to fit into our LCD screen.

**B.1.3 TFT LCD:**

The LCD screen is mounted on the back of the robot to allow for easy viewing. The real-time “video” which is implemented by using multiple pictures was displayed on the TFT LCD in order to help users identify the situation in front of the robot. For better usage of our website, we also included using a stream(threading) to create a concurrent program to display updated stuff on LCD while other functionalities can still work at the same time.

**B.2 Software:**

**B.2.1 The Algorithm:**

We implemented the algorithm to check whether the line is either straight or turning left or right. To detect if the line was straight, the code just checked whether the middle reflective optical sensor was on. For left, the code checked whether the middle and left sensors were on or just the left sensor was on and turned left slightly. Similarly, for turning right. The goal for the algorithm is to re-center the robot so that only the middle sensor is reading on and the others are off. Initially before the readjustment algorithm runs, we have the code calibrate the upper and lower bounds for the analog readings of the current floor surface. The readjustment of the motors was done with the PD part of PID. If any of the turning conditions are met then the code will calculate a new motor adjustment value such that one wheel will speed up that much and the other will slow down that much to re-center the robot to have the middle sensor aligned with the black line. For 90 degree turns, we have a flag value that tracks the last turning value right before the sensors all read off, then the algorithm will turn the robot left or right depending on the flag until it reads the middle sensor.

**B.2.2 MCP3008 ADC converter and code change during demo:**

We included MCP3008 ADC converters as our main functionality so that the readings can be more accurate, and the robot can be tested on any kinds of surfaces with any kinds of track materials. During the first demo, we need to adjust the lower bound and upper bound of reading of a track in the code manually to enable the robot to run on different surfaces. This part of the code is included in Appendix A.1. However, this is inconvenient, and actually not allowed in the demo. Consequently, we develop the algorithm in Appendix G after our first demo. The new code can detect the values of track and its surrounding surfaces before the robot actually moves. It records 10 values of middle sensor which is the track, and 10 values of right and left sensors which is the surface and adjust the upper and lower bound values automatically according to these values. The equations and parameters used for calculation are obtained by trial and error.

**B.2.3 Two separate code files:**

We used two separate Raspberry Pi, one for the main functionality, the other for camera and LCD because of lack of pins for the LCD screen. We tried to combine all of the code into one file, but it got all messed up. So, we keep two copies of file, one(main.py) for the main functionality and web and mobile control, and the other(cameraControl.py) for the camera function and LCD display.

**B.2.4 Headless Pi use:**

Using the VNC Viewer app, we were able to access the Raspberry Pi without connecting to it physically. Some challenges include it not working at first and having to manually reconfigure the Raspberry Pi’s desktop resolution to 1920 x 1080 instead of auto. Using the Pi in headless mode allowed for us to mount the Pi onto the robot without it having a power cable attached to the wall, making the robot fully portable.

**B.3 Battery-operated robot:**

We implemented the battery-operated robot using a portable power bank to power the Raspberry Pi and the 5 AA batteries for the motors. The challenges that came with using AA batteries was that the dc motors are super power hungry as well as decently heavy at the same time. And so, the way to combat this was to have batteries for testing and batteries for the actual demo. Challenges related to using the power bank was that there is a minimum current requirement for the raspberry pi to function properly and so we needed to find a power bank with the right specification. The caveat with using one that satisfied the specification was that it was heavy and the motors would have to carry the extra weight of the power bank.

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

**C.1 Camera:**

In this project, we projected the screen of camera on to the LCD, so as the robot is moving, LCD will display the camera screen with some delay. We first save the pictures taken from the camera to local desktop, then we let the LED to read the picture. We repeat this step continuously in a while loop with framerate set to 30, so that LED is like playing a video. Since LED needs to read the picture every time from the desktop, there will be a little be of delay between each picture. Also, since the LCD’s screen size is limited to 128\*128, we need to rescale the image taken from the camera to an appropriate size.

We simply place the camera onto the robot so that it can take pictures as it moves, and then the LCD will display the pictures that’s taken. Also, the camera view can be seen through VNC stream.

**C.2 Website/Server:**

HTTP server that allows the user to input commands to the robot through the internet. Footage of the robot’s journey, recorded using the Raspberry Pi Camera, is also streamed to the server. The server is implemented using the HTTPServer class found in Python’s standard library. And is hosted on a local host.

The server class contains four main methods: do\_HEAD, \_redirect, do\_GET, and do\_POST. do\_HEAD sets up the webpage with the html contents (html code embedded as string in do\_GET). The host address is the Raspberry Pi’s IP address. The port number is set to 8000. The user can connect to the webpage using the host address and port number. For us, the address to connect is https://137.82.226.231:8000. \_redirect is used for refreshing the page back to the main page after each request from the user is handled. do\_GET sets up the website with several form submission buttons (“Forward”, “Backward”, “Left”, “Right”, and “Stop”) and an info about the current CPU temperature of the Raspberry Pi. When the user clicks on the buttons, a request is sent to the server, the webpage is implicitly redirected to a specific subpage. For example, when the user clicks on the “Forward” button, the page is redirected to https://137.82.226.231:8000/Forward. The do\_POST can then extract the request “Forward” from the subpage link and calls the specific function (moveForward(speed)) in (motorFunctions) to control the robot, thus handling the request. Then, the do\_POST method calls \_redirect to go back to the main webpage to wait for the next user input.

We have two versions of code of the website, one on each Raspberry pi. One is entirely for motor control. The other one controls the camera, VNC screen preview, and the LCD screen. The reason for separating them on two Raspberry Pi is that there are not enough pins for both LCD and MCP3008 to work together. Also, with two separate Pi, the wiring will be cleaner.

**C.3 Mobile phone control over Bluetooth:**

For this project, we implemented a client-server connection over Bluetooth between an Android phone and the robot hardware, namely the Raspberry Pi. This feature relies on setting up Bluetooth and installing appropriate Bluetooth libraries for the Raspberry Pi to work as a Bluetooth server. In addition, this feature requires that our Android phone be paired with the Raspberry Pi prior to using the app.

On the server side, using Python and the PyBluez module for Bluetooth connectivity, the Raspberry Pi opens a server socket using the RFCOMM protocol to listen in for connections and accepts the phone’s connection request. It then begins to listen for messages through its input stream and then performs specific robot motor functions (such as moving forward, backward, turning) based on the received message. When the client sends an exit signal or disconnects, the server will close the sockets and shut down.

The mobile phone app, created in Java and Android Studio, uses the built-in Android Bluetooth libraries to find the paired Raspberry Pi and connect on a separate thread. The thread creates an output stream to the server and writes messages to it as the user interacts with the interface, controlling the robot. For example, a user pressing the button to move forward will send a message in bytes to the server, which will then be interpreted as a move command and subsequently move the robot forward. Through this implementation, there is no additional hardware setup or configuration needed.

**D. Test and evaluations**

**D.1 Robot:**

We made many test tracks using electrical tape including: a circle, an 8-shaped loop, a curvy track, a right-angled track, a straight-line and a track with crosses. We first tested if the robot would continue forward by itself using the straight-line track. Then we added crosses and gaps to ensure that the robot would keep moving forward. For testing if the robot would turn left and right, we started with low-angled turns and then moved to the circle track to test a higher-angled turn. Once we made sure that worked, we tested using the 8-shaped loop which combined curves, circles, crosses and gaps.

**D.2 Motor direction:**

We found that the back part of the robot will be the better place to put the optical sensors. However, by doing that all the movements were reversed. We simply switched the power and ground wire for M2, then everything works fine.

**D.3 Website:**

We firstly did not include the CPU temperature in the webpage. However, we found that the webpage cannot be correctly shown without the instruction: self.wfile.write(html.format(temp[5:]).encode("utf-8")). So, we added the CPU temperature to trigger the instruction to fix the problem. It also becomes an additional feature. We can now monitor the temperature of the Raspberry pi remotely.

We were thinking about streaming the pictures onto the websites. However, the picture overwrites itself quickly after it is created. Also, the streamed version may be some binary codes instead of the real picture. Considering the limited time that we have, we finally gave up streaming on the website. Instead, we use VNC to stream the pictures.

**D.4 Mobile app and Bluetooth server:**

When we first implemented the app and server, we first tested these components separately by just setting up a local Bluetooth server on a laptop (as it was faster to test on than the Raspberry Pi at the time) as well as a basic loop to read from the input stream and printing to console when any button on the app was pressed. This helped us ensure that we could get a proper connection and communicate over the socket. When this was successful, we began testing multiple buttons that would send different messages to make sure that we were able to handle different button inputs from the app.

We also began to experiment with test and hold functionality, where actions would be continuous until the button on the app is released. Again, we tested this through setting up a local server on a laptop, getting a successful connection, and sending inputs and messages to console.

Our last two tests involved integrating the server onto our Raspberry Pi and into our main code. We wanted to run the same tests on the Pi as the laptop to ensure that we could have our Pi act as the Bluetooth server. We faced many challenges with this because of trying to configure the hardware to support Bluetooth development as well as problems with Python not recognizing the Bluetooth modules needed. One major problem associated was that our system would not recognize any (Bluetooth) advertisable device. We found that this was likely because of the OS configurations, and so because of this, we had to re-configure settings in Raspbian to fix this as well as re-installing some Python modules. When these problems were fixed, we were able to successfully run the tests using the Pi.

Our last test, which was integrating our server into the main code, also included actually controlling the motors from the app. This was just adapting code to run different commands based on the message rather than printing it to console. We integrated our server into the main code and were successfully able to control the robot’s motors from the app while retaining all the functions of the main code. This completed our testing of the server.

**E. Conclusions and Reflections**

**E.1. Lessons Learned**

**E.1.1 Hardware:**

We learned that we should double check components are properly attached. Having the sensors attached to the bottom of the robot made them very lose since they are subject to the effects of gravity. The ground wire of the battery supply was also too short, and we had to extend it by attached it to another wire. It was important to make sure the ground wire and its extension was always connected properly. Learning from our previous projects on making sure cables are wired up properly, we just left all the wires of the LCD and motor connected instead of disassembling it at the end of every lab session.

We learned how to control the motors using an Adafruit motor hat. We also learned how to integrate many components together including: Raspberry Pi Camera, Motors/motor hat, and LCD display.

We learned that if the motor is not working in the direction we intended, we can just switch the voltage and ground wires on the Adafruit motor hat.

We learned that a single sensor was not enough and that five sensors was too many before finally deciding that three sensors was optimal.

We learned that the motor did not run at all when giving it low speed inputs (anything 0.2 or under).

We learned that using the sensors to read analog values made it necessary to make manual adjustments for different terrains and lighting conditions.

**E.1.2 Software:**

We learned how to create an Android app to send controls to the robot’s motors using Bluetooth

We learned how to create a web server to send controls to the robot’s motors

We learned how to stream the capture feed of the Raspberry Pi Camera to a web server and to the LCD display

**E.1.3 Teamwork:**

We learned that our team is good at dividing and conquering but also at combining our efforts. We first started off this project by breaking into two smaller groups, one working on software and the other on hardware. Then we split into even smaller groups to work on the server, Android app and the robot’s pathing algorithm. Then through constant communication between the team, we were all able to gain a sufficient understanding of all parts of the project and were able to combine these parts into one functional product.

**E.2 Tricks/Concepts:**

Developed an understanding of how the sensors read input. We had to account for all the cases (where one sensor was on and the other two weren’t, etc.) and how each corresponded to if the robot had to turn left, right, continue going forward, stop, etc.

When integrating all the different codes for components together, it was important to make sure each component’s code was sectioned off in order to make reading easier.

When debugging, we put print statements at various stages inside our while and if loops to see if it was hitting those lines. This made identifying the bug fast and made debugging simple.

**E.3 Reflection:**

**E.3.1 Teamwork:**

We found that we have made a huge improvement from the previous project in terms of communication between teammates. This time we worked in smaller and more close-knit groups in terms of development, but also having team-wide development sessions. This led us to become closer as a group and also improved our workflow.

**E.3.2 Project Management**

For this project, we decided to approach it with a more divide and conquer tactic compared to last time, while also having team meetings to unify our findings. By having a schedule for both work times and meeting times, we have managed this project much more efficiently compared to last time.

**E.3.3 Time Management**

Compared to the previous project, we have made much more efficient use of our time in the lab. By scheduling and prioritizing what needs to be done during a specific lab time, we have streamlined our project design process. By having every agree on tasks before starting, we save time by not having disagreements or digressing from the current task on hand.

**F. References and bibliography**

Adafruit Motor Hat

* <https://learn.adafruit.com/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi/overview>

HTTP Server

* <https://www.afternerd.com/blog/python-http-server/>
* <https://docs.python.org/3/library/http.server.html>

Raspberry Pi Camera

* <https://randomnerdtutorials.com/video-streaming-with-raspberry-pi-camera/>

Mobile app implementation reference

* <https://circuitdigest.com/microcontroller-projects/controlling-raspberry-pi-gpio-using-android-app-over-bluetooth>
* <https://developer.android.com/guide/topics/connectivity/bluetooth>
* <https://developer.android.com/training/basics/firstapp/building-ui>
* <https://developer.android.com/reference/android/view/View.OnTouchListener>
* <http://pages.iu.edu/~rwisman/c490/html/pythonandbluetooth.htm>

**Appendix A – Robot pictures**

**A picture containing indoor

Description automatically generatedA close up of a device

Description automatically generatedA close up of a device

Description automatically generatedA close up of electronics

Description automatically generatedA circuit board

Description automatically generatedA picture containing indoor, wall

Description automatically generated**

**Appendix B - Code**

**Appendix B.1 main.py(controlling the motor original version, needs to manually change the values)**

import sys

import os

import time

import math

import spidev

import bluetooth

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

from adafruit\_motorkit import MotorKit

from http.server import BaseHTTPRequestHandler, HTTPServer

#############################################

# Motor definitions and functions

#############################################

# setup the motor kit

kit = MotorKit()

# define left and right motors

LEFT\_MOTOR = kit.motor1

RIGHT\_MOTOR = kit.motor2

# move motors forward at defined speed

def moveForward(speed):

    LEFT\_MOTOR.throttle = speed

    RIGHT\_MOTOR.throttle = speed

def nudge(speed):

    moveForward(speed)

    time.sleep(0.1)

# rotate the robot left in place

def rotateLeftInPlace(speed):

    RIGHT\_MOTOR.throttle = speed

    LEFT\_MOTOR.throttle = -speed

    # need to call stop() to stop rotating

# rotate the robot right in place

def rotateRightInPlace(speed):

    LEFT\_MOTOR.throttle = speed

    RIGHT\_MOTOR.throttle = -speed

    # need to call stop() to stop rotating

# move motors backward at defined speed

def moveBackward(speed):

    LEFT\_MOTOR.throttle = -1 \* speed

    RIGHT\_MOTOR.throttle = -1 \* speed

# stop the robot

def stop():

    LEFT\_MOTOR.throttle = 0

    RIGHT\_MOTOR.throttle = 0

def turnLeft(speed):

    rotateLeftInPlace(speed)

    time.sleep(0.5)

    #stop()

def turnRight(speed):

    rotateRightInPlace(speed)

    time.sleep(0.5)

    #stop()

def turn180():

    turnLeft()

    turnLeft()

#############################################

# Sensor/spidev definitions and functions

#############################################

# setting up spi

spi = spidev.SpiDev()

spi.open(0,0)

spi.max\_speed\_hz=1000000

# Define sensor channels

lSensor = 0

mSensor = 1

rSensor = 2

# Function to read SPI data from MCP3008 chip

# Channel must be an integer 0-7

def ReadChannel(channel):

  adc = spi.xfer2([1,(8+channel)<<4,0])

  data = ((adc[1]&3) << 8) + adc[2]

  return data

#############################################

# Web server definitions and functions

#############################################

host\_name = '137.82.226.231'    # Change this to Raspberry Pi IP address

host\_port = 8000

class MyServer(BaseHTTPRequestHandler):

    def do\_HEAD(self):

        self.send\_response(200)

        self.send\_header('Content-type', 'text/html')

        self.end\_headers()

    def \_redirect(self, path):

        self.send\_response(303)

        self.send\_header('Content-type', 'text/html')

        self.send\_header('Location', path)

        self.end\_headers()

    def do\_GET(self):

        # generate the html web with four buttons to control the robot to move forward, backward, left and right

        html = '''

            <html>

            <body style="width:960px; margin: 20px auto;">

            <h1>CPEN291 Project 1 Raspberry Pi web robot control</h1>

            <p>Current temperature is {}</p>

            <form action="/" method="POST">

                Robot control: <br />

                <input type="submit" name="submit" value="Forward"> <br />

                <input type="submit" name="submit" value="Left">

                <input type="submit" name="submit" value="Right"><br />

                <input type="submit" name="submit" value="Backward"><br />

                <input type="submit" name="submit" value="Stop">

            </form>

            </body>

            </html>

        '''

        temp = os.popen("/opt/vc/bin/vcgencmd measure\_temp").read()

        self.do\_HEAD()

        self.wfile.write(html.format(temp[5:]).encode("utf-8"))

    def do\_POST(self):

        # we get the request from the user

        # call the specific function in motor functions to handle the request

        content\_length = int(self.headers['Content-Length'])        # Get the size of data

        post\_data = self.rfile.read(content\_length).decode("utf-8") # Get the data

        post\_data = post\_data.split("=")[1]                         # Only keep the value

        moveSpeed = 0.35

        if post\_data == 'Forward':

            moveForward(moveSpeed)

            print("car is moving forward")

        elif post\_data == 'Backward':

            moveBackward(moveSpeed)

            print("car is moving backward")

        if post\_data =='Left':

            turnLeft(moveSpeed)

            print("car is rotating left")

        elif post\_data == 'Right':

            turnRight(moveSpeed)

            print("car is rotating right")

        if post\_data == 'Stop':

            stop()

            print("stopped")

        self.\_redirect('/') # finished handling request, redirect back to the root url

#############################################

# Main loop function

#############################################

# this is for reading parameters given by the command line execution

control\_mode = sys.argv[1]

#default move speed for all three modes

moveSpeed = 0.35

if control\_mode == "1":

    print("bluetooth mode")

    # bluetooth control

    # get server socket and set UUID and port number

    server\_socket=bluetooth.BluetoothSocket( bluetooth.RFCOMM )

    uuid = "56e8a14a-80b3-11e5-8bcf-feff819cdc9f"

    port = bluetooth.PORT\_ANY

    # set up the server socket and listen in on the connection

    server\_socket.bind(("",port))

    server\_socket.listen(1)

    bluetooth.advertise\_service( server\_socket, "Bluetooth Server",

                                service\_id = uuid,

                    service\_classes = [ uuid, bluetooth.SERIAL\_PORT\_CLASS ],profiles = [ bluetooth.SERIAL\_PORT\_PROFILE ]

                    )

    print("Currently looking for connections...")

    # blocking call, waits until a client connects to the socket

    client\_socket,address = server\_socket.accept()

    print("Accepted connection from {}".format(address))

    # loop to receive communication from client

    while 1:

        data = client\_socket.recv(1024)

        print(data)

        # if client sends an exit signal, then break this loop

        if (data == b'\x00'):

            break

        # if client unexpectedly disconnects, also break

        if not data:

            break

        if data == b'\x01':

            stop()

        if data == b'\x02':

            moveForward(moveSpeed)

        if data == b'\x03':

            moveBackward(moveSpeed)

        if data == b'\x04':

            turnLeft(moveSpeed)

        if data == b'\x05':

            turnRight(moveSpeed)

        if data == b'\x06':

            rotateLeftInPlace(moveSpeed)

        if data == b'\x07':

            rotateRightInPlace(moveSpeed)

    # close sockets

    print("Client disconnected. Now quitting...")

    client\_socket.close()

    server\_socket.close()

elif control\_mode == "2":

    print("web server mode")

    #use the webserver

    # setup the server

    http\_server = HTTPServer((host\_name, host\_port), MyServer)

    print("server open")

    try:

        #handle request until the user exit the web

        http\_server.serve\_forever()

    except KeyboardInterrupt:

        print("server exception")

        http\_server.server\_close()

else:

    print("default mode")

    dist = 0

    speedFactor = 0.5

    UPPER = 1000

    LOWER = 500

    delay = 0.03

    #pid control variables

#     KP = 0.245

#     KD = 0.018

    KP=0.248

    KD = 0.02

    error2 = 0.8

    prev\_error = 0.0

    min\_speed = 0

    max\_speed = 1

    direction = 0

    try:

        while True:

            lLevel = ReadChannel(lSensor) in range(LOWER,UPPER)

            mLevel = ReadChannel(mSensor) in range(LOWER,UPPER)

            rLevel = ReadChannel(rSensor) in range(LOWER,UPPER)

            print("l: " + str(ReadChannel(lSensor)), end = " ")

            print("m: " + str(ReadChannel(mSensor)), end = " ")

            print("r: " + str(ReadChannel(rSensor)))

            if mLevel:

                #as long as the middle level is detecting, go straight

                dist = 0

                moveForward(moveSpeed)

                direction = 0

            elif mLevel and not lLevel and not rLevel:

                dist = 0

                moveForward(moveSpeed)

                print("forward")

                prev\_error = 0.0

                direction = 0

            elif mLevel and lLevel and rLevel:

                dist = 0

                moveForward(moveSpeed)

                print("90cross")

                prev\_error = 0.0

                direction = 0

            elif (mLevel and lLevel and not rLevel) or lLevel or direction == 1:

                dist = 0

                #turnLeft(0.3)

                #use the pid control to turn left

                LEFT\_MOTOR.throttle = -max(min(moveSpeed - (error2 \* KP + prev\_error \* KD),max\_speed),min\_speed)

                RIGHT\_MOTOR.throttle = max(min(moveSpeed + (error2 \* KP + prev\_error \* KD),max\_speed),min\_speed)

                prev\_error = (error2 \* KP + prev\_error \* KD)/KP

                print("left, l: " + str(LEFT\_MOTOR.throttle)+" r: "+str(RIGHT\_MOTOR.throttle))

                direction = 1

            elif (mLevel and not lLevel and rLevel) or rLevel or direction == 2:

                dist = 0

                #turnRight(0.3)

                #use pid control to turn right

                LEFT\_MOTOR.throttle = max(min(moveSpeed + (error2 \* KP + prev\_error \* KD),max\_speed),min\_speed)

                RIGHT\_MOTOR.throttle = -max(min(moveSpeed - (error2 \* KP + prev\_error \* KD),max\_speed),min\_speed)

                prev\_error = (error2 \* KP + prev\_error \* KD)/KP

                print("right, l: " + str(LEFT\_MOTOR.throttle)+" r: "+str(RIGHT\_MOTOR.throttle))

                direction = 2

            else:

                print("forward no reading")

                direction = 0

                moveForward(moveSpeed)

                dist += moveSpeed \* 2 \* math.pi \* 3

                if dist >= 33: # 27 for 3cm

                    print (dist)

                    stop()

                    break

            time.sleep(delay)

    except KeyboardInterrupt:

        stop()

**Appendix B.2 cameraControl.py(control the camera)**

import RPi.GPIO as GPIO

import os

import time

from threading import Thread

from http.server import BaseHTTPRequestHandler, HTTPServer

import digitalio

import board

from picamera import PiCamera

from PIL import Image, ImageDraw

import adafruit\_rgb\_display.st7735 as st7735

#import motorFunctions

#region setup

#############################################

# pins variables setup

#############################################

#setup http server

host\_name = '137.82.226.228'    #Raspberry Pi IP address

host\_port = 8000

#setup LCD

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

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

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

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

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

BAUDRATE = 24000000

# Setup SPI bus using hardware SPI:

spi = board.SPI()

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)

#setup camera

camera =PiCamera()

camera.resolution = (640, 480)

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

height = disp.height

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

camera.framerate = 30

# Get drawing object to draw on image.

draw = ImageDraw.Draw(image)

#endregion

#region classes

#############################################

# class set up

#############################################

class MyCamera():

    #class for camera multi-threading

    #so that the camera can take pictures constantly while the user is controlling the robot on the webpage

    def \_\_init\_\_(self):

        self.running = True

    def changeState(self):

        #change the state of running

        #if running == true, the while loop in the run() function will take pictures

        #if running == false, the while loop will not take pictures simply idle

        self.running = not self.running

    def run(self):

        while True:

            if not self.running:

                camera.stop\_preview()

                continue

            #capture the picture and save it to the desktop

            else:

                camera.start\_preview()

            camera.capture('/home/pi/Desktop/picture1.bmp')

            image = Image.open("picture1.bmp")

            # 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)

#set up camera and its thread for control in the MyServer class

c = MyCamera()

cThread = Thread(target = c.run)

class MyServer(BaseHTTPRequestHandler):

    #this is the class that sets up the website and handle requests sent by the user

    def do\_HEAD(self):

        self.send\_response(200)

        self.send\_header('Content-type', 'text/html')

        self.end\_headers()

    def \_redirect(self, path):

        self.send\_response(303)

        self.send\_header('Content-type', 'text/html')

        self.send\_header('Location', path)

        self.end\_headers()

    def do\_GET(self):

        #generate the html web with six buttons

        # first four buttons are used to control the robot to move forward, backward, left and right

        #fifth button is used to stop the robot

        #last button is used to control the camera to start/stop taking pictures

        html = '''

            <html>

            <body style="width:960px; margin: 20px auto;" >

            <h1>CPEN291 Project 1 Raspberry Pi web robot control</h1>

            <p>Current temperature is {}</p>

            <form action="/" method="POST">

                Camera control: <br />

                <input type="submit" name="submit" value="Camera">

            </form>

            <img src="picture1.bmp" width="640" height="480">

            </body>

            </html>

        '''

        #the following code is used to show the current CPU temperature of raspberry pi

        #and make sure that the website can be properly shown

        temp = os.popen("/opt/vc/bin/vcgencmd measure\_temp").read()

        self.do\_HEAD()

        self.wfile.write(html.format(temp[5:]).encode("utf-8"))

    def do\_POST(self):

        #we get the request from the user

        #call the specific function in motor functions to handle the request

        content\_length = int(self.headers['Content-Length'])    # Get the size of data

        post\_data = self.rfile.read(content\_length).decode("utf-8")   # Get the data

        post\_data = post\_data.split("=")[1]    # Only keep the value

        print(post\_data)

        if post\_data=="Camera":

            #when the user press on the camera button,

            #change the state of camera state to start/stop taking pictures

            if not cThread.is\_alive():

                cThread.start()

            else:

                c.changeState()

        self.\_redirect('/')    # finished handling request, redirect back to the root url

#endregion

#############################################

# main function

#############################################

# setup the server

http\_server = HTTPServer((host\_name, host\_port), MyServer)

print("server open")

try:

    #handle request until the user exit the web

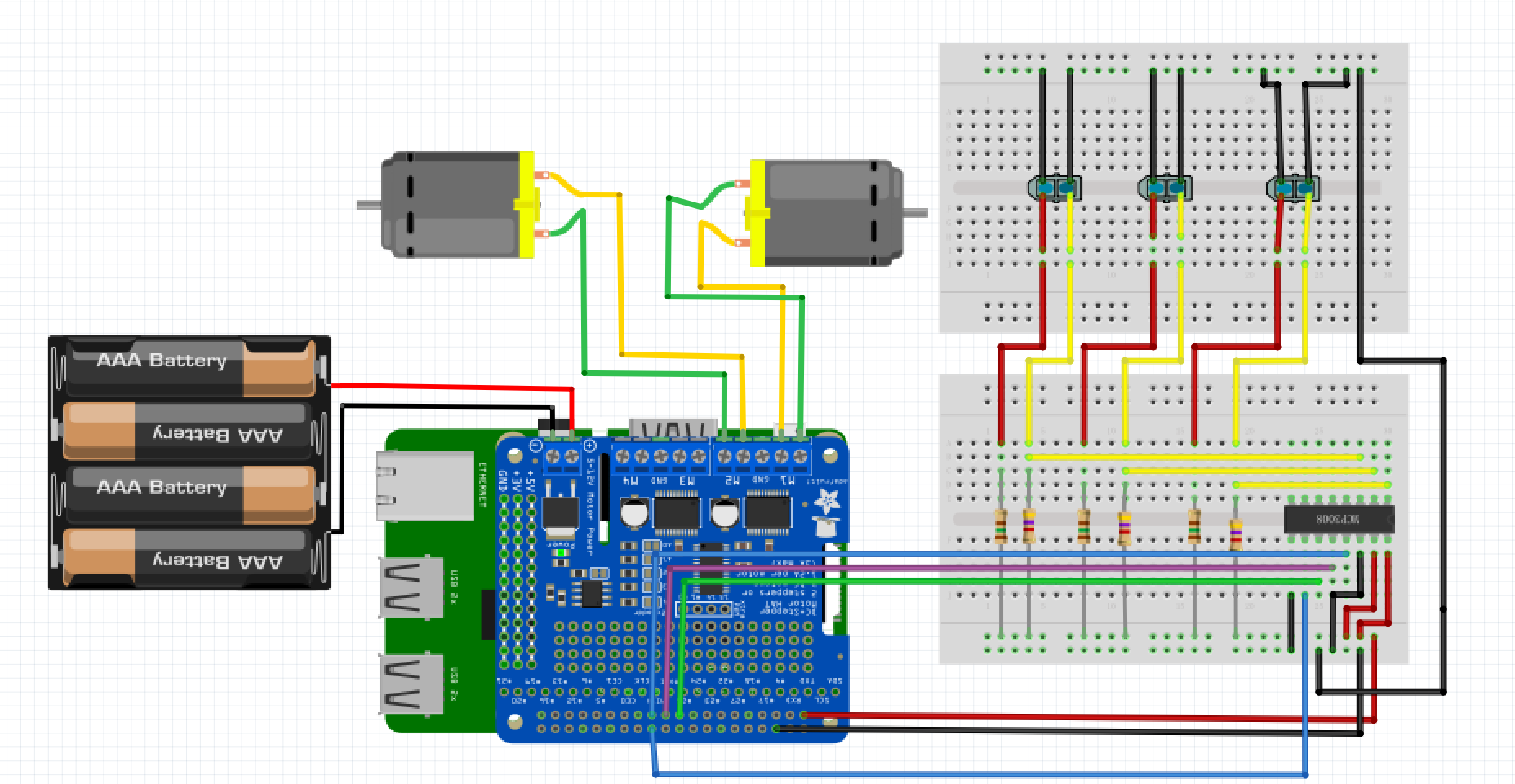
    http\_server.serve\_forever()

except KeyboardInterrupt:

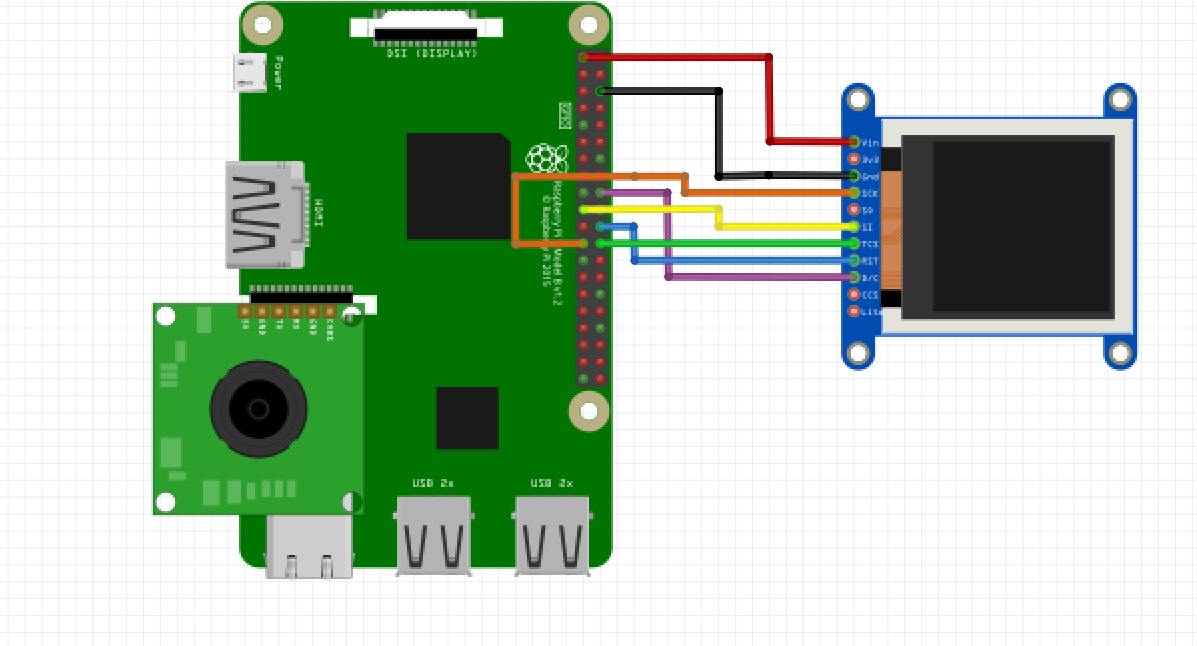
    print("server exception")

    http\_server.server\_close()

**Appendix C - Fritzing**



First Raspberry Pi with motor and optical sensors



Second Pi with Pi Camera and LCD display

**Appendix D - GitHub**

https://github.com/CPEN-291/P1\_A\_G10

**Appendix E – Complete Component list**

2Raspberry Pi 4

1Pi Camera

1DFRobot 2WD Mobile Platform

1Adafruit 1.44’’ color TFT LCD

1MCP3008 ADC converter

3TCRT5000 reflective optical sensors

5AA batteries

1Power bank

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

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

We will first assign different job to each member, then we use github to update and share our code. Also, we have our Facebook group chat to communicate and to solve the problems together.

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 were thinking to use MCP3008 for our reflective light sensor, because it can give us a more accurate result other than 0 and 1. Also, we keep using the LCD to display image. This time, we added more feature to it. We try to project the camera screen on the LCD. To implement this, we need to save the image we took from camera first and let the LCD read the image. If the process is fast enough, the LCD will show the screen of camera with some delay. Also, we have were thinking of using website to control our robot, such as moving forward, backward, stop etc.

1. **Constraint identification**: Explain the constraints that you must consider in the design of the additional functionalities.

To get the LCD working, we have to download some addition packages from the website to set it up properly, and this is time consuming. Also, when we try to create our own website to control the robot, we have to use HTML to create buttons and send request to the server. Again, this requires lots of searching and reading.

1. **Solution generation**: Explain at least two possible alternative additional features that your group rejected due to technical reasons and explain why.

We were thinking of using joystick to control the robot, but somehow, we used the website instead since it’s fancier. Also, we reject the ultrasonic sensor since it’s not helpful in terms of our design.

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

We always test the code we wrote and see if it’s actually working before, we added it to the robot. Also, we always test how the code will affect the movement of the robot. If something goes wrong, we will always try to communicate and figure out a solution.

**Appendix G – Other**

**Improve version for second demo(no need to adjust values manually):**

import sys

import os

import time

import math

import spidev

import bluetooth

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

from adafruit\_motorkit import MotorKit

from http.server import BaseHTTPRequestHandler, HTTPServer

#############################################

# Motor definitions and functions

#############################################

# setup the motor kit

kit = MotorKit()

# define left and right motors

LEFT\_MOTOR = kit.motor1

RIGHT\_MOTOR = kit.motor2

# move motors forward at defined speed

def moveForward(speed):

    LEFT\_MOTOR.throttle = speed

    RIGHT\_MOTOR.throttle = speed

def nudge(speed):

    moveForward(speed)

    time.sleep(0.1)

# rotate the robot left in place

def rotateLeftInPlace(speed):

    RIGHT\_MOTOR.throttle = speed

    LEFT\_MOTOR.throttle = -speed

    # need to call stop() to stop rotating

# rotate the robot right in place

def rotateRightInPlace(speed):

    LEFT\_MOTOR.throttle = speed

    RIGHT\_MOTOR.throttle = -speed

    # need to call stop() to stop rotating

# move motors backward at defined speed

def moveBackward(speed):

    LEFT\_MOTOR.throttle = -1 \* speed

    RIGHT\_MOTOR.throttle = -1 \* speed

# stop the robot

def stop():

    LEFT\_MOTOR.throttle = 0

    RIGHT\_MOTOR.throttle = 0

def turnLeft(speed):

    rotateLeftInPlace(speed)

    time.sleep(0.5)

    #stop()

def turnRight(speed):

    rotateRightInPlace(speed)

    time.sleep(0.5)

    #stop()

def turn180():

    turnLeft()

    turnLeft()

#############################################

# Sensor/spidev definitions and functions

#############################################

# setting up spi

spi = spidev.SpiDev()

spi.open(0,0)

spi.max\_speed\_hz=1000000

# Define sensor channels

lSensor = 0

mSensor = 1

rSensor = 2

# Function to read SPI data from MCP3008 chip

# Channel must be an integer 0-7

def ReadChannel(channel):

  adc = spi.xfer2([1,(8+channel)<<4,0])

  data = ((adc[1]&3) << 8) + adc[2]

  return data

#############################################

# Web server definitions and functions

#############################################

host\_name = '137.82.226.231'    # Change this to Raspberry Pi IP address

host\_port = 8000

class MyServer(BaseHTTPRequestHandler):

    def do\_HEAD(self):

        self.send\_response(200)

        self.send\_header('Content-type', 'text/html')

        self.end\_headers()

    def \_redirect(self, path):

        self.send\_response(303)

        self.send\_header('Content-type', 'text/html')

        self.send\_header('Location', path)

        self.end\_headers()

    def do\_GET(self):

        # generate the html web with four buttons to control the robot to move forward, backward, left and right

        html = '''

            <html>

            <body style="width:960px; margin: 20px auto;">

            <h1>CPEN291 Project 1 Raspberry Pi web robot control</h1>

            <p>Current temperature is {}</p>

            <form action="/" method="POST">

                Robot control: <br />

                <input type="submit" name="submit" value="Forward"> <br />

                <input type="submit" name="submit" value="Left">

                <input type="submit" name="submit" value="Right"><br />

                <input type="submit" name="submit" value="Backward"><br />

                <input type="submit" name="submit" value="Stop">

            </form>

            </body>

            </html>

        '''

        temp = os.popen("/opt/vc/bin/vcgencmd measure\_temp").read()

        self.do\_HEAD()

        self.wfile.write(html.format(temp[5:]).encode("utf-8"))

    def do\_POST(self):

        # we get the request from the user

        # call the specific function in motor functions to handle the request

        content\_length = int(self.headers['Content-Length'])        # Get the size of data

        post\_data = self.rfile.read(content\_length).decode("utf-8") # Get the data

        post\_data = post\_data.split("=")[1]                         # Only keep the value

        moveSpeed = 0.35

        if post\_data == 'Forward':

            moveForward(moveSpeed)

            print("car is moving forward")

        elif post\_data == 'Backward':

            moveBackward(moveSpeed)

            print("car is moving backward")

        if post\_data =='Left':

            turnLeft(moveSpeed)

            print("car is rotating left")

        elif post\_data == 'Right':

            turnRight(moveSpeed)

            print("car is rotating right")

        if post\_data == 'Stop':

            stop()

            print("stopped")

        self.\_redirect('/') # finished handling request, redirect back to the root url

#############################################

# Main loop function

#############################################

# this is for reading parameters given by the command line execution

control\_mode = sys.argv[1]

#default move speed for all three modes

moveSpeed = 0.35

if control\_mode == "1":

    print("bluetooth mode")

    # bluetooth control

    # get server socket and set UUID and port number

    server\_socket=bluetooth.BluetoothSocket( bluetooth.RFCOMM )

    uuid = "56e8a14a-80b3-11e5-8bcf-feff819cdc9f"

    port = bluetooth.PORT\_ANY

    # set up the server socket and listen in on the connection

    server\_socket.bind(("",port))

    server\_socket.listen(1)

    bluetooth.advertise\_service( server\_socket, "Bluetooth Server",

                                service\_id = uuid,

                    service\_classes = [ uuid, bluetooth.SERIAL\_PORT\_CLASS ],profiles = [ bluetooth.SERIAL\_PORT\_PROFILE ]

                    )

    print("Currently looking for connections...")

    # blocking call, waits until a client connects to the socket

    client\_socket,address = server\_socket.accept()

    print("Accepted connection from {}".format(address))

    # loop to receive communication from client

    while 1:

        data = client\_socket.recv(1024)

        print(data)

        # if client sends an exit signal, then break this loop

        if (data == b'\x00'):

            break

        # if client unexpectedly disconnects, also break

        if not data:

            break

        if data == b'\x01':

            stop()

        if data == b'\x02':

            moveForward(moveSpeed)

        if data == b'\x03':

            moveBackward(moveSpeed)

        if data == b'\x04':

            turnLeft(moveSpeed)

        if data == b'\x05':

            turnRight(moveSpeed)

        if data == b'\x06':

            rotateLeftInPlace(moveSpeed)

        if data == b'\x07':

            rotateRightInPlace(moveSpeed)

    # close sockets

    print("Client disconnected. Now quitting...")

    client\_socket.close()

    server\_socket.close()

elif control\_mode == "2":

    print("web server mode")

    #use the webserver

    # setup the server

    http\_server = HTTPServer((host\_name, host\_port), MyServer)

    print("server open")

    try:

        #handle request until the user exit the web

        http\_server.serve\_forever()

    except KeyboardInterrupt:

        print("server exception")

        http\_server.server\_close()

else:

    print("default mode")

    dist = 0

    speedFactor = 0.5

    UPPER = 1000

    LOWER = 500

    delay = 0.02

    upper\_values = []

lower\_values = []

    #automatically detect the current surface and track reflection rate, record them as UPPER and LOWER bounds

    #so that the robot can work on any surfaces and any tracks

    for i in range(10):

        upper\_values.append(ReadChannel(mSensor))

        lower\_values.append(max(ReadChannel(lSensor),ReadChannel(rSensor)))

    max\_upper\_value = 0

    min\_upper\_value = 2000

    for value in upper\_values:

        if value>max\_upper\_value:

            max\_upper\_value = value

        if value < min\_upper\_value:

            min\_upper\_value = value

    max\_lower\_value = 0

    min\_lower\_value = 2000

    for value in lower\_values:

        if value>max\_lower\_value:

            max\_lower\_value = value

        if value < min\_lower\_value:

            min\_lower\_value = value

    UPPER = int(int(max\_upper\_value / 100) \* 100) + 200

    LOWER = int(int(min(max\_lower\_value, min\_upper\_value)/ 100) \* 100) + 150

    print(str(UPPER) + " " + str(LOWER))

    #pid control variables

#     KP = 0.245

#     KD = 0.018

    KP=0.248

    KD = 0.02

    error2 = 0.8

    prev\_error = 0.0

    min\_speed = 0.2

    max\_speed = 0.8

    direction = 0

    try:

        while True:

            lLevel = ReadChannel(lSensor) in range(LOWER,UPPER)

            mLevel = ReadChannel(mSensor) in range(LOWER,UPPER)

            rLevel = ReadChannel(rSensor) in range(LOWER,UPPER)

            print("l: " + str(ReadChannel(lSensor)), end = " ")

            print("m: " + str(ReadChannel(mSensor)), end = " ")

            print("r: " + str(ReadChannel(rSensor)))

            if mLevel:

                #as long as the middle level is detecting, go straight

                dist = 0

                moveForward(moveSpeed)

                direction = 0

            elif mLevel and not lLevel and not rLevel:

                dist = 0

                moveForward(moveSpeed)

                print("forward")

                prev\_error = 0.0

                direction = 0

            elif mLevel and lLevel and rLevel:

                dist = 0

                moveForward(moveSpeed)

                print("90cross")

                prev\_error = 0.0

                direction = 0

            elif (mLevel and lLevel and not rLevel) or lLevel or direction == 1:

                dist = 0

                #turnLeft(0.3)

                #use the pid control to turn left

                LEFT\_MOTOR.throttle = -max(min(moveSpeed - (error2 \* KP + prev\_error \* KD),max\_speed),min\_speed)

                RIGHT\_MOTOR.throttle = max(min(moveSpeed + (error2 \* KP + prev\_error \* KD),max\_speed),min\_speed)

                prev\_error = (error2 \* KP + prev\_error \* KD)/KP

                print("left, l: " + str(LEFT\_MOTOR.throttle)+" r: "+str(RIGHT\_MOTOR.throttle))

                direction = 1

            elif (mLevel and not lLevel and rLevel) or rLevel or direction == 2:

                dist = 0

                #turnRight(0.3)

                #use pid control to turn right

                LEFT\_MOTOR.throttle = max(min(moveSpeed + (error2 \* KP + prev\_error \* KD),max\_speed),min\_speed)

                RIGHT\_MOTOR.throttle = -max(min(moveSpeed - (error2 \* KP + prev\_error \* KD),max\_speed),min\_speed)

                prev\_error = (error2 \* KP + prev\_error \* KD)/KP

                print("right, l: " + str(LEFT\_MOTOR.throttle)+" r: "+str(RIGHT\_MOTOR.throttle))

                direction = 2

            else:

                print("forward no reading")

                direction = 0

                moveForward(moveSpeed)

                dist += moveSpeed \* 2 \* math.pi \* 3

                if dist >= 39: # 39 for 3cm

                    print (dist)

                    stop()

                    break

            time.sleep(delay)

    except KeyboardInterrupt:

        stop()