

PAY AS YOU PARK SMART PARKING SOLUTION

B.Sc. (Hons) Information Technology – Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

September 2021

PAY AS YOU PARK SMART PARKING SOLUTION

Dissertation submitted in partial fulfillment of the requirement for the Bachelor of
Science in Information Technology specialization in Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

September 2021

DECLARATION

I declare that this is our own work and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
M.D.S.M. Antany	IT18012552	
Aadil M.R.M	IT18013092	
Ferreira L.V.	IT18013924	
Priyankara A. D. D.	IT18154672	

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

Signature of the Supervisor:

Date

DEDICATION

This research is dedicated to all the people who daily travel in the road who facing many difficulties in finding a better parking yard to park. In urban cities parking is an essential task. People face to this problem in their day-to-day life.

ACKNOWLEDGEMENT

We would like to present my heartfelt thanks for Ms. Nadeesha Pemadasa (Assistant Lecturer – Faculty of Computing SLIIT) for the support, guidance and motivation throughout the research as our supervisor for make it successful and effective research.

Also would like to thank Ms. Thamali Dissanayake (Lecturer – Faculty of Computing SLIIT) for the support, guidance and motivation throughout the research as our co supervisor for make it successful and effective research.

ABSTRACT

"Pay as You Park" is automation parking system which is developed to reduce the issues in Sri Lankan Parking system. There are four main objectives in the project. One of the most significant tasks before deciding on a parking location is to assess the suitability of the land. As a result, consumers must spend more time and money to seek counsel from experts. In the real-world situation of a vehicle parking solution, registering and validating the appropriateness of new land for a car parking application would be difficult for the system administrator in his hectic daily routine. Furthermore, when looking at the world map, there are millions of acres and garages owned by people in various regions. So, once registration begins following the deployment of a Parking application, and if this process is maintained by a human, there will be a massive line for private land and garage registration and verification processes. As a result, in order to address these issues, this study will offer a machine learning and image processing-based automated solution. Using this method, the parking system can analyze the parking owner's image data of the parking land/garage that was taken during the registration process and provide the suitability level of the ground as well as the necessary suggestion to resubmit images after the parking lot owner has made changes. The "Pay as You Park" smart parking system can be used to solve the day-to-day parking problems that individuals encounter nowadays. Our solution "Pay as You Park" Smart parking system uses already employed surveillance cameras to identify the availability of parking slots inside a parking area. For drivers, parking in cities is the most challenging issue. The number of automobiles on the road has gradually increased as a result of the large population. As a result of the scarcity of parking places, most vehicles have a tough time finding a suitable parking spot. The proposed method provides an optimal and accurate advice for drivers to choose the suitable parking yard for their vehicles based on distance, vehicle physical characteristics, and the availability of open spaces in that parking yard. As a result of the system's prediction of availability and distance, drivers will receive the most optimum parking solution. This system also provides a service that directs customers

to the appropriate parking lots. The suggested system has an environmental impact due to the reduction of fuel waste and the emission of carbon dioxide. Furthermore, the proposed solution will save consumers time wasted on the streets. Although there are many both inside and outside parking spots in Sri Lanka, an urban location like Colombo currently faces a massive traffic bottleneck due to inefficient car parking, the number of automobiles utilized in the country by people, and other factors. When it comes to automobile parking in Sri Lanka, people are sometimes unable to find a free parking slot, and even if they do find a slot, it is difficult to find the exact location of the parking slots. Next research components is internal parking navigation. The system is built on BLE Beacons to discover the path to a specific parking spot, and users can find their way to the reserved parking slot via a smartphone application. Three beacons will be utilized to track the location of the user (Triangulation). Users can utilize this smart parking system to find their way not only to the parking area, but also to a parking space that is not reserved/free within the parking area. Users only need a Bluetooth-enabled mobile device with adequate network connectivity.

Table of Content

DECLARATION	i
DEDICATION	ii
ACKNOWLEDGEMENT	iii
ABSTRACT	iv
Table of Content	vi
List of Figures	viii
List of Tables	xi
List of Abbreviation.....	xii
1. INTRODUCTION	1
1.1 Background and Literature Survey	2
1.1.1 Background	2
1.2 Research Gap	11
1.3 Research Problem.....	14
2.1 Main Objectives	16
2.2 Specific Objectives	17
3 METHODOLOGY	19
4 TESTING AND IMPLEMENTATION.....	34
4.1 Implementation.....	34
4.1.1 Mobile Application Implementation	34
4.1.2 API Server Implementation.....	42
4.1.3 Parking Yard Suggestion Implementation.....	43
4.1.5. Parking Surface Quality Detection Model Implementation	51

4.2 Testing.....	72
5. RESULT AND DISCUSSION	75
5.1 Results.....	76
5.2 Research Findings.....	78
5.3 Discussion	79
6. SUMMARY OF THE STUDENT CONTRIBUTION	81
7. CONCLUSION	84
8. REFERENCES	86

List of Figures

3.1 Component Architecture	20
3.2 ACF and PCF Plot	23
3.3 Component Flow	25
3.4 Component Architecture	26
3.5 Region of Interest.....	28
3.6 Flow Chart of Component	29
3.7 The structure of Internal Navigation	30
3.8 Beacon Triangulation	31
3.9 Workflow Internal Navigation	32
4.1 Initial Map Load	36
4.2 Tracking User Location	37
4.3 Implementation of Location Tracker 1.....	38
4.4 Implementation of Location Tracker 1.....	39
4.5 Direction Interface	40
4.6 Implementation Google Map Widget	41
4.7 Availability View UI	42
4.8 Harvesine Implementation	43
4.9 Stationary Test	45
4.10 Sarima Implementation	46

4.11 Component Implementation	47
4.12 Model Implementation 2	47
4.13 Neural Network Structure.....	48
4.14 Register Parking yard Owner	49
4.15 Register Parking Yard	50
4.16 Trained Data folder Categorizing	52
4.17 Train.py code part 1	52
4.18 dataset.py code part 1.....	53
4.19 dataset.py code part 2	54
4.20 Train.py code part 1	55
4.21 Train.py code part 2.....	55
4.22 Train.py code part 3.....	56
4.23 Train.py code part 4	56
4.24 test.py code part 1	57
4.25 test.py code part 2	58
4.26 Surface quality classification folder structure	59
4.27 Restoring the surface type model	59
4.28 Restoring the asphalt quality model	60
4.29 Restoring the paved quality model	60
4.30 Restoring the unpaved quality model	60

4.31 Main.py code part 1	61
4.32 Main.py code part 2	62
4.33 Main.py code part 3	62
4.34 Main.py code part 4	63
4.35 Flask Implementation	64
4.36 Sample Response.....	65
4.37 Node.js Model.....	65
4.38 Node.js Model.....	65
4.39 Car Park Owner Parking Availability Preview.....	66
4.40 Training Model.....	66
4.41 Occupancy Sample.....	66
4.42 Identifying Availability.....	66
4.43 Boundary boxes on car parking slots.....	66
4.44 Identifying Occupancy.....	67
4.45 Output.....	67
5.1 SARIMA Graph	77
5.2 SARIMA Results	77
6.1 Student Contribution Summary	83

List of Tables

2 Test Results Summary	73
2.1 Testing Issue Clarification	74
2.2 Parking surface type model accuracy	74
2.3 Parking surface quality model accuracy.....	75
2.4 Accuracy of slot identification.....	75
2.5 Testing of slot identification.....	75
5.3 Model Accuracy	78

List of Abbreviation

DNN	Deep Neural Network
NN	Neural Network
RSSI	Received Signal Strength Indicator
ToA	Time of Arrival
TDoA	Time Difference of Arrival
SARIMA	Seasonal Autoregressive Integrated Moving Average
ARIMA	Autoregressive Integrated Moving Average
API	Application Programming Interface
UI	User Interface
UX	User Experience
CNN	Convolutional Neural Network

1. INTRODUCTION

This paper introduces a very good framework for smart parking concepts to cross over the limitations and barriers in smart parking. In current model world concept of smart parking are very popular in urban cities. Along with smart cities with the growth of Artificial Intelligence and Information Technology smart parking solutions are implemented to provide a smart experience for the users in parking process. However, in this paper we introduce a framework to optimize the parking process and minimize the barriers in parking. Parking is an essential task that needs to be performed by users when they move away from their residence and with the limitation of parking spaces users are used to park their vehicles in parking yards as they are concern about security of their vehicles. In this paper it is focused on both the parking yard owners and drivers. With our application it provides users the functionality to ease the parking process and for the parking yard owners it provides a parking management application. It provides a mobile application for the drivers to perform their parking and a web application for parking yard owners. With the introduced mobile application users can get accurate parking suggestions to park their vehicle including the internal navigation to a free parking slot of a selected parking yard. Using this application users can optimize their parking process and save their valuable time.

With the systems we provide our expectation is to provide users accurate and optimized parking experience. The system provides functionality to search for parking and navigate to parking yards and also navigate to slots inside parking yard. Also, system provides users a subscription-based payment approach with pay as you park concept which they have to pay only for the time they have parked. Also, it provides options to register parking yard for the yard owners to the system which would go through an automated validation process as it has to ensure the protection of users. Both the drivers and parking yard owners will be benefited with the system introduced in this paper.

1.1 Background and Literature Survey

1.1.1 Background

As the number of vehicles increase day by day the competition for parking in urban cities also rapidly increased. And when it comes to parking with existing solutions it has been identifies that most of the users are still facing problems related to parking. In the current modern society people are used to park their vehicles in parking yards as there is a limitation in public parking and it is more secure than the public parking.

Though the smart city concepts developed drivers are still facing the problem of finding an optimal parking yard to park their vehicle. This would affect the society and environment in a negative way. When drivers are unable to find the parking yard to park, they may become stressful, and this will lead to road accidents and unusual traffic situation in urban cities. Also, when drivers travel in the roads reasonlessly to find a parking yard it would waste the fuel and cause air pollution. Drivers may use to park their vehicle unethically and illegally when they could not find a proper parking yard to park. Also, for the internal parking users are unaware where is the free spot is located, so in that scenario also user must navigate within the parking yard searching for a free slot.

Effective parking can solve the most problems faced by the people and the environment in busy cities. With the rapid growing rate of population, it is predicted that problems with parking also get increased. A fine solution for these problems would be a smart parking solution using the smart concepts. For a better livelihood of residence in urban cities this smart parking will be affect in a positive way. With the presence of optimal parking suggestion in a smart parking solution it is aimed to solve the social issues present in the urban cities. When we consider about the parking yards

exists there may be inappropriate parking yards which are not suitable for parking purpose. Users may face different problems and may get their vehicle damaged. There must be proper validation mechanism to approve a parking yard is suitable for parking or not. Also, tracking of free parking slots inside a parking yard is an expensive process for the parking yard owners. Most parking solutions use sensor-based architecture for each slot. And yard owners have to invest a huge amount for those sensors. Minimizing the cost for the tracking of free slots inside a parking yard would be beneficial for the parking yard owners.

In fact, people face different problems when they perform parking and the existing application has a huge gap towards what the end users are expecting. In Sri Lanka there is a remarkable wastage of fuel and air pollution in urban cities. It can be minimizing those issues with introducing proper parking system to the urban cities. It would impact on unusual traffic, air pollution, wastage of fuel and other social issues in a positive way.

People spend a lot part of their daily lives indoors. Locations such as schools, colleges, workplaces, hospitals, shopping, Shopping centers, etc. Indoor positioning is a trivial problem, as satellite-based approaches (E.g. GPS) do not work properly within buildings. Wi-Fi, ZigBee, RFID and BLE are more for indoor positioning research, favored.

1.1.2 Literature Survey

Many researches have been conducted to locate the nearest location of user. In research conducted to locate the nearest police station, they have used Global Positioning System (GPS) device to identify the location of user. It works in any weather if the device has a clear line of sight to the satellites [2]. It indicates the GPS technology got a high availability in global. According to many authors GPS is a widely used technology to locate the current position of user, as in current smart world each one got a smartphone on their hands, it will be able to reach many users. A-GPS, Assisted Global Positioning System, works on the same principles as the GPS [2]. The main difference between these is A-GPS obtain the user coordinates via assistant servers while GPS coordinates with satellites directly.

Google Map is a service provide by Google; many researchers use this tool to digitally visualize the locations to the users. Just after the closest location is found, the coordinate of this location is sent on Google Maps [1]. And Google Map API provides different service to integrate with the map. Direction to the location service API is used by many authors to direct user to the destination. Google provides several APIs related to google maps. For this scenario Distance Matrix API which provides the distance and the time to reach data can be used the algorithm. Also google provide the Google Map SDK for both android and IOS mobile operating systems.

Authors have been many Machine Learning algorithms to find the nearest parking location to the user. They have used the user current location coordinates which retrieve by using GPS and A-GPS. In research conducted to identify the nearest medical service provider, they have used algorithm called TOPSIS to get information about nearest medical service provider. TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution) algorithm is a multiple-criteria decision-making algorithm [3].

The Haversine theorem is used to calculate the lengths of two points on the surface of the earth based on latitude and longitude [4]. This research was conducted to find the nearest mosque for a user. Author have used Haversine theorem to identify which is the nearest to the user. Haversine algorithm mostly used to predict the distance between two coordinates, computational cost remain minimum for this algorithm compared to other algorithms used for same purpose. Also, in haversine algorithms it assumes that Earth is spherical but in reality, it is elliptical. Although this assumption is with a minimum error regarding the distance in order to find the shortest distance Haversine still remain as one of the best algorithms with low computational cost.

Ahuja-Dijkstra's Algorithm is to calculate for routing analysis to find the shortest route to reach the nearest facility [5]. In the reading it is noticeable that Ahuja-Dijkstra's Algorithm is mentioned by majority of authors in order to find the shortest path from current location to destination.

In order to predict the occupancy of a parking yard at a given time since parking yard occupancy data possess a seasonality SARIMA can be used as the prediction model since it is widely used for seasonal time series forecasting. Authors of research [6] have compared the timeseries forecasting models and revealed that SARIMA performs better for datasets with seasonality. ARIMA is a model which perform with autoregressive and moving average processes, and SARIMA is advanced with the seasonal package.

As per the above mentioned reading main component of a smart parking system is identifying the most appropriate parking yard to park. By suggesting optimal solution to park, it will be benefit for both users and the environment.

There are many parking applications available worldwide in different regions but no one is providing a space for registering new places (e.g., private car parks, lands, garages, etc.) as parking slots on the map. In urban areas, Car parking is a serious issue

in both developed and developing countries. Many cities are suffering from a lack of vehicle parking places as a result of the fast rise in car ownership, with an imbalance between parking availability and demand, which may be regarded as the root cause of metropolitan parking issues [8]. Considering the virtual parking solution, it is impossible to manage private properties to use as parking spaces in public use. So that “Pay as you park” smart parking solution introducing a new feature to overcome the lack of parking facilities in urban areas by standardizing the private places for public use. So, the “pay as you park” team has researched the best technology to identify the standard of the private places for parking vehicles before it registers to the “Pay as you park” system.

In this case, detecting potholes and identifying the surface type and the quality of the parking place is the major task of this system when registration of the lands. Otherwise, drivers may encounter a variety of issues when parking in those new locations, causing them to lose faith in new parking places provided by the solution. Many types of research have implemented sensor-based systems [9] and machine learning approaches [10][11] to locate potholes, objects, and surface quality. Different authors have been used many machine learning and image processing approaches for different problems such as soil classification, pothole detection, surface cracks detection, etc.

Regarding related work, group of researchers present a literature review about recent publications for soil type classification. In this approach they developed a model an image processing-based soil classifier to identify the soil type of the ground. In that model, they have used seven classes of soil for classification and the preprocessed images have extracted the features, and the data extracted is used to train the Support Vector Machine (SVM) classifier [11].

In another research, CNN-based road surface crack detection model was developed which response to the brightness changes. In that case they come up with a semantic segmentation model with an autoencoder structure for detecting road surface along

with a CNN-based image preprocessing model as a solution to the road surface crack detecting under the different lightning conditions of the images [12].

Another experiment was carried out by a group of researchers to find the best accurate model for monitoring road surface and detecting the pothole. In that work, they have analyzed and applied different deep learning models: convolutional neural networks (CNN), 4 Long short-term memory network (LSTM) networks, and reservoir computing (RC) models. At the end of the research, they have shown the confusion matrices and accuracy for each evaluated model. And they proved that in contrast to CNN, the other models failed to classify the rest of the classes and also the most potholes were correctly classified by itself [13].

There are many kinds of indoor positioning in world-wide. Some of them are,

- Wi-fi based indoor positioning
- BLE Beacon based indoor positioning
- UWB based indoor positioning
- RFID based indoor positioning

Below has the brief description of the technologies,

a. Wi-fi Based Indoor Positioning

Wi-Fi-based real-time indoor positioning systems, such as smartphones, tablets, and Wi-Fi tags, locate and track active Wi-Fi devices. Depending on the preconditions, the accuracy of Wi-Fi used for server-based indoor localization ranges from eight to fifteen meters. [14]

Advantages :

Capability to control all Wi-Fi-enabled devices, ability to track visitor behavior, wide range of devices (up to 150m).

Disadvantages :

The degree of accuracy offered by BLE or RFID, high latencies and randomized MAC address usage are difficult to achieve when the system is not connected to the Wi-Fi network.

b. BLE Beacon Based Indoor Positioning

Beacons are small wireless devices that use Bluetooth Low Energy, also known as Bluetooth Smart, to relay signals. They are relatively inexpensive, can run for up to five years on button cells. With Bluetooth 4.0.0, accuracy is usually less than eight meters. The new 5.1 version of the Bluetooth specification allows for path finding use cases and offers less than one meter of accuracy. Beacons are scalable and highly portable in all sorts of different formats.[14]

Advantages :

Flexibility and Efficiency of Cost

Disadvantages :

Signal dispersion attenuations inside houses, layout alteration instability and radio interference.

c. UWB Based Indoor Positioning

Ultra-wideband is a radio technology with a short range. The precision is less than 30 cm, which is slightly higher than when dealing with Wi-Fi or beacons. You can also reliably calculate height differences.[15]

Advantages:

High accuracy, low latency times, no interferences

Disadvantages:

Higher cost and battery life is less than beacon

d. RFID Based Indoor Positioning

RFID is a type of wireless communication that identifies objects through radio waves. Passive RFID technology only operates close to specialized RFID readers providing a 'point-in-time' role.[16]

Advantages:

Very high accuracy, no battery needed

Disadvantages:

Short range signal transmission, installation need significant planning and infrastructures can be expensive.

There have been several occasions where sensor technologies were used for current parking systems in Sri Lanka and a considerable number of researches have been conducted on sensors and micro-controller-based car parking solutions to identify the availability of a car parking slot. Few of the above-mentioned technologies are,

- Ultrasonic sensors
- Magnetometers
- Infrared Sensors
- LDR - Light Dependent Resistors.

There have been various problems arisen while using these technologies since Sri Lanka has rainy weather condition throughout the year.

1.2 Research Gap

According to our research we have identified for main sections that affect the efficiency of parking. We individually focused on those components and in order to introduce the solution.

First Component is to identify the free slots inside the parking yards. During our site visits to several car parking areas, we have experienced several occurrences which displayed inaccurate sensor data. In some cases, we have driven to the particular parking slot which showed as available and once we reached there, we have seen a vehicle has been already parked there. This proposed "Pay as You Park" smart parking system provides a better user experience by saving both time and money for the user while reducing congestion. Because a single CCTV camera can cover a large area, it saves the cost of having a single sensor every single parking space. This also boosts the system's maintainability by reducing the number of sensors required. The data on available parking spaces and used parking spaces will be provided to the next component, which will direct the user to the appropriate parking spot. This approach solves several major problems in current car parking systems.

Second Component is to suggest optimal parking yard for the drivers, during the research that we have conducted it has been identified the key factors that impact on parking yard availability and suggestions

- Distance
- Availability
- Physical characteristics of vehicle
- Time taken to reach the destination

Existing applications does not consider on all these factors, and the optimality of their suggestion is very low. In our propose solution we will be provide suggestions to the users considering the all those factors and the suggestions will be accurate and optimal.

"Pay as You Park" is the system going to be implemented in the research project. In this system, users are able to find the way to the indoor parking slot, and the proposed indoor navigation technology is BLE beacon-based indoor positioning. At the heart of the indoor positioning market is Bluetooth. The technology itself is not new; since the 1990s, Bluetooth's functionality has been well-known. But it was the emergence of the BLE version of energy-saving Bluetooth that paved the way for many new application scenarios, making Bluetooth an industry standard available today on most devices.

Since GPS does not run indoors, Bluetooth is a strong alternative, and the go-to option for indoor navigation applications is BLE beacons. Beacons can send out Bluetooth signals, but beacons are not able to transmit them. Beacons are relatively inexpensive, can run up to several years on button cells, and have a maximum range of 30 meters indoors (Ranges are different from Beacon series to series) but in present industry has discovered new beacons with 100m accuracy. A location accuracy of up to one meter can be achieved. Positioning happens on the user's mobile device in client- based solutions, ensuring optimal data security. There is a need for an app and Bluetooth must be enabled. Also possible is a server-based solution (asset or individual tracking) using beacons.

Several beacons are needed for positioning in client-based applications. Without much effort, beacons can be glued or screwed to ceilings or walls and can be easily incorporated into any environment.

There are two Range based schemes. Distance Estimation and Position Estimation are two of it. Research attempts have been made in the literature to reduce the distance error and improve the accuracy of the location. The centroid localization algorithm proposed by uses the coordination of beacons to estimate the location of the

unidentified mobile positioning using the centroid formula, but with this algorithm, the position accuracy is very poor. The algorithm for Weighted Centroid Localization uses weight to estimate location as a factor. The AWBCL algorithm based on the WCL algorithm has improved the accuracy of the spot, but the position error is still high.

RSSI based and ToA/TDoA-based are distance estimation schemes. Below are the pros and cons of the distance estimation methods.

Pros and Cons of RSSI Based :

Pros: * Low Complexity

* No Time Sync

* Low Power

Cons: * Low Accuracy

Pros and Cons of ToA/TDoA Based:

Pros: * High Accuracy

Cons: * High Complexity

* Time Sync

* High Power

Fourth component is to validate the parking yard registered

The main tasks of this system when registering lands and entrances in this scenario are detecting potholes and evaluating the surface type and quality of the parking lot. Otherwise, drivers may face a variety of difficulties when parking in those new areas, leading them to lose trust in us. Many studies have used sensor-based systems to identify land potholes [20], but these are ineffective in our situation because the entire procedure should be carried out through the internet. As a result, using image processing and machine learning-based technologies for verification is the greatest choice for dealing with this problem [21][22].

Despite the fact that several research have been conducted on road pothole recognition and surface type detection, none have combined the two aspects. Using surface type classification and identification algorithms, we must also identify the parking land surface type and anticipate circumstances in different seasons across the country. As a result, a verification procedure must be carried out.

1.3 Research Problem

In the present society, parking is not happening according to a proper system or way. especially in Sri Lanka most of the drivers park their vehicles on both sides of the roads. Due to that, the traffic jam is getting increased and also the valuable time is wasting. Not only that but also it may result in an unusual environment and air pollution. Even though there are available free parking yards some people are not able to find those parking yards, there is no proper navigation to the parking yard. Also, navigation inside parking is very hard due to GPS does not give accuracy inside a parking yard or underground because of the obstacles like walls, ceiling, etc. From in drivers' point of view, they may not like to park their vehicle in a parking yard that does not have a proper surface. In some parking yards, there can be a rough surface that is not suitable for parking a vehicle.

By considering all those issues we have introduced a system which solves the all above issues. In the introduced system we have implemented a Parking yard quality validation system. Also, we have implemented a system to find free parking yards by using cctv images and suggesting the nearest optimal parking yard for the users. And finally, when a user enters the parking yard user able to see the available free parking slots inside the parking area.

2 OBJECTIVES

2.1 Main Objectives

In the “Pay as you park” smart parking solution we mainly focus on the user-friendly automated smart parking solution for both drivers and parking yard owners. When considering those factors, we wanted to reduce the problems of the drivers and parking yards owners' problem which they currently facing in day today life. Most of drivers are get in to troubles when they come to the urban cities because lack of parking, unable to find quality parking yard, unable to find the quality and most optimal parking yard for the vehicle and place he visit, wasting time and fuel when finding the empty parking spot in the yard, when finding the empty parking slot at peak time without entering to the parking and also the current unfair charging system of those places. And when considering the parking owner's' aspects, that user role also has to face lot of difficulties with current situation in parking places because of not having a proper smart parking solution. They have to manage their payments themself, registering takes long time for the new system, there are no any method to evaluate parking yard standard so lot of drivers and owners face many troubles when long term run the parking places without maintaining.

Therefore, we have focus on many objectives when researching the smart parking solution. We introduced some of special features to cover up above problems;

1. Availability checking of the parking slot with CCTV footage
2. Suggest the optimal parking yard to the drivers fot their vehicle
3. Internal navigation system until the end of the parking spot
4. Parking yard surface quality validation and register new places

Except those special objectives we have introduced time-based package to park in the places, Parking place management system, Parking place monthly quality evaluation system and many more sub features such as better cross platform user experience both IOS and android users.

2.2 Specific Objectives

- Tracking available parking spaces

This system identifies the availability of parking slots of a parking area using CCTV footages of a parking yard. Using this system, it finds the empty parking lots and indicate those places to suggest the places to end user with connecting the module of optimal parking yard suggestion. The data of the availability is sent to the system for analyzing for the further process in “Pay as you park” system.

- Optimal parking yard suggestions

‘Pay as you Park’ smart parking system identify and suggest the most optimal parking yard for a vehicle based on the current location, availability of free spaces in parking yards, physical characteristics of vehicles like (height, width, length) and the time taken to reach the parking yard.

- Internal Navigation

The 'Pay as you Park' smart parking technology directs users when a user drives into the parking area by providing the available parking slots on a map to drive the automobiles to the available parking space. Although interior navigation is not precise due to barriers such as walls and ceilings, this technique is done by using a low-cost and compact gadget known as a Beacon. Beacons are less expensive than other indoor positioning systems, and they provide far more accuracy than other location technologies such as Wi-Fi.

- Parking yard surface validation

Land suitability evaluation is one of the important procedures before set that place for car parking. So that people have to waste more time and money for that to take advice from their expertise. When considering the real-world scenario of a car parking solution, registering and verifying the suitability of new land for a car parking application would not easily process for the system admin in his busy daily routine. And also, when considering the world map there are millions of lands and garages are owned by people in different regions. So, once the registration begins after a Parking application deployment and if this process maintained by a human being, there would be a huge queue for private land and garage registration and verification processes. Therefore, in order to overcome these problems, machine learning and image processing based automated method is going to introduce in this paper. Using this method, the parking system can analyze the parking owner's image data of the parking land/garage which was taken while the registration process and provide the suitability level of the ground and the necessary suggestion to resubmit images after changes done by the parking lot owner.

3 METHODOLOGY

Identify the availability of car parking slots

To identify the availability of the car parking slots which means the occupancy of a car parking slot, we have used ImageNet-VGG-f model, a model which has been trained on the ImageNet dataset. This Pre-trained deep CNNs have five convolutional layers, each of which has 11x11, 5x5, 3x3, 3x3, and 3x3 image kernels. These layers stride over the entire image, pixel by pixel (except for the first layer, where the stride is four pixels), to generate 3D volumes of feature maps (with the exception of the first layer, where the stride is four pixels). The first convolution layer has a width of 64 pixels, while the remaining layers have a width of 256 pixels. Following the first, second, and final convolution layers, there is a max-pooling layer. The last convolution layer is followed by three fully connected layers of 4096, 4096, and 1000 neurons, respectively, and the final output is a layer with a soft-max classifier as its final output. It is fairly similar to the network architecture illustrated in Figure 1 in terms of its design. Figure 2 depicts a simplified representation of the framework, which consists of two main components: training an SVM classifier and evaluating the classification results of the SVM classifier.

Component Architecture

CCTV captures the footages of the car parking area and the processed images are being further processed using CNN algorithm to identify the availability of the parking slots inside parking area. As the identification process it calls the update service of availability which was developed using Express framework with node.js environment. There are basically 2 services. First service provides the user to get the availability of the parking slots and second service is used to update the services. Both services runs on cloud platform.

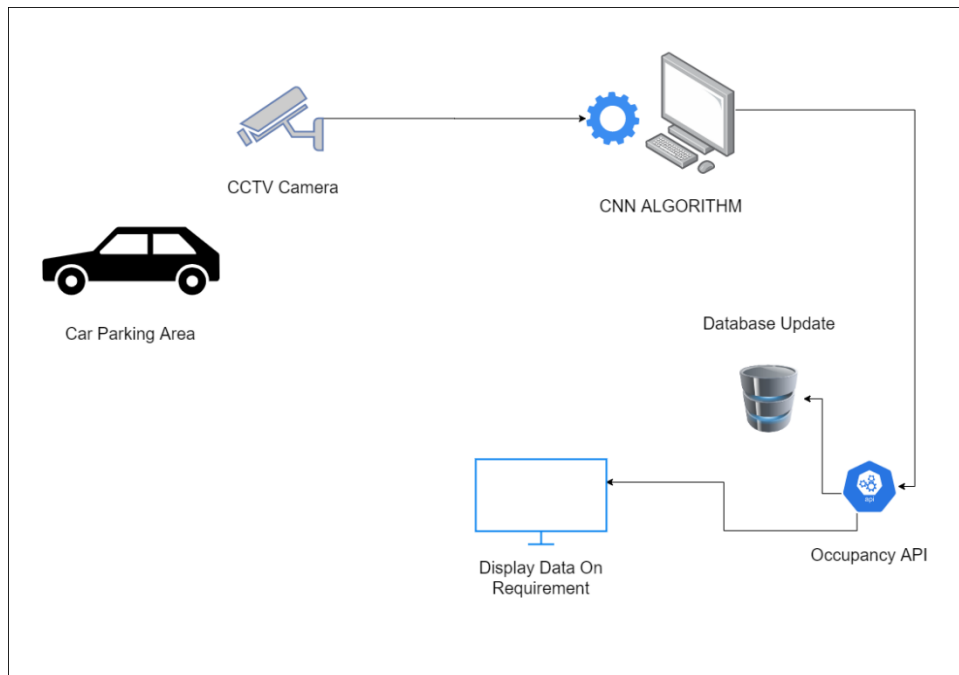


Figure 3.1 Component Architecture

Optimal Parking Yard Suggestion:

In the very beginning of our research, we found that parking systems in Sri Lanka does not provide optimal parking suggestions for the users. And most of the users are facing hard to find a correct parking yard to park their vehicle. So, we decided to implement a solution for this problem with modern existing technology.

First, we wanted to find out the users' thoughts on the current existing parking suggestions in Sri Lanka, so we done a survey on that, and it proved us that most of the users are unhappy on the existing parking suggestions. In that decided to implement an algorithm with high accuracy and optimality.

Since the solution need a portability and availability, we decided to provide a mobile application to users with the functionality of parking yard suggestions. Flutter is used

for mobile application development as it supports cross platform mobile development, and it is currently in the trend of mobile application technology.

- **Get the availability**

In our research the parking yard availability is captured by the cameras in the parking yard in another component and it provide the live availability of a parking yard. Using the integrated API with node.js it can gain the availability of that time. For get the availability the API will be used.

- **Considering the physical characteristic of vehicle**

As discussed earlier, sometime physical characteristics of a vehicle may restrict the vehicle from entering or parking in some parking yards. It is identified that length, width and height of a vehicle are characteristic that cause the restriction. As a solution to the problem, we collected those data from parking yards and store them in the database. In the process of parking yard suggestion through the API services used in algorithm we fetch only the valid parking yards for vehicle. It make sure only the valid parking yards are suggested to vehicle which can perform the parking.

- **Identifying the nearest parking yard:**

To find the nearest parking yard to user it is needed to be calculated the distance to parking yards and order them to process further. Haversine is a widely used algorithm to calculate the distance of two location points was firstly published by James Andrew. Given the longitudes and latitudes of two points on a sphere, it calculates the great-circle distance between them. It is a specific example of the law of haversines, a more general formula in spherical trigonometry that

connects the sides and angles of spherical triangles. Haversine provide us the accurate result with least computational cost.

- **Predict occupancy when user reach parking yard**

Predicting parking yard occupancy when user reach the destination is a main section of this research. In order to get the prediction, we need to get the time taken to reach the destined parking yard. For that we will be using the Google map API platform provided by the google. We have discovered that the Distance Matrix API for this functionality which provide us the duration, distance and other matrix. With the use of Distance Matric API then the time is processed and taken the reach time for the destined parking yard.

It is needed to be predict the occupancy of the parking yard within the time taken to reach the destined parking yard. Since the parking data consist of a seasonality with it. SARIMA a time series forecasting model is used to predict the occupancy of a parking yard when user reach the destined parking yard. SARIMA is an extension of ARIMA which support seasonality. SARIMA is widely used in the industry to forecast the univariate dataset with seasonal component. SARIMA fit well for the model since the parking dataset possess a seasonality with it. In order use the SARIMA fist, it needed to be identified the autocorrelation and partial autocorrelation of dataset. This can be achieved by plotting the correlation between the occupancy and that the same measures of X period before. Autocorrelation process a single lag while partial correlation processes a moving average.

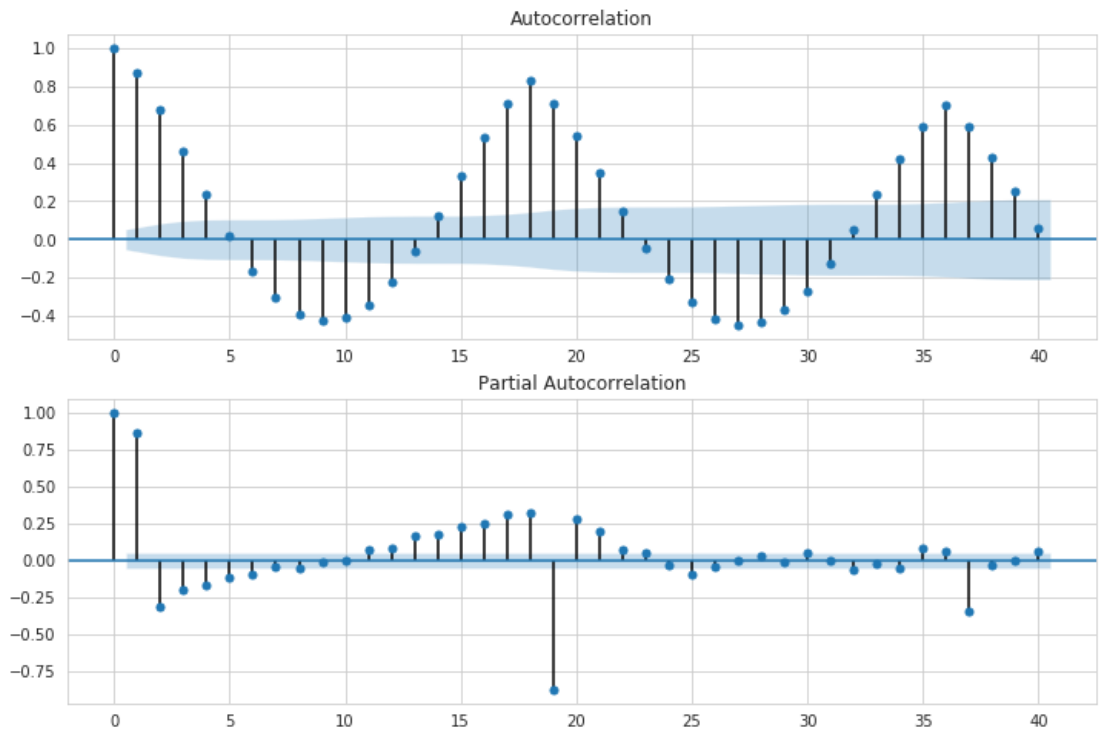


Figure 3.2 ACF and PCF plot

The autocorrelation and partial correlation graphs in Figure 3.2 show that there is a pattern in the data set of occupancy. SARIMA can be applied to this scenario and predict the accuracy by forecasting.

- **Workflow of the component**

As a solution for the issues identified in parking yard suggestions, we come up with an algorithm considering all the key factors that impact on parking yard suggestions. The component flow diagram *figure 3.3* given below demonstrate how the parking yard suggestion algorithm works. With the use of that algorithm, we provide the users optimal parking yard suggestions.

The respective algorithm is deployed with fast API which is a widely using API in the industry. Users' location coordinates and vehicle's physical characters are needed to provide as input parameters and parking yard and location matrix are return as return types. Those return matrix are used in mobile application level along with Google maps route API and GPS services the results are visualized in the mobile.

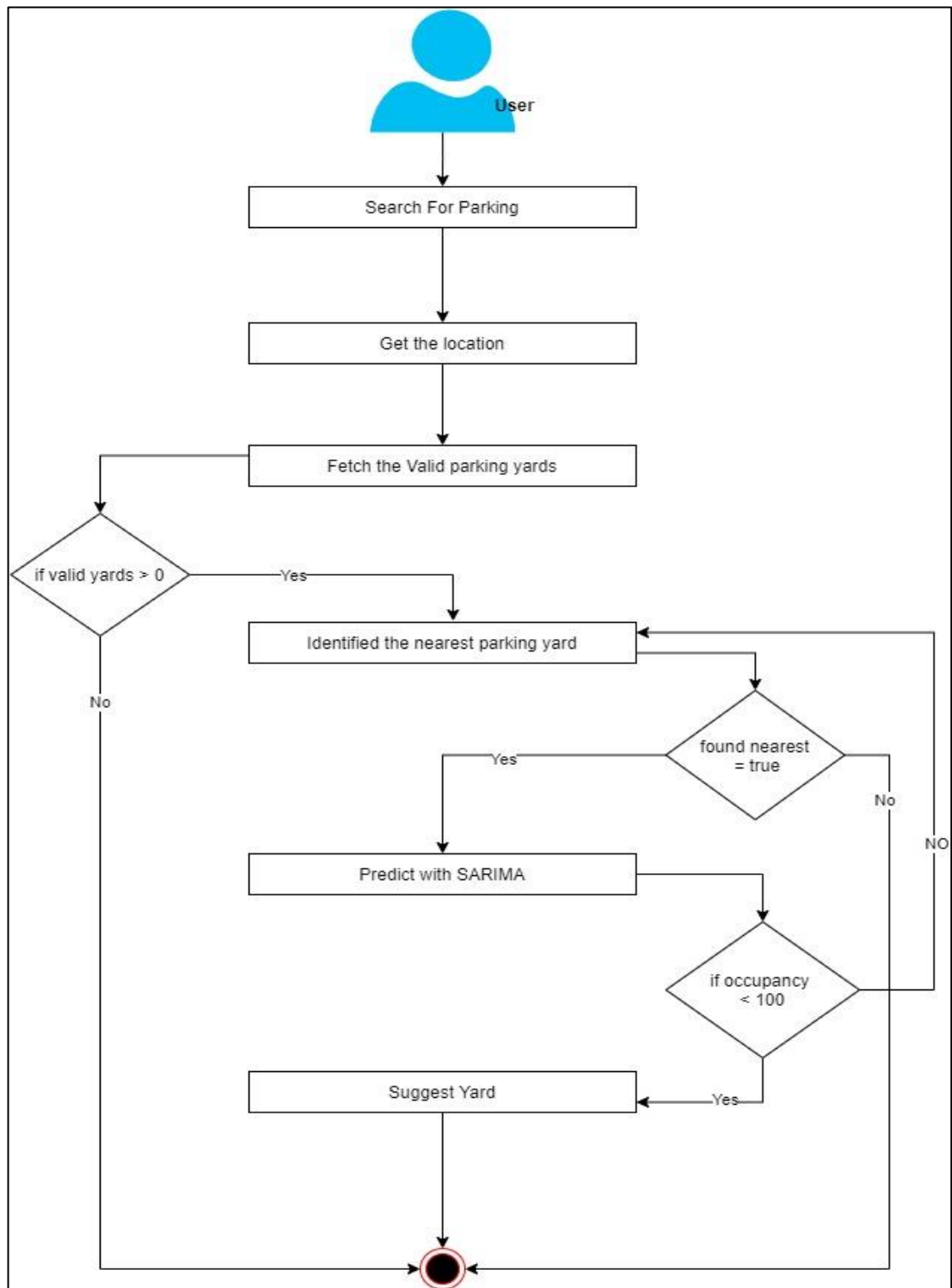


Figure 3.3 Component Flow

- **Component Architecture**

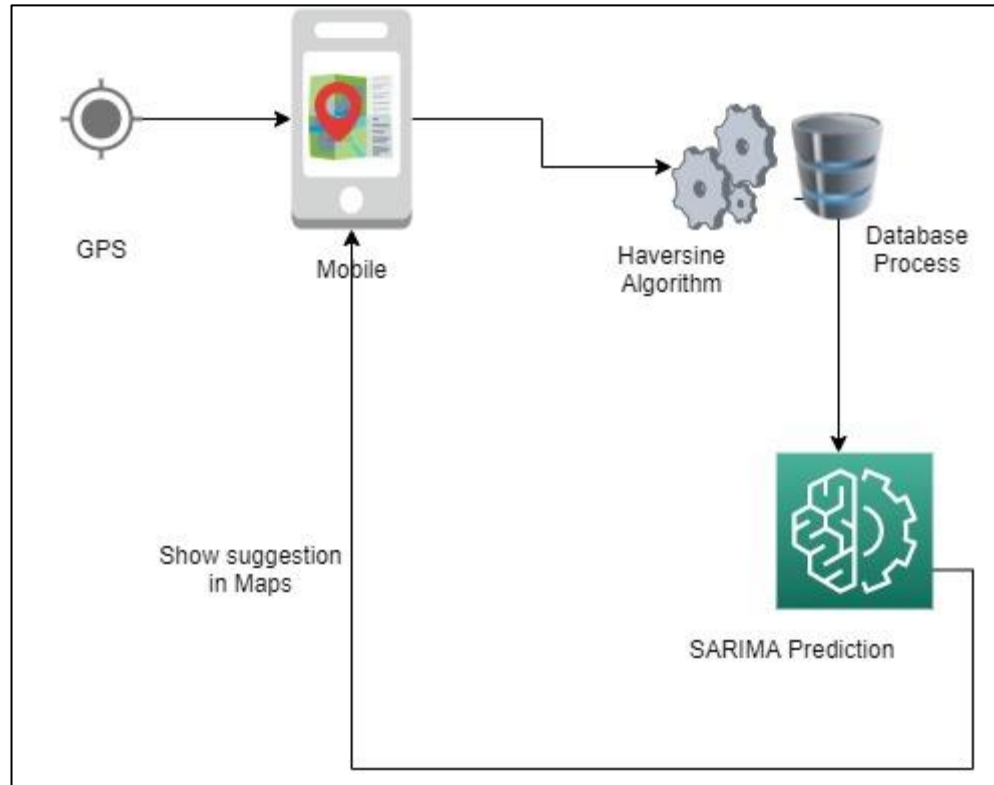


Figure 3.4 Component Architecture

In the figure 3.4 it demonstrates the architecture of the component which describes the communication flow between mobile application and servers.

Parking Yard Surface Quality Validation

When considering the process of the parking yard registration to the system, it is needed to verify whether the parking place surface is at least the minimum standard to park a vehicle in that place. In that case, the suggested solution was to come up with a monthly based automated parking place quality checking method in the system. This methodology works when the parking owner is trying to register his parking yard/garage to the system for renting and the end of the month quality checking of the

parking yard to continue the business. So, to accomplish this work system collects the images of garage/yards and images of the entrance road to the garage/yard through the yard registration approval process using the “Pay as you park” mobile application. In this scenario, technically, the system has to identify the parking yard surface type (asphalt, paved and unpaved) and the quality of each surface type (Standard, Average, Bad) to measure the standard of the parking yard surface (e.g., Asphalt Standard, paved average, etc).

So, A special model was trained for the surface type classification with the following classes: asphalt, paved, and unpaved. Our testing and model training was carried out on a computer using an NVIDIA GeForce GTX 960 graphics running on a Windows 10 operating system with 8 GB RAM. Convolutional Neural Networks (CNN) is used to do the parking surface classification task in our method. Three distinct models were utilized for surface quality for each type of surface. The structure of all four models is the same. The output of the first model is referred to as the particular quality model. A Region of Interest (ROI) (Figure 6) is defined in the pre-processing step for each input image before the CNN structure because it does not need the entire image to categorize the parking surface. The goal of this ROI is to save just the parts of the image that include parking surface pixels. Because in certain frames there are included unnecessary objects that affect the training model badly. So, the top half of the image, as well as a small part of the image bottom have been neglected because of this reason.



Figure 3.5 Region-of-Interest.

As Shown in (Figure 3.5) After that, each frame's brightness is increased or decreased as part of the data augmentation. This improves our training input set and aids our system in learning to recognize the same kind and parking surface types under various lighting situations. The input surface images are passed through a CNN structure that consists of three convolution layers and two fully linked layers.

Then all images were transmitted to the training phase with information on width, height, and number of channels in the first convolution layer. In all of the convolution layers, max-pooling is used to decrease the input dimensionality, which helps in the analysis of feature information in the inputs. After that, ReLU is used as an activation function at the end of each convolution layer.

As the next step, the flatten layer converts the convolution multi-dimensional tensor to the one-dimensional tensor after the convolutional layers. Finally, two fully

connected layers are added. A ReLU activation function is used in the first fully connected layer and the second fully connected layer contains the potential outputs, or desired classes, which will be three classes for parking surface type models, asphalt quality models, and paved quality models, and parking surface unpaved quality models. The probability of each class was calculated using the SoftMax algorithm because it makes greater sense in instances when subjectivity is absent, such as pavement type determination. In the end, the Adam optimizer is used to adjust the network weights based on the input frames used in during the training process.

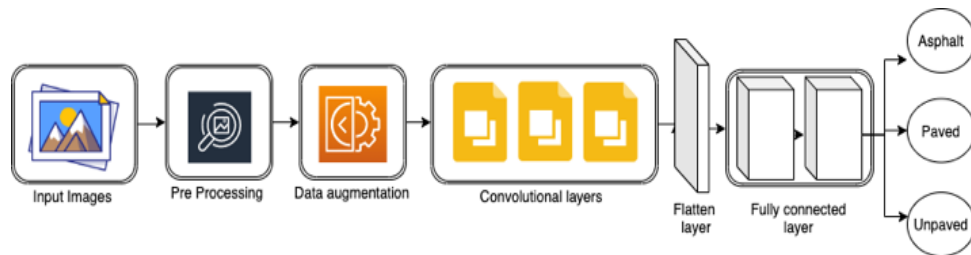


Figure 3.6 Flow Chart of Proposed system

Internal Parking Navigation

According to the current systems in Sri Lankan drivers are wasting their time when finding the exact free slot inside a parking lot after getting into a parking lot. In this system after users get into the parking lot mobile application user must select the name of the parking area when they choose the parking area and then after the system shows a map of the parking area and the available free slots inside the parking area. Free parking slots are fetching from the database details and those available parking slots will be visible to the users on the map.

Although, a user at outside can be tracked by using technology like GPS it is hard to track or cannot track a user's position accurately while the user is inside a building or

underground due to some obstacles like walls, ceilings, etc. [17] By reviewing many indoor positioning systems, in this method, Beacons are being used. According to the literature review, beacons are cheaper than the other technologies and Beacons are more accurate.[18] To get the users positioning three beacons are used with the help of RSSI values of the beacons and Neural Network Machine Learning Algorithm. In this method getting the distance from mobile to each beacon by using the BLE RSSI values of each beacon and passing it to the trained model. Then after the user's X and Y values are being predicted by the model.

After getting the X and Y values of the users these values are displayed on the map. Then the user can see the user's position and the user can find the way to the free parking slot shown on the map. Assume there are two stories in a particular building and how are users able to identify the floor? Yes, that would be a concern. To avoid a few beacons can be used. Beacons have unique ID's and to identify the floors the IDs of those beacons can be used.[19]

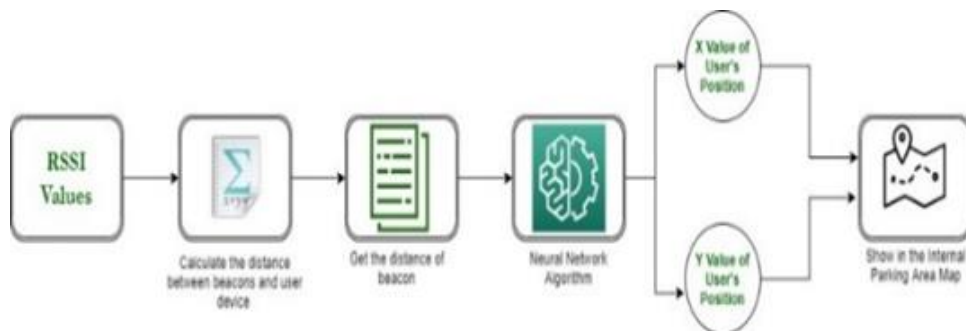


Figure 3.7 Internal Navigation Diagram Structure

Internal Navigation diagram structure shown in Figure 3.7.

Distance calculation between the user's mobile and the three beacons will be calculated using RSSI signal strength and equation for distance calculation is below.

$$\text{RSSI} = 10 * N \log_{10} d + A$$

RSSI - Received Signal Strength Indicator (dBm)

N - Path Loss Exponent

D - The Distance from Transmitter

A - The Reference Value, 1m away

User position Calculation, Triangulation method shown in figure 3.8

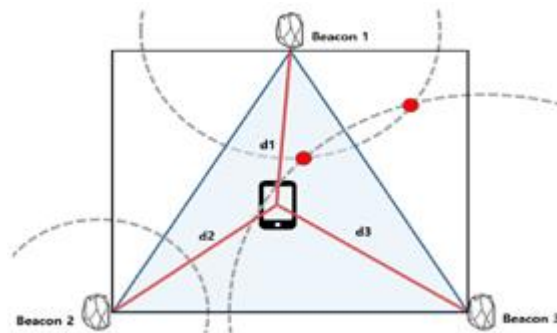


Figure 3.8 Beacon Triangulation

Workflow of the internal navigation system workflow shown in figure 3.9.

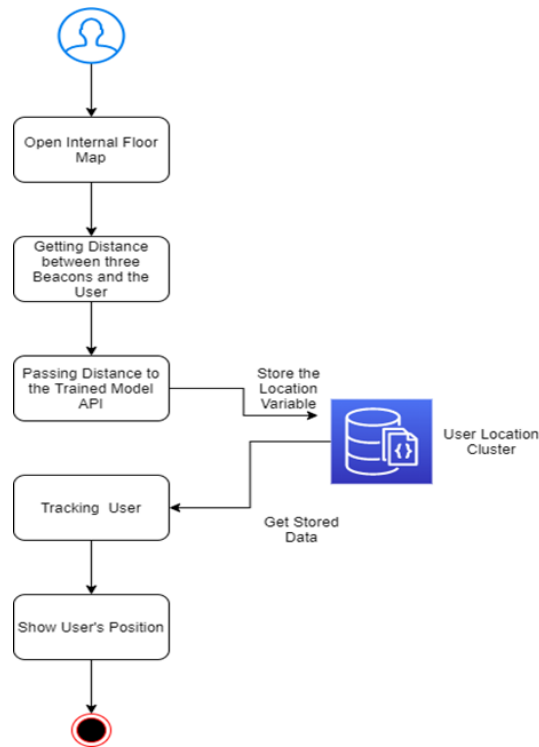


Figure 3.9 Workflow Internal Navigation

Commercialization of the Product

Parking is a highly demanded sector in urban cities and most of the users are facing issues with parking in their day-to-day life. With the introduction of optimal parking suggestions and internal navigation system the accuracy of the parking will improve, and it led users for a better parking experience. The mobile application can gain a high demand with the parking yards and application market will improved with efficiency of the product. Also, the ‘pay as you park’ concept provide a huge benefit for the users and it will save their money. This would be a key commercial factor our product.

With the introduction of parking yard register system, it will be able to maintain the parking yard effective for the parking yard management staff or parking yard owner. This feature would help the product to get the attraction of parking yard owners to

make them register for the application. Since the parking is an essential task and most people face problems with parking in their day-to-day life, our product can gain a trend among the users as it eases the parking process and give users a better experience in parking.

4 TESTING AND IMPLEMENTATION

4.1 Implementation

Pay as You Park smart parking solution system have two applications. One for parking yard owners and it is a web application. For the end users we implemented a mobile application with flutter. It supports both android and iOS operating systems. For my research component I have also used mobile application to visualize the suggestions.

Users need a mobile device with active internet connection in order to access the mobile application.

Optimal Marking Yard Suggestions:

4.1.1 Mobile Application Implementation

Mobile application is developed with flutter which is a widely used cross mobile application development. And it supports widget-based development which is reusable as components. For the ease of development flutter has the feature called hot reload which saves the development time. We do not have to build the whole application again and again after small changes. Flutter will reload the app quickly with function of hot reload.

For the database of whole application Mongo DB is used since it got a high availability and accuracy. Also, it provides object-based database which is useful for the mobile platform. Compared to other cloud providers mongo support flexibility than other.

We have used node.js application for API level development to do operations with Mongo DB. Database operations are executed through node.js using the API. Express.js used as a framework for the API development. Further, the API project is deployed in Heroku and get the Heroku endpoint.

In order to visualize the location in mobile app I have used the google map API service with flutter and implemented the map widget to demonstrate the map in mobile application. Also, to locate the current position of user Location service is used which is provided by dart. Once the algorithm suggests the parking yard to visualize the directions in map, I have used the Google route API which returns the route matrix and other related data.

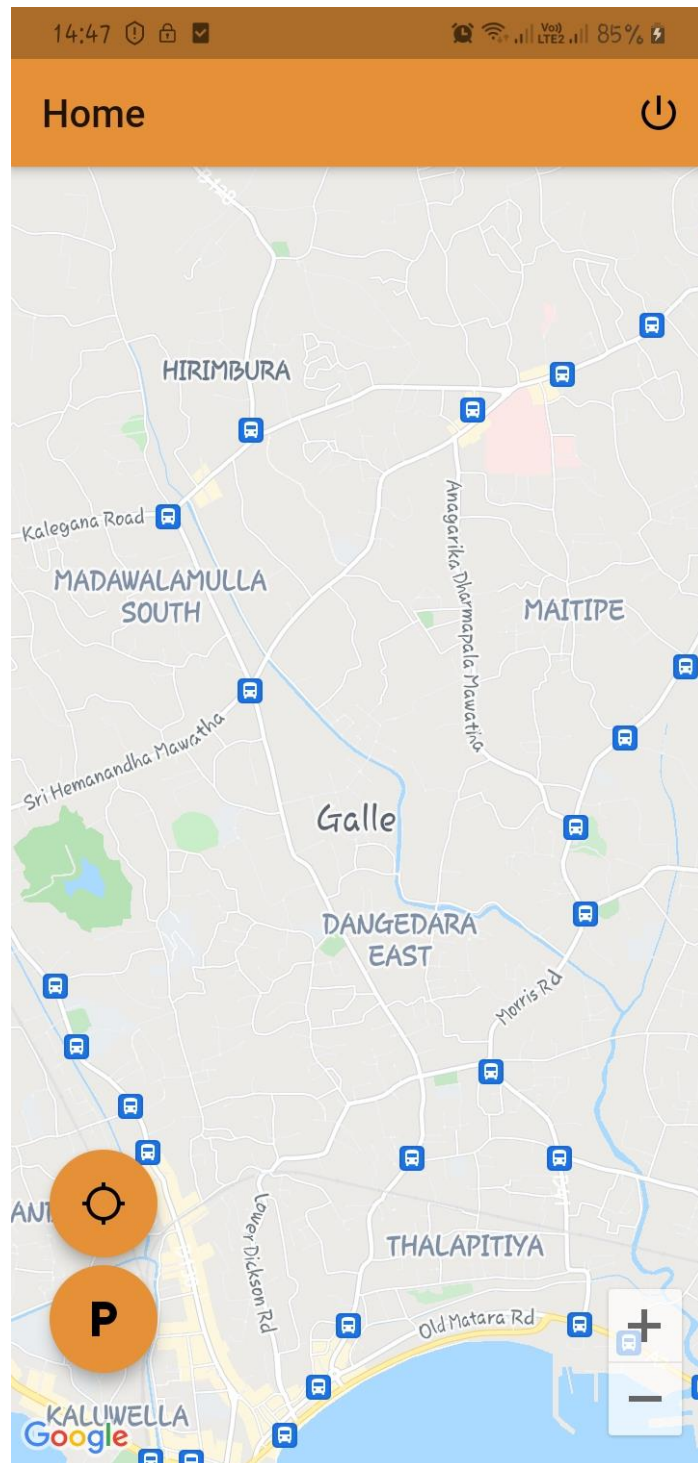


Figure 4.1 Initial Map Load

Initial map load in flutter application with Google map API services. Here it has implemented the Google Maps widget provided by google.

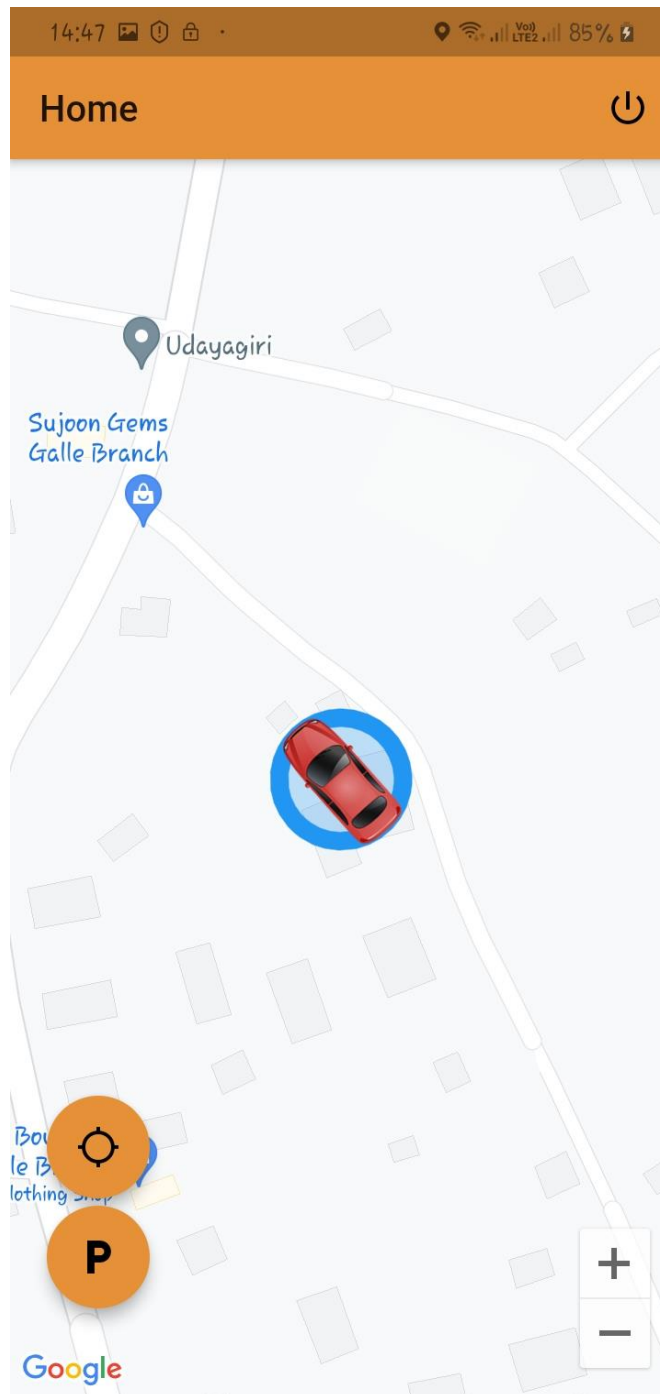


Figure 4.2 Tracking User Location

In figure 4.2 it visualizes the user's current location with the red colored car icon. For this functionality I have used Location service provide by Location Platforms in dart. Also, it tracks user location and update the map when uses change the position. This target is achieved through location tracker service provided by dart. User must grant permission to track the location of the mobile. This function initially prompt to ask the access and then user must grant the access.

```
67
68 void getCurrentLocation() async {
69   try {
70
71     Uint8List imageData = await getMarker();
72     var location = await _locationTracker.getLocation();
73
74     updateMarkerAndCircle(location, imageData);
75
76     marker_active = true;
77
78     /*if (_locationSubscription != null) {
79       _locationSubscription.cancel();
80     }*/
81
82
83     _locationSubscription = _locationTracker.onLocationChanged.listen((newLocalData) {
84       if (_controller != null) {
85         _controller.animateCamera(CameraUpdate.newCameraPosition(new CameraPosition(
86           bearing: 192.8334901395799,
87           target: LatLng(newLocalData.latitude!.toDouble(), newLocalData.longitude!.toDouble()),
88           tilt: 0,
89           zoom: 18.00)); // CameraPosition
90         updateMarkerAndCircle(newLocalData, imageData);
91       }
92     });
93
94   } on PlatformException catch (e) {
95     if (e.code == 'PERMISSION_DENIED') {
96       debugPrint("Permission Denied");
97     }
98   }
99 }
```

Figure 4.3 Implementation of location tracker 1

```

44 void updateMarkerAndCircle(LocationData newLocalData, Uint8List imageData) {
45   LatLng latLng = LatLng(newLocalData.latitude!.toDouble(), newLocalData.longitude!.toDouble());
46   this.setState(() {
47     marker = Marker(
48       markerId: MarkerId("home"),
49       position: latLng,
50       rotation: newLocalData.heading!.toDouble(),
51       draggable: false,
52       zIndex: 2,
53       flat: true,
54       anchor: Offset(0.5, 0.5),
55       icon: BitmapDescriptor.fromBytes(imageData)); // Marker
56   circle = Circle(
57     circleId: CircleId("car"),
58     radius: newLocalData.accuracy!.toDouble(),
59     zIndex: 1,
60     strokeColor: Colors.blue,
61     center: latLng,
62     fillColor: Colors.blue.withAlpha(70)); // Circle
63   });
64 }

```

Figure 4.4 Implementation of Location tracker 2

Above figure 4.3 and figure 4.4 are the implementations for location tracking using Location service by dart. Also, it updates the map time to time when use changes the position. I have used custom marker icons and widgets in order provide a better user experience for the users.

And when it comes to visualize the direction, I have used Poly Lines services which lay over the map and visualize the path which is gained by the Google Route API. And it starts the location tracker service along with directions. So user can navigate to the destined parkin yard with the help of navigations provide by the application.

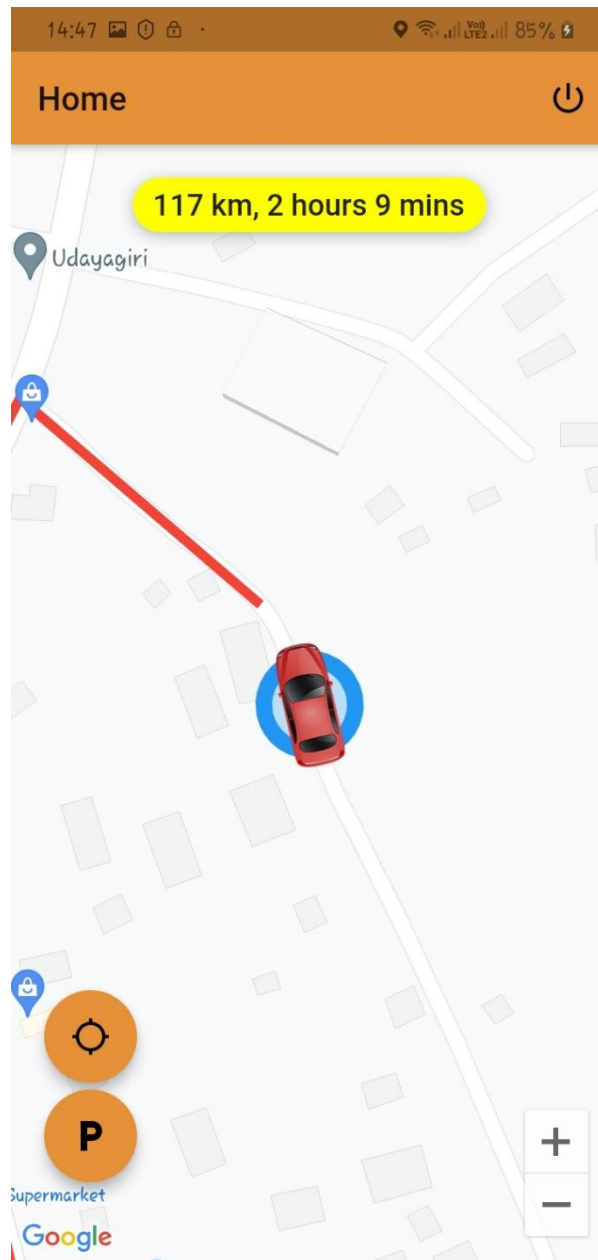


Figure 4.5 Directions Interface

Above figure 4.5 demonstrate the visualization of directions in mobile application with Google Maps and Poly Lines.

```

135   @override
136   Widget build(BuildContext context) {
137     return Scaffold(
138       appBar: _buildAppBar(),
139       body: Stack(alignment: Alignment.center, children: <Widget>[
140         GoogleMap(
141           mapToolbarEnabled: true,
142           mapType: MapType.normal,
143           initialCameraPosition: initialLocation,
144           markers: Set.of((marker_active != false) ? [marker] : []),
145           circles: Set.of((marker_active != false) ? [circle] : []),
146           polylines: {
147             if (destination_active != false)
148               Polyline(
149                 polylineId: const PolylineId('overview_polyline'),
150                 color: Colors.red,
151                 width: 5,
152                 points: _info.polylinePoints
153                   .map((e) => Latlng(e.latitude, e.longitude))
154                   .toList(),
155               ), // Polyline
156             },
157           onMapCreated: (GoogleMapController controller) {
158             _controller = controller;
159           },
160         ), // GoogleMap
161         Positioned(
162           bottom: 110,
163           left: 20,
164           child: FloatingActionButton(
165             child: Icon(Icons.location_searching),
166             heroTag: 1,
167             onPressed: () {
168               getCurrentLocation();
169             },
170           ), // FloatingActionButton, Positioned
171         Positioned(
172           bottom: 50,
173           left: 20,
174           child: FloatingActionButton(
175             child: Icon(Icons.local_parking)

```

Figure 4.6 Implementation Google Map Widget

Above figure 4.6 demonstrate the implementation of Google Map with the Locations and Poly Lines.

In order to visualize the internal parking floor map, I have used an image passing to the flutter application and there are two colors (Figure 4.7) of pin markers in this application to identify the available slots inside the parking area. Availability will be getting from the Mongo DB as a Boolean value and color will be changed accordingly.

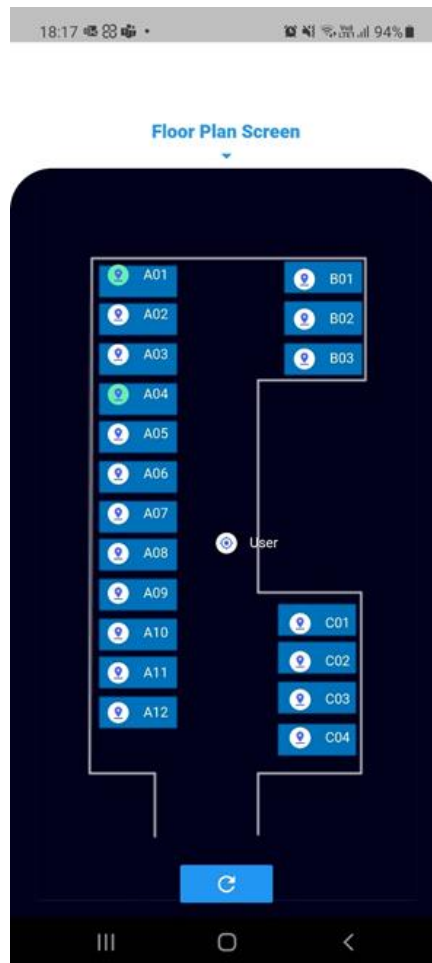


Figure 4.7 Availability View UI

4.1.2 API Server Implementation

To achieve the functionality of communicate with the Mono DB with mobile application we have used a node.js server application project using Express.js. Using this provide we provide endpoints for each collection in Mongo cluster to perform several operations. And the application is deployed in Heroku and using the endpoint

provide by Heroku communication is done between server and client mobile application.

4.1.3 Parking Yard Suggestion Implementation

To achieve the target of achieving suggest the optimal parking yard for a user to park the vehicle I have implemented an algorithm to suggest a parking yard considering the distance, availability, physical characteristics of vehicle and time taken to reach the destined parking yard.

- **Haversine Implementation**

Haversine is used for identifying the nearest parking yard for a user. It takes the users location coordinates as inputs and get the nearest for that location. I have implemented the haversine algorithm with python which is widely used for dataset handling. As a first step I fetch all the parking yard in the Mongo DB into a dataset. Which is used in the next stages to process. Among the dataset fetched it will fetch a set of valid nearest parking yards to the user location with the use of haversine algorithm. Initially it will pick up the initial parking yard with shortest distance to the user.

	ID	lat	lon	Name	address	no of slots
0	P1	79.882664	6.914951	LOLC Car Park	21, 25 Chandreleka Mawatha, Colombo 00800	120
1	P2	79.876301	6.883533	Best Western Elyon Colombo	Baseline Road, 102A Kirulapone Ave, Colombo 00500	75
2	P3	79.850030	6.933701	Fort	Fort, Colombo	100
3	P4	79.864816	6.928458	Maradana Railway Station	Jayantha Weerasekara Mawatha, Colombo 01000	225
4	P5	79.870644	6.879944	Tenaga Carparks (Pvt)Ltd	Level 4, 124 Maya Ave, Colombo 00500	300

Haversine Formula

```

from math import radians, cos, sin, asin, sqrt
def dist(lat1, long1, lat2, long2):
    # convert decimal degrees to radians
    lat1, long1, lat2, long2 = map(radians, [lat1, long1, lat2, long2])
    # haversine formula
    dlon = long2 - long1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371 * c
    return km

def find_nearest(lat, long):
    distances = parkin_yards.apply(
        lambda row: dist(lat, long, row['lat'], row['lon']),
        axis=1)
    return parkin_yards.loc[distances.idxmin(), 'ID']

```

Figure 4.8 Haversine Implementation

• SARIMA model Implementation

SARIMA is used the forecasting model to predict the occupancy at a given time of a parking yard. I have implemented this model using python with Jupiter Notebooks and then build a pickle dataset for the predictions. However, built pickle file is used in Algorithm in order to predict the occupancy when the time is given.

First and initial phase of the model implementation is to analyze the parking dataset and clean the data set as there may be unwanted and improper data which would affect the final predictions. In the analyzation process it includes autocorrelation and partial correlation functions. Since SARIMA is compatible

with dataset with seasonal component those functional tests need to be perform in order to use the SARIMA model. Also, there is a process for test stationary before fitting to the model. In this analysis phase we have make sure that the means and means of standard deviation are stable in the endogenous variable are stable through time. Usually, the Dickey-Filler test is used to test is used to test for stationary.

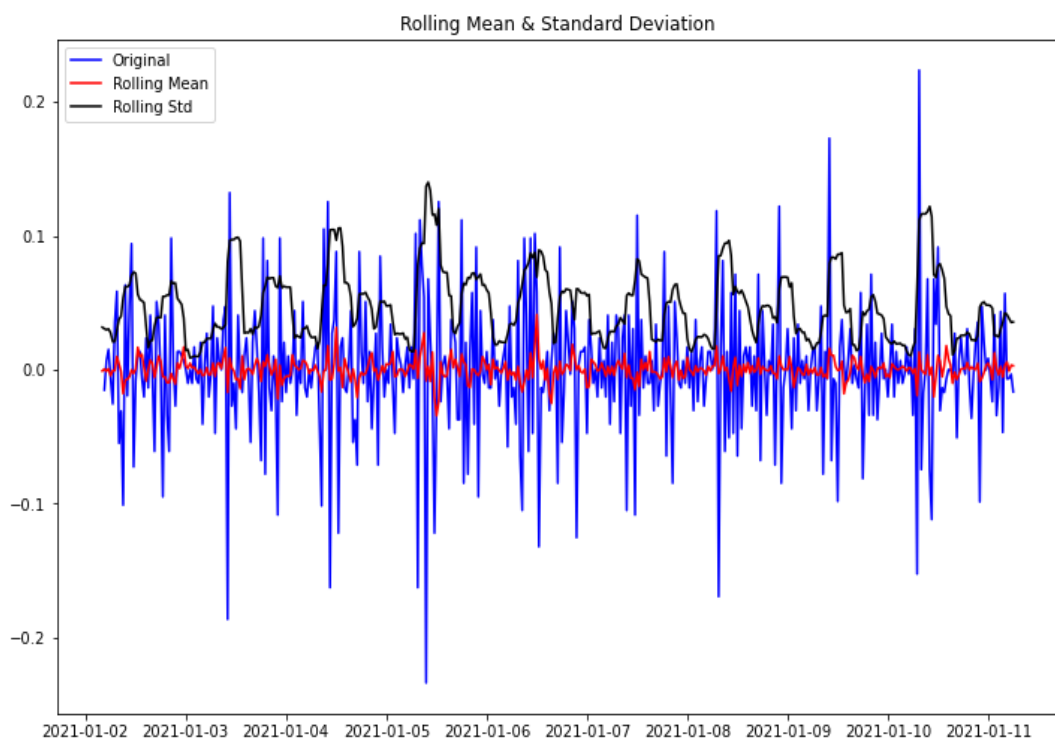


Figure 4.9 Stationary Test

Above figure 4.9 shows the stationary test used for 48 periods of the day. Which means I have split an hour for two half hour sections. In the SARIMA model I have done implementations to fit the model to 48 periods.

```

%%time
# Define and fit SARIMA model
my_seasonal_order = (1, 1, 1, 48)
sarima_model = SARIMAX(train, order=(1, 0, 1), seasonal_order=my_seasonal_order)
results_SAR = sarima_model.fit(dispatch=1)

C:\Users\moham\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:524: ValueWarning: No frequency information was provided, so inferred frequency 30T will be used.
  warnings.warn('No frequency information was')
C:\Users\moham\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:524: ValueWarning: No frequency information was provided, so inferred frequency 30T will be used.
  warnings.warn('No frequency information was')

Wall time: 9.44 s

plt.figure(figsize=(16,6))
plt.title('SARIMA Model on Aggregate Data')
plt.plot(train, label='Training Actual Occupancy Rate')
plt.xlabel('Date')
plt.ylabel('Percent Occupied')
y_pred_sar = pd.Series(results_SAR.forecast(steps=len(test)).values, index=test.index)
plt.plot(test, label='Testing Actual Occupancy Rate')
plt.plot(y_pred_sar, color='red', label='SARIMA Predicted Occupancy Rate')
plt.legend()

plt.show()

print('-'*77)
print('SARIMA Model Metrics on Test Data')
print('-'*77)
report_metrics(test.squeeze(), y_pred_sar.squeeze())

```

Figure 4.10 SARIMA implementation

After the analyzation and dataset processes using the above code that I have implemented the dataset can be fit to the SARIMA.

As shown in the figure 4.10 the implementations of final algorithm have done using python by integrating all the key facts that need to be consider when suggest parking yard to users. PyMongo is used in the algorithm to fetch the data from mongo DB. Mongo Client is a API provided by pymongo which is used to do operations in collection python. Also, distance matrix API is used to get the distance and time taken to reach the destinated parking yard. Python provides a package called google maps which provide access to the google APIs. Initially we must create a project in google developer console and must enable the APIs we need to use. Using the generated API key, we can use the subscribed API within the project.

```

25     dbname = get_database()
26
27     # Create a new collection
28     collection_name = dbname["parking_yard"]
29     query = {"Capacity" > "Occupancy"}
30
31     parking_yards = collection_name.find()
32
33     # convert the dictionary objects to dataframe
34     items_df = DataFrame(parking_yards)
35
36     # see the magic
37     #print(items_df)
38
39     items_df['Capacity'] = items_df.Capacity.astype(int)
40     items_df['Occupancy'] = items_df.Occupancy.astype(int)
41     items_df['MaxWidth'] = items_df.MaxWidth.astype(int)
42     items_df['MaxLength'] = items_df.MaxLength.astype(int)
43     items_df['MaxHeight'] = items_df.MaxHeight.astype(int)
44     items_df['Latitude'] = items_df.Latitude.astype(str).astype(float)
45     items_df['Longitude'] = items_df.Longitude.astype(str).astype(float)
46
47     available_yards_df = items_df.loc[(items_df.Capacity > items_df.Occupancy) & (items_df.MaxHeight > user_max_height) & (items_df.MaxWidth > user_max_width)]
48
49     # print(available_yards_df)
50
51     found_yard = False
52     suggest_yard_id = ''
53     yard_id = ''
54     latitude = ''
55     longitude = ''

```

Figure 4.11 Component Implementation

```

60
61     while found_yard == False:
62         yard_id = find_nearest(user_lat, user_lon, available_yards_df)
63         latitude = available_yards_df.loc[available_yards_df['PID'] == yard_id].Latitude
64         longitude = available_yards_df.loc[available_yards_df['PID'] == yard_id].Longitude
65         # print(yard_id)
66         # print(list(latitude))
67
68         origins = (user_lat, user_lon)
69         destination = (longitude, latitude)
70
71         #print(latitude)
72         # print(longitude)
73
74         result = gmaps.distance_matrix(origins, destination, mode="driving")["rows"][0][0]["elements"][0][0]["duration"]["value"]
75         # result = 1331
76
77         # datetime object containing current date and time
78         now = datetime.datetime.now()
79         reach_time = now + datetime.timedelta(seconds=result)
80         print(reach_time)
81
82         occupancy = model.predict(start=reach_time, end=reach_time)
83
84         # print(list(occupancy)[0])
85         # print(type(list(occupancy)[0]))
86
87         if list(occupancy)[0] < 100:
88             found_yard = True
89
90         # found_yard = True
91
92
93     return{
94         "occupancy": list(occupancy)[0],
95         "yard_id": yard_id,
96         "latitude": list(latitude)[0],
97         "longitude": list(longitude)[0],
98         'found_yard': found_yard
99     }

```

Figure 4.12 Model Implementation 2

4.1.4 User Positioning Model Implementation

User prediction model is trained by using Neural Network and below I have briefly described about the NN Machine Learning algorithm.

A neural network is a set of algorithms that attempts to recognize underlying relationships in a batch of data using a technique similar to how the human brain works. In this context, neural networks are systems of neurons that might be organic or artificial in nature. Because neural networks can adapt to changing input, they can produce the best possible results without having to rethink the output criteria. The neural network concept, which has its roots in artificial intelligence, is quickly gaining traction in the creation of trading systems. Graphical structure of NN is shown in figure 4.13.

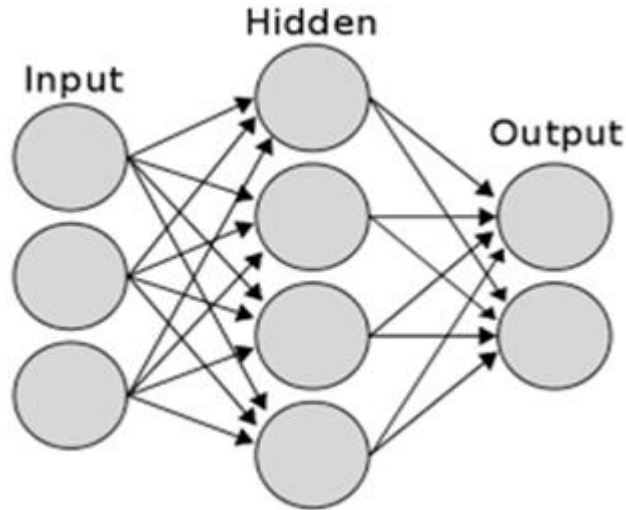


Figure 4.13 Neural Network Algorithm

Model of getting user's position is trained by using Neural Network Machine Learning algorithm.


4.2 Web Application Implementation

Web application is developed with Reactjs which is a widely used web application development. Those who utilize ReactJS can expect higher performance than those that use other frameworks. Because ReactJS helps to prevent DOM updates, apps will run faster and provide a better user experience. ReactJS was created with the goal of improving the total number of pages rendered by the website server.

Mongo DB is utilized for the application's database because of its high availability and accuracy. It also includes an object-based database, which is beneficial for mobile platforms. When compared to other cloud services, Mongo offers more freedom. To execute operations with Mongo DB, we used a node.js application at the API level. The API uses node.js to conduct database operations, while Express.js is utilized as a framework for API development. The API project is also launched to Heroku, and the Heroku endpoint is obtained.

And to register parking yard we used Mapbox API for the location service with react and implement the location picking while the place registering to the system. And it also used to visualize the all-registered places in one custom map using the latitude and longitude of the registered places.

In order to validate the parking yard images we had to deploy predictive deep learning model in the flask server using Heroku. After creating the flask API our api could take the image and returned the json result for the further registration process.



Sign up

First Name *

Last Name *


Email Address *

Password *



Repeat Password *

SIGN UP

 GOOGLE SIGN IN

ALREADY HAVE AN ACCOUNT? SIGN IN

Figure 4.14 Register parking yard owner

In figure 4.15 it visualizes the parking yard owner registration to the system. In this functionality owner can register using his details by typing each field or else owner can register using his own Gmail address by using Google authentication.

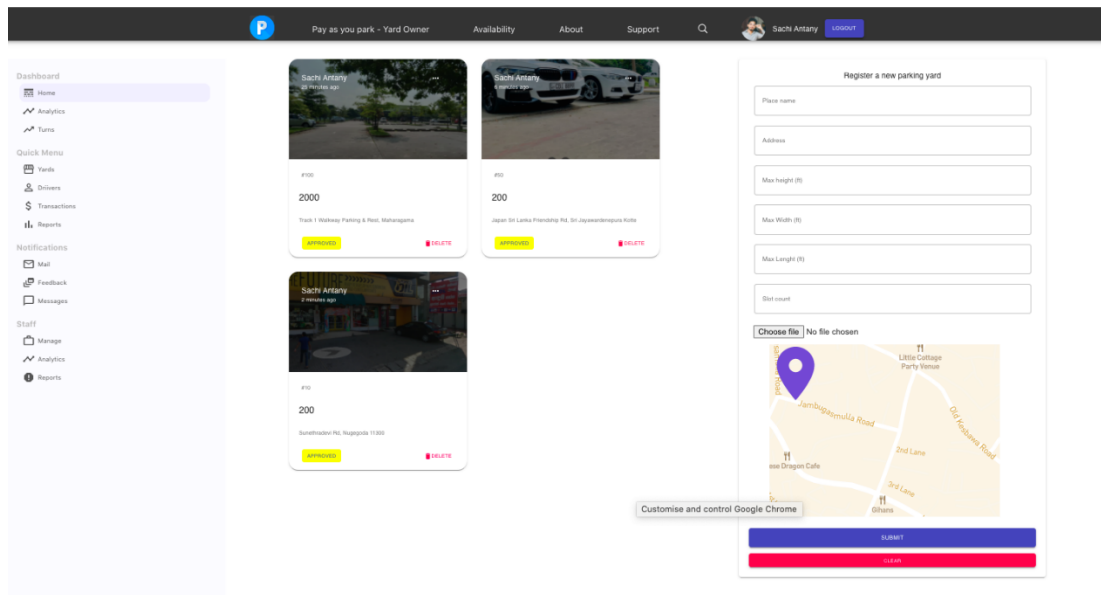


Figure 4.15 Register parking yard

Once after register the owner, owner can register his parking yards to the system by giving necessary information to the system. In that case, we are asking for parking place name, Address of the place, Dimension of the parking place (Height, Width, Length), Parking slot count, Images of the parking yard and pick the location from the map. Initially map picker ask for GPS access for the location picking and gives the current location for the registration, but owner can change it as their wish by navigation the marker to the correct place. After that, data push to the mongo db and image data send to the

4.1.5. Parking Surface Quality Detection Model Implementation

To accomplish the quality measuring of the new parking yards for register in the system, I have implemented the CNN structure to classify the surface and the detect the quality of the surface using the yard images collected by from the parking yard

owner. For this purpose, I have to train two types of models, which are parking yard classification model set and parking yard quality classification model.

- **Parking Surface Classification**

First of all, we have to prepare the data before the model training. So that collected data has categorized to three separate folders with the surface type name (Figure 4.16)

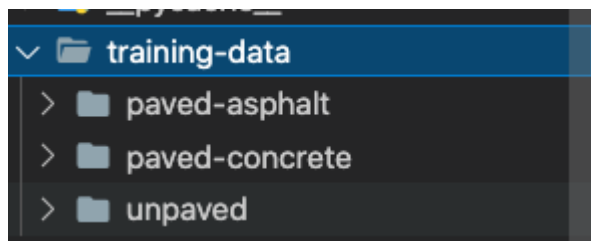


Figure 4.16 Training data folder categorizing

Then collected data separated to 20% for automatically validation, set batch size as 32 and then data pushed thorough the dataset.py class for the further training data utilization (Figure 4.17)

```
18 #Prepare input data
19 classes = os.listdir('training-data')
20 num_classes = len(classes)
21
22 # 20% of the data will automatically be used for validation
23 validation_size = 0.2
24 img_size = 128
25 num_channels = 3
26 train_path='training-data'
27
28 # Training the dataset
29 data = dataset.read_train_sets(train_path, img_size, classes, validation_size=validation_size)
```

Figure 4.17 Train.py code part 1

Using the dataset.py, we have adjusted the dataset quality and utilize the data frames with ROI and the data argumentation. (Figure 4.18 & 4.19)

```
8 def adjust_gamma(image):
9     # adjusted gamma values
10     gamma = 0.5
11     invGamma = 1.0 / gamma
12     table = np.array([((i / 255.0) ** invGamma) * 255
13                       for i in np.arange(0, 256)]).astype("uint8")
14
15     # apply gamma correction
16     return cv2.LUT(image, table)
17
18 def increase_brightness(img, value):
19     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
20     h, s, v = cv2.split(hsv)
21
22     li (variable) v: Any
23     v[v > lim] = 255
24     v[v <= lim] += value
25
26     final_hsv = cv2.merge((h, s, v))
27     img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
28     return img
29
```

Figure 4.18 dataset.py code part 1

```

37     for fields in classes:
38         index = classes.index(fields)
39         print('Reading {} files (Index: {})'.format(fields, index))
40         path = os.path.join(train_path, fields, '*g')
41         files = glob.glob(path)
42         for fl in files:
43             image = cv2.imread(fl)
44
45             height, width = image.shape[:2]
46             newHeight = int(round(height/2))
47             image = image[newHeight-5:height-50, 0:width]
48
49             brght_img = increase_brightness(image, value=150)
50
51             shaded_img = adjust_gamma(image)
52
53             image = cv2.resize(image, (image_size, image_size), 0, 0, cv2.INTER_LINEAR)
54             image = image.astype(np.float32)
55             image = np.multiply(image, 1.0 / 255.0)
56
57             brght_img = cv2.resize(brght_img, (image_size, image_size), 0, 0, cv2.INTER_LINEAR)
58             brght_img = brght_img.astype(np.float32)
59             brght_img = np.multiply(brght_img, 1.0 / 255.0)
60
61             shaded_img = cv2.resize(shaded_img, (image_size, image_size), 0, 0, cv2.INTER_LINEAR)
62             shaded_img = shaded_img.astype(np.float32)
63             shaded_img = np.multiply(brght_img, 1.0 / 255.0)

```

Figure 4.19 dataset.py code part 2

Next, we defined CNN layers in the train.py and passed all images through those layers (Figure 4.20).

First two layers contain 32 filters with size of 3 by 3. Padding was set to 0 and all strides were set to 1. The weights are initialized using a normal distribution. A max-pooling is performed to all the convolution layers in order to decrease the inputs dimensionally, which aids in the analysis of features contained in the input sub-regions. A ReLU is used as an activation function at the conclusion of each convolution layer, following the max-pooling function.

```

65 def create_convolutional_layer(input,
66                                num_input_channels,
67                                conv_filter_size,
68                                num_filters):
69
70     ## define the weights that will be trained using create_weights function.
71     weights = create_weights(shape=[conv_filter_size, conv_filter_size, num_input_channels, num_filters])
72     ## create biases using the create_biases function.
73     biases = create_biases(num_filters)
74
75     ## Creating the convolutional layer
76     layer = tf.nn.conv2d(input=input,
77                          filters=weights,
78                          strides=[1, 1, 1, 1],
79                          padding='SAME')
80
81     layer += biases
82
83     ## Using max-pooling.
84     layer = tf.nn.max_pool2d(input=layer,
85                              ksize=[1, 2, 2, 1],
86                              strides=[1, 2, 2, 1],
87                              padding='SAME')
88     ## Output of pooling is fed to Relu.
89     layer = tf.nn.relu(layer)
90
91     return layer
92

```

Figure 4.20 Train.py code part 2

Afterward, the flatten layer is used to convert the convolution multi-dimensional tensor into a one-dimensional tensor after the convolutional layers (Figure 4.21).

```

95 def create_flatten_layer(layer):
96     layer_shape = layer.get_shape()
97
98     num_features = layer_shape[1:4].num_elements()
99
100     ## Flatten the layer, reshape to num_features
101     layer = tf.reshape(layer, [-1, num_features])
102
103     return layer

```

Figure 4.21 Train.py code part 2

At the end, two fully connected layers are added and a ReLU activation function is used after those layers (Figure 4.22).

```

106 def create_fc_layer(input,
107     num_inputs,
108     num_outputs,
109     use_relu=True):
110
111     #define trainable weights and biases.
112     weights = create_weights(shape=[num_inputs, num_outputs])
113     biases = create_biases(num_outputs)
114
115     # Fully connected layer takes input x and produces wx+b.using matmul function in Tensorflow
116     layer = tf.matmul(input, weights) + biases
117     if use_relu:
118         layer = tf.nn.relu(layer)
119
120     return layer

```

Figure 4.22 Train.py code part 3

The probability of each class was calculated using the SoftMax algorithm. At the end, we utilize the Adam optimizer to adjust the network weights based on the input data used during training. After that we trained the model using the Train.py python script and prepared the model for performance (Figure 4.23).

```

123 layer_conv1 = create_convolutional_layer(input=x,
124     num_input_channels=num_channels,
125     conv_filter_size=filter_size_conv1,
126     num_filters=num_filters_conv1)
127
128 layer_conv2 = create_convolutional_layer(input=layer_conv1,
129     num_input_channels=num_filters_conv1,
130     conv_filter_size=filter_size_conv2,
131     num_filters=num_filters_conv2)
132
133 layer_conv3 = create_convolutional_layer(input=layer_conv2,
134     num_input_channels=num_filters_conv2,
135     conv_filter_size=filter_size_conv3,
136     num_filters=num_filters_conv3)
137
138 layer_flat = create_flatten_layer(layer_conv3)
139
140 layer_fc1 = create_fc_layer(input=layer_flat,
141     num_inputs=layer_flat.get_shape()[1:4].num_elements(),
142     num_outputs=fc_layer_size,
143     use_relu=True)
144
145 layer_fc2 = create_fc_layer(input=layer_fc1,
146     num_inputs=fc_layer_size,
147     num_outputs=num_classes,
148     use_relu=False)
149
150 y_pred = tf.nn.softmax(layer_fc2,name='y_pred')
151
152 y_pred_cls = tf.argmax(input=y_pred, axis=1)
153 session.run(tf.compat.v1.global_variables_initializer())
154 cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc2,
155     labels=tf.stop_gradient(y_true))
156 cost = tf.reduce_mean(input_tensor=cross_entropy)
157 optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)
158 correct_prediction = tf.equal(y_pred_cls, y_true_cls)
159 accuracy = tf.reduce_mean(input_tensor=tf.cast(correct_prediction, tf.float32))

```

Figure 4.23 Train.py code part 4

Afterward we define a class in test.py script to input the link of the image and read the image data to evaluate the result using the graphs of trained models (Figure 4.24). Then test.py script return the type of the surface (Asphalt, paved, unpaved) by predicting the image (Figure 4.25).

```
20 def predict(path):
21
22     cap = cv.VideoCapture(path)
23
24     image_size=128
25     num_channels=3
26     images = []
27
28     data = {
29         'label': '',
30         'quality': '',
31         'probability': ''
32     }
33
34     width = round(cap.get(cv.CAP_PROP_FRAME_WIDTH))
35     height = round(cap.get(cv.CAP_PROP_FRAME_HEIGHT))
36
37     newHeight = int(round(height/2))
38
39     graph = tf.Graph()
40     graphAQ = tf.Graph()
41     graphPQ = tf.Graph()
42     graphUQ = tf.Graph()
43
44     default_graph = tf.compat.v1.get_default_graph()
```

Figure 4.24 test.py code part 1

```

80
81     if index == 0:
82         label = 'Asphalt'
83         prob = str("{0:.2f}".format(value))
84         color = (0, 0, 0)
85     elif index == 1:
86         label = 'Paved'
87         prob = str("{0:.2f}".format(value))
88         color = (153, 102, 102)
89     elif index == 2:
90         label = 'Unpaved'
91         prob = str("{0:.2f}".format(value))
92         color = (0, 153, 255)
93

```

Figure 4.25 test.py code part 2

- **Parking Surface and Quality Classification**

When classifying the quality of each surface type I used the same technic used in the above model which is classifying the surface type. For this work, all training images categorized according to the (Figure 4.26). Then trained three models for each surface types qualities.

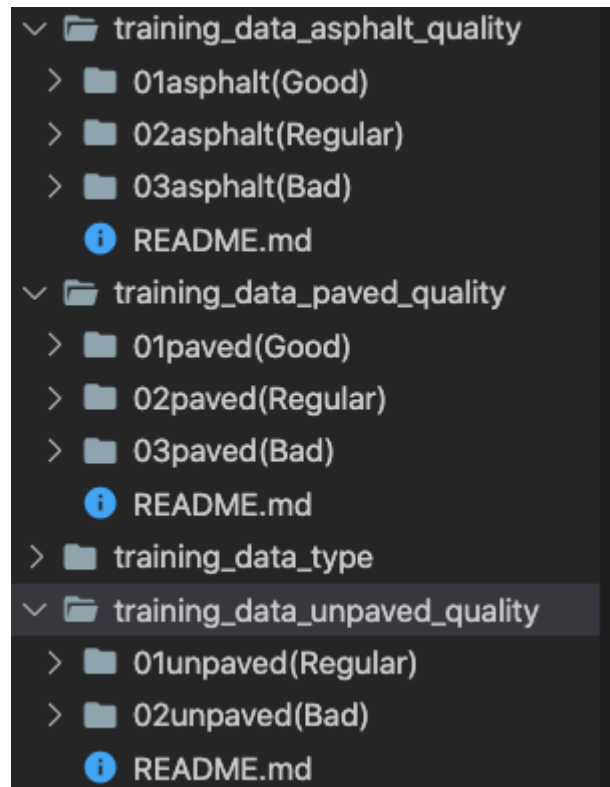


Figure 4.26 Surface quality classification folder structure

Then I used the surface type classification model to identify the surface type and the used other three surface quality models to identify the quality of each surface.

```

46     # Restoring for types model
47     with graph.as_default():
48         saver = tf.compat.v1.train.import_meta_graph('parkingsurfaceType-model.meta')
49
50     # Accessing the graph
51     y_pred = graph.get_tensor_by_name("y_pred:0")
52
53     x = graph.get_tensor_by_name("x:0")
54     y_true = graph.get_tensor_by_name("y_true:0")
55     y_test_images = np.zeros((1, len(os.listdir('training_data_type'))))
56
57     sess = tf.compat.v1.Session(graph = graph)
58     saver.restore(sess, tf.compat.v1.train.latest_checkpoint('typeCheckpoint/'))

```

Figure 4.27 Restoring the surface type model

```
61 # Restoring the asphalt quality model
62 with graphAQ.as_default():
63     saverAQ = tf.compat.v1.train.import_meta_graph('parkingsurfaceAsphaltQuality-model.meta')
64
65     # Accessing the graph
66     y_predAQ = graphAQ.get_tensor_by_name("y_pred:0")
67
68     xAQ = graphAQ.get_tensor_by_name("x:0")
69     y_trueAQ = graphAQ.get_tensor_by_name("y_true:0")
70     y_test_imagesAQ = np.zeros((1, len(os.listdir('training_data_asphalt_quality'))))
71
72 sessAQ = tf.compat.v1.Session(graph = graphAQ)
73 saverAQ.restore(sessAQ, tf.train.latest_checkpoint('asphaltCheckpoint/'))
```

Figure 4.28 Restoring the asphalt quality model

```
75 # Restoring the paved quality model
76 with graphPQ.as_default():
77     saverPQ = tf.compat.v1.train.import_meta_graph('parkingsurfacePavedQuality-model.meta')
78
79     # Accessing the graph
80     y_predPQ = graphPQ.get_tensor_by_name("y_pred:0")
81
82     xPQ = graphPQ.get_tensor_by_name("x:0")
83     y_truePQ = graphPQ.get_tensor_by_name("y_true:0")
84     y_test_imagesPQ = np.zeros((1, len(os.listdir('training_data_paved_quality'))))
85
86 sessPQ = tf.compat.v1.Session(graph = graphPQ)
87 saverPQ.restore(sessPQ, tf.compat.v1.train.latest_checkpoint('pavedCheckpoint/'))
```

Figure 4.29 Restoring the paved quality model

```
89 # Restoring unpaved quality model
90 with graphUQ.as_default():
91     saverUQ = tf.compat.v1.train.import_meta_graph('parkingsurfaceUnpavedQuality-model.meta')
92
93     # Accessing the graph
94     y_predUQ = graphUQ.get_tensor_by_name("y_pred:0")
95
96     xUQ = graphUQ.get_tensor_by_name("x:0")
97     y_trueUQ = graphUQ.get_tensor_by_name("y_true:0")
98     y_test_imagesUQ = np.zeros((1, len(os.listdir('training_data_unpaved_quality'))))
99
100 sessUQ = tf.compat.v1.Session(graph = graphUQ)
101 saverUQ.restore(sessUQ, tf.compat.v1.train.latest_checkpoint('unpavedCheckpoint/'))
102
```

Figure 4.30 Restoring the unpaved quality model

Using the input image data this script now classifying the surface type and its relevant quality in just a few seconds. Once after the classification we return the all the classified information as object.

```
04 while cv.waitKey(1) < 0:
05     hasFrame, images = cap.read()
06
07     finalimg = images
08
09
10     if not hasFrame:
11         print("Classification done!")
12         cv.waitKey(3000)
13         break
14
15     images = images[newHeight-5:height-50, 0:width]
16     images = cv.resize(images, (image_size, image_size), 0, 0, cv.INTER_LINEAR)
17     images = np.array(images, dtype=np.uint8)
18     images = images.astype('float32')
19     images = np.multiply(images, 1.0/255.0)
20
21     x_batch = images.reshape(1, image_size, image_size, num_channels)
22
23     y_test_images = np.reshape(images, (-1, 3)) #Reshape
24     #
25     feed_dict_testing = {x: x_batch, y_true: y_test_images}
26     result = sess.run(y_pred, feed_dict=feed_dict_testing)
27
28
29     outputs = [result[0,0], result[0,1], result[0,2]]
30
31     value = max(outputs)
32     index = np.argmax(outputs)
33
```

Figure 4.31 Main.py code part 1

```

135     if index == 0: #Asphalt
136         label = 'Asphalt'
137         prob = str("{0:.2f}".format(value))
138         color = (0, 0, 0)
139         x_batchAQ = images.reshape(1, image_size, image_size, num_channels)
140
141         y_test_imagesAQ = np.reshape(images, (-1, 3)) #Reshape
142
143         feed_dict_testingAQ = {xAQ: x_batchAQ, y_trueAQ: y_test_imagesAQ}
144         resultAQ = sessAQ.run(y_predAQ, feed_dict=feed_dict_testingAQ)
145         outputsQ = [resultAQ[0,0], resultAQ[0,1], resultAQ[0,2]]
146         valueQ = max(outputsQ)
147         indexQ = np.argmax(outputsQ)
148         if indexQ == 0: #Asphalt - Standard
149             quality = 'Standard'
150             colorQ = (0, 255, 0)
151             probQ = str("{0:.2f}".format(valueQ))
152         elif indexQ == 1: #Asphalt - Average
153             quality = 'Average'
154             colorQ = (0, 204, 255)
155             probQ = str("{0:.2f}".format(valueQ))
156         elif indexQ == 2: #Asphalt - Bad
157             quality = 'Bad'
158             colorQ = (0, 0, 255)
159             probQ = str("{0:.2f}".format(valueQ))
160     elif index == 1: #Paved

```

Figure 4.32 Main.py code part 2

```

160     elif index == 1: #Paved
161         label = 'Paved'
162         prob = str("{0:.2f}".format(value))
163         color = (153, 102, 102)
164         x_batchPQ = images.reshape(1, image_size, image_size, num_channels)
165
166         y_test_imagesPQ = np.reshape(images, (-1, 3)) #Reshape
167
168         feed_dict_testingPQ = {xPQ: x_batchPQ, y_truePQ: y_test_imagesPQ}
169         resultPQ = sessPQ.run(y_predPQ, feed_dict=feed_dict_testingPQ)
170         outputsQ = [resultPQ[0,0], resultPQ[0,1], resultPQ[0,2]]
171         valueQ = max(outputsQ)
172         indexQ = np.argmax(outputsQ)
173         if indexQ == 0: #Paved - Standard
174             quality = 'Standard'
175             colorQ = (0, 255, 0)
176             probQ = str("{0:.2f}".format(valueQ))
177         elif indexQ == 1: #Paved - Average
178             quality = 'Average'
179             colorQ = (0, 204, 255)
180             probQ = str("{0:.2f}".format(valueQ))
181         elif indexQ == 2: #Paved - Bad
182             quality = 'Bad'
183             colorQ = (0, 0, 255)
184             probQ = str("{0:.2f}".format(valueQ))
185     elif index == 2: #Unpaved

```

Figure 4.33 Main.py code part 3

```

185 elif index == 2: #Unpaved
186     label = 'Unpaved'
187     prob = str("{0:.2f}".format(value))
188     color = (0, 153, 255)
189     x_batchUQ = images.reshape(1, image_size, image_size, num_channels)
190
191     y_test_imagesUQ = np.reshape(images, (-1, 2)) #added 4
192     #
193     feed_dict_testingUQ = {xUQ: x_batchUQ, y_trueUQ: y_test_imagesUQ}
194     resultUQ = sessUQ.run(y_predUQ, feed_dict=feed_dict_testingUQ)
195     outputsQ = [resultUQ[0,0], resultUQ[0,1]]
196     valueQ = max(outputsQ)
197     indexQ = np.argmax(outputsQ)
198     if indexQ == 0: #Unpaved - Average
199         quality = 'Average'
200         colorQ = (0, 204, 255)
201         probQ = str("{0:.2f}".format(valueQ))
202     elif indexQ == 1: #Unpaved - Bad
203         quality = 'Bad'
204         colorQ = (0, 0, 255)
205         probQ = str("{0:.2f}".format(valueQ))
206
207     data = {
208         'label': label,
209         'quality': quality,
210         'probability': prob
211     }
212
213     sess.close()
214     (variable) sessPQ: Any
215     sessPQ.close()
216     sessUQ.close()
217     time.sleep(5)
218
219     return data

```

Figure 4.34 Main.py code part 4

Finally, we have created flask API using the above created classification script to use in the pay as you park web application surface quality identification before the parking place registration (Figure 4.35)

```

221 app = Flask(__name__)
222
223 @app.route("/", methods=["GET", "POST"])
224 def index():
225     if request.method == "POST":
226         url = request.json
227         content = url['url']
228         if content is None or content == "":
229             return jsonify({"error": "no file"})
230
231         try:
232             prediction = predict(content)
233             return jsonify(prediction)
234         except Exception as e:
235             return jsonify({"error 1": str(e)})
236
237     return "OK"
238
239
240 if __name__ == "__main__":
241     app.run(debug=True)

```

Figure 4.35 Flask Implementation

4.1.6 Availability API Implementation

Availability of car parking area is updated and displayed to the car parking owners using two node.js services. The first API updates the availability, and the second API returns availability as a JSON response. The APIs are currently hosted on Heroku App. For the database purpose, MongoDB NoSQL has been utilized since it supports convenient handling than traditional SQL databases for web and mobile applications. ReactJS has been used for the front-end development since the React framework supports various libraries for both user interfaces and back-end service handlings. Also React has a Virtual DOM which is faster than other frameworks.

```
{
  "PID": "P27",
  "Name": "Park Street Car Park",
  "Occupancy": "28",
  "Slots": [
    { "id": "01",
      "availability": true
    },
    { "id": "02",
      "availability": false
    }
  ]
}
```

Figure 4.36 Sample Response

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const ObjectId = mongoose.Schema.Types.ObjectId;

const ParkingAreaSchema = new Schema( definition: {
  user_id: {
    type: ObjectId,
    ref: 'users',
    required: true
  },
  PID: {
    type: String,
    required: true
  },
  Latitude: {
    type: String,
    required: true
  },
  Longitude: {
    type: String,
    required: true
  },
  Name: {
    type: String,
    required: true
  },
  Address: {
    type: String,
    required: false,
    default: ""
  },
  Capacity: {
```

Figure 4.37

```

    Capacity: {
      type: Number,
      required: true
    },
    Occupancy: {
      type: Number,
      required: true
    },
    MaxWidth:{
      type:String,
      required: false,
      default: ""
    },
    slots: [{
      slot_id:{
        type: ObjectId,
        ref: 'slot'
      }
    }]
  }
});

module.exports = ParkingArea = mongoose.model( name: "ParkingArea", ParkingAreaSchema);

```

Figure 4.38

```

const mongoose = require("mongoose");
mongoose.set('useFindAndModify', false);
const Schema = mongoose.Schema;

//create Schema
const SlotSchema = new Schema(
  definition: {
    slotNumber : {
      type : Number ,
      require : true
    },
    availability :{
      type: Boolean ,
      require: false,
      default: false
    }
  },
);

module.exports = Slot = mongoose.model( name: "slot" , SlotSchema);

```

Figure 4.39

Above figures shows the model of node.js APIs. And figure 4.36 shows the availability of the car parking slots response from the API.

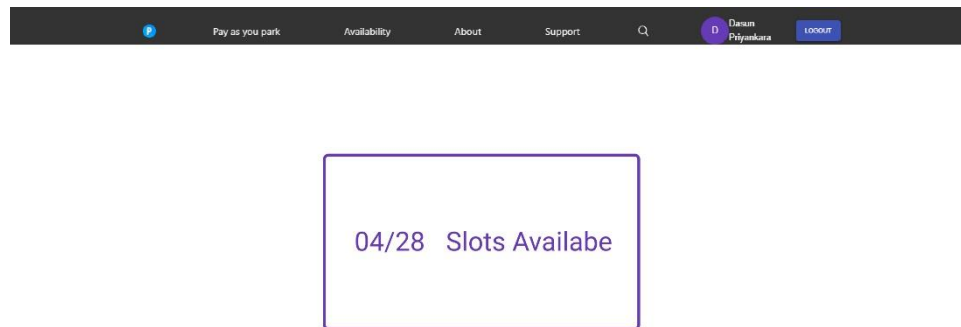


Figure 4.39 Car Park Owner Parking Availability Preview

As in figure 4.39 the car park owner will be able to check the current count of available slots inside his or her car parking area once he is logged in to his registered account.

Availability Detection Using CCTV

To identify the availability of car parking slots inside a car parking area, for the use of driver, car park owner and other members of my research group I have developed an algorithm to afford the target. To process and generate the availability results I have utilized CCTV cameras since most of the car parking areas already have installed CCTV camera systems.

- Deep CNN Implementation

To identify the availability of the car parking slots which means the occupancy of a car parking slot, we have used ImageNet-VGG-f model, a model which has been trained on the ImageNet dataset. This Pre-trained deep CNNs have five convolutional layers, each of which has 11x11, 5x5, 3x3, 3x3, and 3x3 image kernels. These layers stride over the entire image, pixel by pixel (except for the first layer, where the stride is four pixels), to generate 3D volumes of feature maps (with the exception of the first layer, where the stride is four pixels). The first convolution layer has a width of 64 pixels, while the remaining layers have a width of 256 pixels. Following the first, second, and final convolution layers, there is a max-pooling layer. The last convolution layer is followed by three fully connected layers of 4096, 4096, and 1000 neurons, respectively, and the final output is a layer with a soft-max classifier as its final output. It is fairly similar to the network architecture illustrated in Figure 1 in terms of its design. Figure 2 depicts a simplified representation of the framework, which consists of two main components: training an SVM classifier and evaluating the classification results of the SVM classifier. The MATLAB environment has been used here. To train the model PKLot dataset has been utilized since it holds more than 12,000 car parking CCTV images and more than seven hundred thousand segmented parking slot images. Figure 4.40 and Figure 4.40 displays the process.

```
clear;clc; close all;% clear workspace and command window

if ~exist('MATLABCodeCNNSVM.zip','file')
    disp('Downloading file (218 MB)...');
    URL = 'https://melbourne.figshare.com/ndownloader/files/24726374';
    websave('MATLABCodeCNNSVM.zip',URL);
else
    disp('Zip file exists')
end

Zip file exists

unzip('MATLABCodeCNNSVM.zip')
disp('Unzipped files')

Unzipped files

addpath('FinalCodeSVM/')
addpath('FinalCodeSVM/SupportingFunctions/')
```

Figure 4.40 Training model

```
% Visualise the segmented images of PKLot dataset
% (We sampled 1500 occupied and 1500 Empty spaces)
PKLotEmpty = imresize(imread(...
    'FinalCodeSVM\PKLotSegmentedSampled\Empty\2012-09-11_15_45_57#004.jpg'), [70 50]);
PKLotOccupied = imresize(imread(...
    'FinalCodeSVM\PKLotSegmentedSampled\Occupied\2012-09-11_15_36_32#100.jpg'), [70 50]);
figure
subplot(1,2,1)
imshow(PKLotEmpty);
title ('1500 Empty slots')
subplot(1,2,2)
imshow(PKLotOccupied);
title ('1500 Occupied slots.')
```

Figure 4.41 displays how the model has identified the occupancy of the images using the dataset.



Figure 4.41 Occupancy Samples

For the testing purposes a dataset of Barry Street has been used. The code lines in figure 4.42 shows the process of identifying the parking slots using ground truth images and the output is displayed on the figure 4.43.

```

% Load image to train.
BarryStreetImage = imread('FinalCodeSVM\BarryStreetData\DSC_0455.JPG');

% load the parking slot markings and occupancy from ground truth image
load('GroundTruthBarryStreet.mat');

% Visualise Barry Street
figure
BarryStreetImageAnn = insertShape(BarryStreetImage,'rectangle',ParkingSlots,...
    'LineWidth', 2);
imshow(BarryStreetImageAnn);

```

Figure 4.42 Identifying Availability

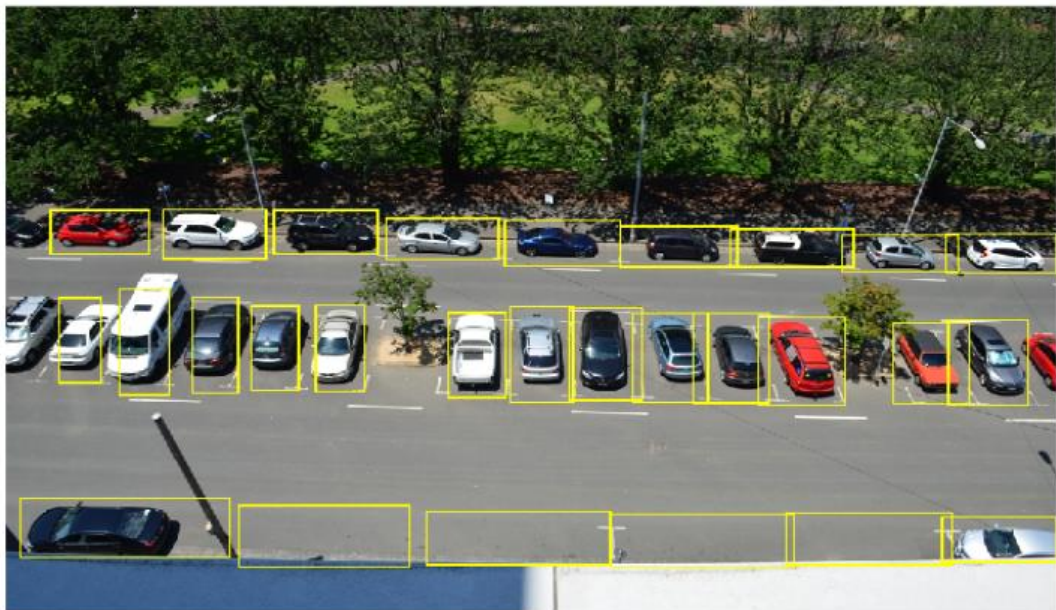


Figure 4.43 Boundary Boxes on Car Parking Slots

```

figure
AnnotatedImage = imread("FinalCodesVM\BarryStreetData\OSC_0169.JPG");
AnnotatedFinal;
emptySlots = 0;
occupiedSlots = 0;

emptySlotArray = [];
occupiedSlotsArray = [];

for n=1:28
    if YpredClass(n) == "Empty"
        AnnotatedImage = insertShape(AnnotatedImage,'FilledRectangle',...
        Parkingslots(n,:), 'LineWidth', 1, 'Color', "green", 'Opacity', 0.4);
        AnnotatedImage = insertText(AnnotatedImage, [Parkingslots(n,1) Parkingslots(n,2)], ...
        n, 'FontSize', 18, 'TextColor', "white", 'BoxOpacity', 0.0);
        emptySlotArray = [emptySlotArray, n];
        emptySlots = emptySlots + 1;
    else
        AnnotatedImage = insertShape(AnnotatedImage,'FilledRectangle',...
        Parkingslots(n,:), 'LineWidth', 1, 'Color', "red", 'Opacity', 0.4);
        AnnotatedImage = insertText(AnnotatedImage, [Parkingslots(n,1) Parkingslots(n,2)], ...
        n, 'FontSize', 18, 'TextColor', "white", 'BoxOpacity', 0.0);
        occupiedSlotsArray = [occupiedSlotsArray, n];
        occupiedSlots = occupiedSlots + 1;
    end

    %disp(Parkingslots(n,:));
    %disp(Parkingslots(n,2));
end
imshow(AnnotatedImage)
text (100, 50, ['Occupied slots: ' num2str(occupiedSlots)], 'Color', 'red', 'FontSize', 20);
text (100, 100, ['Empty slots: ' num2str(emptySlots)], 'Color', 'green', 'FontSize', 20);

```

Figure 4.44 Identifying occupancy



Figure 4.45 Output

Figure 4.45 in the above displays the final outcome of dataset. It takes an image and displays the occupancy of the vehicle parking slots and the id of the particular parking

slot. After identification of parking slots, the parking slots occupancy update service will be called, and the API will update the mongo DB database's data.

4.2 Testing

Testing is a compulsory phase in software development. Since parking is an essential task accuracy and the efficiency of the application must test. I have performed the individual unit testing as well as integrated testing in the application. Since we are considering the privacy of the research, we provided our application with only a limited number of users who used this application to test the location tracking and parking yard suggestions.

Testing of Optimal Parking Yard Suggestions:

Following table 2 demonstrate summary of Unit testing done by our team.

Function Process	Issues Yes/No
Haversine testing	No
API testing	No
Physical Characteristics Validation	No
SARIMA testing	No
Location Tracking	No
Direction Suggestion	No
Integrated Testing	No

Table 2 Test Results Summary

Testing of Internal Parking Navigation

Testing is a required step in the software development process. Because parking is such an important duty, the application's accuracy and efficiency must be tested. In the application, I performed both individual unit testing and integration testing. We offered our application to only a restricted number of users who used our application to test the location tracking inside the parking lot because we care about the privacy of the research.

Following table demonstrate summary of the testing which I have done

Function Process	Issues Yes/No
API testing	No
Location Tracking	No
Integrated Testing	No

Table 2.1 Testing Issue Clarification

Testing of Parking Surface Classification

For the training purposes, more than 6000+ frames were used from our dataset to train our model for parking surface classification, which were manually divided into three categories as asphalt, paved, and unpaved. All training data were divided into two parts such as 80% for training and 20% for validation. The number of frames chosen for each class was proportionate to the amount of data available. Around 70% of the frames are asphalt, 20% are paved, and 10% are unpaved.

Model Name	Training Accuracy	Validation Accuracy	Validation Loss
Surface Type Model	100%	93.8%	0.146

Table 2.2 Parking surface type model accuracy

Testing of Parking Surface and Quality Classification

Function	Issues Yes/No
API testing	No
Availability Testing	No
Integrated Testing	No

When

consider the surface quality, this method used three models for the surface quality classification which is used to identify the quality of each surface type. In the training stage, we have used a few frames: 1500+ frames for the asphalt quality model, 600+ frames for the paved quality model, and 500+ frames for the unpaved quality model. We separated 80 percent of the data for training, and the remaining 20 % was picked at random for validation.

Model Name	Training Accuracy	Validation Accuracy	Validation Loss
Asphalt quality model	100%	88.4%	0.011
Paved quality model	100%	96.9%	0.190
Unpaved quality model	100%	92.7%	0.206

Table 2.3 Parking surface quality model accuracy

Testing of Parking Slots Availability Identification

For the training use I have used PKLot dataset which has more than 12000 CCTV images and more than seven hundred thousand of segmented parking slot.

Table 2.4 Accuracy of Slot

Training Accuracy	Validation Accuracy	Validation Loss
99.83%	98.89%	0.94

Identification

Table 2.5 Testing of Slot Identification

5. RESULT AND DISCUSSION

5.1 Results

My research component is to suggest optimal parking yards to the users. To achieve that I have found the factors that make impact on parking yard suggestions. As per the research conducted, I was able to gain UI/UX and data accuracy.

1. Users don't need to consider about availability of parking yard
2. Simple UI to improve use experience

As a result of the research, I was able to come up with an algorithm which can derive the suggestions for a user to park the vehicle with:

1. Simple mobile app
2. Less screens
3. Accurate suggestions

Following graph shows the predicted result and trained data in the dataset. The predicted result got a high accuracy with data set and only a minor error.

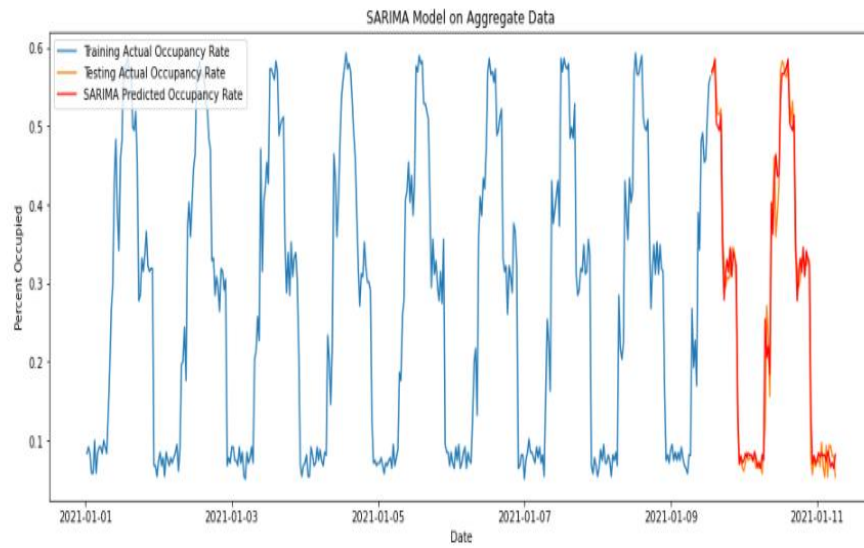


Figure 5.1 SARIMA Graph

As shown in the figure 5.1 the predicted results are mostly like the trained results. Its verdict that the model predicts the occupancy rate mostly like the actual occupancy of parking yard.

```
-----
SARIMA Model Metrics on Test Data
=====
Explained Variance:
    0.9771461256899351
MAE:
    0.01961308780436944
```

Figure 5.2 SARIMA Results

Figure 5.2 demonstrate the model explained variance and mean absolute error. The model got an explained variance above 97% and error rate lower than 0.02. These results verdict the model got a high accuracy.

When considering getting the user's positions Authors have been used many algorithms such as DNN, NN, and XGBoost algorithms. Authors have used NN to train the model due to its high accuracy. DNN and XGBoost algorithms are low inaccuracy when getting the actual distance of a user. By using about 300 frames from a dataset, authors have been trained the models and the trained model accuracy of getting X value of a user is 90% and the accuracy of getting Y value of a user is 100%. Accuracy is shown in Table 5.3

Model Name	Training Accuracy of X	Training Accuracy of Y
Neural Network model	100%	84%

Table 5.3 Model Accuracy

5.2 Research Findings

Our research is based on a mobile application, and we implemented using flutter so both the android and iOS users can access the application. Our aim is to provide a better parking experience for the users in parking process.

With research conducted along with experts and end users it helps us to derived below things,

- Better UI/UX experienced theme for the application
- Facts impacting the parking yard suggestions
- Main problems in parking suggestions

- Effect of unethical parking
- Internal navigation approaches
- Mechanism to validate a parking yard
- Track availability with the CCTV cameras in the parking yard

When we consider on the existing solution for parking, they are using sensor-based architecture which would be very expensive. But in the presented solution it would track the availability of parking yard with the use of CCTV cameras in the parking yard.

When it comes to parking yard suggestion, we would be providing a better UI/UX for the users to search for park along with the maps and user tracking features. Here our parking yard suggestion are generated by considering the all-key factors impact on parking yard suggestions. And a beacon based internal navigation system is introduced with the paper to navigate inside the parking yard.

5.3 Discussion

Throughout my research my primary goal was to provide a better parking for the users who faced different kind of problems in their daily routine. There are factors that affect on the parking yard suggestions with my research I was considering deriving a solution which will minimize of those impacts on those parking yard suggestions. Also, I was focused on the real time mobile application development which provide the end users a better UI and UX. When it comes to mobile level application our team focused on using a cross platform mobile application development where we can ignore the native development. It was added as a big advantage to the research. And to evaluate the key factors impact on parking I have used haversine and SARIMA model as it gives the accurate result with least computational cost. We focused on using latest tech stack evolving in the industry which would give advantage to the application level. When it

comes to parking yard suggestions, I got attention on what the most users want and what are the problems they are faced in their parking activity. As per the response of users gained by the users, it clearly demonstrated that parking suggestions are not accurate and optimal in the existing parking yard suggestions. So, I focused more on the problems and factors that cause the problems and implemented an algorithm including the model derived through haversine and SARIMA. Also, we decided to use node.js as server application where we gained the endpoints to our mobile application. Node.js API application implemented in Express.js framework which connects with Mongo DB and performs database operations. Mongo DB plays a key role in this application since it provides high availability, accuracy and efficiency. Through the solution that I have implemented for parking yard suggestions; we can use this component in final product to provide users a better parking experience which may result in a positive way to environment and society.

6. SUMMARY OF THE STUDENT CONTRIBUTION

Member	Component	Task
M.D.S.M. Antany	Parking yard quality validation	<ol style="list-style-type: none"> 1. Analysis of the methodology 2. Implementing the surface type classifying models using CNN 3. Implementing the three surface quality classification models using CNN 4. Flask API Implementation and deploying the classification models in the local server 5. Web application implementation using MERN Stack 6. Implementing the Parking owner login/registration using Google auth and JWT web token 7. Implementing the parking yard validation and registration using the classification models
Aadil M.R.M	Suggesting nearest optimal parking yard	<ol style="list-style-type: none"> 1. Analysis of parking yada data 2. Model implementation with SARIMA to

		<p>forecast the occupancy</p> <ol style="list-style-type: none"> 3. Algorithm implementation to suggest the optimal parking yard to the user 4. Mobile application server implementation and deployment to Heroku 5. Mobile application development for register user, login, suggest parking yard with flutter. 6. Fast API implementation to deploy model with AWS EC2 instance.
Ferreira L.V.	Internal Parking Navigation	<ol style="list-style-type: none"> 1. Analyze of distance calculation by getting beacon RSSI values. 2. Model Implementation for get user's position by using Neural Network algorithm 3. Mobile Application particular server implementation for the component and deployment in Heroku 4. Mobile Application

		development to view user's position
Priyankara A.D.D	Car Parking Slot Occupancy	<ol style="list-style-type: none"> 1. Model implementation of availability of car parking slots in a car parking area using CCTV camera. 2. Web application development to view availability to car park owners 3. Web Service to update and view availability on database and deployment on Heroku.

Table 6.1 Student Contribution Summary

7. CONCLUSION

Divers and parking lot owners can use this application to get a fantastic smart parking solution experience while earning some revenue and prizes. Features such as registering people's own lands and garages, with the exception of existing parking yards, will broaden the business case and increase the availability of current services to end customers. To accelerate this process of increasing the market with the maximum growth, it is necessary to give a dependable service on a continual basis by earning the trust of people. This item describes the process of standardizing parking spaces prior to registering with the system. Using the mobile application to perform parking users can get the optimal and accurate parking yard suggestions to perform their parking in effective and efficient way. As a result of the estimates obtained, it is apparent that BLE technology is the most optimal method for indoor navigation. Bluetooth Low Energy provides the finest mix of efficiency, ease of implementation, and pricing among all technologies. To build an interior navigation system, BLE is a must-have technology nowadays. Internal parking of two or more stories will be completed in the future. By using the parking mobile application, users can receive the optimal internal parking system to do their parking in an effective and efficient manner. They can use the application by accessing it with an Android or iOS mobile smartphone. Furthermore, the program will be readily available, and users will be able to save time by utilizing it. Most individuals are having issues because they have a variety of issues when they enter a parking lot, and my approach addresses those concerns. We employed the latest technology stack and focused on the performance of our application to provide a better experience for our customers. Adoption of a 'Pay as you Park' smart parking solution can alleviate users' difficulties in finding an available parking slot inside a parking area, and the parking process can be improved, benefiting both the environment and society. With the use of a 'Pay as you Park' smart parking solution, users' concerns with finding a yard to park may be reduced, and the parking procedure can be simplified, benefiting the environment and society. As soon as the product is for placement in a live environment and since this is a considerable problem

in Sri Lanka parking systems the users can get an advantage after the user pays the fee. According to the current parking systems in Sri Lanka, User has parking time of thirty minutes by thirty minutes payment packages or hour by hour payment packages. Consider if a user parked a vehicle in a parking area for ten minutes user will be charged for whole thirty minutes. As a unique feature in this proposed system, the user only must do the payment for the exact parking time.

8. REFERENCES

- [1] Bhavani, D. S., & Ghalib, M. R. "Internet of Things Based Smart Car Parking System Using K-Nearest Neighbor Algorithm to Find the Nearest Slot. *Journal of Computational and Theoretical Nanoscience*, "2018, 15(6), 2040–2045. doi:10.1166/jctn.2018.7403
- [2] M. A. P. Chamikara, Y. P. R. D. Yapa, S. R. Kodituwakku and J. Gunathilake, "An Efficient Algorithm to Detect The Nearest Location Of A Map For A Given Theme," 2013 *International Journal of Scientific & Technology Research*.
- [3] Y. Dian Harja and R. Sarno, "Determine the best option for nearest medical services using Google maps API, Haversine and TOPSIS algorithm," 2018 *International Conference on Information and Communications Technology (ICOIACT)*, Yogyakarta, Indonesia, 2018, pp. 814-819, doi: 10.1109/ICOIACT.2018.8350709.
- [4] Siahaan, Andysah P. U. 2017. "Haversine Method in Looking for the Nearest Masjid." *INA-Rxiv*. September 22. doi:10.31227/osf.io/eb3ja.
- [5] M. Abinaya and R. Ganesan, "Effective search mechanism for finding nearest healthcare facilities," 2015 *Global Conference on Communication Technologies (GCCT)*, Thuckalay, India, 2015, pp. 534-538, doi: 10.1109/GCCT.2015.7342719.
- [6] I. Fokker ES, Koch T, van Leeuwen M, Dugundji ER. Short-Term Forecasting of Off-Street Parking Occupancy. *Transportation Research Record*. September 2021. doi:10.1177/03611981211036373
- [7] S. Shinde, A. Patil, S. Chavan, S. Deshmukh and S. Ingleshwar, "IoT based parking system using Google," 2017 *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, 2017, pp. 634-636, doi: 10.1109/I-SMAC.2017.8058256.
- [8] Ibrahim, Hossam El-Din, *Car Parking Problem in Urban Areas, Causes and Solutions* (November 25, 2017). 1st *International Conference on Towards a Better Quality of Life*, 2017, Available at SSRN: <https://ssrn.com/abstract=3163473> or <http://dx.doi.org/10.2139/ssrn.3163473>
- [9] M. M. Forrest, Z. Chen, S. Hassan, I. O. Raymond and K. Alinani, "Cost Effective Surface Disruption Detection System for Paved and Unpaved Roads," in *IEEE Access*, vol. 6, pp. 48634-48644, 2018, doi: 10.1109/ACCESS.2018.2867207.
- [10] Nienaber, S & Booysen, M.J. (Thinus) & Kroon, Rs. (2015). *Detecting Potholes Using Simple Image Processing Techniques and Real-World Footage*. 10.13140/RG.2.1.3121.8408.

- [11] Taluja, Chandan & Thakur, Ritula. (2018). *An Intelligent Model for Indian Soil Classification using various Machine Learning Techniques*. 2250-3005.
- [12] Lee, T.; Yoon, Y.; Chun, C.; Ryu, S. *CNN-Based Road-Surface Crack Detection Model That Responds to Brightness Changes*. *Electronics* 2021, 10, 1402. <https://doi.org/10.3390/electronics10121402>
- [13] Varona, B., Monteserin, A., & Teyseyre, A. (2019). *A deep learning approach to automatic road surface monitoring and pothole detection*. *Personal and Ubiquitous Computing*. doi:10.1007/s00779-019-01234-z
- [14] Batistic, L., & Tomic, M. (2018). *Overview of indoor positioning system technologies*. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*.
- [15] Intelligent Indoor Parking Győző Gódor*, Árpád Huszák* and Károly Farkas†, *† Inter-University Centre for Telecommunications and Informatics, Debrecen, Hungary *Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary Email: {godorgy, huszak, farkask}@hit.bme.hu
- [16] Improve Indoor Positioning Accuracy Using Filtered RSSI and Beacon Weight Approach in iBeacon Network Laial Alsmadi Faculty of Engineering and IT University of Technology Sydney Sydney, Australia Xiaoying Kong Faculty of Engineering and IT University of Technology Sydney Sydney, Australia Kumbesan Sandrasegaran Faculty of Engineering and IT University of Technology Sydney Sydney, Australia
- [17] Improve Indoor Positioning Accuracy Using Filtered RSSI and Beacon Weight Approach in iBeacon Network Laial Alsmadi Faculty of Engineering and IT University of Technology Sydney Sydney, Australia Xiaoying Kong Faculty of Engineering and IT University of Technology Sydney Sydney, Australia Kumbesan Sandrasegaran Faculty of Engineering and IT University of Technology Sydney Sydney, Australia
- [18] Smart Parking System Based on Bluetooth Low Energy Beacons with Particle Filtering Andrew Mackey, Student Member, IEEE, Petros Spachos, Senior Member, IEEE, and Konstantinos N. Plataniotis, Fellow, IEEE
- [19] Bluetooth-based Indoor Navigation Mobile System Adam Satan Institute of Information Science University of Miskolc Miskolc, Hungary satan@iit.uni miskolc.hu. Satan, A. (2018). *Bluetooth-based indoor navigation mobile system*. 2018 19th International Carpathian Control Conference (ICCC)
- [20] Nienaber, S & Booysen, M.J. (Thinus) & Kroon, Rs. (2015). *Detecting Potholes Using Simple Image Processing Techniques and Real-World Footage*. 10.13140/RG.2.1.3121.8408.
- [21]. Taluja, Chandan & Thakur, Ritula. (2018). *An Intelligent Model for Indian Soil Classification using various Machine Learning Techniques*. 2250-3005.

[22]. Mingxing Gao, Xu Wang, Shoulin Zhu, Peng Guan, "Detection and Segmentation of Cement Concrete Pavement Pothole Based on Image Processing Technology", *Mathematical Problems in Engineering*, vol. 2020, Article ID 1360832, 13 pages, 2020. <https://doi.org/10.1155/2020/1360832>