Premises Security System

Nathan Cusack

Bachelor of Software & Electronic Engineering

Galway-Mayo Institute of Technology

2019/2020

Github: https://github.com/NatCusack/Premises-Security-Project

Premises Security System

Name: Nathan Cusack
Email: G00338306@gmit.ie
Github: Github.com/NatCusack

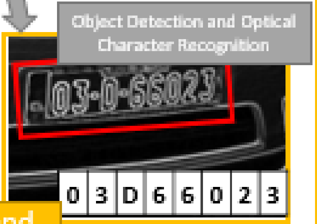
Operation

- Ultrasonic Sensor detects approaching car
- A picture is captured and scanned for Licence plate
- The plate is then check against the database
- If approved the gate opens, else an email is sent to the owner

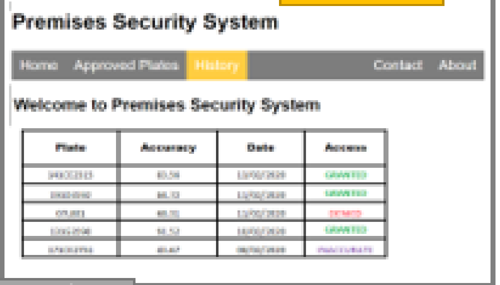
Functionality of ALPR

ALPR uses computer vision to extract a string from a licence plate. There are a number steps:

- Pre-processing – the image is filtered and processed for easier object detection
- Object detection – The image is scanned using a trained model for a licence plate
- Optical Character Recognition – In this step the picture is scanned for characters that are parsed to a string



Website



Premises Security System

Home | Approved Plates | History | Contact | About

Welcome to Premises Security System

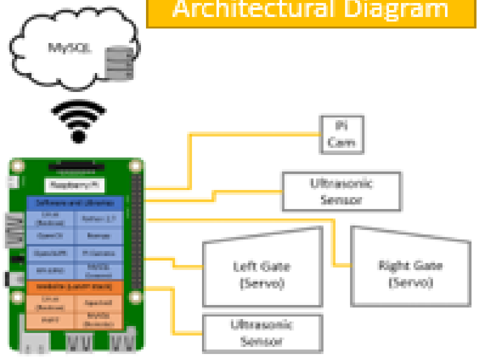
Plate	Accuracy	Date	Access
[illegible]	[illegible]	[illegible]	GRANTED
[illegible]	[illegible]	[illegible]	GRANTED
[illegible]	[illegible]	[illegible]	DENIED
[illegible]	[illegible]	[illegible]	GRANTED
[illegible]	[illegible]	[illegible]	[illegible]

Technologies used

Software
- Python
- Bash
- Apache2 Web Server
- Html/CSS/JavaScript
- Structured Query Language
- PHP
- Computer Aided Design (CAD)

Hardware
- Raspberry Pi
- Pi Camera
- Ultrasonic Sensors
- Servo Motors
- 3D Printer

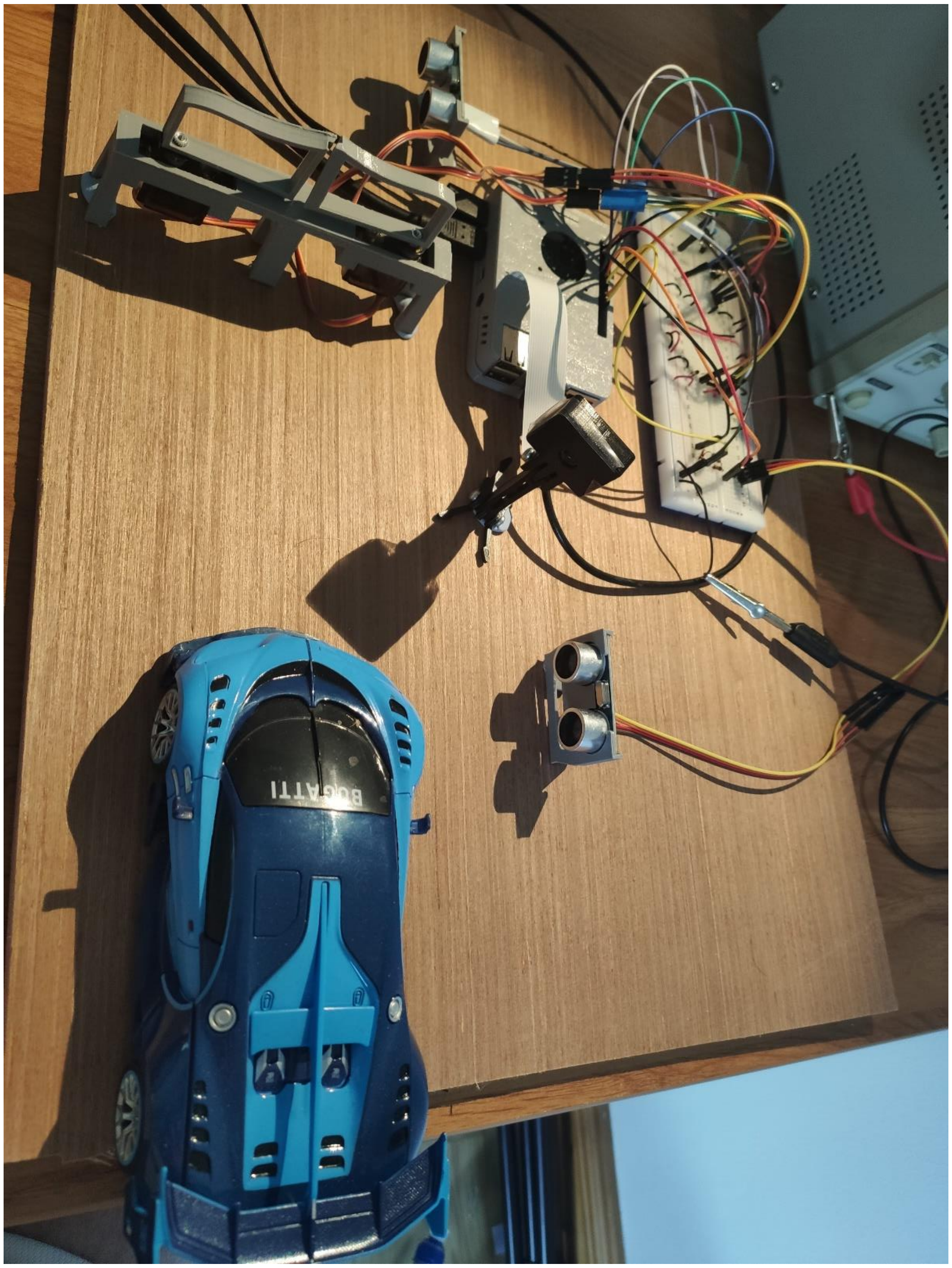
Architectural Diagram



Conclusion and Future work

All features of my project worked well together. Time spent researching really benefited the progression of the project. There are still a number of features I would like to tweak and add in the future such as:

- Mobile App Integration
- Livestream from camera over cloud
- Increase ALPR accuracy


BUGATTI

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

[illegible]

Acknowledgements

I would like to thank all my lecturers from Software and Electronic Engineering, in particular Mairtin O'Conghaile for his support and guidance throughout the year.

I would also like to take this opportunity to thank my Uncle, Francis Fahy, and cousin, Mark Fahy, who allowed me to pick their brain time after time.

Thank you.

Table of Contents

1 Summary

For my final year project, I decided to design and build an automatic electric gate system using Automatic Licence Plate Recognition (ALPR) technology. The system is designed to detect approaching vehicles, scan the licence plate, check said licence plate against a pre-approved list of plates designated by the premises owner. The idea came to me one evening when travelling home for work on a wet, stormy night. I had arrived at my house to locked gates. After I had gone through the process of opening and closing the gates behind me, I was drenched. I immediately started to think of a way to automate this process. After spending some time researching, I was pointed towards the direction of ALPR technology used in systems like barrier free toll on the M50 [1]. After developing my idea further, I decided to pursue this project.

When designing the system, I made the following features a priority:

- Secure: The main purpose of this project is to add an extra sense of security for the user
- Functional Offline: In the case of no access to the internet, the system should be able to operate offline
- Easy to use, straightforward user interface (UI): The UI should be self-explanatory and aid the user that is interacting with the system
- Stored Database: The system should have an organized catalog of all scanned plates stored remotely in the cloud

The project itself is mainly based in Python utilising OpenCV [2] (a library for optimal computer vision coding) and OpenALPR [3] (an open source licence plate recognition library with prebuilt weights for various nations licence plates) running on the latest raspberry pi 4 model B [4]. The system uses a Raspberry Pi camera and two ultrasonic sensors for both motion detection and obstruction detection. The website was built using the Linux, Apache, MySQL, and PHP stack commonly referred to as the LAMP stack [5].

2 Introduction

In this report I will cover the process I undertook for this project. This will cover research, building, developing and fully prototyping the Premises Security System.

The interest in my project originally stemmed from my work placement with LMI Ericsson where I got the opportunity to work on a lot of machine learning projects as part of the Prototype and Client Trials team. I was tasked with two separate research projects one of which involved an application using Optical Character Recognition (OCR). The more I researched this discipline of computer vision, the more I started to grow an interest in applying it in my final year project.

Since there was a broad range of applications for computer vision I wanted to find a way that I could both challenge myself in multiple new technologies while also reinforcing previous skills and knowledge I had gained over my years studying Software and Electronic Engineering.

In this project I have used several technologies I would have considered myself unfamiliar with (These include: Structed Query Language, PHP, Raspberry Pi GPIO, Servo Motors and OpenCV) and also a number of technologies that I have past experience using (Linux, 3D modelling, Ultrasonic Sensors) .

3 Hardware

In this section I will discuss the research, basic operation, and integration of said hardware to the Premises Security System

3.1 Raspberry Pi

The brains of the operation, the raspberry pi 4 model B, is a credit card sized computer that is both cost effective and power efficient. Whilst researching my project I realised that to run all the services I need for the system I would need a minimum of 2 GB RAM. Looking through my options for development boards with enough power while also having a larger number of GPIO pins I came back with two main competitors: NVidia's Jetson TX2 [7] (a relatively new development board with integrated graphic processing units) or the Raspberry Pi 4 Model B. Upon deeper investigation, I decided to go for the Raspberry Pi 4 due to the large community of makers supporting the system and most importantly the cost at only €54.

3.1.1 Technical Specifications

Although this new Raspberry Pi offers a surplus of new and updated features, here is a list of the key features that made the Model B the superior option for this system:

- 1.5GHz 64-bit quad-core CPU
- 4GB of RAM
- 2.5GHz and 5GHz wireless
- Gigabit Ethernet
- 40 Pin GPIO pin header
- 2 lane MIPI CSI camera port
- OpenGL ES 3.0 graphics
- 5V 3A Power In

3.1.2 Operating System

Just like a standard computer, the Raspberry Pi must be flashed with an operating system (OS). There are a number of available OS for the Pi, the majority of them Unix based with exception to a few, such as Windows 10 and Android Things.

From a development standpoint there was only one clear option, Raspbian, a Debian based computer operating system that has been specifically optimized for the Raspberry Pi architecture. The OS is still under active development and has a large support community which helped greatly throughout this project as I found many issues I had answered by the very active Raspberry Pi Forums.

I flashed my Pi with Raspbian Buster Lite – a minimal image with the bare minimum software installed. I did this because of concerns for storage.

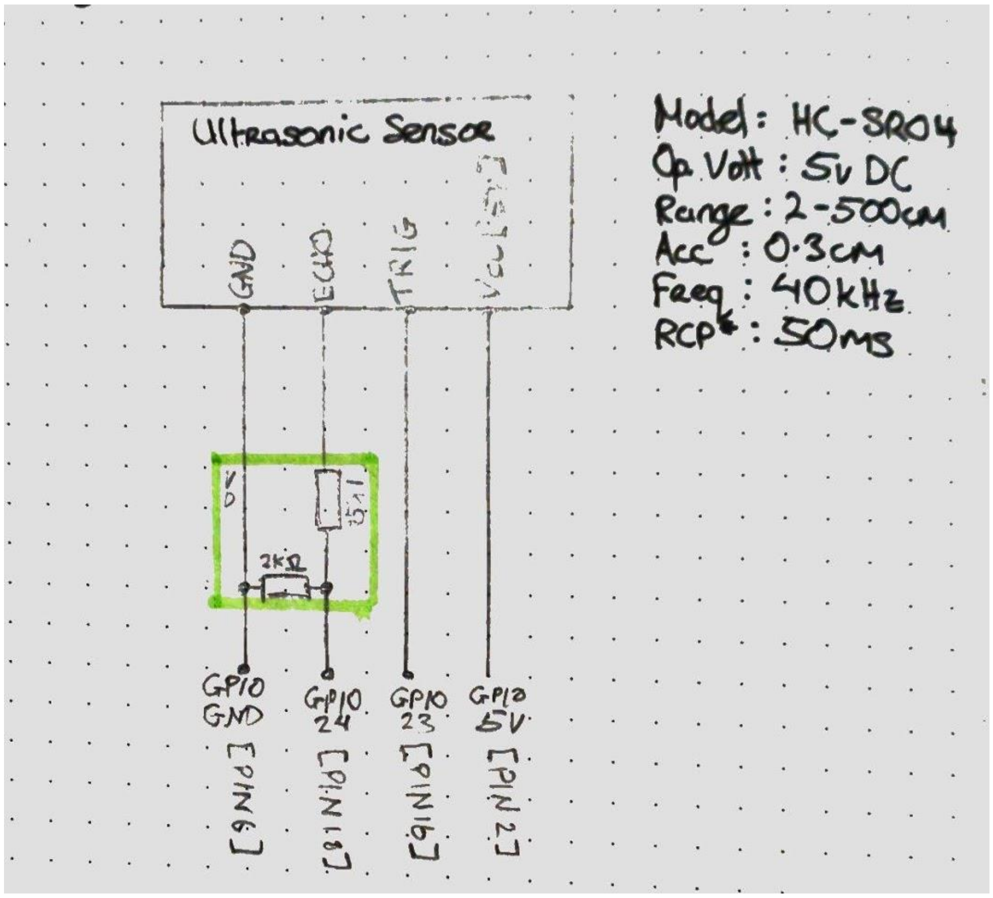
3.2 Ultrasonic Sensor

In total there are two ultrasonic sensors. I used the HC-SR04 ultrasonic sensor module [8]. One positioned in front and the other behind the gate. Both work in similar operation but provide two very different functions. One sensor is used for motion detection whilst the other is sensor is used for obstruction detection.

3.2.1 How Ultrasonic Sensors Work

The ultrasonic sensor sends out a high frequency pulse controlled by the Raspberry Pi. The pulse will bounce off an object and return to the receiver on the sensor and it will relay that information to the Pi. The time difference between the pulse being emitted and the pulse being received can be used to calculate the distance between the ultrasonic sensor and the object using the equation distance = speed of sound in air * time / 2.

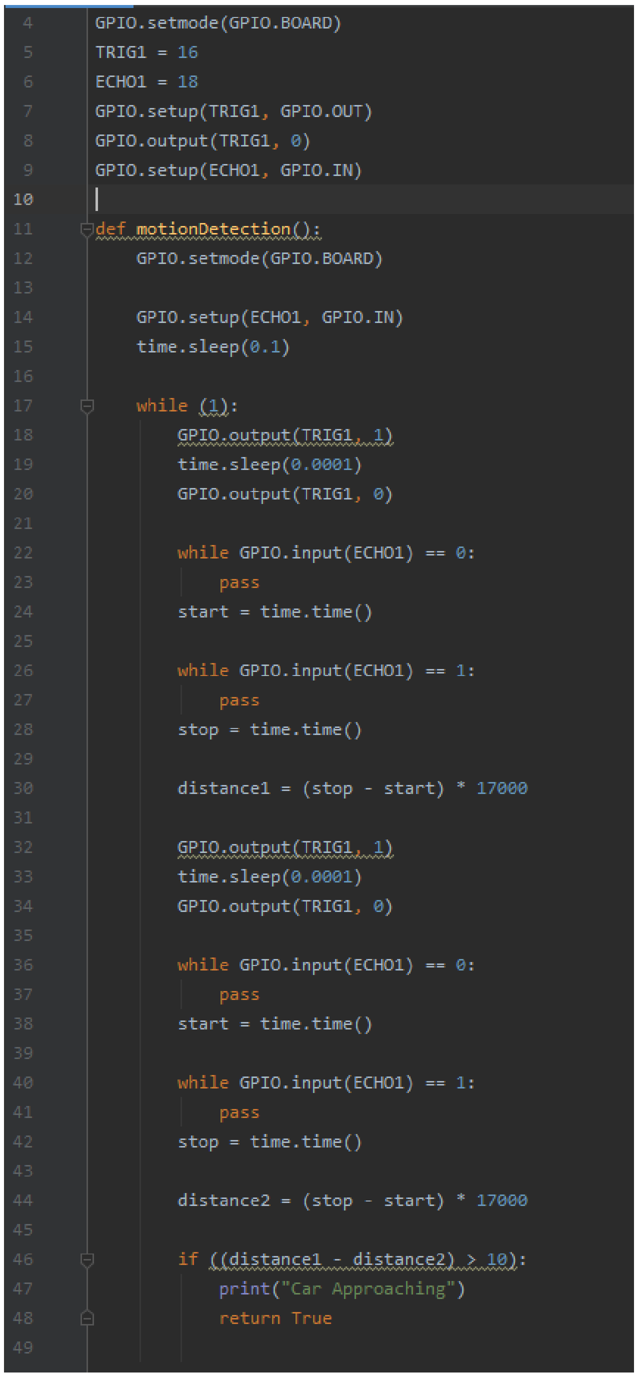
3.2.2 Wiring Diagram



The operating voltage of the ultrasonic sensor is 5v which can be powered by the Raspberry Pi, however, the operating voltage of the Raspberry Pi GPIO is 3.3V. The green highlighted box in the diagram above indicates a simple voltage divider circuit used to step down the input voltage for the Pi's GPIO.

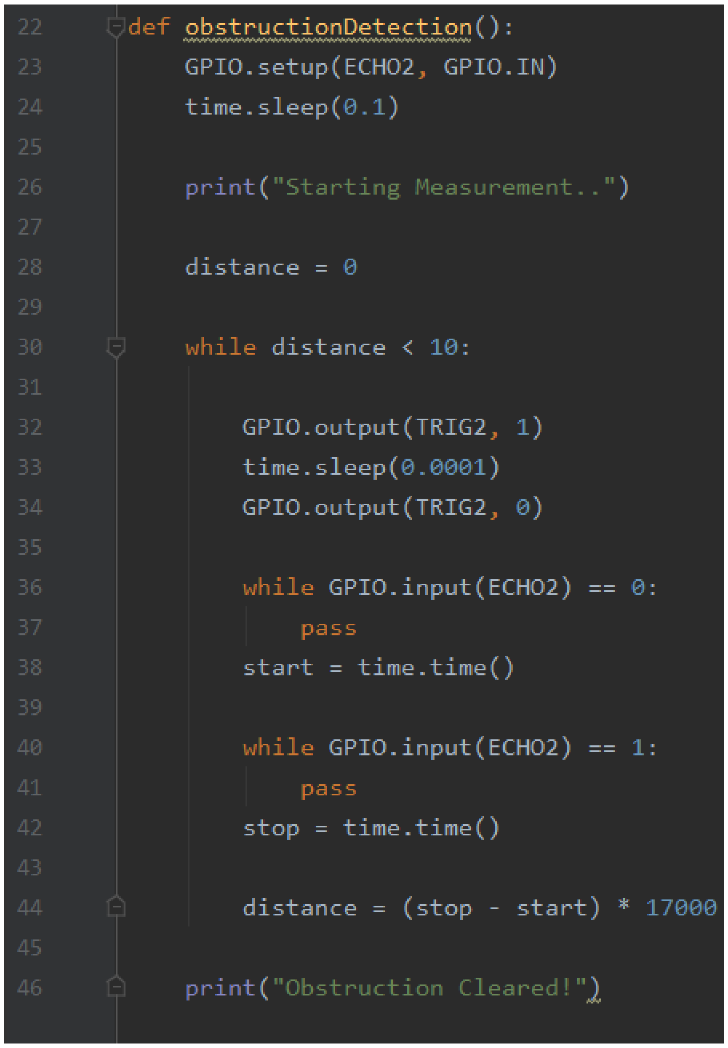
3.2.3 Motion Detection Code

To the right-hand side, you can see my code for motion detection. first the pins for the output frequency and received frequency are initialised (lines 4 – 10). The motion function then sets up the echo pin on each execution. This is to avoid "pin already in use" errors. Then the function enters a while loop getting an initial measurement. A second measurement is then taken. The first measurement is compared to the first measurement and if the difference is greater than 10 cm, it will exit the loop returning a True statement. This is a blocking statement that will wait until motion is detected.



```
4   GPIO.setmode(GPIO.BOARD)
5   TRIG1 = 16
6   ECHO1 = 18
7   GPIO.setup(TRIG1, GPIO.OUT)
8   GPIO.output(TRIG1, 0)
9   GPIO.setup(ECHO1, GPIO.IN)
10  
11  def motionDetection():
12      GPIO.setmode(GPIO.BOARD)
13  
14      GPIO.setup(ECHO1, GPIO.IN)
15      time.sleep(0.1)
16  
17      while (1):
18          GPIO.output(TRIG1, 1)
19          time.sleep(0.0001)
20          GPIO.output(TRIG1, 0)
21  
22          while GPIO.input(ECHO1) == 0:
23              pass
24          start = time.time()
25  
26          while GPIO.input(ECHO1) == 1:
27              pass
28          stop = time.time()
29  
30          distance1 = (stop - start) * 17000
31  
32          GPIO.output(TRIG1, 1)
33          time.sleep(0.0001)
34          GPIO.output(TRIG1, 0)
35  
36          while GPIO.input(ECHO1) == 0:
37              pass
38          start = time.time()
39  
40          while GPIO.input(ECHO1) == 1:
41              pass
42          stop = time.time()
43  
44          distance2 = (stop - start) * 17000
45  
46          if ((distance1 - distance2) > 10):
47              print("Car Approaching")
48              return True
49  
```

3.2.4 Obstruction Detection



```
def obstructionDetection():
    GPIO.setup(ECHO2, GPIO.IN)
    time.sleep(0.1)

    print("Starting Measurement..")

    distance = 0

    while distance < 10:

        GPIO.output(TRIG2, 1)
        time.sleep(0.0001)
        GPIO.output(TRIG2, 0)

        while GPIO.input(ECHO2) == 0:
            pass
        start = time.time()

        while GPIO.input(ECHO2) == 1:
            pass
        stop = time.time()

        distance = (stop - start) * 17000

    print("Obstruction Cleared!")
```

On the left is my function for obstruction detection. This code is following the same principle as the previous method. It enters a blocking while loop. Instead of two measurements, it checks if the first measurement is less than 10cm. The function will remain looping until the measurement is greater than 10cm meaning that obstruction has been cleared and that the gates can close safely.

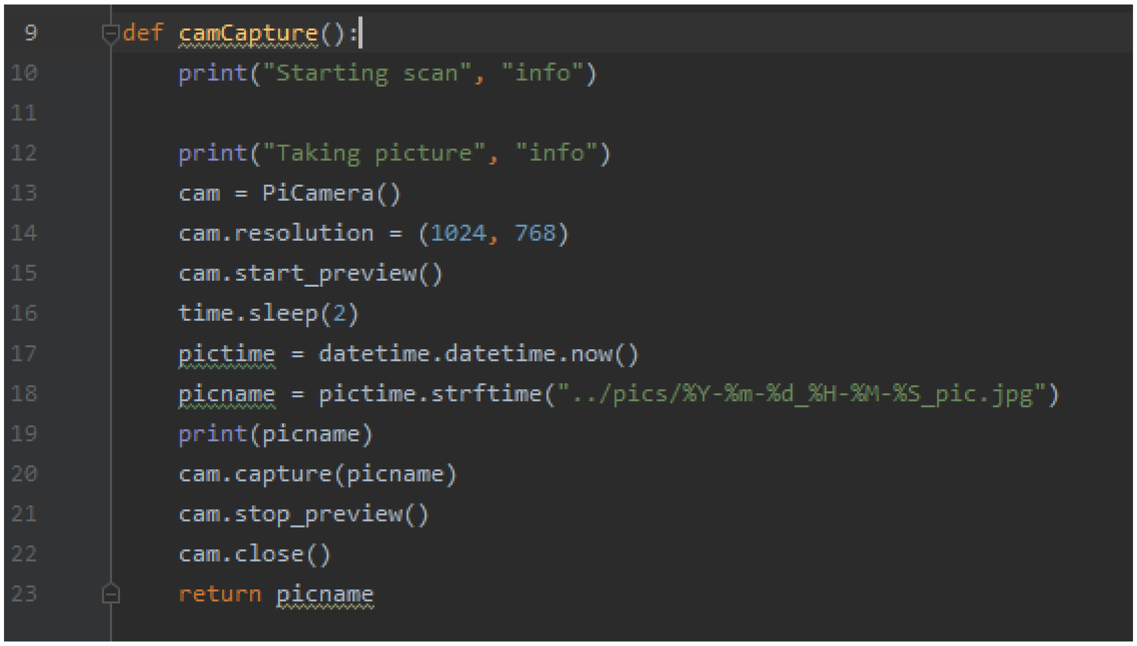
3.3 Raspberry Pi Camera

The Raspberry Pi Camera [9] is a low-cost, high quality camera designed and build for the most Raspberry Pi models and can be easily integrated into any python project thanks to third party libraries such as PiCamera.

3.3.1 Technical Specifications

I chose to use the infra-red Pi Camera V2 to ensure that the system could work at all possible light environments. At the time of ordering the component however (Early October) there was a back log of orders for the infra-red version of the camera that the Raspberry Pi Foundation were trying to accommodate. Because of this and being conscious of keeping on track of my timeline, I decided to use the standard model.

3.3.2 Image Capture Code



```python
def camCapture():
    print("Starting scan", "info")

    print("Taking picture", "info")
    cam = PiCamera()
    cam.resolution = (1024, 768)
    cam.start_preview()
    time.sleep(2)
    pictime = datetime.datetime.now()
    picname = pictime.strftime("../pics/%Y-%m-%d_%H-%M-%S_pic.jpg")
    print(picname)
    cam.capture(picname)
    cam.stop_preview()
    cam.close()
    return picname
```

Above is my code for capturing an image. This function is called when a vehicle is detected approaching the premises. First the resolution is set for the image that is about to be captured. cam.start_preview() will start a live stream locally from the camera. This is only for testing purposes to ensure the vehicle is in focus. The name given to the photo is path to the directory where the image will be stored and the current datetime variable parsed to a string using the strftime method, which will automatically pass the year value to %Y, month value to %m, day value to %d, hour value to %H, minutes value to %M and seconds value to %S. I decided to use this process of naming the captured JPEG to ensure there would be no error in duplicate file names. cam.capture(picname) will then capture the image and save it to the to directory previous specified with the filename previously specified.

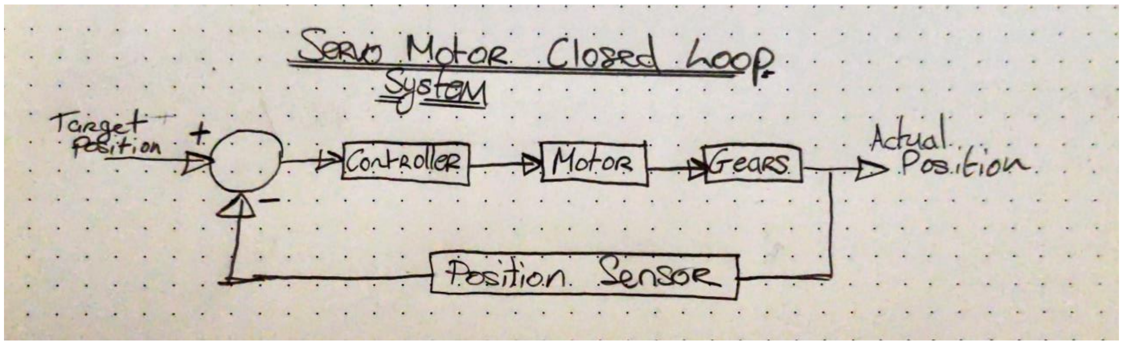
I had planned to use the Raspberry Pi Camera package “Motion” [10] for motion detection to trigger my python script to capture an image. Unfortunately, the motion function from the package caused conflict errors with my python script. After finding out that many others shared this issue decided to find an alternative method for motion detection using an ultrasonic sensor.

3.4 Servo Motor

Since I am building this system to be integrated into an electric gate system, I decided to use servo motors [11] to demonstrate the operation of the system. I used metal geared, 270° servo motor that operates at 5V. Since the Pi cannot provide a sufficient amount of current to drive the motors, I wired the supply voltage from an external power supply. I had to ensure that the servos would not draw too much current from the Pi's GPIO pins. This can be easily avoided by putting a 200 Ohm resistor in series of the PWM pin.

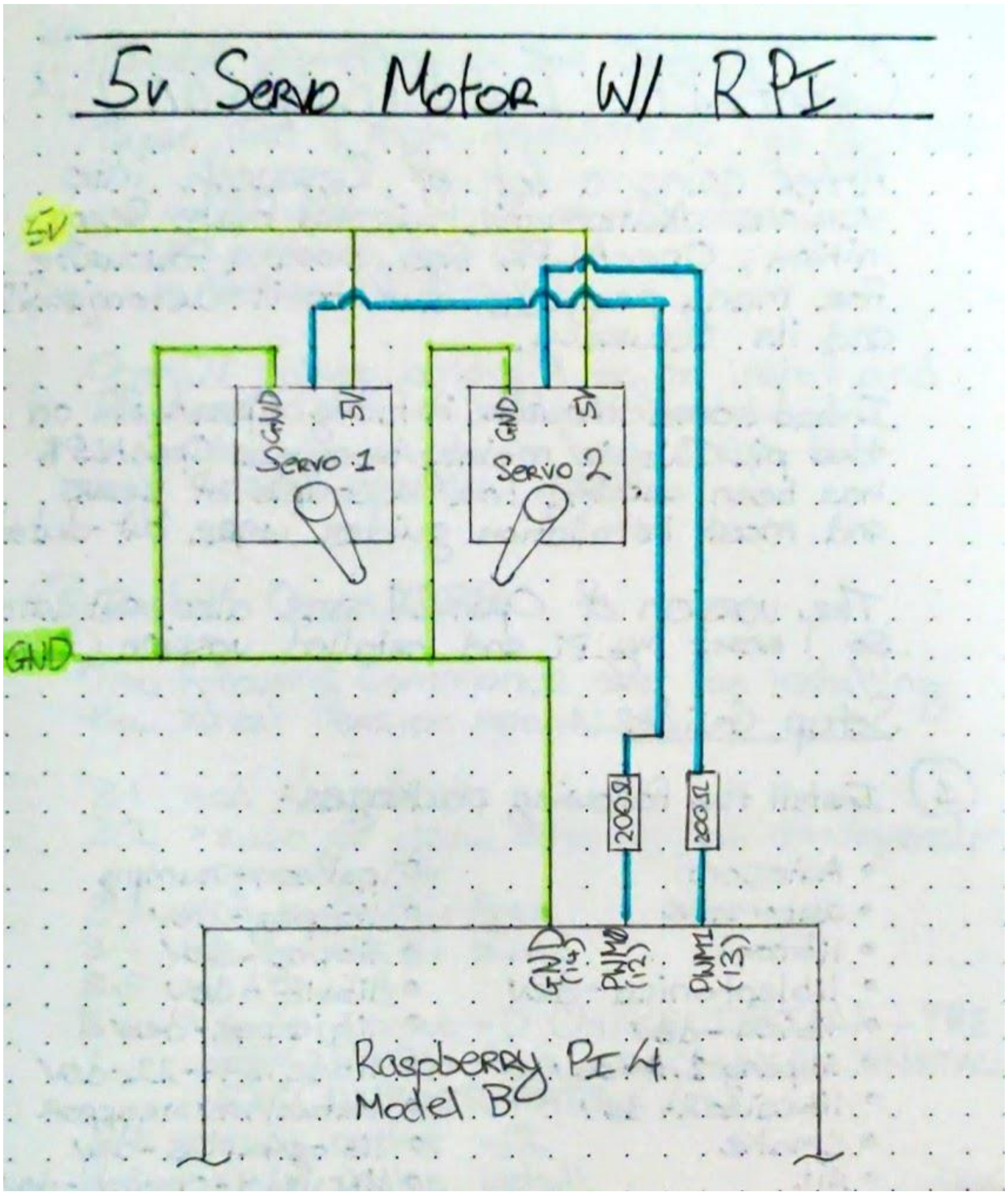
3.4.1 How Servo Motors Work

A servo motor is a motor that at any moment in time knows its rotary position. It is a closed loop system that ensures precise degrees of rotation. Standard servo motors are most used in the automation industry due to the precise control. Hobby motors, like the MG270s servo motors I am using for my project, are more commonly used in product prototyping and remote-controlled vehicles.

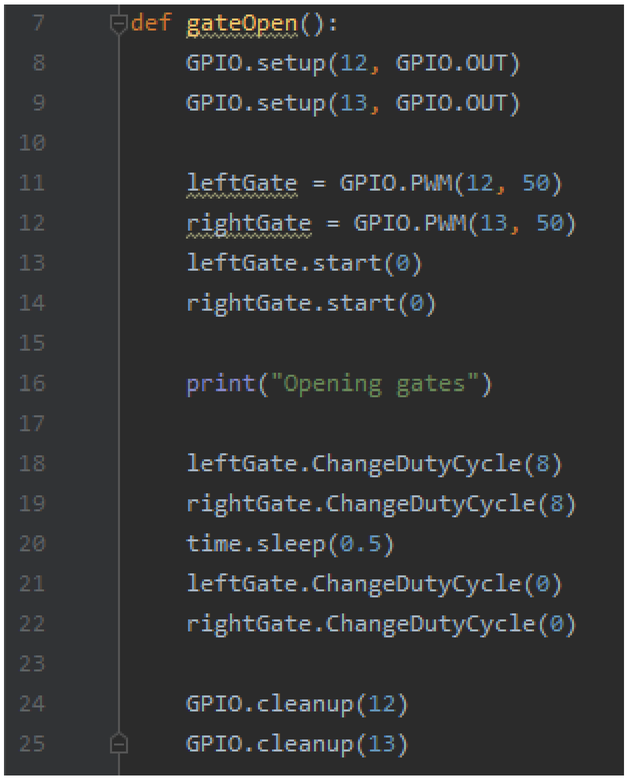


Both types of servo motors work the same way. A position is inputted to the servo through a 50Hz PWM signal. Note that the input frequency of the motor can vary based on the servo. In my case it is 50Hz. The width (or duration) of the pulse will decide the position of the servo. A 1 millisecond duration is said to be a 0° position, 1.5 millisecond equates to a 90° position and 2 millisecond duration equates a 180° position. The Inputted position is altered by subtracting the current position which is discovered but using a binary encoder attached to the shaft of the motor, or DC hobbyist servos, a potentiometer is connected to the output gear of the motor. When the position signal matches the target signal the servo is static.

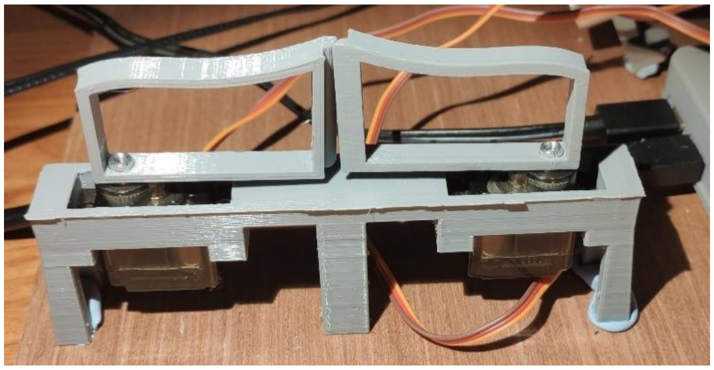
3.4.2 Wiring Diagram



3.4.3 Gate Operation Code



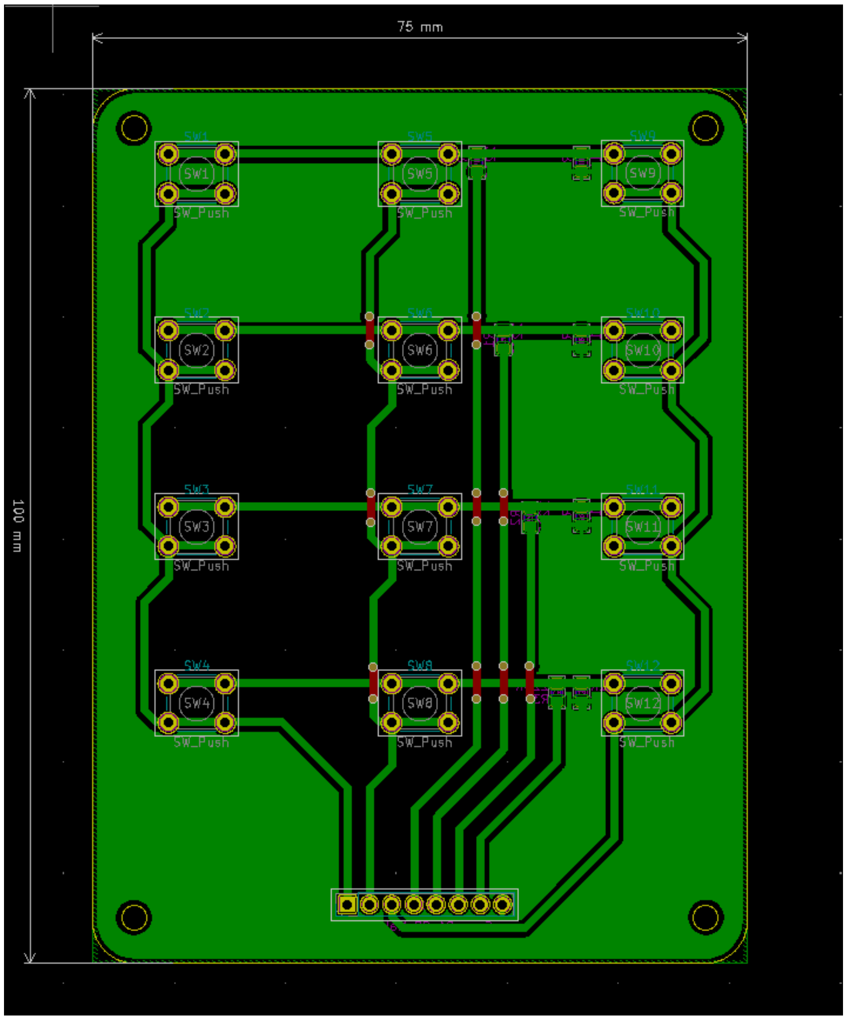
```
 7  def gateOpen():
 8      GPIO.setup(12, GPIO.OUT)
 9      GPIO.setup(13, GPIO.OUT)
10
11      leftGate = GPIO.PWM(12, 50)
12      rightGate = GPIO.PWM(13, 50)
13      leftGate.start(0)
14      rightGate.start(0)
15
16      print("Opening gates")
17
18      leftGate.ChangeDutyCycle(8)
19      rightGate.ChangeDutyCycle(8)
20      time.sleep(0.5)
21      leftGate.ChangeDutyCycle(0)
22      rightGate.ChangeDutyCycle(0)
23
24      GPIO.cleanup(12)
25      GPIO.cleanup(13)
```



Above is the code for controlling the motors of the gate. As mentioned in the previous section (3.4.1) , most hobbyist servo motors work off a 50 Hz PWM signal. In lines 11 to 12 you can see that I set the two output pins, “leftGate” and “rightGate”, for pulse width modulation (PWM) at a frequency of 50Hz. Meaning the period of one cycle is 20ms. Duty cycle then refers to what percentage of the cycle. Since these values account for positions in the real world, they are adjusted to the arrangement of the servo motors. At lines 13 and 14, I start the PWM signal with a duty cycle of 0. Then I up the duty cycle to 8 to open the gates. The delay is there to all the motors time to reach the assigned position. Then I set the duty cycle back to zero before clearing the pins. I do this to stop the motor stuttering when it has reached its assigned position. The reason the motor stutters is the potentiometer is not accurate enough to detect the position so it constantly changing by a miniscule percent of a degree which draws current and adds noise to the system.

3.5 Keypad

For an added layer of security, I wanted to add a keypad with a 4 digit passcode for entry. I started by researching how 3x4 keypad would work. I wanted to display my skill and knowledge of PCB design so I opted for the idea of building my own. Down below is my an image of gerber file. Due to the closure of the college I wasn't able to get my keypad printed.



4 Software

For this section I decided to break this section into two parts, the ALPR system, and the web server.

4.1 ALPR System

4.1.1 Python

I chose to use python as the main programming language for my project. There are few reasons for this: While I was on placement with Ericsson, all the projects I was working on were in python. I grew a fondness for is simplistic coding style and how easy it is to integrate different technologies. Another reason is because I had done 2 previous projects, in 2nd and 3rd year of college where I used C++ and Java. I wanted to broaden my horizons in languages and challenge myself. In hindsight, by using a language I was not familiar with it in turn, hindered a level of advancement to my project. However, I am happy without the project turn out.

The main reason I wanted to use Python is because I wanted to mainly focus on the features and functionality of my project.

4.1.2 OpenALPR

I had initially planned to code the pipeline for ALPR myself. I spent the majority of my research phase trying to learn OpenCV (an image processing/computer vision library) but source material for ALPR was slim and upon further research I discovered that it was an unrealistic goal to build on my own.

Upon the early stages of research, I came across OpenALPR, a company that offers an open source approach to ALPR as well as a commercial cloud-based API. The OpenALPR library is an open source automatic licence plate recognition library that is written in C++ with bindings in Python and C#. The library is built to analyse static images and extract licence plates to a string for further processing in an application.

4.1.2.1 How OpenALPR works

The OpenALPR pipeline is the brains of this operation. It goes through 6 stages before a licence plate can be extracted from an image:

1) **An Image is inputted to the System** – regardless if this image is working on an live stream from a camera or a collecting of images stored. The pipeline will be dealing with an image (or static frame from video)
2) **Plate Localisation and Isolation** – Through edge detection software and predefined plate pattern comparison the licence plate in the image is located and the image is then cropped to remove everything but the image
3) **Adjust Rotation and Skew** – The isolated plate is rotated and skewed so that it looks like that plate is facing you. This enables the image to be taken at an angle which helps accuracy
4) **Adjust Brightness and Contrast** – The image must be brightened and contrasted to clearly show the characters of the licence plate
5) **Character Segmentation** – Each character is individually separated. This is because in the next step, each character has to be individually inputted
6) **Character Recognition** – Each character is inputted one by one into a character recognition algorithm and concatenated to a string which is outputted from the pipeline

4.1.2.2 Installation of OpenALPR

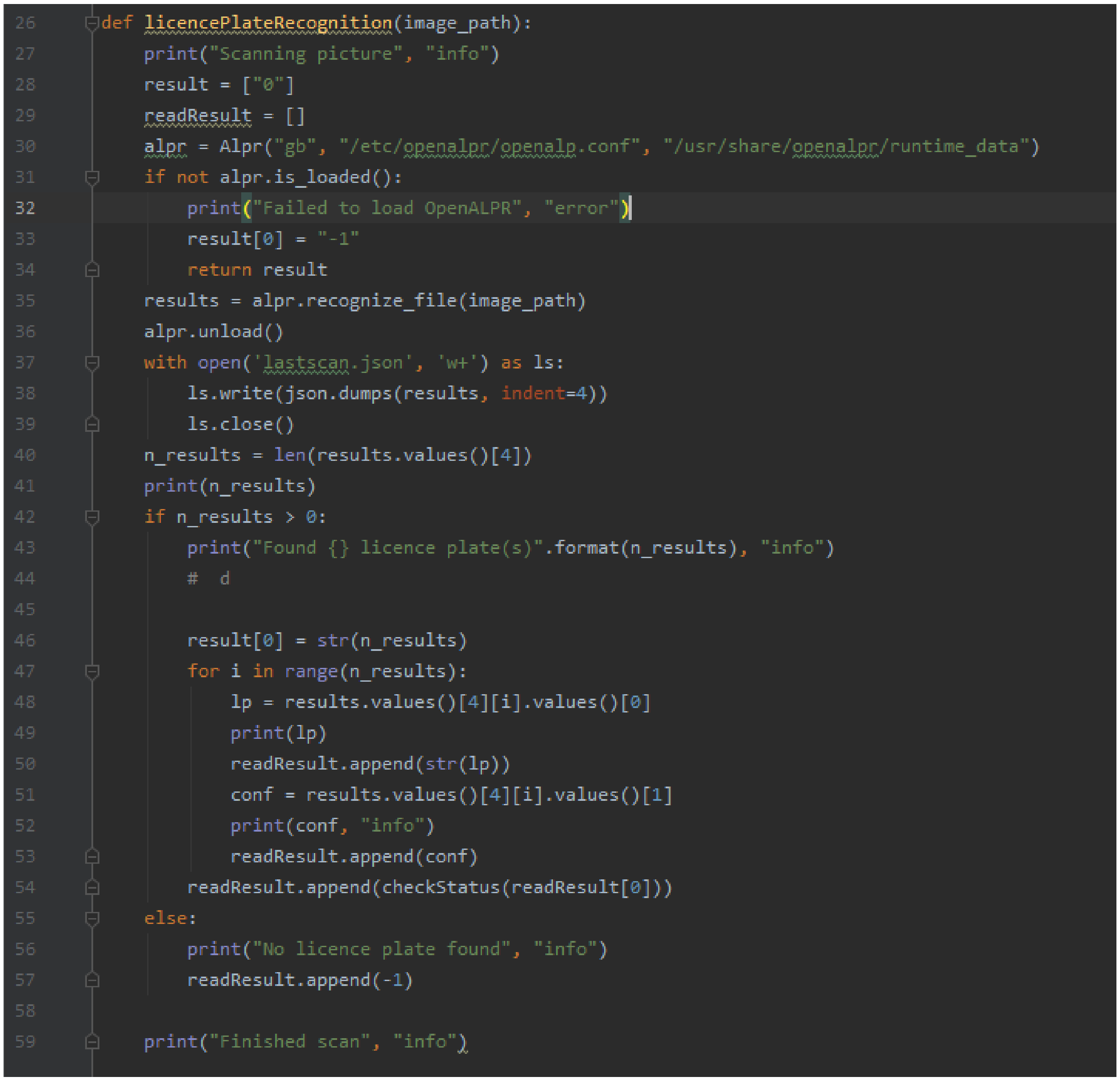
The process for installing OpenALPR [12] is quite difficult in its own way. The library relies on a lot of dependencies. Most importantly the OpenCV library where it takes most of its image processing methods. Other important libraries that OpenALPR work off are Leptonica [13], which is used for image analysis and Tesseract [14], for its optical character recognition algorithms.

4.1.2.3 Improving Accuracies on Irish Plates

Although OpenALPR has pre-built models for a large number of the world licence plates, the best accuracies for Irish licence plates was under the Great British LP models. Unfortunately, at best I was getting a 30% success rate. I decided to try and make my own model [15] to integrate with the library which was greeted with mixed result. This meant making my own configuration file. This involved inputting every possible character on a licence plate into an image library (defined as an optical character recognition language) as well as including measurements. This process was long and time consuming. Here is an example of the measurements (All in mm):

- width of full plate
- height of full plate
- width of a single character
- height of a single character
- whitespace between the character and the top of the plate
- whitespace between the character and the bottom of the plate

4.1.2.4 Licence Scan Code



```
def licencePlateRecognition(image_path):
    print("Scanning picture", "info")
    result = ["0"]
    readResult = []
    alpr = Alpr("gb", "/etc/openalpr/openalp.conf", "/usr/share/openalpr/runtime_data")
    if not alpr.is_loaded():
        print("Failed to load OpenALPR", "error")
        result[0] = "-1"
        return result
    results = alpr.recognize_file(image_path)
    alpr.unload()
    with open('lastscan.json', 'w+') as ls:
        ls.write(json.dumps(results, indent=4))
        ls.close()
    n_results = len(results.values()[4])
    print(n_results)
    if n_results > 0:
        print("Found {} licence plate(s)".format(n_results), "info")
        #  d

        result[0] = str(n_results)
        for i in range(n_results):
            lp = results.values()[4][i].values()[0]
            print(lp)
            readResult.append(str(lp))
            conf = results.values()[4][i].values()[1]
            print(conf, "info")
            readResult.append(conf)
        readResult.append(checkStatus(readResult[0]))
    else:
        print("No licence plate found", "info")
        readResult.append(-1)

    print("Finished scan", "info")
```

This function is the brains of the operation. The function takes in the path to the image that is to be scanned. I coded the function so that the ALPR library is loaded each time it is executed. This is because the pipeline takes up a lot of resources which the my hardware cannot keep up with. The image path is then passed to alpr.recognise() where it will begin the pipeline for ALPR by default the system runs the pipeline 10 times, slightly adjusting the image pre-processing (steps 2, 3 and 4 of the pipeline) and then it will output all 10 results in an array in order of most confident. Then I append the array to an about array with the most accurate plate and its confidence level. I also output all of the scanned information to a json file called “lastscan.json” This is mainly for debug reasons.

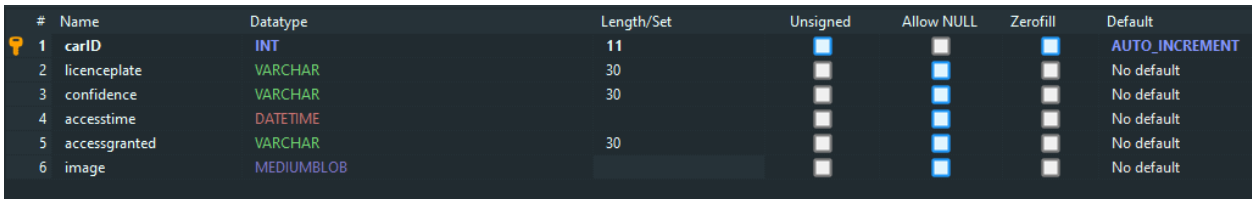
4.1.3 MySQL Server

I remotely run a MySQL server [16] on my laptop. On this server created a database where I built a table for storing car plates, images and the date and time of the scan. Originally, I had planned on running this on the Pi, but storage became an issue when I included images to the database.

4.1.3.1 Structed Query Language

I briefly worked on a project involving SQL while on placement which really helped me in building my database. Since I was going to be connecting to the database remotely, I first had to create a user that had remote access privileges.

I had spent a decent proportion of my time researching the datatypes that I would be storing in my database.

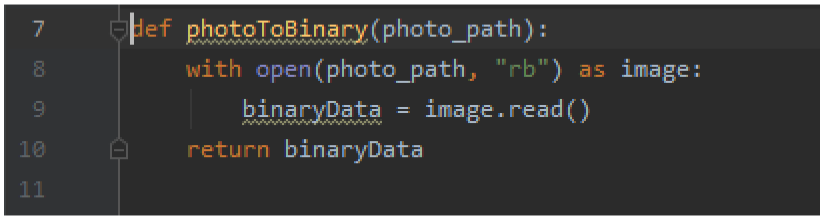


#	Name	Datatype	Length/Set	Unsigned	Allow NULL	Zerofill	Default
1	carID	INT	11	✓	☐	✓	AUTO_INCREMENT
2	licenceplate	VARCHAR	30	☐	✓	☐	No default
3	confidence	VARCHAR	30	☐	✓	☐	No default
4	accesstime	DATETIME		☐	✓	☐	No default
5	accessgranted	VARCHAR	30	☐	✓	☐	No default
6	image	MEDIUMBLOB		☐	✓	☐	No default

The main two datatypes which I had not used where Blob and Datetime. Blob is a binary large object and is used for storing large files in databases. To store an image in a text blob I had to first convert the image into binary and then send as a string to the database. Datetime is a single object variable that stores date and time information from the year all the way to the millisecond. This was very important for organising my data.

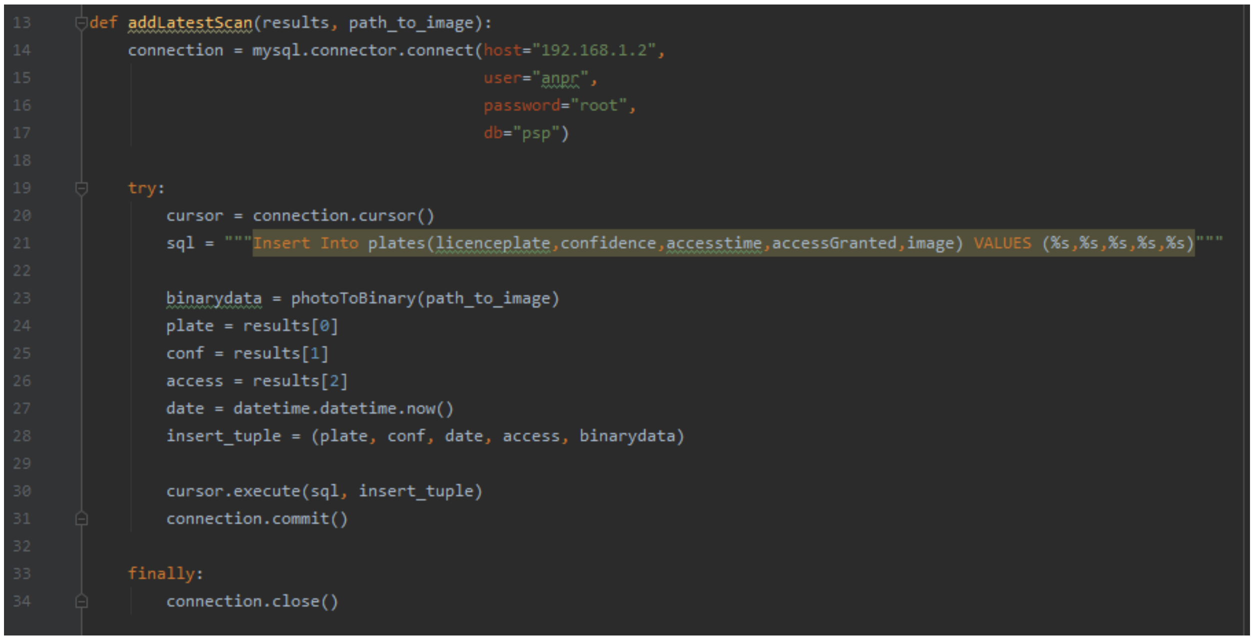
4.1.3.2 Interfacing with MySQL Server

To interface with the remote MySQL server I used the PyMySQL library but before I could interface with the database I first had to convert my image to binary which I did through the following code.



```python
def photoToBinary(photo_path):
    with open(photo_path, "rb") as image:
        binaryData = image.read()
    return binaryData

```

For connecting to my remote database through python I had to enter my login details. This include the host IP, username, password, and database name. Then using prepared statements inserted the values using the following function:

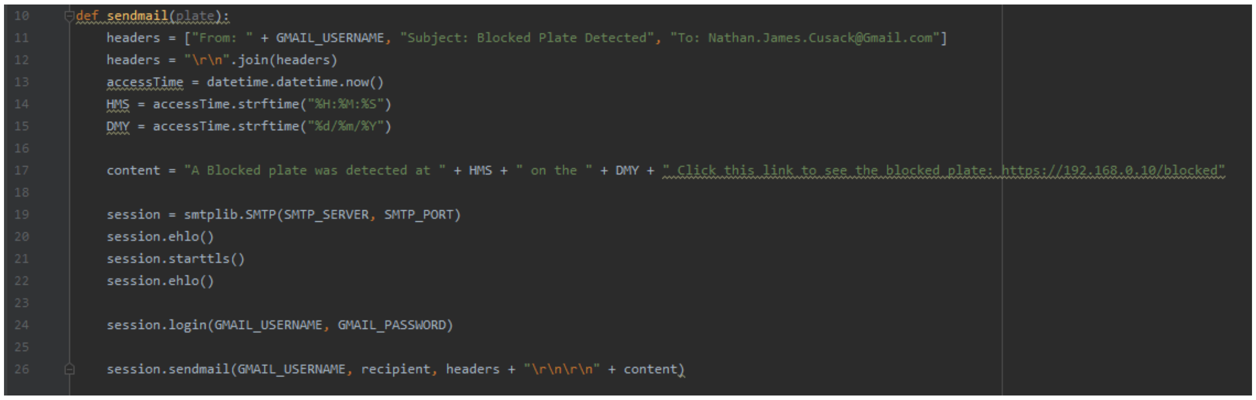


```python
def addLatestScan(results, path_to_image):
    connection = mysql.connector.connect(host="192.168.1.2",
                                         user="anpr",
                                         password="root",
                                         db="psp")

    try:
        cursor = connection.cursor()
        sql = """Insert Into plates(licenceplate,confidence,accesstime,accessGranted,image) VALUES (%s,%s,%s,%s,%s)"""

        binarydata = photoToBinary(path_to_image)
        plate = results[0]
        conf = results[1]
        access = results[2]
        date = datetime.datetime.now()
        insert_tuple = (plate, conf, date, access, binarydata)

        cursor.execute(sql, insert_tuple)
        connection.commit()

    finally:
        connection.close()
```

4.1.4 Simple Mail Transfer Protocol Server

To send emails from my systems I installed a Simple Mail Transfer Protocol (SMTP) server [17]. This server allows me to link an email account to it and send emails. For now I have the system send an email to the client whenever a blocked plate is recognized. SMTP is a set of commands that authenticates and directs the transfer of an email. SMTP uses Transmission Control Protocol (TCP) to ensure the delivery of an email. This makes sure that the email entered is active and available to send emails to, among other things.

4.1.4.1 SMTP Code



```
10  def sendmail(plate):
11      headers = ["From: " + GMAIL_USERNAME, "Subject: Blocked Plate Detected", "To: Nathan.James.Cusack@Gmail.com"]
12      headers = "\r\n".join(headers)
13      accessTime = datetime.datetime.now()
14      HMS = accessTime.strftime("%H:%M:%S")
15      DMY = accessTime.strftime("%d/%m/%Y")
16
17      content = "A Blocked plate was detected at " + HMS + " on the " + DMY + " Click this link to see the blocked plate: https://192.168.0.10/blocked"
18
19      session = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
20      session.ehlo()
21      session.starttls()
22      session.ehlo()
23
24      session.login(GMAIL_USERNAME, GMAIL_PASSWORD)
25
26      session.sendmail(GMAIL_USERNAME, recipient, headers + "\r\n\r\n" + content)
```

First a session is created by connecting to the SMTP server (line 19), then we identify ourselves to the server (line 20). Then we encrypt our traffic using transport Layer security protocol (TLS) and then reidentify ourselves with the server (lines 20 and 21 respectively). Then using our predefined user email and login details, we log in to the email of origin where we can now send our email to notify the user securely.

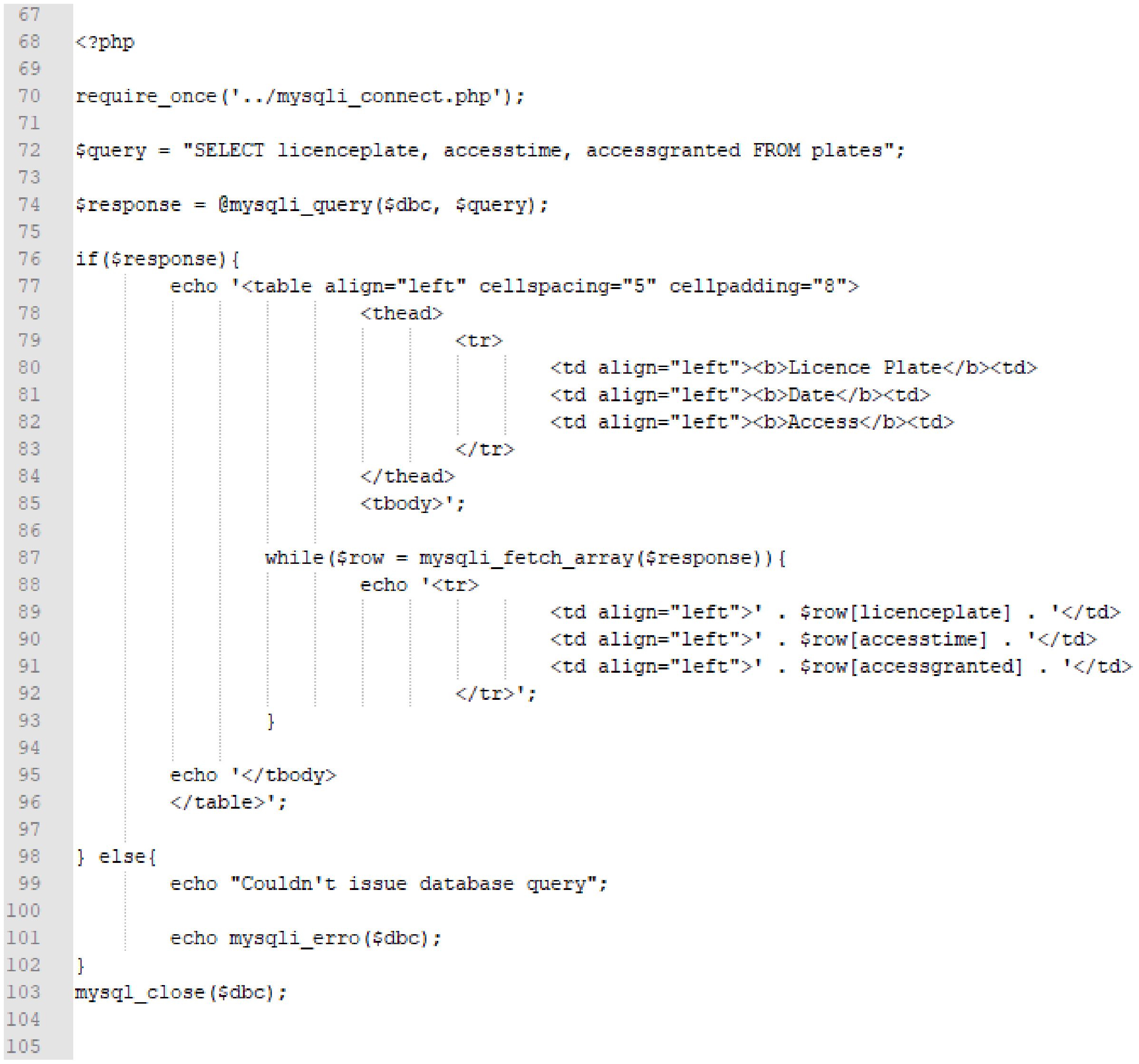
4.2 Web Server

For the website interface I decided to you the LAMP stack. This stands for Linux, Apache, MySQL, and PHP. The reasons I chose LAMP is because I had already integrated a MySQL database into my system, I also have familiarity with Linux and Apache, and I welcomed the new challenge of learning PHP. Currently the server is being run on my Raspberry Pi, same as the system, although it is full independent and from the system and could be run in the cloud.

The Website is coding using the standard html, css, and javascript languages for building the website. For the backend I used mainly PHP.

4.2.1 Connecting to The Remote Database

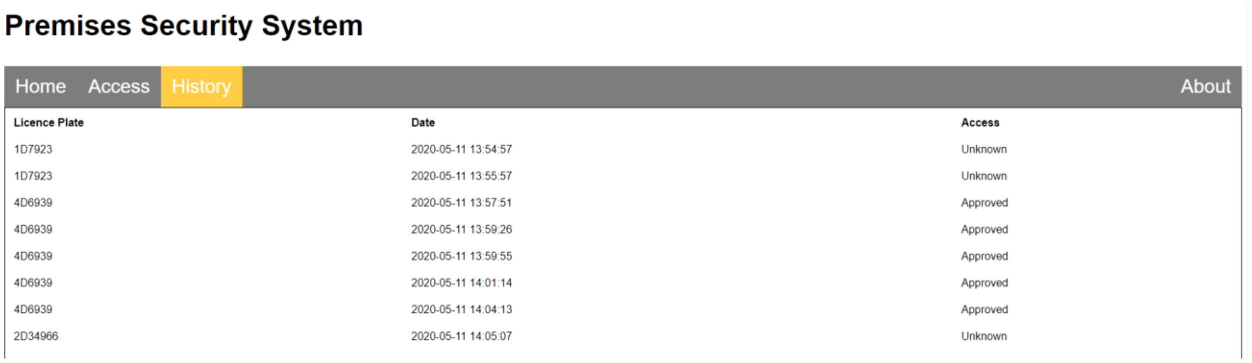
To connect to my remote database, I had to make a external php file with the connection details. This is to prevent anyone from getting the access information to my database. This php file contains variables for the host IP, username, password and database name. It also creates the connection to the database. This means the only code that that is in the page relating to the database is the prepared statement for selecting data.



```

<?php

require_once('../mysqli_connect.php');

$query = "SELECT licenceplate, accesstime, accessgranted FROM plates";

$response = @mysqli_query($dbc, $query);

if($response){
        echo '<table align="left" cellspacing="5" cellpadding="8">
                        <thead>
                                <tr>
                                        <td align="left"><b>Licence Plate</b><td>
                                        <td align="left"><b>Date</b><td>
                                        <td align="left"><b>Access</b><td>
                                </tr>
                        </thead>
                        <tbody>';

                while($row = mysqli_fetch_array($response)){
                        echo '<tr>
                                        <td align="left">' . $row[licenceplate] . '</td>
                                        <td align="left">' . $row[accesstime] . '</td>
                                        <td align="left">' . $row[accessgranted] . '</td>
                                </tr>';
                }

        echo '</tbody>
        </table>';

} else{
        echo "Couldn't issue database query";

        echo mysqli_erro($dbc);
}
mysql_close($dbc);


```

In the above php code you can see that the database is queried using a Select statement and returns an array containing a dictionary of all requested values. This each value is assigned a different cell in table by using the index of the array (which is a row in the database table) by using the dictionary id.

This displays a table like this:



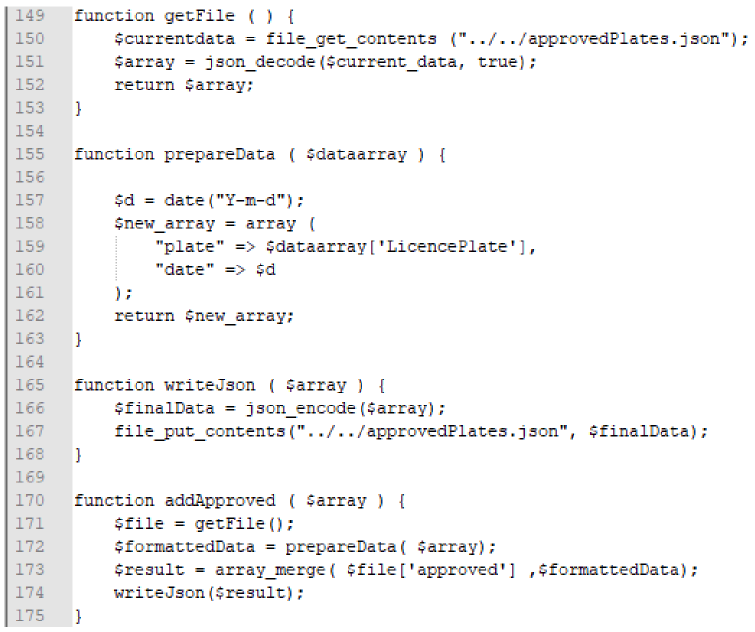
Licence Plate	Date	Access
1D7923	2020-05-11 13:54:57	Unknown
1D7923	2020-05-11 13:55:57	Unknown
4D6939	2020-05-11 13:57:51	Approved
4D6939	2020-05-11 13:59:26	Approved
4D6939	2020-05-11 13:59:55	Approved
4D6939	2020-05-11 14:01:14	Approved
4D6939	2020-05-11 14:04:13	Approved
2D34966	2020-05-11 14:05:07	Unknown

4.2.2 Appending to a Json File

This task was much more difficult than I originally had thought it would be. I used a JavaScript Object Notation (Json File) to store the approved plates and another Json file to store the blocked plates. Originally the plan for this was to store the plates locally on the ALPR system so that if the network connection was to go down, the system could still function with out error. After researching this approach to the problem, I realised that using a database would have been the better option.

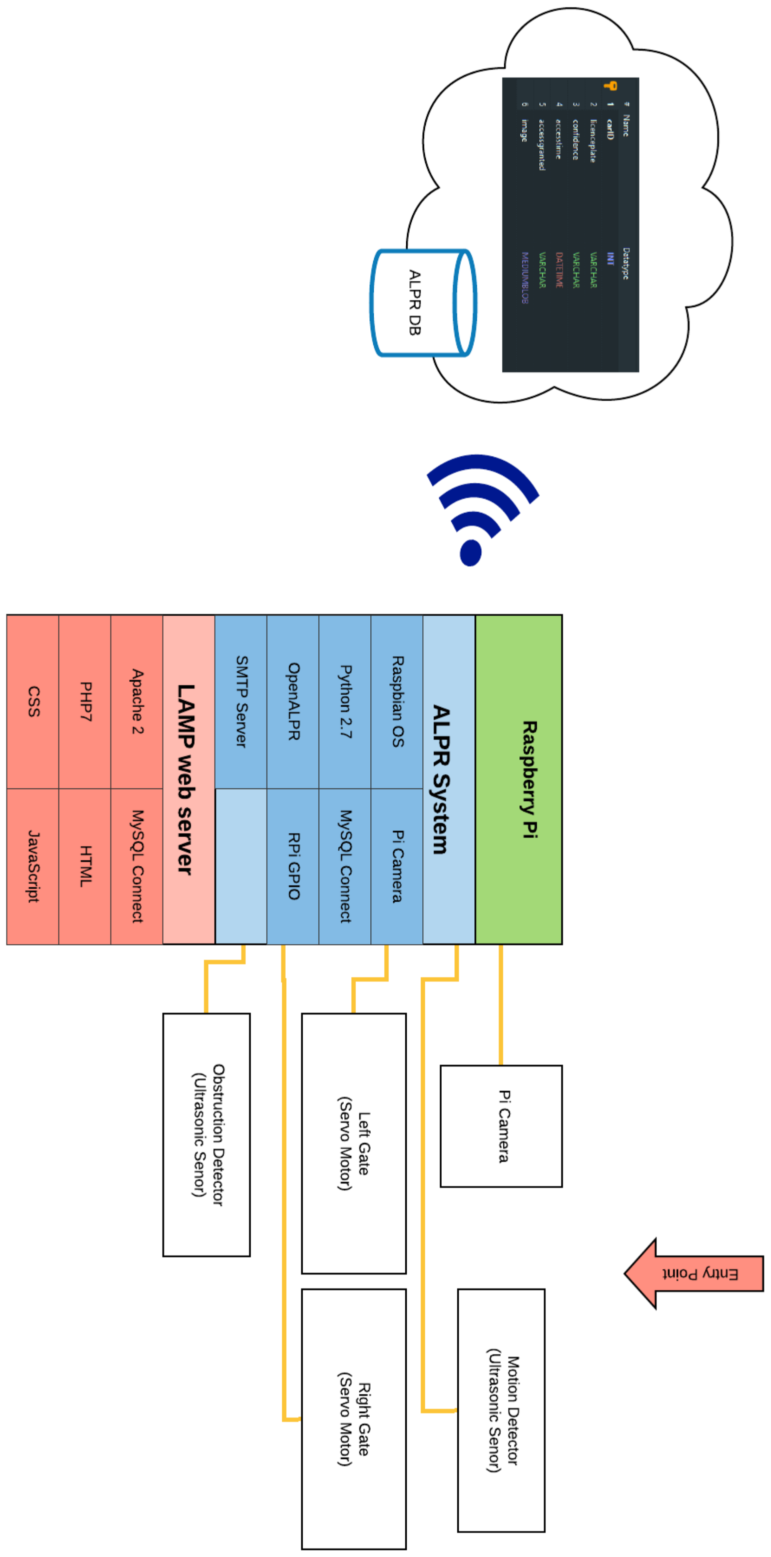
The functions on the right are for:

- Reading a Json file into an array
- Formatting the new data to match the Json format
- Appending the new data array to the contents of the Json file and writing it back



```php
function getFile ( ) {
    $currentdata = file_get_contents ("../../approvedPlates.json");
    $array = json_decode($current_data, true);
    return $array;
}

function prepareData ( $dataarray ) {

    $d = date("Y-m-d");
    $new_array = array (
        "plate" => $dataarray['LicencePlate'],
        "date" => $d
    );
    return $new_array;
}

function writeJson ( $array ) {
    $finalData = json_encode($array);
    file_put_contents("../../approvedPlates.json", $finalData);
}

function addApproved ( $array ) {
    $file = getFile();
    $formattedData = prepareData( $array);
    $result = array_merge( $file['approved'] ,$formattedData);
    writeJson($result);
}
```

5 Project Architecture



#	Name	Datatype
1	carID	INT
2	licenceplate	VARCHAR
3	confidence	VARCHAR
4	accesstime	DATETIME
5	accessgranted	VARCHAR
6	image	MEDIUMBLOB

5.1 Tools and Software Used for Development

I used the following software and tools for the development of my project.

- **PyCharm Ultimate IDE** – I used PyCharm IDE for coding some of the python scripts on my laptop. PyCharm Ultimate is a professional IDE for python coding created by JetBrians. This software is easily used for script, testing and debugging python programs.
- **MobaXTerm** – MobaXTerm is an advance terminal software for Windows with a tabbed shh client. This made it easy for working on my Raspberry Pi for navigation through the complex file structure.
- **VNC Viewer** – This software allows me to view the screen of my Raspberry Pi through a VNC server I set up on the Pi.
- **PHPMyAdmin** – PhpMyAdmin is a useful software tool for administrative work on the MySQL Server. I used this tool to monitor my MySQL server remotely.
- **HeidiSQL** – HeidiSQL is a useful MySQL client that lets you easily alter contents of a database as well as run sql commands in with SQL error checking
- **Solid Works** – Solid Works is a computer aided design software that I am familiar with. I used it to prototype models used in my project demonstration
- **Cura** – Cura is a slicing software used to render 3d models into GCode which is the code used for 3D printing
- **NotePad++** - I used NotePad++ to build my website. It is a simple text editor with autofill abilities when coding in HTML, CSS and JavaScript
- **ThonnyIDE** – ThonnyIDE is a light weight development software that allows the user to edit scripts on the Raspberry Pi without being resource intensive
- **Linux Terminal** – Most of the work I have done for this project has been controlled through the linux terminal. This was to both familiarise myself with command and to learn more about the capabilities of the powerful interface.
- **KiCad -** A Cross Platform and Open Source Electronics Design Automation Suite that I used to design the pcb for the keypad

6 Conclusion

This project posed many challenges to me. It took a lot of my problem solving skills, time management and most importantly my organisation. Thankfully all my hard work paid off.

Overall, I am very happy with the outcome of my project. The system function well and I achieved the main goals of my project. There is still a lot more I can improve on, new features I would like to add and most importantly improve the overall security of the project. I will carry on my work of this project after college as the concept still intrigues and challenges me.

7 References

Some example references are given here. For example, the section in your report on your servo motors should have a sentence that includes [1] in it, referring to reference [1] here.

[1] eFlow, “Dublin’s M50 Toll”, https://youtu.be/aKP3PVRtS1k, 2017

[2] OpenCV, “OpenCV - About”, https://opencv.org/about/, 2020

[3] OpenALPR, “ALPR Python Documentation”, http://doc.openalpr.com/opensource.html?highlight=python, 2020

[4] Raspberry Pi Foundation, “Raspberry Pi 4 Model B”, https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf, 2019

[5] Stack Overflow, “Is there a way to have more than 14 output pins on Arduino?”, stackoverflow.com, 2014.

[6] IBM, “What is the LAMP Stack”, https://www.ibm.com/cloud/learn/lamp-stack-explained, 2019.

[7] Nvidia, “Jetson TX2 SOM”, https://download.kamami.pl/p569306-DATA%20SHEET%20-%20NVIDIA%20Jetson%20TX2%20System-on-Module.pdf, 2015

[8] ElecFreaks, “Ultrasonic Ranging Module HC-SR04 datasheet”, https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf, 2011

[9] Raspberry Pi Foundation, “Raspberry Pi Camera V2”, https://www.raspberrypi.org/documentation/hardware/camera/, 2016

[10] Motion, ‘’Documentation”, https://motion-project.github.io/motion_guide.html, 2020

[11] Jameco Electronics, “How Do Servos Work?”, https://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html, Unknown

[12] OpenALPR, “Linux Installation”, http://doc.openalpr.com/opensource.html#linux, 2020

[13] Leptonica, “Applications of Leptonica”, http://www.leptonica.org/, 2020

[14] Tesseract - OCR, “About Tesseract Optical Character Recognition”, 2020

[15] OpenALPR, “Train OCR”, https://github.com/openalpr/train-ocr , 2016

[16] Wikipedia, “MySQL”, https://en.wikipedia.org/wiki/MySQL, Unknown

[17] GeeksforGeeks, “Simple Mail Transfer Protocol”, https://www.geeksforgeeks.org/simple-mail-transfer-protocol-smtp/, 2019

[18] PHP, “What is PHP?”, https://www.php.net/manual/en/intro-whatis.php, 2015