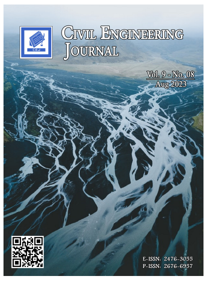
Available online at www.CivileJournal.org

Civil Engineering Journal

(E-ISSN: 2476-3055; ISSN: 2676-6957)

Vol. 9, No. 08, August, 2023



IoT Raspberry Pi Based Smart Parking System with Weighted K-Nearest Neighbours Approach

Md Shohel Sayeed [1*], Huzaifah Abdulrahim [1], Siti Fatimah Abdul Razak [1], Umar Ali Bukar [1], Sumendra Yogarayan [1]

[1] *Faculty of Information Science and Technology, Multimedia University, 75450 Melaka, Malaysia.*

Received 10 January 2023; Revised 06 July 2023; Accepted 13 July 2023; Published 01 August 2023

Abstract

Due to the limited availability of parking slots in parking areas, drivers often have difficulty finding an empty parking slot. The number of parking slots available at a particular location is usually less than the number of vehicles. Hence, drivers spend a lot of time looking for vacant parking slots, which eventually delays the completion of their tasks, such as paying bills, attending a meeting, or visiting a patient at the hospital, etc. There are a couple of parking guidance systems that have been highlighted by the other researchers, but most of them lack real-time, convenient guidance. This research proposed a smart parking guidance system made of an IoT Raspberry Pi combined with an Android application that makes use of the weighted k nearest neighbours for positioning the vehicle. This was achieved through the use of Wi-Fi signal strength indicator fingerprinting, allowing for real-time navigation and parking detection. In order to achieve real-time parking over the internet, Raspberry Pi hardware and the ThingSpeak IoT cloud with ultrasonic sensors are used in the proposed method. An Android application was involved in this parking detection system, which adopted IoT approaches to estimate the location of users in real-time and provide routes using route-finding techniques to assist drivers in finding their desired parking slots. Data from the sensors was processed and translated into the Raspberry Pi using the Python programming language. They were sent using the Message Telemetry Transport protocol to send parking data to the ThingSpeak IoT cloud in real-time. This data was displayed via the Android app. The user is then able to view each available parking slot, acquire the route, and be directed with high accuracy to the parking slots of their choice. In this study, advanced sensing and communication technologies were used together with the weighted k nearest neighbours algorithm for positioning and wayfinding in order to improve parking guidance accuracy. Based on the experimental results, the proposed system showed a lower average error rate of 1.5 metres in comparison to other positioning techniques, such as GPS, or other similar algorithms for positioning, such as maximum a posteriori, which have shown average errors of 2.3 metres and 3.55 metres, respectively, a potential increase of more than 35% from the previous error rate.

Keywords: Smart Parking; IoT; WKNN; Parking Guidance; Raspberry Pi; Positioning System.

1. Introduction

Every year, technology is revolutionising our lives and making them more comfortable, especially with the introduction of the Internet of Things (IoT) and its potential to benefit humanity. Moreover, the accelerated activity and innovation in IoT on the factory front, in terms of how cyber-physical systems improve productivity in the manufacturing process as well as how we interact with things in our daily lives, indicates that the technology has broad potential in the future. The Internet of Things (IoT) plays an increasingly important role in connecting the things within our environment to the Internet and makes our lives easier due to the ability to access these non-internet items from any location (outside

* Corresponding author: shohel.sayeed@mmu.edu.my

http://dx.doi.org/10.28991/CEJ-2023-09-08-012

our local network), regardless of their location. Keeping up with the growing technology is challenging as it grows more rapidly than expected.

The Internet of Things has the potential to solve a variety of problems, including the ability to find vacant parking spots in a parking area. Since high production and demand for vehicles have led to a high number of cars being produced over the years, most people find this to be a significant challenge, particularly in malls and shopping markets. It could also be due to the size of the parking area being too big or multi-floored. With the tremendous developments in sensors over the last few years, finding solutions to such problems has become easier, especially after the development of many sensors that can handle a variety of issues, including measuring temperature, pressure, motion, and intrusions. In the case of parking sound wave sensors, they are capable of converting electrical energy into acoustic energy, and vice versa, in order to detect vehicles parking. They may also be used for level monitoring, for example, in dairy factories to monitor the level of fill in silos and tanks.

The sound wave sensor can also serve as an object detector to detect the presence or absence of an object. Using a sensor called the ultrasonic wave sensor, this sensor can detect objects by measuring distance using sound waves. Each parking slot will have an ultrasonic wave sensor installed, which sends the data to a Raspberry Pi microcontroller, which in turn sends it to the cloud. Empty parking slots can thus be detected and indicated to the user through an application that gets the data from the cloud. Therefore, the user is able to know which slots are empty and which are not. This application is not only limited to that—since the system can be used over the internet, the user does not need to be in the same network vicinity as the parking area, thanks to the cloud technology, which allows the system to be accessed from anywhere.

2. Related Work

2.1. Technologies Used in Positioning Systems

2.1.1. Wi-Fi Access Points

Wi-Fi uses radio waves to send and receive data based on the 802.11 family of standards, which is used for Local Area Networking (LAN). Wi-Fi networks can be described as a group of devices, also known as nodes, that use wireless data connections to communicate data, typically from the Internet. Wi-Fi access points amplify the signal so that a device that is far from them can still be connected to the network. Recently, with the huge development in Wi-Fi technology, it can be used in indoor positioning systems. In fact, Wi-Fi is considered one of the most accurate solutions when it comes to indoor positioning. Currently, every IoT device has a Wi-Fi adapter, which allows multiple methods for tracking such a device using a Wi-Fi network, but the two primary methods are Received Signal Strength Indication (RSSI) and fingerprinting.

The RSSI of a Wi-Fi signal is proportional to its distance and is used to determine a device's location, but since it uses signals, just like most signals, it is affected by obstructions and reflections, which can sometimes lead to inaccurate measurements. The second method, fingerprinting, is based on RSSI, but the signal power of many access points is recorded in a database along with the known coordinates of the access points and the client machine. Fingerprinting, however, can significantly improve accuracy. There are other techniques like Angle of Arrival (AoA) and Time of Flight (ToF) that can also determine locations accurately, but they require expensive equipment such as antennas and clock synchronisation.

Wi-Fi is accessible, has a wide range, has a high data throughput, and, thanks to current technology, is widely available. Hence, it is used in indoor localisation as there can be multiple Wi-Fi access points in a room or a floor, and generally, the more access points in a particular location, the more accurate the localisation is. Due to its wide range, to implement a Wi-Fi network for positioning, it may require little to no infrastructure costs. These Wi-Fi access points have the same concept as GPS satellites.

2.1.2. Bluetooth Beacons

Bluetooth technology is another way to determine an asset's location. Bluetooth beacons work similarly to Wi-Fi access points, and they are used in some systems to build a localisation system. Bluetooth Low Energy (BLE) beacon devices are built based on Bluetooth wireless technology. Beacons are mounted on objects, structures, walls, ceilings, and other places, from where they emit radio signals at predetermined intervals. Devices within the emission area are then able to detect the signals, and this is established if the two (emitter and receiver) are within range of each other [1]. This tracking technique operates in the same way as the Wi-Fi location tracking technique.

RSSI can be used to estimate the distance between the Bluetooth beacons, and of course, the more beacons installed at a location, the more accurate the positioning is. Although Bluetooth beacons do not have built-in location intelligence, they have standard protocols to determine which beacon is transmitting or receiving in order to determine the location of an asset. However, before using Bluetooth, its radius of coverage needs to be considered, which is generally smaller than that of Wi-Fi. For some cases, Wi-Fi is much more efficient; this efficiency allows a positioning accuracy of half a metre.

2.1.3. Ultra-Wide Band UWB

Ultra-Wideband (UWB) is a wireless radio technology communication technology that has positioning applications as it requires low power consumption, yet provides high bandwidth connectivity. UWB sends information at a wide bandwidth rate that is greater than 500 MHz, with a frequency range between 3.1 to 10.6 GHz, using short pulses for data transmission [2]. The precision of UWB is considered to be greater than that of Wi-Fi and Bluetooth technologies, as it is able to achieve an accuracy of up to 10 cm, which is considered pinpoint accuracy. The particular reason for this is the use of a wide bandwidth that is not affected by multipath, signal fading, and interference, thus overcoming some of the limitations of other technologies because it can penetrate objects.

Nevertheless, although it has advantages over other technologies, UWB also has its own limitations, which are generally the requirement of its own hardware, which makes it infrastructure-dependent as scalability, costs, and coverage area, among others, need to be decided [3]. In addition to that, the UWB data transfer rate is lower than for Bluetooth and Wi-Fi, it is more highly priced for location tracking, and some regulations limit its operation range as some frequencies are still used by civilians and the military. UWB can be used in different applications other than positioning as well, such as tracking, navigation, and peer-to-peer ranging that allows distance calculation between two assets [4] (see Table 1).

Table 1. Comparison between Technologies Used in Positioning Systems

Technology	Used method	Application	Precision	Advantages	Disadvantages
GPS	Trilateration	Outdoors	3-6 m	- Widely used for outdoor positioning. - GPS sensor is embedded in almost every device.	- Can be affected by environmental changes and obstacles. - Can't be used indoors due to its inaccuracy.
Wi-Fi	RSS fingerprinting	Indoors and outdoors	1-3 m	- New infrastructure is not needed. - Pre-installed Wi-Fi access points can be reused. - Commonly used due to its high data transfer rate and low cost.	- The need to integrate with other algorithms to improve accuracy. - Can be affected by obstacles and interference. - Limited range depending on antennas.
Bluetooth beacons	RSS fingerprinting	Indoors	1-3 m	- Low battery consumption.	- Low range and data transfer rate. - Not cost efficient.
Ultra-wideband		Indoors	Up to 30 cm	- Pinpoint accuracy. - Not affected by obstacles. - Wide bandwidth rate. - Low power consumption.	- Infrastructure-dependent. - Complex. - Frequency is regulated by the government and has low transfer rate.

2.2. Comparison between the Various Existing Parking Guidance

There is a need for parking systems as one of the concerns in smart cities. In addition, even if they know the location, they cannot go to the parking location without a guarantee of slot availability, as they risk their valuable time. Nowadays, searching for car parking is becoming the greatest challenge in large cities during peak hours [5].

Therefore, several parking systems have been developed using a variety of approaches and technologies in order to address this issue. Recent parking guidance systems use various types of technologies and approaches to implement the system and make them appropriate for certain scenarios. This section will discuss various smart parking guidance systems that have been recently developed in addition to some of their potential limitations as well as advantages.

When talking about parking systems or smart parking guidance, a lot of the systems include e-parking techniques, like in the proposed system IoT E-Parking System [6]. The system proposes to have a component called Parking Metre (PM), which consists of multiple components like ultrasonic sensors, Arduino, cameras, Wi-Fi models, solar panels, and more, which can be effective but not really efficient. This is because the cost of one PM increases with the quality of its components, which is a concern when considering large parking areas. Simpler components can be found in an IoT-based parking system using Google [7], where the system proposes simpler modules that are connected to an IoT cloud. A simple interface is provided to view the available parking area, but the guidance aspect is reliant on Google Maps, which guides the user to the parking area but not to the specific parking slot, which means the drivers need to find a parking slot themselves among the many.

On the other hand, some systems take advantage of the tremendous development in computing power and microcontrollers by using a fairly powerful microcontroller, the Raspberry Pi, which is used in a lot of IoT-related solutions by allowing easy and reliable connectivity between the system sensors and the IoT cloud in an easy way, as

Pham et al. [8] proposed in their system, the Internet of Things-Based Smart Vehicle Parking Access System. They used a Raspberry Pi as a server to connect the sensors to the cloud, in addition to an LED screen that is used to guide the driver to an available parking slot with the help of green and red-light indicators. Meanwhile, Salma et al. [9] used a Raspberry with one 360-degree camera to detect multiple parking slots. The testing was held in an outdoor prototyped environment, ultimately proposing that the solution is good for parking area monitoring but is affected by the surroundings, e.g., structural obstacles and lighting. In addition, if one camera fails, the user loses monitoring capability for whatever parking slot it was viewing.

The most recent systems used by Thakre et al. [10], Kodali et al. [11], and Zhao et al. [12] suggested making use of Radio-Frequency Identification (RFID), Long Range Radio (LoRa), and ultra-wideband (UWB), respectively, for parking detection. Thakre et al. [10] encourage each user to acquire RFID tags after the registration process and focus only on counting the available parking slots and having the information displayed to the user via LED. The LoRa technology used in Kodali et al. [11], on the other hand, has a larger coverage with low power consumption, and although it can indeed be suitable when implemented in long-distance areas, it still has some limitations—data transmission rates in LoRa are very low, while its receivers can only receive data from one transmitter at a time, requiring more sensors and transmitters to be added, increasing detection delay and unreliability, and thus leading to a large-scale performance drop in positioning estimation. On the other hand, UWB in Zhao et al. [12] is known to provide pinpoint-accuracy positioning. However, similar to LoRa, it is limited by low data transmission rates, requires the purchase of extra equipment, such as a UWB positioning bracelet, and is infrastructure-dependent in some scenarios, which makes it very expensive to implement.

Most of the previously mentioned systems focus on detecting and counting parking slots except for S-Park [13], which provides a map and uses GPS to direct the user to the parking area. However, GPS accuracy is considered lower than other technologies like Bluetooth beacons [14], UWB, and Wi-Fi. A lot of smart parking systems use these technologies in other applications but do not use them for navigational parking or make use of existing Wi-Fi technology indoors or outdoors to make the parking experience easier. In general, most parking guidance solutions for parking areas lack real-time guidance that guides drivers to their desired parking slot. The system proposed by Aditya et al. [15] has been implemented using Raspberry Pi, NodeMCU, Radio Frequency Identification (RFID), and Infrared (IR) sensors.

A lot of systems have been proposed by researchers to improve the parking management system using different technologies such as IoT, Machine Learning, Deep Learning, Image Segmentation, etc. [16–18]. In order to enhance the security and privacy of the system, Ali and Khan [19] propose a hybrid approach that combines machine learning algorithms with trust management in order to make the system more secure and private. This approach consists of SVM and ANNs, taking into account credibility, availability, and honesty as key parameters. In the context of a smart city scenario, an intelligent transport system for IOV-based vehicular network traffic is proposed by Rani and Sharma [20] based on Decision Tree, Random Forest, Extra Tree, and XGBoost machine learning models.

3. Methodology

3.1. IoT-Based Smart Parking Guidance System

3.1.1. System Architecture

The suggested system consisted of two main parts, namely the parking system and the positioning system. The parking system involved two main components, as previously mentioned, which are ultrasonic sensors and the Raspberry Pi Model 3. The Pi was responsible for handling data collection from the connected ultrasonic sensors and sending it to the IoT cloud. Then, the Android application received the collected data from the cloud to enable the user to identify and see the count of available parking slots. In addition, the application also acted as a positioning system to help drivers be routed to their desired parking slots using the app navigator, and this was done by retrieving a preregistered set of RSS signals for the area as well as its radio map. After the retrieval, the WKKN algorithm was used to estimate the user's location, where the route could be established. The programme used to get the distance from the ultrasonic sensors and translate it into Available or Occupied status was written in Python using a Raspberry Pi, and the connectivity between the Pi and the IoT cloud was made using the MQTT protocol. This particular protocol was used as it is one of the most popular ones used in IoT and can also be used in machine-to-machine connectivity as it is fast and does not need a large bandwidth.

Java was used for client-side to get a post message from the cloud, allowing the application to receive the data from the IoT cloud and then present this data in an appropriate way to the user. The user was able to search for the nearest parking area as long as it was registered in the database, where the area floor plan, RSS values, and radio maps were also located. Upon selection of the parking area, the users were able to choose their desired parking slot, and then the routes and other previously mentioned information were retrieved, allowing the user to experience a better parking experience. Figure 1 shows the overall architecture of the proposed system, while Figure 2 shows the overall system flow chart.

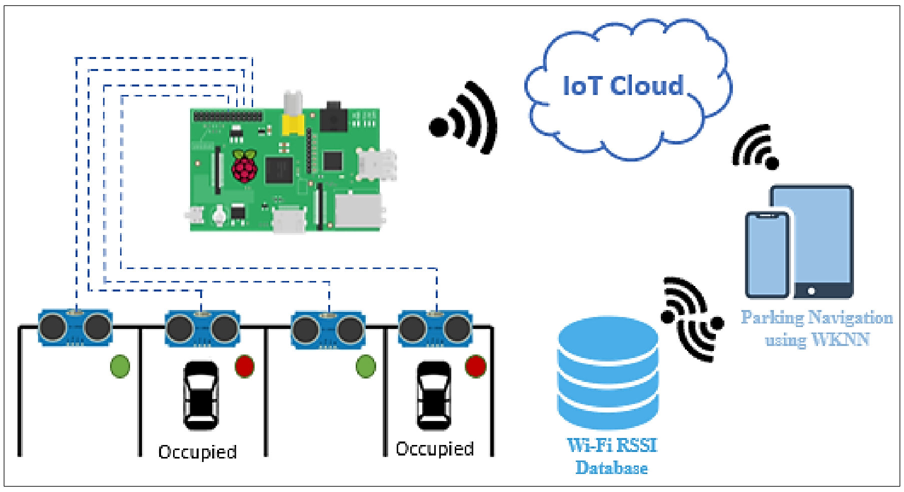


Figure 1. System Architecture of IoT Based Smart Parking Guidance System

3.1.2. System Flow Chart

Figure 2 shows how the system works from the user's perspective. It started with viewing the nearest parking area based on their location; if the parking areas were not shown, the user could press a button to view them. After the areas were shown, the user chose the desired area to check its parking availability. Upon checking and choosing the desired parking spot, the floor plan, user's location, and routes were retrieved from the server and displayed to the user for navigation. Lastly, the status of the chosen parking spot was changed to Occupied.

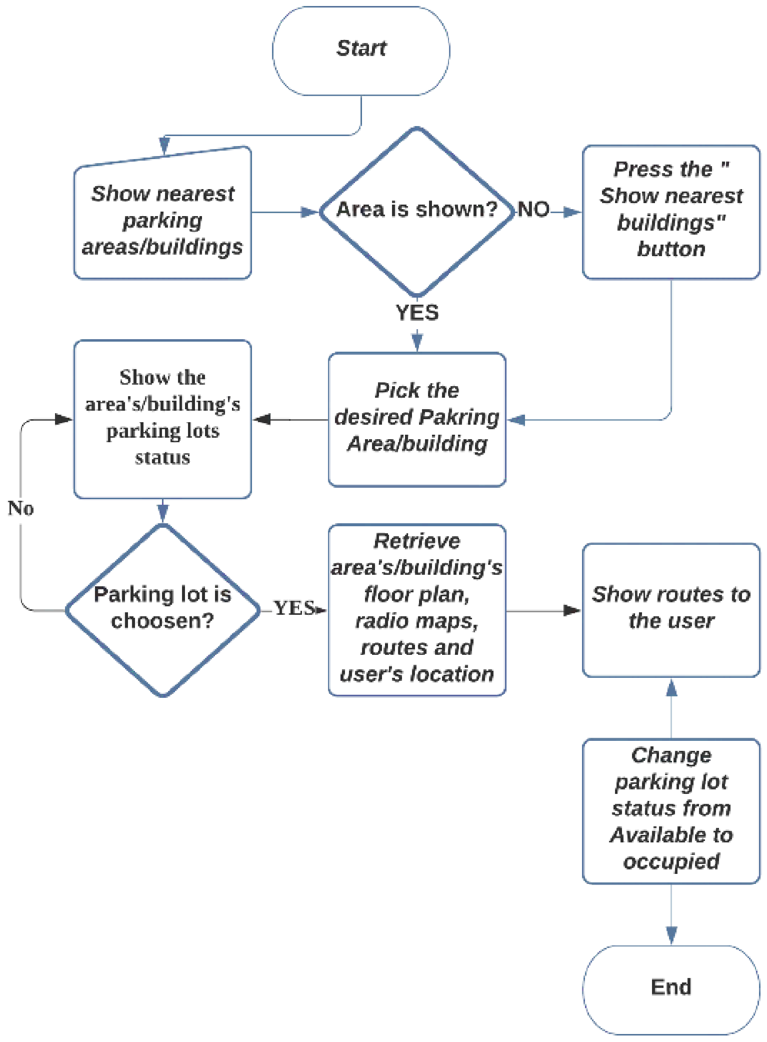

Start
Show nearest parking areas/buildings
Area is shown?
NO
Press the " Show nearest buildings" button
YES
Show the area's/building's parking lots status
Pick the desired Pakring Area/building
No
Parking lot is choosen?
YES
Retrieve area's/building's floor plan, radio maps, routes and user's location
Show routes to the user
Change parking lot status from Available to occupied
End

Figure 2. Overall system flowchart

3.1.3. Raspberry-Pi Configuration

For the proposed methodology, Raspberry Pi played a big role in holding up the system's back-end processing, programming, and configurations, as it held several parts that were responsible for reading ultrasonic sensor data, establishing connectivity between itself and the ThingSpeak IoT cloud, and then sending it in the field to the cloud for easier monitoring. In addition, it held the server data, which was responsible for hosting the web architect and storing files and values in the database, including the floor plans of parking areas with their recorded RSS values, coordinate points, and points of interests (POIs), creating a viewable parking area infrastructure with its radio map. Figure 3 shows the flow chart of how the Raspberry Pi was programmed to read raw data from sensors and process it.

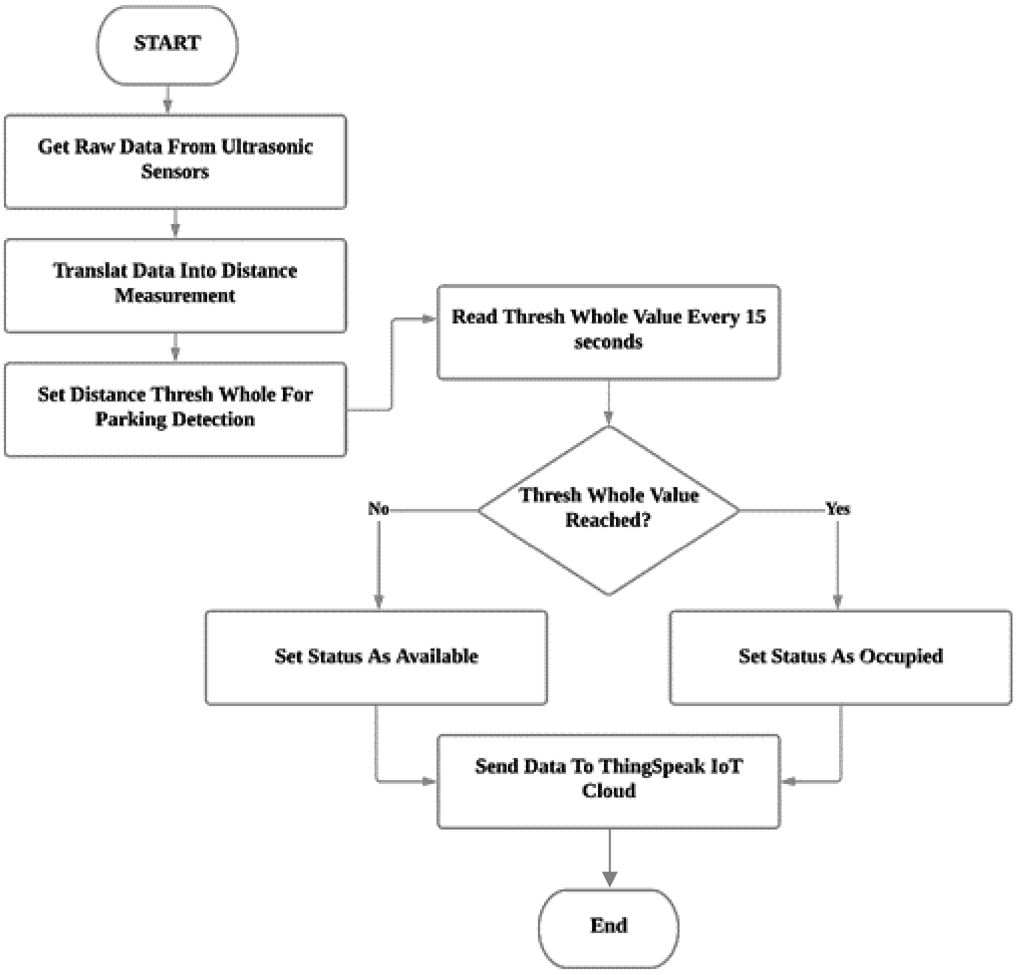

START
Get Raw Data From Ultrasonic Sensors
Translat Data Into Distance Measurement
Set Distance Thresh Whole For Parking Detection
Read Thresh Whole Value Every 15 seconds
Thresh Whole Value Reached?
No
Yes
Set Status As Available
Set Status As Occupied
Send Data To ThingSpeak IoT Cloud
End

Figure 3. Raspberry Pi program for parking detection flowchart

The Raspberry Pi was configured to get raw readings from the ultrasonic sensors and programmed using the Python programming language to calculate the distance between the vehicle and sensors to set a threshold. This threshold value would determine the presence or absence of a vehicle in a specified parking slot and was set based on the sensor location installation. Based on the set threshold, the Raspberry Pi could determine whether a parking slot was available or occupied and send this information to the ThingSpeak IoT cloud using the MQTT protocol. This process uses the Write API Key according to the channel ID to send it to a private channel that contains several fields, with each field representing a sensor status and reading. This helped in monitoring the sensor readings along with the parking area and slot counts. The fields could also present the readings in charts for further analysis, if needed.

3.1.4. Raspberry-Pi with ThingSpeak IoT Cloud

The parking system used a Raspberry Pi microcontroller to collect sensor data and send it to the ThingSpeak IoT cloud using the MQTT protocol. MQTT is commonly used in IoT applications to connect low-level devices and sensors since it is a publish/subscribe protocol, where the publisher sends a certain type of data and the subscriber receives the desired type of data. In this case, Raspberry Pi worked as a publisher to send the sensor data to the ThingSpeak cloud by passing short messages to and from the cloud broker, which contained the values taken by the ultrasonic sensors. After sending the messages, the cloud receives them and stores the content in a cloud channel, where the data could be visualised or analysed (Figure 4).

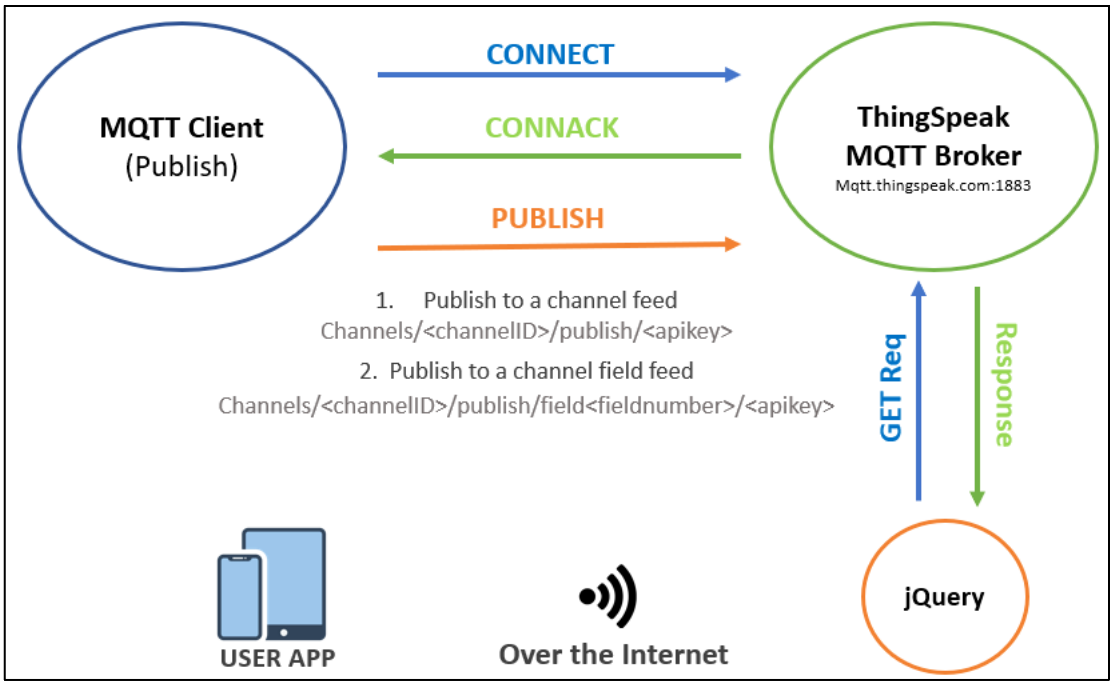

MQTT Client
(Publish)
CONNECT
CONNACK
PUBLISH
ThingSpeak
MQTT Broker
Mqtt.thingspeak.com:1883
1. Publish to a channel feed
Channels/<channelID>/publish/<apikey>
2. Publish to a channel field feed
Channels/<channelID>/publish/field<fieldnumber>/<apikey>
GET Req
Response
USER APP
Over the Internet
jQuery

Figure 4. ThingSpeak connectivity

More than one channel can be added to and used in ThingSpeak, and they can be accessed using the ThingSpeak API through the channels read link—one channel can contain more than one field, which holds sensor values. The API allows access to a certain field as well using the field read API link, where the sensor data is stored in JSON format and can be retrieved easily. A code was written to fetch the data from the ThingSpeak cloud in the Android app using a GET request to retrieve the channel JSON data and assign it to a JSON array as there was more than one entry in the cloud channel. The JSON array ran through a loop to fetch each field (sensor entry) and assign it to a JSON object, after which the data was formatted well and displayed in the app for the user. The app refreshed the page every two seconds to keep it updated with any value changes, while, as a free ThingSpeak plan was used, the cloud updated the JSON data every 15 seconds. Available parking slots, parking area levels, and the counts of available parking could be viewed by the drivers, who may also refer to the LED light indicators, to help find a parking slot in a shorter amount of time. These LED light indicators turn red if the parking space is occupied and green if it is available.

3.1.5. Raspberry-Pi as Server

To set up and deploy the Raspberry Pi as a server, some dependencies needed to be installed or updated, and some other files needed to be configured. The configuration was mostly done on its front end, as the held files were written in AngularJS and Google MapsJS. The server on the Raspberry Pi held a few applications, but the most important one of them was the web architect, which uses AngularJS and Grunt.

AngularJS is a framework used for dynamic web applications, allowing the use of HTML as a template language and extending its syntax as well for server-related tasks rather than making use of HTML as a declarative language only. It also helps with API and controller calls to perform server tasks. On the other hand, Grunt is software that is used to perform repetitive tasks; it can also be called a JavaScript task runner. It runs in the command-line interface (CLI) to run defined tasks and also allows the selection of different AngularJS versions to be used based on compatibility. It is written with NodeJS and distributed through the Node Package Manager (NPM), which therefore requires NPM installation.

The process of this setup started with updating and upgrading the Pi operating system dependencies to avoid any incompatibilities and errors during the deployment process. After the update was done, installing the NodeJS and NPM packages was required to help run JavaScript and AngularJS-related files. The server Bower dependencies were then installed, including jQuery and some other dependencies that would help in data fetching between the database files and the web architect, along with reducing the loading time required by the web architect. Grunt Task CLI helped with managing AngularJS versions and automatically chose the recommended version compatible with the installed dependencies for a smoother installation process. Furthermore, a server file was created and programmed, which contained some of the configurations related to the chosen port that the server would run on, along with others related to the SSL certificate and host machine. Lastly, the web architect page was modified and configured with new API keys, and the server was deployed. Figure 5 shows a block diagram for the process.

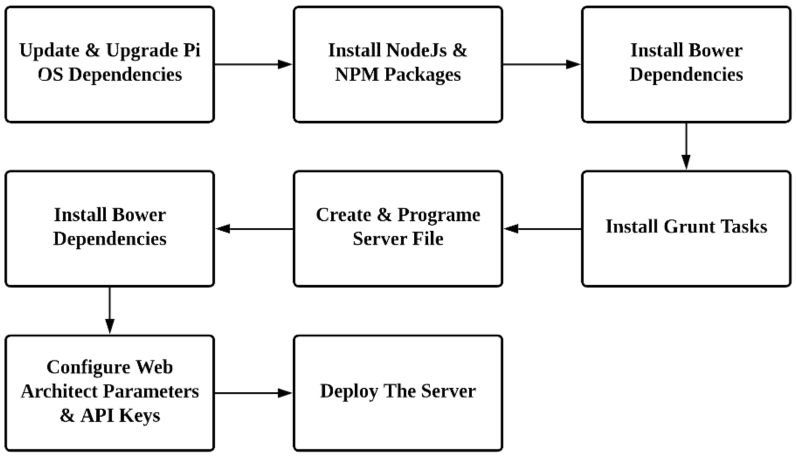

Update & Upgrade Pi OS Dependencies
Install NodeJs & NPM Packages
Install Bower Dependencies
Install Grunt Tasks
Create & Programe Server File
Install Bower Dependencies
Configure Web Architect Parameters & API Keys
Deploy The Server

Figure 5. Block Diagram for Raspberry Pi as a Server

3.2. Positioning System

The positioning system was based on the Anyplace v4 open-source service tool, which offers GPS-less localisation and navigation inside a building since spaces in general are becoming bigger and, in some other scenarios, more complex. Hence, there is a need for a more accurate localisation, navigation, and mapping system. Anyplace uses crowdsourcing and scalability mechanisms for handling different amounts of sensed data from different mobile devices. The abundance of sensed data from a smartphone, such as the Wi-Fi signal strength and inertial measurements, helps the localisation to be more accurate and reliable.

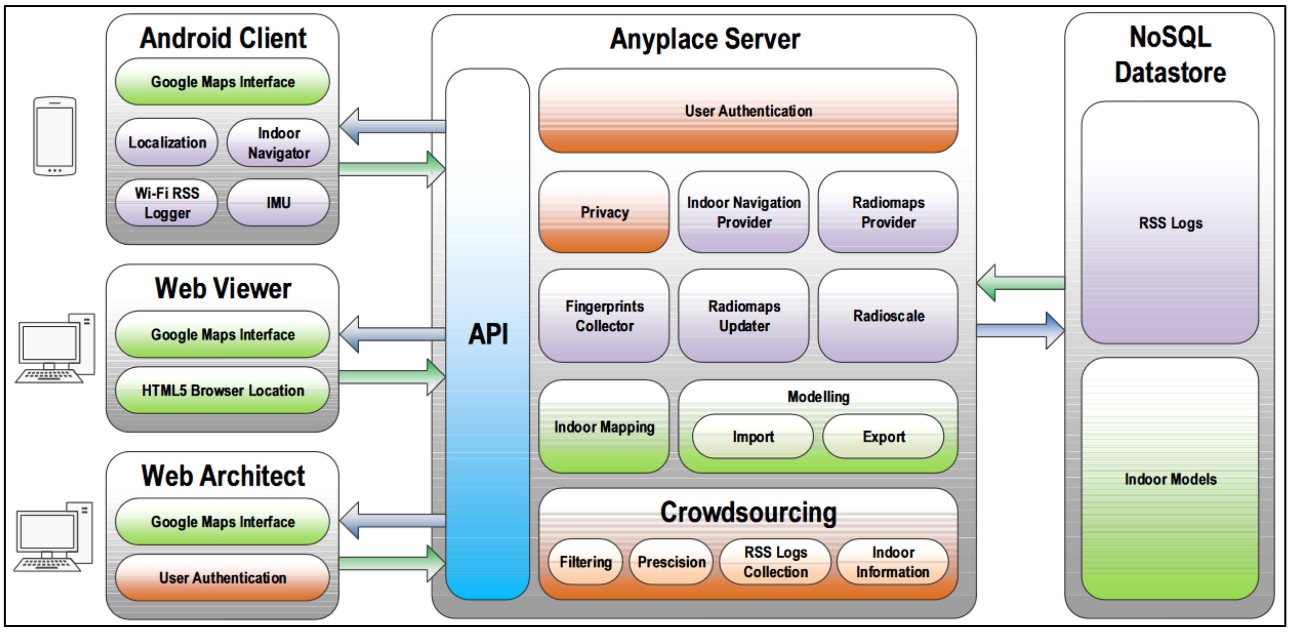


Figure 6. Anyplace Information Service Architecture [21]

Anyplace architecture consists of many parts: Server, Architect, Viewer, and NoSQL database. The floor plan was preferred to be designed using AutoCAD for better mapping. Then, POIs were added for every parking spot as well as every entrance. Lastly, after adding the floor plan, an application was used to collect Wi-Fi RSS radio maps through Logger, which was going to store the collected data in the database, and help to calculate the user location, even if they were not connected through Wi-Fi, by downloading a small part of the radio maps to the user device.

3.2.1. RSS Fingerprinting Recordings and WKNN

The weighted k nearest neighbour algorithm was used to estimate the user's location upon arrival at the parking area based on Wi-Fi RSSI fingerprinting. In order to assess the user's location, two sections were examined: the RSSI fingerprinting recording and the WKNN algorithm.

In order to measure positioning accuracy through Wi-Fi fingerprinting, two phases are usually taken into consideration: an online phase, also known as positioning, and an offline phase, also known as calibration. In the area where the Wi-Fi extenders were installed, the RSS values of these extenders at specific reference points were measured and recorded in dBm, with each value being accompanied by its coordinates (x,y). Afterwards, they were pinned onto the floor map of the area to create a radio map of the area. It was the first step of the calibration phase in which the averages of several recorded samples were stored in the database of the area, which was carried out during the first part of the calibration process. A WKNN algorithm has been used to estimate users' locations during the online phase by measuring RSS values at unknown locations and comparing them to those previously recorded in the radio map [22]. The phases of RSS fingerprinting are shown in Figure 7.

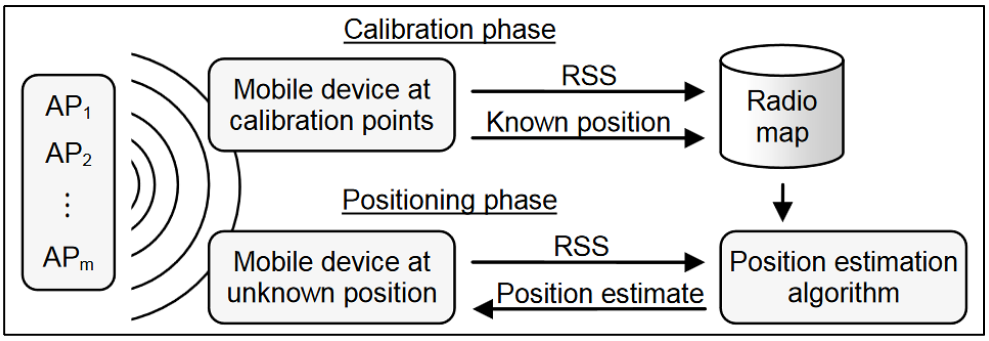

Calibration phase
AP1
AP2
⋮
APm
Mobile device at calibration points
RSS
Known position
Radio map
Positioning phase
Mobile device at unknown position
RSS
Position estimate
Position estimation algorithm

Figure 7. RSS Recording Process [22]

WKNN Algorithm: The weighted k nearest neighbours algorithm was used to estimate user location by finding the k indices from the radio map with the nearest RSS values using Euclidean distance to the reference points, given a vector measured at the unknown location. The Euclidean distance equation used to calculate the distance between the test point (TP) and reference point (RP) is shown in Equation 1, where the t value is two [23].

$$D_{iT} = \left(\frac{\sum_{j=1}^{m}\left(\left|RSS_T^j - RSS_i^j\right|\right)^t}{m}\right)^{\frac{1}{t}}, i = 1,2,3,\dots,n \qquad (1)$$

and where RSS_T^j is the RSS value at the TP from the j-th Wi-Fi extender, and RSS_i^j is the RSS value at the RP from the j-th Wi-Fi extender, while m is the total number of detected extenders at the TP and RP simultaneously. Each RP in the WKNN algorithm had a weighted average to help detect the nearest RF as a TP better, with the weights of the values of the selected points differing from each other, and inversely proportional to the calculated Euclidean distance, meaning the smaller the distance, the higher the weight. Weights for the RF were calculated using equation 2 [23].

$$w_{iT} = \frac{\frac{1}{D_{iT}}}{\sum_{i=1}^{K}\frac{1}{D_{iT}}} \qquad (2)$$

After each RF weight was calculated, the coordinates of the TF were computed using another equation based on the values obtained from the previous ones. Equation 2 was applied to determine the user location in the floor plan.

$$(x_T, y_T) = \sum_{i=1}^{K} w_{iT}(x_i, y_i) \qquad (3)$$

3.2.2. Experimental Setup

The system was tested in different stages. The first stage was an initial testing stage, where the parking system was tested first for parking detection activity. Then, the positioning system was tested separately to ensure both systems were working correctly before implementing them together in a parking location. The initial test for the parking detection system was done in two separate stages as well, the first being prototyping. A prototype for parking detection was made to ensure that the presence and absence of an object were recorded properly and that the information was sent to the ThingSpeak cloud correctly. The second stage was done by taking the prototyped programme to the real world and testing it in a total of five locations, including the three main locations that will be mentioned later on. The same process was done with the positioning system, which was tested for localisation first. Both systems showed fairly similar results to the final one after being set correctly. Finally, the two systems were combined to continue on to the final experimental test.

To test the accuracy of the system, it was tested in three different locations with different characteristics. For each location, ten points were taken and used as test points. The distance of navigation between the test points and the reference points (or the destination) was measured using several approaches, i.e., mobile GPS, WKNN with the building Wi-Fi, and maximum a posteriori (MAP). The characteristics of the three parking area locations are described in the following sections:

3.2.3. Parking Area for Location 1

Location 1 was a part of a medium-to-large on-campus parking area, almost 200 metres in length. This location was considered an outdoor facility, with weak to no Wi-Fi signal due to the unavailability of outdoor access points on the premises. In addition, the presence of trees might have decreased the signal strength (Figure 8).

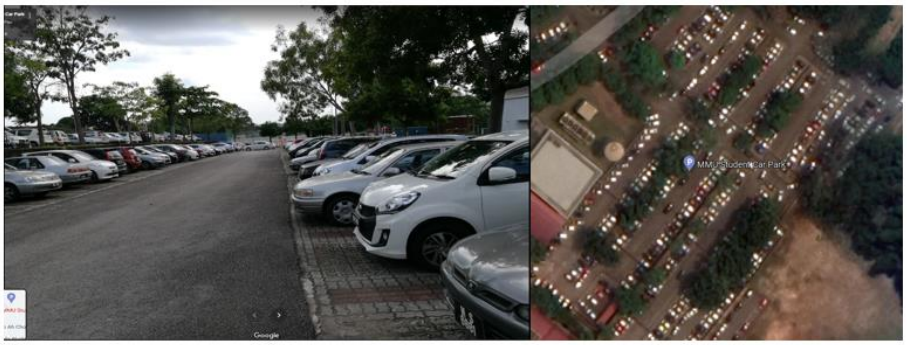


Figure 8. Pictures of Location 1 Parking Area

3.2.4. Parking Area for Location 2

The second location was also an on-campus parking area that was between indoors and outdoors as it was surrounded by buildings from all sides. It was a considerably smaller area compared to locations 1 and 3, being a little under 50 by 35 metres in space, but with a better Wi-Fi signal than location 1 (Figure 2).

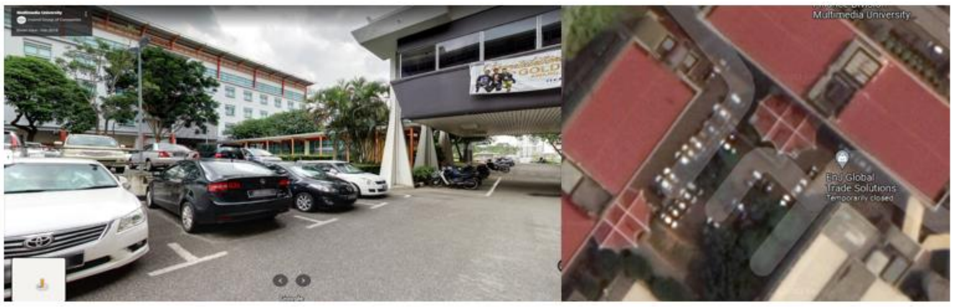


Figure 9. Pictures of Location 2 Parking Area

3.2.5. Parking Area for Location 3

Location 3 was a medium-square outdoor parking area with a space of 65 by 65 metres, with line-of-sight view, and minimal signal interference, but no pre-installed access points available. The location was good for testing how the approach would perform in vast open parking areas (Figure 3).

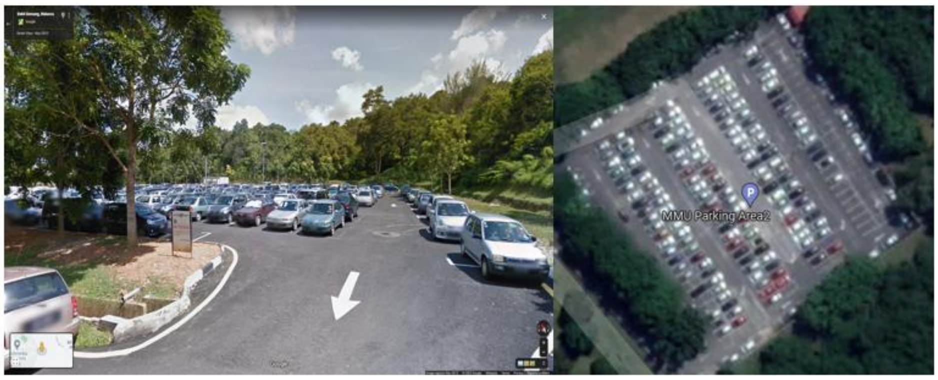
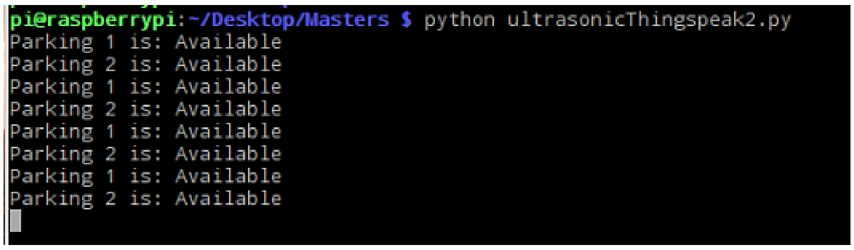


Figure 10. Pictures of Location 3 Parking Area

4. Result and Discussion

4.1. Parking Detection

The Raspberry Pi and HC-SR04 worked together, but by default, the analogue input from the HC-SR04 only showed the distance between itself and the object in millimetres. Therefore, a code was written to convert those readings into more usable information for parking detection. A lot of codes were available online for distance measurement in millimetres, so it was necessary to convert them to centimetres. The convergence was needed for testing to assign a threshold for the distance, as if it exceeded a certain value, an "Available" status had to be assigned for the parking lot. The ideal distance set for the "Available" status was approximately one metre from the sensor based on the tests done in real parking areas. Hence, if there was a space in front of the HC-SR04 sensor that was more than a metre, the parking slot would be marked as available, and if not, it would be considered occupied. The tested code view is shown in Figure 11.



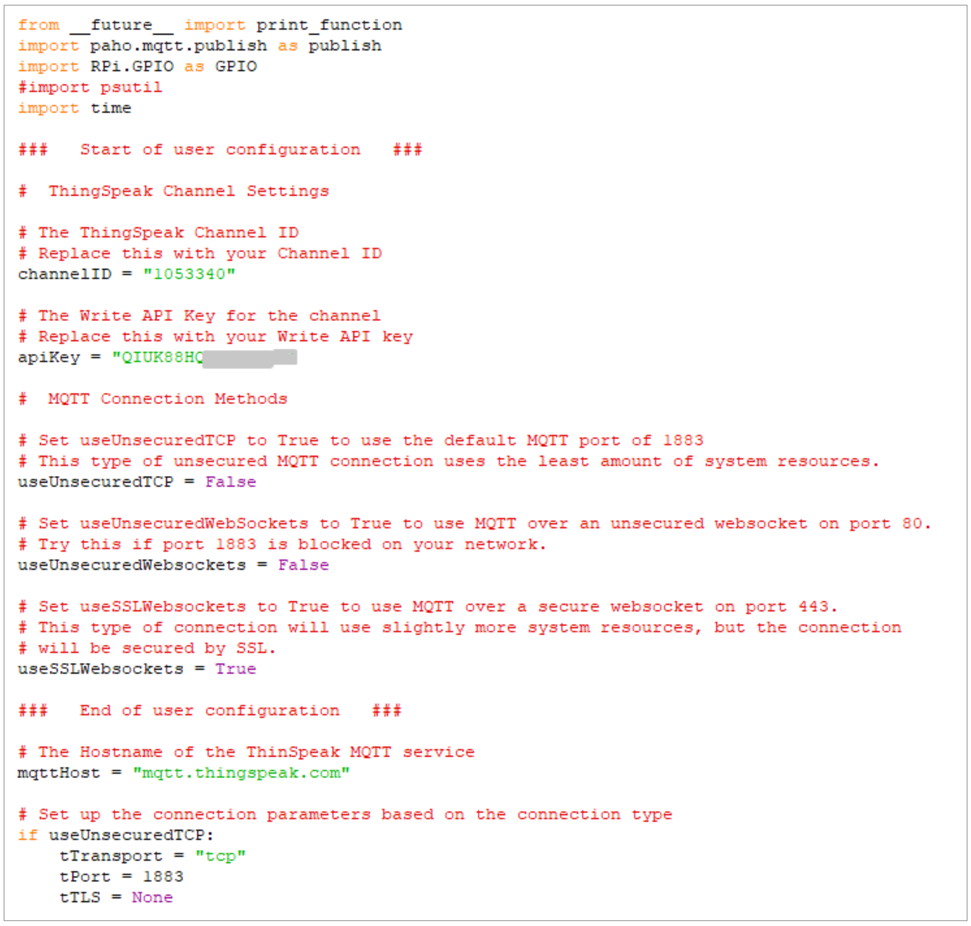
```
pi@raspberrypi:~/Desktop/Masters $ python ultrasonicThingspeak2.py
Parking 1 is: Available
Parking 2 is: Available
Parking 1 is: Available
Parking 2 is: Available
Parking 1 is: Available
Parking 2 is: Available
Parking 1 is: Available
Parking 2 is: Available
```

Figure 11. Tested Code for Parking Detection

After the assignment of the parking status, this information needed to be sent to the ThingSpeak cloud so that it could be retrieved later by the mobile app. For this part, a connection needed to be established between the Raspberry Pi and the cloud. To do so, several steps were involved:

- ThingSpeak account creation: An account was created to add a channel that could hold the sent data upon connection establishment. Each channel had its own unique ID for data reading.
- Obtaining an API key: After account creation, a unique API key was obtained. There were two types of API keys: read API keys and write API keys. For the purpose of connection establishment, an API key was used.
- Adding the ThingSpeak MQTT host: After getting all the previous information, the ThingSpeak MQTT host was added with an assigned port for connection. In this case, port 443 was used for an SSL WebSocket connection to guarantee a secure connection and that the data could not be sniffed. The default ThingSpeak MQTT host link was "mqtt.thingspeak.com".

How the API key, ThingSpeak MQTT host, and channel ID were added to the code is shown in Figure 12.



```
from __future__ import print_function
import paho.mqtt.publish as publish
import RPi.GPIO as GPIO
#import psutil
import time

###   Start of user configuration   ###

#  ThingSpeak Channel Settings

# The ThingSpeak Channel ID
# Replace this with your Channel ID
channelID = "1053340"

# The Write API Key for the channel
# Replace this with your Write API key
apiKey = "QIUK88HQ[redacted]

#  MQTT Connection Methods

# Set useUnsecuredTCP to True to use the default MQTT port of 1883
# This type of unsecured MQTT connection uses the least amount of system resources.
useUnsecuredTCP = False

# Set useUnsecuredWebSockets to True to use MQTT over an unsecured websocket on port 80.
# Try this if port 1883 is blocked on your network.
useUnsecuredWebsockets = False

# Set useSSLWebsockets to True to use MQTT over a secure websocket on port 443.
# This type of connection will use slightly more system resources, but the connection
# will be secured by SSL.
useSSLWebsockets = True

###   End of user configuration   ###

# The Hostname of the ThinSpeak MQTT service
mqttHost = "mqtt.thingspeak.com"

# Set up the connection parameters based on the connection type
if useUnsecuredTCP:
    tTransport = "tcp"
    tPort = 1883
    tTLS = None
```

Figure 12. ThingSpeak API Key & Channel ID Declaration

Finally, after completing the steps mentioned above, the information was wrapped inside a publish function with the provided port under the MQTT publish library, and connection was successfully established to the cloud, and information was passed to the channels. Two channels had been added to ThingSpeak—one, for monitoring inputs from the HC-SR04 sensors and the other for updating parking slot status. After testing and running the code, all the collected outputs from the sensors were sent to the cloud, as shown in Figure 13. The figure shows how the ThingSpeak cloud could view the sensor data in the form of graphs and indicators, as Field 1 and 2 charts show the distance output by the sensors from an object at a certain time and indicate it using light green and dim green figures.

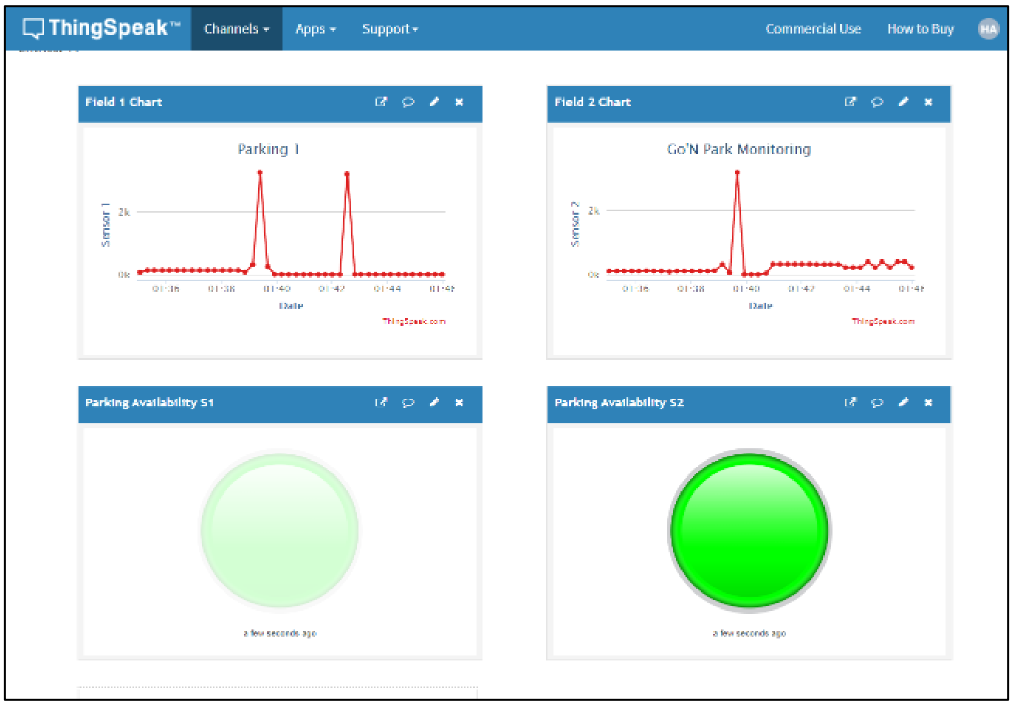


Figure 13. ThingSpeak Graphs Readings

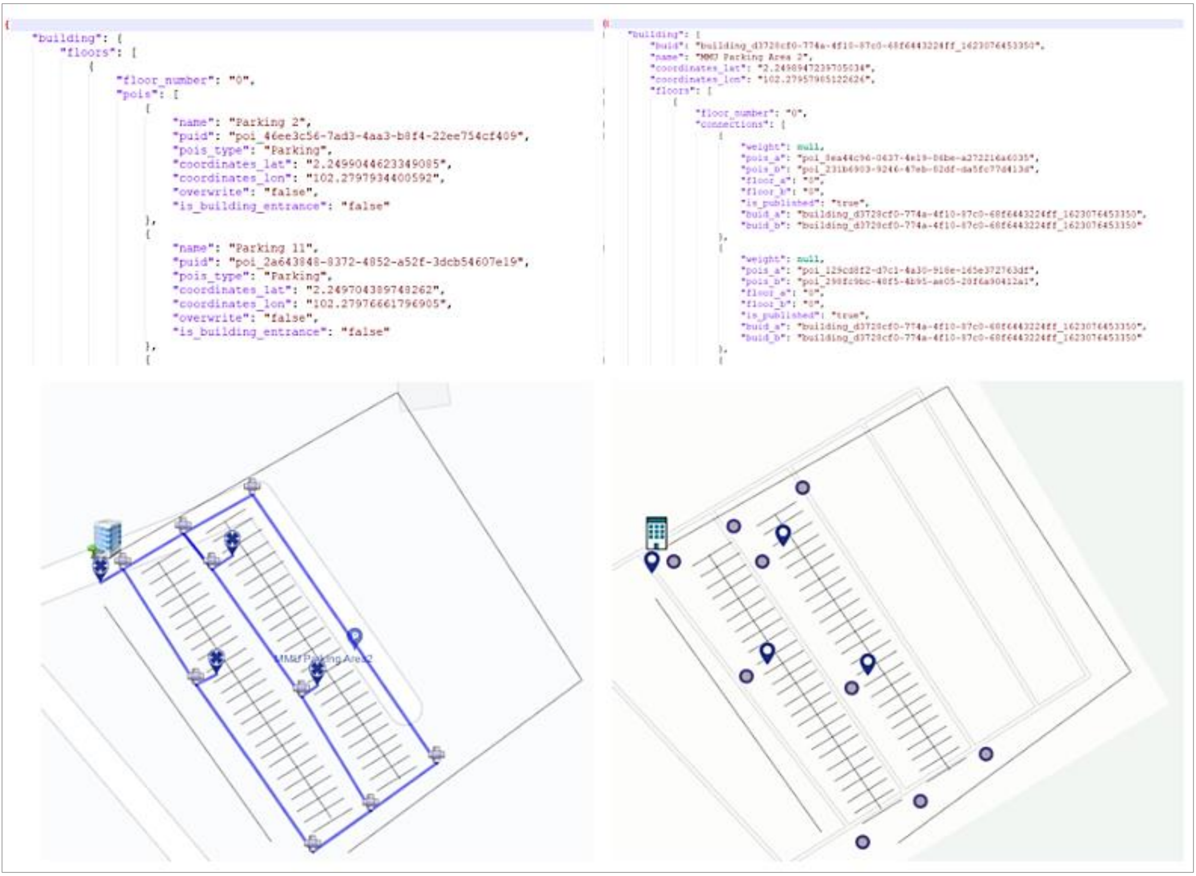
4.2. Web Architect

Web architect could now be used after all the needed libraries and dependencies had been installed and the server was deployed. As mentioned before, the web architect had a user-friendly interface that allowed for managing a building/area indoors or outdoors. However, since the web architect used the Google Maps API, a Google Maps JavaScript API token had to be created on the Google Cloud platform and accessed by the web architect to load Google Maps correctly.

To add a building or a parking area using the architect, there were several steps needed. The first was designing the building structure or the parking area plan using any third-party design software, such as Photoshop, AutoCAD, etc. After acquiring the floor plan blueprints, the area was added using the "Add Building" button, which allowed the uploading of the floor plan to the map API. Then, the entrance and points of interest (POIs) were added; in this case, the POIs were the parking slots, with every slot having its own decryption and ID to help with navigation later. Once all POIs were ready, the route to connect all these points together, including the building entrance, was created by adding POI connectors and routes. All the values and input were then published and submitted to the database. Figure 14 shows how the parking area looks on the map after adding the floor plans, POIs, and route connectors and shows how their values were stored in the database.

4.3. Wi-Fi RSS Recordings

After ensuring that the building was added and could be viewed through the app, the area's Wi-Fi coverage was examined using the Logger app before the installation of the extra Pix-Link Wi-Fi extenders. The Logger app was used to record Wi-Fi RSS. The app retrieved the building/area floor plan from the server, where the recording points started from the entrance. This was done in straight lines, while on every turn, a point was marked, which helped with access point range determination before every turn as multiple access points were used. After recording, these values were uploaded to the database to allow the web architect to create the area's radio map and visualise the recorded points, as shown in Figure 15.



```
{
    "building": {
        "floors": [
            {
                "floor_number": "0",
                "pois": [
                    {
                        "name": "Parking 2",
                        "puid": "poi_46ee3c56-7ad3-4aa3-b8f4-22ee754cf409",
                        "pois_type": "Parking",
                        "coordinates_lat": "2.2499044623349085",
                        "coordinates_lon": "102.2797934400592",
                        "overwrite": "false",
                        "is_building_entrance": "false"
                    },
                    {
                        "name": "Parking 11",
                        "puid": "poi_2a643848-8372-4852-a52f-3dcb54607e19",
                        "pois_type": "Parking",
                        "coordinates_lat": "2.249704389748262",
                        "coordinates_lon": "102.27976661796905",
                        "overwrite": "false",
                        "is_building_entrance": "false"
                    },
                    {
```

```
{
    "building": {
        "buid": "building_d3728cf0-774a-4f10-87c0-68f6443224ff_1623076453350",
        "name": "MMU Parking Area 2",
        "coordinates_lat": "2.2498947239705034",
        "coordinates_lon": "102.27957985122626",
        "floors": [
            {
                "floor_number": "0",
                "connections": [
                    {
                        "weight": null,
                        "pois_a": "poi_8ea44c96-0637-4e19-86be-a272216a6035",
                        "pois_b": "poi_231b6903-9246-47eb-82df-da5fc77d413d",
                        "floor_a": "0",
                        "floor_b": "0",
                        "is_published": "true",
                        "buid_a": "building_d3728cf0-774a-4f10-87c0-68f6443224ff_1623076453350",
                        "buid_b": "building_d3728cf0-774a-4f10-87c0-68f6443224ff_1623076453350"
                    },
                    {
                        "weight": null,
                        "pois_a": "poi_129cd8f2-d7c1-4a30-918e-165e372763df",
                        "pois_b": "poi_298fc9bc-48f5-4b95-ae05-20f6a90412a1",
                        "floor_a": "0",
                        "floor_b": "0",
                        "is_published": "true",
                        "buid_a": "building_d3728cf0-774a-4f10-87c0-68f6443224ff_1623076453350",
                        "buid_b": "building_d3728cf0-774a-4f10-87c0-68f6443224ff_1623076453350"
                    },
                    {
```

Figure 14. Parking Area POIs & Routes with their Values

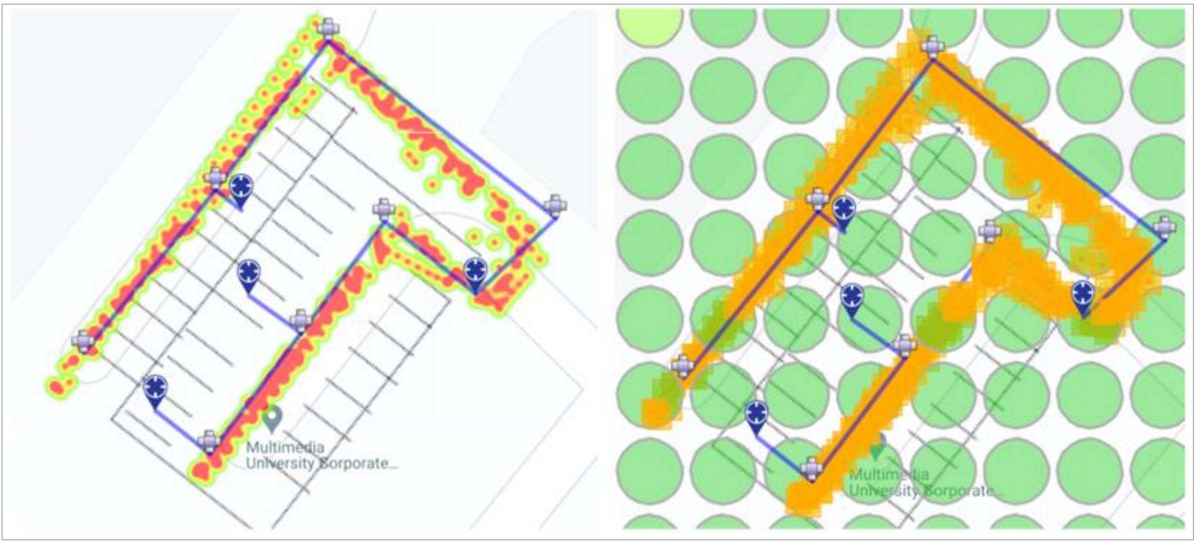
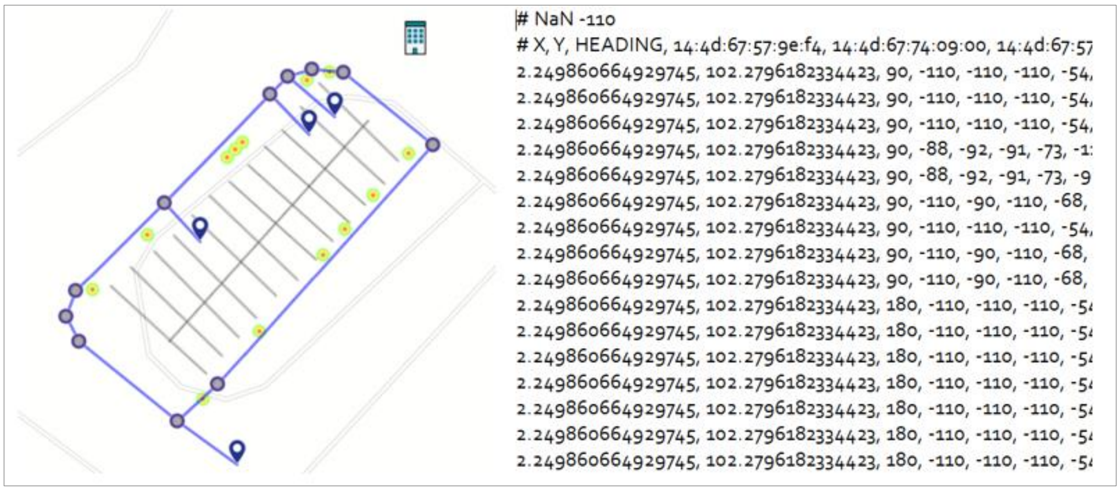

Multimedia University Corporate...
Multimedia University Corporate...

Figure 15. Visualised Recorded Fingerprints

During the recording process, a location with pre-installed Wi-Fi was examined to see how well the system was able to capture weak Wi-Fi signals. It is worth mentioning that the areas' Wi-Fi was installed for campus use only and not for localisation usage, which led to weak to no signal around the parking areas. The signal at the parking area was around -80 dBm to -92 dBm, depending on where the signal was captured. Recording fingerprints under a weak signal caused some points to be unrecorded, which led to some accuracy drops, such as shown in Figure 16. In such cases, recording the RSS fingerprints multiple times could have increased the accuracy, but the better solution would have been to improve the Wi-Fi coverage around the area. Finally, the ultrasonic sensors were connected to the Raspberry Pi 3 to check for vacant parking slots, establish the connection, and send information to the IoT cloud. The data regarding the parking slot's availability could then be viewed and analysed.



```
# NaN -110
# X, Y, HEADING, 14:4d:67:57:9e:f4, 14:4d:67:74:09:00, 14:4d:67:57
2.249860664929745, 102.2796182334423, 90, -110, -110, -110, -54,
2.249860664929745, 102.2796182334423, 90, -110, -110, -110, -54,
2.249860664929745, 102.2796182334423, 90, -110, -110, -110, -54,
2.249860664929745, 102.2796182334423, 90, -88, -92, -91, -73, -1
2.249860664929745, 102.2796182334423, 90, -88, -92, -91, -73, -9
2.249860664929745, 102.2796182334423, 90, -110, -90, -110, -68,
2.249860664929745, 102.2796182334423, 90, -110, -110, -110, -54,
2.249860664929745, 102.2796182334423, 90, -110, -90, -110, -68,
2.249860664929745, 102.2796182334423, 90, -110, -90, -110, -68,
2.249860664929745, 102.2796182334423, 180, -110, -110, -110, -5
2.249860664929745, 102.2796182334423, 180, -110, -110, -110, -5
2.249860664929745, 102.2796182334423, 180, -110, -110, -110, -5
2.249860664929745, 102.2796182334423, 180, -110, -110, -110, -5
2.249860664929745, 102.2796182334423, 180, -110, -110, -110, -5
2.249860664929745, 102.2796182334423, 180, -110, -110, -110, -5
2.249860664929745, 102.2796182334423, 180, -110, -110, -110, -5
```

Figure 16. Weak Signal Fingerprints and Fingerprints Values in the Database

4.4. Mobile App for Navigation and Parking

The Go'NPark Android app was developed based on the Anyplace user interface (UI) to test the discussed parking approach with the navigation and positioning algorithms. The app was developed to simulate how the system could all work together, providing the user with several interactive pages to demonstrate how navigation and positioning techniques could be implemented and possibly help the user find empty parking slots more easily. A few demo pages were developed to simulate the process that the user could face using such an approach. The first page was the welcome page that would appear once the app was launched by the user. After the app was launched, the first page showed three clickable buttons, which were used for testing. The first button, "Find Parking Areas," was responsible for retrieving the building/area information from the server added by the web architect, and it could also show if there were any added parking areas near the user. The "View Map" button showed the Google Maps API and current user location upon allowing the app to access the user's location. The map API also included user access to the added buildings/areas, and offered the choice of being navigated there or searching for other areas. The mentioned pages are shown in Figure 17.

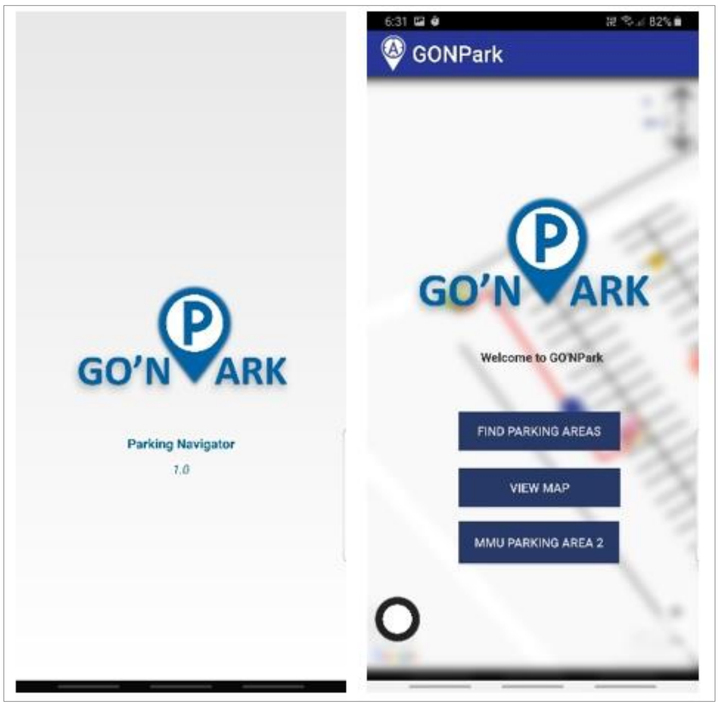


Figure 17. App Launch Page and Main Page

The last page, "MMU Parking Area 2", showed one of the locations parking slot availability, used for simulating and testing purposes, where the parking status would change when it was available or occupied. The "Park" button under every parking slot was meant to change the parking slot status from Available to Occupied for a short period of time, allowing the user to have enough time to get to that parking slot and reducing the conflict that may happen from multiple users heading to the same parking slot. Upon arrival at the parking slot, the sensor reading would read as Occupied, and if the user decided to change destinations, the status would change to Available. The page is shown in Figure 18.

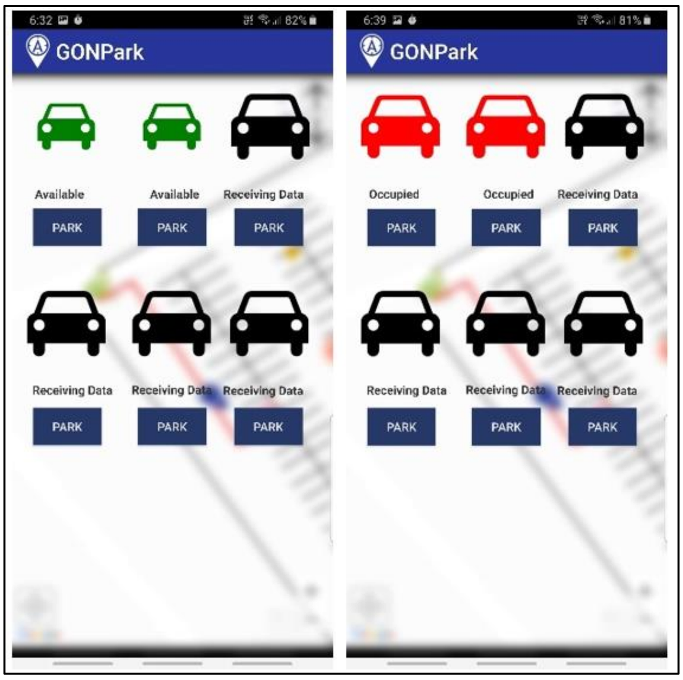


Figure 18. Parking availability detection page

4.5. Mobile App Experimental Results and Discussion

Pix-Link portable extenders were added in all areas for better Wi-Fi coverage. All three locations showed similar results for the tested scenarios. For the first approach, where GPS was used for parking navigation, the error rate indicated that the system was unreliable in all three locations compared to the other approaches. The average error rate was a little over 3.5 metres away from the average tested point in all areas. Using WKNN showed better results for parking navigation when applied with the pre-installed building Wi-Fi. However, since it was for campus usage, Wi-Fi was not installed in the parking area in location 1, which led to a weak to no Wi-Fi signal around the parking area. Nevertheless, it still showed a reduction in average error rates for locations 1 and 2 by about 35%, averaging at 2.3 metres. The second approach was not applied in location 3 since, as mentioned before, it had no pre-installed Wi-Fi.

After adding Wi-Fi extenders to improve the coverage around the parking areas, the navigation accuracy increased, while at the same time, the error rate decreased to 1.5 metres from the tested points in all three locations. Lastly, WKNN was replaced with MAP and tested in the same environment. An almost similar result was obtained as GPS navigation showed an error rate of 2.2 metres. Table 2 shows the error rate for each point at every tested location along with their averages using the three approaches, while Figures 20 to 23 visualise them in the form of graphs. Finally, Figure 24 shows the total averages of all the locations combined in a graph.

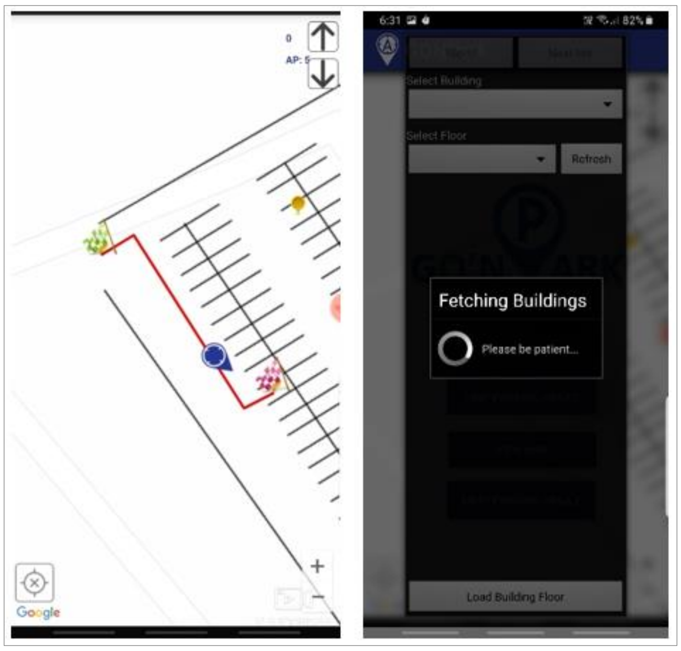


Figure 19. Parking Area Floor Plan & Routing Retrieval

Table 2. Locations Points Error Rates with Different Approaches

Tested locations	1	2	3	4	5	6	7	8	9	10	Average error rate.
Navigation method used: GPS											
	Point number & Error rate.										
Location 1.	2.0 m	3.0 m	5.5 m	3.5 m	5.0 m	3.5 m	4.0 m	2.0 m	2.5 m	2.0 m	3.3 m
Location 2.	3.0 m	4.5 m	7.0 m	4.5 m	6.5 m	4.0 m	7.5 m	2.5 m	4.5 m	3.5 m	4.75 m
Location 3.	2.5 m	2.5 m	4.5 m	4.0 m	6.5 m	3.0 m	3.0 m	1.5 m	3.0 m	2.0 m	3.25 m
Navigation Method Used: WKNN + Pre-installed Wi-Fi											
	Point No.										
Location 1.	2.5 m	2.0 m	1.0 m	2.5 m	2.5 m	3.5 m	1.5 m	1.5m	2.0 m	1.0 m	2.0 m
Location 2.	1.5 m	4.0 m	2.0 m	2.5 m	3.5 m	4.5 m	1.5 m	2.5m	3.0 m	2.0 m	2.7 m
Navigation Method Used: WKNN + Pix-Link Extenders											
	Point No.										
Location 1.	1.0 m	2.0 m	2.5 m	1.0 m	2.0 m	1.5 m	2.5 m	2.0m	2.0 m	1.5 m	1.8 m
Location 2.	1.5 m	1.0 m	1.5 m	0.5 m	2.5 m	1.5 m	2.0 m	2.5m	1.5 m	1.0 m	1.55 m
Location 3.	0.5 m	1.5 m	2.0 m	1.5 m	1.5 m	1.0 m	1.5 m	1.5m	1.0 m	0.5 m	1.25 m
Navigation Method Used: MAP + Pix-Link Extenders											
	Point No.										
Location 1.	1.5 m	3.0 m	3.0 m	3.5 m	4.0 m	2.0 m	2.0 m	2.5m	1.5 m	3.5 m	2.65 m
Location 2.	1.0 m	2.0 m	2.5 m	2.5 m	3.0 m	1.5 m	2.5 m	1.0m	2.5 m	2.5 m	2.1 m
Location 3.	2.0 m	2.5 m	3.5 m	1.5 m	2.0 m	1.0 m	1.5 m	0.5m	3.0 m	1.5 m	1.9 m

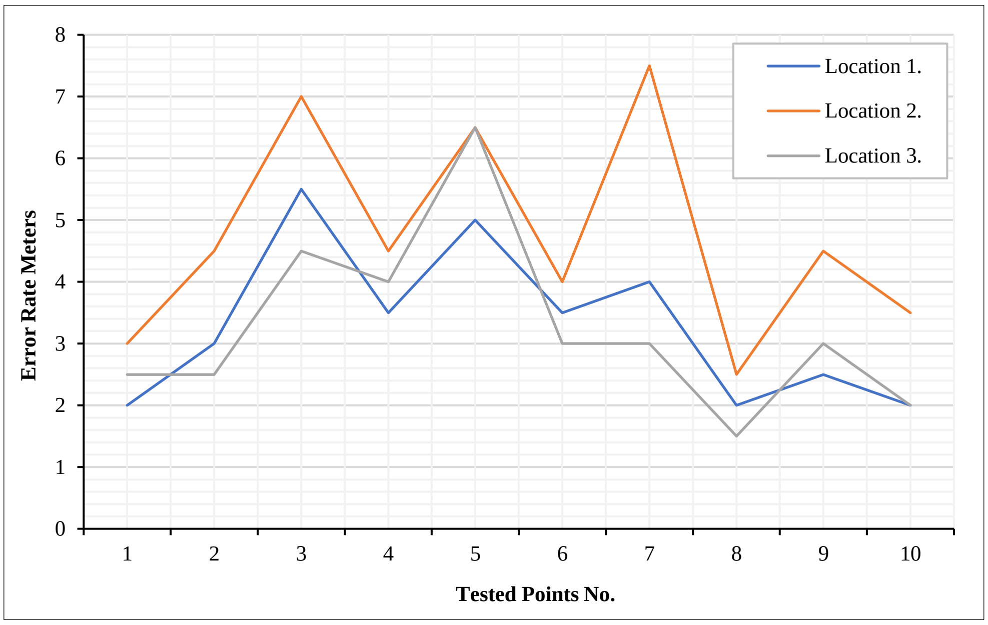


Figure 20. Using GPS for Navigation Error Rate Graph

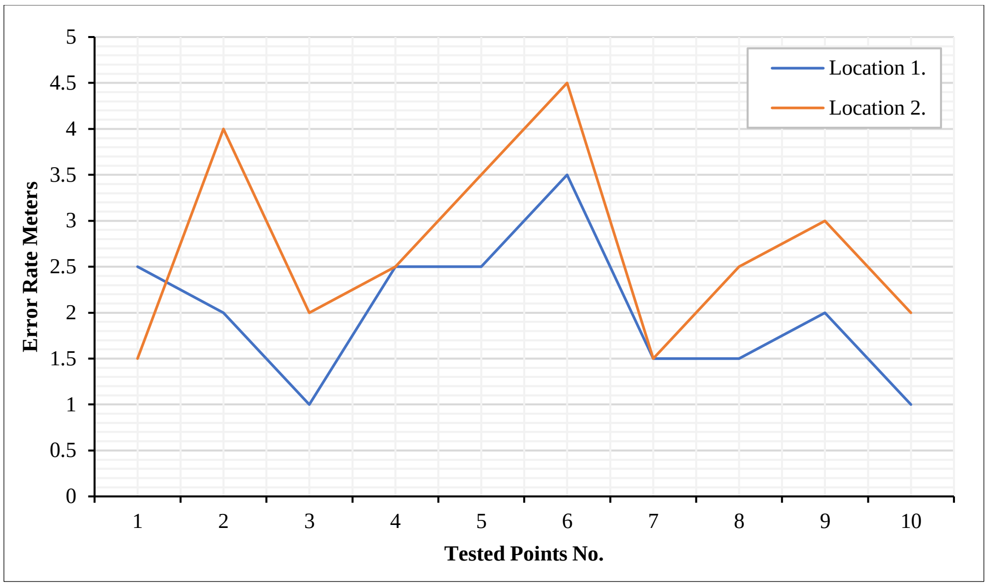


Figure 21. WKNN + Pre-Installed Wi-Fi Error Rate.

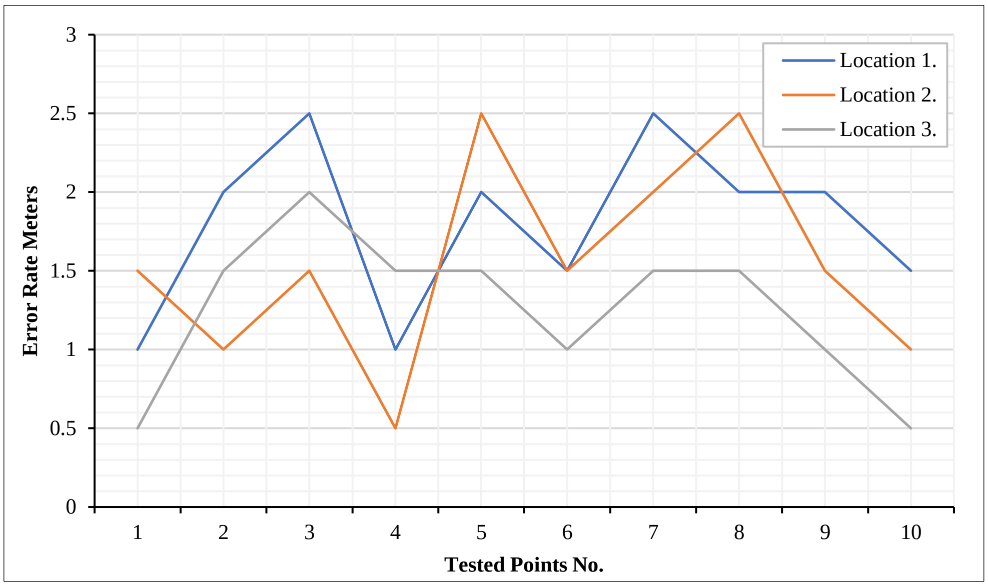


Figure 22. WKNN + Pix-Link Wi-Fi Extenders Error Rate

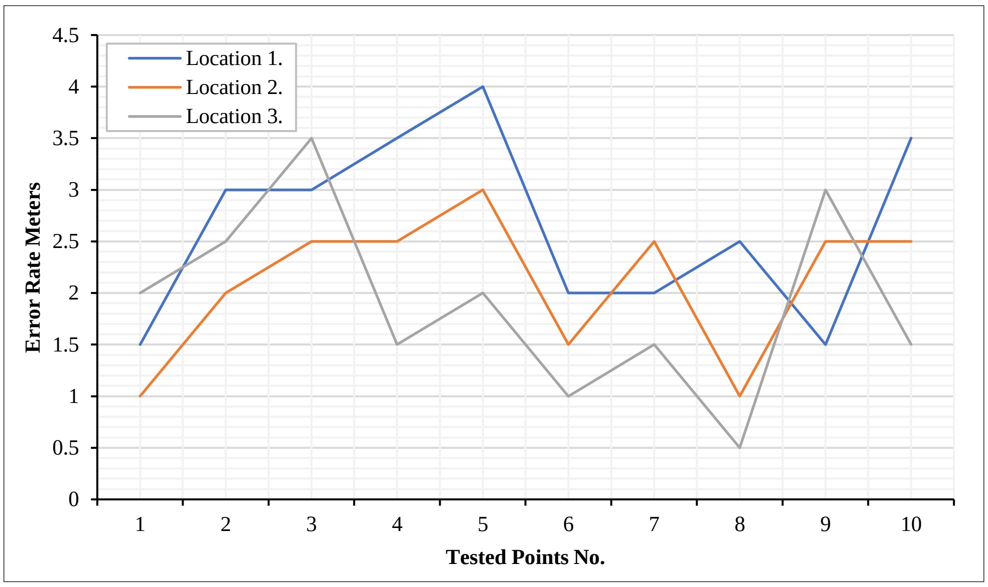


Figure 23. MAP + Pix-Link Wi-Fi Extenders Error Rate

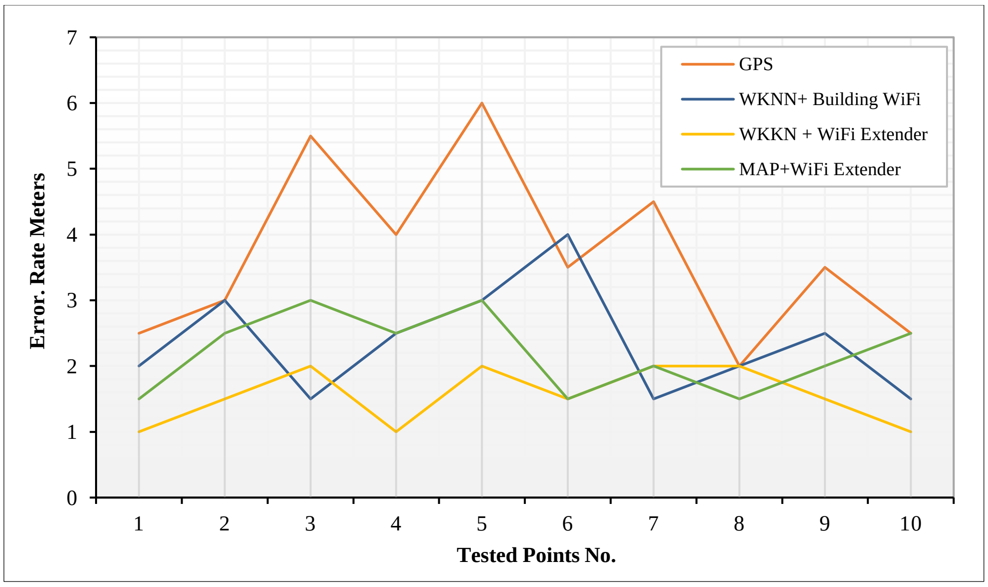


Figure 24. Overall Positioning Error Rate

From the graphs below, the different tested approaches are shown, and the difference in error rates is noticeable among them. Since the error rate refers to the distance between the physical reference and the Android application coordinates, an estimation of how far a point is from a parking slot could be done. The graphs present the error rates in metres, with the Y axis for each point on the X axis, while the tables summarise the output of each point's error rate and the overall averages on the ten tested points for the locations.

Figure 20 shows the navigation performance of the mobile GPS in all three locations. It showed low error rates at the first point because it was the starting point. Generally, the starting point is considered a stable point as there is no movement involved yet. Then, there was an observable rise in error rate as the user started moving. This pattern could be noticed in all the figures, i.e., Figures 20 to 24. Once there was movement, an aggressive rise in the error rates was introduced as the location changed, reaching a rate of more than 7 metres at the 7th point for the second location, which is considered a very large distance from the real coordinates, making it unreliable for parking guidance. The reason why the first and third locations showed lower error rates than the second was related to the fact that the GPS technology was most highly affected by the surrounding environment in the second location. The second location was between the indoor and outdoor areas, as it was surrounded by buildings from all sides, which reduced the signal received from satellites. Such a distance error rate is generally normal when using a technology like GPS because it is highly affected by satellite geometry, atmospheric conditions, and signal blockage from the surrounding environment.

Similarly, Figure 21 presents the error rate values, but for only two locations, which are the first and second locations, seeing as the proposed approach used the WKNN algorithm requiring pre-installed Wi-Fi, and the unavailability of one at the third location parking area made it impossible. Nevertheless, the same low error rate pattern could be seen at the first point for both the first and the second locations. When the user moved, the values began to rise, but overall, they showed lower high values compared to the GPS approach.

The graph spiked at the 6th point to around 3.5 and 4.5 metres for the first and second locations, respectively, compared to the highest spikes in figure 20. The reason for the value changes and error rate fluctuation in Figure 21 being slightly different from the ones in Figure 20 was not only because of user movement around the tested location but also because mobile devices switched connections between different access points. The closer the mobile phones are to an access point, the more accurate the localisation it provides, therefore resulting in a lower error rate. As a result, it is clear that the parking area access points were installed near the 3rd, 7th, and 10th points, as they showed the lowest values in the graph.

Furthermore, in Figure 22, even lower error rate values were observed after installing the Pix-Link portable Wi-Fi extenders. This helped with a more stable signal around the locations 1 and 2 parking areas, allowing better coverage and smoother connectivity switching between access points, as it was one of the issues faced when using the premises' pre-installed Wi-Fi. Therefore, even lower values are shown in Figure 22, especially between the 3rd and 4th points, as well as at the 7th and 10th points, due to them being the closest to these added access points. The problem with pre-installed Wi-Fi at the two locations was that, due to it being installed only for campus use, it provided low coverage in the parking areas, thus making the signal quite weak. This explains why the overall error rate for the first location was lower than for the second location, as shown in Figure 21.

A similar environment used to test the proposed approach with a similar algorithm that could be used under a similar environment and could make use of the same parameters was the maximum a posteriori algorithm. The resulting error rate values are shown in Figure 23, and they turned out to be higher than the proposed algorithm.

Finally, Figure 24 shows the overall average error rates for all three locations using the aforementioned tested scenarios by taking the error rates from each point in each location and then calculating their average for each of the tested scenarios. The error rates of the first point in locations 1, 2, and 3 were averaged to get the first point of the overall positioning error rate for the GPS graph. The same was done for the rest of the points, making the error rate line graph the average of using GPS technology for parking navigation at all three locations.

The overall average error rate of WKNN using Wi-Fi, on the other hand, was calculated for two locations in the same way that was mentioned previously, by taking all the points from location 1 and location 2 parking areas and averaging them. Location 3 was not considered in this scenario as it did not have pre-installed Wi-Fi. Likewise, for the rest of the line graphs, the averages were taken the same way, showing a low average error rate at the first point, then increasing when movement occurred, displaying the same patterns shown in Figures 21 to 23.

5. Conclusion, Limitations and Recommendations

The purpose of this research is to make vehicle parking easier for society as a whole. According to some studies, this is not a problem that affects only a few individuals but rather affects the majority of people. It can be a frustrating and time-consuming experience to locate parking spots in large parking slots or multi-level parking areas. This is especially true today, when everything is so fast-paced. It is therefore imperative to work every hour possible to keep up with this rapid growth so that advanced systems can take our society to the next level of comfort.

In this research, several parking guidance systems have been discussed with regard to their strengths and limitations, and most of them do not provide convenient real-time guidance. Furthermore, this research also aimed to help people and societies by making it easier for them to find parking openings in large parking slots, which was a major challenge. There is also an aim to develop technologies related to the Internet of Things (IoT) as well as smart cities.

In conclusion, this research achieved its aims by configuring and programming the Raspberry Pi to monitor and detect parking spots through the ThingSpeak IoT cloud. Additionally, it included the adoption of the WKNN algorithm for real-time accurate parking navigation through the developed Android application, with the intention of adopting and examining its use for car guidance in parking systems. In comparison with other parking navigation techniques, the implementation of this approach resulted in an up to 35% improvement in parking navigation accuracy and reliability. Therefore, this could contribute to reductions in parking time, improved quality of life, especially in urban areas, decreased fuel consumption from cruising around parking areas, and reduced CO_2 emissions by vehicles for a greener future.

On the basis of the four main categories of smart parking systems, the discussed system falls under the category of parking information and guidance systems since it relies on information technology for the parking system as well as navigation guidance to guide users to their parking slots. Further, it is often difficult to comprehend the divergence of parking system categories, as some categories may be in conflict with one another or share some technologies. With this being mentioned, in order to know what improvements can be made, an understanding of the other categories could help tremendously in such a process:

- Using e-parking technologies: Use technologies that are used for parking reservations in e-parking systems to make advanced parking reservations and payment systems for private parking areas, allowing the user to allocate a parking slot before even arriving and thus waste zero time searching for a parking slot by getting routed directly to the reserved slot.
- Using better hardware components: Better hardware can definitely improve the overall performance and accuracy of the system, especially when implemented in a vast, congested parking area where data transmission rate and processing time can be affected.
- Development of more creative positioning algorithms: The development of more algorithms for positioning can reduce the error rate and provide better results without the need for massive changes in the hardware, as such algorithms are effective in a loT applications. Furthermore, it is recommended that the usage of those algorithms be considered for different applications in smart cities and be provided with diverse graphical user interfaces (GUIs).
- Development of a more effective algorithm for parking detection: The developed algorithm for parking detection used a Raspberry Pi and ultrasonic sensors for users to view the parking status. However, the aspect of choosing a parking slot can be improved further by having more real-life scenario testing to provide a better parking experience.

Using different types of platforms and APIs. The development and use of different types of open-source, cross-platform APIs can enhance the user experience and performance of the parking search process. Thus, there is a need for more research to be conducted on different types of positioning cross platforms that could be used for parking guidance systems, as well as their potential applications.

Finally, collaborative efforts between organisations can enhance the parking experience in a significant way. In view of this, organisations are encouraged to join together in supporting research initiatives related to smart cities, smart parking, and parking guidance, which would help to advance the development of IoT as a whole.

6. Declarations

6.1. Author Contributions

Conceptualization, M.S.S. and H.A.; methodology, M.S.S.; software, H.A.; validation, H.A.; formal analysis, H.A.; investigation, U.A.B.; data curation, H.A.; writing—original draft preparation, H.A. and M.S.S.; writing—review and editing, S.F.A.R. and S.Y.; visualization, U.A.B.; supervision, M.S.S. All authors have read and agreed to the published version of the manuscript.

6.2. Data Availability Statement

The data presented in this study are available in article.

6.3. Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

6.4. Conflicts of Interest

The authors declare no conflict of interest.

7. References

[1] Kingatua, A. (2020). Bluetooth Indoor Positioning and Tracking Solutions. Medium, San Francisco, United States. Available online: https://medium.com/supplyframe-hardware/bluetooth-indoor-positioning-and-asset-tracking-solutions-8c78cae0a03 (accessed on April 2023).

[2] Krishnaveni, B. V., Reddy, K. S., & Reddy, P. R. (2020). Position Estimation of Ultra Wideband Indoor Wireless System. 2020 International Conference on Artificial Intelligence and Signal Processing (AISP). doi:10.1109/aisp48273.2020.9073234.

[3] Blazek, J., Jiranek, J., & Bajer, J. (2019). Indoor Passive Positioning Technique using Ultra Wide Band Modules. 2019 International Conference on Military Technologies (ICMT). doi:10.1109/miltechs.2019.8870099.

[4] Dabove, P., Di Pietra, V., Piras, M., Jabbar, A. A., & Kazim, S. A. (2018). Indoor positioning using Ultra-wide band (UWB) technologies: Positioning accuracies and sensors' performances. 2018 IEEE/ION Position, Location and Navigation Symposium (PLANS). doi:10.1109/PLANS.2018.8373379.

[5] Said, A. M., Kamal, A. E., & Afifi, H. (2021). An intelligent parking sharing system for green and smart cities based IoT. Computer Communications, 172, 10–18. doi:10.1016/j.comcom.2021.02.017.

[6] Sadhukhan, P. (2017). An IoT-based E-parking system for smart cities. 2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017, 2017-January, 1062–1066. doi:10.1109/ICACCI.2017.8125982.

[7] Shinde, S., Patil, A., Chavan, S., Deshmukh, S., & Ingleshwar, S. (2017). IoT based parking system using Google. 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC). doi:10.1109/i-smac.2017.8058256.

[8] Pham, T. N., Tsai, M. F., Nguyen, D. B., Dow, C. R., & Deng, D. J. (2015). A cloud-based smart-parking system based on Internet-of-Things technologies. IEEE Access, 3, 1581-1591. doi:10.1109/ACCESS.2015.2477299.

[9] Salma, Olanrewaju, R. F., & Arman, M. M. (2019). Smart parking guidance system using 360o camera and Haar-cascade classifier on IoT system. International Journal of Recent Technology and Engineering, Special Issue, 8(2S11), 864–872. doi:10.35940/ijrte.b1142.0982s1119.

[10] Thakre, M. P., Borse, P. S., Matale, N. P., & Sharma, P. (2021). IOT Based Smart Vehicle Parking System Using RFID. 2021 International Conference on Computer Communication and Informatics (ICCCI). doi:10.1109/iccci50826.2021.9402699.

[11] Kodali, R. K., Borra, K. Y., Sharan Sai, G. N., & Domma, H. J. (2019). An IoT Based Smart Parking System Using LoRa. Proceedings - 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2018, 151–154. doi:10.1109/CyberC.2018.00039.

[12] Zhao, J., Wu, Q., Chen, J., & Huang, Y. (2019). Parking, Intelligent Parking System. 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC). doi:10.1109/iaeac47372.2019.8997654.

[13] Anand, A., Kumar, A., Rao, A. N. M., Ankesh, A., & Raj, A. (2020). Smart Parking System (S-Park) – A Novel Application to Provide Real-Time Parking Solution. 2020 Third International Conference on Multimedia Processing, Communication &amp; Information Technology (MPCIT). doi:10.1109/mpcit51588.2020.9350429.

[14] Mackey, A., Spachos, P., & Plataniotis, K. N. (2020). Smart Parking System Based on Bluetooth Low Energy Beacons with Particle Filtering. IEEE Systems Journal, 14(3), 3371–3382. doi:10.1109/JSYST.2020.2968883.

[15] Aditya, A., Anwarul, S., Tanwar, R., & Koneru, S. K. V. (2023). An IoT assisted Intelligent Parking System (IPS) for Smart Cities. Procedia Computer Science, 218, 1045–1054. doi:10.1016/j.procs.2023.01.084.

[16] Mago, N., Mittal, M., Bhimavarapu, U., & Battineni, G. (2022). Optimized outdoor parking system for smart cities using advanced saliency detection method and hybrid features extraction model. Journal of Taibah University for Science, 16(1), 401–414. doi:10.1080/16583655.2022.2068325.

[17] Krishnan, R. S., Narayanan, K. L., Bharathi, S. T., Deepa, N., Murali, S. M., Kumar, M. A., & Prakash, C. R. T. S. (2022). Machine Learning Based Efficient and Secured Car Parking System. Recent Advances in Internet of Things and Machine Learning. Intelligent Systems Reference Library, 215, Springer, Cham, Switzerland. doi:10.1007/978-3-030-90119-6_11.

[18] Anwarul, S. (2022). An Efficient Minimum Spanning Tree-Based Color Image Segmentation Approach. Advanced Computing, IACC 2021, Communications in Computer and Information Science, 1528. Springer, Cham, Switzerland. doi:10.1007/978-3-030-95502-1_44.

[19] Ali, J., & Khan, M. F. (2023). A Trust-Based Secure Parking Allocation for IoT-Enabled Sustainable Smart Cities. Sustainability (Switzerland), 15(8), 1–18. doi:10.3390/su15086916.

[20] Rani, P., & Sharma, R. (2023). Intelligent transportation system for internet of vehicles based vehicular networks for smart cities. Computers and Electrical Engineering, 105, 108543. doi:10.1016/j.compeleceng.2022.108543.

[21] Georgiou, K., Constambeys, T., Laoudias, C., Petrou, L., Chatzimilioudis, G., & Zeinalipour-Yazti, D. (2015). Anyplace: A Crowdsourced Indoor Information Service. 2015 16th IEEE International Conference on Mobile Data Management. doi:10.1109/mdm.2015.80.

[22] Jekabsons, G., Kairish, V., & Zuravlyov, V. (2011). Analysis of Wi-Fi based indoor positioning accuracy. 6th International Conference on Electrical and Control Technologies, ECT 2011, 45–50.

[23] Peng, X., Chen, R., Yu, K., Ye, F., & Xue, W. (2020). An improved weighted k-nearest neighbor algorithm for indoor localization. Electronics (Switzerland), 9(12), 1–14. doi:10.3390/electronics9122117.