Smart Car Parking

Management System

Final Year Project Report

GSN: Fall 21-16

Group Members

Abdur Raheem(TL)	18-1269
Mohammad Hassaan Mumtaz Ali	18-1292
Syeda Maarij Hassan	18-1326

Advisor Dr. Saima Zafar

Client Dr. Saima Zafar

24th June, 2022

Department of Electrical Engineering
National University of Computer and Emerging Sciences, Lahore

CERTIFICATIONS

This document has been prepared by all of us together and we take joint ownership of its contents. We have provided references to the material consulted in preparing this document and, to the best of our knowledge, have not plagiarized anything.

Abdur Raheem, 18-1269 _____ Date: _____

Mohammad Hassan Mumtaz Ali, 18-1292 _____ Date: _____

Syeda Maarij Hassan, 18-1326 _____ Date: _____

I am the client of the product proposed in this document and the product specifications and other details are according to my requirements.

Client:

Dr.Saima Zafar _____ Date: _____

The final year project proposal in this document is being submitted to the department of Electrical Engineering with my approval.

Advisor:

Dr. Saima Zafar. _____ Date: _____

Head of Department:

Dr. Saima Zafar _____ Date: _____

Abstract

The infrastructure of our country can be improved with the incorporation of technology. Traffic congestion is one of the biggest challenges our country is facing currently. Especially in commercial areas, traffic can be a serious issue that not only compromises security but also contributes to pollution and time wastage. These shortcomings all together pave way for accidents, traffic jams, and long waiting hours. Smart Car Parking Management system is an IoT-based model which detects the availability of an empty parking slot in a parking lot and displays it to the user where he can directly go and park the car. The system uses sensory inputs, which consists of an Internet of Things (IoT) based wireless sensor node, where a real-time parking map is displayed which can be accessed by the user through a mobile or web application. There are reserved slots for differently-abled persons as well which can be seen on the digital map. further features of the management system include authorization of vehicles to enter the parking slot through image processing of the car license plate. The information of each person/vehicle is stored in the database which is checked and entry is allowed accordingly. This system is designed for private buildings and institutes due to the authorization feature but can also be used for public buildings with a few modifications. The project aims to counter problems faced by drivers on a daily basis especially in larger parking slots. It will enable the drivers to identify an available parking slot and directly head towards it without manually searching for one and hence, saving their time and decreasing the risk of traffic congestion.

Keywords – IoT, Real-Time Updating, Image Processing, Car Location Monitoring, Barrier System, Firebase, CRUD, Real-Time Parking Map.

Table of Contents

List of Figures

List of Tables

Acknowledgements

We would like to thank our advisor, *Dr. Saima Zafar*, whose support and encouragement paved the way for us to complete our project successfully. Her guidance directed us in the right direction which kept us focused and ambitious towards our project. In addition, we stand in gratitude to our evaluation committee members *Ms. Akbare Yaqoob* and *Ms. Sara Kiran* for providing us with valuable insights to further enhance the project.

Chapter 1: Introduction

Smart Car Parking Management System is an IoT-based management system to identify which parking area is occupied or available, dependent on sensory inputs to design a real-time parking map. The map will be available to the driver through a mobile application and the web before entering the parking lot, thus avoiding traffic congestion and time wastage. In each parking slot, there will be sensors that will detect cars' movement and update it on the digital map. The project is designed according to the university parking. Hence, it will also keep track of authorized vehicles entering the parking lot and notify each person's payment. The system makes it easier to detect empty parking spots and provides security features to allow access to only authorized vehicles. This document discusses the features of the project in detail.

Chapter 2 focuses on the Problem Definition stating all requirements by the client and record of meetings. It also includes problem formulation, specifications, and expected functionality. Moving forward, Chapter 3 covers Problem Analysis which discusses the problem in detail, analyzes the solution according to society and environment, engineering problem model, and distinguishing features of the project. Following this, Chapter 4 discusses Design and Implementation and Chapter 5 includes Investigation and Testing of the project. Finally, before we Conclude our Project, there is a User Guide for users to easily interpret and make use of the Smart Car Parking System.

Chapter 2: Problem Definition (Client Requirements)

After meetings with the client, the requirements of the parking system are as follows:

1. Real-time update on an available parking spot
2. Digital parking map on mobile application
3. Reserved parking spots for the handicapped
4. Authorization and barrier system
5. Data Insights

2.1 Problem Formulation

Due to an increase in cars and everyday tasks, parking smartly is a huge task. From searching for a suitable empty slot to trying to save time, everyone looks for a solution to this problem. On top of increased traffic, most of the time is spent looking for an empty parking slot among already filled slots. It also impacts pollution in CO2 emissions, noise, and other pollutants. Another issue is that it greatly increases management costs, as, in conventional times, on-street parking may have required investments in parking meters or parking inspectors. Smart Parking technology can reduce these overheads by automated processes and by providing targeted enforcement activity. When a driver knows exactly where they need to go, it reduces idling and unnecessary driving, leading to time management, increased efficiency, and reduced pollution and management costs. Our client also faces a similar problem and would like to meet these goals.

2.2 Record of Meetings with Client

Sr. No	Date of Meeting	Progress	Meeting Agenda
1.	17-09-21	Introduction to Project Idea, problems, aims, and tasks	Discussed the basic rundowns of what our project was and highlight the different solutions this project caters to
2.	8-10-21	Refine execution strategy	Projected a possible solution with regards to the estimated cost and our overall plan with regards to selection of components and solution
3.	29-10-21	Progress on the initial design	Produced possible solutions to our 4 key Client Requirements.
4.	12-11-21	Progress on design further refining hardware and software components	Generated a Work-Breakdown Structure for division of tasks on basis of Hardware and Software side.
5.	26-11-21	Progress on design and starting draft writing of FYP I report	Provided possible frameworks for Mobile and Web Applications, possible Databases, and Image Processing
6.	10-12-21	Progress on design and sharing initial FYP I report	Comparison between pure-IoT(ThingSpeak) and Cloud Computing platforms, Hardware Devices and various Microprocessors.
7.	24-12-21	Final Presentation of FYP I	Selection of appropriate components and reason as to disregarding others.
8.	25-02-22	Discussed Image Progressing Algorithms	Discussed various Image Processing techniques and selected the one with the least time and space complexity.
9.	11-03-22	Demonstrated the working of ultrasonic sensors with NodeMCU	Connected NodeMCU to Real-Time Database and showed updation of data
10.	01-04-22	Demonstrated the Image Processing Algorithm on	Implemented the Image Processing Algorithm from

		Raspberry Pi	Meeting#8 on Raspberry Pi.
11.	15-04-22	Presented the Front End of the Web App	Showed updation and deletion of cars from parking areas on Web Application using manual updation
12.	29-04-22	Demonstrated the Interfacing of node mcu with realtime database and web application	Showed updation and deletion of cars from parking areas on Web Application using sensor based updation
13.	13-05-22	Demonstrated the Data collected and the mobile app	Showed the collected dataset of people that entered the parking area from 23th to 27th of May, and the running mobile application
14.	20-05-22	Final Presentation and Demonstration of FYP II	Showed complete implementation of project.

Table 2.1, Record of Meeting with Client

2.3 Preliminary Product Specification

The Product that we would be creating would contain: -

1. HC-SR04 ultrasonic sensor which operates at a voltage of 5V and 15mA current. This ultrasonic sensor can detect an object 2cm - 4m distance away from it.
2. Raspberry Pi 4 which is a high-performance 64-bit quad-core processor and would be having 4GB of ram and also operates at a voltage of 5V.
3. Pi camera v2 is a high-quality 8-megapixel camera based around the Sony IMX219 image sensor. It's capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p90 video.
4. Tower Pro SG 90 servo motor which has an operating speed is 0.1s/60° and a torque of 2.5kg/cm. It also operated at a voltage of 5V.
5. IR sensor which has an effective distance range of 2cm to 80cm and works on 3.3 to 5V voltage. A preset potentiometer is also attached to this sensor which is used to fine-tune distance range.

2.4 Expected Functionality of Product

The Raspberry Pi Camera will take a photograph after sensing the availability of a car, based on the input of Infrared Sensors attached at the entrance. The Raspberry Pi will process the input and based on the data in a database, either allow or disallow the car entry through a barrier system engaged using a Servo Motor interfaced with our Raspberry Pi. The user then chooses an available parking spot using the mobile application or alternatively the web application on the attached display panel, after which he/she parks their car in any of the available parking spaces. This results in:

1. Increased efficiency in identifying parking spots
2. Decreased traffic congestion
3. Decreased noise and air pollution.
4. Optimized time and space for users in the relocation of parking spots in cases of large parking areas.

Chapter 3: Problem Analysis

3.1 Engineering Problem Model

After looking at multiple solutions from different resources regarding a management model that solves our problem and is fast and cost-efficient. Some of the solutions that we looked into are as follows:

1. The first solution that we looked at solved the problem by the use of computer vision. In this solution, each parking spot, through machine learning and neural networks algorithms, would detect if a parking spot was free or occupied. This information was then stored in the database, and the database would show the output of this information on the mobile app. The drawback of this solution was regarding the accuracy of this model. Any inaccuracy in the model could play a huge impact on the overall user experience.
2. Another solution that we looked at used sensory inputs in order to detect the status of a parking spot. The sensors were connected to the microcontroller and had Led lights attached to the parking spot. The color of the led light would change if the parking spot was free. This solution successfully showed the availability of the parking spot but a user would not be able to know if a parking spot is available or not until he/she is near that parking spot.

3.2 Recent Similar Projects

The basis of communication from sensors to the server in most systems is the integration of the Application Layer, which is essential for the core processing of information and dissemination of data to end-to-end users. Most systems contain registration of cars, storage of data to a database, administration of IoT-based smart parking systems [2], an interface to the cloud-based services,

and even prediction systems that process the allocation and the reservation of parking spaces [3]. Moreover, the user interface, either based on Web or Mobile Application, should act on back-end data processing to create solutions to the problems addressed before.

Tsai and Pham[2] is primarily based on GPS systems that wirelessly access and reserve places, which update the cloud, using web services based on RFID technologies to manage the availability, placement, and the number of cars in specific parking lots.

Kotb and Shem[3] proposed a system developed on the basis of ZigBee technology, which is the development of the information of other servers through a gateway. The server is a subsequent result of the updates on the database.

Fraifer and Fernström[4] use a CCTV delivered feed to perform a Computer Vision (CV) algorithm to maximize detection of free spaces. The drawbacks of such a system are that regardless of it being fast, they don't take into account digital mapping onto a database which can be inefficient. The procedure of using such a system is that it accesses scattered parking spots through the use of databases to access smart parking locations for users.

Our web-based model differentiates in such a way that it uses sensors that update the digital parking map through interaction with an MCU. The MCU will further be connected to the database to display occupied and free parking spots in real-time. Having a web-based model will allow the authorized users to access the map remotely.

3.3 Distinguishing Features of this Project

There are a number of important distinguishing features that alleviate this project from the ones elaborated on in *Section 3.2*:

1. A fully automated map accessible on both web and mobile devices.
2. A fully automated barrier system that opens and closes based on the access provided by the Raspberry Pi
3. Specific parking spots with ease of availability for handicapped persons.
4. Access is provided based on inputs from Raspberry Pi using Image Processing as compared to Fraifer and Fernström[4] and Tsai and Pham[2] using CCTV cameras integrated with high-level Deep Learning Algorithms and RFID technology, respectively.
5. Providing insights using data visualization to improve the technical and structural features of the parking area, and to look into future enhancements.

3.4 Societal and Environmental Implications of the Project

Smart parking can significantly decrease air and noise pollution levels through sustainable urban mobility, reducing traffic problems, saving time and energy, and bringing air and noise pollution down. By the use of sensors, the availability of parking slots is detected, therefore eliminating the time drivers waste on searching for parking. By directing drivers to open spaces quickly, smart parking can reduce congestion and resultant emissions. [7]

Through the data collected, smart parking systems can also enable city administrations to access crucial data to pave the way for cleaner and safer environments.

Furthermore, smart parking technology can reduce traditional management and meter costs by automated processes and providing targeted enforcement activity.

Through features like duration notification in the technology, staff can be alerted about the drivers who have overstayed in the parking lot and take the required action. The technology will have a significant impact on security as well by reducing accidents; drivers can maintain their attention rather than browsing for spaces or making rash maneuvers. [8]

Chapter 4: Design and Implementation

4.1 Design Requirements and Constraints:

The method of finding cost and time-effective implementations to the problems encountered by the client were based on a number of factors:

1. To provide a real-time update to the parking map, a high-frequency sensor with a small sampling rate needs to be attached, such that data is received to the server as soon as possible. The data is then processed based on the sensor's location and tends to provide access to the user.
2. The lack of digitization and visual representation, in multistorey let alone single floor parking areas, seemed like a problem with an inefficient solution. Hence, a fully automated parking map generated solely for that parking specific area, available both on phone and web with accessible and inaccessible locations, made solving this problem far easier.
3. The ability to have reserved areas for handicapped persons increased the ability to diversify students with injuries/disabilities, by giving them preference.
4.
    i. Authorization can be done using a Raspberry Pi Camera module which captures high-definition images, stores them in the file directory on its parent Raspberry Pi, processes the image, and then sends data to update the eligibility of the driver.
    ii. Using a servo motor module with the Raspberry can facilitate the barrier system to give access(0-degree upshift in barrier controller i.e., don't provide access, or 90-degree upshift in barrier controller i.e., provide access) to the parking area.

5. Having the ability to get free parking based on showing their FAST-NUCES card caused a huge downfall in maintenance costs for the parking area, as funds available from providing parking spots for students based on paid fees were unavailable, hence providing a system to generate profit from a time-sensitive system seems like a probable

solution.

6. Based on the data sets created and data visualization plots generated, we can provide insights into the number of cars present on a given day. The plots can also provide future guidelines to the expansion of the current area by increasing the total number of slots available for parking through extensive Machine Learning models.

However, due to certain limitations in lieu of weather, time constraints, and design flaws, there could be delays, significantly in:

1. Image processing of license plates might be affected in poor weather conditions.
2. There is a risk of mechanical damage to the ultrasonic sensing element.
3. Breakdown of a component may deny entry of authorized vehicles into the parking lot.
4. For many new users who are not familiar with the parking management system, it could be a little confusing for them to operate at first.
5. Installation of systems without the knowledge of building structures may lead to dangers and irregularities.
6. Parking slots cannot be reserved before time.

Hence, having readily available solutions in place beforehand seems like a viable option:

1. Installation of cameras in open space with suitable lighting for the camera to detect it.
2. In case of damage to a component, it is replaceable through weekly maintenance checks of the system.
3. Use of more than one sensing principle to cater to environmental issues and increase accuracy. For example, using an IR Sensor in a controlled environment was of high importance, as light rays can continuously detect unwarranted changes in light intensity if resistance settings aren’t done accurately.
4. A contract can be drawn with the supplier or manufacturer to ensure that it is done without hindrance to regular activities and that the cost it incurs is low.
5. For new users, a manual can be provided or a set of instructions so that they can know how to proceed.
6. While making the system, it is important to know the area where it is going to be installed and make a map to ensure all areas are covered.

4.2 Preliminary Design

The initial design consists of a Raspberry Pi microprocessor, a microcontroller NodeMCU integrated with the ESP8266 WiFi-chip, Ultrasonic Sensors, Infra-red sensors, a Servo Motor, and a Raspberry Pi Camera. The work flow is as follows:

1. The microprocessor detect change in IR Sensor,
2. The microprocessor snaps a picture of the object(assuming only a car can enter the area),
3. The microprocessor processes the image with a list of allowed users in a database,
4. The microprocessor opens the gate, if the person is allowed entry,
5. The person parks in an empty spot, which updates the microcontroller using Ultrasonic Sensors ,
6. The microcontroller updates the parking map on both web and mobile applications.
7. Upon exit the microprocessor detects change in another Ultrasonic Sensor to allow the user to exit.

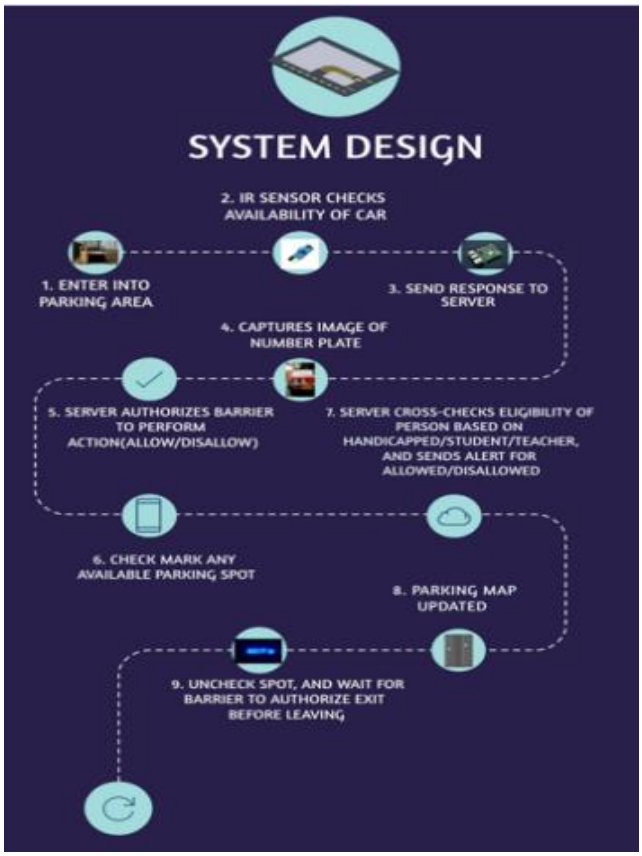


Figure 4.1, System Design Chart

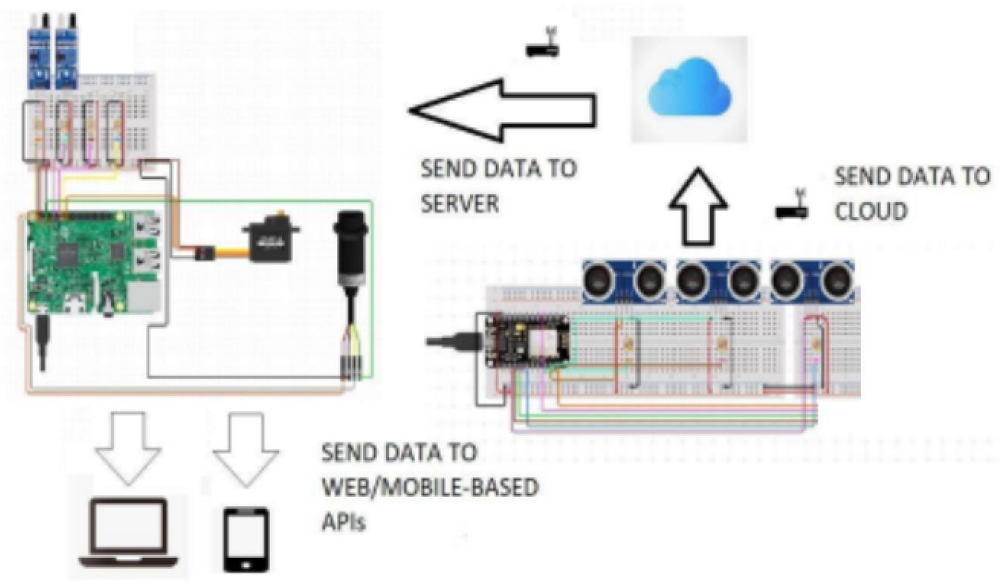


Figure 4.2, Prototype Design Implementation

4.2.1 Hardware Block Diagram

The hardware block diagram shows the integration of the sensors with not only the Microcontroller(NodeMCU V3) but also the microprocessor(Raspberry Pi V4). The power supply provides power to them, which then in turn provides the necessary power to their own respective components.

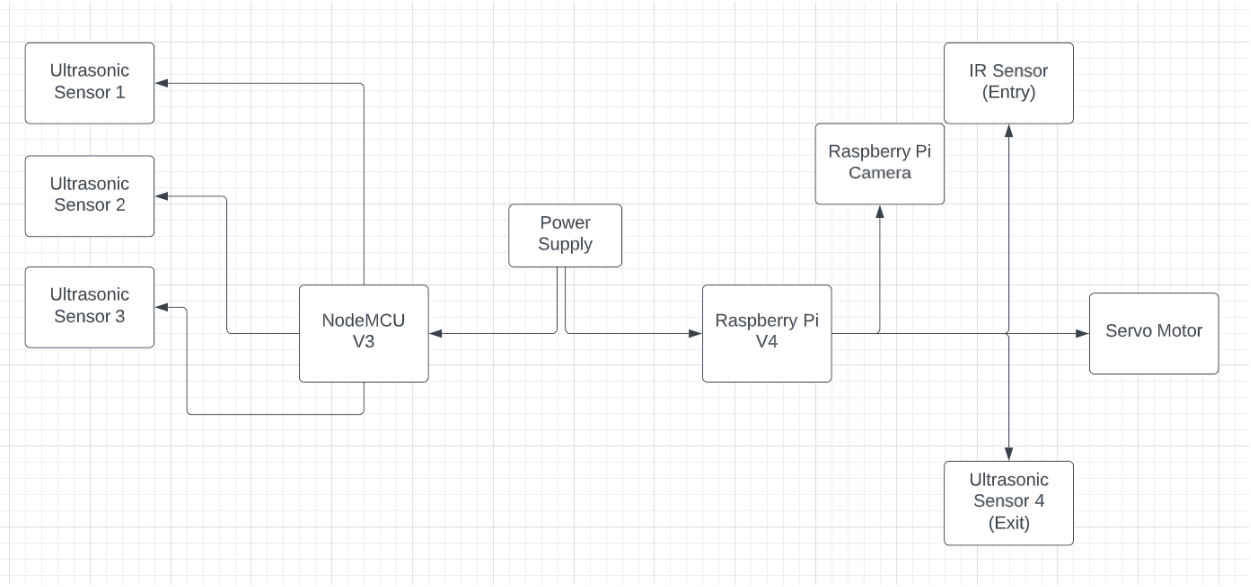


Figure 4.3, Hardware Block Diagram

4.2.2 Software Block Diagram

The software block diagram shows the software components of our project. The NodeMCU receives signals from the Ultrasonic Sensors attached to it and sends data to the Firebase Real-Time Database, whereas the Raspberry Pi V4 interacts with the Cloud Firestore. The Cloud Firestore then in turn interacts with the Refi App. The Firestore and Real-Time Database then interact with the Mobile and Web Application to provide users with availability of parking spots.

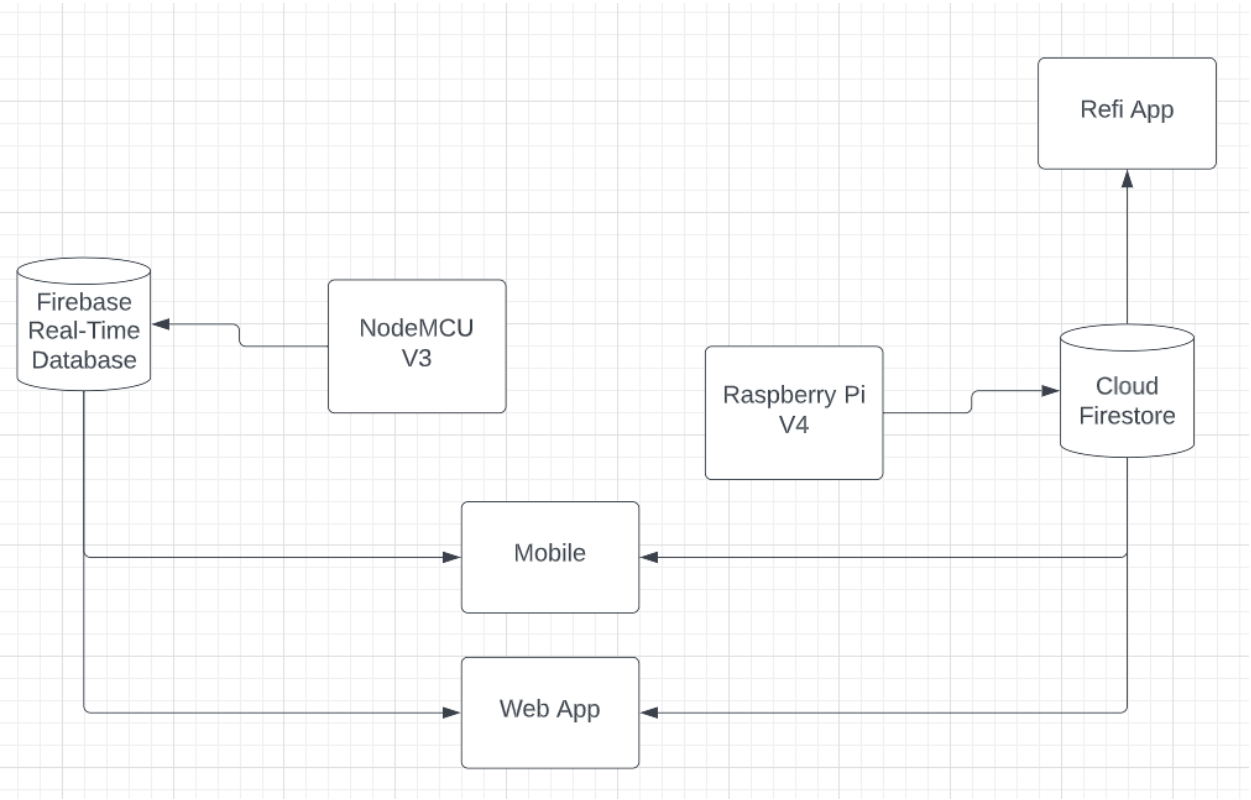


Figure 4.4, Software Block Diagram

4.3 - Detailed Hardware and Software Design

The hardware and software design is as follows:

4.3.1 Hardware Design

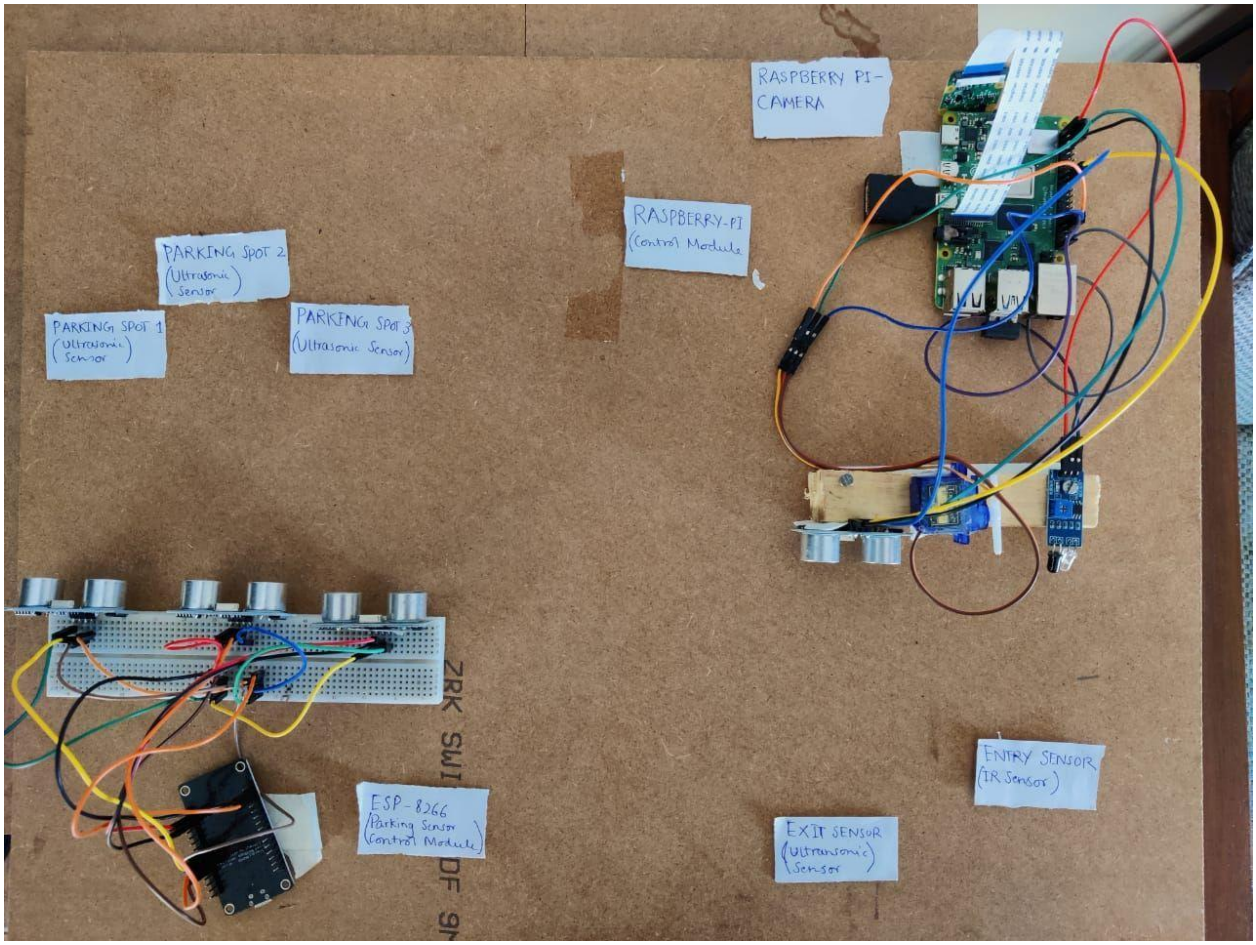


Figure 4.5, Hardware Diagram

4.3.1.1 Hardware Components

4.3.1.1.1 Ultrasonic Sensor

The ultrasonic sensor[9] operates at 5V DC Supply with an operating current of 15mA. The operating frequency of this distance sensor is 40Hz, which is twice the maximum hearing frequency of an average human. A generic ultrasonic sensor is provided with a Vcc(5V) supply, a Trigger pin to trigger these ultrasonic pulses, an Echo pin to read back these pulses in a specific duration of time, and Ground.

Functionality:

Upon receiving a 10µs to the trigger pin pulse when an input is applied to the digital input i.e.

Trigger pin, the sensor then transmits eight sonic pulses at 40KHz. The echo pin then generates a high-level pulse with a maximum duration of 38ms, in case the signal is not received back. In case there is an echo back the echo pin goes back to LOW with a time of 150µs to 20ms indicating the time it took for the pulses to bounce back from the object.

The echoed back signal gives the duration of this callback which is then used to calculate the distance of the object from the sensor.

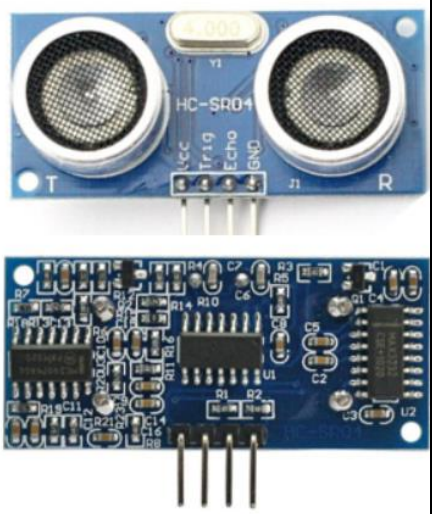
Type of components	Ultrasonic Sensor	
Model no.	HC-SR04	
Operating voltage	DC 5V	
Operating current	15 mA	
Operating Frequency	40Hz	
Range (Minimum)	2cm	
Range (Maximum)	4m	
Resolution	~0.3cm	
Measuring Angle	30°	
Advantages	● Not affected by color or transparency of objects. ● Greater accuracy than many other methods at measuring thickness and distance to a parallel surface. ● Their high frequency, sensitivity, and penetrating power make it easy to detect external or deep objects.	

Table 4.1, Ultrasonic Sensor Specifications

4.3.1.1.2 NodeMCU WiFi Module

The NodeMCU ESP8266[10] microcontroller is a low-cost yet very efficient device for IoT applications with a built-in Wifi Module and 17 GPIO pins to connect sensors, with the ability to connect multiple digital/analog-based devices to the internet by sending JSON based data

through HTTP

Functionality:

The NodeMCU interacts with different sensors to connect them for further processing either locally using Arduino IDE or using a remote client to depict data on a local database, mainly using HTTP WiFi protocol. It provides capability to interact with devices based on PWM, ADC, or basic GPIO interfacing.

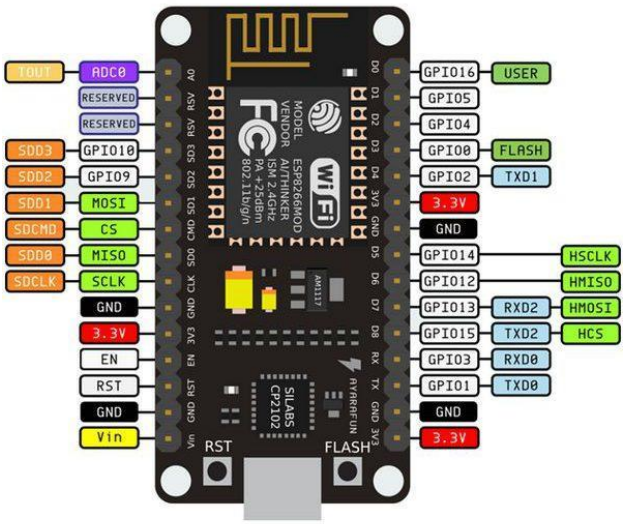
Type of Module	**Microcontroller**	
Model no.	ESP 8266 NodeMCU WiFi	
Operating voltage	DC 3V to 5V	
Working current	10 uA to 130mA	
Microprocessor	ESP-8266	
WiFi Signal Range	<100m	
Network Protocols	IEEE 802.11 b/g/n, TCP/IP Protocol stack	
Working temperature	-25°C to 75°C	
Advantages	1. Low Energy Consumption 2. Integrated WiFi	

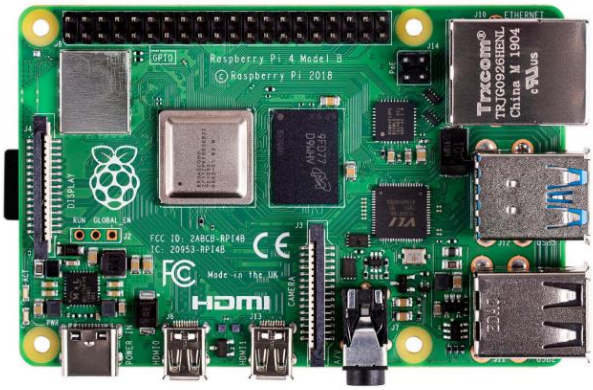
Table 4.2, NodeMCU Specifications

4.3.1.1.3 Raspberry Pi v4

Raspberry Pi 4[11] is a high-performance 64-bit quad-core processor and has 4GB of ram and also operates at a voltage of 5V.

Functionality:

The Raspberry Pi can provide an interface to connect to it as a Desktop PC capable of running various web applications, interfacing GPIO, PWM, ADC sensors to interface with its main interface, capable of sending and receiving data from remote cloud based servers.

Type of Module	**Microprocessor**	
Model no.	Raspberry Pi 4 Computer Model B	
Operating voltage	DC 5V	
Working current	3A	
Microprocessor	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	
Memory	4GB LPDDR4	
Network Connectivity	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet	

	2 × USB 3.0 ports 2 × USB 2.0 ports	
Working temperature	0°C to ~60°C	
Video & Sound	2 × micro HDMI ports (up to 4Kp60 supported)	
Advantages	1. Huge processing power in a compact board 2. Many interfaces (HDMI, multiple USB, Ethernet, onboard Wi-Fi and Bluetooth, many GPIOs, USB powered, etc.) 3. Supports Linux, Python 4. Readily available examples with community support	

Table 4.3, Raspberry Pi Specifications

4.3.1.1.4 Infrared Sensor

The B143 Infra-red sensor operates at 5V DC Supply with an operating current of 20mA. The maximum sensing distance of this sensor is 80cm which will enable us to detect an obstacle(Car) at the entrance. The layout of the sensor is such that there are only three pins i.e. VCC(5V), Ground and Digital Input.

Functionality:

The IR Sensor is capable of detecting objects at a distance of 80cm and provides users with the ability to use it as an object detection module, with the output ranging from a binary 1 or binary 0 to depict whether something is present or not. The IR sender sends out pulses which are then then received by the onboard Photodiode resistor. The photodiode resistor then forwards data to the LM358 Operation Amplifier that simply compares the voltage generated from the Photodiode resistor due to resistance drops and compares with preset threshold voltage(preset by calibrating the distance resistor). The onboard LED lights up the moment an object is

detected.

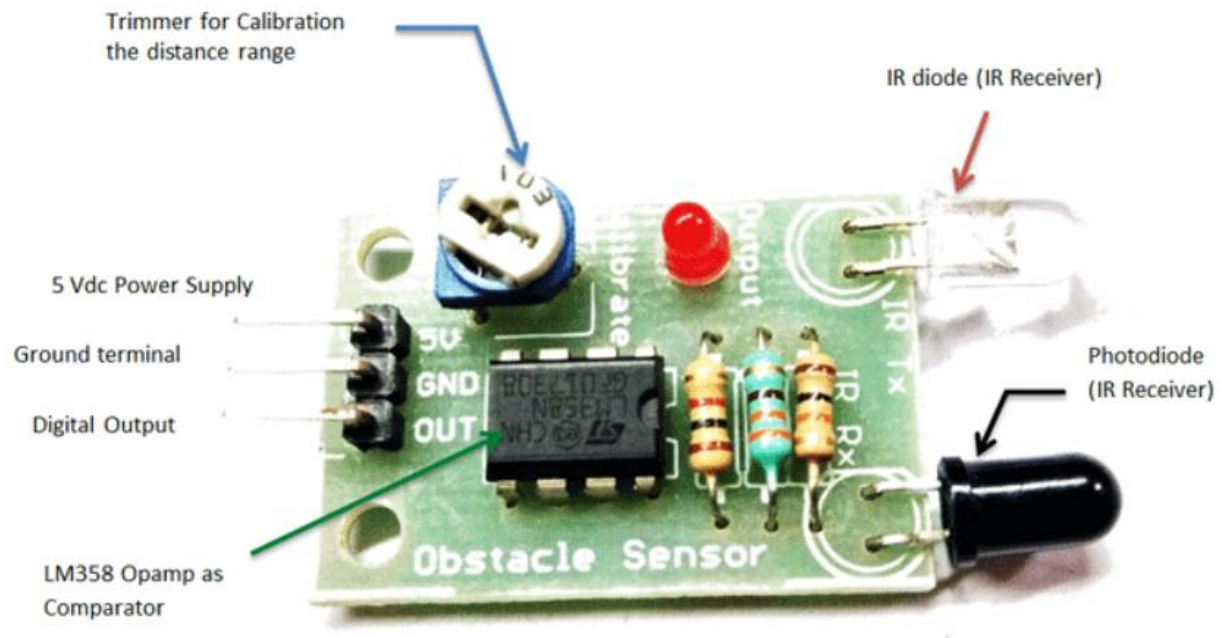


Figure 4.5(1), IR Sensor

4.3.1.1.5 Raspberry Pi Camera V2

Pi camera v2 is a high-quality 8-megapixel camera based around the Sony IMX219 image sensor. It's capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p90 video.

Functionality:

The camera uses less power, smaller physical size, faster bandwidth, and captures live images with higher resolutions, higher frame rates, and reduced latency of image captures by capturing images directly.

4.3.1.1.6 Servo Motor

Tower Pro SG 90 servo motor which has an operating speed is 0.1s/60° and a torque of 2.5kg/cm. It also operated at a voltage of 5V.

Functionality:

The function of the servo motor is to convert the control signal of the controller into the rotational angular displacement or angular velocity of the motor output shaft. This allows for use of a singular shaft to allow for entry or exit, to any area.

4.3.1.2 Hardware Calculations

Since this report describes an engineering project and product, there will be formulas and calculations having numbers with engineering units.

There will be calculations for choosing resistor, inductor and capacitor values, and to determine power dissipation to select appropriate transistors. There will be calculations to select proper diodes and relays and other electrical components. There may be calculations about the mechanical aspects of your product e.g. how much weight can that electric vehicle carry.

The preliminary calculations will be with respect to the sensors that need to be calibrated so that accurate measurements can be performed:

4.3.1.2.1 Ultrasonic Sensor

Distance Calculations:

The Ultrasonic Sensor takes the value produced by its Echo pin which calibrates the distance of the object from the sensor. To calculate the distance of the object through

$$v = d/t$$

v: speed of object

d:distance of object

t:time

Since ultrasonic pulses move at the speed of sound we calibrate this formula to be $d = 0.034 * t$

The resulting distance is in centimeters.

Position Calculations:

The Ultrasonic Sensor has to be placed in front of the car such that it can accurately senses when the car arrives. This required height is not an issue for our prototype system, but in the case of a full-fledged system, this height is measured out to be 7 inches high. However, in our prototype, this distance is set to be 1 inch.

4.3.1.2.2 Infra-Red Sensor

Position Calculations:
The average length of a prototype car is taken to be 10 inches long, so we intend to deploy our sensor at these distances. This takes into account that the distance from the Servo Motor to the IR Sensor is approximately 5 centimeters to accommodate any object in place, without interfering with the Servo Motor functionality.

4.3.1.2.3 Raspberry Pi Camera

Position Calculations:
The number goal of the Pi Camera was to capture images with high resolution and quality. To achieve this, we created a stand capable of holding our raspberry Pi camera module. The stand similar to one depicted in Fig.5 was created using plywood of length 32 inches and height 19 inches. This allowed us to place our Pi camera for stability purposes for capturing of images for future use.

Figure 4.6, Stand

4.3.1.3 Hardware Design

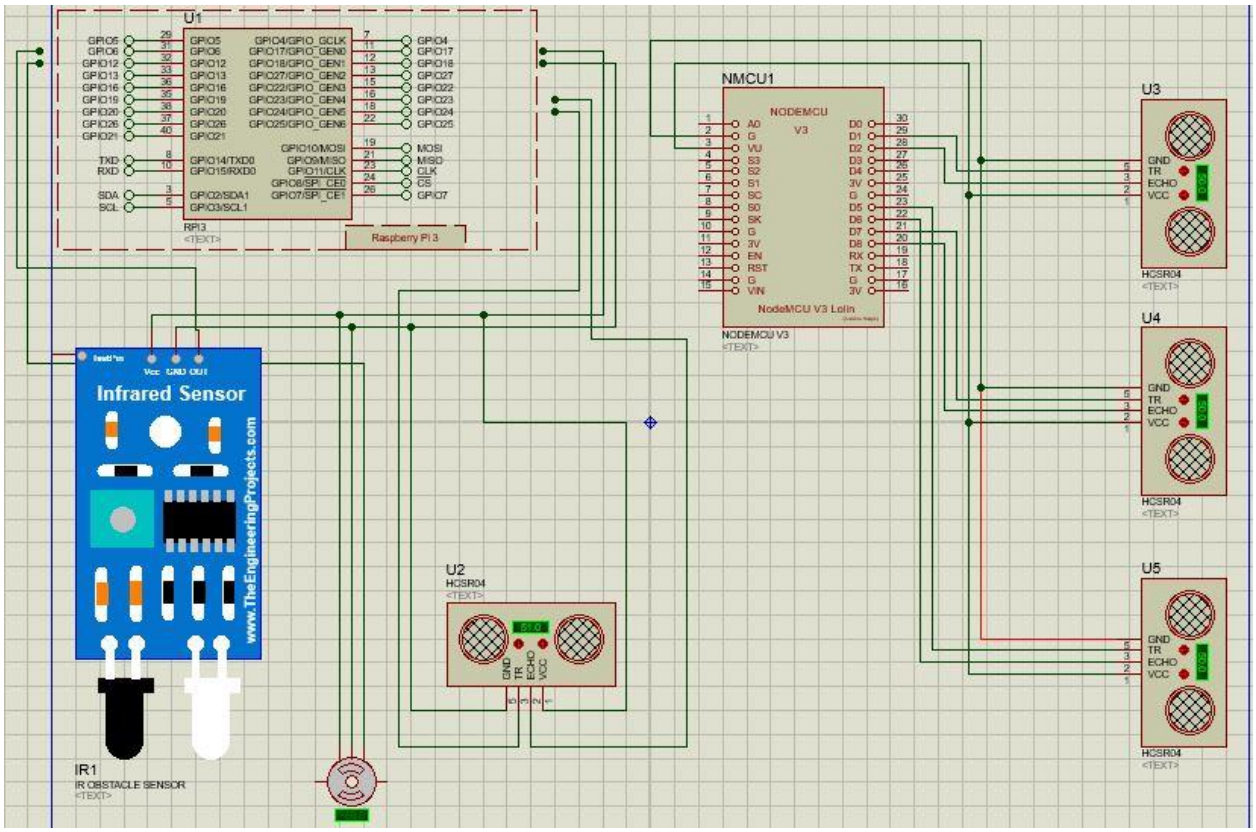


Figure 4.7,Hardware Schematic

The overall hardware design is segmented into a number of nodes depicted in Section 4.2.1 (Hardware Block Diagram).

4.3.1.3.1 NodeMCU V3 Implementation

The Microcontroller i.e. NodeMCU V3 is responsible for interacting with the Real-Time Database and only sends data extracted from the Ultrasonic Sensors based on position of the car. The Ultrasonic Sensor consists of 4 primary pins i.e. Ground(GND), Trigger(TR),Echo(ECHO),Power Input(VCC). Figure 6 extensively shows the common grounds and the common power input to properly operate the Ultrasonic Sensor with the Node MCU. The VCC is attached to Pin 3 (VU) of the NodeMCU and the GND is attached to Pin 2(G) of the NodeMCU. This allows for a steady 5V supply and a 0V ground to be attached to all three Ultrasonic Sensors. The TR and ECHO pin for each respective Ultrasonic Sensor is different as each one of them records different readings. The pins are then attached to the specific GPIO on the NodeMCU. For example, the pins of Ultrasonic Sensor 1 are attached at GPIO5(D1) and GPIO4(D2). This allows for smooth flow of readings from the Trigger pin to the Echo pin for effective time calculations.

```
const int trigPin3 = 5;  //D1
const int echoPin3 = 4;  //D2
```

Figure 4.8(a),Input Pins for Ultrasonic Sensor 1

```
long duration3;
int distance3;
bool car3;
```

Figure 4.8(b),Variables for Ultrasonic Sensor 1

```
pinMode(trigPin3, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin3, INPUT); // Sets the echoPin as an Input
```

Figure 4.9,Setting Input Pins for Ultrasonic Sensor 1

The Ultrasonic Sensor interfaced with the NodeMCU works in:

1. Set TRIG to LOW for 10µs
2. Set TRIG to HIGH for 10ms for Pulse to be Generated.
3. Set TRIG to LOW after 10ms have elapsed.
4. Read Pulse duration using *pulseIn* function of NodeMCU.
5. Calculate distance of object using formula from Section 4.3.1.2 ($d = 0.034 * t$)
6. Write *boolean:true* to "bool car3" in Figure 8 if distance is less than 10, otherwise write *false.*
7. Send this boolean value to the real time database.

```
// Clears the trigPin
digitalWrite(trigPin3, LOW);

delayMicroseconds(2);

// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin3, HIGH);


delayMicroseconds(10);
digitalWrite(trigPin3, LOW);
duration3 = pulseIn(echoPin3, HIGH);


// Calculating the distance
distance= duration*0.034/2;
```

Figure 4.10, Configuration of Ultrasonic Sensor

```
Firebase.setBool("Parking_Spots/Spot_3/availability", not(car3));
Firebase.setInt("Parking_Spots/Spot_3/index", 3);
```

Figure 4.11, Updating Data in Real Time Database

4.3.1.3.2 Raspberry Pi Hardware Implementation

The Raspberry Pi integrates both hardware and software fluently in a singular python script. This allows for a fluent workflow. Initially, we set up our Raspberry Pi to integrate the IR Sensor, Servo Motor and the Ultrasonic Sensor.

IR Sensor:

The initializations with regards to all three components are depicted in Figure 11. The IR Sensor works by detecting an object at a distance of 4cm as explained in Section 4.3.1.1.4. Upon detection of an object the IR Sensor triggers a set of commands to perform an array of tasks. Initially we perform our Image Processing algorithm that produces the number plate of the car, and upon successful comparison of allowed users in the database, discussed in Section 4.3.2.2.3, we move onto opening of the gate using the Servo Motor.

Servo Motor:

We first set our servo motor to its *mid* position that is approximated to be 45°. For 2 seconds before setting the value to *max* position for 1 second. This allows our gate to be open for a total of 3 seconds before it returns to its original position.

Ultrasonic Sensor:

The Ultrasonic Sensor works in a similar way as to the one described for the NodeMCU V3 Implementation section. However, the distance here is set to be a maximum of 4cm, i.e. the location of the car should be less than 4cm for us to execute a certain algorithm. The algorithm executed here is the same from the IR Sensor section, albeit being the Servo Motor is removed upon exit i.e. a person can only trigger the Ultrasonic Sensor to depict exit time upon exiting.

4.3.1.3.3 Power Supply

Regular phone chargers integrated with USB-C and USB port were used for powering Raspberry Pi and NodeMCU, respectively.

4.3.2 Software Design

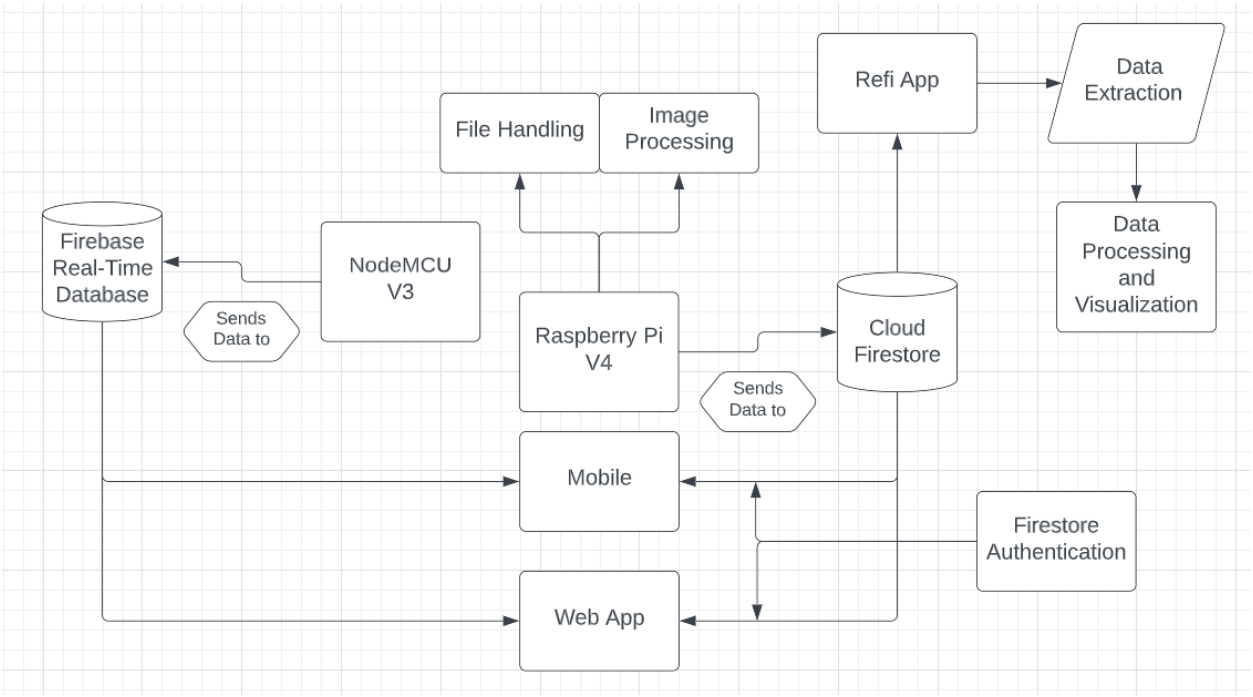


Figure 4.12, Complete Software Development Lifecycle

4.3.2.1 Software Components

1. NodeMCU V3:

The NodeMCU ESP8266 microcontroller is a low-cost yet very efficient device for IoT applications with a built-in Wifi Module and 17 GPIO pins to connect sensors, with the ability to connect multiple digital/analog-based devices to the internet by sending JSON based data through HTTP.

2. Raspberry Pi V4:

Raspberry Pi 4 is a high-performance 64-bit quad-core processor and would be having 4GB of ram and also operates at a voltage of 5V.

3. Firebase Real-Time Database:

Firebase Real-Time Database[12] is a NoSQL(document based) database hosted on the cloud that allows storing and retrieving data from any location in the world. Real-time synchronization of data allows for quick and efficient transfer of data to end users.

4. Cloud Firestore:

The Google Cloud Firestore[13] will act as the primary NoSQL database that lets us easily store, synchronize, and query data based on mobile and web app needs. The data that is sent

and received through the collections(alternative to tables in SQL databases) which acts as a container for the documents to organize the data and apply queries on them. This can be done by applying the CRUD Algorithm [22] to these collections.

5. Refi App:

Refi App is an open-source API that lets us interact with Cloud Firestore by allowing us to "Export JSON" or "Export CSV" documents for local use.

6. Jupyter Notebook:

Jupyter Notebooks[14] are documents produced for containing Python based documents for both human-readable documents containing analysis reports and dashboards for data analysis, and running python scripts.

7. Firestore Authentication:

Firebase Authentication provides backend services, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers etc. It can also use Google, Facebook and other popular identity providers to perform authentication.

8. Web Application:

A web application is application software that runs in a web browser, unlike software programs that run locally on the operating system of the device. Web applications are delivered on the World Wide Web to users with an active network connection.

9. Mobile Application:

A mobile application is the application software that runs on a mobile device. They can be run locally as well as using an active network connection according to the functionality required.

4.3.2.2 Software Design and Implementation

4.3.2.2.1 Node MCU V3 Software Implementation

Node MCU provides us with the capability to interface our GPIO inputs(Ultrasonic Sensors) to communicate directly with the Cloud Real-Time Database(RTDB). To do that we must first extensively understand the basis of how our Arduino IDE uses libraries we have attached to generate the required output.

Libraries:

```
#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>

// Set these to run example.
#define FIREBASE_HOST "dummy-dl8d5-default-rtdb.firebaseio.com"
#define FIREBASE_AUTH "GiuXtbp0vwY2pbjFU5nEGf6gLeegOouxpVLNJL4X"
#define WIFI_SSID "HassaanRouter"
#define WIFI_PASSWORD "12345678"
```

Figure 4.13, NodeMCU Libraries and Database Service Keys

The libraries that we primarily use with NodeMCU are the "ESP8266WiFi.h" that allows NodeMCU to send HTTP GET/POST requests to any location. In our case we define Firebase as the host end-point with the FIREBASE_HOST acting as the URL for RTDB and FIREBASE_AUTH acting as the authentication key for reading and writing data to it.

Implementation:

After configuring the required hardware as discussed earlier we get the *boolean* variable that we use to depict whether parking spots are available or not. A *true* value shows that a parking spot is available whereas a *false* value indicates that parking is not available.

```
// set bool value
Firebase.setBool("Parking_Spots/Spot_1/availability", not(car));
Firebase.setInt("Parking_Spots/Spot_1/index", 1);
```

Figure 4.14, Sending Data to Real-Time Database.

Figure 4.14, shows how we access document type NoSQL databases to access data in a specific format so that we can use it further in our web and mobile applications.

4.3.2.2.2 Real-Time Database Implementation

The final design of the NoSQL database is shown in Figure 4.15:

Figure 4.15, Format of Data on Real-Time Database.

The input data comes from the continuous monitoring of Ultrasonic Sensors causing continuous updates on RTDB. This allows for data to be sent and received by the Web/Mobile Applications depending on the speed of the internet, delay timing in code, and accuracy of the Ultrasonic Sensor. The delay of the code is set to be 1000ms, this allows for 1 second time to be added to the overall cycle, besides the extra time that is estimated between the maximum time for the last sensor to detect. For example, Ultrasonic Sensor 1 detects in 63µs, Ultrasonic Sensor 2 detects in 62µs, and Ultrasonic Sensor 3 detects in 113µs. This next to negligible time to the time for the 1s.

4.3.2.2.3 Raspberry Pi V4 Software Implementation

The Raspberry Pi provides users with the ability to interact with both GPIO Sensors for both Ultrasonic and Infra-red Sensors, along with the Raspberry Pi Camera. Due to the Raspberry Pi being a microprocessor both the hardware and software segment is deeply interwoven.

Libraries:

```
import RPi.GPIO as IO
import cv2
import numpy as np
import imutils
import pytesseract
import time
import os
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
from datetime import datetime
from gpiozero import Servo
from time import sleep
cred = credentials.Certificate("/home/pi/Desktop/Images_Data/dummy-d18d5-firebase-adminsdk-ickgv-9114c9a039.json")
firebase_admin.initialize_app(cred)
```

Figure 4.16, Libraries on Raspberry Pi.

The libraries attached show us the basic libraries that are needed in our overall implementation.

1. RPi.GPIO: It allows us to interact with the Ultrasonic and Infra-red Sensor.
2. cv2: It allows for efficient image processing using computer vision using OpenCV[15]. It can efficiently process images, read images, perform edge detection algorithms, perform gaussian blurring and even perform contour detection.
3. numpy: It provides users with the capability to interact with arrays and perform essential operations.
4. imutils[16]: It provides a series of functions to make basic image processing functions such as rotation,translating,resizing, contour sorting, edge detection.
5. pytesseract: It is primarily used for Optical Character Recognition(OCR), which allows for transformation of a two-dimensional image of text to readable text. Tesseract OCR[17] and its python equivalent pytesseract is downloaded directly onto the

Raspberry Pi to recognize an image using Long Short Term Memory(LSTM). LSTM is a Recurrent Neural Network(RNN) that works by:

i. Word finding organizes lines of text segmented into regions for proportionality in text.
ii. Line finding then breaks the words according to character spacing
iii. Recognition is done by passing recognized data to an adaptive classifier(changing in behavior based on input text line).

6. time: It is used in calculating real-time for the ECHO signal in Ultrasonic Sensor(similar to the one explained in Section 4.3.1.3.1.
7. firebase_admin: This allows us to use our access key to access the basic Firestore functionality. This allows us to extensively query data based on the parameters we pass.
8. datetime: This allows us to access current timestamps and convert date strings to datetime objects.
9. gpiozero: This allows us to interact with the Servo Motor to send commands based on some input.
10. time: The sleep functionality in the time library allows us to interact with the "sleep" function to act as a "wait key".

Implementation:

The code for the Raspberry Pi is based on a number of sections.The first section contained the libraries defined above. The second section handles extracting the string that contains the current file number. For example, the "pictures.txt" file will contain a temporary value i.e. "pic10.jpg", and we intend to update it to a new variable for the next cycle of implementation. This is done so as to create a temporary sense of a real-time parking, i.e."pic10.jpg" will contain the image for a certain car that was captured beforehand using the Pi Camera. For the second section, we define our Image Processing Algorithm that extracts the necessary text image from the image using a number of filtering algorithms. Section 4 contains the function that allows us to interact with the text image and extract the text data using OCR. The following section contains the Firestore querying the data to find the person that has the same number plate as the one that we extracted in Section 4(To be discussed in the Firestore section). This allows us to know if the person is a student or not. Finally, we update the "pictures.txt" for the next value i.e. "pic11.jpg" will be stored in this section.

1. Section 1:

   (Discussed in the *Libraries* section)

2. Section 2:

   Here we intend to extract the value of the current car in the queue. This allows us to *read* from the file and append this value to a “data” variable. After this, we extract the number that is contained in the string i.e. for the case of “pic10.jpg” we extract 10 and convert it to an integer so that we can increment this value by 1 to be assigned to variable “update” that we will use in Section 7, where we perform File Handling again. Now this variable will contain “pic11.jpg” for the next iteration of our infinite while loop.

3. Section 3:

   This is the Image Processing section that works by first extracting and converting the string “text” to an image using *cv2.imread.* We then either resize the image based on the current size or directly convert it to grayscale for more refined algorithms attached to it.

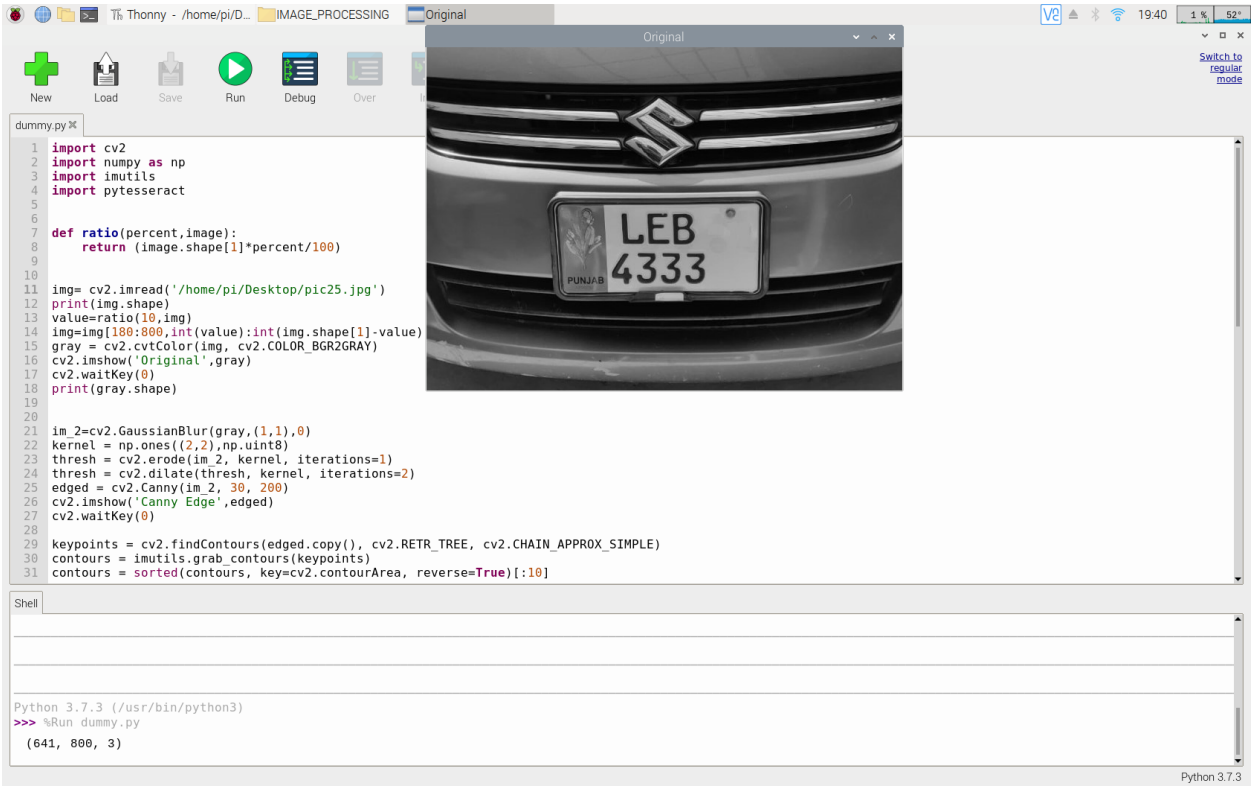


Figure 4.17, Original Image.

Firstly, we apply *Gaussian Blurring[18]* that reduces image layering by decreasing the amount of Noise and detail from the grayscale image. It removes non-essential low density edges and

smoothes the image for further processing. We then apply *Canny Edge Detection[19]* to it. This algorithm works by first removing noise(done by Gaussian Blurring) after which we apply a Sobel filter to it by calculating the gradient of image intensity at each pixel, i.e. shows the largest increase from light to dark. Next we find the non-maximum suppression removes the unwanted pixels which may not constitute an edge. This is done by finding local maximums from each pixel and removing all other values that don't correlate to a local maxima. The final step contains hysteresis thresholding which contains a maxVal and minVal. Any value connected to a maxVal is considered as an edge regardless of it being between the values of maxVal and minVal. All minVal connections are then disregarded. The image we get is similar to the one in Figure 4.18.

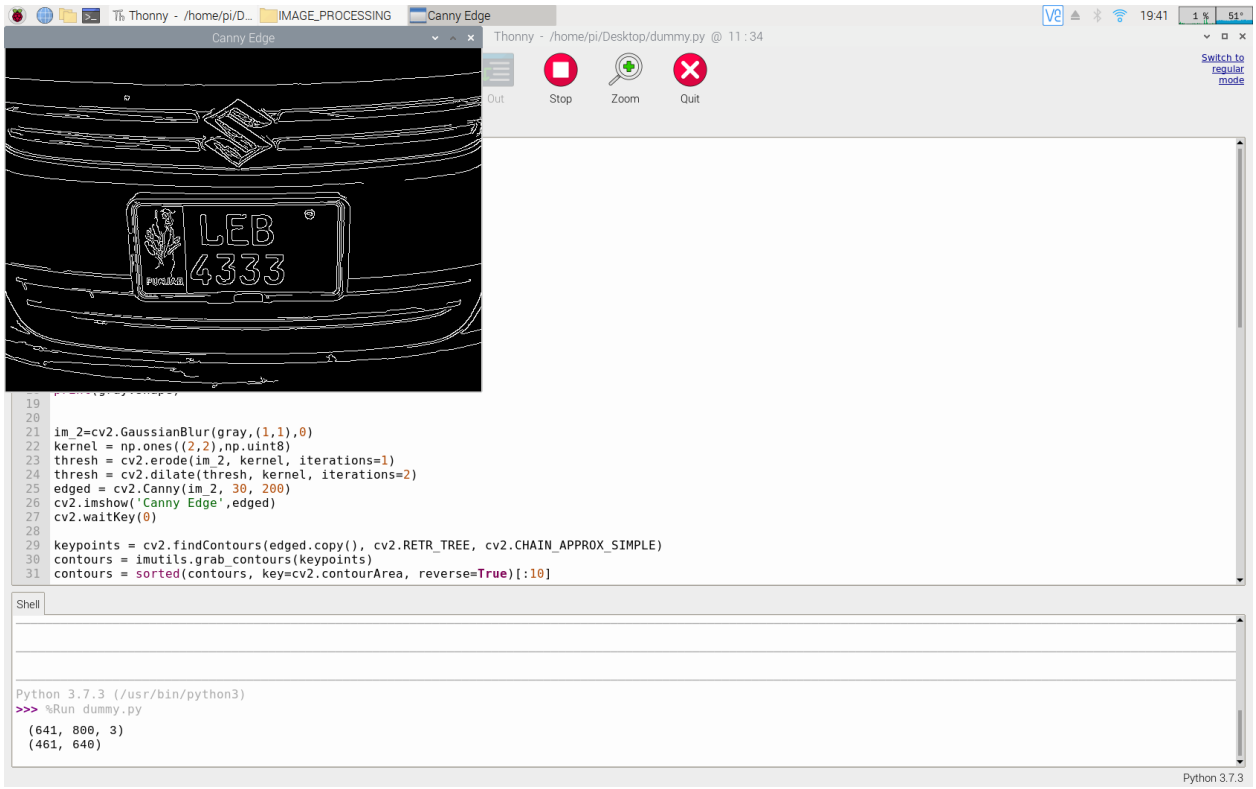


Figure 4.18, Canny Edge Detection Image.

After this we apply the *cv2.findContours[20]* that allows us to quite literally find contours using cv2.RETR_TREE that constructs a tree structure of nested contours.The next parameter cv2.CHAIN_APPROX_SIMPLE removes redundant points by compressing the contour. We then grab the contours, sort them in order of area, and then approximate a contour using a for loop to extract those contours that match a rectangle(rectangles are approximations for a number plate). The len(approx)==4 allows us to estimate the number of sides to be equal to 4. After finding the contours we create a mask that is dependent on the shape of the grayscale

image. This creates a black background image that we use *cv2.drawContours* that allows us to draw a bounded box on the image that contains only that area where the number plate has been localized. This produces a convolved image as the one illustrated in the following figure.

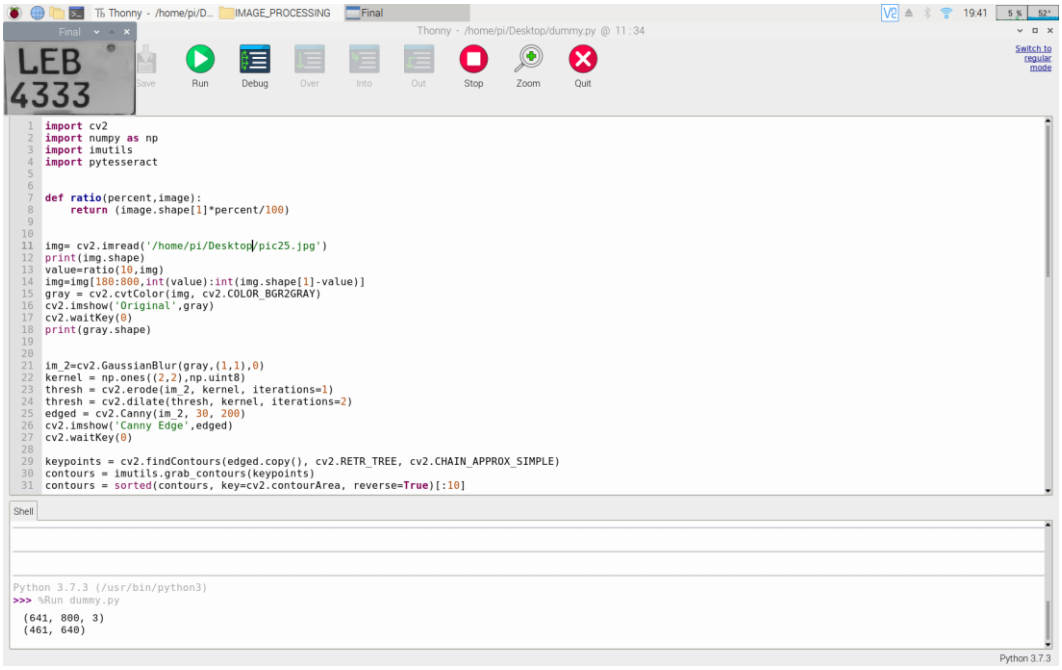


Figure 4.19,Convolved Image.

Finally, we apply *cv2.threshold* that uses OTSU_BINARIZATION to convert our grayscale image to a binary image. This allows for easy extraction of text from the image of our number plate.

Lastly, we dilate our image to morph the white background into the black text on the forefront.

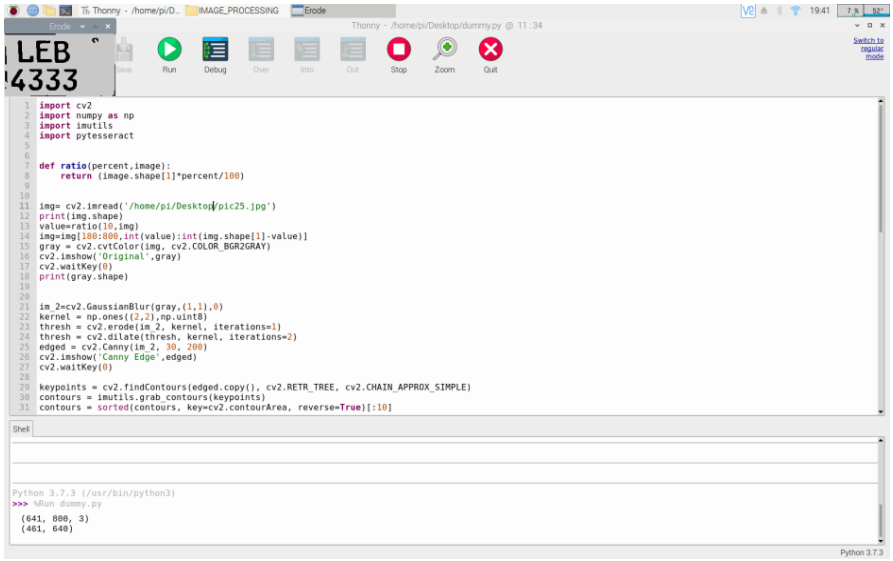


Figure 4.20, Final Dilated Image.

4. Section 4:

   We use the pytesseract[21] (equivalent of TesseractOCR in Python) to use the *pytesseract.image_to_boxes* on the dilate image to extract the text.However, the image contains unwanted characters that we use functions to remove so that from the text in Figure 4.20, we can extract LEB4333.

dummy.py

```
import cv2
import numpy as np
import imutils
import pytesseract


def ratio(percent,image):
    return (image.shape[1]*percent/100)


img= cv2.imread('/home/pi/Desktop/pic25.jpg')
print(img.shape)
value=ratio(10,img)
img=img[180:800,int(value):int(img.shape[1]-value)]
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('Original',gray)
cv2.waitKey(0)
print(gray.shape)


im_2=cv2.GaussianBlur(gray,(1,1),0)
kernel = np.ones((2,2),np.uint8)
thresh = cv2.erode(im_2, kernel, iterations=1)
thresh = cv2.dilate(thresh, kernel, iterations=2)
edged = cv2.Canny(im_2, 30, 200)
cv2.imshow('Canny Edge',edged)
cv2.waitKey(0)

keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(keypoints)
contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]
```

Shell

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run dummy.py
  (641, 800, 3)
  (461, 640)
  LEB °
  4333
```

Figure 4.21, Extracted Text.

5. Section 6:

   Here we use the *update* variable that we created in Section 1 to manually update the value in the “pictures.txt” file

Now what we do is we repeat this procedure for whenever the Exit Ultrasonic Sensor is triggered so that we can essentially see which person has left the parking spot and at what time. This allows for a complete overhaul of the parking area by having complete control over who enters the parking spots.

4.3.2.2.4 Cloud Firestore Implementation

Cloud Firestore acts as our primary backend for storing all essential information regarding students. It contains two primary collections i.e. “people" and “Current_People” both of which have similar structures. “people” document contains all necessary information of all the students registered as students for the parking area. An example of such a collection contains a wide array of documents, in our case 20 for each Person registered. An example of this is shown in Figure 4.22.

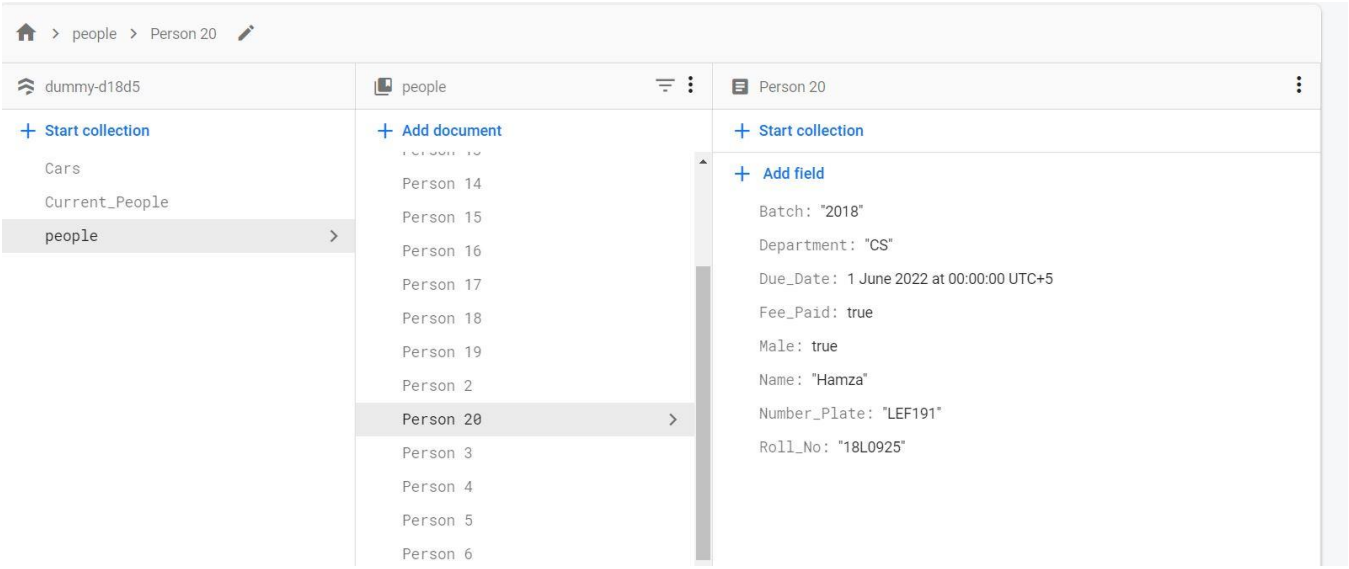


Figure 4.22, Document Data.

The entire encapsulation of Cloud Firestore is implemented real-time on Raspberry Pi that uses the number plate from the Section 4.3.2.2.3 section to compare using queries using CRUD operations[22].

```
db=firestore.client()
docs=db.collection('people').where("Number_Plate","==",number_plate).get()
```

Figure 4.23, Querying for Available Number Plates.

In our case the number plate successfully returns only one document as there is only one person applicable per car. As a result we intend to indent this document into our "Current_People" collection. What we do is convert our JSON document extracted from the Cloud Firestore that contains all details of that person to a Python dictionary(key-value pairs). If the "Fee_Paid" value is *boolean true* then we add it with a new value i.e. Time of entry using the *datetime* library to get the current time. We can see the code attached in the following figure.

```
if(people['Fee_Paid']==True):
    people['Time_Of_Entry']=datetime.now()
    db.collection('Current_People').add(people)
```

Figure 4.24, Comparing Data.

We successfully add this document to the "Current_People" collection, and use the Servo Motor to open the gate. Otherwise, we print "No" if the Fee_Paid returns false. Similarly, no action takes place if there is no document with that number plate.

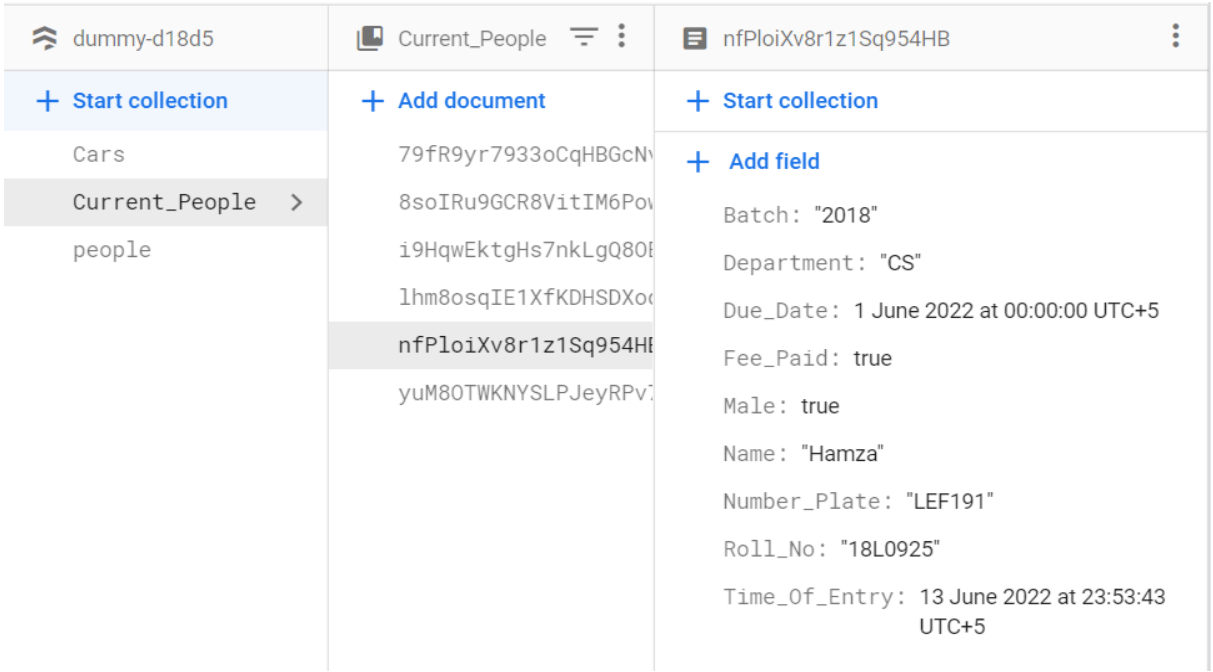


Figure 4.25, Successful addition to Current_People.

In the case of Exit Sensor we use the same code but add the extra functionality of Exit_Time to a new document. This is shown in Figure 4.26.

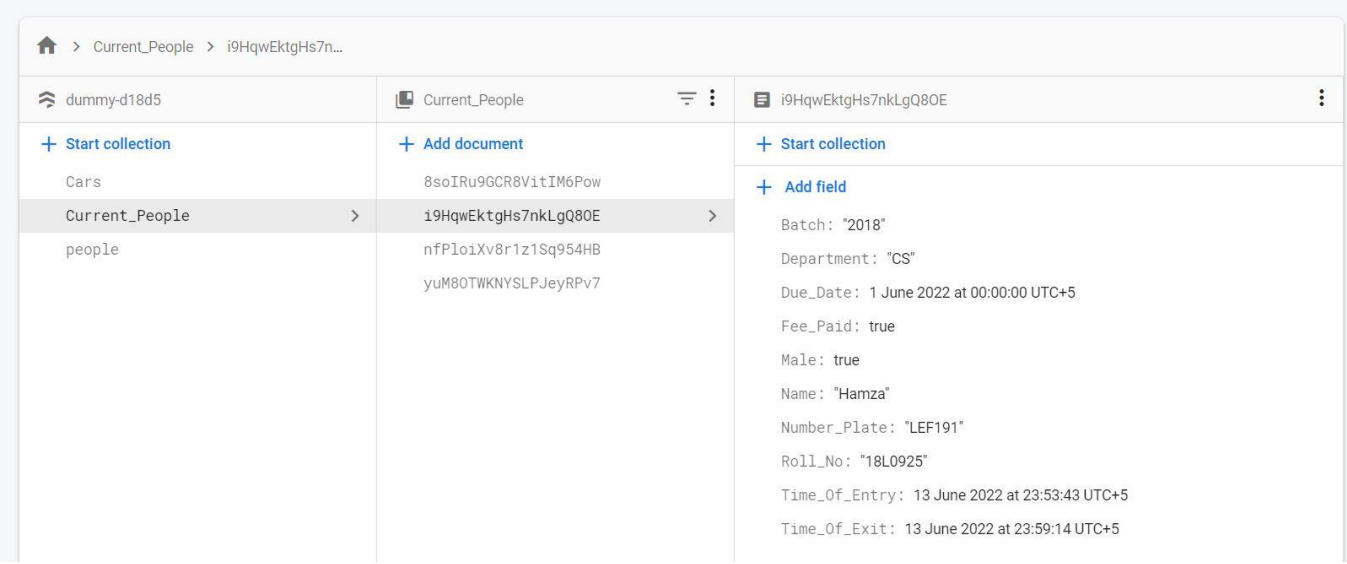


Figure 4.26, Successful updation of Current_People.

4.3.2.2.5 Refi App

Refi App is used as a secondary interface to the web-based site of Cloud Firestore as we use Refi App to Export the JSON documents for use in Data Visualization in the following section.

4.3.2.2.6 Jupyter Notebook Implementation

Libraries:

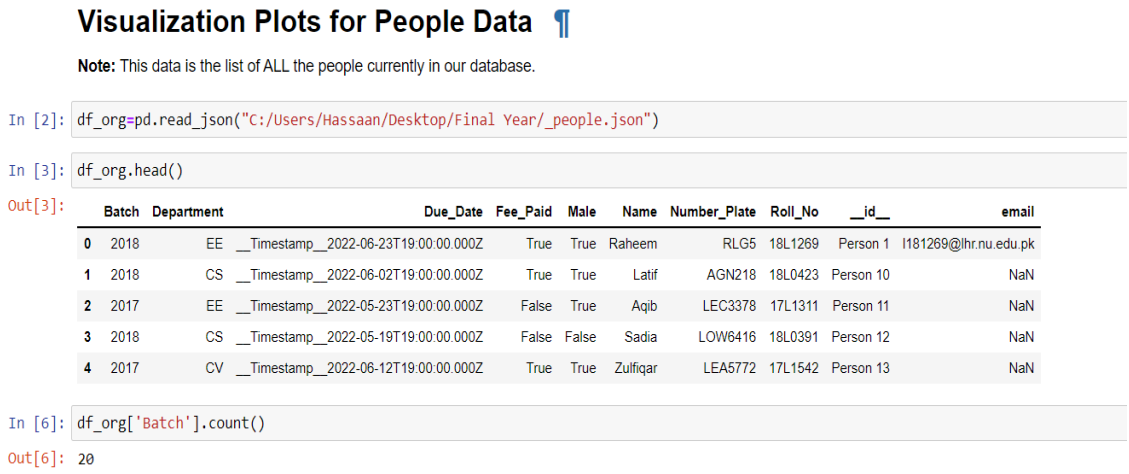
```
import pandas as pd
from datetime import datetime
import seaborn as sns
sns.set_theme(style="darkgrid")
```

Figure 4.27, Libraries for Data Analysis.

The Jupyter Notebook works by using Pandas[23](Python equivalent of SQL tables) by creating a table from a NoSQL data type(JSON document). We use the JSON documents that we extracted from the previous section to create DataFrames(Tables) in Pandas. To perform necessary data analysis we also use the Seaborn[24] library which creates count,bar,line,pair-plot, etc. In our design we primarily use count and bar plots. In regards to implementational

design we first perform *Data Cleaning(DC)* followed by *Exploratory Data Analysis(EDA)* on our data before we can extensively create plots from it. We use both the "_people.json" "FinalData.json" file which contains the entire list of people that have visited our parking lot from Monday the 23th of May to Friday the 27th of May. Having data for 20 people who have entered our parking lot at various times creates a sense of knowledge with regards to future improvements in the number of people that can be provided parking spaces. Having provided these insights essential Machine Learning models can be implemented on this dataset for future improvements.

To perform DC we first use the inbuilt pandas function i.e. *pd.read_json[25]* which reads the JSON document into a tabular form. This gets our data from the "_people.json" document. As there was no DC to be performed on this table we simply created all necessary visualization plots for the EDA segment. EDA allows us to look into the details of the data and what is expected from the statistical data. We simply use the DataFrame variable we have created to show the head(the first 5 rows of data) that is depicted in Figure 4.28.



Visualization Plots for People Data ¶

Note: This data is the list of ALL the people currently in our database.

```
In [2]: df_org=pd.read_json("C:/Users/Hassaan/Desktop/Final Year/_people.json")

In [3]: df_org.head()
```

Out[3]:

	Batch	Department	Due_Date	Fee_Paid	Male	Name	Number_Plate	Roll_No	__id__	email
0	2018	EE	__Timestamp__2022-06-23T19:00:00.000Z	True	True	Raheem	RLG5	18L1269	Person 1	l181269@lhr.nu.edu.pk
1	2018	CS	__Timestamp__2022-06-02T19:00:00.000Z	True	True	Latif	AGN218	18L0423	Person 10	NaN
2	2017	EE	__Timestamp__2022-05-23T19:00:00.000Z	False	True	Aqib	LEC3378	17L1311	Person 11	NaN
3	2018	CS	__Timestamp__2022-05-19T19:00:00.000Z	False	False	Sadia	LOW6416	18L0391	Person 12	NaN
4	2017	CV	__Timestamp__2022-06-12T19:00:00.000Z	True	True	Zulfiqar	LEA5772	17L1542	Person 13	NaN

```
In [6]: df_org['Batch'].count()
Out[6]: 20
```

Figure 4.28,Data Table for All People.

The basic values in this dataset are based on the Batch, the Department, Date of Fee Payment, whether the Person has Paid their fees, if that Person is Male or Female, the Name of that Person, the Number Plate of their Car, their Roll Number, the NoSQL Auto-ID generated by Firestore, and their **Firebase Authentication** Email.

Our primary JSON document i.e. "FinalData.json" contains all the essential information for every person that has been allowed into the parking space. To get the necessary results we need

to get the DataFrame, using the same read_json function. The sorted values based on Number_Plate are shown in the following figure.

Visualization Plots for People in Parking Lot

Note: This data is the list of ALL the people that have fully PAID their dues, that have entered the parking lot on days from 23-05-2022(Monday) to 27-05-22(Friday).

```
In [11]: df=pd.read_json("C:/Users/Hassaan/Desktop/Final Year/FinalData.json")

In [16]: df.sort_values(by='Number_Plate').head(10)
```

Out[16]:

	Batch	Department	Due_Date	Fee_Paid	Male	Name	Number_Plate	Roll_No	Time_Of_Entry	__id__	Time_Of_E
100	2018	EE	__Timestamp__2022-06-08T19:00:00.000Z	True	False	Maarij	AAD419	18L1326	__Timestamp__2022-05-27T14:32:57.404Z	vuPci9vv4hOYVQcPguOO	__Timestamp__202 05-27T16:13:08.61
32	2018	EE	__Timestamp__2022-06-08T19:00:00.000Z	True	False	Maarij	AAD419	18L1326	__Timestamp__2022-05-23T08:37:57.404Z	FoByt2oFzBqxjdBX76BX	__Timestamp__202 05-23T11:42:49.64
93	2018	EE	__Timestamp__2022-06-08T19:00:00.000Z	True	False	Maarij	AAD419	18L1326	__Timestamp__2022-06-11T11:32:57.404Z	qiDSjSNCKF7OyRfMprwc	Na
43	2018	EE	__Timestamp__2022-06-08T19:00:00.000Z	True	False	Maarij	AAD419	18L1326	__Timestamp__2022-05-25T08:24:57.404Z	LvcNGoIm6CgpRE3DCW1N	__Timestamp__202 05-25T11:54:21.60
13	2018	EE	__Timestamp__2022-06-08T19:00:00.000Z	True	False	Maarij	AAD419	18L1326	__Timestamp__2022-06-11T15:33:08.788Z	8nD42fL6VTufVyq5TAxQ	Na
80	2018	EE	__Timestamp__2022-06-08T19:00:00.000Z	True	False	Maarij	AAD419	18L1326	__Timestamp__2022-06-11T17:04:53.719Z	lMN0lslwoVT0eEKPX0P9	Na
71	2018	EE	__Timestamp__2022-06-08T19:00:00.000Z	True	False	Maarij	AAD419	18L1326	__Timestamp__2022-05-26T08:38:57.404Z	hxKHQ522xvRD4v8Sf86R	__Timestamp__202 05-26T13:09:50.13
12	2018	EE	__Timestamp__2022-06-08T19:00:00.000Z	True	False	Maarij	AAD419	18L1326	__Timestamp__2022-06-11T13:10:04.111Z	8bQO73ZuwxwaylSjTJ5f	Na
52	2018	CS	__Timestamp__2022-06-07T19:00:00.000Z	True	True	Ali	AAN797	18L0653	__Timestamp__2022-05-25T14:34:20.070Z	TlSTlfVhd1F511Xqbmzh	__Timestamp__202 05-25T16:15:45.05
63	2018	CS	__Timestamp__2022-06-07T19:00:00.000Z	True	True	Ali	AAN797	18L0653	__Timestamp__2022-05-26T10:04:20.070Z	btQk44lHOhVfScg8dyow	__Timestamp__202 05-26T14:16:34.52

Figure 4.29,Data Table for People that have used Parking Area.

Before we can extensively look into what the count plots we can build we need to look into what our data contains and we do that by using the *df.info()* function. This allows us to show the number of values in each column.

```
In [17]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106 entries, 0 to 105
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Batch          106 non-null    int64
 1   Department     106 non-null    object
 2   Due_Date       106 non-null    object
 3   Fee_Paid       106 non-null    bool
 4   Male           106 non-null    bool
 5   Name           106 non-null    object
 6   Number_Plate   106 non-null    object
 7   Roll_No        106 non-null    object
 8   Time_Of_Entry  106 non-null    object
 9   __id__         106 non-null    object
 10  Time_Of_Exit   53 non-null     object
 11  email          22 non-null     object
dtypes: bool(2), int64(1), object(9)
memory usage: 8.6+ KB
```

Figure 4.30,Info function for DataFrame.

This function highlights the number of times a person has exited the parking area as 53 compared to the number of times that person has entered which is exactly double. Looking into

the data table we can see that repetition occurs every time a person enters the parking lot as they first enter only with their Time_of_Entry whereas they exit with an additional Time_Of_Exit value. So what we intend to do is remove those values that only have a Time_Of_Entry under the assumption that every car that enters will eventually leave.

To do this step we first drop those values where the Time_Of_Exit is NULL(No value exists). After which we drop those columns that are redundant for example only a Name is necessary in finding the details of the person whereas neither Roll_No,__id__, or email are useful, so we remove these variables. Figure 4.31. shows the new dataframe after this step in DC.

```
df2.head()
```

	Batch	Department	Due_Date	Fee_Paid	Male	Name	Number_Plate	Time_Of_Entry	Time_Of_Exit
2	2018	CS	__Timestamp__2022-06-02T19:00:00.000Z	True	True	Latif	AGN218	__Timestamp__2022-05-23T09:16:00.199Z	__Timestamp__2022-05-23T13:16:09.355Z
4	2018	EE	__Timestamp__2022-05-30T19:00:00.000Z	True	False	Mahreen	LEB4333	__Timestamp__2022-05-25T08:30:17.779Z	__Timestamp__2022-05-25T10:03:27.508Z
5	2018	EE	__Timestamp__2022-06-23T19:00:00.000Z	True	True	Raheem	RLG5	__Timestamp__2022-05-24T08:40:49.283Z	__Timestamp__2022-05-24T13:16:27.313Z
8	2018	CS	__Timestamp__2022-06-16T19:00:00.000Z	True	True	Kashan	ADJ010	__Timestamp__2022-05-23T08:26:11.647Z	__Timestamp__2022-05-23T10:07:37.510Z
10	2018	EE	__Timestamp__2022-05-30T19:00:00.000Z	True	False	Mahreen	LEB4333	__Timestamp__2022-05-26T14:28:44.752Z	__Timestamp__2022-05-26T16:06:02.463Z

Figure 4.31,New DataFrame.

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53 entries, 2 to 105
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Batch          53 non-null     int64 
 1   Department     53 non-null     object
 2   Due_Date       53 non-null     object
 3   Fee_Paid       53 non-null     bool  
 4   Male           53 non-null     bool  
 5   Name           53 non-null     object
 6   Number_Plate   53 non-null     object
 7   Time_Of_Entry  53 non-null     object
 8   Time_Of_Exit   53 non-null     object
dtypes: bool(2), int64(1), object(6)
memory usage: 3.4+ KB
```

Figure 4.32,Info function for New DataFrame.

The very previous figure shows how we have performed *dimensionality reduction* by reducing the number of features in our design. Furthermore, we can improve the number of the quality of our data by converting the “string timestamps” to actual “datetime timestamps”. This will allow us to get time related information using simple functions inbuilt into the Python 3 kernel.

To do this we create a function called “get_DateTime” that takes in the string array as a parameter then splits it on the basis of the “__” characters, which is then split into another

segment to get "2022-06-08T19:00:00.000" from "__Timestamp__2022-06-08T19:00:00.000Z".

We convert this array to a datetime and append this as "Time" to a new DataFrame called timestamp. Now the "Time" column for timestamp will contain the date and time for the entire dataset. Lastly, we extract the news of the day using a *lambda* function. The code is elaborated in the following figure:

```
def get_DateTime(col):
    due_date=df2[col].str.split('__')
    array=[]
    for i in range (0,df2['index'].count()):
        array.append(due_date[i][2].split('Z')[0])
    newTime=pd.to_datetime(array)
    timestamp=pd.DataFrame(newTime,columns=['Time'])
    timestamp['Day']=timestamp['Time'].apply(lambda x:x.day_name())
    return timestamp
```

Figure 4.33, get_DateTime function .

To make use of this function with our original DataFrame we use it as follows:

```
dueDate=get_DateTime('Due_Date')
df2['Due_Date']=dueDate['Time']
df2['Due_Date_Day']=dueDate['Day']
```

Figure 4.34, get_DateTime function implemented on DataFrame .

This step is repeated for the Time_of_Entry,Due_Date, and Time_Of_Exit. The final dataframe is shown below:

```
In [26]: df2.head()
```

Out[26]:

lex	Batch	Department	Due_Date	Fee_Paid	Male	Name	Number_Plate	Time_Of_Entry	Time_Of_Exit	Due_Date_Day	Time_Of_Entry_Day	Time_Of_Exit_Day
2	2018	CS	2022-06-02 19:00:00	True	True	Latif	AGN218	2022-05-23 09:16:00.199	2022-05-23 13:16:09.355	Thursday	Monday	Monday
4	2018	EE	2022-05-30 19:00:00	True	False	Mahreen	LEB4333	2022-05-25 08:30:17.779	2022-05-25 10:03:27.508	Monday	Wednesday	Wednesday
5	2018	EE	2022-06-23 19:00:00	True	True	Raheem	RLG5	2022-05-24 08:40:49.283	2022-05-24 13:16:27.313	Thursday	Tuesday	Tuesday
8	2018	CS	2022-06-16 19:00:00	True	True	Kashan	ADJ010	2022-05-23 08:26:11.647	2022-05-23 10:07:37.510	Thursday	Monday	Monday
10	2018	EE	2022-05-30 19:00:00	True	False	Mahreen	LEB4333	2022-05-26 14:28:44.752	2022-05-26 16:06:02.463	Monday	Thursday	Thursday

Figure 4.35, Final DataFrame .

Lastly, we extract the Department,Male,Name,and Due_Date into another dataframe called "MalesAndFemales" where we have only those people that have entered the parking lot at some point from the 23th to the 27th of May.

```
In [28]: data=[df2['Department'],df2['Male'],df2['Name'],df2['Due_Date']]
         headers=['Department','Male','Name','Due_Date']
         MalesAndFemales=pd.concat(data,axis=1,keys=headers)

In [29]: MalesAndFemales.drop_duplicates(inplace=True)

In [30]: MalesAndFemales
```

Out[30]:

	Department	Male	Name	Due_Date
0	CS	True	Latif	2022-06-02 19:00:00
1	EE	False	Mahreen	2022-05-30 19:00:00
2	EE	True	Raheem	2022-06-23 19:00:00
3	CS	True	Kashan	2022-06-16 19:00:00
5	CV	True	Zulfiqar	2022-06-12 19:00:00
6	CS	True	Saad	2022-06-08 19:00:00
7	EE	True	Hassaan	2022-06-23 19:00:00
8	CS	True	Ali	2022-06-07 19:00:00
9	BBA	True	Yusuf	2022-06-02 19:00:00
12	EE	True	Muaz	2022-05-29 19:00:00
13	EE	False	Maarij	2022-06-08 19:00:00
16	EE	True	Adil	2022-05-31 19:00:00
17	CS	True	Hamza	2022-05-31 19:00:00
22	CS	True	Faizan	2022-06-02 19:00:00
27	CS	False	Fatima	2022-05-31 19:00:00

Figure 4.36, DataFrame for Males and Females .

4.3.2.2.7 Firebase Authentication

Firebase Authentication allows us to create an authorization process for our application without the need of writing the backend code. With the help of this authentication the data and the application is only accessible to the users that are allowed. This authentication process can be used for both the mobile and the web application.

Implementation:

In order to set up Firebase Authentication, firebase configurations need to be added in your application project files. For that we create a new javascript file in our mobile and web app in which we add the configurations. We install the latest SDK[26] by running the following command in the terminal.

```
$ npm install firebase
```

Figure 4.37, Installing Firebase .

The configurations can be copied by going to your created projects setting in firebase.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyCCHLkcXGD-QSX5bBePv1S_KjiBwtSsYog",
  authDomain: "dummy-d18d5.firebaseapp.com",
  databaseURL: "https://dummy-d18d5-default-rtdb.firebaseio.com",
  projectId: "dummy-d18d5",
  storageBucket: "dummy-d18d5.appspot.com",
  messagingSenderId: "269803896644",
  appId: "1:269803896644:web:aca268cb21d466409ce111",
  measurementId: "G-WEKQDX0K18"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
```

Figure 4.38, Config file of Firebase.

Then we go to our project in firebase and click on the Authentication tab. First we will set the Sign In method for our users. For our project we enabled the email and password method as shown below:

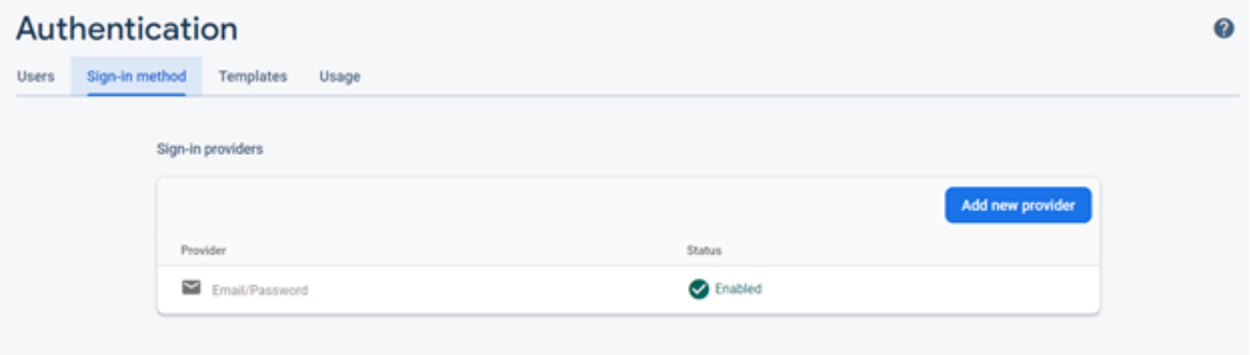


Figure 4.39, Sign-in Method .

And then we went to the users' tab and added all the users that we wanted to be registered and be authorized to use the apps.

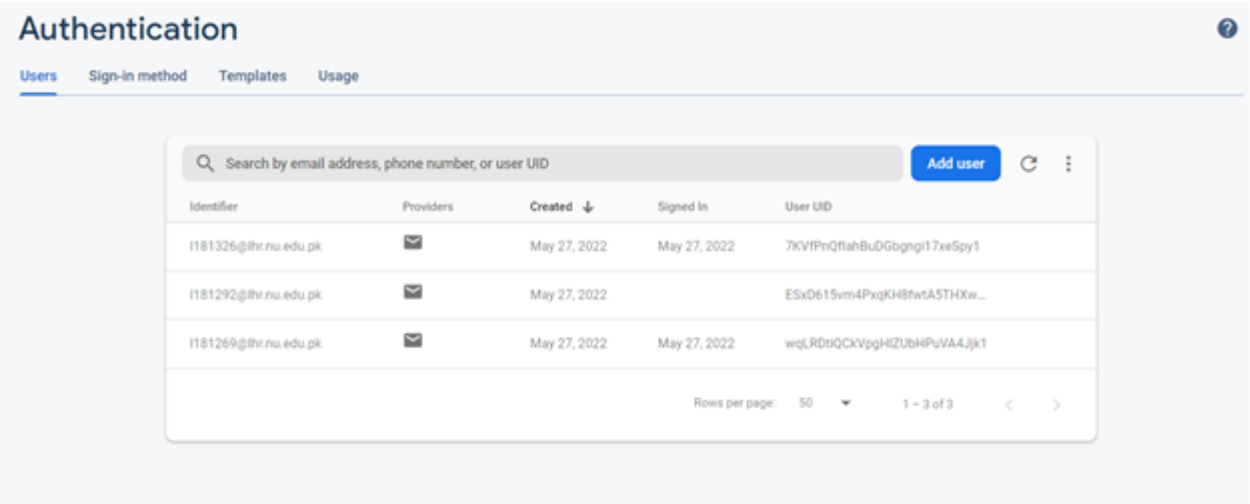


Figure 4.40, Users list.

This completes our implementation of Firebase Authentication.

4.3.2.2.8 Web Application

The Web App is used as a secondary source for displaying the digital map of the parking spot. This web app would be used by those users who do not have a mobile app or an active internet connection available at the time of parking. The web app would be available to be viewed on a display panel at the entrance of the parking lot so that the user can find the available parking spot and park their car over there.

Libraries:

```
1  import "bootstrap/dist/css/bootstrap.min.css";
2  import React, { useState, useEffect } from "react";
3  import firebase from "./firebase";
4  import "firebase/database";
```

Figure 4.41, Web App Libraries.

1. Bootstrap: It is an open source library that focuses on simplifying the development of informative web pages.

2. useState and useEffect: These are react hooks which are used for state management and for setting actions when some state or function has been changed or reloaded.

3. Firebase: This is imported in order to use Firebase services such as cloud firestore, realtime database and authentication.

Implementation:

Since we are using React[27] we created functional components to make our web app fast and responsive. We created two components:

1) Available.js

2) UnAvailable.js

These were imported into the app.js file where all the code was being compiled.

```
5  import Available from "./Available";
6  import UnAvailable from "./UnAvailable";
```

Figure 4.42, Components.

In order to get the data of of the parking spots from the database we created a function getParking_Spots() which would tell the availability of each parking spot.

```
const [parking_spots, setparking_spots] = useState([]);
const [loading, setloading] = useState(false);
const ref = firebase.database().ref("Parking_Spots");

function getParking_Spots() {

  ref.on("value", (snapshot) => {
    let docs = snapshot.val();
    const parking = [];
    for (let doc in docs) {
      parking.push({
        index: docs[doc].index,
        availability: docs[doc].availability,
      });
    }
    console.log(parking);
    setparking_spots(parking);
    setloading(false);
  });
}
```

Figure 4.43, getParkingSpots function.

In order to fetch data every time the app refreshes we used useEffect as follows:

```
useEffect(() => {
  getParking_Spots();
}, []);
```

Figure 4.44, useEffect for parking spots.

After creating these functionality we created stylings for our web app using bootstrap and css. We created the complete layout with on app.js but if at a parking spot a car was parked then the Available.js component would run and if at a parking spot the space was free then the UnAvailable.js component would run.

```
<div className=" col-2  p-0">
  {parking_spots.some(
    (p) => p.availability === true && p.index === 1
  ) ? (
    <Available spot={"01"}></Available>
  ) : (
    <UnAvailable spot={"01"}></UnAvailable>
  )}
</div>
<div className="  col-2  p-0 ">
  {parking_spots.some(
    (p) => p.availability === true && p.index === 2
  ) ? (
    <Available spot={"02"}></Available>
  ) : (
    <UnAvailable spot={"02"}></UnAvailable>
  )}
</div>
<div className=" col-2   p-0">
  {parking_spots.some(
    (p) => p.availability === true && p.index === 3
  ) ? (
    <Available spot={"03"}></Available>
  ) : (
    <UnAvailable spot={"03"}></UnAvailable>
  )}
</div>
```

Figure 4.45, Conditional statements.

4.3.2.2.9 Mobile App

The Web App is used as the primary source of displaying the digital map of the parking. Users will sign up to the app using their emails and password and would be able to see which parking spots are available. For this app we used react native with Expo CLI[28] which is a command line app to create cross platform apps easily.

Libraries:

```
import { StatusBar } from "expo-status-bar";
import React from "react";
import { StyleSheet, Text, View } from "react-native";
import { NavigationContainer } from "@react-navigation/native";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
```

Figure 4.46, Libraries for Mobile App.

In the mobile app we imported different components for our app. These components are from react as well as from expo tools.

1) StatusBar: to control the app status bar to change its text color, background color, hide it etc.

2) StyleSheet: StyleSheet is used to add styles in your front end.

3) View: View is a container in react native which is similar to div in react.

4) NavigationContainer: The NavigationContainer is responsible for managing your app state and linking your top-level navigator to the app environment.

5) CreateNativeStackNavigator: Provides a way for your app to transition between screens where each new screen is placed on top of a stack.

Implementation:

For our mobile app we created two screen:

1) Login Screen

2) Home Screen

In the Login Screen we input the email and password and the app authenticates the information with firebase.

```
const handleLogin = () => {
  auth
    .signInWithEmailAndPassword(email, password)
    .then((userCredentials) => {
      const user = userCredentials.user;
      console.log("Logged in with:", user.email);
    })
    .catch((error) => alert(error.message));
};
```

Figure 4.47, HandleLogin.

If the information is correct then the app redirects to the home screen otherwise it does not let the user redirect.

```
const LoginScreen = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  const navigation = useNavigation();

  useEffect(() => {
    const unsubscribe = auth.onAuthStateChanged((user) => {
      if (user) {
        navigation.replace("Home");
      }
    });

    return unsubscribe;
  }, []);
```

Figure 4.48, Login Screen.

In the Home Screen we implemented the same getParking_Spot() function that we implemented on the Web App.

```
const [parking_spots, setparking_spots] = useState([]);
const [loading, setloading] = useState(false);
const ref = firebase.database().ref("Parking_Spots");

function getParking_Spots() {

  ref.on("value", (snapshot) => {
    let docs = snapshot.val();
    const parking = [];
    for (let doc in docs) {
      parking.push({
        index: docs[doc].index,
        availability: docs[doc].availability,
      });
    }
    console.log(parking);
    setparking_spots(parking);
    setloading(false);
  });
}
```

Figure 4.49, Getparking spots function.

After getting data of each parking spot we used the same approach as in the web app in checking if a parking spot was occupied or free.

```
{parking_spots.some((p) => p.availability === true && p.index === 1) ? (
  <Image
    resizeMode="contain"
    style={styles.parking1}
    source={require("./handicappedspot1.png")}
  ></Image>
) : (
  <Image
    resizeMode="contain"
    style={styles.parking1}
    source={require("./occupiedspot1.png")}
  ></Image>
)}

{parking_spots.some((p) => p.availability === true && p.index === 2) ? (
  <Image
    resizeMode="contain"
    style={styles.parking2}
    source={require("./emptyspot2.png")}
  ></Image>
) : (
  <Image
    resizeMode="contain"
    style={styles.parking2}
    source={require("./occupiedspot2.png")}
  ></Image>
)}
```

Figure 4.50, Conditional Statements Mobile App.

For signing out we created a handleSignout() function which is as follows:

```
const handleSignOut = () => {
  auth
    .signOut()
    .then(() => {
      navigation.replace("Login");
    })
    .catch((error) => alert(error.message));
};
```

Figure 4.51, HandleSignout.

Chapter 5: Investigation and Testing

Various tests were performed on the final product to check if it meets the specifications outlined by the client. The following table relates the tests performed to client's specifications.

Specifications	Tests Performed
Real-time update on an available parking spot	HC-SR04 Sensor and Real Time Database Test
Digital parking map on applications	Web and Mobile App Test
Authorization and barrier system	Image Processing and Cloud FireStore Test
Data Insights	Exploratory Data Analysis Test
Barrier Control	Barrier Testing

Table 5.1, Tests Performed

5.1- Image Processing and Cloud FireStore Test

We created a dataset of 20 license plates and added each of them to our database with the status of fee paid of some to true and some to false, besides regular data to uniquely identify these values. Then we ran our image processing algorithm that extracted the text from that respective image.As a result for each number plate we checked that according to the status of fee paid does the barrier open or not. This test ensured that our image processing algorithm and our interfacing of this algorithm with our barrier system was successful or not. The following figures show successful extraction of text from image to successful updation in Cloud Firestore.

Figure 5.1, Original Image.

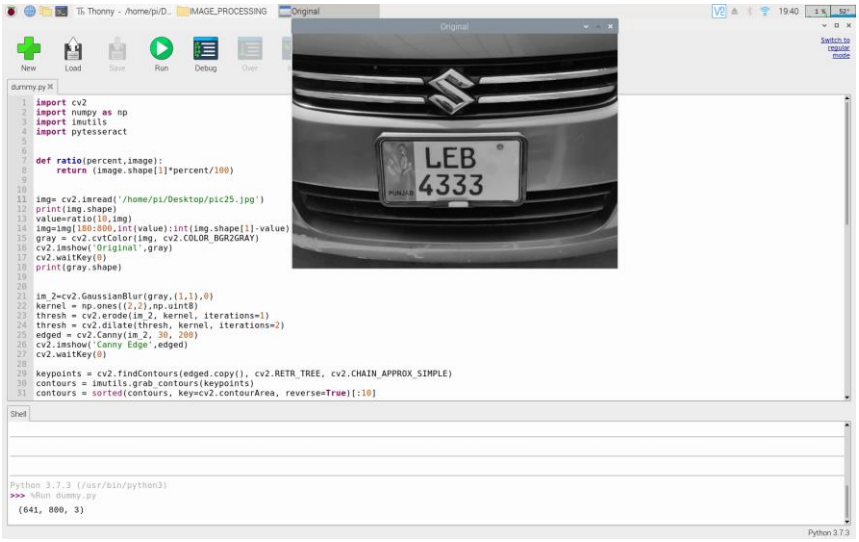


Figure 5,2, Grayscale Image.

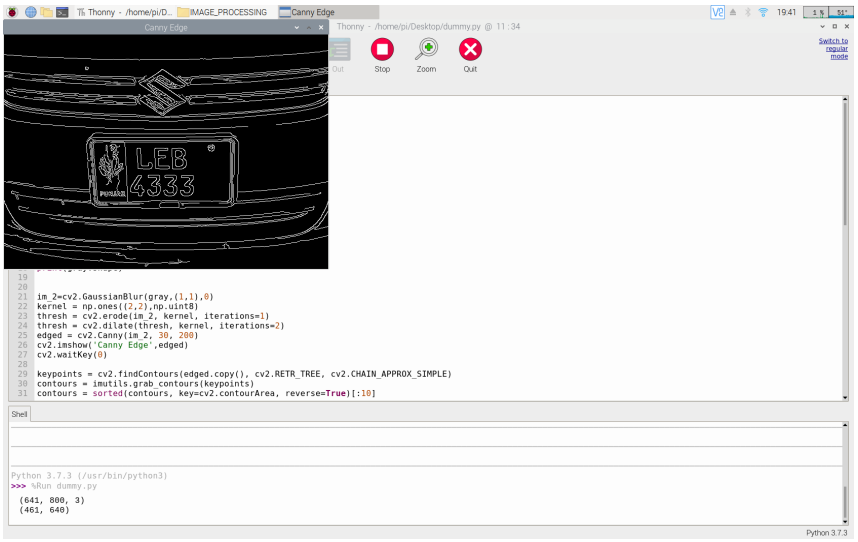


Figure 5.3, Canny Detection Image.

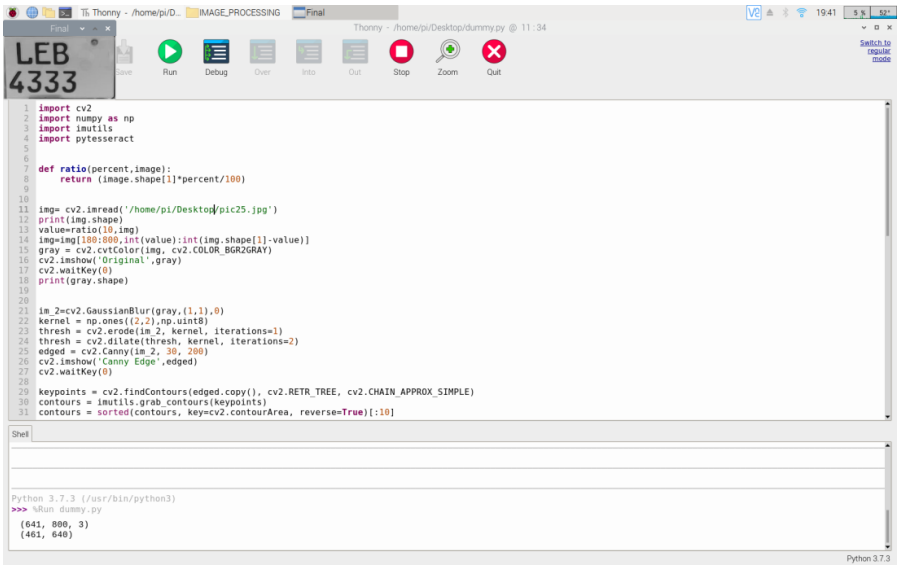


Figure 5.4, Convolved Image.

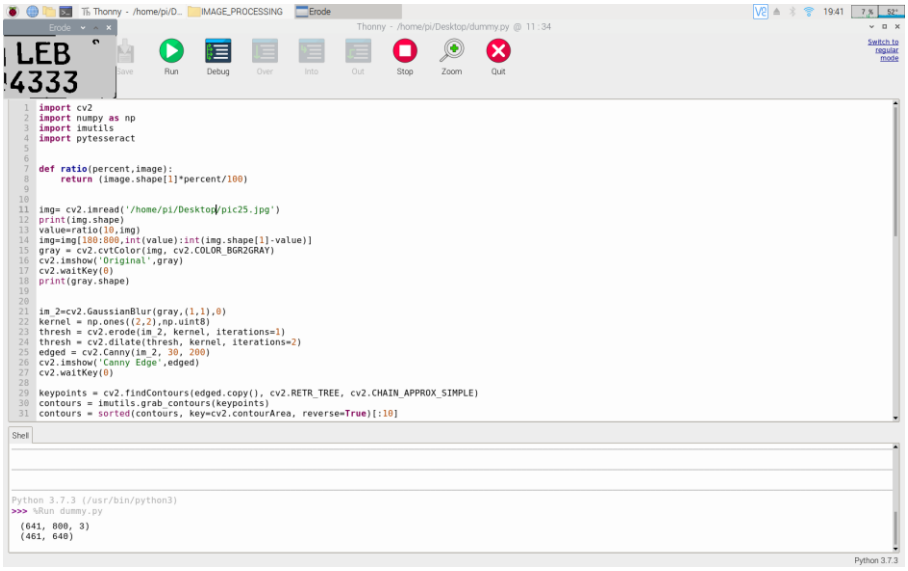


Figure 5.5, Dilated Image.

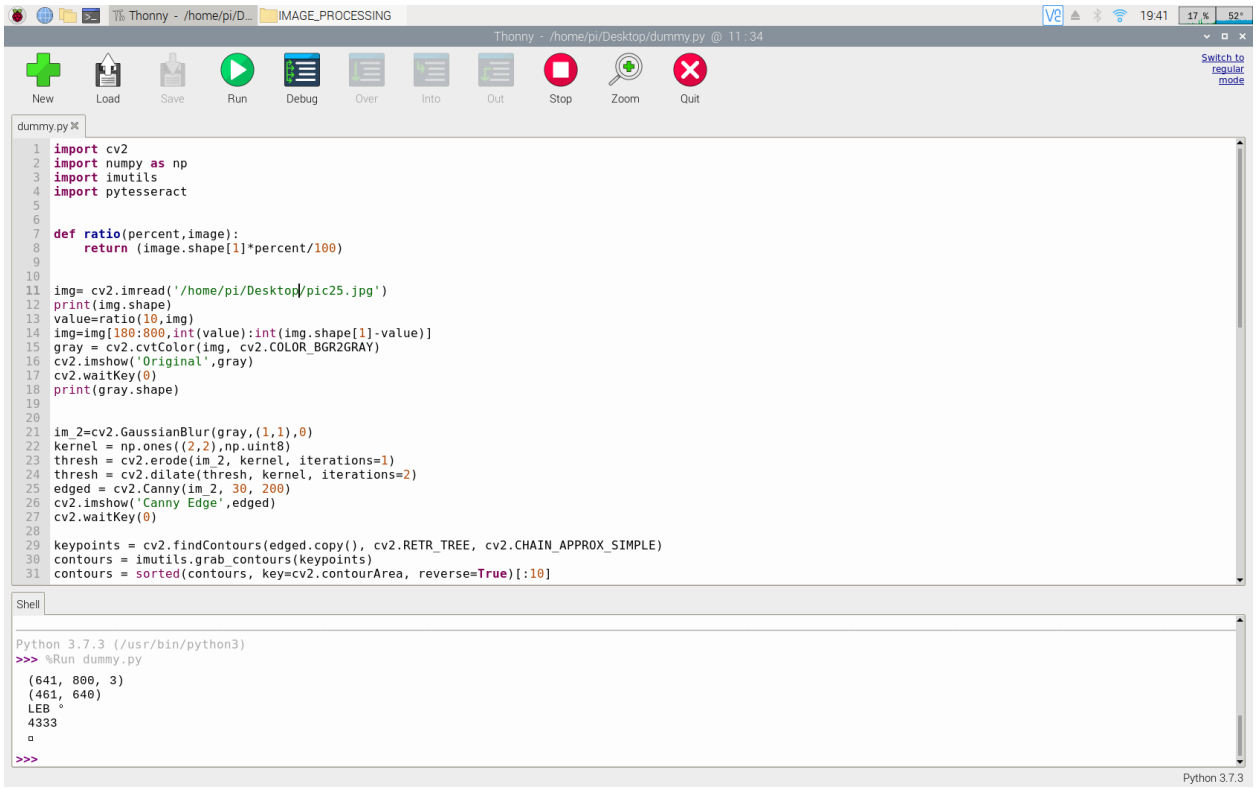


Figure 5.6, Extracted Text.

```
print(number_plate)
```

```
LEB4333
```

Figure 5.7, Corrected Number Plate.

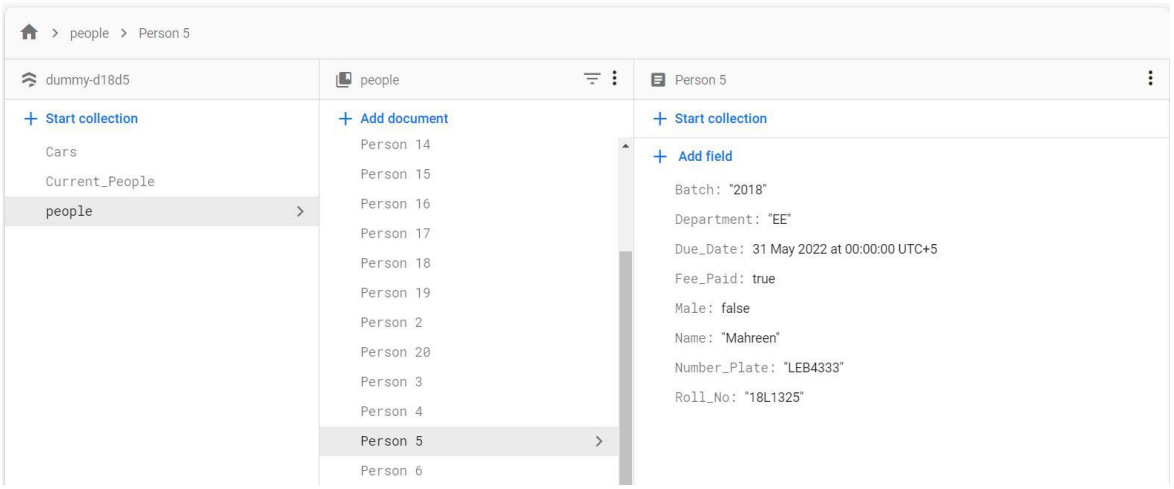


Figure 5.8, Person with Number LEB4333.

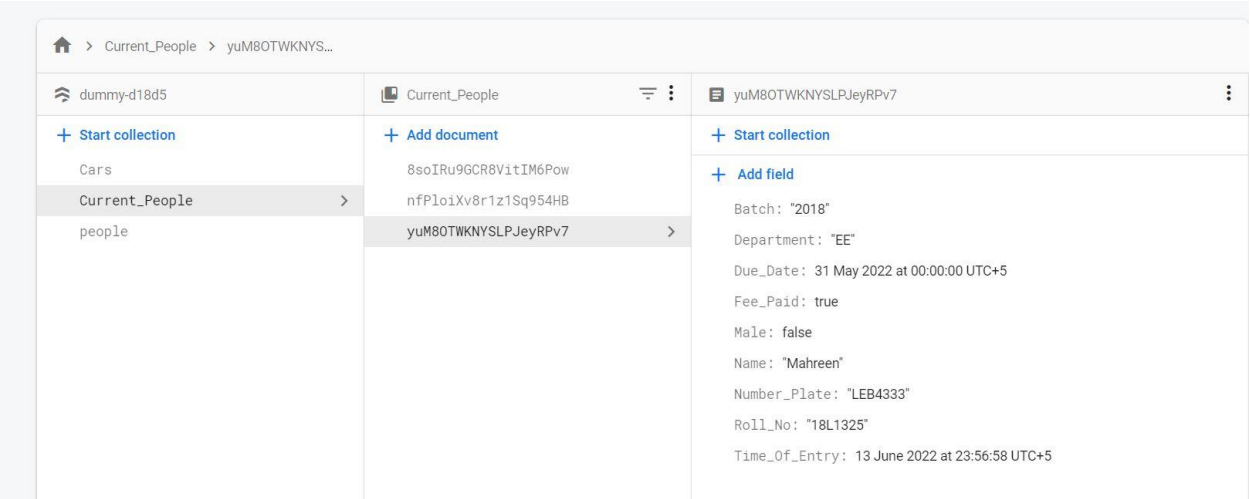


Figure 5.9,Successful Addition of Person 1.

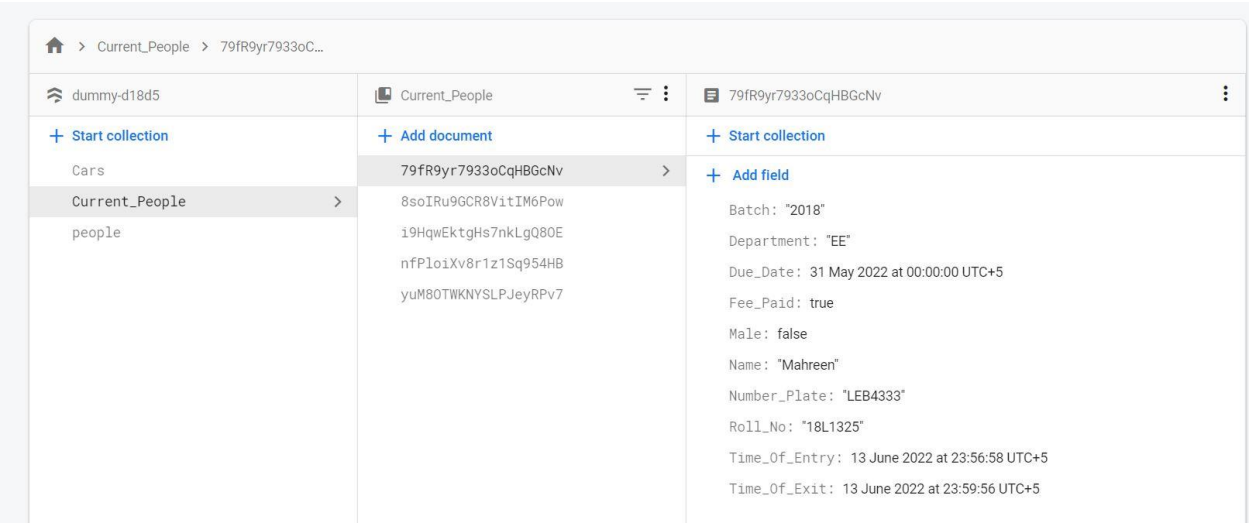


Figure 5.10,Successful Updation of Person 1.

We repeat this procedure for two other people to correct our implementation. This is done using the number plates “LEB1561” and “LEF191”. Their respective figures are attached:

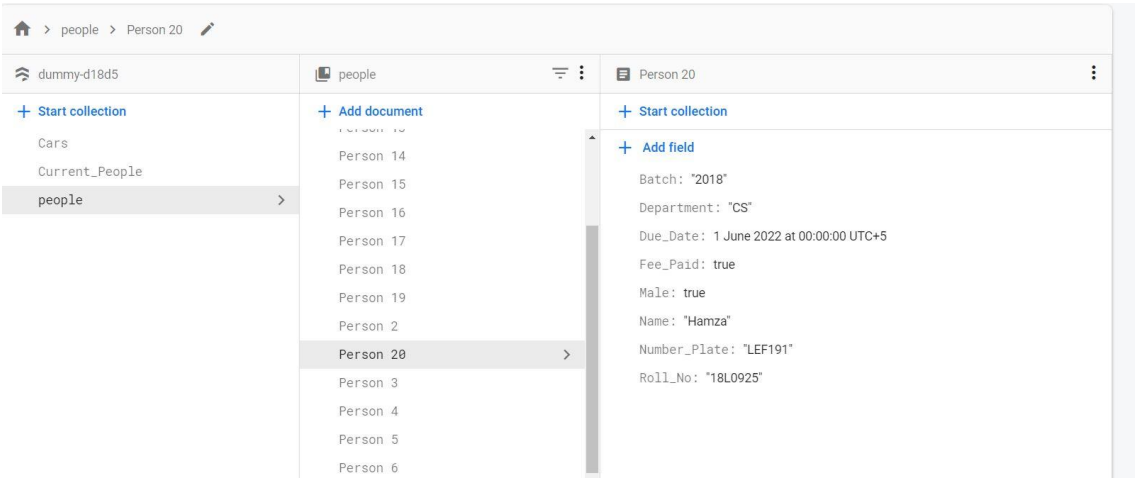


Figure 5.11, Person with Number LEF191.

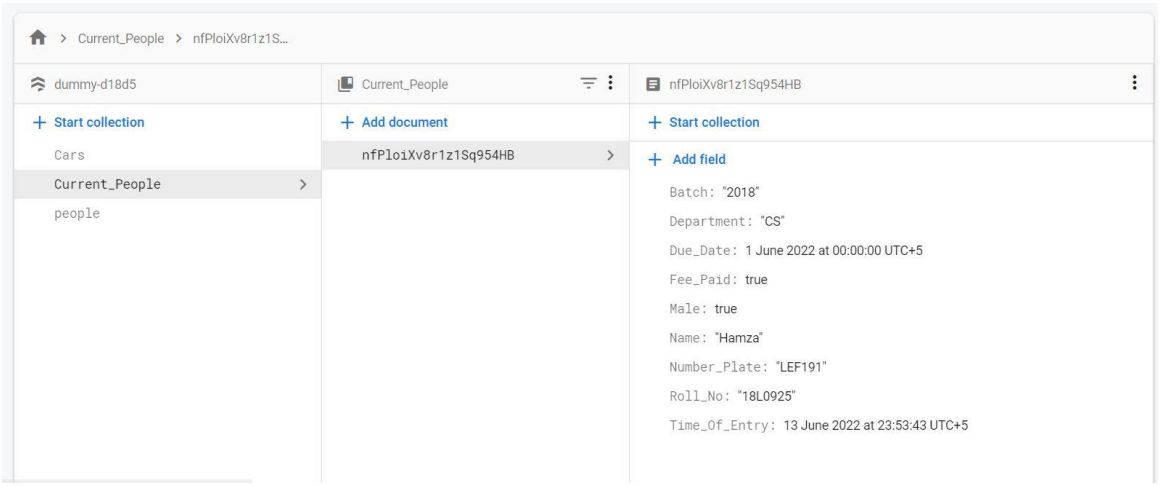


Figure 5.12,Successful Addition of Person 2.

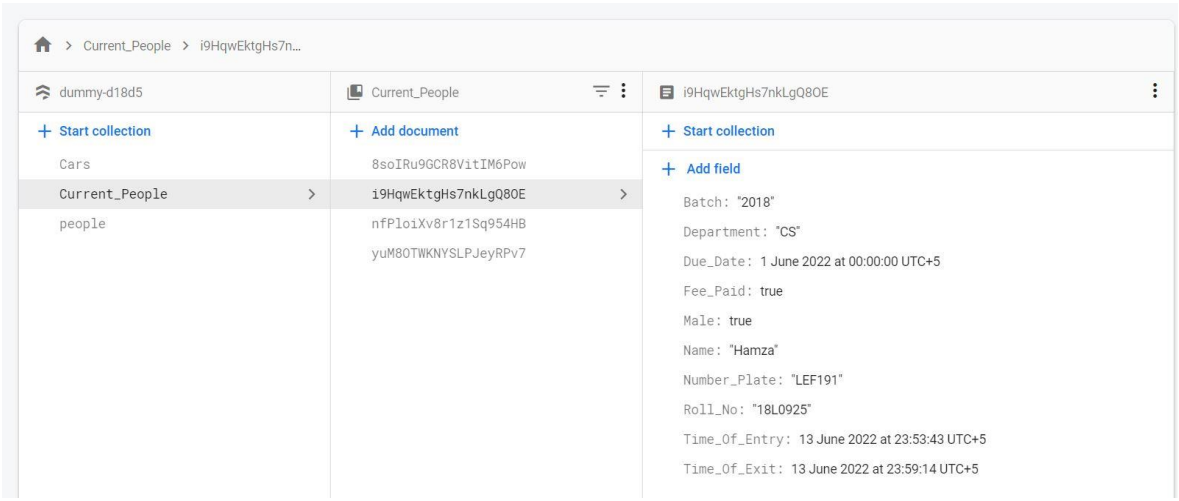


Figure 5.13,Successful Updation of Person 2.

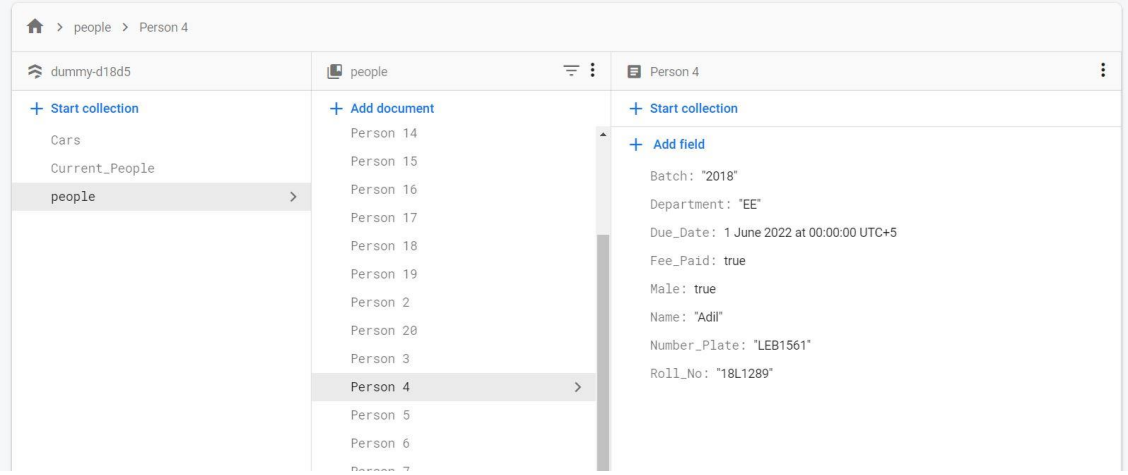


Figure 5.14, Person with Number LEB1561.

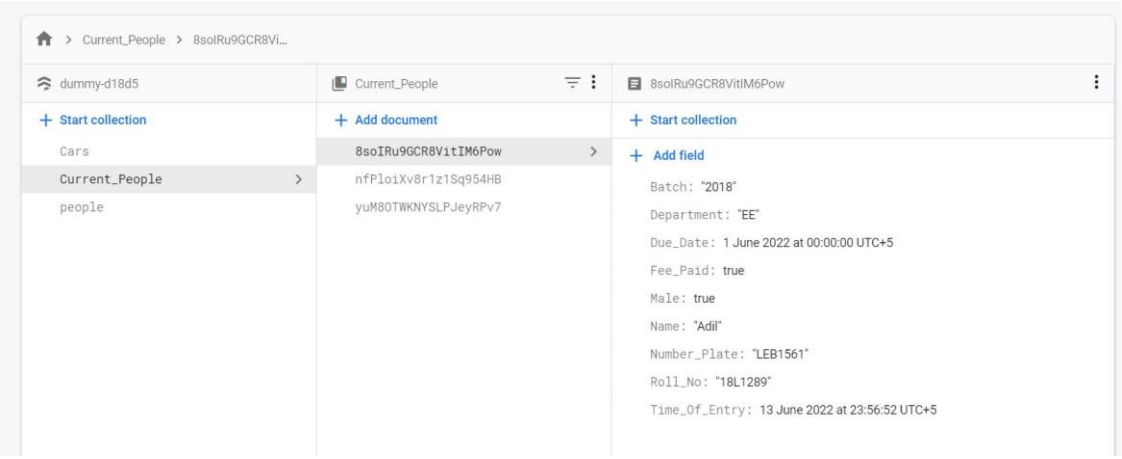


Figure 5.15,Successful Addition of Person 3.

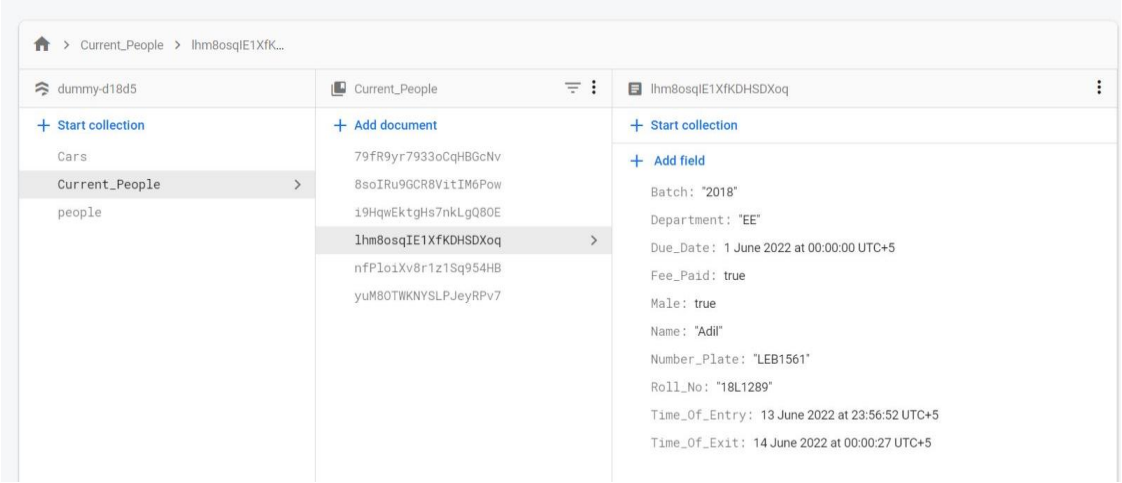


Figure 5.16,Successful Updation of Person 3.

5.2 – Barrier Testing

In order to test whether the barrier is working or not we need to know beforehand if the person is allowed or not. If persay, the person is allowed then the barrier will open for 3 seconds. The following figures show the pictures with number plates “LEB4333” and “LEC3378”.

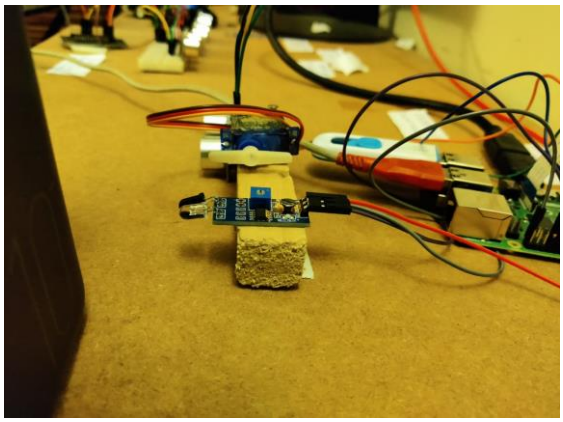


Figure 5.17,Before Entry into the Parking Area.

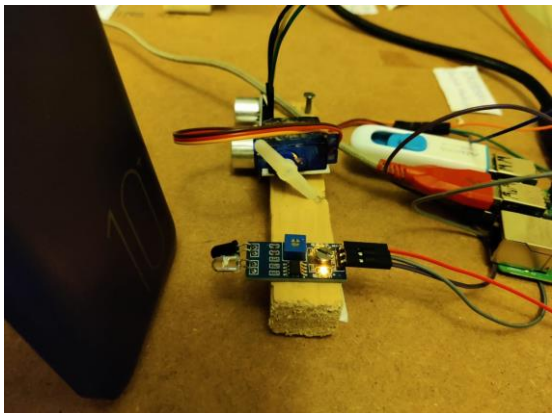


Figure 5.19, LEB4333 Successful Entry into the Parking Area.

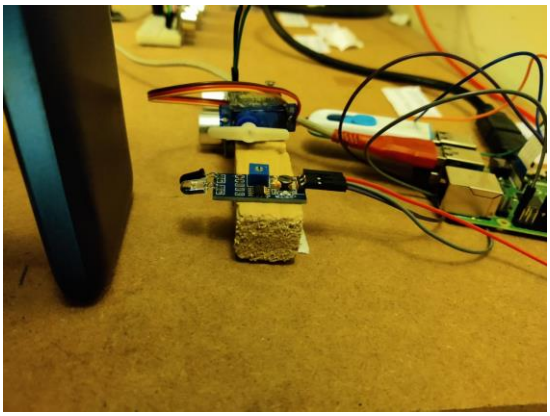


Figure 5.20, LEC3378 Un-successful Entry into the Parking Area.

5.3 – HC-SR04 Sensor and Real Time Database Test

In order to test the sensor HC-SR04, three sensors were connected to Node MCU and objects were placed in front of them to check if the sensors are detecting these objects or not, along with the distance of each object from the sensor. This test ensured us that the parking spot is being updated in real-time from the hardware side.

In order to test if the real-time update occurs on the software side as well. The detection of the object was exported to the real-time firebase database and we checked if it was being updated in firebase or not. The following figures show this:

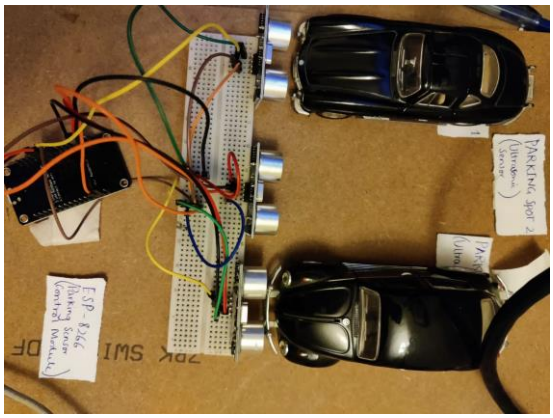


Figure 5.21,Two Cars in a Specific Area.

Figure 5.22,Real-Time Updation on Spot 1 and 3.

The values in the availability are set to “false” the moment a car is brought in front of the parking spot.

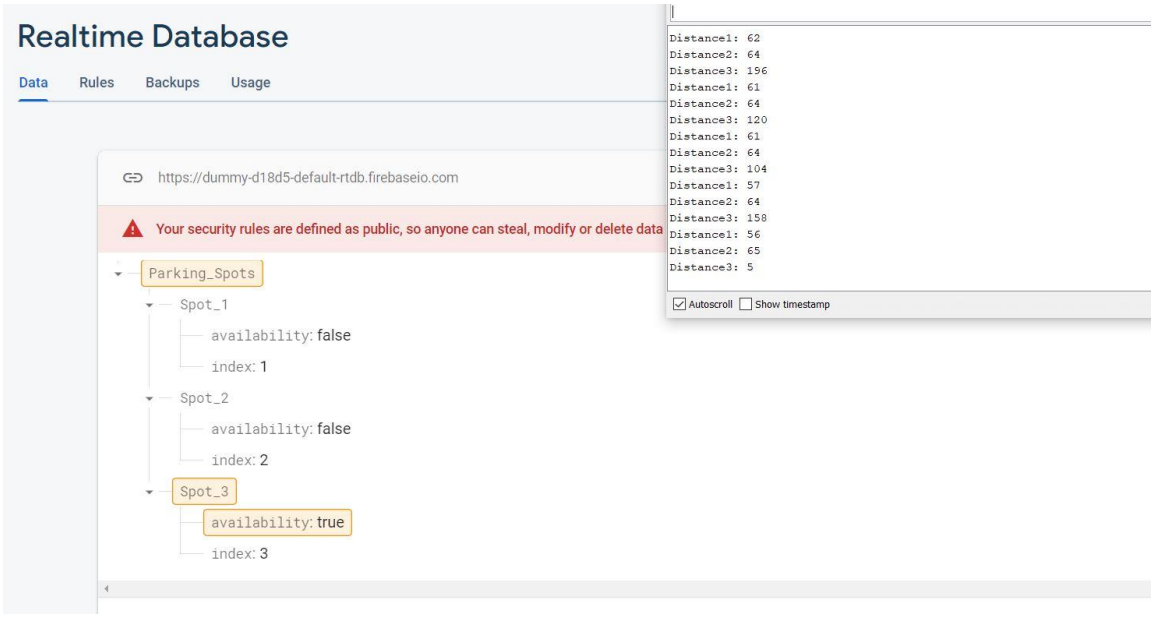


Figure 5.23,Real-Time Updation on Spot 3.

Figure 5.24,Real-Time Updation on All Three Spots.

5.4 – Web And Mobile App Test

In the database that we designed and configured with the Web and Mobile App. We added dummy values of each parking spot in the database and checked whether these values are updated in the mobile and web app as well or not.

For Figure 5.22 in Section 5.3 where only Spot 2 was available the digital map on the mobile and web app showed as follows:

Figure 5.25, 2nd spot available.

For figure 5.23 in HC-SR04 Sensor and Real Time Database Test where only spot 3 was available the digital map on the mobile and web app showed as follows:

Figure 5.26, 3rd spot available.

For figure 5.24 in HC-SR04 Sensor and Real Time Database Test where all three spots were available the digital map on the mobile and web app showed as follows:

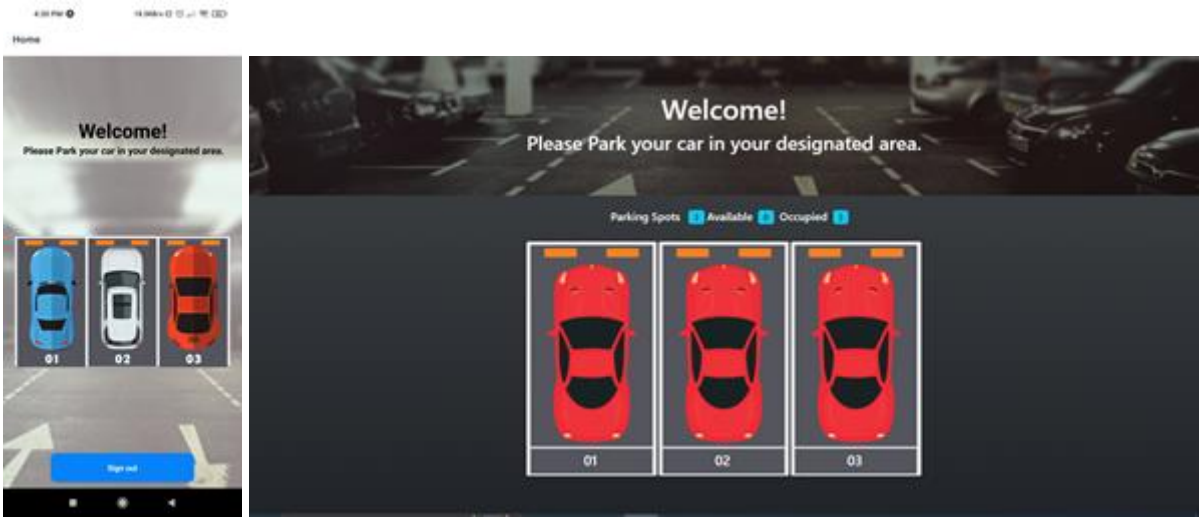


Figure 5.27, All three occupied.

5.5- Exploratory Data Analysis Test

In order to show data insights we would need at least 5 days for proper visualization of data. So in order to test our Data Insights algorithm we created a dummy dataset using our Entry(IR) Sensor and Exit(Ultrasonic) Sensor to add people to the list with their Time_Of_Entry and Time_Of_Exit. This test ensured that our Data Insights feature is working properly.

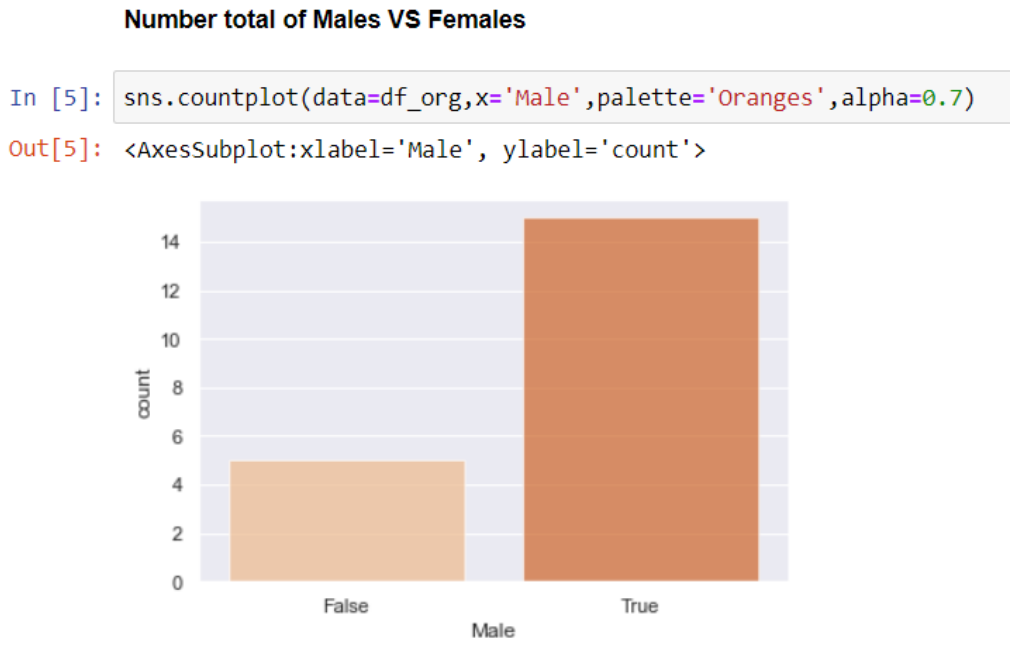


Figure 5.28, Total Number of People based on Gender.

The “df_org” data contains all the necessary information for every single person regardless of Fee_Paid. This plot shows us that a total of 15 Males VS 5 Females are present.

Fee Paid VS Unpaid

```
In [6]: sns.countplot(data=df_org,x='Fee_Paid',palette='Accent',alpha=0.7)
Out[6]: <AxesSubplot:xlabel='Fee_Paid', ylabel='count'>
```

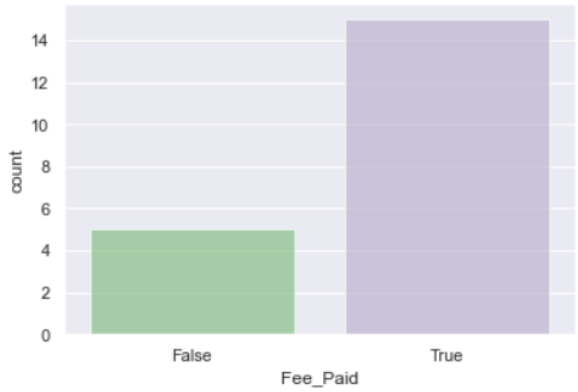


Figure 5.29, Total Number of People based on Fee_Paid.

Number of Total People based on Gender in each Department

```
In [7]: sns.countplot(data=df_org,x='Department',hue='Male',palette='GnBu',alpha=0.7)
Out[7]: <AxesSubplot:xlabel='Department', ylabel='count'>
```

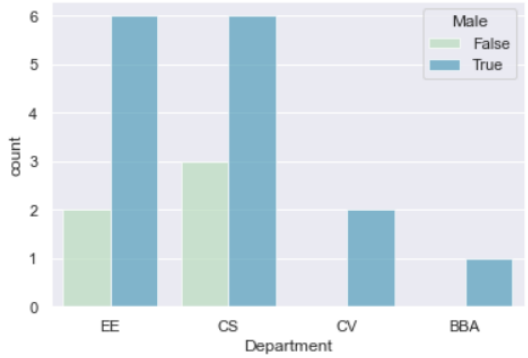


Figure 5.30, Total Number of People in each Department based on Gender.

Number of Total People based on Gender in each Batch

```
In [8]: sns.countplot(data=df_org,x='Batch',hue='Male',palette='Blues',alpha=0.7)
Out[8]: <AxesSubplot:xlabel='Batch', ylabel='count'>
```

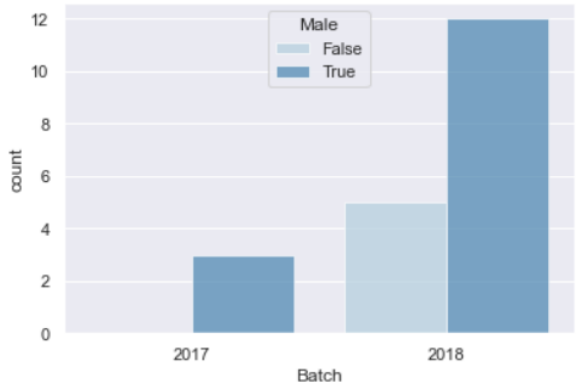


Figure 5.31, Total Number of People in each Batch based on Gender.

We then use our DataFrame that we created in Section 4.3.2.2.6 using a time string to convert to a datetime object. We name this DataFrame "df". Besides "df" we have another DataFrame called "MalesAndFemales" showing each person from every department that entered the parking lot.

Number of Allowed People based on Gender in each Department

```
In [29]: sns.countplot(data=MalesAndFemales,x='Department',hue='Male',palette='magma',alpha=0.7)
Out[29]: <AxesSubplot:xlabel='Department', ylabel='count'>
```

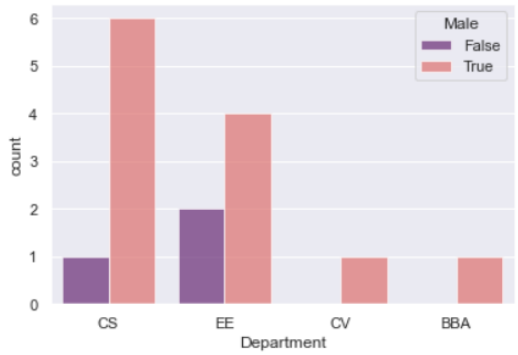


Figure 5.32, Total Number of People in each Department based on Gender and with Fee_Paid=True.

From the previous Seaborn plot we can analyze that a majority of people that entered the parking area were mainly Male and the cumulative sum of parkers were from the CS department.

Due Dates

```
In [30]: groupedDue=df2.groupby(MalesAndFemales['Due_Date'].apply(lambda x:x.date()))['index'].count()
         groupedDue.plot(kind='bar',cmap='terrain',alpha=0.7)
Out[30]: <AxesSubplot:xlabel='Due_Date'>
```

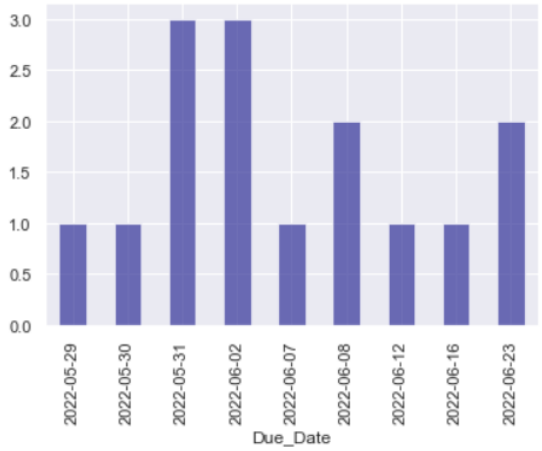


Figure 5.33, Total Number of People with respect to Due Dates.

Figure 5.33 shows that a large number of people have their dues to be paid on either 31-05-22 or on 02-06-22.

Allowed Males and Females

```
In [93]: sns.countplot(data=MalesAndFemales,x='Male',palette='viridis')
Out[93]: <AxesSubplot:xlabel='Male', ylabel='count'>
```

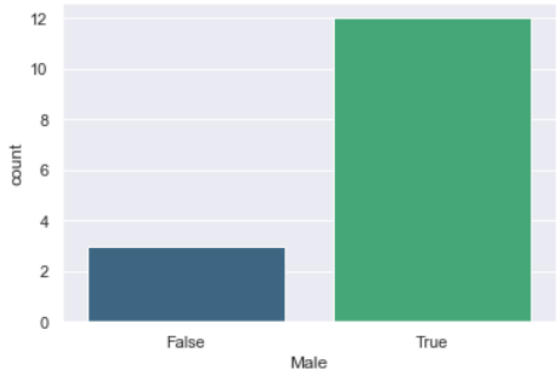


Figure 5.34, Total Number of Males and Females in Parking Lot.

The total number of people that have entered the parking lot from 23-05-22 to 27-05-22 are 15 where we can visualize that 12 of them were Male, whereas only 3 were Female from a total of 20 students(as illustrated in Figure 5.34). So we come to the conclusion that the

majority of students are Male.

Number of People that entered Parking Lot on specific Days

```
In [94]: groupedEntry=df2.groupby(df2['Time_Of_Entry'].apply(lambda x:x.date()))['index'].count()
         groupedEntry.plot(kind='bar',cmap='viridis',alpha=0.7)

Out[94]: <AxesSubplot:xlabel='Time_Of_Entry'>
```

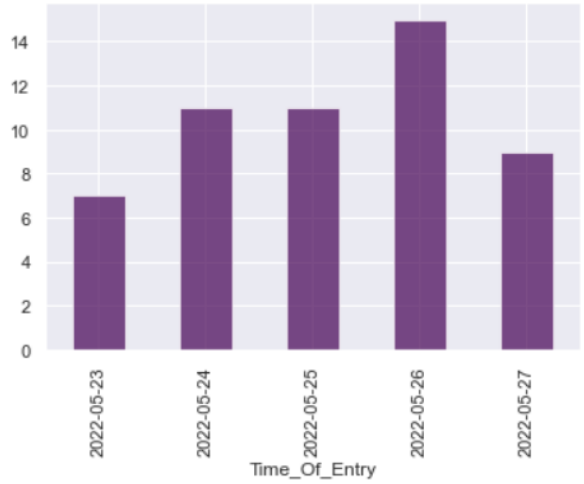


Figure 5.35, Total Number of People on specific days.

The previous plot extensively shows us that the least number that entered the parking lot were on Monday while a maximum number of 14 students parked their cars throughout Thursday. Furthermore, this shows us that most of these students tended to find parking difficult in this constrained lot as with only limited spots there was a high chance only a few people would have this parking area as accessible.

The following plots will show us the number of people that parked or exited the lot in certain time ranges. We provide these time ranges as between 8am and 10am, 10am and 12pm, 12pm and 2pm, and finally 2pm and 4pm.

NUMBER OF PEOPLE THAT ENTERED PARKING AREA:

Number of People that entered Parking Lot on specific Days from 8 to 10am

```
In [98]: timeOfEntry_8_10=df2[(df2['Time_Of_Entry'].apply(lambda x:x.hour)>=8) & (df2['Time_Of_Entry'].apply(lambda x:x.hour)<=10)]
         grouped_timeOfEntry_8_10=timeOfEntry_8_10.groupby(['Time_Of_Entry_Day']).count()
         grouped_timeOfEntry_8_10['index'].plot(kind='bar',cmap='cividis',alpha=0.7)

Out[98]: <AxesSubplot:xlabel='Time_Of_Entry_Day'>
```

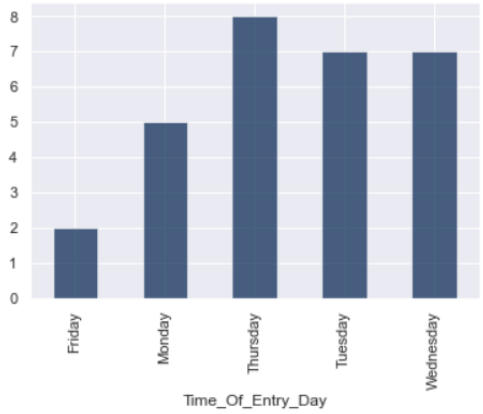


Figure 5.36, Total Number of People parked between 8 and 10am.

Number of People that entered Parking Lot on specific Days from 10am to 12pm

```
In [33]: timeOfEntry_10_12=df2[(df2['Time_Of_Entry'].apply(lambda x:x.hour)>=10) & (df2['Time_Of_Entry'].apply(lambda x:x.hour)<=12)]
         grouped_timeOfEntry_10_12=timeOfEntry_10_12.groupby(['Time_Of_Entry_Day']).count()
         grouped_timeOfEntry_10_12['index'].plot(kind='bar',cmap='Pastel1',alpha=0.8)

Out[33]: <AxesSubplot:xlabel='Time_Of_Entry_Day'>
```

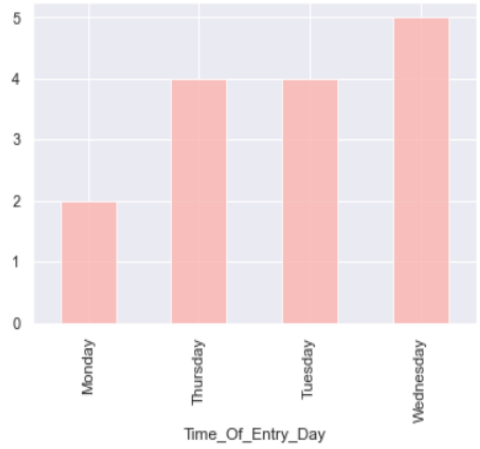


Figure 5.37, Total Number of People parked between 10am and 12pm each da

Number of People that entered Parking Lot on specific Days from 12 to 2pm

```
In [32]: timeOfEntry_12_2=df2[(df2['Time_Of_Entry'].apply(lambda x:x.hour)>=12) & (df2['Time_Of_Entry'].apply(lambda x:x.hour)<=14)]
grouped_timeOfEntry_12_2=timeOfEntry_12_2.groupby(['Time_Of_Entry_Day']).count()
grouped_timeOfEntry_12_2['index'].plot(kind='bar',cmap='plasma',alpha=0.7)

Out[32]: <AxesSubplot:xlabel='Time_Of_Entry_Day'>
```

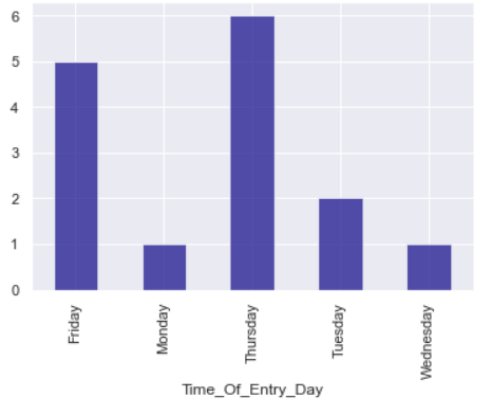


Figure 5.38, Total Number of People parked between 12 and 2pm each day.

Number of People that entered Parking Lot on specific Days from 2 to 4pm

```
In [101]: timeOfEntry_2_4=df2[(df2['Time_Of_Entry'].apply(lambda x:x.hour)>=14) & (df2['Time_Of_Entry'].apply(lambda x:x.hour)<=16)]
grouped_timeOfEntry_2_4=timeOfEntry_2_4.groupby(['Time_Of_Entry_Day']).count()
grouped_timeOfEntry_2_4['index'].plot(kind='bar',cmap='Spectral',alpha=0.65)

Out[101]: <AxesSubplot:xlabel='Time_Of_Entry_Day'>
```

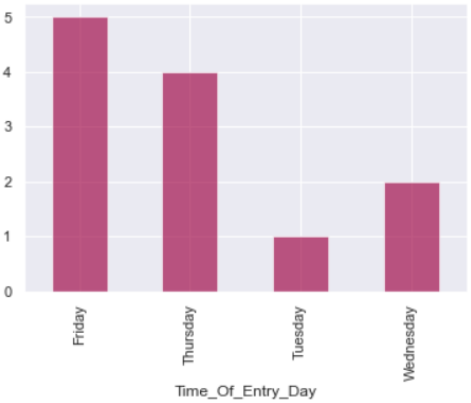


Figure 5.39, Total Number of People parked between 2 and 4pm each day.

NUMBER OF PEOPLE THAT EXITED PARKING AREA:

Number of People that exit Parking Lot on specific Days from 8 to 10am

```
In [37]: timeOfExit_8_10=df2[(df2['Time_Of_Exit'].apply(lambda x:x.hour)>=8) & (df2['Time_Of_Exit'].apply(lambda x:x.hour)<=10)]
         grouped_timeOfExit_8_10=timeOfExit_8_10.groupby(['Time_Of_Exit_Day']).count()
         grouped_timeOfExit_8_10['index'].plot(kind='bar',cmap='PRGn',alpha=0.7)

Out[37]: <AxesSubplot:xlabel='Time_Of_Exit_Day'>
```

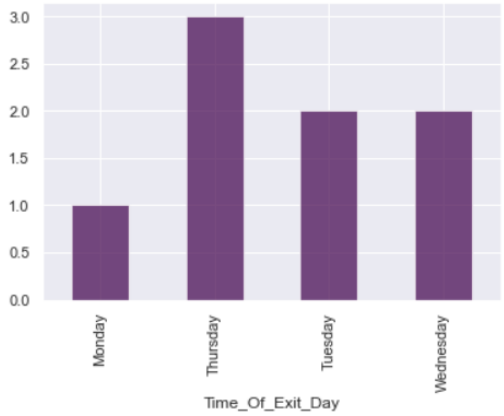


Figure 5.40, Total Number of People exit the area between 8 and 10am each day.
Note: No one exit on Friday between 8 and 10am.

Number of People that exit Parking Lot on specific Days from 10am to 12pm

```
In [45]: timeOfExit_10_12=df2[(df2['Time_Of_Exit'].apply(lambda x:x.hour)>=10) & (df2['Time_Of_Exit'].apply(lambda x:x.hour)<=12)
         grouped_timeOfExit_10_12=timeOfExit_10_12.groupby(['Time_Of_Exit_Day']).count()
         grouped_timeOfExit_10_12['index'].plot(kind='bar',cmap='RdGy',alpha=0.7)

Out[45]: <AxesSubplot:xlabel='Time_Of_Exit_Day'>
```

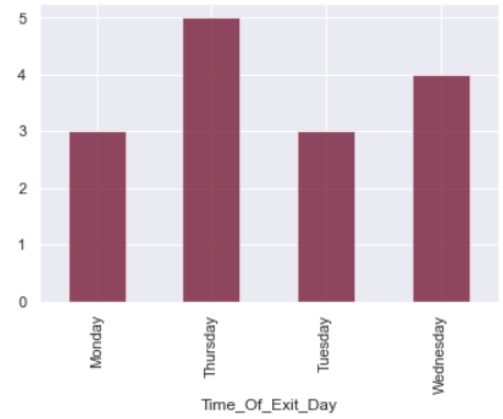


Figure 5.41, Total Number of People exit the area between 10am and 12pm each day.

Note: No one exit on Friday between 10am and 12pm.

Number of People that exit Parking Lot on specific Days from 12 to 2pm

```
In [47]: timeOfExit_12_2=df2[(df2['Time_Of_Exit'].apply(lambda x:x.hour)>=12) & (df2['Time_Of_Exit'].apply(lambda x:x.hour)<=14)]
grouped_timeOfExit_12_2=timeOfExit_12_2.groupby(['Time_Of_Exit_Day']).count()
grouped_timeOfExit_12_2['index'].plot(kind='bar',cmap='RdYlGn',alpha=0.7)

Out[47]: <AxesSubplot:xlabel='Time_Of_Exit_Day'>
```

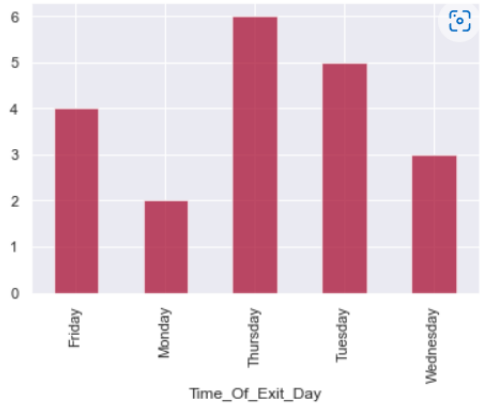


Figure 5.42, Total Number of People exit the area between 12 and 2pm each day.

Number of People that exit Parking Lot on specific Days from 2 to 4pm

```
In [51]: timeOfExit_2_4=df2[(df2['Time_Of_Exit'].apply(lambda x:x.hour)>=14) & (df2['Time_Of_Exit'].apply(lambda x:x.hour)<=16)]
grouped_timeOfExit_2_4=timeOfExit_2_4.groupby(['Time_Of_Exit_Day']).count()
grouped_timeOfExit_2_4['index'].plot(kind='bar',cmap='twilight_shifted',alpha=0.6)

Out[51]: <AxesSubplot:xlabel='Time_Of_Exit_Day'>
```

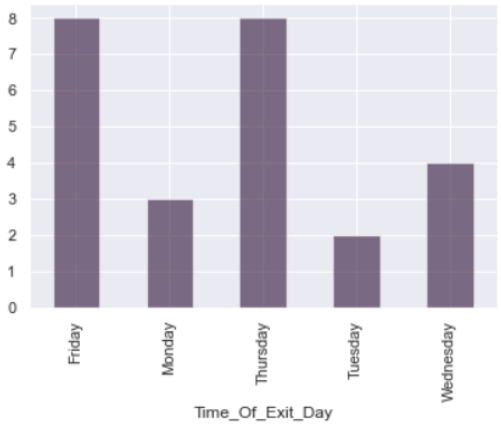


Figure 5.43, Total Number of People exit the area between 2 and 4pm each day.

Hence, we come to the conclusion that these visualizations will be increasingly beneficial whenever there is going to be an increase in the parking area. This will cause a comprehensive Machine Learning model to be generated on a real-time dataset similar to the dummy dataset we have created, hence allowing for an effective increase in parking spots without going overboard.

5.6 – Summary of Tests Performed

Tests Performed	Result
HC-SR04 Sensor and Real Time Database Test	Successful
Web and Mobile App Test	Successful
Image Processing and Cloud FireStore Test	Successful
Exploratory Data Analysis Test	Successful
Barrier Testing	Successful

Table 5.2, Summary of Tests Performed

Chapter 6: User Guide

Mobile App:

1) Install Expo Go Application on your android or IOS device.

IOS: https://itunes.apple.com/app/apple-store/id982107779

Android: https://play.google.com/store/apps/details?id=host.exp.exponent&referrer=www

2) After Installation is completed scan the following QR code or go the following link.

Link: exp://exp.host/@ab.raheem26/smart-parking?release-channel=default

Figure 6.1, QR Code.

3) The App would launch as follows:

Figure 6.2, Sign In Page.

4) Login to the app using the registered email and password. For demo use the following credentials:

Email: guest@guest.com

Password: guest123

5) When the login is successful the app parking map would be displayed as follows:

Figure 6.3, Home Screen.

6) The parking spots in which there is a car shows that these spots are occupied whereas the spots without a car are free and you can park there.

7) In order to sign out from this app click on the sign out button and you will be redirected to the Login page.

Web App:

1) The web app would be displayed at the entrance of the parking on a display panel as follows:

Figure 6.4, Web App Display.

2) This web app is only for viewing purposes in which we can see the digital map, total parking spots and the number of parking spots that are available or occupied.

Barrier System and Authorization:

1) In order to enter the parking area, the pi camera would scan your license plate for authorization. This may take 8 to 9 seconds.

2) After the authorization process is completed, if your car is authorized the barrier would open and you would be allowed to enter.

Chapter 7: Deliverables and Cost

7.1 Deliverables

1. Prototype Design for Smart Car Parking System
2. Full-Stack Web and Mobile Application
3. Data Set and Visualization Plots for future expansion
4. Specialized map for parking areas in regards to handicapped/regular student parking.

7.2 Project Plan

Gantt Chart

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		**Smart Car Parking System**	**100 days**	**Mon 2/14/22**	**Fri 7/1/22**	
2		**Software Implementation**	**85 days**	**Mon 2/14/22**	**Fri 6/10/22**	
3		**Web Application**	**16 days**	**Tue 5/3/22**	**Tue 5/24/22**	
4		Real-Time Digitized Map of Parking Area	4 days	Thu 5/19/22	Tue 5/24/22	6
5		Login Page	4 days	Thu 5/19/22	Tue 5/24/22	6
6		Creating User Friendly Interface	12 days	Tue 5/3/22	Wed 5/18/22	23
7		**Mobile Application Development**	**13 days**	**Wed 5/25/22**	**Fri 6/10/22**	
8		Creating Real Time Digitized Map of Parking Area	3 days	Fri 6/3/22	Tue 6/7/22	10
9		Login Page	3 days	Wed 6/8/22	Fri 6/10/22	8

Figure 7.1a, Gannt Chart.

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
10		Creating User Friendly Interface	7 days	Wed 5/25/22	Thu 6/2/22	5
11		**User Authentication**	**10 days**	**Sat 4/2/22**	**Fri 4/15/22**	
12		Image Feature	5 days	Sat 4/2/22	Thu 4/7/22	18
13		Image Database	4 days	Tue 4/5/22	Fri 4/8/22	12
14		Image Processing	5 days	Mon 4/11/22	Fri 4/15/22	13
15		**Databases**	**25 days**	**Mon 2/28/22**	**Fri 4/1/22**	
16		Creating Data	9 days	Mon 2/28/22	Thu 3/10/22	22
17		Updating of Database	5 days	Fri 3/11/22	Thu 3/17/22	16
18		Establishing Connections to Databases	10 days	Mon 3/21/22	Fri 4/1/22	17
19		**Microcontroller Programming**	**10 days**	**Mon 2/14/22**	**Fri 2/25/22**	
20		Sensor Progr	4 days	Mon 2/14/22	Thu 2/17/22	
21		Internet Mod	3 days	Fri 2/18/22	Tue 2/22/22	20

Figure 7.1b, Gantt Chart .

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
22		IoT Programming	5 days	Mon 2/21/22	Fri 2/25/22	20
23		**Gate System Programming**	**3 days**	**Mon 4/18/22**	**Wed 4/20/22**	
24		Entry on Auth	3 days	Mon 4/18/22	Wed 4/20/22	14
25		**Hardware Implementation**	**15 days**	**Mon 6/13/22**	**Fri 7/1/22**	
26		Gate Installation	2 days	Thu 6/30/22	Fri 7/1/22	33
27		Microcontroller Installation	3 days	Wed 6/15/22	Fri 6/17/22	28
28		Sensor Installation	2 days	Mon 6/13/22	Tue 6/14/22	7
29		Server Installation'	3 days	Tue 6/21/22	Thu 6/23/22	31
30		Router Intallatio	1 day	Mon 6/20/22	Mon 6/20/22	27
31		Battery Pack Installation	1 day	Mon 6/20/22	Mon 6/20/22	27
32		Camera Installation	3 days	Fri 6/24/22	Tue 6/28/22	29
33		Display Panel Installation	1 day	Wed 6/29/22	Wed 6/29/22	32

Figure 7.1c, Gantt Chart.

Work breakdown structure

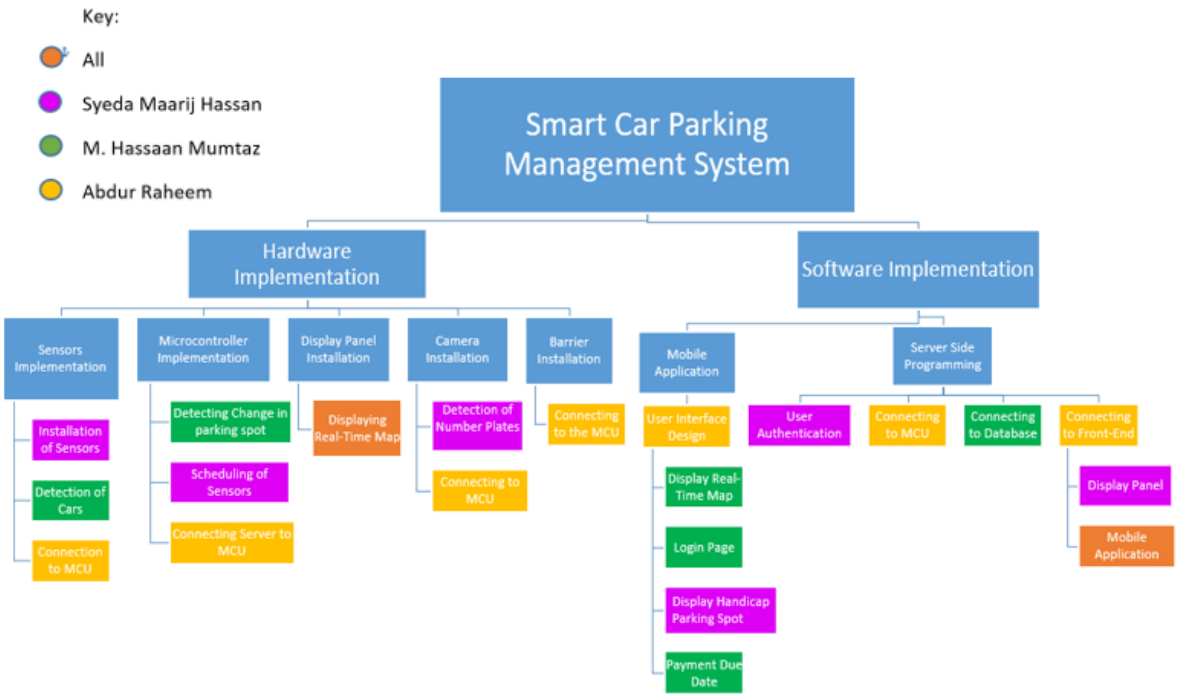


Figure 7.2, Work breakdown Structure.

Project Management Plan

Figure 7.3, Management Plan.

7.3 Project Cost

Device	Quantity	Device Cost per Unit / PKR	Device Cost Total / PKR
Ultrasonic Sensor (HCSR04)	4	150	600
IR Sensor	1	210	210
Nodemcu V3	1	750	750
Raspberry Pi 4	1	37.300	37,300
Servo Motor(Tower Pro SG 90)	1	250	250
Pi Camera	1	6800	6800
Total Cost			45,910

Table 7.1, Project Cost.

Chapter 8: Conclusion

Smart Car Parking Management System is an efficient IoT based parking system, which will help us counter the upcoming challenges faced by drivers on a daily basis. We were successful in achieving the desired objectives of the project including a display of a real-time digital map, mobile and web application, data insights and an authorization and barrier system. The project can further be enhanced using additional security features like a counter to record the duration of stay of each car and an alarm system.

References

[1] Soaibuzzaman, A.S.; Rahman, M.S.; Rahaman, M. A Blockchain-Based Architecture for Integrated Smart Parking Systems. In Proceedings of the International Conference on Pervasive Computing and Communications Workshops, Kyoto, Japan, 11–15 March 2019.

[2] Tsai, M.; Pham, T.N.; Nguyen, D.B.; Dow, C.; Deng, D. A Cloud-Based Smart-Parking System Based on Internet-of-Things Technologies.

[3] Kotb, A.O.; Shen, Y.; Huang, Y. Smart Parking Guidance, Monitoring and Reservations: A Review. IEEE Intell. Transp. Syst. Mag.

[4] Fraifer, M.; Fernström, M. Investigation of Smart Parking Systems and their technologies. In Proceedings of the Thirty Seventh International Conference on Information Systems, IoT Smart City Challenges Applications.

[5] Funke Olanrewaju, R. and Malik Arman, M., 2019. Smart Parking Guidance System on IoT. International Journal of Recent Technology and Engineering, 8(2S11), pp.2793-2798.

[6] Qadri, M.T. & Asif, Muhammad. (2009). Automatic Number Plate Recognition System for Vehicle Identification Using Optical Character Recognition. 335 - 338. 10.1109/ICETC.2009.54.

[7] Gautam, S., 2021. Benefits of Smart Parking: How Smart Parking Reduces Traffic. [online] Parking Network. Available at: https://www.parking-net.com/parking-industry-blog/get-my-parking/how-smart-parking-reduces-traffic

[8] Gautam, S., Pansare, R. and Chaudhary, A., 2021. Benefits of Smart Parking Series: How Smart Parking Reduces Pollution - Get My Parking Blog. [online] Get My Parking Blog. Available at:

https://blog.getmyparking.com/2019/02/21/benefits-of-smart-parking-series-how-smart-parking-reduces-pollution

[9] R. Burnett, “Understanding how ultrasonic sensors work,” MaxBotix Inc, 24-Mar-2020.[Online]. Available at: https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm

[10]T. Pund, H. Sidagam, A. Pinjari, and B. Somnath, “IOT based smart parking system using NODEMCU,” Jetir.org.[Online].Available: https://www.jetir.org/papers/JETIR2112527.pdf

[11]Raspberry Pi Ltd, “Raspberry Pi 4 Model B specifications –,” Raspberry Pi.[Online].Available:https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[12]“Saving data on android, chapter 12: Introduction to Firebase Realtime Database,” raywenderlich.com. [Online]. Available:https://www.raywenderlich.com/books/saving-data-on-android/v1.0/chapters/12-introduction-to-firebase-realtime-database

[13]“Cloud firestore,” Firebase. [Online]. Available: https://firebase.google.com/docs/firestore.

[14]“The Jupyter Notebook — Jupyter Notebook 6.4.12 documentation,” Readthedocs.io. [Online]. Available: https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

[15] FreeCodeCamp.org, “OpenCV Course - Full Tutorial with Python,” 03-Nov-2020. [Online]. Available: https://www.youtube.com/watch?v=oXlwWbU8l2o

[16] Tim Gates, “Imutils”, Github.com. [Online]. Available at: https://github.com/PyImageSearch/imutils

[17] “Tesseract user manual,” tessdoc. [Online]. Available: https://tesseract-ocr.github.io/tessdoc/

[18]“OpenCV: Smoothing Images,” Opencv.org. [Online]. Available: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

[19] Sahir, S., 2019. Canny Edge Detection Step by Step in Python—Computer Vision. [online] Medium.
Available at: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

[20]Randolph Maire, M., 2009. Contour Detection and Image Segmentation. [ebook] Berkeley: University
of California, Berkeley. Available at:
http://www.vision.caltech.edu/mmaire/papers/pdf/mmaire_thesis.pdf

[21]“Python For Character Recognition – Tesseract,” Topcoder.com. [Online]. Available: https://www.topcoder.com/thrive/articles/python-for-character-recognition-tesseract

[22] Altvater, A., 2017. What are CRUD Operations: How CRUD Operations Work, Examples, Tutorials & More. [online] Stackify. Available at: https://stackify.com/what-are-crud-operations

[23]“Data Analysis with Python and pandas using Jupyter Notebook,” Socrata.com. [Online]. Available: https://dev.socrata.com/blog/2016/02/01/pandas-and-jupyter-notebook.html

[24]E. Rovira, “Pandas & Seaborn - A guide to handle & visualize data in Python,” Tryolabs, 16-Mar-2017. [Online].Available: https://tryolabs.com/blog/2017/03/16/pandas-seaborn-a-guide-to-handle-visualize-data-elegantly

[25] M. Harman, “Cleaning and extracting JSON from pandas DataFrames,” Towards Data Science, 19-Jan-2021.[Online].Available: https://towardsdatascience.com/cleaning-and-extracting-json-from-pandas-dataframes-f0c15f93cb38

[26] “Add Firebase to your JavaScript project,” Firebase. [Online]. Available: https://firebase.google.com/docs/web/setup

[27] A. Yudin, “Getting started with react,” in Building Versatile Mobile Apps with Python and REST, Berkeley, CA: Apress, 2020, pp. 65–91. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started

[28] “Expo CLI,” Expo Documentation. [Online]. Available: https://docs.expo.dev/workflow/expo-cli/

Appendices

Complete Source Code:
Github link: https://github.com/abraheem26/Smart_Car_Parking

Raspberry Pi Code(Python)

```
#Section 1
import RPi.GPIO as IO
import cv2
import numpy as np
import imutils
import pytesseract
import time
import os
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
from datetime import datetime
from gpiozero import Servo
from time import sleep
cred = credentials.Certificate("/home/pi/Desktop/Images_Data/dummy-d18d5-firebase-adminsdk-ickgv-9114c9a039.json")
firebase_admin.initialize_app(cred)
IO.setwarnings(False)

IO.setmode(IO.BCM)

INPUT = 6
GPIO_TRIGGER = 24
GPIO_ECHO = 23

IO.setup(GPIO_TRIGGER, IO.OUT)
IO.setup(GPIO_ECHO, IO.IN)

IO.setup(INPUT,IO.IN) #GPIO 14 -> IR sensor as input
servo=Servo(12)


servo.max()
servo.value=None
```

```
def distance():
    # set Trigger to HIGH
    IO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    IO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while IO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while IO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance
while True:
    time.sleep(0.5)
    if(not(IO.input(6)==True)):
#Section 2
        f2=open('pictures.txt', 'r')
        data = f2.read()
        lst=list(data)
        f2.close()
        number=[]
        for word in lst:
            if((word)>='0' and (word)<='9'):
                number.append(word)
        car_number = ([int(elem) for elem in number])
        num=(car_number[0])*10 +car_number[1]+1
        num=str(num)
        update='pic'+num+'.jpg'

        def ratio(percent,image):
            return (image.shape[1]*percent/100)
#Section 3
```

```
        check=cv2.imread(data)
        if(check.shape[1]>670 and check.shape[0]>498):
            img = cv2.resize(check, (800,500), interpolation = cv2.INTER_AREA)
            value=ratio(10,img)
            img=img[180:800,int(value):int(img.shape[1]-value)]
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        else:
            gray = cv2.cvtColor(check, cv2.COLOR_BGR2GRAY)
        im_2=cv2.GaussianBlur(gray,(1,1),0)
        kernel = np.ones((2,2),np.uint8)
        thresh = cv2.erode(im_2, kernel, iterations=1)
        thresh = cv2.dilate(thresh, kernel, iterations=2)
        edged = cv2.Canny(im_2, 30, 200)

        keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
        contours = imutils.grab_contours(keypoints)
        contours = sorted(contours, key=cv2.contourArea, reverse=True)

        location = None
        for contour in contours:
            approx = cv2.approxPolyDP(contour, 10, True)
            if len(approx) == 4:
                location = approx
                break
        mask = np.zeros(gray.shape, np.uint8)
        new_image = cv2.drawContours(mask, [location], 0,255, -1)
        new_image = cv2.bitwise_and(gray, gray, mask=mask)
        (x,y) = np.where(mask==255)
        (x1, y1) = (np.min(x), np.min(y))
        (x2, y2) = (np.max(x), np.max(y))
        cropped_image = gray[x1:x2+1, y1:y2+1]

        further=cropped_image.copy()

        ret, thresh1 = cv2.threshold(further, 199, 200, cv2.THRESH_OTSU |
cv2.THRESH_BINARY)

        rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
        rect_kernel2 = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))

        dilate = cv2.dilate(thresh1, rect_kernel, iterations = 1)

#Section 4
        plate=pytesseract.image_to_boxes(dilate)
        number=[]
        for numbers in plate.splitlines():
```

```
            numbers=numbers.split()
            if((ord(numbers[0])>64 and ord(numbers[0])<91) or (numbers[0]>='0' and
numbers[0]<='9')):
                    number.append(numbers[0])

        plate = ' '.join([str(elem) for elem in number])
        number_plate=plate.replace(" ","")
        if(ord(str(number_plate[0]))>64 and ord(str(number_plate[0]))<91):
            number_plate=number_plate
        elif(ord(str(number_plate[0]))>47 and ord(str(number_plate[0]))<58):
            number_plate=number_plate[1:]
#Section 5
        #FireStore Updating
        db=firestore.client()
        docs=db.collection('people').where("Number_Plate","==",number_plate).get()
        if(docs!=[]):
            for doc in docs:
                people=doc.to_dict()
            if(people['Fee_Paid']==True):
                people['Time_Of_Entry']=datetime.now()
                db.collection('Current_People').add(people)
                servo.mid()
                sleep(2)
                servo.max()
                sleep(1)
                servo.value=None
            else:
                print("No")
                sleep(2)
        elif(docs==[]):
            print("No entry")
#Section 6
        #Updating Number Plate
        f2=open('pictures.txt', 'w')
        f2.truncate(0)
        f2.write(update)
        f2.close()

        cv2.waitKey(1000)

    elif(not(IO.input(6)==False)):#object is far
        print("Not working")

    if(distance()<4):
        f2=open('/home/pi/Desktop/Images_Data/exitImages/picturesExit.txt', 'r')
        data = f2.read()
        lst=list(data)
```

```
        f2.close()
        number=[]
        for word in lst:
            if((word)>='0' and (word)<='9'):
                number.append(word)
        car_number = ([int(elem) for elem in number])
        num=(car_number[0])*10 +car_number[1]+1
        num=str(num)
        update='pic'+num+'.jpg'

        def ratio(percent,image):
            return (image.shape[1]*percent/100)

        check=cv2.imread('/home/pi/Desktop/Images_Data/exitImages/'+data)
        if(check.shape[1]>670 and check.shape[0]>498):
            img = cv2.resize(check, (800,500), interpolation = cv2.INTER_AREA)
            value=ratio(10,img)
            img=img[180:800,int(value):int(img.shape[1]-value)]
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        else:
            gray = cv2.cvtColor(check, cv2.COLOR_BGR2GRAY)
        im_2=cv2.GaussianBlur(gray,(1,1),0)
        kernel = np.ones((2,2),np.uint8)
        thresh = cv2.erode(im_2, kernel, iterations=1)
        thresh = cv2.dilate(thresh, kernel, iterations=2)
        edged = cv2.Canny(im_2, 30, 200)

        keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
        contours = imutils.grab_contours(keypoints)
        contours = sorted(contours, key=cv2.contourArea, reverse=True)

        location = None
        for contour in contours:
            approx = cv2.approxPolyDP(contour, 10, True)
            if len(approx) == 4:
                location = approx
                break


        mask = np.zeros(gray.shape, np.uint8)
        new_image = cv2.drawContours(mask, [location], 0,255, -1)
        new_image = cv2.bitwise_and(gray, gray, mask=mask)
        (x,y) = np.where(mask==255)
        (x1, y1) = (np.min(x), np.min(y))
        (x2, y2) = (np.max(x), np.max(y))
        cropped_image = gray[x1:x2+1, y1:y2+1]
```

```
        further=cropped_image.copy()

        ret, thresh1 = cv2.threshold(further, 199, 200, cv2.THRESH_OTSU |
cv2.THRESH_BINARY)

        rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
        rect_kernel2 = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))

        dilate = cv2.dilate(thresh1, rect_kernel, iterations = 1)
        erode = cv2.erode(dilate, rect_kernel2, iterations = 1)


        plate=pytesseract.image_to_boxes(dilate)
        if(data=="pic29.jpg"):
            plate=pytesseract.image_to_boxes(erode)



        number=[]
        for numbers in plate.splitlines():
            numbers=numbers.split()
            if((ord(numbers[0])>64 and ord(numbers[0])<91) or (numbers[0]>='0' and
numbers[0]<='9')):
                    number.append(numbers[0])

        plate = ' '.join([str(elem) for elem in number])
        number_plate=plate.replace(" ","")
        if(ord(str(number_plate[0]))>64 and ord(str(number_plate[0]))<91):
            number_plate=number_plate
        elif(ord(str(number_plate[0]))>47 and ord(str(number_plate[0]))<58):
            number_plate=number_plate[1:]

        #FireStore Updating
        db=firestore.client()
        docs=db.collection('Current_People').where("Number_Plate","==",number_plate).get()
        if(docs!=[]):
            for doc in docs:
                people=doc.to_dict()
            if(people['Fee_Paid']==True):
                people['Time_Of_Exit']=datetime.now()
                db.collection('Current_People').add(people)
            else:
                print("No")
                sleep(2)
        elif(docs==[]):
            print("No entry")
```

```
#Updating Number Plate
f2=open('/home/pi/Desktop/Images_Data/exitImages/picturesExit.txt', 'w')
f2.truncate(0)
f2.write(update)
f2.close()

cv2.waitKey(1000)
```

NodeMCU Code (Arduino C++)

```
// FirebaseDemo_ESP8266 is a sample that demo the different functions
// of the FirebaseArduino API.



// FirebaseDemo_ESP8266 is a sample that demo the different functions
// of the FirebaseArduino API.

#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>

// Set these to run example.
#define FIREBASE_HOST "dummy-d18d5-default-rtdb.firebaseio.com"
#define FIREBASE_AUTH "GiuXtbp0vwY2pbjFU5nEGf6gLeegOouxpVLNJL4X"
#define WIFI_SSID "HassaanRouter"
#define WIFI_PASSWORD "12345678"

const int trigPin = 14;  //D5
const int echoPin = 12;  //D6

const int trigPin2 = 13;  //D7
const int echoPin2 = 15;  //D8

const int trigPin3 = 5;  //D1
const int echoPin3 = 4;  //D2



long duration;
int distance;
bool car;

long duration2;
int distance2;
bool car2;
```

```
long duration3;
int distance3;
bool car3;


void setup() {
  Serial.begin(9600);

  // connect to wifi.
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("connecting");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("connected: ");
  Serial.println(WiFi.localIP());

  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input

  pinMode(trigPin2, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin2, INPUT); // Sets the echoPin as an Input

  pinMode(trigPin3, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin3, INPUT); // Sets the echoPin as an Input


  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}

int n = 0;

void loop() {

// Clears the trigPin
digitalWrite(trigPin, LOW);

delayMicroseconds(2);

// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);


delayMicroseconds(10);
digitalWrite(trigPin, LOW);
// Reads the echoPin, returns the sound wave travel time in microseconds
duration = pulseIn(echoPin, HIGH);
```

```
// Clears the trigPin
digitalWrite(trigPin2, LOW);

delayMicroseconds(2);

// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin2, HIGH);


delayMicroseconds(10);
digitalWrite(trigPin2, LOW);
duration2 = pulseIn(echoPin2, HIGH);


// Clears the trigPin
digitalWrite(trigPin3, LOW);

delayMicroseconds(2);

// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin3, HIGH);


delayMicroseconds(10);
digitalWrite(trigPin3, LOW);
duration3 = pulseIn(echoPin3, HIGH);


// Calculating the distance
distance= duration*0.034/2;

distance2= duration2*0.034/2;

distance3= duration3*0.034/2;

Serial.print("Distance1: ");
Serial.println(distance);

Serial.print("Distance2: ");
Serial.println(distance2);

Serial.print("Distance3: ");
Serial.println(distance3);


//first distance
if(distance <10){
  car=true;
```

```
}

else{
  car=false;
}
//second distance
if(distance2 <10 && distance2>0){
  car2=true;
}


else{
  car2=false;
}

//third distance
if(distance3 <10 && distance3>0){
  car3=true;
}


else{
  car3=false;
}


  // set bool value
  Firebase.setBool("Parking_Spots/Spot_1/availability", not(car));
  Firebase.setInt("Parking_Spots/Spot_1/index", 1);

  Firebase.setBool("Parking_Spots/Spot_2/availability", not(car2));
  Firebase.setInt("Parking_Spots/Spot_2/index", 2);

  Firebase.setBool("Parking_Spots/Spot_3/availability", not(car3));
  Firebase.setInt("Parking_Spots/Spot_3/index", 3);

  if (Firebase.failed()) {
     Serial.print("setting /truth failed:");
     Serial.println(Firebase.error());
     return;
  }
  delay(1000);
}
```

Glossary

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CNN	Convolutional Neural Networks
CO2	Carbon Dioxide
CRUD	Create Read Update Delete
CSS	Cascade Styling Sheet
CV	Computer Vision
DC	Direct Current
GB	Giga Byte
GPIO	General Purpose Input/Output
GPS	Global Position System
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IR	Infra-red
JS	Java Script
JSON	JavaScript Object Notation
MCU	Microcontroller Unit
ML	Machine Learning
NUCES	National University Of Computer and Emerging Sciences
OCR	Optical Character Recognition
RFID	Radio-frequency identification
SQL	Structured Query Language
UI	User Interface
VCC	Voltage Common Collector