American University of Sharjah

College of Engineering

Department of Computer Engineering

COE 410

Project: Smart Toll Gate

Date of Submission: 15 July 2023

Group Members:

Hassan Alhilo	b00085602
Muhammad Sadaqat	b00090340
Dwayne Fonseca	b00088027

Table of Contents

Hardware Block Diagram

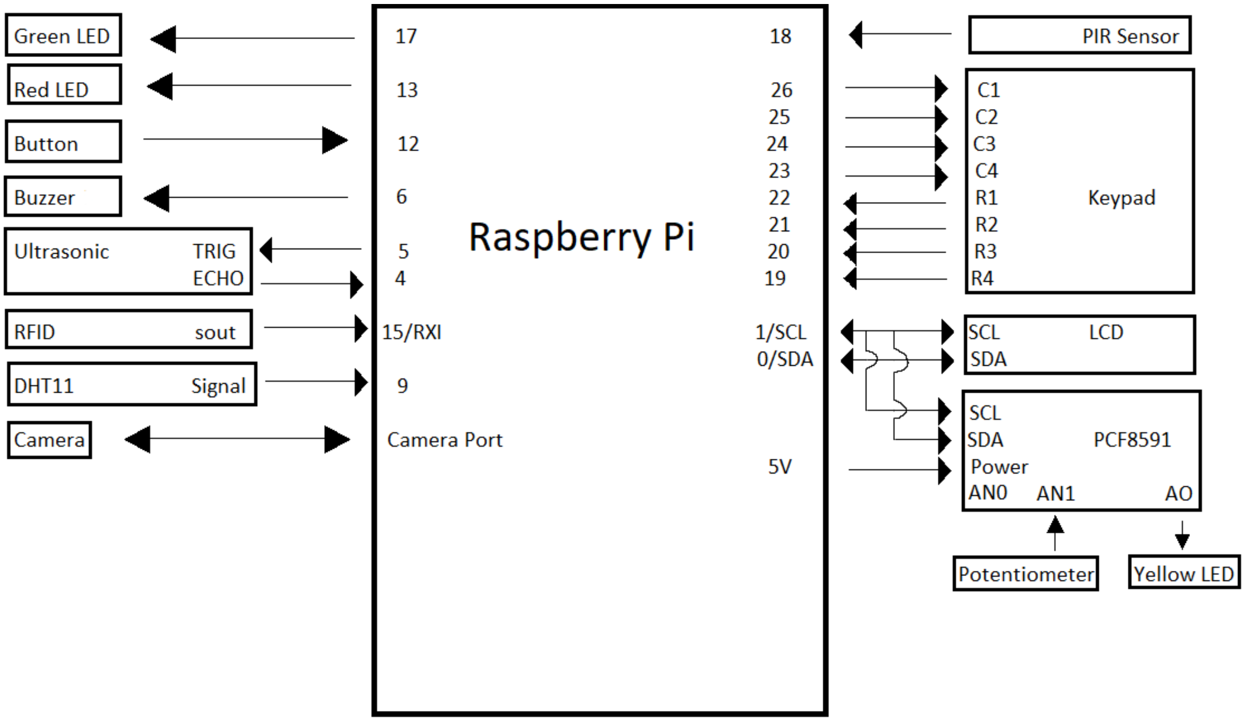


Figure 1: Hardware diagram of Smart Toll Gate

Software Flowchart

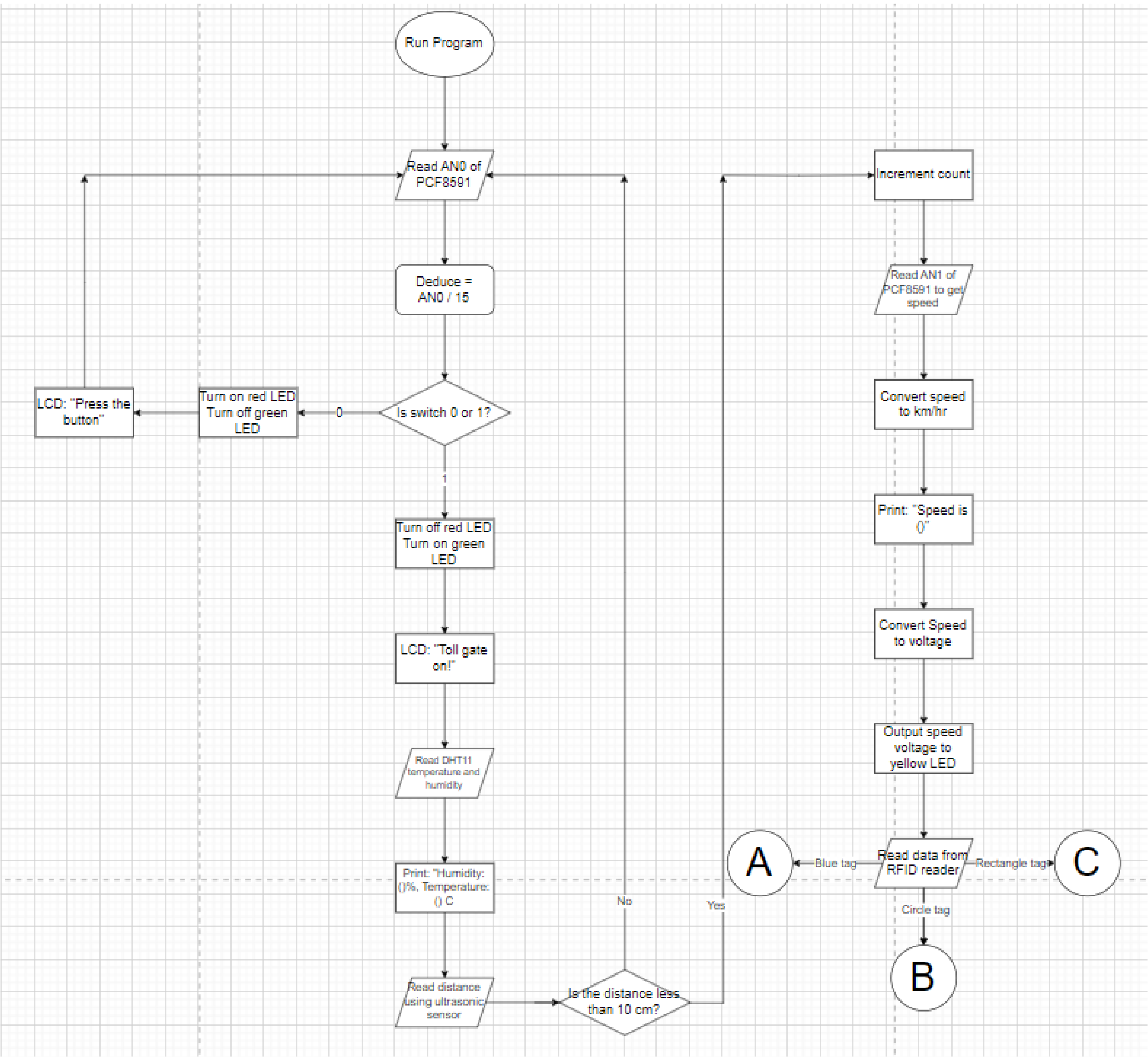

Run Program
Read AN0 of PCF8591
Deduce = AN0 / 15
Is switch 0 or 1?
0
Turn on red LED Turn off green LED
LCD: "Press the button"
1
Turn off red LED Turn on green LED
LCD: "Toll gate on!"
Read DHT11 temperature and humidity
Print: "Humidity: ()%, Temperature: () C
Read distance using ultrasonic sensor
Is the distance less than 10 cm?
No
Yes
Increment count
Read AN1 of PCF8591 to get speed
Convert speed to km/hr
Print: "Speed is ()"
Convert Speed to voltage
Output speed voltage to yellow LED
Read data from RFID reader
Blue tag
A
Rectangle tag
C
Circle tag
B

Figure 2: Flowchart for first half of main program loop

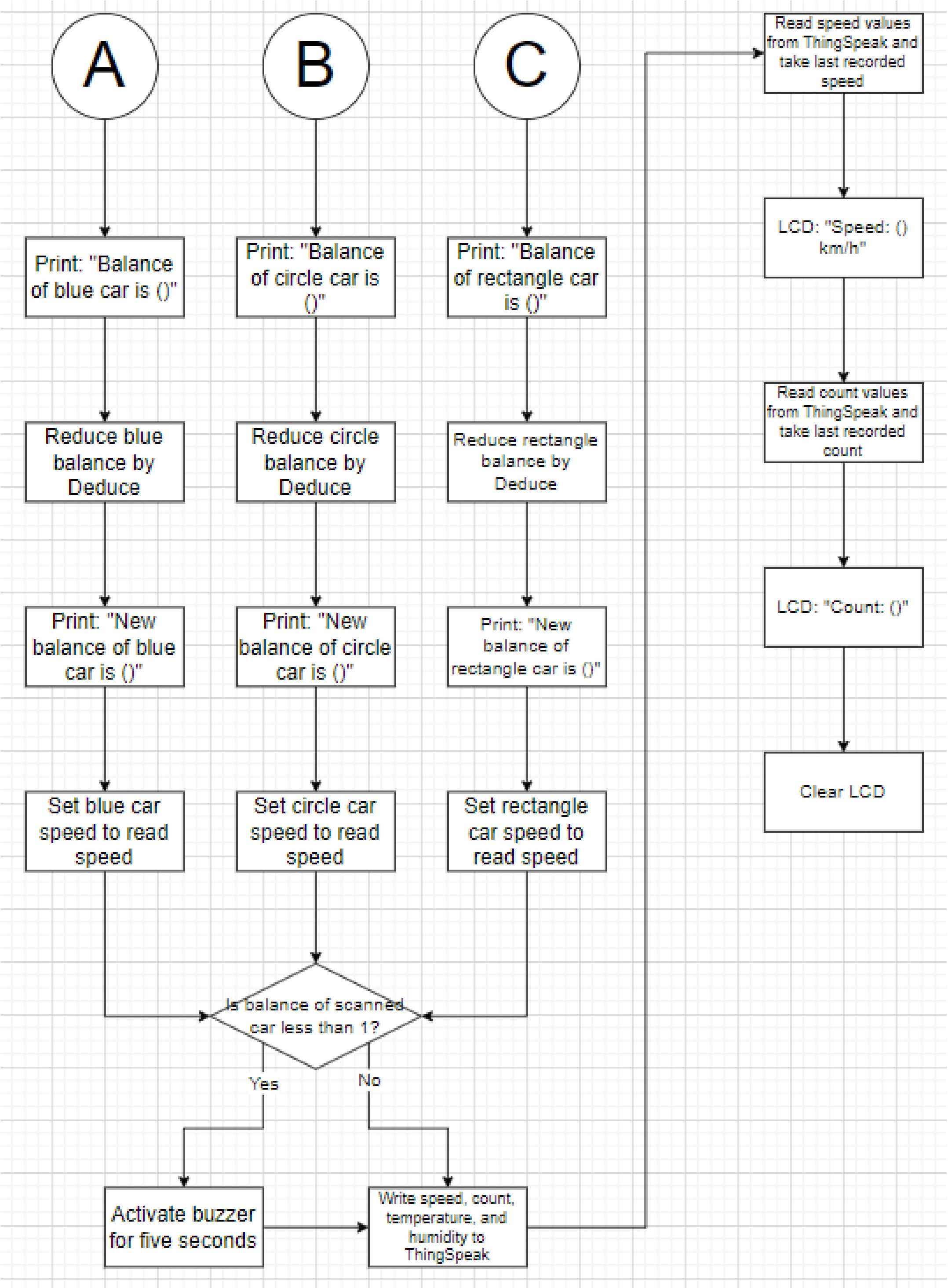

A
B
C
Print: "Balance of blue car is ()"
Print: "Balance of circle car is ()"
Print: "Balance of rectangle car is ()"
Reduce blue balance by Deduce
Reduce circle balance by Deduce
Reduce rectangle balance by Deduce
Print: "New balance of blue car is ()"
Print: "New balance of circle car is ()"
Print: "New balance of rectangle car is ()"
Set blue car speed to read speed
Set circle car speed to read speed
Set rectangle car speed to read speed
Is balance of scanned car less than 1?
Yes
No
Activate buzzer for five seconds
Write speed, count, temperature, and humidity to ThingSpeak
Read speed values from ThingSpeak and take last recorded speed
LCD: "Speed: () km/h"
Read count values from ThingSpeak and take last recorded count
LCD: "Count: ()"
Clear LCD

Figure 3: Flowchart of second half of main program loop

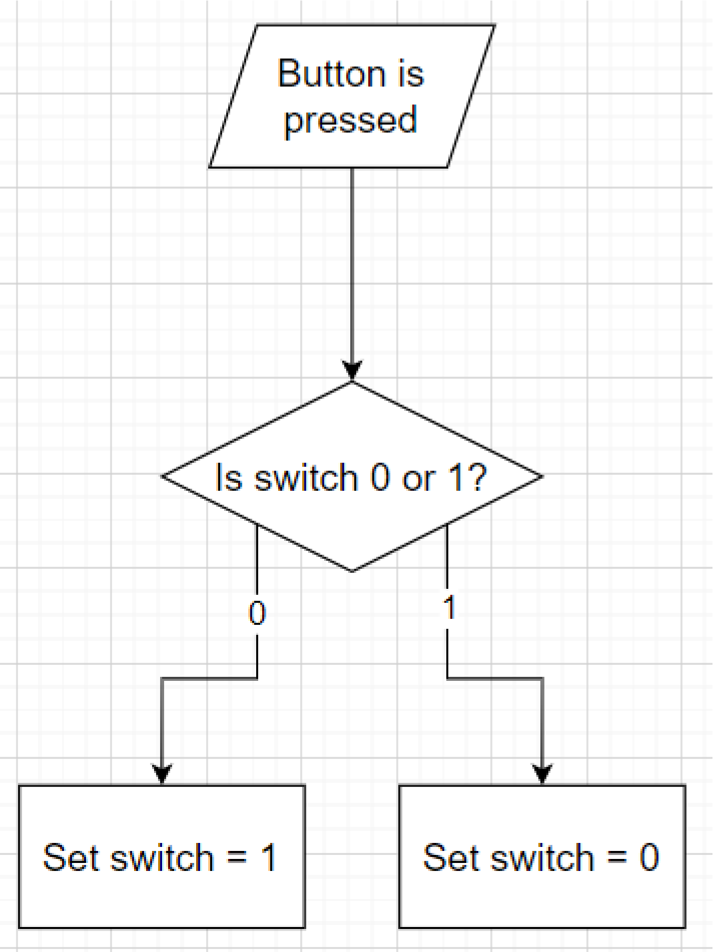


Figure 4: Flowchart for first interrupt when the button is pressed

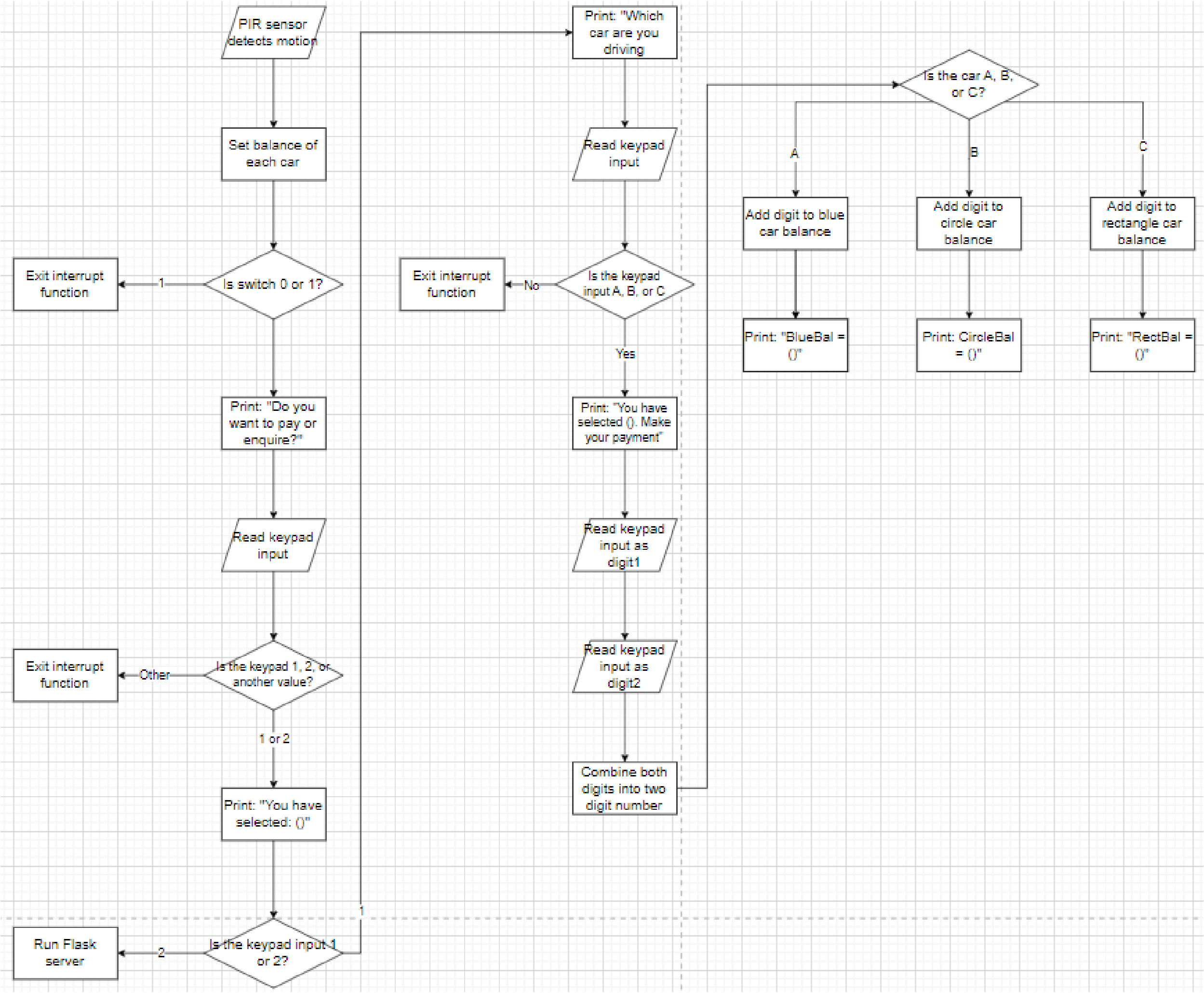
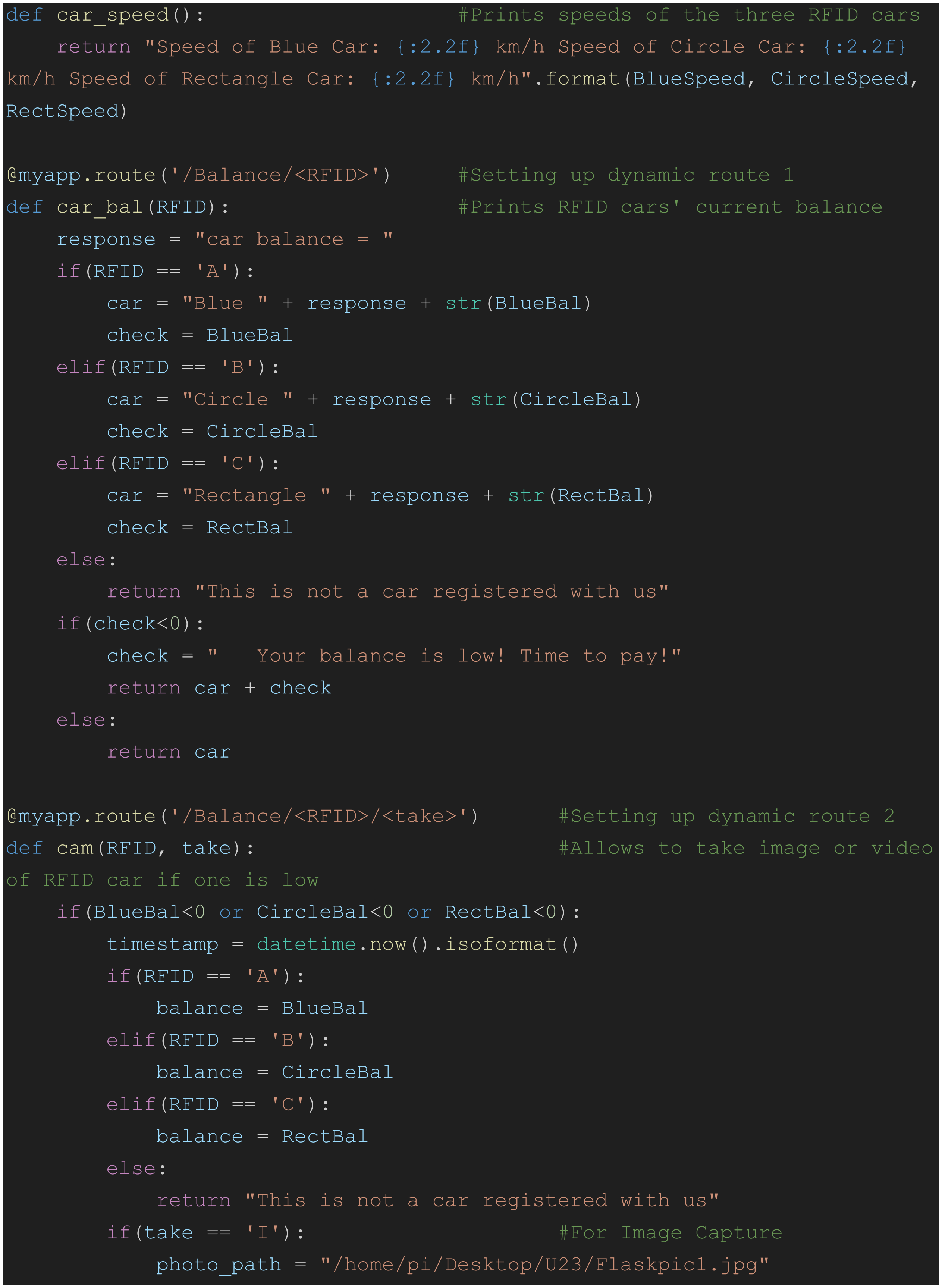

PIR sensor detects motion
Set balance of each car
Is switch 0 or 1?
1
Exit interrupt function
Print: "Do you want to pay or enquire?"
Read keypad input
Is the keypad 1, 2, or another value?
Other
Exit interrupt function
1 or 2
Print: "You have selected: ()"
Is the keypad input 1 or 2?
2
Run Flask server
1
Print: "Which car are you driving
Read keypad input
Is the keypad input A, B, or C
No
Exit interrupt function
Yes
Print: "You have selected (). Make your payment"
Read keypad input as digit1
Read keypad input as digit2
Combine both digits into two digit number
Is the car A, B, or C?
A
B
C
Add digit to blue car balance
Add digit to circle car balance
Add digit to rectangle car balance
Print: "BlueBal = ()"
Print: CircleBal = ()"
Print: "RectBal = ()"

Figure 5: Flowchart for second interrupt when infrared detects motion

Code with comments

```
import RPi.GPIO as GPIO     # Importing libraries used in program
import time
from datetime import datetime
import LCD1602 as LCD
import keypadfunc as key    # do key.shiftkeypad
import RFIDTest as RFID
import DHT11
import PCF8591 as ADC
import urllib.request
from flask import Flask
from flask import send_file
from picamera import PiCamera
import serial

API_KEY = "FDQ2OAPC0L7M0G17"    # Write API key for ThingSpeak channel
Ch_id = "2216444"               # Channel ID of ThingSpeak channel
field_no = 1
readings = 1
elem_no = 0
values = []
values1 = []
mycamera = PiCamera()           # Setting Raspberry Pi camera
mycamera.resolution = (1280,720)    # Setting resolution of Raspberry Pi
camera

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)          #Set to BCM mode

GREEN = 17             #Setting GPIO pins as variables
YELLOW = 16
RED = 13
Button = 12
Buzz = 6
TRIG = 5
ECHO = 4
IR = 18
Blue = "1400415E76"      #Setting RFID numbers as variables
Circle = "46003BB194"
```

```
Rectangle = "5300C7E99B"
BlueBal = 20              #Setting initial balance for each RFID
CircleBal = 10
RectBal = 0
count = 0                  #Variable to count number of vehicles ie RFID tags, initialized to zero
i = 0
switch = 0                 #Variable to indicate if the toll gate system is on or not
SERIAL_PORT = '/dev/ttyS0'

GPIO.setup(GREEN,GPIO.OUT)   #Setting all three LEDs as outputs
GPIO.setup(YELLOW,GPIO.OUT)
GPIO.setup(RED,GPIO.OUT)
GPIO.setup(Button,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)  #Setting Push Button as input with pull-down resistor
GPIO.setup(Buzz,GPIO.OUT)     #Setting buzzer as output
GPIO.setup(TRIG,GPIO.OUT)     #Setting Trigger as output and Echo as input for Ultrasensonic sensor
GPIO.setup(ECHO,GPIO.IN)
GPIO.setup(IR,GPIO.IN,pull_up_down=GPIO.PUD_UP)   #Setting PIR sensor as input with pull-up resistor
p = GPIO.PWM(Buzz,500)      #Setting PWM signal output to Buzzer with frequency 500
LCD.init(0x27,1)            #Initializing LCD with I2C address 27
ADC.setup(0x48)             #Setting up ADC with I2C address 48



myapp = Flask(__name__)  #Setting flask web server
@myapp.route('/')        #Setting up index page
def index():
    return "Welcome to the Smart Toll Gate!"

@myapp.route('/General_Info')      #Setting up static route 1
def gen_info():
    return "Number of cars that pass through: {}\nTemperature: {:2.2f} C\nHumidity: {} %".format(count, temperature, humidity)

@myapp.route('/Vehicle_Speed')      #Setting up static route 2
```



```
def car_speed():                    #Prints speeds of the three RFID cars
    return "Speed of Blue Car: {:2.2f} km/h Speed of Circle Car: {:2.2f}
km/h Speed of Rectangle Car: {:2.2f} km/h".format(BlueSpeed, CircleSpeed,
RectSpeed)

@myapp.route('/Balance/<RFID>')     #Setting up dynamic route 1
def car_bal(RFID):                  #Prints RFID cars' current balance
    response = "car balance = "
    if(RFID == 'A'):
        car = "Blue " + response + str(BlueBal)
        check = BlueBal
    elif(RFID == 'B'):
        car = "Circle " + response + str(CircleBal)
        check = CircleBal
    elif(RFID == 'C'):
        car = "Rectangle " + response + str(RectBal)
        check = RectBal
    else:
        return "This is not a car registered with us"
    if(check<0):
        check = "   Your balance is low! Time to pay!"
        return car + check
    else:
        return car

@myapp.route('/Balance/<RFID>/<take>')      #Setting up dynamic route 2
def cam(RFID, take):                        #Allows to take image or video
of RFID car if one is low
    if(BlueBal<0 or CircleBal<0 or RectBal<0):
        timestamp = datetime.now().isoformat()
        if(RFID == 'A'):
            balance = BlueBal
        elif(RFID == 'B'):
            balance = CircleBal
        elif(RFID == 'C'):
            balance = RectBal
        else:
            return "This is not a car registered with us"
        if(take == 'I'):                    #For Image Capture
            photo_path = "/home/pi/Desktop/U23/Flaskpic1.jpg"
            mycamera.start_preview()
            mycamera.annotate_text=" Pic taken at time {} with balance:
{}".format(datetime.now().isoformat(), balance)
            time.sleep(10)
            mycamera.capture(photo_path)

            mycamera.stop_preview()
            response = send_file(photo_path,mimetype="image/jpeg")

        elif(take == 'V'):                  #For Video Recording
            mycamera.start_preview()
            mycamera.start_recording('/home/pi/Desktop/U23/video.h264')
            mycamera.annotate_text="Video taken at time {} with balance
{}".format(datetime.now().isoformat(), balance)
            time.sleep(10)
            mycamera.stop_recording()
            mycamera.stop_preview()
            photo_path = "/home/pi/Desktop/U23/video.h264"
            response = send_file(photo_path,mimetype="video/h264")
        return response
    else:
        return "None of the balances are critically low"

def distance():         # Function to measure distance of object using
Ultrasonic sensor
    GPIO.output(TRIG, GPIO.LOW)
    time.sleep(0.000002)
    GPIO.output(TRIG,1)
    time.sleep(0.00001)
    GPIO.output(TRIG, 0)       # Activating trigger of ultrasonic sensor
for 10 microseconds
    while GPIO.input(ECHO) == 0:     # Using dummy variable to wait until
echo signal activates to take t1 time
        a=0
    time1 = time.time()
    while GPIO.input(ECHO) == 1:     # Using dummy variable to wait until
ultrasonic singal is fully recieved to take t2 time
        a=0
    time2= time.time()
    duration = time2-time1
    return duration*1000000/58      # Converting time to distance in
centimeter using ultrasonic formula


def action(self): # Function after interrupt from PIR sensor (Infrared)
    IRed = 1
    global i
    global BlueBal
    global CircleBal
    global RectBal
    if (i == 0):
        BlueBal = 20
        CircleBal = 10
        RectBal = 0
        i = i + 1
    while (IRed and not switch):      #Checks if toll gate system is first
off
        print("Do you want to pay or enqiure (Press 1 for payment, 2 for
enquiry, click anything else to cancel)")
        affirm = key.shiftkeypad()
        time.sleep(0.5)
        if(affirm == 1 or affirm == 2):
            print("You have selected: {}".format(affirm))
        if (affirm == 2 and __name__ == '__main__'):
            myapp.run(host = '0.0.0.0', port = 5040)    #Opens and allows
for flask to function
        elif (affirm == 1):
            print("Which car are you driving? Choose between A for blue, B
for circle, and C for rectangle")
            type = key.keypad()     #Keypad used to type in values
            time.sleep(0.5)
            if(type == 'A' or type == 'B' or type == 'C'):
                print("You have selected: {}. Make your payment (between
01 and 99)".format(type)) # doesnt print right says A is 10
                digit1 = key.shiftkeypad()  #Keypad used to accept numbers
                time.sleep(0.5)
                print("First digit recieved")
                digit2 = key.shiftkeypad()
                time.sleep(0.5)
                print("Second digit recieved")
                digit =str(digit1) + str(digit2)
                if(type == 'A'):
                    s = BlueBal
                    s = s + int(digit)      #Increments keypad-inputted
amount into current balance
                    BlueBal=s
                    print("BlueBal = {}".format(BlueBal))       #Prints
new balance after payment
                    IRed = 0
                elif(type == 'B'):
                    s = CircleBal
                    s = s + int(digit)
                    CircleBal=s
                    print("CircleBal = {}".format(CircleBal))
                    IRed = 0
                else:
                    s = RectBal
                    s = s + int(digit)
                    RectBal=s
                    print("RectBal = {}".format(RectBal))
        else:
            break       #If invalid value pressed, breaks out loop

def validate_rfid(code):    #validating RFID data from reader
          s = code.decode("ascii")
          if (len(s) == 12) and (s[0] == "\n") and (s[11] == "\r"):
              return s[1:11]
          else:
              return False

ser = serial.Serial(baudrate = 2400,  bytesize = serial.EIGHTBITS,  parity
= serial.PARITY_NONE,  port = SERIAL_PORT, stopbits = serial.STOPBITS_ONE,
timeout = 1)

def codenum():
    while switch:
        ser.flushInput()
        ser.flushOutput()
        data=ser.read(12)
        code = validate_rfid(data)
        if code:
            print ("RFID tag: {}".format(code))
            return code
        time.sleep(1)

def change(self):      #Function after interrupt from PushButton that
changes the state of the gate system
    global switch
    if (switch == 0):
        switch = 1
    else:
        switch = 0

def read_speed():     #Reads from channel 1 (variable potentiometer) and
returns digital values to modify speed of car
    ADC1_units= ADC.read(1)
    ADC1_volts=(ADC1_units*3.3)/256
    time.sleep(0.5)
    return (ADC1_units)

def read_rate():      #Reads from channel 0 (potentiometer with a fixed
value) and returns digital values to modify rate of deduction after
passing toll gate
    ADC0_units= ADC.read(0)
    ADC0_volts=(ADC0_units*3.3)/256
    time.sleep(0.5)
    return (ADC0_units)

GPIO.add_event_detect(IR,GPIO.FALLING,callback=action,bouncetime=2000)
#Interrupt without polling for PIR sensor with a bouncetime of 2 seconds
GPIO.add_event_detect(Button, GPIO.FALLING, callback=change,
bouncetime=2000) #Interrupt without polling for Push Button with a
bouncetime of 2 seconds

while True:
    code = 1
    Bal = 1
    Deduce = int(read_rate() / 15)      #Determing the value of deduction
for toll gate
```

```
    if not switch:                  #When the toll gate system is off
        GPIO.output(RED, True)      #Red LED is on
        GPIO.output(GREEN, False)   #Green LED is off
        LCD.write(0, 0, "Press the button")

    elif (switch == 1):      #When the toll gate system is turned on
        p.start(0)           #Setting initial Duty Cycle to zero
        GPIO.output(RED, False)     #Red LED is turned off
        GPIO.output(GREEN, True)    #Green LED is turned on
        LCD.write(0,0,"                ")    #LCD clears
        LCD.write(0,0, "Toll Gate on!")
        time.sleep(1)

        result = DHT11.readDht11(27)      #Storing read data from DHT11
(Temperature, Humidity Sensor) to a variable
        if result:
            humidity, temperature = result
            print ("Humidity: {}%,  Temperature: {} C".format(humidity,
temperature))   #Printing Humidity and Temperature values to terminal
            time.sleep(1)
        while(distance() < 10):     #Allows into loop only if distance of
vehicle is less than 10 cm ie car approaching toll gate
            p.ChangeDutyCycle(0)    #Change Duty Cycle of PWM signal of
buzzer to zero
            if (count == 0):
                BlueSpeed = 0
                CircleSpeed = 0
                RectSpeed = 0
            if(code == 1):
                code = codenum()    #Stores read data from RFID reader to
a variable
                print("code = {}".format(code))
            count = count+1         #Incremenets count of cars
            speed = 180*read_speed()/255    #Sets speed value from range
of 0 to 255 to 0 to 180 km/hr
            print("Speed is {}".format(speed))
            LEDvolt = 3.3*speed*256/180     #Converts speed to voltage
value with range 0 to 3.3 V and stores to variable
```

```
            ADC.write(LEDvolt)              #Writes this voltage value to ADC which is connected to Yellow LED and affect its brightness based on the voltage value from a range of 0 to 3.3 V
            if(code == Blue):               #Checks for specific RFID
                print("Balance of blue car is {}".format(BlueBal))
                BlueBal-=Deduce
                print("New balance of blue car is {}".format(BlueBal))
                Bal = BlueBal
                BlueSpeed = speed
                print("Speed of blue is {:2.2f} km/h".format(BlueSpeed))
            elif (code == Circle):
                print("Balance of Circle car is {}".format(CircleBal))
                CircleBal-=Deduce
                print("New balance of Circle car is {}".format(CircleBal))
                Bal = CircleBal
                CircleSpeed = speed
                print("Speed of circle is {:2.2f} km/h".format(CircleSpeed))
            elif (code == Rectangle):
                print("Balance of Rectangle car is {}".format(RectBal))
                RectBal-=Deduce
                print("New balance of Rectangle car is {}".format(RectBal))
                Bal = RectBal
                RectSpeed = speed
                print("Speed of rectangle is {:2.2f} km/h".format(RectSpeed))
            if(Bal<1):      #If balance of any one car is zero or below, the buzzer is swicthed on
                p.ChangeDutyCycle(1)    #Duty Cycle of PWM signal changed to full (100%) to turn on Buzzer
                time.sleep(5)           #Buzzer rings for 5 seconds
                p.ChangeDutyCycle(0)    #Duty Cycle set back to zero, setting off the buzzer
            x = urllib.request.urlopen("https://api.thingspeak.com/update?api_key={}&field1={}&field2={}&field3={}&field4={}".format(API_KEY, speed, count, temperature, humidity)) #Reads (downloads) RPi values for speed, count of cars, temperature, and humidity to ThingSpeak
            y = urllib.request.urlopen("https://api.thingspeak.com/channels/{}/fields/{}.csv?results={}".format(Ch_id, field_no, readings)) #Writes (uploads) speed value from ThingSpeak
            data=y.read().decode('ascii')
            data=",".join(data.split("\n"))
            for i in range(5, readings*3+3, 3):
                values.append(data.split(",")[i])
            element=float(values[-1])       #Choosing most reent speed value
            LCD.write(0,1, "Speed:{:2.2f}km/h".format(element)) #Prints speed value on LCD
            time.sleep(3)   #Allows LCD to display value for 3 seconds
            LCD.write(0,1,"                ")   #Clears the first line of the LCD
            y = urllib.request.urlopen("https://api.thingspeak.com/channels/{}/fields/{}.csv?results={}".format(Ch_id, 2, readings))   #Writes (uploads) count value from ThingSpeak
            data=y.read().decode('ascii')
            data=",".join(data.split("\n"))
            for i in range(5, readings*3+3, 3):
                values1.append(data.split(",")[i])
            element=float(values1[-1])  #Choosing most recent count value
            LCD.write(0,1, "Count:{}".format(element))  #Prints count value on LCD
            time.sleep(3)
            LCD.write(0,0,"                ")   #Clears both lines of the LCD as it moves back into the loop and restarts
            LCD.write(0,1,"                ")
```

System Photo

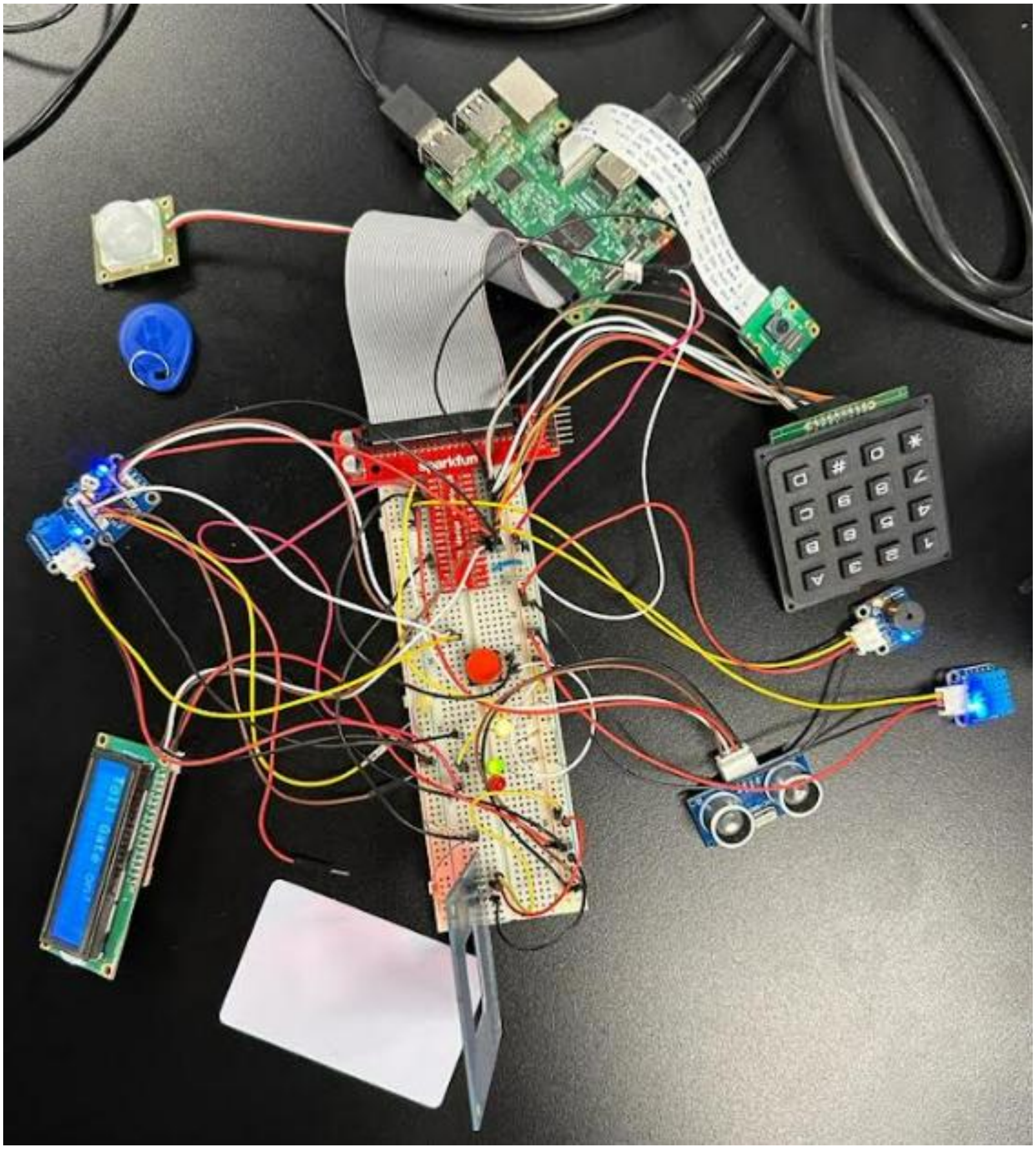


Figure 6: System photo of Smart Toll Gate

Test Results

When the program is first run, the toll gate is put in the “OFF” mode, which is indicated by the red LED being on, the green LED being off, and the LCD displaying the message “Press the button”, which can be seen in Figures 7 and 8.

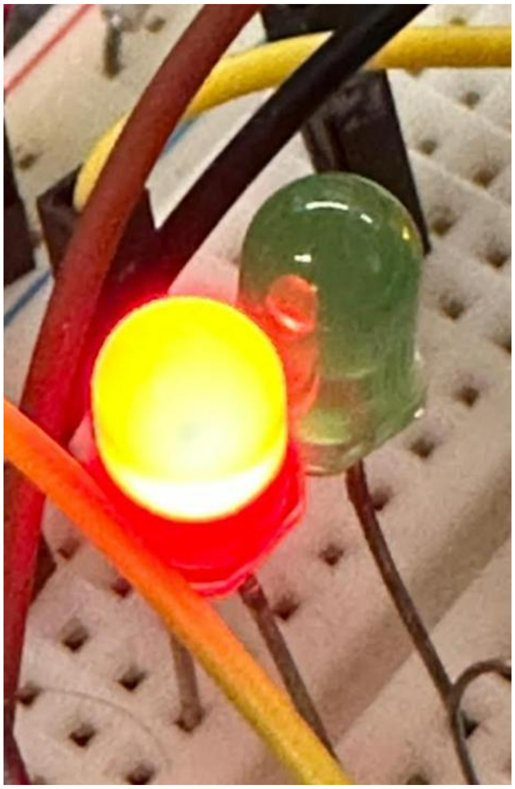
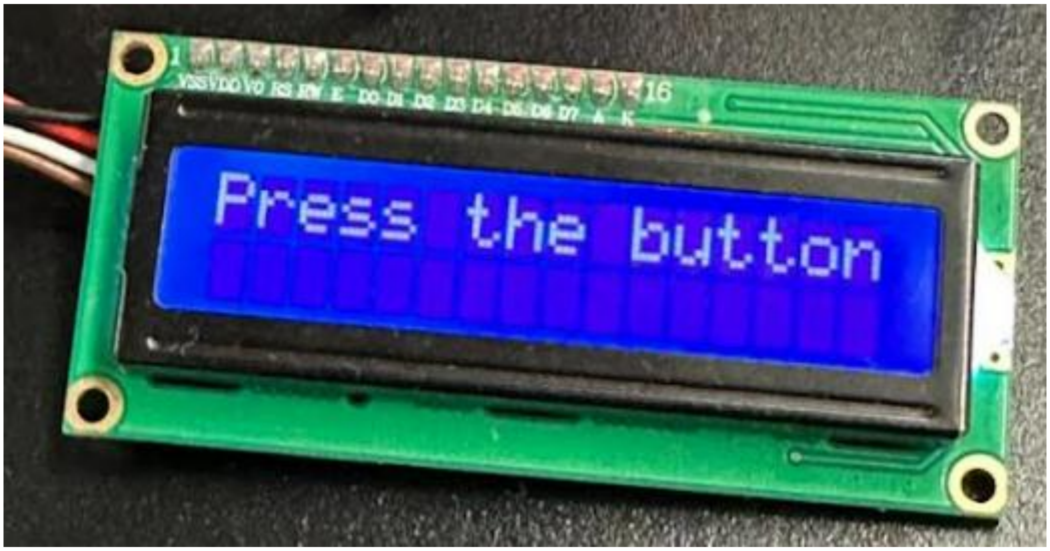


Figure 7: Red LED on and green LED off



Press the button

Figure 8: LCD displaying message in “OFF” mode

In the off mode, the user is able to utilize the infrared interrupt, which can be used to add to their balance or run the flask server, which are accessed by pressing 1 or 2 on the keypad respectively. This interrupt should be activated at the start, as it allows for the balances of each car to be initialized. After being initialized, the user should press a button other than 1 or 2 then press the button, exiting the infrared interrupt and allowing the main function to run. When the button is pressed, the "ON" mode of the program begins, where the red LED turns off, the green LED turns on, and the LCD displays the message "Toll gate is on!" as can be seen in Figures 9 and 10.

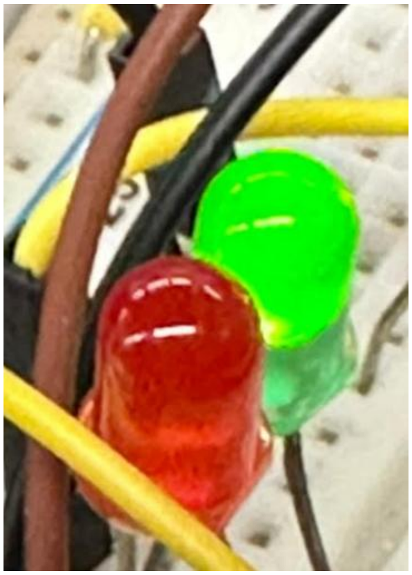


Figure 9: Red LED off and green LED on

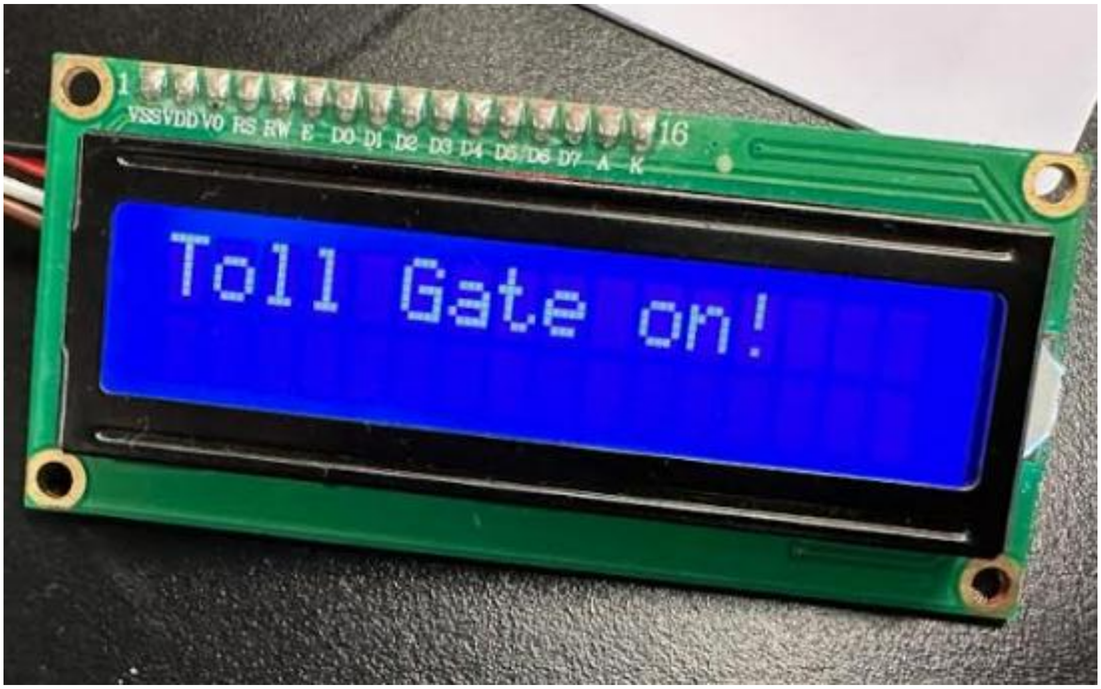
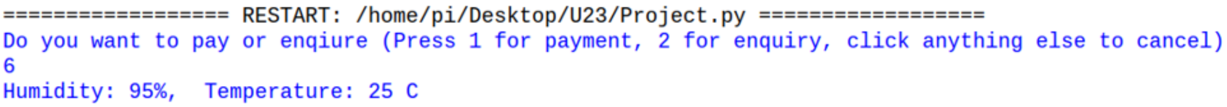


Figure 10: LCD displaying message in "ON"

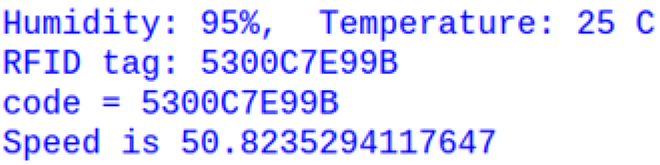
In the “ON” state, the program first reads the analog input from the AN0 channel of the PCF8591 device and uses it to set the toll rate that would be reduced from the passing cars’ balance. Next, the program reads the temperature and humidity through the DHT11 sensor and displays it in the terminal. Figure 11 shows the user deactivating the infrared interrupt by clicking a button other than 1 or 2 on the keypad followed by the temperature and humidity being displayed, indicating that the user pressed the button.



```
================== RESTART: /home/pi/Desktop/U23/Project.py ==================
Do you want to pay or enqiure (Press 1 for payment, 2 for enquiry, click anything else to cancel)
6
Humidity: 95%,  Temperature: 25 C
```

Figure 11: Exiting infrared interrupt and printing DHT11 output

Next, the program waits for the ultrasonic sensor to detect an object less than 10 cm away, which represents a car driving into the toll gate. After this occurs, the program waits for an RFID tag to be read, which is then printed. Next, the count is incremented, the speed of the car is read from the potentiometer through the AN1 channel of the PCF8591, and the speed is printed on the terminal. Then, the speed is converted to voltage and used to power the yellow LED. The printed messages on the terminal are shown in Figure 12 while the yellow LED changing based on the voltage is shown in Figures 13 and 14.



```
Humidity: 95%,  Temperature: 25 C
RFID tag: 5300C7E99B
code = 5300C7E99B
Speed is 50.8235294117647
```

Figure 12: Printed RFID and speed on terminal

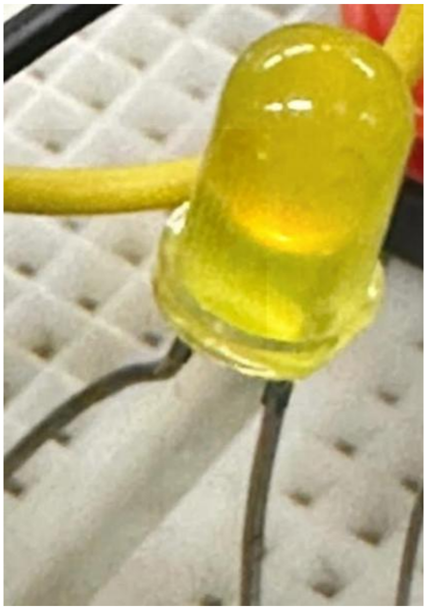


Figure 13: Yellow LED at smaller voltage

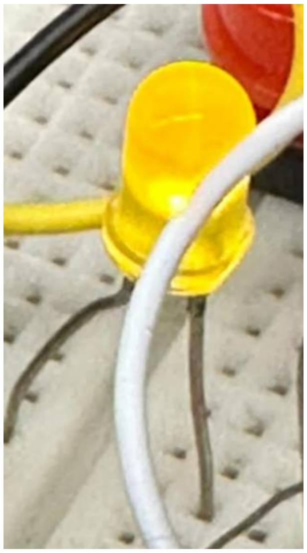
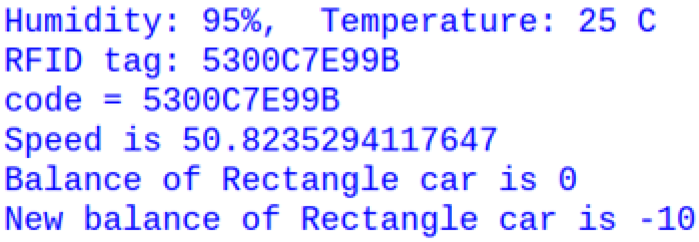


Figure 14: Yellow LED at larger voltage

With the RFID and speed of the car read, the next step is to take the toll from the car. Using the previously obtained toll value and the car's balance, the balance before and after the toll are printed on the terminal, as shown in Figure 15. If the new balance is less than 1, then the buzzer will be activated for 5 seconds to warn the driver that their balance is low.



```
Humidity: 95%,  Temperature: 25 C
RFID tag: 5300C7E99B
code = 5300C7E99B
Speed is 50.8235294117647
Balance of Rectangle car is 0
New balance of Rectangle car is -10
```

Figure 15: Printed balance in terminal

Next, the values of the speed, count, temperature, and humidity are uploaded to ThingSpeak, as shown in the ThingSpeak screenshots on page 27. The values of speed and the count are then read from ThingSpeak and the last read values are displayed on the LCD, as shown in Figures 16 and 17.

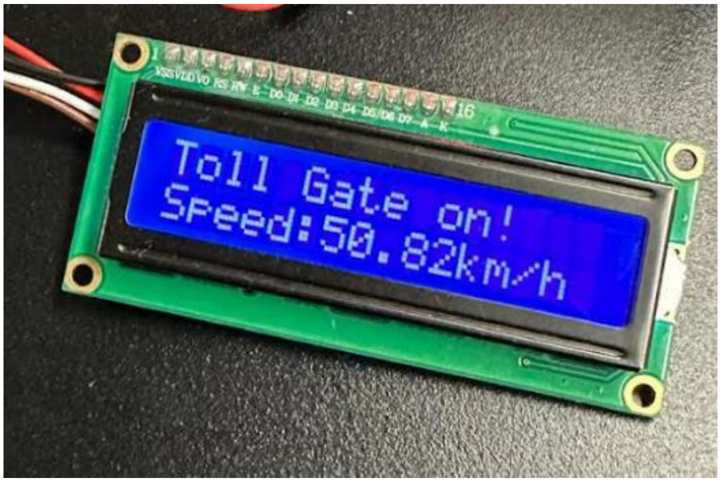


Figure 16: LCD displaying last recorded speed

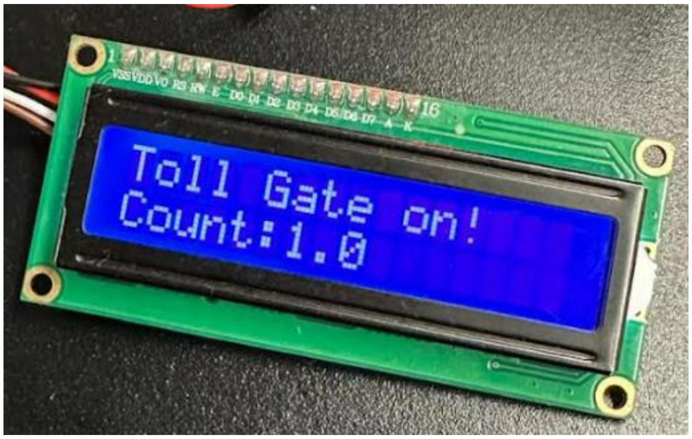
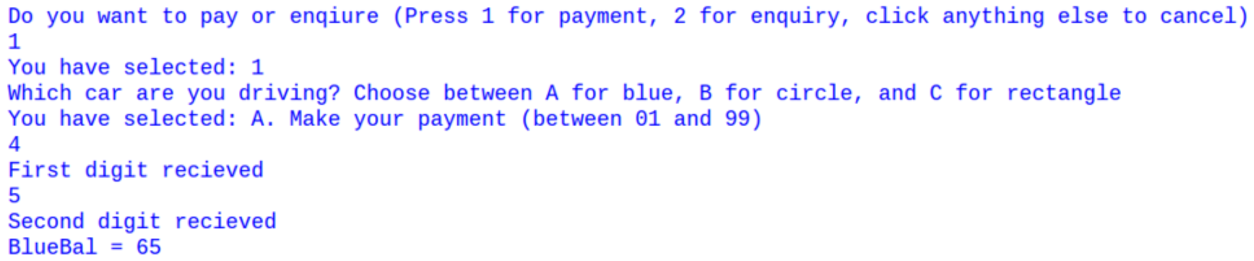


Figure 17: LCD displaying last recorded count

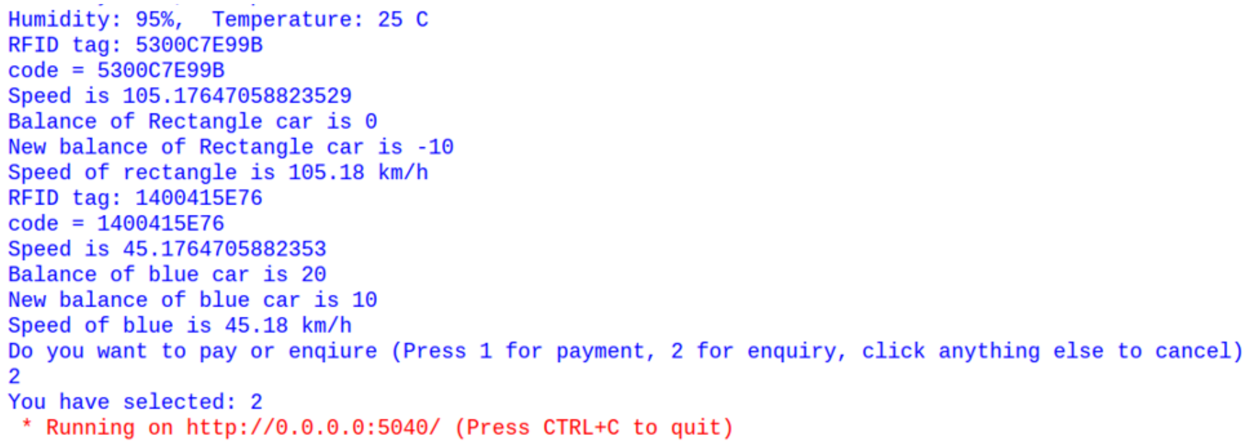
After a delay, the LCD is cleared and the program loops back to the beginning. When the user wants to add to their balance, they put the system into the “OFF” mode by pressing the button, which can be done at any point in the process as it operates using an interrupt. After the system is turned off, the user can access the payment and Flask server by first activating the infrared interrupt, which is done by moving their hand in front of the PIR sensor. When this occurs, a message will be displayed asking what the user would like to do. If they click something other than 1 or 2, then the interrupt will end as shown in Figure 11 above. However, if the user would like to add to their balance, then they click 1 on the keypad. Next, they will be prompted to select which car they are driving, with the blue car assigned the letter A, the circle car assigned the letter B, and the rectangle car assigned the letter C. Next, the user puts in a two digit number through the keypad, which is then added to their account and displayed on the terminal. Figure 18 shows the user of the blue car, who has an initial balance of 20, add 45 to their account.



```
Do you want to pay or enqiure (Press 1 for payment, 2 for enquiry, click anything else to cancel)
1
You have selected: 1
Which car are you driving? Choose between A for blue, B for circle, and C for rectangle
You have selected: A. Make your payment (between 01 and 99)
4
First digit recieved
5
Second digit recieved
BlueBal = 65
```

Figure 18: Adding to balance through infrared interrupt

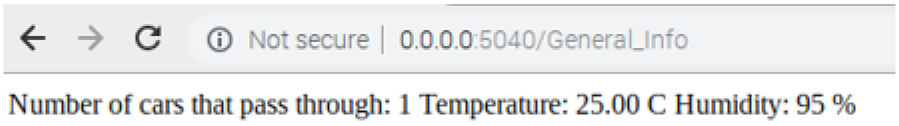
If the user clicks 2 instead of 1, then they are able to run the Flask server. Figure 19 shows the speed and balance of the blue car and the rectangle car after they pass the toll gate before the Flask server is run. These values will be shown on the routes of the Flask server shown in later Figures.



```
Humidity: 95%,  Temperature: 25 C
RFID tag: 5300C7E99B
code = 5300C7E99B
Speed is 105.17647058823529
Balance of Rectangle car is 0
New balance of Rectangle car is -10
Speed of rectangle is 105.18 km/h
RFID tag: 1400415E76
code = 1400415E76
Speed is 45.1764705882353
Balance of blue car is 20
New balance of blue car is 10
Speed of blue is 45.18 km/h
Do you want to pay or enqiure (Press 1 for payment, 2 for enquiry, click anything else to cancel)
2
You have selected: 2
 * Running on http://0.0.0.0:5040/ (Press CTRL+C to quit)
```

Figure 19: Flask server being activated

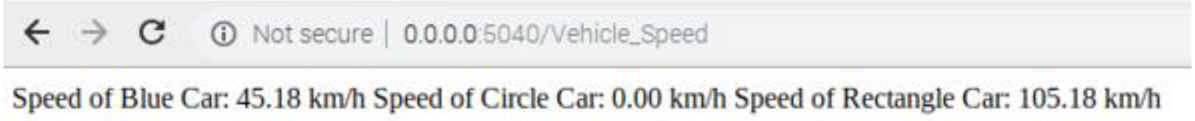
The first web server that the user can access is the index, which simply displays a greeting to the user. The first static route allows the user to observe the general information of the toll gate, such as the count, temperature, and humidity, as shown in Figure 20.



Number of cars that pass through: 1 Temperature: 25.00 C Humidity: 95 %

Figure 20: First static route of Flask server

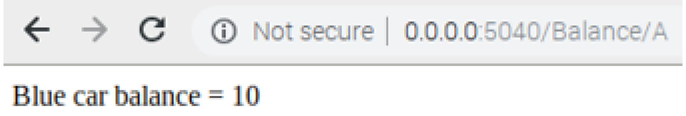
The other static route displays the last recorded speed of each car as they passed through the toll gate, shown in Figure 21. The speeds recorded match the speeds shown in Figure 19 above.



Speed of Blue Car: 45.18 km/h Speed of Circle Car: 0.00 km/h Speed of Rectangle Car: 105.18 km/h

Figure 21: Second static route of Flask server

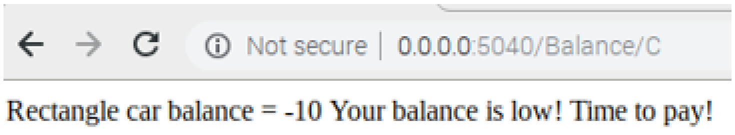
Next, the user can access the amount they have in their balance using a dynamic route, where the user of the car uses the same letter assigned to them in the keypad. If their balance is low, then a warning message is displayed. The balance of the blue car, who has high balance, and the rectangle car, who has low balance, are shown below in Figures 22 and 23.



Not secure | 0.0.0.0:5040/Balance/A

Blue car balance = 10

Figure 22: First dynamic route of Flask server when balance is high

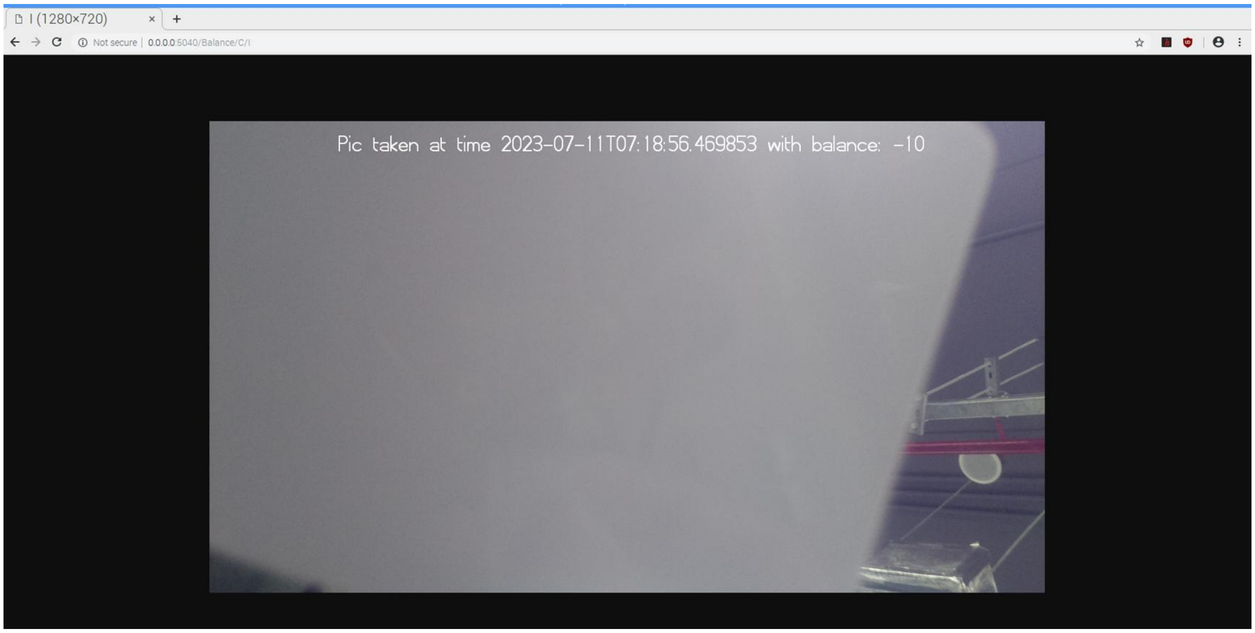


Not secure | 0.0.0.0:5040/Balance/C

Rectangle car balance = -10 Your balance is low! Time to pay!

Figure 23: First dynamic route of Flask server when balance is low

The other dynamic route allows the user to either take a picture or video. The picture and video will be annotated with the time and the balance of the user who accessed the picture. The picture of the rectangle car, who has low balance, is shown in Figure 24 while the download of the video is shown in Figure 25.

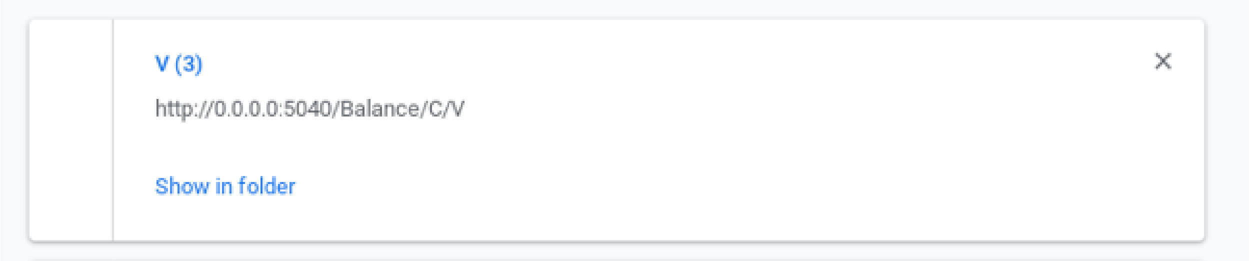


I (1280×720)

Not secure | 0.0.0.0:5040/Balance/C/I

Pic taken at time 2023-07-11T07:18:56.469853 with balance: -10

Figure 24: Second dynamic route of Flask server to take image



V (3)

http://0.0.0.0:5040/Balance/C/V

Show in folder

Figure 25: Second dynamic route of Flask server to take video

ThingSpeak Screenshots

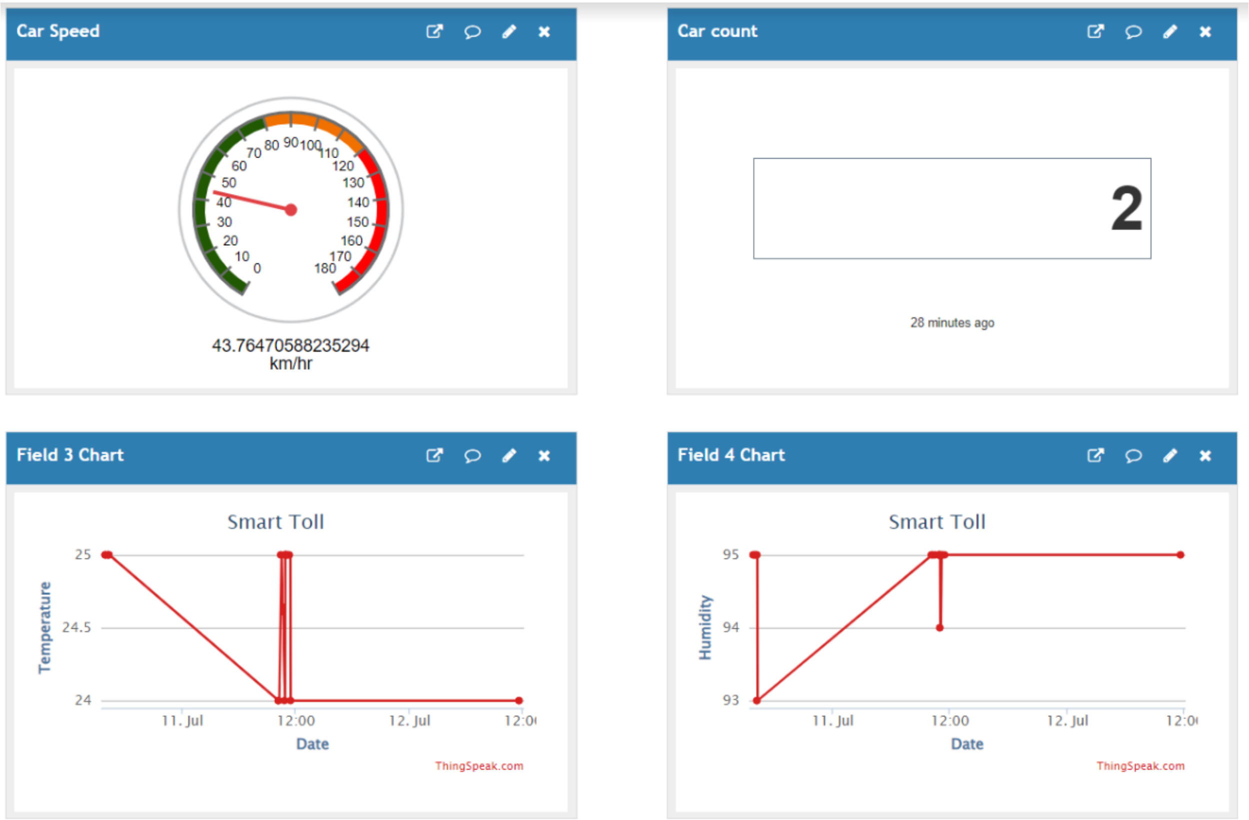


Figure 27: Thinkspeak widgets

MIT App Screenshots

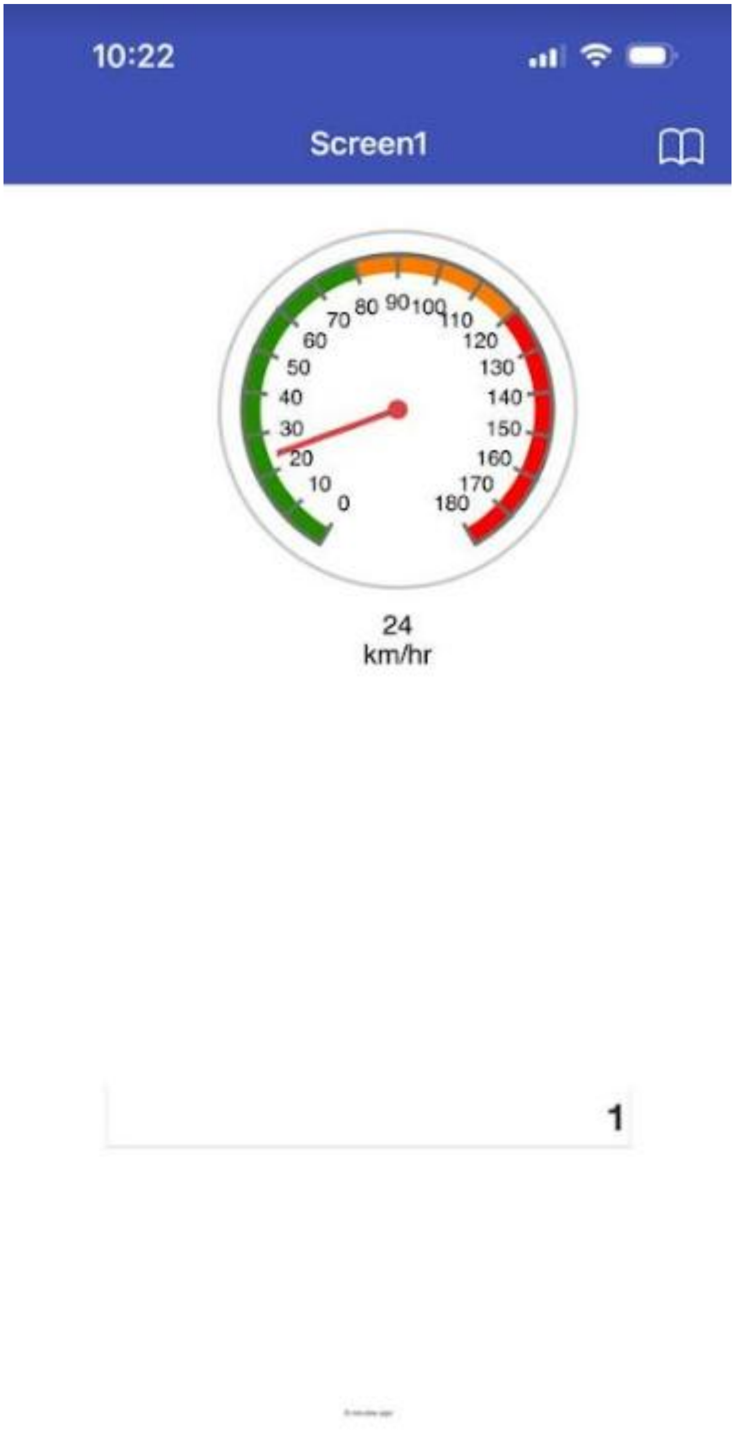


Figure 28: MIT app