

Using IoT to create a parking bay sensor and provide real time updates via an app based on external factors

Author: Tarikh Chouhan

Supervisor: Dimitris Dracopolous

Date of Submission: 18/04/2018

Department Name: Faculty of Science and Technology

Keywords: Machine learning, Car Parks, Sensor, Real Time Embedded Systems, Mobile App

This report is submitted in partial fulfilment of the requirements for the BSc (Hons) Software Engineering Degree at the University of Westminster.

Abstract

This project revolves around finding a solution to an ongoing problem that many drivers face everyday, finding parking bays in close proximity without having to constantly drive and search for one. The solution outlined in this report uses an app, sensor (which was built using a NodeMCU microcontroller) and a server. Furthermore, this report outlines proposed solutions already offered as well as potential solutions. Upon implementation of the app, server and the sensor, the solution outlined was a success as the majority of the requirements was achieved hence making this project a success to an extent.

Acknowledgments

I would like to thank Dimitris Dracopolous for his quick replies to my emails and for keeping my confidence high at times when I doubted my own skills.

I would like to thank my family for supporting me during this intense time.

Table of Contents

List of figures	iii
List of tables	iv
1 - Introduction	1
2 - Literature Review	2
2.1 - Current and future problems of not finding parking bays efficiently	2
2.2 - Current methods proposed to tackle the problem statement	4
2.3 - Security	7
2.4 - Queuing Theory	9
2.5 Human reaction towards AI in cars	12
2.6 - Machine learning: The basics	14
2.6.1 - Machine learning data and analysis	16
2.6.2 - Creating machine learning data	19
2.6.3 - Analysing data through Matlab	21
2.6.4 - Conclusion of data	27
2.7 - Introduction to the KNN learning algorithm	28
2.8 - Introduction to the logistic regression algorithm	29
2.9 - Microcontrollers and Components	30
2.9.1 - Analogue readings	31
3 - Aim and Objectives:	32
4 - Requirements	34
5 - Design of software	40
5.1 - Sequence diagrams	40
5.2 - Action Diagrams	42
5.3 - Use Case Diagrams	43
6 - Methodology	44
6.1 - Source Control	44
6.2 - Agile ethics	46
6.3 - Tech Stack	48
7 – Implementation	50
7.1 - UI of app	50
7.2 - Data structure of the parking bay:	52
7.3 - Machine learning implementation:	54
7.3.1 KNN algorithm:	54

7.4 - Google Maps route finder:	58
7.5 - Saving and loading data:.....	61
7.6 - SSL.....	63
7.6.1 - Creation of keystore:	64
7.7 - Sensor.....	66
7.7.1 - Thermistor:	68
7.7.2 - Ultrasonic:	71
7.8 - Server.....	72
8 - Testing Strategies.....	75
8.1 - Unit Tests:	75
8.1.1 - Loosely coupling code.....	76
8.1.2 - Maven testing lifecycle	77
8.2 - Blackbox Testing:	79
9 - Evaluation	81
9.1 - Logistic Regression:.....	81
9.2 - Timing of algorithms.....	84
9.2.1 Code used to test speed of KNN algorithm:	86
9.2.2 - Code used to test speed of logistic regression:	87
9.3 - Sensor.....	88
9.4 - SSL.....	89
9.5 - Usability and impressions	92
9.6 - Future prospects.....	93
9.7 - Tackling the problem statement.....	95
References	97
Bibliography.....	101
Appendices.....	103

List of figures

- Figure 1.1 - Negative connotations of self-driving/AI cars. Page 13.
- Figure 1.2 - Positive connotations of self-driving/AI cars. Page 13.
- Figure 1.3 – Time profile of parked vehicles, for non-work purposes. Page 17.
- Figure 1.4 - Variation in start time profile of parking by day of week. Page 17.
- Figure 1.5 - Algorithms being tested on hourly escort data. Page 21.
- Figure 1.6 - Confusion matrix on logistic regression. Hourly escort data. Page 22.
- Figure 1.7 - Confusion matrix on KNN. Hourly escort data. Page 22.
- Figure 1.8 - Algorithms being tested on hourly shopping data. Page 23.
- Figure 1.9 - Confusion matrix on logistic regression. Hourly shopping data. Page 23.
- Figure 1.10 - Confusion matrix on KNN. Hourly shopping data. Page 24.
- Figure 1.11 - Confusion matrix on logistic regression. Based on hourly social data. Page 24.
- Figure 1.12 - Confusion matrix on KNN. Based on hourly social data. Page 25
- Figure 1.13 – Plot of test data. Blue indicates vacant and orange indicates occupied. Page 26
- Figure 1.14 - Confusion matrix based on logistic regression. Based on minute data. Page 26.
- Figure 1.15 - Confusion matrix based on KNN. Based on minute data. Page 26.
- Figure 1.16 - Breadboard Overview. Page 30.
- Figure 2 – UI of AppyParking app. Page 36.
- Figure 2.1 – UI of AppyParking app. Page 36.
- Figure 2.2 – UI of AppyParking app. Page 37.
- Figure 2.3 – UI of AppyParking app. Page 37.
- Figure 2.4 – UI of AppyParking app. Page 38.
- Figure 2.5 – UI of Google Maps. Page 39.
- Figure 2.6 – UI of Google Maps. Page 39.
- Figure 3 – Shows the interaction with user and app upon happy scenario. Page 40.
- Figure 3.1 – Shows the sensor interaction with the server and database. Page 40.
- Figure 3.2 – Shows the interaction of the user if server cannot connect to database. Page 41.
- Figure 3.3 – Shows the interaction of the user when the user wants to check for a parking spot in a certain area. Page 41.
- Figure 3.4 – Shows the activity diagram of the sensor. Page 42.
- Figure 3.5 – Shows the activity diagram of the user interacting with the app. Page 42.
- Figure 3.6 – Shows the uses cases of the backend of the system. Page 43.
- Figure 3.7 – Shows the use cases of the user requesting to a new location to look for a parking bay. Page 43.
- Figure 4.1 – Typical GitFlow chart. Page 44.
- Figure 4.2 – Gantt chart depicting the work flow of the project. Page 47.
- Figure 5 – UI of app showing message to user. Page 50.
- Figure 5.1 – UI of app. Page 50
- Figure 5.2 – Showing navigation to a parking bay to the user. Page 51.
- Figure 5.3 – Overview of the circuit for the sensor. Page 66.
- Figure 5.4 – Graph depicting the rate of error in a thermistor as the temperature increases. Page 69.
- Figure 5.5 – Screenshot depicting the process of running a clean build using Maven. Page 78

List of tables

Table 1 - 20 councils in England which generated the largest on- and off-street parking surpluses in 2016-17. Page 2.

Table 1.1 - Table regarding shopping hours. Page 19.

Table 1.2 - Table regarding social hours. Page 20.

Table 1.3 - Table regarding escort. Page 20

Table 6 – Table showing blackbox test results. Page 79.

Table 7 - Results for execution time for KNN algorithm. Page 84.

Table 7.1 - Results for execution time for Logistic Regression. Page 85.

1 - Introduction

This section of the report will cover the general scope of the project and what problem the project aims to tackle.

As car manufacturers continuously unveil new cars to the public every year and as the world's population continues to grow, a trend can be seen with these two factors. There are more cars appearing on the road every year due to growing population (BBC, 2016). This trend can have negative repercussions in terms of environmental and financial factors (e.g. more money being spent on petrol, insurance claims increasing, pollution and hazardous chemicals entering the atmosphere). However, there are positive effects to this trend, such as new technology emerging through innovative solutions and moving towards a society revolving around self-driving cars (Garret, 2017). Due to the trend outlined, finding parking bays to park in will be a challenge as more cars will be on the road. Hence, this project proposes a solution to combat this problem.

This project comprises of chapters and segments. Each chapter will individually contain highly detailed information in order to fully understand this report. This report will go through an overview of a mathematical concept that is seen every day in our lives and has only become popular in the last 100 years: queueing theory. Furthermore, this report will give an insight on traffic engineering and seeing what is in play at the current moment. In addition, feedback and information will be collated together from companies revolving around transport engineering. This report will also delve into an interesting and highly sophisticated part of computer science: machine learning. An overview of what is essentially machine learning and AI will be outlined, as well as, discussing the model chosen to use in my solution. Moving onto the technical aspect, this report will show the tech stack behind the proposed solution, as well as, explaining the choice for the chosen technologies. Diving deeper, it will show the source code behind the solution and explain concepts that might not be familiar with university students like dependency injection, the maven build life cycle and using GIT for source control.

2 - Literature Review

There have been multiple documents produced and published regarding the issue around car parks; whether it is the mathematics behind it or simulating the construction of a parking lot. Alongside this, there are documents providing in-depth articles from established bodies, such as the RAC foundation. These documents can vary from articles and publications to theses. In this section of the report, you will be updated and equipped with the latest works currently being undertaken in the parking community, as well as, getting a brief overview of the mathematics behind it.

2.1 - Current and future problems of not finding parking bays efficiently

As more cars will be on the road, available parking bays will be less frequent, which in turn creates frustration in drivers as they look for an available bay. As a result of their frustration, drivers tend to park illegally and end up having to pay a penalty/fine. Local councils are generating massive amounts of revenue by handing out parking fines. The following statistics paint a picture on how significant the car parking industry is:

<u>Local authority</u>	<u>2012-13</u> £,000s	<u>2013-14</u> £,000s	<u>2014-15</u> £,000s	<u>2015-16</u> £,000s	<u>2016-17</u> £,000s	<u>Ranking by</u> <u>2016-17</u> <u>surplus</u>
Westminster	39,705	51,037	46,426	55,875	73,191	1
Kensington & Chelsea	30,437	33,512	32,997	34,237	32,174	2
Camden	23,531	24,869	24,468	25,228	26,751	3
Hammersmith & Fulham	19,395	22,960	23,787	22,672	23,077	4
Brighton & Hove	16,254	18,090	18,642	20,075	21,213	5
Wandsworth	15,887	19,692	20,350	21,174	20,506	6
Islington	8,216	10,381	13,732	15,532	19,111	7
Haringey	5,213	5,700	16,145	14,917	14,635	8
Hackney	7,756	8,219	10,758	12,920	14,505	9
Hounslow	6,407	7,814	7,655	7,196	11,972	10
Lambeth	12,004	7,219	9,683	9,942	11,923	11
Milton Keynes	6,668	8,160	9,042	10,757	11,143	12
Birmingham	6,869	7,756	9,699	9,816	11,129	13
Brent	2,666	8,310	10,506	7,954	10,534	14

Merton	6,868	7,015	7,226	6,681	10,227	15
Cornwall	8,078	8,019	8,693	9,813	9,742	16
Bristol	4,222	7,495	6,053	7,696	9,537	17
Tower Hamlets	7,000	8,318	10,038	9,479	9,504	18
Newham	8,163	7,202	7,327	7,692	8,886	19
Barnet	813	7,879	346	6,703	8,643	20

Table 1 - 20 councils in England which generated the largest on- and off-street parking surpluses in 2016-17. (RAC Foundation, 2017).

Furthermore, unable to find available parking bays could have negative repercussions on a global scale, such as the increase of greenhouse gases since harmful emissions would be emitted from the car as it spends more time and fuel to look for a parking bay having the driver arrive at its-destination. In addition, driving around to look searching for a parking bay will use up fuel, hence the driver would need to spend money to fill their cars more frequently compared to finding a parking bay that is readily available.

Even more so, The British Parking Association has said that a solution to this has been very slow since councils have been strapped for cash. Therefore, implementing and planting a physical sensor onto the roads will take time but they are approaching a stage where smart parking will soon be available.

2.2 - Current methods proposed to tackle the problem statement

There are many theoretical solutions that have been stated in many journals and papers regarding the ongoing crisis of finding parking bays. The solutions proposed tackle the issue in different ways; some of which are better suited to other environments (i.e. indoors rather than outdoors), as well as, the structure of the car parks itself.

A proposed solution effectively deal with the problem visually in terms of object identification. The paper titled 'Vision-Based Automated Parking System' discusses how the current problem can be tackled by implementing an object identification algorithm using cameras. This approach has several advantages, such as the low cost to the business implementing this feature, as well as, a high success rate in determining whether the parking bay is occupied or vacant. Whilst this approach would be ideal due to minimal wiring needed to implement it, the disadvantage of this method would be to wire more cameras in the parking lot. This is because the camera in their project was only able to identify 4 parking bays. To overcome this problem, the paper states "...be able to pan and tilt to have a larger view of the car park area and thus use less cameras..." (Hamada, 2010). Although this solution is more dynamic and adaptable to change, it is not as accurate as sensors. Furthermore, this solution would only be ideal for indoor car parks as opposed to outdoors. Placing the solution outdoors would require permission from private land owners, councils, businesses and the likes, in order to put up the camera needed for this solution.

Secondly, another solution proposed was to use a sensor attached to a rail situated near the ceiling of the car park, and the sensor would travel along the rail whilst scanning the parking bays below (Kevin, 2012). This solution is another ideal solution as it does tackle the problem statement whilst keeping business cost in mind. This solution outlined in the paper 'Automated Parking Space Locator: RSM' went into great depth regarding the structure and materials used to deploy this idea. It took into consideration the weight of the materials and the dimensions, as well as, the feasibility of the cost. The negative criticism to this idea was once again the inability to implement it outdoors and falls into the same problem addressed above; one would have to seek council/land owners' permission to have a rail on their street. This would

not be practical as the material would then have to be fit for harsh weather conditions, as well as, constantly charging the sensor unit. Implementing it outdoors would cause the cost of the solution to rise, which defeats the purpose of its own principles of keeping costs low for themselves.

Furthermore, another solution outlined utilizes machine learning and sensors. This approach is different to the previous two mentioned above because those approaches have only either used machine learning or sensors – not both in conjunction. The paper titled “Automatic Parking Management System and Parking Fee Collection Based on Number Plate Recognition” aims to tackle two problems: finding out if there are parking bays available and calculating the parking fee for cars already parked (Muhammed 2012). Focusing on their main feature, which is finding if a parking bay has been taken up or not, the solution outlined revolves around simply checking if the ultrasonic sensor has detected an object up close. As this paper is mostly concentrated around the aspect of calculating the fee of a parked car, not much thought was given into the method of finding if a parking bay was occupied or not. Therefore, although the paper went into great detail on how it implements an optical character recognition (OCR) algorithm, as well as, documenting the challenges it faced (such as having their cameras recognise different cars and license plates), it did not go into detail in terms of finding out if a parking bay was taken. Furthermore, with the current implementation of their sensor to find out whether a parking bay is taken or not, it would not be practical nor viable to use in the public streets because more work needs to be done on the sensor.

Overall, there have been many attempts in combating this ongoing problem. Some of the solutions proposed are more suited in ideal situations (i.e. inside a parking lot) but my solution is generic and can be used in many places regardless of the environment (e.g. it can be used out on the streets, as well as, inside parking lots). There are some flaws to this approach – like the high cost of planting a sensor physically on to a parking bay and maintenance cost. However, the benefits outweigh the negatives as it will aim to keep the roads less cramped and easier to find a parking bay.

The other solutions outlined, more specifically the solutions utilising OCR, may have to store the CCTV image data onto their database that essentially means storing users' data (e.g. the model and make of the car and the license plate number). Storing this sort of data relates to many legal laws that must be abided too otherwise legal cases might ensue. There have been numerous times where a well-known company have taken users data without their permission such as the OnePlus scandal (Chris, 2017) and the antivirus app developed by DU (Tony, 2017). More recently, at time of writing this report, Cambridge Analytica accessed data of around 50 million Facebook users without their consent and this data was used by Donald Trump's campaign team to psychologically target relevant people (Brennan, 2018). This project will not implement any sort of personal data.

2.3 - Security

TLS/SSL (transport layer security/secure sockets layer) is one of the most important aspects when it comes to exchanging data over the internet. SSL is all about encrypting the data when communication is happening between a client and a server (Agathoklis,2017).

It is very important to implement SSL as it can prevent a lot of mishaps from occurring. The gravity of this concept is so immense that Google have decided on the v62 update of Chrome they would penalise websites that do not have SSL implemented on their site by displaying an insecure message to notify users to be wary and be cautions of the website (Paul, 2017). SSL encrypts data by using certificates, cryptography and digital signatures (Cisco, 2002).

Cryptography relates to encrypting the data. There are two methods of encrypting the data: symmetric encryption and asymmetric encryption. These both have their advantages and disadvantages. An example of symmetric encryption uses the same key to encrypt/decrypt a message. (Cisco, 2002).

Asymmetric encryption has the same concept but instead of one key being shared there are two: a public key and a private key. These keys are created in pairs; the private key is only meant to be for the server and cannot be shared under any circumstances and the public key can be distributed to clients (i.e. users of the website). The public key can only encrypt messages, whilst the private key can only decrypt messages (eTutorials, no date).

Both methods have their advantages and disadvantages. With the symmetric approach, encryption and decryption is faster compared to asymmetric encryption although symmetric encryption is not as secure as asymmetric encryption. This is because if someone got their hands on the key, they could essentially eavesdrop and decrypt the message (such as man in the middle attack) but this would not occur if an asymmetric encryption was used, because the private key gets sent to nobody and that is the most important factor.

Whilst SSL ensures integrity of the data, it is important to realise that relying on SSL alone is not the best method of ensuring a secure infrastructure for your server as you will need additional components, such as the likes of a proxy

server and firewall. An example of how one should not place all their trust in SSL is the moment when Google recognised certificates using the SHA-1 algorithm as flawed and stopped trusting those certificates (Andrew, 2016) as it was proven to be broken (ShatteredIO, no date) (Lily, 2017). Advances in this field has drastically changed and most certificates are now using the new generation of algorithms such as SHA-256 and SHA-3 which are much more complicated to break.

Furthermore, in asymmetric encryption, the keys are created using an RSA algorithm. This algorithm essentially gets two very huge prime numbers and the product of the primes becomes a composite number. The public key and the two prime numbers are involved in the making of the private key (Burt, no date). In order to 'crack' the public key and get the two prime numbers involved, one would have to try factorising the composite number and get the two primes involved. Because factorising is hard despite the advances in discovering faster factorising methods, it would take 1000s of years for a computer to figure out the prime numbers. Whilst this sounds bulletproof, there is a slight flaw. The flaw relates to making sure the numbers are prime numbers. As the number gets bigger, the confidence level of whether or not it is 100% a prime reduces as it would take more time in computing whether or not the number is in fact a prime. Because of this trade-off between time and confidence level, different RSA algorithms exist so that some are slightly less secure for faster key creation and some are slow but for more secure keys (eTutorials, no date).

2.4 - Queuing Theory

Queues are everywhere. They can be observed in the most obvious places such as customers lining up to pay for their goods in a shop, patients being on a waiting list to see the GP or drivers waiting their turn to fill up their car in a petrol station. Queues can also be found in places where the average person would not typically realise they would find a queue, such as instructions being executed on a CPU or sending and receiving packets of data to browse the internet. Queuing theory was first written by Danish mathematician, Agner Krarup Erlang, back in 1909. Erlang worked at a telephone exchange that consisted of using jack plugs and plugging them into a circuit to route phone calls. He wanted to know how many circuits was needed to provide a sufficient service to a local village, thus, began researching and then publishing his findings in a paper titled 'The Theory of Probabilities and Telephone Conversations' (Achak, 2014).

Queueing theory in its simplest form, deals with problems involved with queues or waiting. Most problems regarding this concept have two entities in common: 'queue' and 'activity'. 'Queue' is the current wait and 'activity' is the server. So a practical situation would be a queue that represented a queue of customers and activity would represent a staff member at the cash till. The staff member deals with the customers one by one effectively taking care of the queue. There are some characteristics to these entities that are present in every queuing problem. The activity would need to determine on what the queue discipline would be, such as FIFO (first in first out) and LIFO (last in first out). Another concept present in queuing problems is understanding what type of queue we are dealing with. Examples include: baulking, which is where customers decide not to join the queue if it is too long; reneging, which is where customers leave the queue if they have waited too long to be served; or jockeying, which is where customers switch between queues if it will help them get served quicker.

Furthermore, another important variable to consider when dealing with queuing theory is understanding the behaviour of the arrival process. This means understanding how customers would join the queue. For example, in fixed timed intervals or variable times and whether they would join in as a group or as a single entity.

Kendall's notation is used to describe the mathematical notation of queues in queueing theory. This takes the form of $A/B/m/K/n/D$ in which "A represents the distribution function of the interval times, B represents distribution function of the service times, m represents the number of servers, K represents the maximum capacity of the system, n represents the number of sources of customers and D represents the service discipline" (Sztrik, 2011, p14). Usually, if in a notation, D and E are omitted, it is assumed they are infinite. M is also used in the Kendall notation to denote a Poisson arrival distribution or an exponential service time distribution. Furthermore, lamda (λ) and mu (μ) are both commonly used to represents the average arrival rate and average service rate respectively. The simplest and basic queue in queueing theory is the M/M/1 queue. This represents a queue with Poisson arrival distribution, an exponential service time and a single server (Beasley, no date).

'Applicability of information technologies in parking area capacity optimization' written by Maršanić Robert and Pupavac Drago is a research paper how to efficiently design parking areas based on waiting-line models; also known as queueing theory. In their paper (Maršanić, 2010), they were trying to find an efficient model to use for their car park, "Delta" located in a city in Croatia. They compared their findings with different models of car parks, such as a car park with a single-channel queueing model and a multichannel queueing model. They found out that having a single-channel queueing model is not as efficient as having a multichannel queueing model. This is because the single-channel service deteriorates in peak hours as it cannot cater for all the vehicles arriving during peak hours.

Furthermore, the research paper from Shuguo Yang and Xiaoyan Yang titled 'The Application of the Queueing Theory in the Traffic Flow of Intersection' delves into the concept of using queueing theory to analyse traffic conditions on an intersection (Shuguo, 2014), which is similar to analysing car parks as it revolves around the same concept. Their paper uses first hand data as they acquire their data from the intersection. By collecting first hand data, the results generated from their paper is reliable only in the location that they acquired their results from. They use the data to find the average number of cars arriving to the intersection. Their paper goes on to give the reader a clear and concise

conclusion by comparing their results from different scenarios. For example, comparing the overall probability there will be zero cars left in the queue in an intersection with two, three and four lanes. It further enforces the fact that using queuing theory is a sound and practical approach when dealing with vehicles and roads as this model can give a huge insight to vehicles waiting on the road.

2.5 Human reaction towards AI in cars

Machine learning is on the rise when it is used in conjunction with vehicles. Increasingly, more vehicles are incorporating AI into them. An example of this is Tesla's autopilot feature. As technology progresses further, vehicle progresses up the autonomous levels. We are moving towards a Level 3 autonomous society. Level 3 autonomy revolves around the car actively scanning and monitoring the environment by using external sensors, such as the likes of LiDAR, infrared sensors and ultrasonic sensors. Quite recently, a vehicle that aims to provide complete level 4 autonomy was showcased in CES 2018 (Dave, 2018). This further instigates research to be done to make roads smarter due to vehicles becoming 'smarter'.

However, there is a drawback of vehicles climbing up the autonomy scale and it revolves around the public perception. As pointed out by M Konig and L Neumayr in their paper (Konig, 2016), the overall public perception of AI cars is very murky. In terms of the public perception, the majority of the wariness comes from the technical aspect of the artificial intelligence, such as someone hacking into the car or whether the car miscalculates and executes a wrong instruction. Because driving is a very serious topic, as it can lead to life threatening situations like car crashes, the general public would rather have matters in their own hands rather than a computer because they see more of a threat if it is not under their control. Although the general public has an on/off attitude towards AI cars, the majority of people who are positive towards self-driving cars are the younger generation. Along with their report, M Konig and L Neumayr conducted an online questionnaire in which 489 people over 33 different countries took part in. As it was online, the questionnaire produced slightly bias results because the older generation are not typically affiliated with the online sphere, therefore, out of the 489 people that took part 129 were 31-69 and 39 were 60 or older. This resulted in a total of 34.5% of respondents being aged 31 or over.

Despite the negative connotation revolving around AI cars, there are positive aspects to it. As a result, Konig and Neumayr's research found that there was a mean of 3.50 that thought positive about AI cars. Even more so, the discussion that took place with the respondents said they would greatly value the self-

driving cars if there was some sort of method to park without having to look for any parking bays.

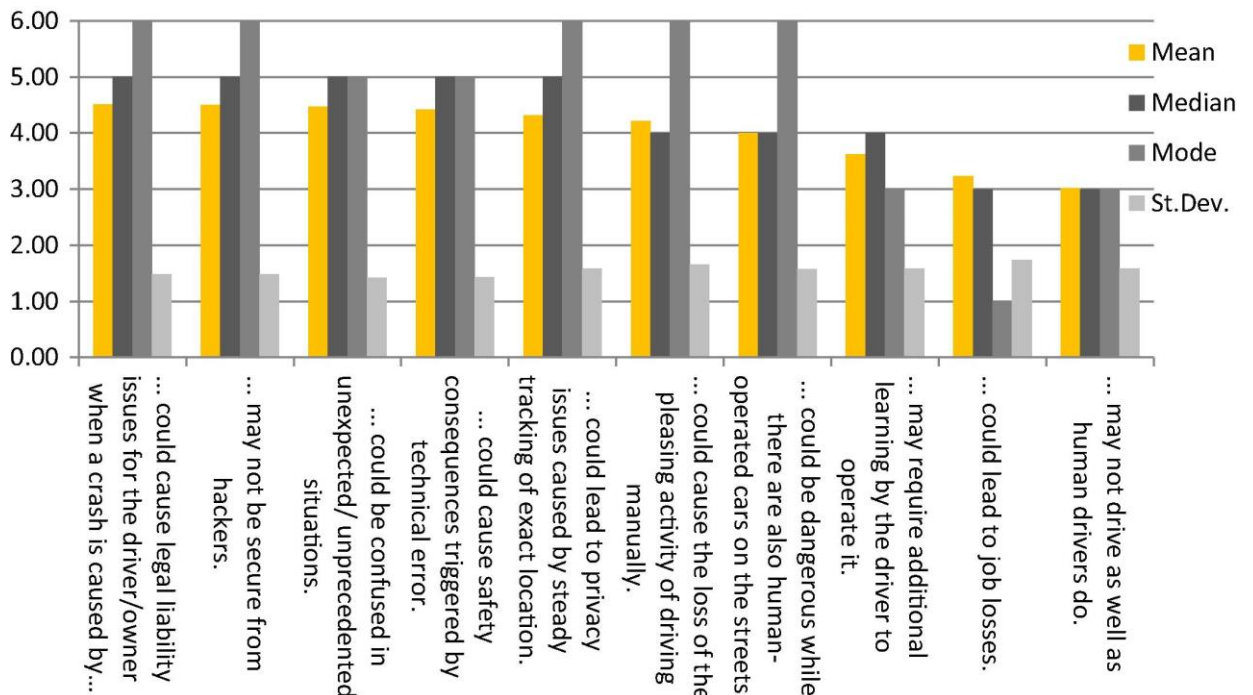


Figure 1.1 - Negative connotations of self-driving/AI cars (Konig M, Neumayr L. (2016))

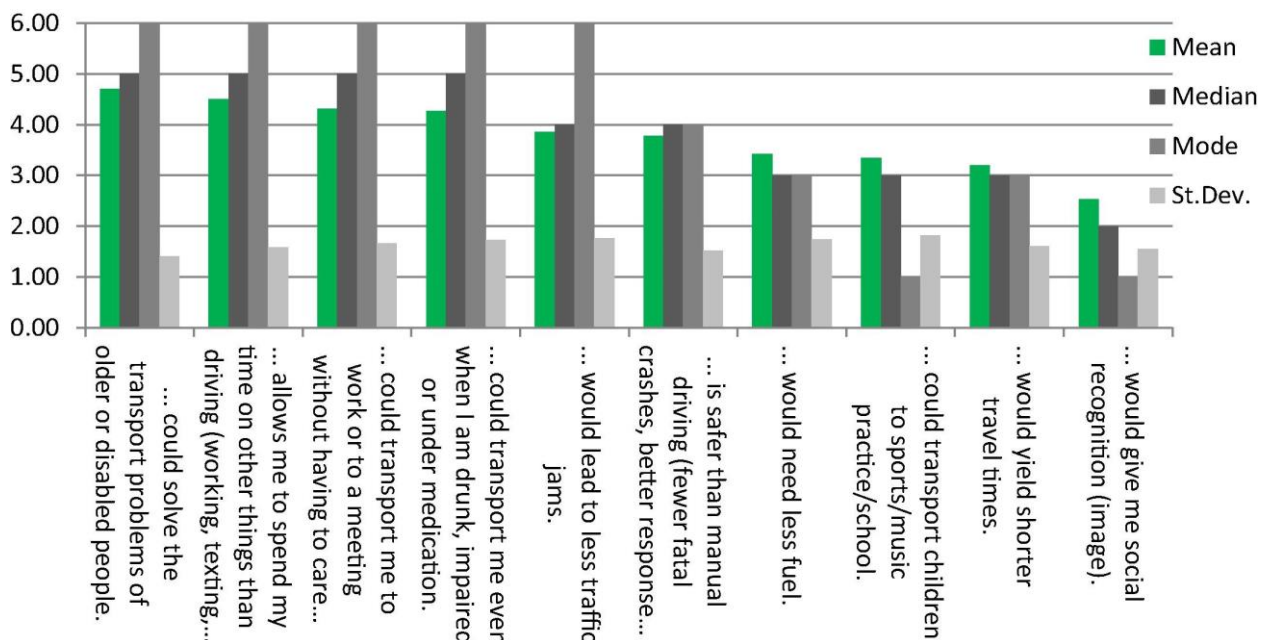


Figure 1.2 - Positive connotations towards self-driving/AI cars (Konig M, Neumayr L. (2016))

2.6 - Machine learning: The basics

There are many algorithms to use when it comes to incorporating machine learning. An important factor in choosing what algorithm to use depends on the data you are dealing with. There are two main types of machine learning algorithms: classification learning and regression learning. Regression learning revolves around continuous data and is usually used in scenarios where a value is to be predicted such as 'What will the average house price be in 10 years' time'. Classification learning on the other hand is more about predicting something that has a binary output, for example yes/no and 0/1. A typical question that would use the classification learning is: 'Will it rain today?' Here, there are only two possible outcomes to this question – either a yes or no.

However, classification and regression learning algorithms are used if the scenario revolves around 'supervised learning'. Supervised learning is used when the algorithm is given the inputs and outputs and the algorithm is trained to come up with the best prediction. As Bostjan Kaluza wrote in his book (Bostjan, 2016, Chapter 1, Section 6): "...given a set D of learning examples described with features, X , the goal of supervised learning is to find a function that predicts a target variable, Y . The function f that describes the relation between features X and class Y is called a model: $f(X) \rightarrow Y$ ". Essentially, we are supervising the machine to come up with a model.

On the other hand, unsupervised learning revolves around the computer identifying complex patterns with minimal human guidance. A prime example of unsupervised learning can be seen when products are being suggested to you on e-commerce websites, or relevant ads being displayed to you as you surf the web. Unsupervised learning is more complex than supervised learning as you are not explicitly giving the algorithm as much information. Thus, it is dependent upon the machine to come up with relevant labels to the data. The machine may do this in a variety of different ways but the common approach is to group the data based on some properties of the data and see which data is plotted next to each other. As these data gets plotted, there will be clusters forming. Depending on how dense the cluster is compared to another cluster and how far a cluster is from another cluster, it will associate a label to these clusters. This approach is used in the 'k-means clustering' algorithm, which is

used in unsupervised learning. Essentially, we are not supervising the machine to come up with a model but, rather, the machine comes up with a model.

2.6.1 - Machine learning data and analysis

Referring to the context of parking bay sensors, each sensor will theoretically have its own data of when it is being occupied or vacant. This is because each sensor will be used differently depending on where it is located. For example, if a sensor was placed near a school the general times it would be occupied ranges between 3pm to 5pm and if a sensor was placed near a business car park, the times it would be in use vary greatly like 9am to 6pm. Not only will time make a difference to the data, but the day of the week as well. An example of this is if a parking bay was near a shopping mall, it will be used more heavily on the weekends rather than the weekdays due to more people roaming on weekends. This is why it is incumbent to consider these external social factors.

In this report, data will be created for one sensor as generating data for more than one will be time consuming. There are a few approaches to creating this sort of data. One way is predicting how an individual sensor will be used and creating the data based on it (e.g. if this sensor will be placed in a business car park, expect to see the sensor to show up a 'occupied' between the hours of 9am-6pm). Another way of gathering the data to use would be to physically place the sensor on a parking bay. This approach would give the ideal data but has its drawbacks. For example, the sensor is not built to withstand any weather conditions or intense weight and should a car accidentally drive over it, it can cause damage. A better approach would be to look at a report regarding car parks and base the data on the findings of that report, such as the RAC foundation has done.

The RAC Foundation has done intense research and surveys to gather information revolving around car parks. The report has an immense amount of information ranging from supply and demand of car parks to management of parking. It also contains timings on how car parks are being used and for what reason. Below are graphs corresponding to parking bays from their report.

Time profile of parked vehicles, for non-work purposes

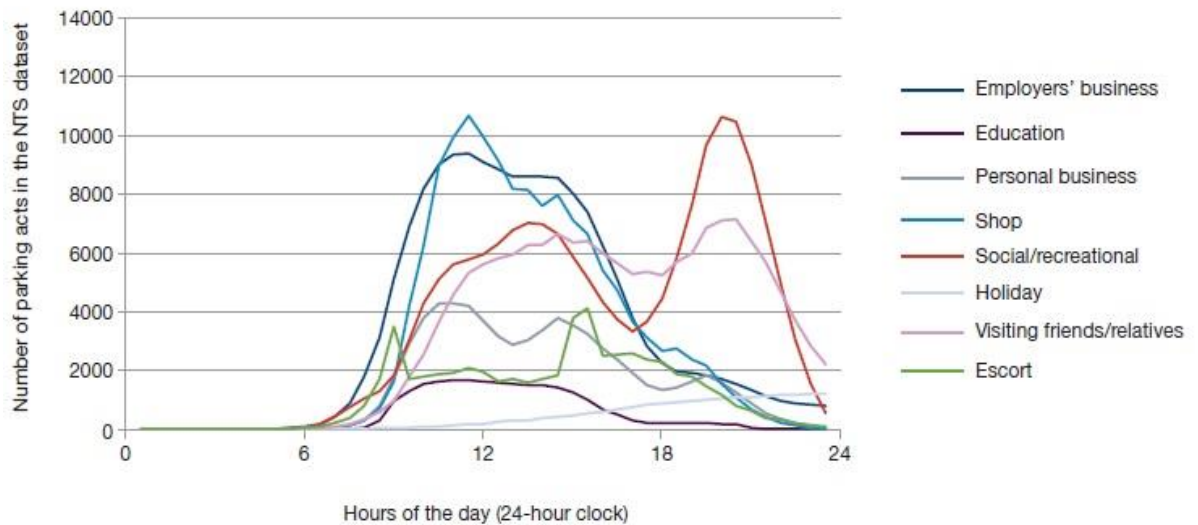


Figure 1.3 – Time profile of parked vehicles, for non-work purposes (Bates J and Leibling D, 2012)

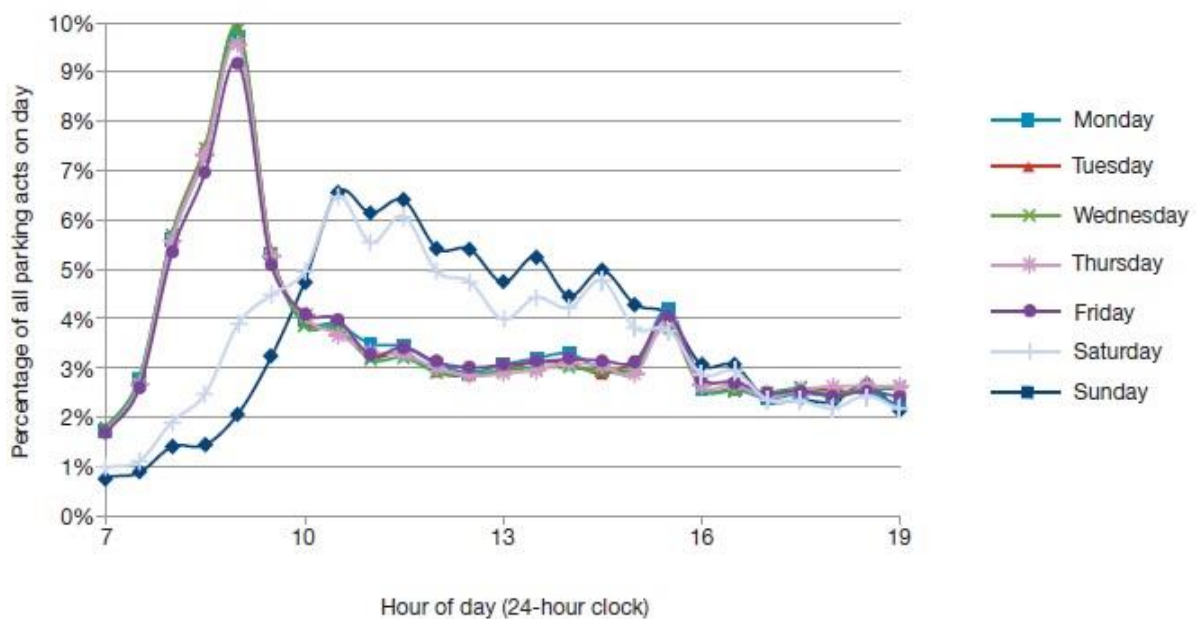


Figure 1.4 - Variation in start time profile of parking by day of week (Bates J and Leibling D (2012)

From the graphs (figures 1.3 and 1.4), we can see a multitude of information that will be relevant to the data to be created to feed to the machine learning algorithm. From the first graph (figure 1.3), we can deduce that some categories follow the same trend as other categories. You can see this by comparing the trend between 'shop' and 'employers business'. Furthermore,

the majority of the trends may have a different pattern to each other but they all usually follow the same downwards trend. This is because after 18:00 hours, 'Visiting friends/relatives, 'Escort', 'Shop' and 'Employers business' begins to spiral down. The social category tends to pick up after 18:00 hours as the report outlines that "...public car parks are especially used by shoppers and those travelling for social and recreational activities..." Therefore, a lot of people leave work around the 18:00 hours mark and go out and socialise.

Furthermore, as seen from the second graph (figure 1.4), all five weekdays follow almost an exact trend as each other whilst the weekends follow the same trend as each other.

2.6.2 - Creating machine learning data

Based on the graphs from RAC foundation, the following tables can be deduced to feed the algorithms:

Time (hours)	Status
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	0
15	0
16	0
17	0
18	1
19	0
20	0
21	0
22	0
23	0
24	0

Table 1.1 - Table regarding shopping hours

1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	1
11	1
12	1
13	1
14	1
15	0
16	0
17	0
18	0

19	1
20	1
21	1
22	1
23	0
24	0

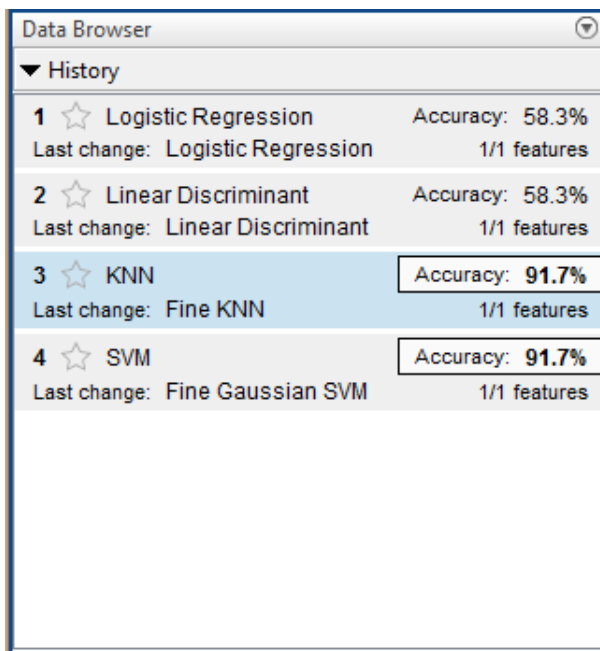
Table 1.2 - Table regarding social hours

1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	0
19	0
20	0
21	0
22	0
23	0
24	0

Table 1.3 - Table regarding escort

2.6.3 - Analysing data through Matlab

Once we feed these data into Matlab, we can use the 'Classification Learner' app and look at the different algorithms that perform on this data. Furthermore, we can critically analyse the performance of the learning algorithm and decide what algorithm should be implemented in the app to predict the availability of parking bays. There is no right algorithm because every algorithm has its strength and weaknesses ranging from different factors from computability and complexity.



Data Browser		
▼ History		
1 ☆	Logistic Regression	Accuracy: 58.3%
Last change: Logistic Regression		1/1 features
2 ☆	Linear Discriminant	Accuracy: 58.3%
Last change: Linear Discriminant		1/1 features
3 ☆	KNN	Accuracy: 91.7%
Last change: Fine KNN		1/1 features
4 ☆	SVM	Accuracy: 91.7%
Last change: Fine Gaussian SVM		1/1 features

Figure 1.5 - Algorithms being tested on hourly escort data

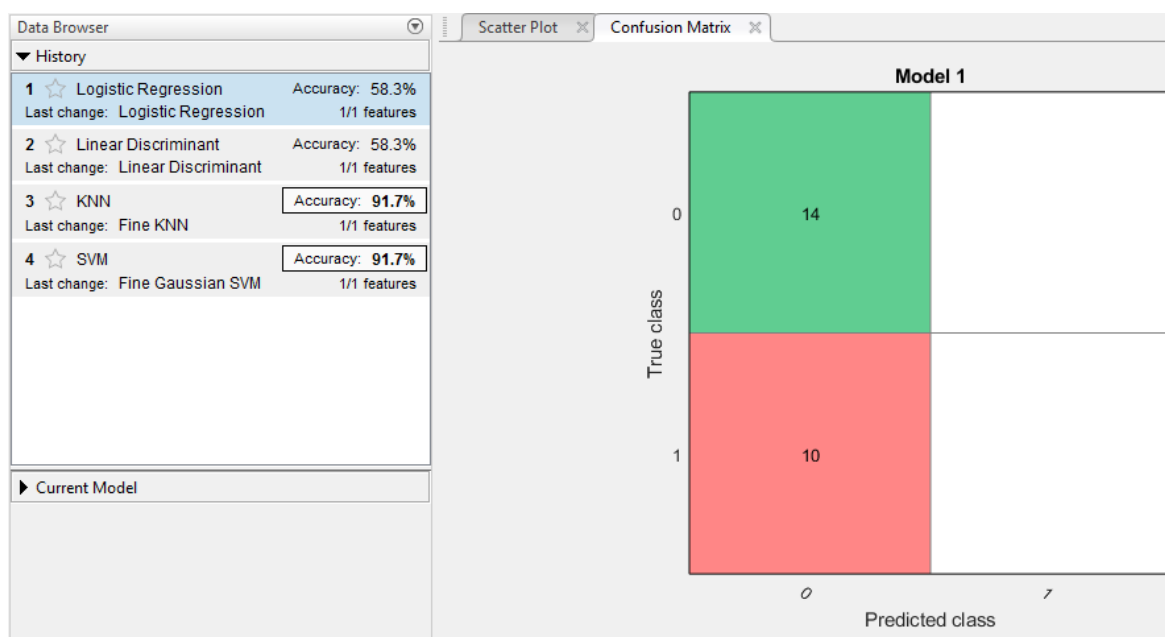


Figure 1.6 - Confusion matrix on logistic regression. Hourly escort data

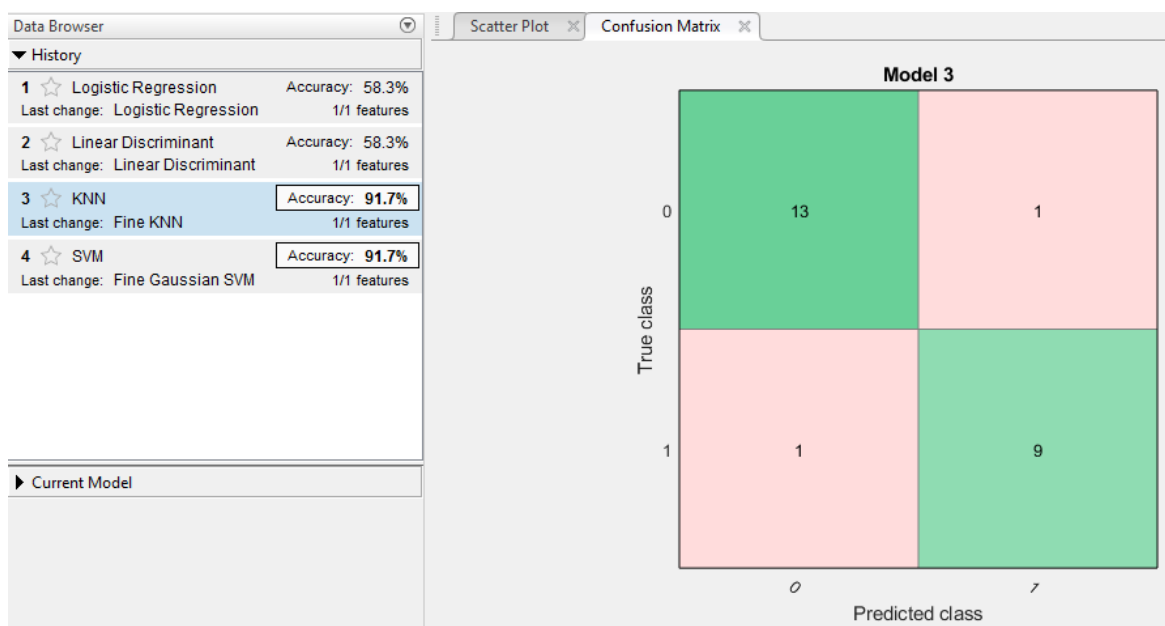


Figure 1.7 - Confusion matrix on KNN. Hourly escort data.

Data Browser		
▼ History		
1	☆ Logistic Regression	Accuracy: 62.5%
	Last change: Logistic Regression	1/1 features
2	☆ Linear Discriminant	Accuracy: 62.5%
	Last change: Linear Discriminant	1/1 features
3	☆ KNN	Accuracy: 83.3%
	Last change: Fine KNN	1/1 features
4	☆ SVM	Accuracy: 87.5%
	Last change: Fine Gaussian SVM	1/1 features

Figure 1.8 - Algorithms being tested on hourly shopping data.

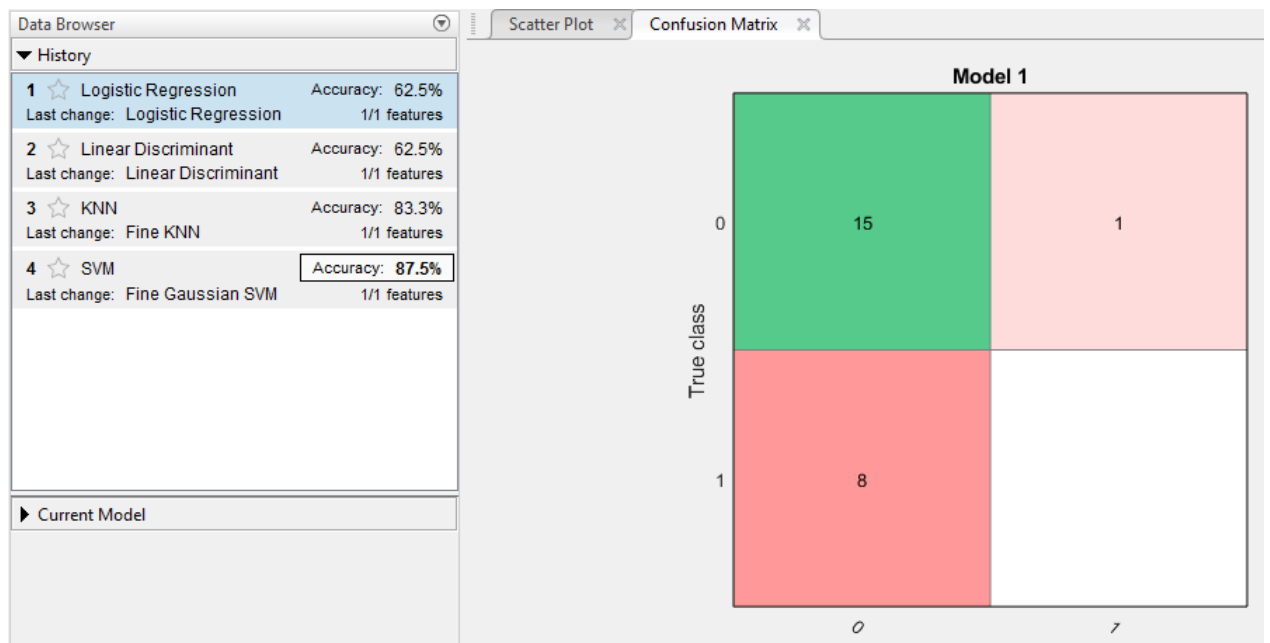


Figure 1.9 - Confusion matrix on logistic regression. Hourly shopping data.

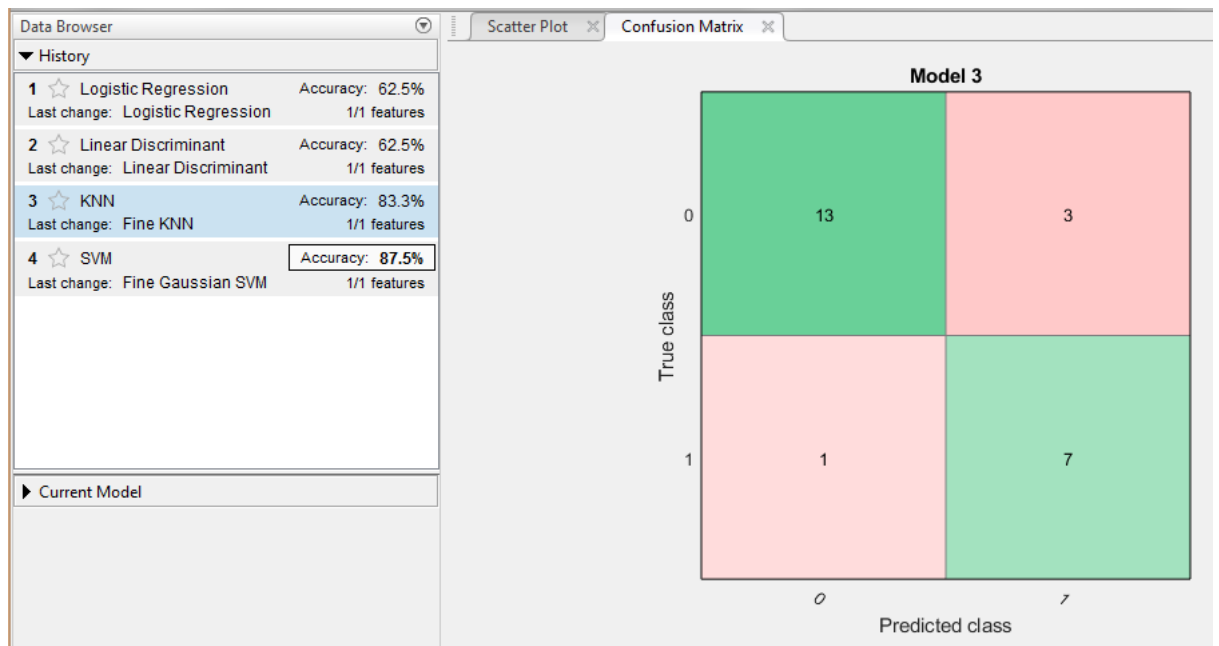


Figure 1.10 - Confusion matrix on KNN. Hourly shopping data

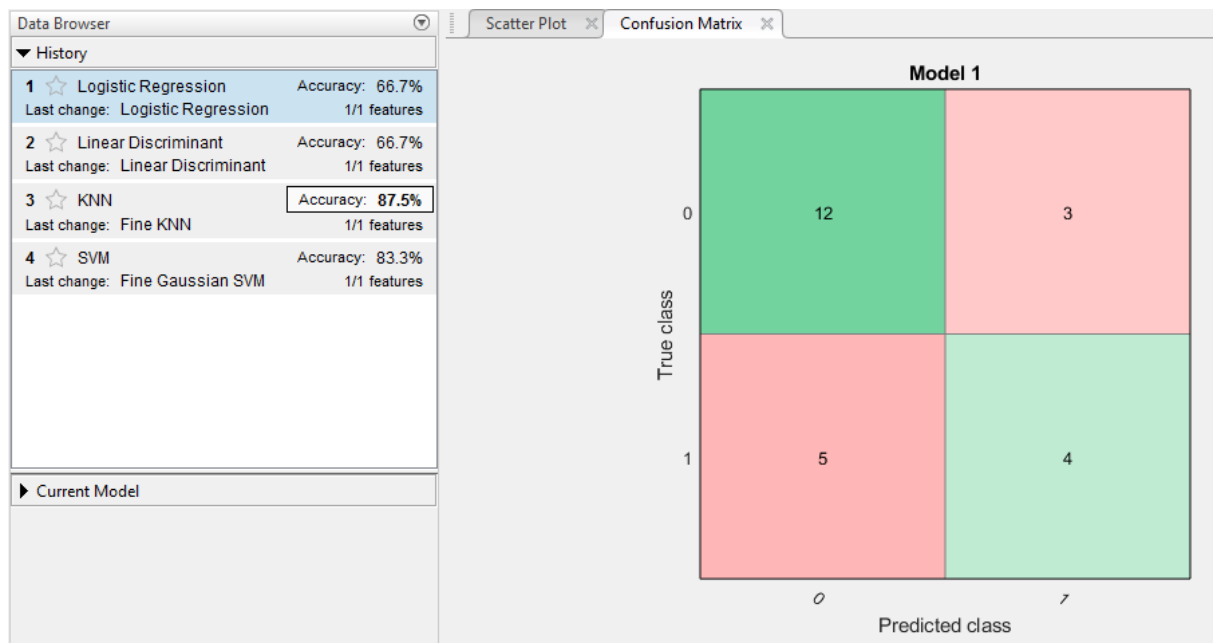


Figure 1.11 - Confusion matrix on logistic regression. Based on hourly social data.

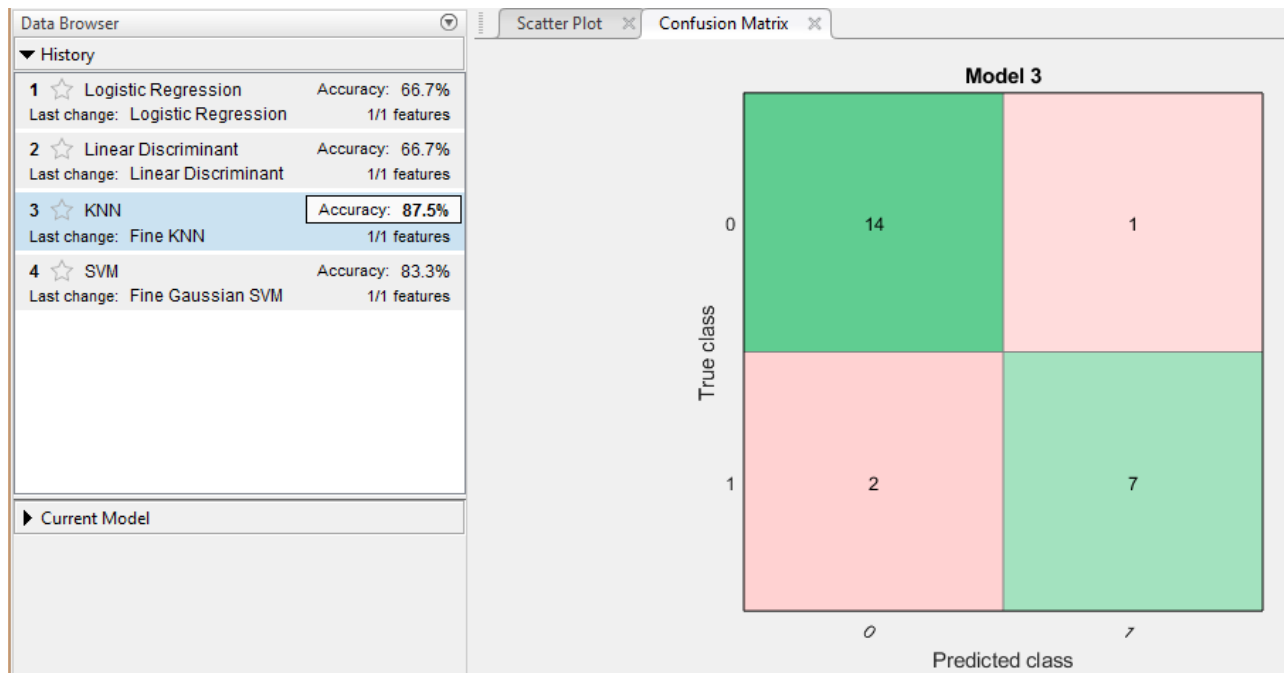


Figure 1.12 - Confusion matrix on KNN. Based on hourly social data.

Furthermore, I have created a table that ranges from 1-1440 which these numbers corresponds to minutes past 12:00am. I have also made the table follow a linear trend. For example, the parking bay starts as vacant but gradually becomes occupied as time passes throughout the day.

The table is too big to display but these are the results and graph of the created data:

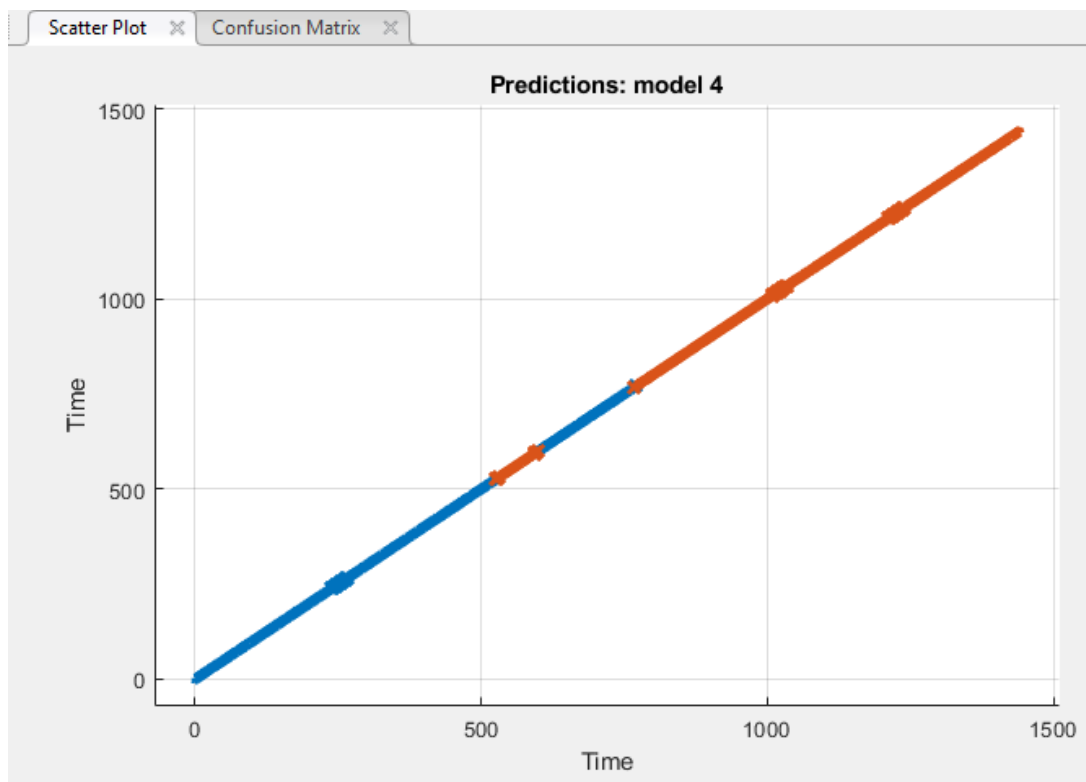


Figure 1.13 – Plot of test data. Blue indicates vacant and orange indicates occupied.

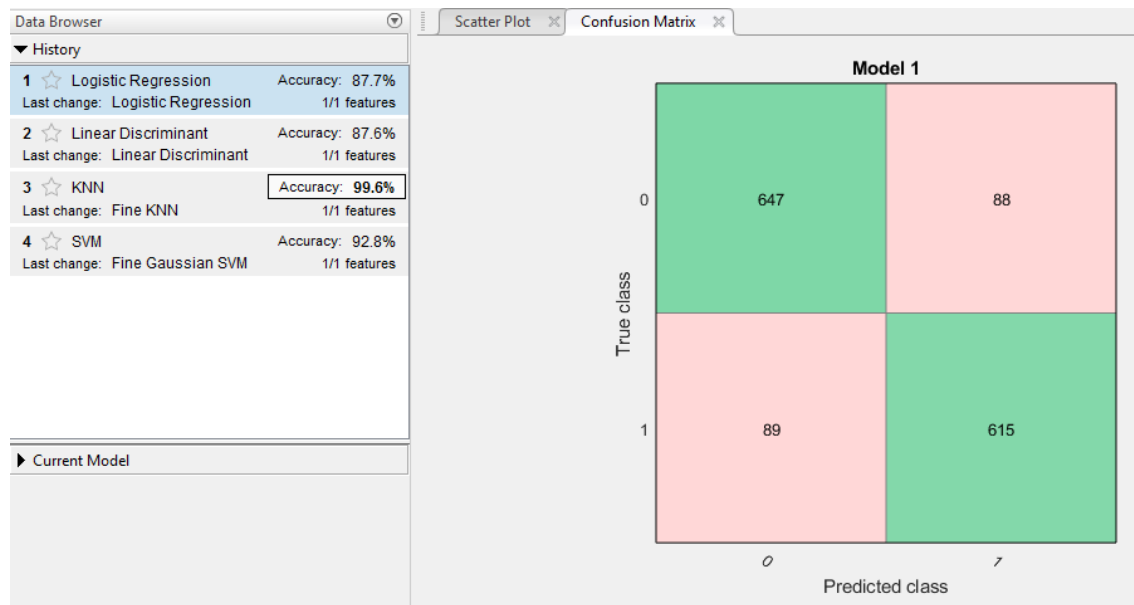


Figure 1.14 - Confusion matrix based on logistic regression. Based on minute data.

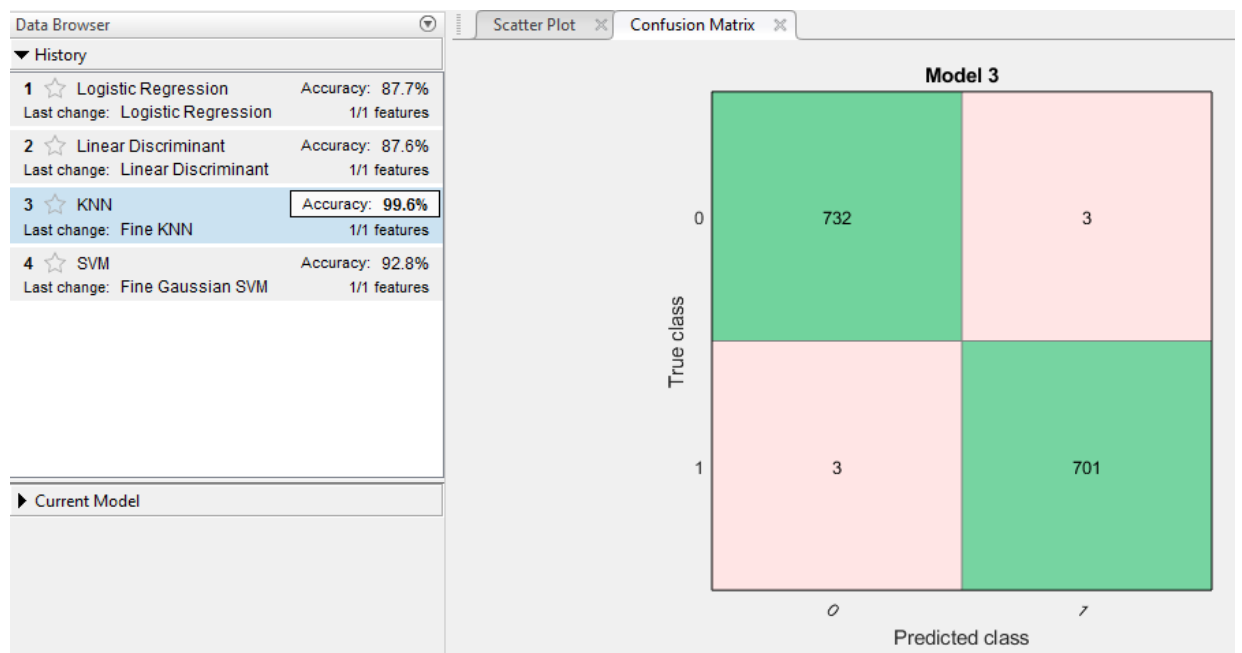


Figure 1.15 - Confusion matrix based on KNN. Based on minute data.

2.6.4 - Conclusion of data

As the above graphs suggests, some algorithms are better than other depending on the data. Logistic regression performed the same as the linear discriminant algorithm. From the tests, they both averaged at an accuracy of 68.8%. Whilst on the other hand, 'Fine KNN' scored an average accuracy of 90.5%. The confusion matrix also shows what the algorithm predicted and what it was supposed to predict. On the hourly shopping confusion matrix, logistic regression predicted a status of 1 but could not actually attain a true value of the status. Therefore, this suggests logistic regression is not typically suitable for data that has a lot of variety; also, it does not typically follow a linear trend. On the other hand, if the data follows a linear trend, logistic regression does a good job as the confusion matrix reported healthy numbers on the above picture.

Furthermore, KNN outperforms logistic regression as this algorithm can be used on data where there is not necessarily a linear trend. On all occasions, Fine KNN managed to get a higher accuracy than logistic regression, hence, reporting an average accuracy score of 90.5%. The default settings were used when testing the KNN algorithm, which is when $K = 1$. KNN generally works by calculating the distance of the new element from all already stated elements. Once the distance has been calculated, it picks the Kth closest elements the new point it is close to and examines the elements. Based on which type of element is more popular within the group of Kth elements, it will be the prediction of the algorithm.

2.7 - Introduction to the KNN learning algorithm

The KNN algorithm is a lazy algorithm. This means that there are typically no training phase and no variables/constants to calculate in a given equation (sebastianraschka, no date). The algorithm uses the data you feed to it directly. Despite this, KNN is advantageous in a number of situations, as seen in the Matlab findings. The general flow of the algorithm is to plot your data and give a new point. Then, it checks the nearest K elements from that point based on some property/properties. Despite its simplicity, the algorithm can easily be implemented in many programming languages without any additional libraries or any third party support which makes it ideal (Adi, 2017). Few advantages of KNN are that it has a high level of accuracy, as well as, it being versatile. It can be useful for classification and regression scenarios.

The disadvantage to this algorithm is the memory cost it generates. It is computationally expensive as it needs to store all the data and cannot use any predefined constants/variables to quickly calculate a new entry. Therefore, this algorithm does take a toll on the memory.

Whilst the algorithm does have its pros and cons, the cons can easily be rectified by making sure the data does not get big. In this context, only keep one days' worth of data rather than constantly storing the data. Furthermore, from Java 8, streams can be used to iterate through the elements, giving the option to iterate it even faster by utilizing the other cores in the CPU through the use of the `'parallelStream()'` method.

2.8 - Introduction to the logistic regression algorithm

Unlike the KNN algorithm, logistic regression utilises models based on the training data rather than the actual data itself. It fulfils the same purpose as the KNN algorithm, giving a binary output.

The logistic regression comprises of a sigmoid function: $1 / (1 + e^{-Z})$ (Jason, 2016), which is essentially the heart of the algorithm. This sigmoid function ensures that the data is kept with the 0 and 1 limits. This is to be expected as the value closer to 0 represents a class and the value closer to 1 represents another class (e.g. class 0 may represent occupied parking bay whilst 1 may represent vacant parking bay).

In the above equation, Z would correspond to the coefficients based on the dataset, which would typically look like ' $Z = b_0 + b_1 * x_1 + b_n * x_n$ ' depending on how much data is in the dataset (Jason, 2016). Finding the value of these coefficients depends on what type and method of learning algorithm will be chosen. In this case, it will be stochastic gradient descent.

Stochastic gradient descent is a learning algorithm which has the form: ' $\theta = \theta + \alpha \nabla \theta J(\theta; x(i), y(i))$ ' (Stanford, no date). Essentially, it works by calculating the prediction for each instance and calculating the error for each prediction. ' α ' is known as the learning rate and finding out the ideal learning rate can sometimes comprise of setting the learning rate really small and slightly incrementing it to see how it performs (Hafidz, no date).

After iterating the stochastic gradient descent through all the data, it starts again. This is known as an epoch and the more epochs that is conducted, the better the outcome. Once the coefficients are known, it is a matter of plugging them into the above equation.

2.9 - Microcontrollers and Components

The sensor will be built using a NodeMCU microcontroller. The reason for this is because NodeMCU is an open source hardware microcontroller with a thriving community. As a result, it will be easier to learn as there are many guides on how to work with the NodeMCU. Furthermore, there are guides on how to connect and use other components, such as a circuit breadboard, LDR (light dependent resistor) and an ultrasonic sensor.

A breadboard is one of the major components when designing a prototype. It is a solderless device used for temporarily designing a circuit electrically or testing a part of a circuit. Breadboards are mainly used in DIY projects. The breadboard is a rectangular structure comprising of pins laid out horizontally and vertically like thus:

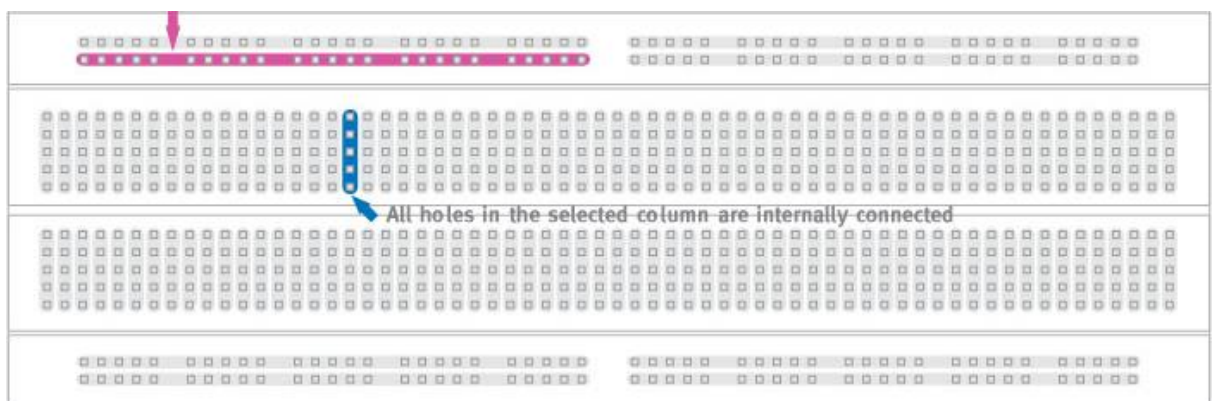


Figure 1.16 - Breadboard Overview (Wiring. (No Date))

The two outer rows on each both outer edges of the breadboard are all connected, which are typically used for providing voltage to one row and making the other row grounded. Each column in the two inner section of the breadboard is all connected vertically. Therefore, if current was provided to the top row of the blue rectangle, all the pins in the blue rectangle would be electrically charged.

2.9.1 - Analogue readings

Some components provide an analogue reading rather than a digital reading. An analogue reading is essentially reading the amount of voltage flowing through. For example, a thermistor or a potentiometer would give analogue readings as there is a distinct value it cannot output, such as when the output is not binary. The NodeMCU has several pins dedicated to read analogue, as well as, having an inbuilt analogue-to-digital converter. In addition, the library provides a useful method 'analogRead()' for reading such entities (Arduino, no date). This will be highly useful when wiring the thermistor into the circuit during the implementation phase.

3 - Aim and Objectives:

The aim of this project is to help solve an on-going problem most car drivers face daily, which is finding an available bay to park in. This project will aid and help car drivers find a car parking bay that they will be able to park in without driving further than necessary.

The objectives of the project are:

1. Creating the sensor :

This will be achieved by creating a sensor (using an Arduino Uno) that will record the data from an ultrasonic sensor, GPS module, a thermistor module, as well as, an integrated Wi-Fi module.

2. Code the sensor to detect a change in the environment :

Once the modules acknowledge a drastic change in the environment, the results will be shown to the user via an app. Hence, indicating whether or not the bay is vacant or occupied based on the change in environment.

3. Incorporate machine learning :

The app should have some classification algorithm as this will enhance the functionality of the app, as well as, increasing the usability of the app.

4. Create the server and the database :

The server will be used to send the data to and from the app and the sensor, over the HTTP/HTTPS protocol. The database will be used to store the sensor's information it gathers.

5. Design of the sensor :

Designing an aesthetically pleasing sensor in order to make it more efficient in terms of accurately reading the changes in environment.

6. Developing the app :

The app will be developed for the android platform. As well as that, it will be developed using the 'Agile' methodology. The app will fetch data from the server and populate a map, which will show parking bays that are vacant/occupied near the user. Furthermore, the app will need to be user friendly and not have too much elements for the user to interact as it can pose a danger when driving. This app will typically be used whilst the user is behind the wheel of a car; every effort will be made to ensure the user focuses on the road and not on the app. The app will be laid out in such a way that the user should know enough information by interacting with it by no more than 10 seconds.

7. App should show parking bays available near the user through the use of GPS.
8. App should provide directions to the parking bays by showing a route from the user's location to the parking bay for increased ease of usability.

4 - Requirements

To gather the requirements for this project, a range of requirement gathering techniques has been used, such as creating a prototype to see any improvements or to see any missing requirements. By doing this, the practicality of the prototype will be determined. Furthermore, different devices/apps already out in the public trying to combat the problem stated were compared. By comparing and contrasting different devices to each other, I will be able to gain an insight into the common requirements they fulfil, as well as, seeing any potential requirements they may have missed out. Thus, making the prototype unique.

Firstly, I have looked at alternative solutions proposed by different companies to combat the problem stated. One of the IT giants, Google, is already making progress in terms of finding a solution to this problem. They have implemented a solution and is available on Google Maps. Their solution works using historic data with machine learning to predict the availability of car parking bays (Google, 2017). Whilst this is a step in the right direction to solving the problem described, it is not as accurate as having a physical sensor embedded to the parking bays. As the world progresses further in technology, more and more devices are being connected to the internet. The concept of IoT (internet of things) will be greatly beneficial here and will outperform the use of machine learning.

In addition, another start-up company called AppyParking, aims to tackle the problem by using a sensor embedded onto the road. It provides real time updates to the user via their app. Their sensors has immensely helped with the problem outlined as it has helped Coventry City Council recognise an approximate £475,000 lost revenue due to parking bays that were not either used or placed efficiently (AppyParking, no date) whilst providing users real time updates to the available parking bays, which in turn has led to 30% fewer miles driven looking for bays and 22% reduction in parking congestion during peak hours. Whilst this is similar to my proposed solution, it lacks the use of machine learning which would be greatly beneficial because in the unfortunate event that the sensor stops working, the app would not be able to notify the user if the bay is vacant or occupied.

Moreover, another company that is closely related to this field is Inrix. They work closely with companies such as BMW and Audi. Inrix also gathers data regarding car parks and congestion on the roads in order to provide a huge collection of data in order to understand the current trend and patterns of road usage, as well as, making the roads more efficient. One idea that they have proposed to combat the problem statement is to use ultrasonic sensors (Jason, 2017). Their method revolves around fitting cars with ultrasonic sensors as opposed to more evasive methods, such as physically implanting sensors on the road. This has its pros and cons. It is more economically viable to implant sensors on cars rather than on roads, however, it does have its drawback. For example, the amount of data that would be gathered about the driver and how Inrix would store, or even share, the data.

By comparing the above innovations proposed by these companies, as well as, realising other possible solutions that were implemented in the literature review section, I can see each of them does have its strength and weaknesses. Therefore, I aim to build my prototype to include the main logical purposes the above solutions serve and also to include what some of them have missed, such as machine learning. I will be using a microcontroller to build the prototype, a Wi-Fi module for wireless communication, as well as, wiring a thermistor and an ultrasonic sensor.

Furthermore, I will need to cater requirements for the UI of the app. Below, are the images of the AppyParking app for Android. Initial reaction to the app was that there is too much information on one screen. It should have been spaced out more and having all the tabs and icons centralised at the bottom of the screen was not efficient. Upon using it, it was pretty confusing to navigate through the app because the tabs on the icon did not represent what the tab would do/perform. Even more, the app became unresponsive as I navigated through the map. This might be due to the fact that it tries to display all the map data at once, hence, sending multitudes of http requests and receiving responses in a short amount of time whilst the phone was busy constantly updating the UI.

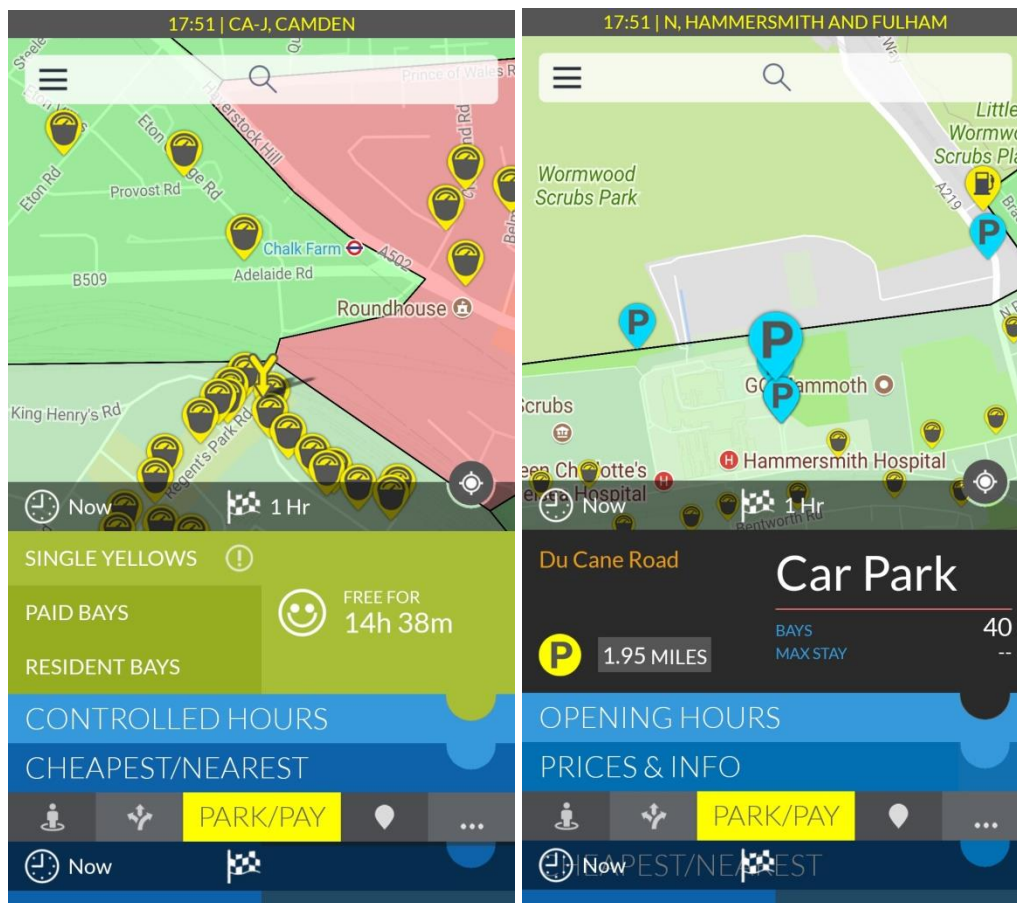


Figure 2 (left) – UI of AppyParking app. Figure 2.1 (right) – UI of AppyParking app.

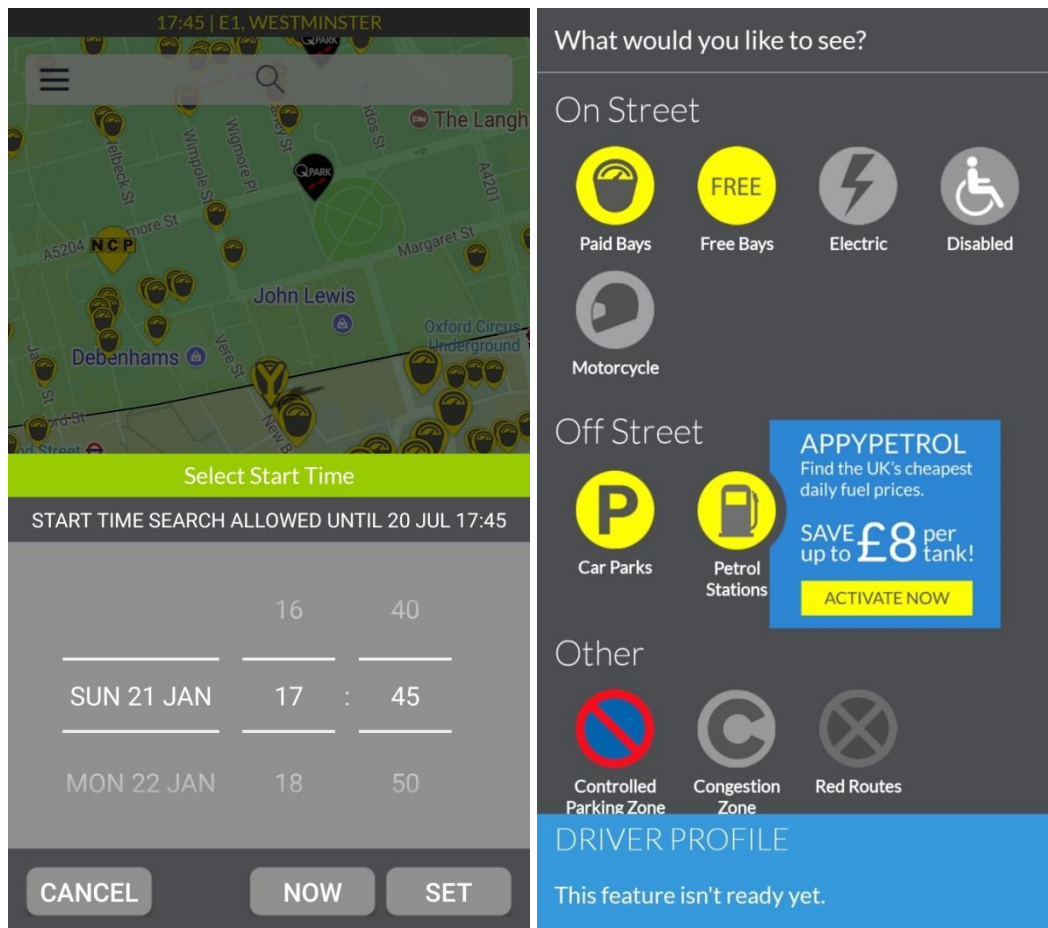


Figure 2.2 (left) – UI of AppyParking app. Figure 2.3 (right) – UI of AppyParking app.

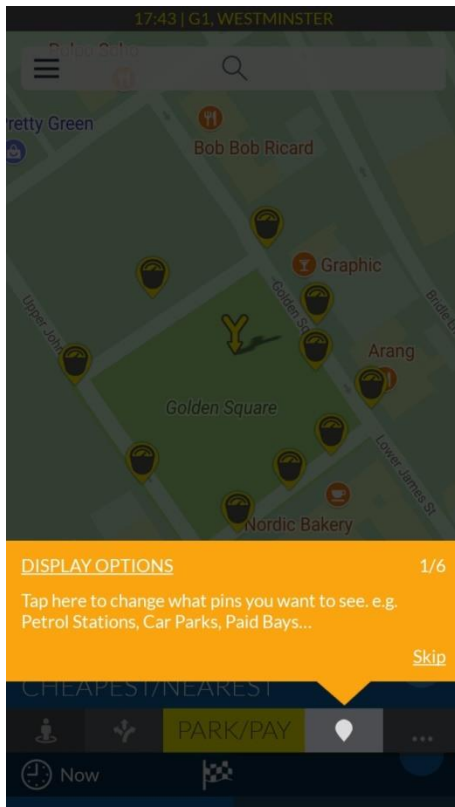


Figure 2.4 – UI of AppyingParking app.

As a result of this, I will be developing the app with a simplistic UI. One has to keep in mind that this app will mostly be used when a driver needs to find a parking bay. Hence, the UI will not have many navigation tabs in the front page and it will need to give a clear and concise outcome within ten seconds as the driver cannot have their attention diverted from the wheel.

However, Google Maps has a very simplistic layout. This is important because the Google Maps app can be found on many devices across different operating systems (Ben, 2017). Therefore, it is important that many users, regardless if they are young or old or inexperienced with tech, should have a basic intuition on how to use it. From a personal viewpoint, the layout is well spread as the map takes up the whole screen and most of the options are hidden in the task bar, therefore less clutter on the screen.

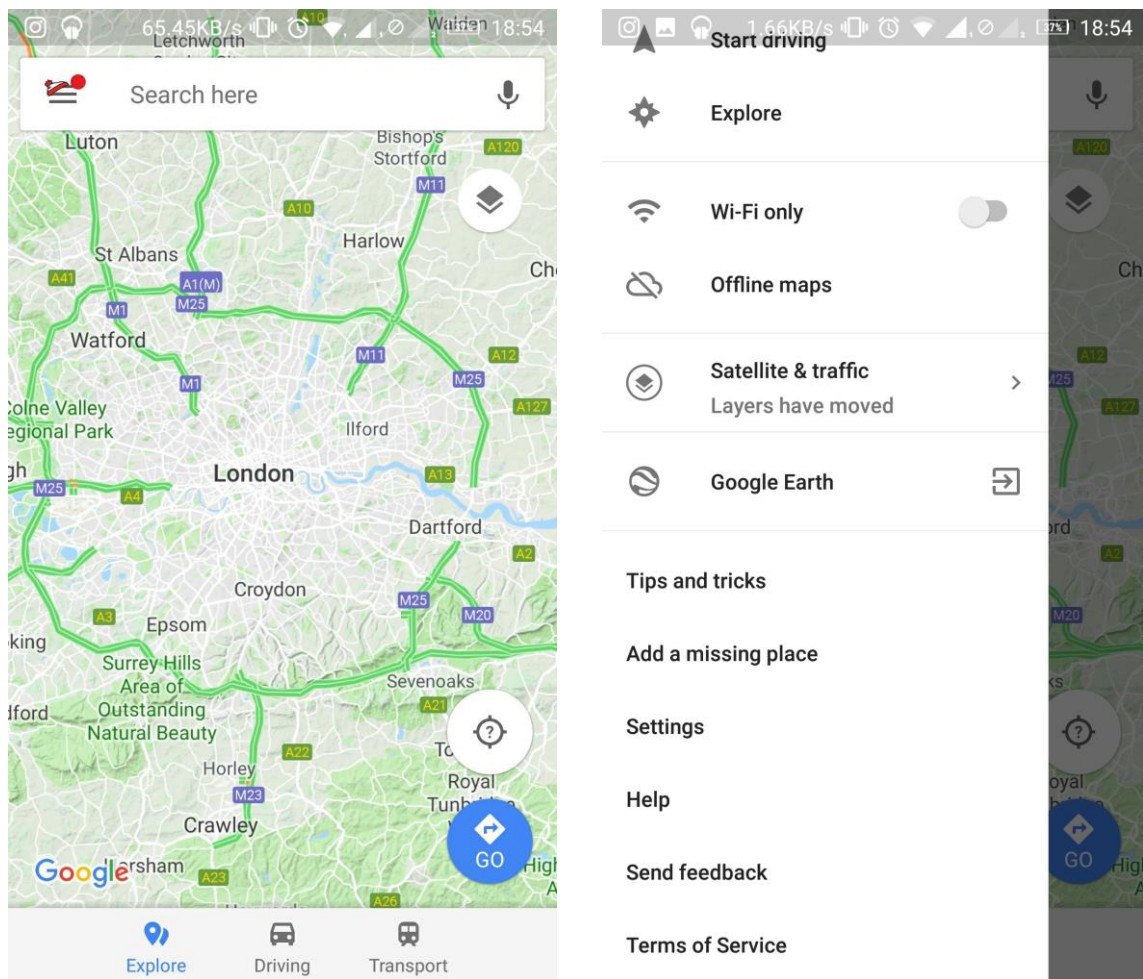


Figure 2.5 (left) – UI of Google Maps. Figure 2.6 (Right) – UI of Google Maps.

Despite Google maps not being used to find parking bays, it does provide one of the main functionalities my app will have to do, which is to output a map. Therefore, the simple layout from Google maps will be followed as it is the design and layout that majority of phone users are familiar with.

5 - Design of software

This section of the report will cover all the design aspect for the project. It will include action diagrams, sequence diagrams, case diagrams, as well as, storyboards. Furthermore, it will show the project in an architectural manner in terms of the structure and how the app is built by UML diagrams.

5.1 - Sequence diagrams

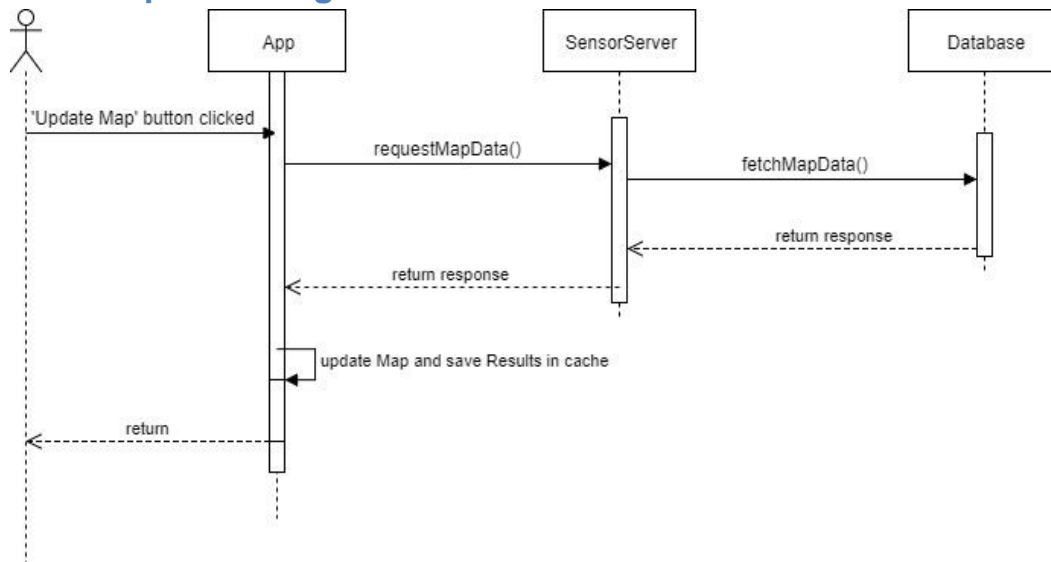


Figure 3 – Shows the interaction with user and app upon happy scenario.

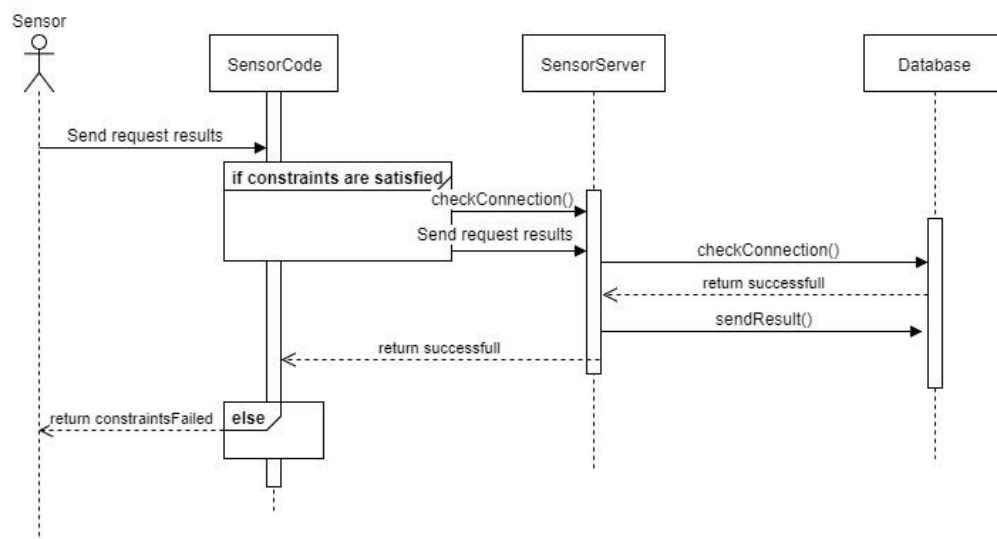


Figure 3.1 – Shows the sensor interaction with the server and database

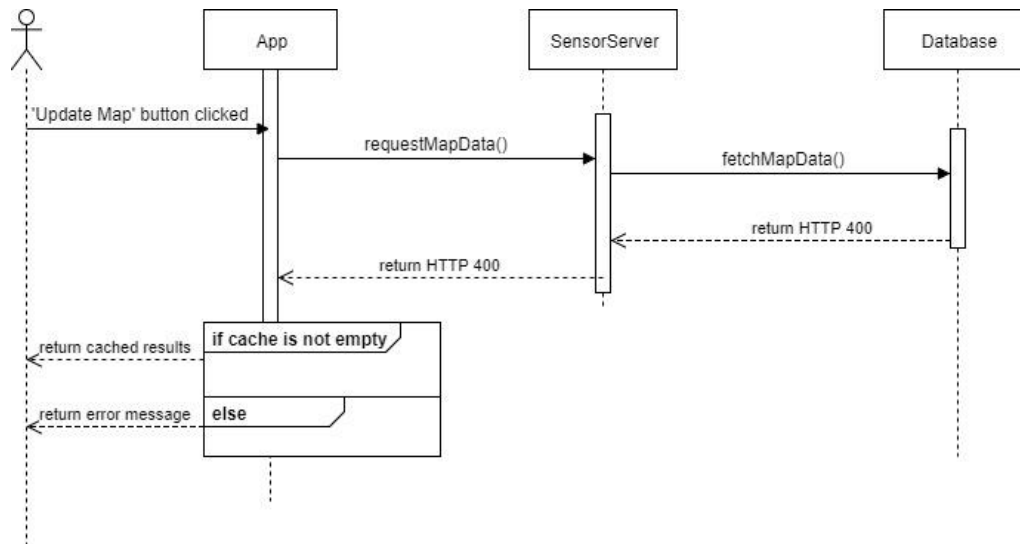


Figure 3.2 – Shows the interaction of the user if server cannot connect to database

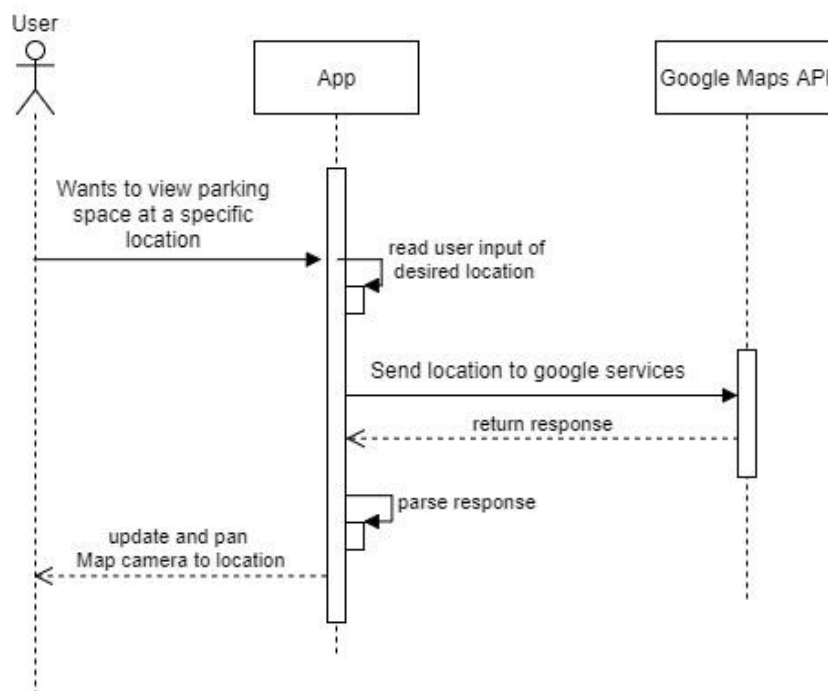


Figure 3.3 – Shows the interaction of the user when the user wants to check for a parking spot in a certain area.

5.2 - Action Diagrams

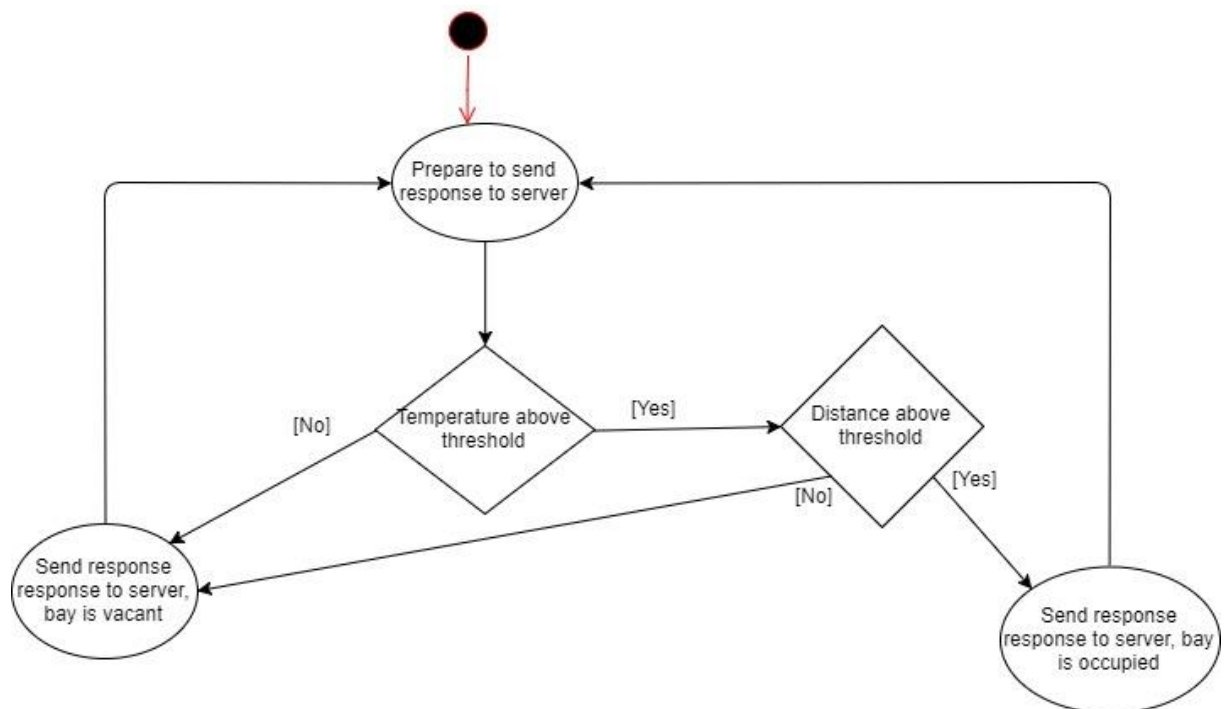


Figure 3.4 – Shows the activity diagram of the sensor

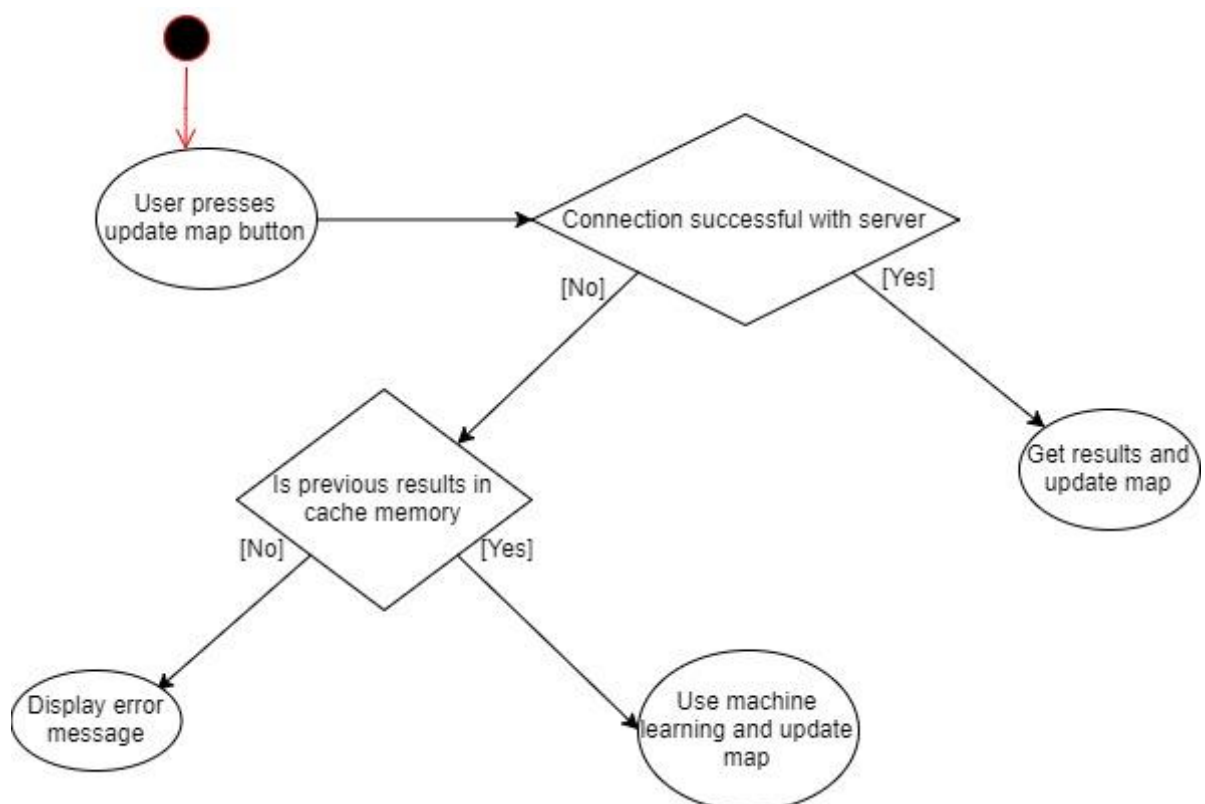


Figure 3.5 – Shows the activity diagram of the user interacting with the app.

5.3 - Use Case Diagrams

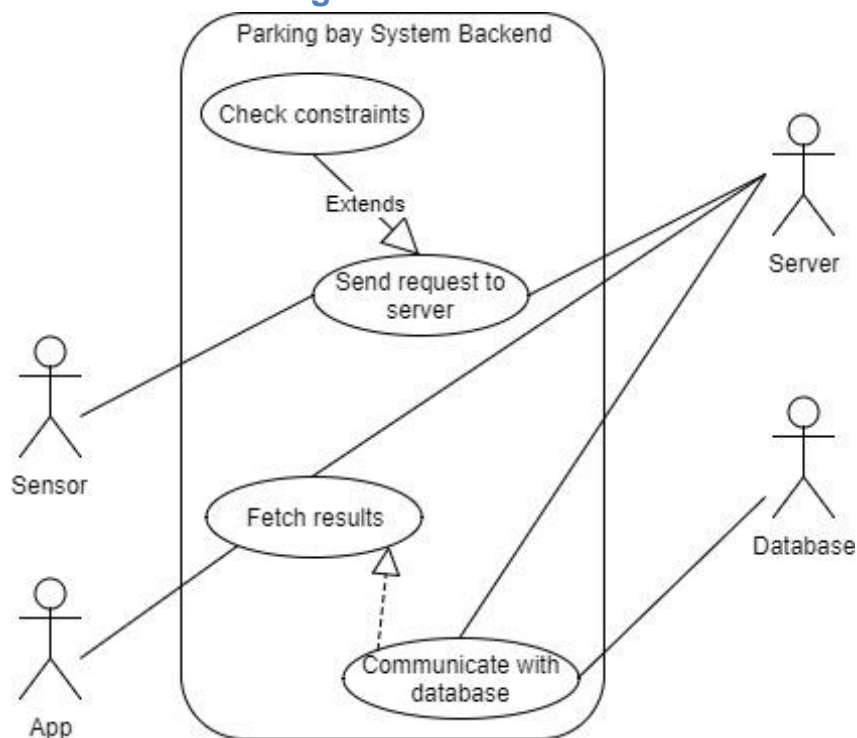


Figure 3.6 – Shows the uses cases of the backend of the system

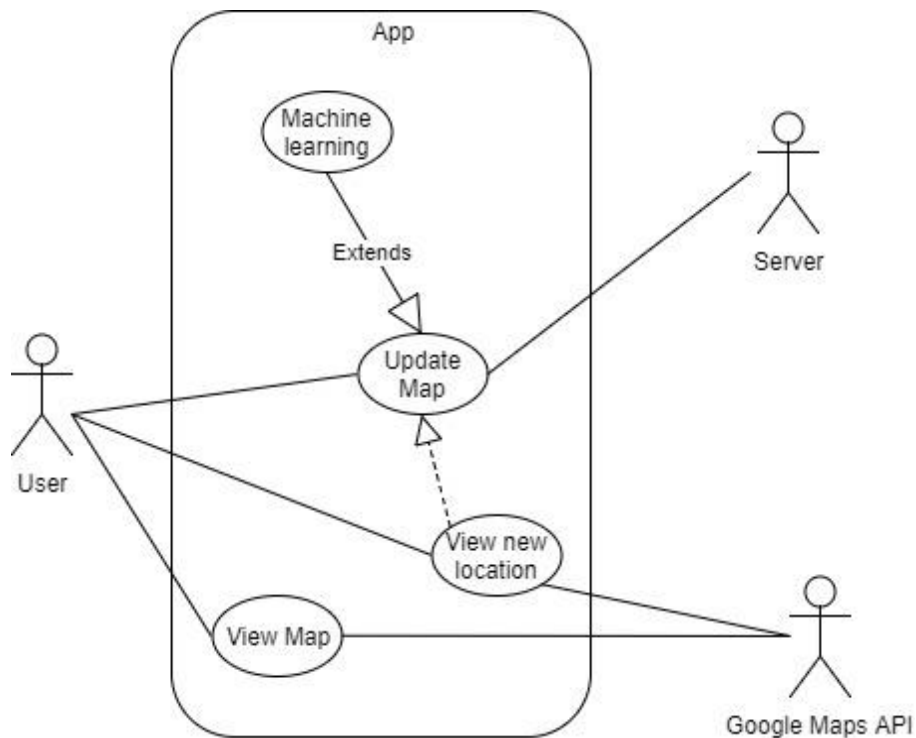


Figure 3.7 – Shows the use cases of the user requesting to a new location to look for a parking bay

6 - Methodology

This section covers the coding methodologies that will be used in the implementation. Even more, it will introduce general methodologies (such as using Git and working in an agile manner) that will aid in implementation phase.

6.1 - Source Control

Source control is important when it comes to writing code. Not only is it a generally good practise to use but this is also practised in many jobs in the computing industry. There are many varieties of source control out there such as Git, Subversion and Mercurial to name a few. In this project, I have chosen to use git mainly because I am familiar with the concept of it.

The use of git is needed but it needs to be used in a suitable manner to prevent work from being overwritten in an accidental commit and accidentally pushing invalid code to the master branch. To prevent this from occurring, the GitFlow methodology will be used. GitFlow is a branching model for Git as it very clear and concise to use. Due to the way the GitFlow model is structured, it is quite easy for developers to release emergency fixes to patch any serious bugs. Furthermore, it allows developers to work collaboratively due to the nature the branches are laid out. Below is an example of how the GitFlow model will look like:

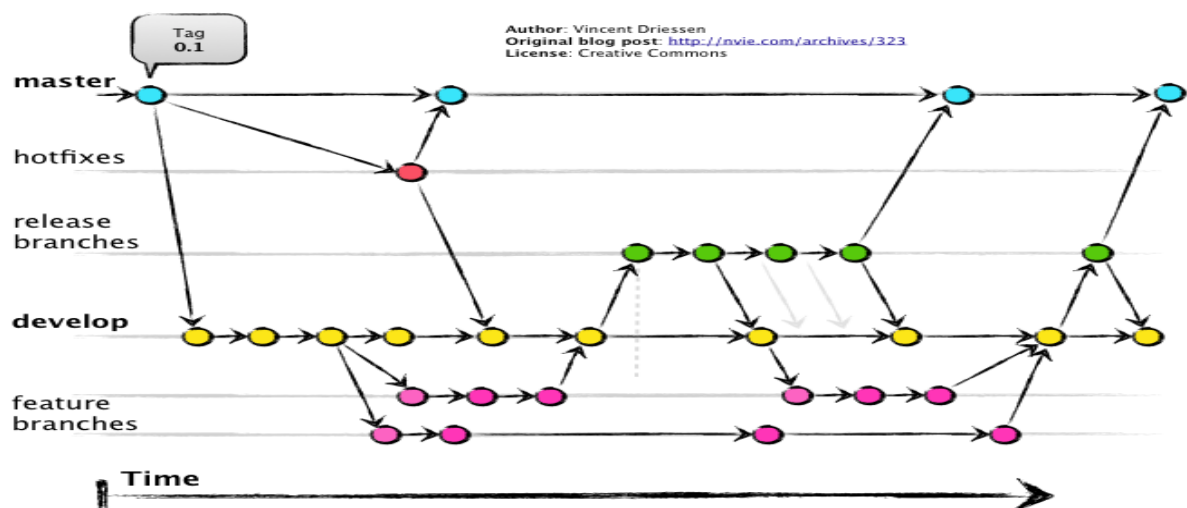


Figure 4.1 – Typical GitFlow chart (Driessen, 2010)

As you can see from the above diagram, the branches are laid out in a nice and structural manner in which it is easy to revert back to a previous commit if necessary, as well as, making it easier to track what is going on.

6.2 - Agile ethics

Furthermore, I will be implementing the SCRUM methodology in the way I work. Agile has become popular over the years and a lot of companies is embracing the new style of work and favouring it rather than the waterfall method. SCRUM is a subset of the Agile methodology, which is based on iterative development. SCRUMs core concept is the use of development cycles called Sprints which allows one to dynamically adapt to changes, whether it be from issues in code or requirements changing. Furthermore, issues/tasks are placed in Sprints and within the duration of the Sprint, one aims to get everything finished. Upon completion of the Sprint, there is a Sprint planning in which issues and tasks are taken from a backlog and placed into the current Sprint. By having this ability, this is what makes SCRUM more dynamic and adaptable to change (CPrime, no date). See the following page for a gantt chart that will also be used to help prioritise workload, as well as, making sure everything is done in due time.

In regards to the software aspect, it is more of logical errors and bugs that will cause a lot of the problems. This will be fixed by following a TDD (test driven development) approach, as this is a sort of development style heavily used in the industry. It ensures that codes are in working condition since it shows codes have been revolved around the unit tests rather than the unit tests being revolved around the code.

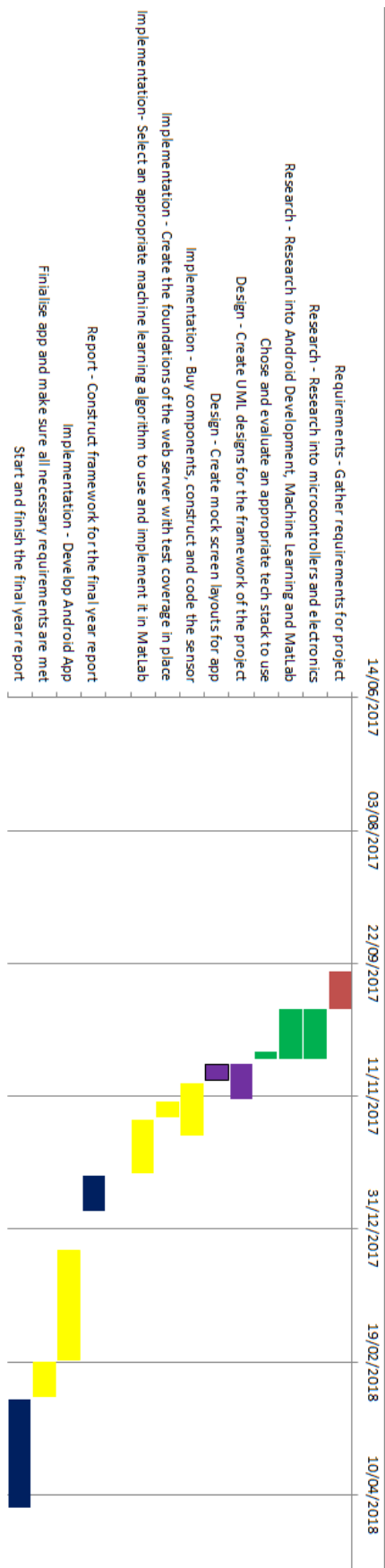


Figure 4.2 – Gantt chart depicting the work flow of the project.

6.3 - Tech Stack

Maven – Maven is a build automation tool. It is primarily used for Java projects. The advantage of using Maven is that it can download the libraries you wish to use in your project. Furthermore, it will also download further dependencies the library requires, hence, it is efficient when it comes to deployment. All the lifecycle is contained in a file called POM.xml in which you can choose what happens in different stages as your project gets built.

Spring Boot – Spring Boot is part of the Spring framework, a highly used framework which incorporates a lot of nice features like dependency injection and web applications (i.e. Spring MVC). Maven is incorporated with Spring Boot, which is ideal for me, as well as that, it contains an embedded servlet container so it will have the choice of my server: Tomcat. This is ideal in terms of deployment as everything will be packaged in one jar file.

MongoDB – MongoDB is a noSQL database. It uses JSON-like documents with schemas, which will be handy for me as I will be handling JSON structures from the server to the database and to the app so it will be better to keep one uniform structure throughout the process. Secondly, the reason behind choosing mongoDB is due to the fact that noSQL is known for its speed as it can be horizontally scaled. This means that the more database servers you have in the server pool, the faster it is to perform operations rather than adding more power (i.e. powerful hardware) to the servers (known as vertical scaling). I do plan on taking this project and developing it to a commercial grade. Hence, it is important to think about the overall big picture as, theoretically, there will be thousands of sensors writing to the database cluster.

Java – Java is an object orientated programming language. It is well known throughout the industry and has been used in the early days of programming. Through its many updates, Java has become one of the most popular languages. As of January 2018, the current version of Java is Java 9. I will be developing the majority of my project in Java: more specifically, the server and the app. The reason for choosing Java is because android apps are primarily developed using Java and is not worth the hassle of trying to develop an android app using another language through other means such as using their

NDK (native development kit) which allows developers to code android apps in C, C++.

NodeMCU – NodeMCU is a microcontroller. More specifically, it is an open source hardware and software company. I will be using this microcontroller to build my sensor. The reason for using a NodeMCU is because of the lack of experience with electronics and microcontrollers I have and the Arduino and NodeMCU have a thriving community to learn from.

7 – Implementation

This chapter will revolve around the details of the core implemented parts.

7.1 - UI of app

The following shows how GUI (graphical user interface) of the app was implemented in the app.

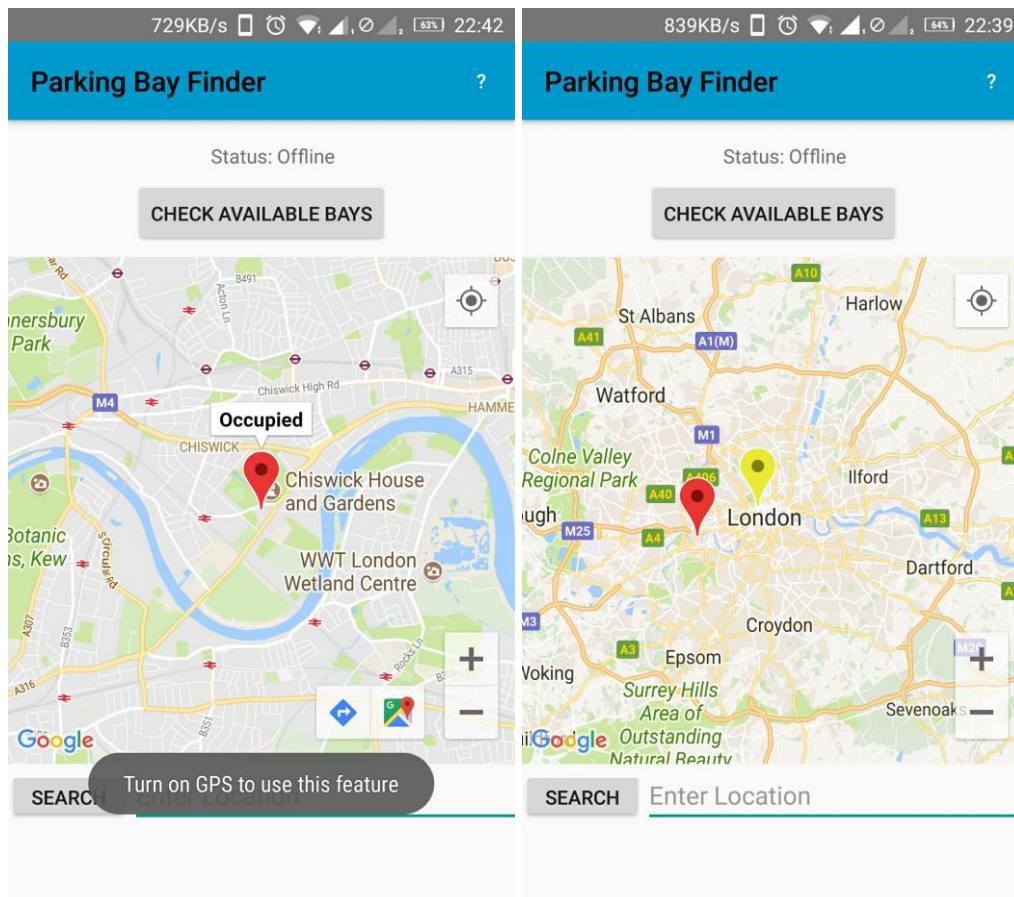


Figure 5 (Left) – UI of app showing message to user. Figure 5.1 (Right) – UI of app.

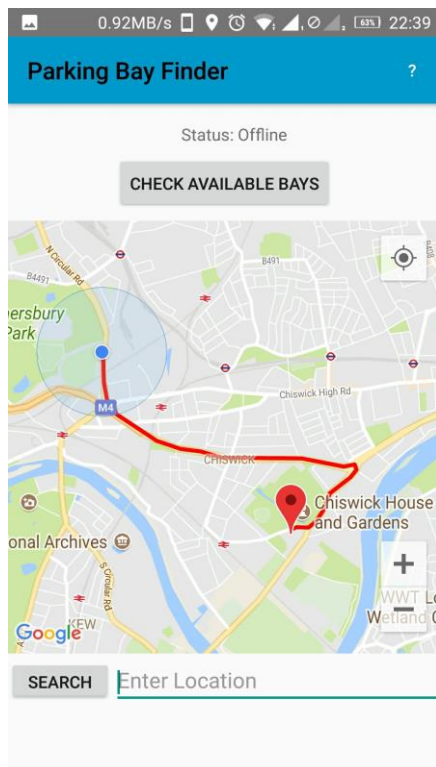


Figure 5.2 – Showing navigation to a parking bay to the user.

From the above diagrams, we understand that the UI of the app is simple yet effective. It displays a vast majority of information in one screen whilst not cramping additional components, unlike the AppyParking app. Furthermore, it is designed to be used for a short duration of time (an average of 10-30 seconds) as it takes into consideration a driver's attention span and road safety.

Furthermore, it is UI friendly as tips pop-up for the user at different intervals of its programme, which is seen in Figure 5 above. Also, there is a manual on the app that can be accessed from clicking the top right '?' button, which is displayed in all the screenshots. This has been designed to provide further information to the user on how to efficiently operate the app and is intended to be read before driving and the information given is condensed and appropriate for all drivers to understand.

The layout consists of two files: 'activity_main.xml' and 'content_main.xml'. The former layout file consists of a toolbar (figures 5-5.2) and the latter is where the heart of the layout lies wherein major components can be found. It uses a constraint layout as this layout allows for large and complex layouts with flat hierarchy, which is the most suitable type of layout to use in this situation.

7.2 - Data structure of the parking bay:

The data structure of the parking bays was designed to keep everything simple and elegant. Therefore, the class of the parking bay was created based on the idea of pojo (plain old java objects). Using the definition of a pojo from the Spring community, 'POJO means Plain Old Java Object. It refers to a Java object (instance of definition) that isn't bogged down by framework extensions.'(Spring, no date). In essence, this means that the class should not extend or inherit anything from any framework class.

The model class, which represents a parking bay, consists of getters and setters. As well as that, the getters and setters conform to the JSON schema so that the json2pojo plugin will be able to create a java object from the JSON response. Furthermore, there is an arraylist which takes 'Bays' object. The bays class is used for the KNN algorithm that will be explained in the next section.

The following is a typical JSON response from the server that the app would parse:

```
{
  "_id": "50",
  "longitude": "-0.141136",
  "latitude": "51.518220",
  "timeDateOfUsage": [
    "14\01\18 12:40:29",
    "0"
  ]
}
```

The above ID key is unique and represents an individual sensor. The longitude and latitude represents the area of where the sensor is currently located. The 'timeDateOfUsage' is an array of String text that has the order of first outputting the date, time and status. For example, in the above JSON response, at 12:40am on the 14/01/2018, the status of the car park is vacant. The structure of the object can be seen from the following snippet:

```

public class SensorBay {
    public ArrayList<Bays> timeUsage;

    public SensorBay(){
        timeUsage = new ArrayList<>();
    }

    private String id;

    public String get_id() { return this.id; }

    public void set_id(String id) { this.id = id; }

    private String longitude;

    public String getLongitude() { return this.longitude; }

    public void setLongitude(String longitude) { this.longitude = longitude; }

    private String latitude;

    public String getLatitude() { return this.latitude; }

    public void setLatitude(String latitude) { this.latitude = latitude; }

    private ArrayList<String> timeDateOfUsage;

    public ArrayList<String> getTimeDateOfUsage() { return
this.timeDateOfUsage; }

    public void setTimeDateOfUsage(ArrayList<String> timeDateOfUsage) {
this.timeDateOfUsage = timeDateOfUsage; }

    @Override
    public String toString(){
        return "ID: " + get_id() + ". Longitude: " + getLongitude() + ".
Latitude: " + getLatitude() + ". time: " + getTimeDateOfUsage().toString();
    }

}

```

7.3 - Machine learning implementation:

One of the fundamental requirements for this app was to implement machine learning. The reason behind this choice is in the unfortunate event when a server may be down, the app will be rendered useless due to there being no source of result displayed to the user. Therefore as a precaution, machine learning had to be implemented. Another situation that arose from implementing this was where the machine learning should be implemented. If implementing on the app, then the algorithm used would need to be lightweight in terms of memory size, as well as, not being CPU intensive by performing complex calculations. This would drain the battery of the phone, which would not be user friendly. If implementing on the server, there would need to be another server involved just solely for machine learning that would compute the algorithm and send data to the app. However, the same scenario would still arise: what if the server went down? Therefore, the logical way forward would be to implement it on the app whilst keeping complex calculations to a minimal.

7.3.1 KNN algorithm:

The KNN algorithm is essentially a straight forward algorithm: find the closest variables to your chosen point and take into account the nearest K variables depending on their properties. Whichever types of elements is more prominent, that will be the output. Because this algorithm will be implemented on the app, keeping the calculations as straight forward as possible is critical without introducing any complexities. Therefore, $K = 10$ if the size of the data will be bigger than 10 otherwise $K = 1$. It will determine the output using the Bays objects. The Bays object is a representation of the status of a parking bay at a specific minute. Using the status attribute associated with the Bays object, it will get the nearest 10 Bays object and check if the overall status of the Bays is above 5.

Furthermore, the algorithm is implemented in another thread other than the main thread. This is to prevent the app from freezing as the algorithm computes the data which would degrade the usability of the app. Even more, it is good programming practise to put extensive computations on a different thread other than the main thread so that the main thread is not bottled up performing

extensive calculations. Implementation of the algorithm can be seen from the following snippet:

```

public class KNNML extends AsyncTask<Void, Void, Void> {
    List<SensorBay> listOfSensor;
    GoogleMap googleMap;
    GMap gmap;
    Context ctx;
    ArrayList<MarkerOptions> markers;

    public KNNML(Context ctx, GoogleMap map, List<SensorBay> res, GMap gmap) {
        listOfSensor = res;
        this.ctx = ctx;
        this.googleMap = map;
        this.gmap = gmap;
    }

    @Override
    protected Void doInBackground(Void... voids) {
        markers = new ArrayList<>();
        int currentTime = (int) getCurrentTime();
        Log.d("MLL", "CURRENT KNN TIME: " + currentTime);

        for (SensorBay bay : listOfSensor) {
            for (int i = 0; i < bay.timeUsage.size(); i++) {
                int distance = (int) Math.sqrt((currentTime -
                    bay.timeUsage.get(i).getHour()) * ((currentTime - bay.timeUsage.get(i).getHour())))
                bay.timeUsage.get(i).setDistant(distance);
            }
            Collections.sort(bay.timeUsage);

            if (!bay.timeUsage.isEmpty()) {
                int statusOccupied = 0;
                MarkerOptions marker = new MarkerOptions()
                    .position(new LatLng(
                        Double.parseDouble(bay.getLatitude()),
                        Double.parseDouble(bay.getLongitude())
                    ));

                //Get the average if size is greater than 10
                if (bay.timeUsage.size() > 10) {
                    for (int i = 0; i < 10; i++) {
                        if (bay.timeUsage.get(i).getStatus() == 1) {
                            statusOccupied++;
                        }
                    }
                }
                if (statusOccupied > 5) {

```

```

marker.setIcon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED));
        marker.title("Occupied");
    } else {

marker.setIcon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_YELLOW));
        marker.title("Vacantt");
    }
    } else {

        if (bay.timeUsage.get(0).getStatus() == 1) {
            //occupied

marker.setIcon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED));
            marker.title("Occupied");

        } else {
            //vacant

marker.setIcon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_YELLOW));
            marker.title("Vacantt");
        }
    }
    markers.add(marker);
}
}
return null;
}

@Override
protected void onPostExecute(Void aVoid) {
    Log.d("MLL", "EXECUTED MACHINE LEARNING");
    gmap.updateMapKNN(markers);
}
}
}

```


7.4 - Google Maps route finder:

This feature was implemented to give more functionality to the user. When the user clicks on any of the markers on the map, provided that GPS is enabled, it will display the path from the user to the marker. This required using the Google Distance API. Parsing the Distance API was challenging, as the response contained complex structures that had to be parsed and converted to different objects for it to be compatible with the GoogleMap object. Fortunately, the template of parsing the Google Distance response is provided, as a gesture of courtesy, by the open source community and is included in this project. Furthermore, implementing the parsed data for it to be used on the GoogleMap object was tricky, hence, a few snippets of code from a tutorial is used that shows how to use the parsed data (George, 2013). Implementation of the code can be seen from the following snippet:

```

public List<List<HashMap<String,String>>> parse(JSONObject jObject){
    List<List<HashMap<String, String>>> routes = new ArrayList<>() ;
    JSONArray jRoutes;
    JSONArray jLegs;
    JSONArray jSteps;

    try {

        jRoutes = jObject.getJSONArray("routes");

        /** Traversing all routes */
        for(int i=0;i<jRoutes.length();i++){
            jLegs = ( (JSONObject)jRoutes.get(i)).getJSONArray("legs");
            List path = new ArrayList<>();

            /** Traversing all legs */
            for(int j=0;j<jLegs.length();j++){
                jSteps = ( (JSONObject)jLegs.get(j)).getJSONArray("steps");

                /** Traversing all steps */
                for(int k=0;k<jSteps.length();k++){
                    String polyline = "";
                    polyline =
                        (String)((JSONObject)((JSONObject)jSteps.get(k)).get("polyline")).get("points");
                    List<LatLng> list = decodePoly(polyline);

                    /** Traversing all points */
                    for(int l=0;l<list.size();l++){
                        HashMap<String, String> hm = new HashMap<>();
                        hm.put("lat",
                            Double.toString((list.get(l)).latitude) );
                        hm.put("lng",
                            Double.toString((list.get(l)).longitude) );
                        path.add(hm);
                    }
                }
                routes.add(path);
            }
        }

    } catch (JSONException e) {
        e.printStackTrace();
    } catch (Exception e){
    } return routes;}

```

```

/**
 * Method to decode polyline points
 * Courtesy : http://jeffreysambells.com/2010/05/27/decoding-polylines-from-google-maps-direction-api-with-java
 * */
private List<LatLng> decodePoly(String encoded) {

    List<LatLng> poly = new ArrayList<>();
    int index = 0, len = encoded.length();
    int lat = 0, lng = 0;

    while (index < len) {
        int b, shift = 0, result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lat += dlat;

        shift = 0;
        result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lng += dlng;

        LatLng p = new LatLng((((double) lat / 1E5)),
                                (((double) lng / 1E5)));
        poly.add(p);
    }

    return poly;
}

```

7.5 - Saving and loading data:

As pointed out in the Android developers guide, there are multiple methods of reading and writing data in order to save and load. Such methods include using SharedPreferences, SQLite and Room (a new DAO framework developed by Google in 2017). Although the ideal choice would be using a DOA framework when it comes to storing data, SharedPreferences will be used because it is simply a key value pair structure. Furthermore, it is lightweight as it needs no tables or additional structures to be created, unlike in SQLite or Room.

The way this project uses SharedPreferences is by essentially acquiring a list containing parking bay objects and then converts the whole list into a JSON string. This JSON string is then saved as the value of the key-value pair in SharedPreferences. As per the android life cycle, the onCreate() method gets called first. It is in this method that it checks if there is a SharedPreferences available with the key 'listOfSavedBays'. If the key exists, then it fetches the value and converts the JSON string to an ArrayList containing the parking bay objects. This is done through the use of the library 'jackson' – a widely used library in the IT industry catering for JSON to Java conversions. Implementation of this can be seen from the following:

```

public class SaveRetrieveData {
    public static void saveData(Context ctx, List<SensorBay> listOfResponses) {
        SharedPreferences pref =
PreferenceManager.getDefaultSharedPreferences(ctx);
        SharedPreferences.Editor edit = pref.edit();
        ObjectMapper mapper = new ObjectMapper();
        String jsonList = "";
        try {
            jsonList = mapper.writeValueAsString(listOfResponses).toString();
            Log.d("SAVE", "SAVING: " + jsonList);
            edit.putString("listOfSavedBays", jsonList);
            edit.commit();
        } catch (JsonProcessingException e) {
            printToast(ctx, "Error saving data", Toast.LENGTH_SHORT);
            e.printStackTrace();
        }
    }

    public static void clearSharedPreferences(Context ctx){
        SharedPreferences pref =
PreferenceManager.getDefaultSharedPreferences(ctx);
        SharedPreferences.Editor editor = pref.edit();
        editor.clear();
        editor.commit();
    }

    public static ArrayList<SensorBay> loadData(Context ctx) {
        ArrayList<SensorBay> returnSensorBayList = new ArrayList<>();
        SharedPreferences preferences =
PreferenceManager.getDefaultSharedPreferences(ctx);
        String listOfBays = preferences.getString("listOfSavedBays", null);
        Log.d("SAVE", "LOADING: " + listOfBays);
        ObjectMapper mapper = new ObjectMapper();

        try {
            returnSensorBayList = new
ArrayList<>(Arrays.asList(mapper.readValue(listOfBays, SensorBay[].class)));
            printToast(ctx, "Successfully loaded data", Toast.LENGTH_SHORT);
        } catch (IOException e) {
            printToast(ctx, "No saved data to load.", Toast.LENGTH_SHORT);
            e.printStackTrace();
        }

        return returnSensorBayList;
    }
}

```

7.6 - SSL

Using self-signed certificates, SSL has been implemented on the server. This is vital as the user would want to have the confidence that whatever they are sending to the server should be encrypted and safe at all times.

The disadvantage of using self-signed certificates is that it is not recognised with other CA (certificate authorities). Google has outlined necessary instructions for developers to make their apps utilise self-signed certificates. The general steps to be taken are to get the public certificate and place into the project directory, use the certificate to create a keystore and place it into the TrustManager. Google defines a TrustManager as: 'A TrustManager is what the system uses to validate certificates from the server and—by creating one from a KeyStore with one or more CAs—those will be the only CAs trusted by that TrustManager' (Android Developer, no date).

Implementing SSL on the server consisted of adding the HTTPS port in addition to the current HTTP port. This was done by editing the application.properties file, which is the file responsible for the settings for Spring Boot.

As well as that, tomcat by default only allows one connector. A connector is essentially allowing the server to listen to a specific port and parse the HTTP requests and send the responses (Sohail, no date). In order to make the server handle HTTPS requests, an additional connector was implemented. This was implemented by creating a 'Connector' object and placing it in the tomcat servlet and can be seen from the following:

```

@SpringBootApplication(scanBasePackages = {"Controller", "DAO"})
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
    private Connector createStandardConnector() {
        Connector connector = new
Connector("org.apache.coyote.http11.Http11NioProtocol");
        connector.setPort(8080);
        return connector;
    }
    @Bean
    public EmbeddedServletContainerFactory getEmbeddedServletContainerFactory() {
        TomcatEmbeddedServletContainerFactory containerFactory = new
TomcatEmbeddedServletContainerFactory();
        containerFactory.addAdditionalTomcatConnectors(createStandardConnector());
        return containerFactory;
    }
}

```

7.6.1 - Creation of keystore:

The keystore used in the server was generated through the use of using the keytool application included as part of the JRE (Java Runtime Environment). Keystores hold public and private keys. As discussed earlier in the literature review, private keys are meant to remain in the server and should not be shared. Running the following command will create a keystore:

```
keytool -genkey -alias sensor -keyalg RSA -keysize 2048 -validity 700 -keystore sensor.jks
```

This creates a keystore and further defines the algorithm used, the size of the key, as well as, the validity of the certificate. This keystore is then used in the server that is used in the 'application.properties' file.

```
server.port=8440
server.ssl.key-store-type=JKS
server.ssl.key-store=classpath:sensor.jks
server.ssl.key-store-password=123abc
server.ssl.key-alias=sensor
```


7.7 - Sensor

The following is the circuit diagram of the sensor:

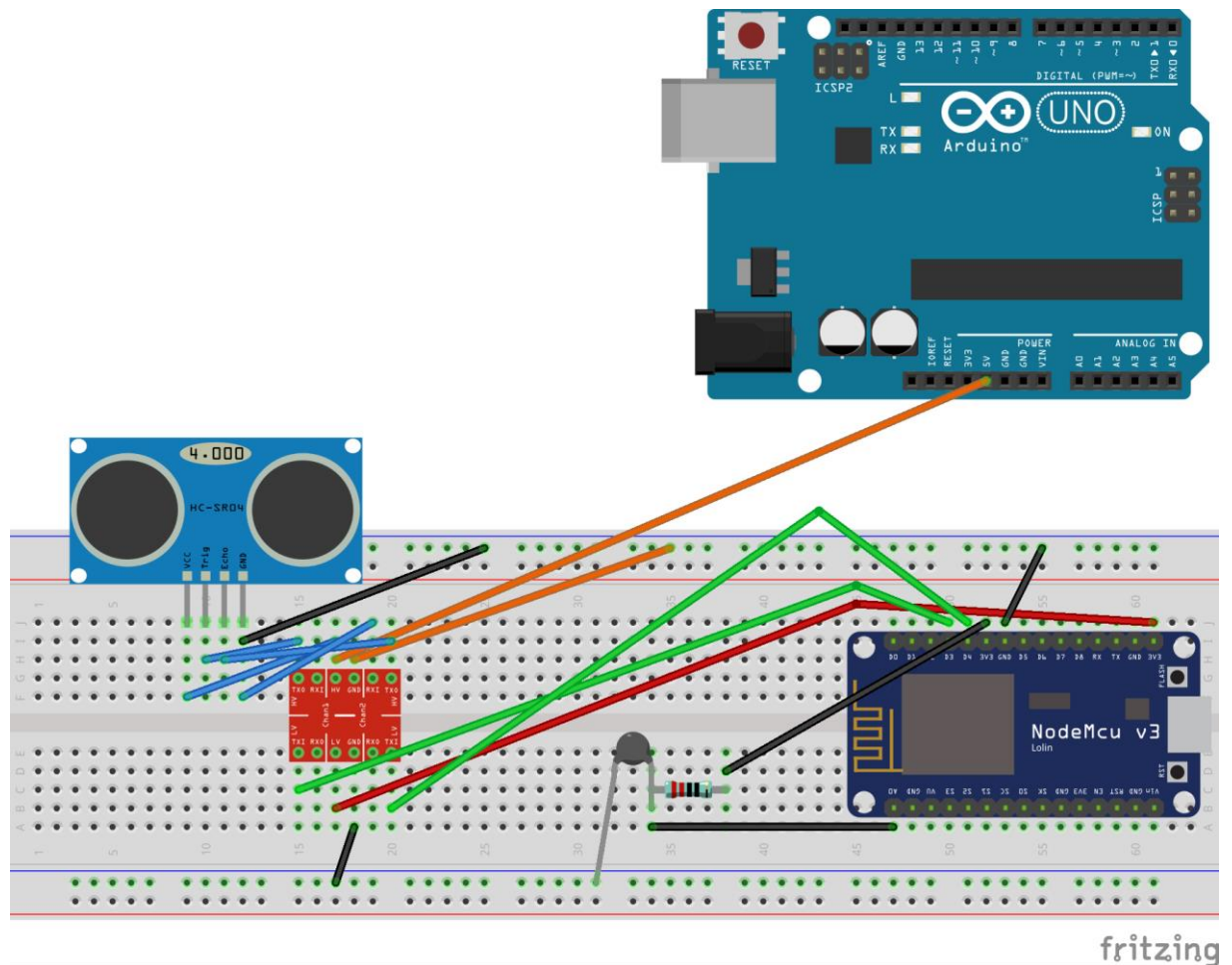


Figure 5.3 – Overview of the circuit for the sensor.

The sensor circuit comprises of a NodeMCU microcontroller and is powered by an Arduino Uno microcontroller, an ultrasonic sensor, a thermistor and a logic level converter.

The reason for choosing the NodeMCU as the main microcontroller is because it is similar to the Arduino Uno in terms of connections and features, but the main difference is that it has built-in WiFi capabilities. The sensor is coded in C using the Arduino IDE. The Arduino IDE is different to many IDEs mainly because it is specifically designed by the Arduino team, as well as, the feature to change the baud rate and view the serial monitor. In this case, the Arduino Uno is used to power the ultrasonic sensor requiring 5V that the Arduino Uno can output.

Furthermore, a logic level converter is used to safely step up and step down voltage. This is necessary as the NodeMCU can only output 3.3V but the ultrasonic requires 5V to be used. Therefore, the Arduino Uno is being used to power the module and all the connections from the ultrasonic sensor steps down from 5V to 3.3V using the logic level converter. Once stepped down, all the data from the ultrasonic is fed into the NodeMCU. Moreover, the thermistor is connected and its data is being fed to the NodeMCU.

The anatomy of an Arduino code has two core methods: `setup()` and `loop()`. The `setup` method is called when the microcontroller is turned on and this is typically the method where you setup up and initialise your variables. After the `setup()` method has finished executing, the `loop()` method will constantly run. When the microcontroller finishes executing the last line of code in the `loop()` method, it will execute the `loop()` method again. This type of structure suits my sensor requirements as there is a type of loop where it can continuously send data to the server.

The implementation of the code is split into three logical components: thermistor, ultrasonic and sending the data over to the server.

7.7.1 - Thermistor:

The thermistor is a widely used component in electronics. Thermistors typically work by lowering its resistance as the temperature rises, and it can also do the opposite. This is all dependent on the material the thermistor is constructed from. When the temperature rises, the lower its resistance becomes, hence, more current will flow through the thermistor. This is known as Ohms law which can be rearranged to get the following equation, $I = \frac{V}{R}$ (I is the current, V is voltage and R is the resistance.)

There are two main equations that revolve around thermistors: Stein-Hart equation and Beta equation. These two equations are used to interpolate the resistance vs temperature characteristic of thermistors. The following is the Steinhart equation created by John S Steinhart and Stanley R Hart in 1968 and is released in their paper, 'Calibration curves for thermistors': $T^{-1} = A + B \log R + C (\log R)^3$ where T is the temperature in Kelvin, R is the resistance and A, B and C are constants. This equation is typically used to calculate the temperature of the thermistor to a very accurate point. Whilst on the other hand, the Beta equation: $\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \ln\left(\frac{R}{R_0}\right)$ does give near accurate readings. The beta equation consists of T_0 which is room temperature, B is the coefficient of the thermistor and $\frac{R}{R_0}$ is average resistance divided by resistance at room temperature (Ada, no date). The value of the coefficient of the thermistor can be found by conducting an experiment and using the following formula:

$$B = \frac{\ln\left(\frac{R_{T1}}{R_{T2}}\right)}{\left(\frac{1}{T_1} - \frac{1}{T_2}\right)}$$

In our case, B has been calculated for us as the value came in the handbook for the thermistor.

A thorough experiment took place in order to investigate the accuracy of both equations. It was found that the B equation was the less accurate of the two (Gregg, no date). Results of the B equation are as follows:

Figure 1.

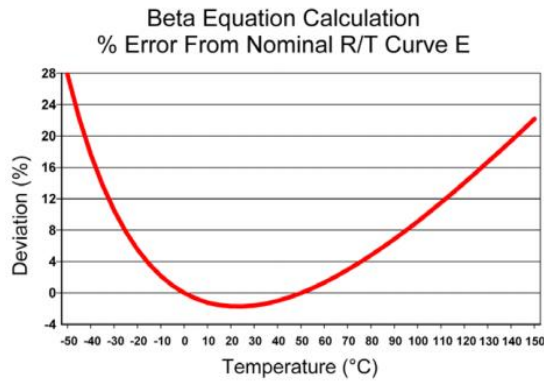


Figure 2.

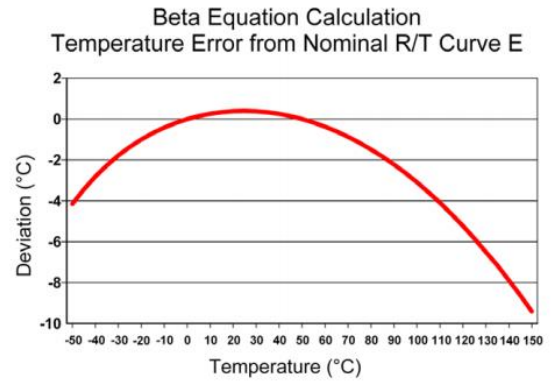


Figure 5.4 – Graph depicting the rate of error in a thermistor as the temperature increases. (Lavenuta , no date).

As the graph depicts, using the B equation gives inaccurate results depending on the temperature. From 0 – 60 degrees, the error is minimal as the deviation is less than 1 degree but as the temperature increases or decreases below 0 Celsius, the error becomes apparent.

The implementation of the thermistor in the sensor is using the B equation because, firstly, the equation does not have as much variables/constants to work out as opposed to the Steinhart's equation. Secondly, despite the B equation not as accurate as the Steinhart equation, the inaccuracy occurs in temperatures that will not be dealt with in this project therefore in this context, one can ignore the inaccuracy. Even more so, a 10K resistor, connected in series, is used to convert the reading attained from the analogue-to-digital pin and convert the number to an average resistor value over the thermistor using the following equation: $R = 10000 / (1023/ADC-1)$ (Ada, no date).

The implementation of the code can be seen from the following snippet:

```

float averageOfThermistorReadings() {
    int i;
    float average;

    for (i = 0; i < NUMSAMPLES; i++) {
        samples[i] = analogRead(THERMISTORPIN);
        delay(10);
    }
    average = 0;
    for (i = 0; i < NUMSAMPLES; i++) {
        average += samples[i];
    }
    average /= NUMSAMPLES;
    return average;
}

float steinhartConversion(float average) {
    //using B equation
    float temp;
    temp = average / THERMISTORNOMINAL;    // dividing average by the nominal
(resistance at room temp)
    temp = log(temp);                      // ln(average)
    temp = temp / BCOEFFICIENT;            // currentValue of steinhart / B
    temp = temp + 1.0 / (TEMPERATURENOMINAL + 273.15); // currentValue of steinhart +
(1/To)
    temp = 1.0 / temp;                      // Invert
    temp = temp - 273.15;                   // convert to C from kelvins
    return temp;
}

```

7.7.2 - Ultrasonic:

The calculation of the ultrasonic module is trivial. The following equation is used: $v = s/t$ ('v' is velocity, in this case the speed of sound; 's' is displacement; and 't' is time). The general idea is to time how long it took for the sound emitted from the trigger port to reach the echo port. By rearranging the above formula, we can get $s = vt$. This equation takes into account the time taken for the sound to bounce back from whatever it reflected from, hence, we will need to divide by 2 to cut the time in half. Thus, the formula becomes $s = vt/2$, where 'v' is 340 (if meters is required) or 0.034 (if cm is required).

Furthermore, many tutorials and forums have stated to set the trigger pin of the ultrasonic sensor to 'high' for 10 microseconds as it is the convention of using such components (Nedelkovski, no date). 10 microseconds is generally used as it sends out an 8 cycle sonic burst from the transmitter which then bounces from an object, back to the receiver which causes the echo pin to be in a 'high' state. We time how long the echo pin is in a 'high' state by using the `pulseIn()` method.

```
float ultrasonicCalculation() {  
  
    digitalWrite(trigPin, LOW);  
    delayMicroseconds(2);  
  
    // Sets the trigPin on HIGH state for 10 micro seconds  
    digitalWrite(trigPin, HIGH);  
    delayMicroseconds(10);  
    digitalWrite(trigPin, LOW);  
  
    // Reads the time it took for the echo pin to stay in a high state  
    return duration = pulseIn(echoPin, HIGH);  
}  
...  
distance = ultrasonicCalculation() * 0.034 / 2;
```

7.8 - Server

This project mainly revolves around two spectrums of communications: machine-to-machine and client-to-machine. Client-to-machine has been with us for a long time. An example of client-to-machine is browsing the internet. The clients (humans) are communicating with a machine (the server). Whilst this method of communication is widely recognized, the other form of communication is M2M (machine-to-machine). This type is becoming widely adopted and is mainly used in areas revolving around IoT. Quoting Carles and Mischa's definition of M2M, "M2M generally refers to information and communications technologies (ICT) able to measure, deliver, digest, and react upon information in an autonomous fashion, i.e., with no or really minimal human interaction during deployment, configuration, operation, and maintenance phases." (Carles, Chapter 1.1.1). This essentially means machines follow a set of protocols and manipulate data. In their book, 'Machine-to-Machine (M2M) Communications', they delve into more detail as to what revolves around the concept of machine-to-machine communication. One of the categories classifying m2m communication is real-time: "...real time allows making optimal and timely decisions based on a large amount of prior collected historical data. The trend is to move away from decision making based on long-term averages to decisions based on real-time or short-term averages, making a real difference to the large amount of non ergodic industrial processes" (Carles, Chapter 1.1.1). Furthermore, the other category that classifies m2m communication is 'reliability' of the data as pointed out in their book. Hence, they make noticeable to use a sensor, as well as, machine learning for optimizing the source of data.

This project essentially is split into two methods of communication: machine-to-machine (for the communication between the sensor and server) and client-to-machine (for the communication between the user/app and the server). The fact that m2m is applied to the sensor-server side further cements that this project revolves around IoT. The data is reliable as there are multiple sensors each contributing to an output of whether or not the parking bay is free or not. Furthermore, the data is fresh as the sensor sends data every few seconds,

which is near 'real-time'. Throughout this process, no human contact is needed as it is all automated. There are no set standards, formalities or 'design pattern' equivalent in the machine-to-machine realm due to m2m's infancy. However, as time progresses, there might be a framework in the future that may advance the m2m sphere (Carles, Chapter 2.2).

The communication between the server and the sensor is automated, therefore, nothing much can go wrong in terms of code implementation during runtime/code execution as the sensor's output will be following a convention/schema with no human interaction and also the server will expect an output from the sensor that conforms to a structure. Therefore, as long as the sensors' output conforms to the schema, nothing theoretically can go wrong in that concept. The following method is a request method to parse the output from the sensor:

```
@Controller
public class PostToDatabaseController {
    @Autowired
    DAOInterface db;
    @RequestMapping(value = "/posttodb", method = RequestMethod.GET)
    @ResponseBody // @ResponseBody allows you to return a string rather than a
    thymeleaf template
    public String posttodb(
        @RequestParam(value = "id", required = true) String id,
        @RequestParam(value = "longitude", required = true) String longitude,
        @RequestParam(value = "latitude", required = true) String latitude,
        @RequestParam(value = "status", required = true) String status
    ) {
        db.addGPSEntry(id, longitude, latitude, status, Constants.database,
        Constants.collection);
        return "Added the following: " + id + ". Long: " + longitude + ". Latitude:
        " + latitude + ". Status: " + status;
    }
}
```

As you can see, the http request looks for four request parameters when a request is sent to the '/posttodb' URI. The four request parameters relates to the 'id', 'longitude', 'latitude' and 'status' of the of the parking bay. After a request is made to this URI, it calls the addGPSEntry, which connects to the mongo database and stores the record there.

Furthermore, from the above code it is evident to see that dependency injection is in use through the use of the '@Autowired' annotation. Dependency injection is mainly used to loosely couple objects from each other so if a major component was to be swapped, it should not break the code.

Using Spring Boot conventions, a class can be turned into a 'spring bean' by adding the '@Component' annotation above the class file. This can be seen in the DAOImplementation class, which implements the 'DAOInterface' interface file. Once a class is annotated with 'Component', Spring knows, upon compile time, to instantiate this class and places it in the Spring container. Thus, by using the @Autowired annotation, Spring knows to look for a bean in the container of the type requested, which in this context is 'DAOInterface'. Because a 'DAOImplementation' bean was created of type 'DAOInterface', Spring is clever enough to use this bean. By using this approach, it negates the use of using the 'new' keyword, thus, making the code as independent as possible.

Another example of dependency injection is in the DAOImplementation class, the MongoClient object is being injected into the class through autowiring a MongoClient bean.

8 - Testing Strategies

This chapter will delve into the details of the testing mechanism that was used in the project.

8.1 - Unit Tests:

Throughout this project, unit tests were used to make sure the business logic of the application worked, as well as, ensuring that any modification made to the code should be able to pass the unit tests. The server has unit tests and the android application. Furthermore, the android application has a few additional testing frameworks to assert that the app is behaving as expected. One of these frameworks is called 'Instrumented Tests'.

In the android implementation, instrumented tests have been used to assert that the right activity is being shown when a certain button is clicked. Furthermore, the unit tests assert that the main functions that are used in the business logic are correct and accurate.

Throughout the project implementation, loosely coupling the code has been stressed as much as possible. By loosely coupling the code, it is easier to unit test and makes the code cleaner. It will not be prone to breaking if a component should be swapped with another component in the future, such as if a new database were to be used then the code should easily be able to integrate with the new database.

8.1.1 - Loosely coupling code

In the android implementation, there is a 'Helper' class that contains methods that are used throughout the class in the modules. One of the method that is loosely coupled is the 'getCurrentTime()' method. This method essentially gets the current time and returns a float representing how many minutes have passed from 12:00am to the current time. Upon initial inspection, one might instantiate a calendar object in the method and then conduct the relevant business logic, such as:

```
public static float getCurrentTime() {
    Calendar calandar = new Calendar();
    //... business logic that uses the calandar object
}
```

There is nothing wrong with this approach, however, it would be impossible to correctly test this method because you would not be able to test your logic on a random time. Therefore, asserting that the business logic works. The way to get around this problem is to make the object independent of the method; essentially loosely coupling the method and the object. As demonstareted from the following code:

```
private static TimeHelper timeHelper = new TimeHelperImpl();
public static void loadTime(TimeHelper impl){
    timeHelper = impl;
}
public static Calendar getInstance(){
    return timeHelper.getTiming();
}
public static float getCurrentTime() {
    Calendar calandar = HelperFunction.getInstance();
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm");
    String[] time = simpleDateFormat.format(calandar.getTime()).split(":");
    int hour = Integer.parseInt(time[0]) * 60;
    float currentTime = hour + Integer.parseInt(time[1]);
    return currentTime;
}
```

The calendar object is not strictly tied to the method. This is because the reference of the calendar object is created through a 'TimeHelperImpl' class which provides a method that returns a Calendar object. The effectiveness of this is that you can essentially create your own Calendar object and then load it into the Helper class for the getCurrentTime() method to use, which is needed to effectively unit test the business logic in the method. As can be seen from the following unit test:

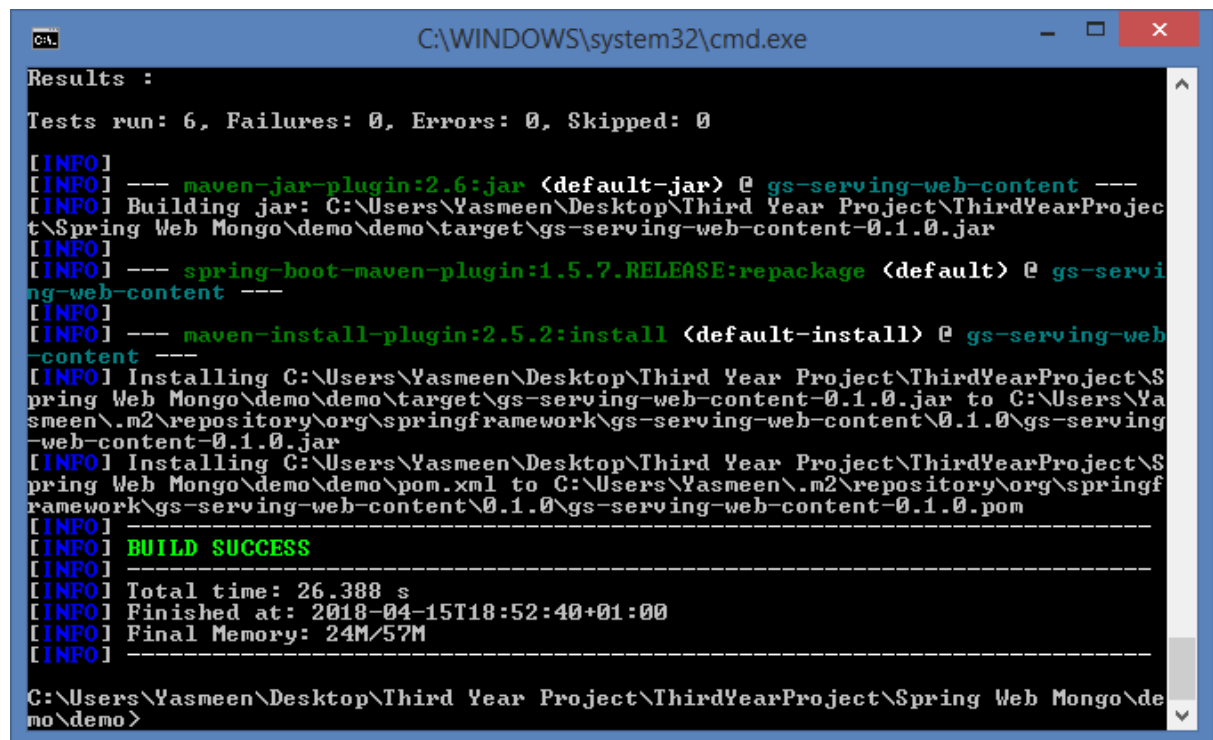
```
@Test
public void shouldReturn180FromThreeAM(){
    Calendar testCal = Calendar.getInstance();
    testCal.set(Calendar.HOUR, 03);
    testCal.set(Calendar.MINUTE, 00);
    testCal.set(Calendar.SECOND,00);
    TimeHelper mockObject = mock(TimeHelperImpl.class);
    HelperFunction.loadTime(mockObject);
    when(mockObject.getTiming()).thenReturn(testCal);
    float actualResult = HelperFunction.getCurrentTime();
    float expectedResult = 180f;
    assertTrue(actualResult == expectedResult);
}
```

A calendar object is created and is assigned a time of 03:00 am. Using mockito, a powerful testing framework, a mock object of the TimeHelperImpl class is created and whenever the getTiming() method is called, we return our custom Calendar object, which contains a time of 03:00 am. Thus, our business logic will be tested on the custom time provided.

8.1.2 - Maven testing lifecycle

Since Maven is a build tool, it is useful to continuously check that the unit tests are being passed after any changes to the code and also that the project can be executed. Maven has seven build lifecycle stages, which can be used to our advantage when developing (Maven, 2015). In terms of testing, this is very useful as it will outline and break the build once it encounters any tests that may have not passed. By writing the following in the command line, `mvn clean install`, it will clean the project, delete any generated files that may have occurred from the previous execution of the project and build the project again.

Since the 'install' phase comes after the 'test' and 'package' phase respectively, it will run the tests, and package the project and then install the project in a local repository. By following this methodology, the build will break upon encountering a failed test, therefore, a packaged solution of the project will not be created until the code/test has been rectified. By running the above command, the following output will appear if all tests have been passed:

A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays the output of a Maven build. The output starts with "Results :", followed by "Tests run: 6, Failures: 0, Errors: 0, Skipped: 0". It then shows several informational messages in green and blue text, including "Building jar: C:\Users\Yasmeen\Desktop\Third Year Project\ThirdYearProject\Spring Web Mongo\demo\demo\target\gs-serving-web-content-0.1.0.jar" and "Installing C:\Users\Yasmeen\Desktop\Third Year Project\ThirdYearProject\Spring Web Mongo\demo\demo\target\gs-serving-web-content-0.1.0.jar to C:\Users\Yasmeen\.m2\repository\org\springframework\gs-serving-web-content\0.1.0\gs-serving-web-content-0.1.0.jar". The build concludes with "BUILD SUCCESS" in green, followed by summary statistics: "Total time: 26.388 s", "Finished at: 2018-04-15T18:52:40+01:00", and "Final Memory: 24M/57M". The prompt "C:\Users\Yasmeen\Desktop\Third Year Project\ThirdYearProject\Spring Web Mongo\demo\demo>" is visible at the bottom.

```
C:\WINDOWS\system32\cmd.exe
Results :
Tests run: 6, Failures: 0, Errors: 0, Skipped: 0

[INFO] --- maven-jar-plugin:2.6:jar <default-jar> @ gs-serving-web-content ---
[INFO] Building jar: C:\Users\Yasmeen\Desktop\Third Year Project\ThirdYearProject\Spring Web Mongo\demo\demo\target\gs-serving-web-content-0.1.0.jar
[INFO] --- spring-boot-maven-plugin:1.5.7.RELEASE:repackage <default> @ gs-serving-web-content ---
[INFO] --- maven-install-plugin:2.5.2:install <default-install> @ gs-serving-web-content ---
[INFO] Installing C:\Users\Yasmeen\Desktop\Third Year Project\ThirdYearProject\Spring Web Mongo\demo\demo\target\gs-serving-web-content-0.1.0.jar to C:\Users\Yasmeen\.m2\repository\org\springframework\gs-serving-web-content\0.1.0\gs-serving-web-content-0.1.0.jar
[INFO] Installing C:\Users\Yasmeen\Desktop\Third Year Project\ThirdYearProject\Spring Web Mongo\demo\demo\target\gs-serving-web-content-0.1.0.pom to C:\Users\Yasmeen\.m2\repository\org\springframework\gs-serving-web-content\0.1.0\gs-serving-web-content-0.1.0.pom
[INFO] BUILD SUCCESS
[INFO] Total time: 26.388 s
[INFO] Finished at: 2018-04-15T18:52:40+01:00
[INFO] Final Memory: 24M/57M
C:\Users\Yasmeen\Desktop\Third Year Project\ThirdYearProject\Spring Web Mongo\demo\demo>
```

Figure 5.5 – Screenshot depicting the process of running a clean build using Maven.

8.2 - Blackbox Testing:

Blackbox testing is another category of testing. It is essentially testing the functionality of the product without looking into the implementation/structure or design on the underlying product (SoftwareTestingFundamentals, no date). It is used to test whether the functionalities and requirements of the product are working. There are many ways to document the findings from a blackbox test such as using a 'cause-effect' table. Below is a table outlining the expected and actual outcomes based on the requirements/functionalities outlined earlier in this report.

Functionality	Fulfilled
Sensor detecting changes in environment	Y
Sensor able to predict current status if no internet is available through machine learning	Y
Communication should be all under HTTPS	Y/N
App should show results in real time	Y
App should lock onto users position and see any parking bays available nearby	Y
App should provide directions to bay for usability	Y
App should let user search an area to show availabilities of car parks	Y

Table 6 – Table showing blackbox test results

From the table, the vast majority of the requirements were met. The only requirement that was partly fulfilled is the requirement revolving around HTTPS.

HTTPS was partly implemented in the server but not between the server and the sensor, server and the app and also not between the server and the database, which is explained in the following chapter.

9 - Evaluation

This section of the report will conclude my findings and explain any difficulties faced whilst implementing the code, as well as, talking about the sources and references used.

9.1 - Logistic Regression:

Initially, logistic regression was implemented on the app. This was because it was the first algorithm that was researched for this project. The algorithm uses maximum likelihood estimation (MLE) to obtain the coefficients in the equation ' $Z = b_0 + b_1 * x_1$ '. The implementation uses the stochastic gradient descent to obtain the values of the coefficients and refine the coefficients using the equation: $\text{coefficient} = \text{coefficient} + \alpha * (y - \text{prediction}) * \text{prediction} * (1 - \text{prediction}) * x$.


```

public void calculateMachineLearning(){
    for (SensorResponse parkingBay:listOfSensor) {
        parkingBay.betaOne = 0.0f;
        parkingBay.betaZero = 0.0f;

        float output = 0.0f;
        float prediction = 0.0f;

        int epoch = 0;
        float alpha = 0.0000003f;
        while(epoch < 3){
            int i = 1;
            while(i < parkingBay.timeXAxis.length){

                output = (float) parkingBay.betaZero + (parkingBay.betaOne *
parkingBay.timeXAxis[i]);
                prediction = (float) (1/(1+ Math.exp(-output)));

                //Refining Coefficients
                parkingBay.betaZero = parkingBay.betaZero + alpha *
(parkingBay.statusYAxis[i] - prediction) * prediction * (1-prediction) * 1.00f;
                parkingBay.betaOne =(float) (parkingBay.betaOne + alpha *
(parkingBay.statusYAxis[i] - prediction) * prediction * (1 - prediction) *
parkingBay.timeXAxis[i]);
                i++;
            }
            epoch++;
        }
    }
}

```

As can be seen from the implementation of the algorithm, the code is intensive to compute on a phone. To save battery life on the phone and to make sure the phone does not heat up due to performing too many calculations, the epochs had to be lowered because as the data grows, the time to compute the algorithm also grows. Furthermore, the training of the weights would add more time and complexity to it and the data structure used to incorporate this algorithm required more space/memory than to the KNN algorithm. As it can be seen in the following code snippet, the data structure used for the logistic

regression, by default, reserves two Integer arrays of size 1440: one array corresponding to time and the other array corresponding to the status of the bay.

```
//global variable in a parking bay object  
public Integer[] timeXAxis = new Integer[1440];  
public Integer[] statusYAxis = new Integer[1440];
```

Having two arrays of size 1440 instantiated upon creation of the parking bay object is intensive, compared to the dynamic data structure of the KNN algorithm in which it uses an ArrayList to store the relevant objects. As a result of using a dynamic data structure, it does not take up 1440 elements upon creation of the object.

9.2 - Timing of algorithms

In order to find out which implementation of the machine learning algorithm was more efficient practically, it was timed using Java's static method 'currentTimeMillis()' found in the System class. The results are shown below for the KNN algorithm

# of data	K	Time (ms) (avg of 5 exectutions)
1000	1	0
2000	1	6.4
4000	1	3.2
8000	1	9.4
16000	1	9.4
32000	1	9.2
64000	1	15.8
1000	10	3.2
2000	10	0
4000	10	3.2
8000	10	9.6
16000	10	6.4
32000	10	12.2
64000	10	15.6
1000	100	6.2
2000	100	6.2
4000	100	6.4
8000	100	6.2
16000	100	9.6
32000	100	12.2
64000	100	12.6

Table 7 - Results for execution time for KNN algorithm

# of data	Epoch	Time (ms) (avg of 5 exectutions)
1000	3	6.4
2000	3	9.6
4000	3	28.0
8000	3	22.4
16000	3	31.4
32000	3	37.8
64000	3	78.0
1000	10	9.6
-2000	10	12.8
4000	10	21.0
8000	10	33
16000	10	50.2
32000	10	93.8
64000	10	193.8
1000	100	34
2000	100	53

4000	100	96.4
8000	100	188
16000	100	375
32000	100	800
64000	100	1554

Table 7.1 - Results for execution time for Logistic Regression.

As it can be seen for the KNN timings, the execution time does increase relative to the size of the data and the timing does not seem to be dependent on the value of K. KNN is far more efficient compared to the logistic algorithm because the timing for the logistic algorithm depends on how many epochs are made. The reason behind this is that the logistic regression has to train the weights, hence, the looping and the intensive calculations.

Despite logistic regression not being as efficient as KNN, there is potential of using it in the app due to one probably training the weights once a day for each parking sensor rather than continuously calculating the weights. In other words, caching the weights for a set interval. Therefore, the only calculation that needs to be calculated is the probability using the cached weights.

However, as seen from the evaluations from Matlab, KNN is far more accurate and faster, hence, the reason to choose it over logistic regression.

9.2.1 Code used to test speed of KNN algorithm:

```
public static void main(String[] args) {
    final long startTime = System.currentTimeMillis();
    //initData();
    for(int i = 1; i <= timeSize; i++){
        if(Math.random() >= 0.5){
            bays.add(new Bays(i, 1));
        }else{
            bays.add(new Bays(i, 0));
        }
    }
    for(Bays bay : bays) {
        int distance = (int) Math.sqrt( (HOURTOPREDICT-
bay.hour)*(HOURTOPREDICT-bay.hour) );
        bay.distant = distance;
    }
    //sort out bays based on distance
    Collections.sort(bays);
    getKNeighbours(GETNEARESTK);
    //calculateDistance();
    final long endTime = System.currentTimeMillis();
    System.out.println("Total execution time: " + (endTime - startTime)
);
}
//Method to calculate at point
static void calculateDistance() {
    for(Bays bay : bays) {
        int distance = (int) Math.sqrt( (HOURTOPREDICT-
bay.hour)*(HOURTOPREDICT-bay.hour) );
        bay.distant = distance;
    }
    //sort out bays based on distance
    Collections.sort(bays);
    getKNeighbours(GETNEARESTK);
}
```

9.2.2 - Code used to test speed of logistic regression:

```
public static void main(String[] args) {
    final long startTime = System.currentTimeMillis();

    generateTime();
    generateStatus();

    int epoch = 0;
    float alpha = 0.0000003f;

    while (epoch < maxEpochSize) {
        int i = 0;
        while (i < time.size()) {
            //CALCULATING PREDICTION
            output = (float) (b0 + (b1 * Integer.parseInt(time.get(i))));
            prediction = (float) (1 / (1 + Math.exp(-output)));
            //System.out.println("Prediction == " + prediction);

            //REFINEMENT OF COEFFICIENTS
            b0 = b0 + alpha * (Integer.parseInt(status.get(i)) - prediction) * prediction * (1 - prediction) * 1.00f;
            b1 = (float) (b1 + alpha * (Integer.parseInt(status.get(i)) - prediction) * prediction * (1 - prediction) * Integer.parseInt(time.get(i)));

            i++;
        }
        epoch++;
        output = (float) (b0 + (b1 * 900));
        prediction = (float) (1 / (1 + Math.exp(-output)));

        final long endTime = System.currentTimeMillis();
        System.out.println("Total execution time: " + (endTime - startTime));
    }
}
```

9.3 - Sensor

Originally, this project was made solely using the Arduino Uno and not the NodeMCU microcontroller. However, that decision was changed as difficulty arose when trying to program/flash the Wi-Fi module that was supposed to be connected to the Arduino. Originally, an ESP8266 chip was used but because this component was the first generation of Wi-Fi modules available for the Arduino Uno, but it was difficult to program as it had limited capabilities, as well as few GPIO pins. Furthermore, it could not communicate with any Wi-Fi access points that had security enabled, which is a major flaw. This was the reason why a NodeMCU microcontroller was used as it had almost the same capabilities as the Arduino Uno but with integrated Wifi. Although the NodeMCU having Wi-Fi integrated inside it, it could not access the University's Wi-Fi access point. The reason being is that the University Wi-Fi's access point is an enterprise access point that the NodeMCU is not capable of connecting due to security features.

Furthermore, the sensor originally had a GPS module connected to it. This was going to be used to dynamically get the longitude and latitude of the sensor rather than hardcoding the coordinates. This was later dropped as the GPS module was not very efficient in terms of functionality. The GPS module needed to be placed outside and the sky must be clear.

9.4 - SSL

SSL was not fully implemented between all communications to the server. The issue with implementing it was that there was not enough time to implement SSL between the sensor, database and app. Effort was made to implement SSL between the server and the app but a few issues arose, which is explained as follows.

Firstly, when using the Volley framework to send out HTTPS requests, you need to place your public certificate associated with the server's keystore. This was done using the following method:

//Volley uses the following in method when creating a Request Queue object

```
private SSLSocketFactory getSocketFactory() {
    CertificateFactory cf = null;
    try {
        cf = CertificateFactory.getInstance("X.509");
        InputStream caInput = getResources().openRawResource(R.raw.publiccert);
        Certificate ca;
        try {
            ca = cf.generateCertificate(caInput);
        } finally {
            caInput.close();
        }
        String keyStoreType = KeyStore.getDefaultType();
        KeyStore keyStore = KeyStore.getInstance(keyStoreType);
        keyStore.load(null, null);
        keyStore.setCertificateEntry("ca", ca);
        String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
        TrustManagerFactory tmf =
TrustManagerFactory.getInstance(tmfAlgorithm);
        tmf.init(keyStore);
        HostnameVerifier hostnameVerifier = new HostnameVerifier() {
            @Override
            public boolean verify(String hostname, SSLSession session) {
                return hostname.compareTo("192.168.43.49")==0;
            }
        };
        HttpURLConnection.setDefaultHostnameVerifier(hostnameVerifier);
        SSLContext context = null;
        context = SSLContext.getInstance("TLS");

        context.init(null, tmf.getTrustManagers(), null);

        HttpURLConnection.setDefaultSSLSocketFactory(context.getSocketFactory());
        SSLSocketFactory sf = context.getSocketFactory();

        return sf;

    } catch (CertificateException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (KeyStoreException e) {
        e.printStackTrace();
    } catch (FileNotFoundException e) {

```

```

        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (KeyManagementException e) {
        e.printStackTrace();
    }

    return null;
}

```

Essentially what this method¹ does is get the certificate and places it in the default keystore the Android OS uses. This approach worked although there was a bug that led to many sockets being opened in the background. Upon executing the app, around a hundred sockets would open up within a minute of using the app. As a result of this, the map would not update and a grey background would appear, therefore rendering this whole app useless. It was decided to use another approach rather than using the Volley framework for sending HTTPS requests.

The second approach was to make the HTTPS request using the `HttpsURLConnection` object. This yet involved in reading the certificate and placing it in a custom keystore in a 'bouncy castle' format (TransOceanic, no date)²³. This method did not work as errors occurred due to the conversion from a certificate exported from a jks keystore to a bks (bouncycastle) format despite tools such as 'Portecle' being used.

Overall, it is possible to communicate with the server through the HTTPS protocol but only through a desktop browser.

¹

<https://github.com/tcho16/ThirdYearProject/blob/b59d1d8000eb5a8ed5379ff11abfae191c53260e/Android/app/src/main/java/com/example/tarikh/myapplication/ParkingBayMain.java>

² <http://transoceanic.blogspot.co.uk/2011/11/android-import-ssl-certificate-and-use.html>

³ <https://stackoverflow.com/questions/1217141/self-signed-ssl-acceptance-on-android>

9.5 - Usability and impressions

After developing the app, a questionnaire was conducted regarding the functionality and the usability of the app. There are many psychological aspects when it comes to writing a questionnaire such as asking a close ended question rather than an open ended one because respondents are more likely to skip the latter (Chase, 2007).

The following depicts the result of the questionnaire that was conducted by 10 people:

In terms of the layout, 70% had said the layout of the app was 'good' (a rating of 4) whilst three people stated the app was 'okay' (a rating of 3). In terms of functionality, usability and performance, all respondents rated it 4 or more. Furthermore, nine people stated that they did not think about the data being collected whilst using the app and ten stated that they did not think about how the data is stored.

From the results, it can be seen that the purpose of the project has been fulfilled thoroughly as the majority of respondents rated the app 4 or more on the questionnaire, although the vast majority did not think about how the data was treated. This is alarming because it shows the level of trust the user has in the app in terms of storing/using its data, which further cements the fact that SSL is needed throughout all communications.

9.6 - Future prospects

There are many areas that can make this prototype into a well-established device in the IoT sphere and for it to be used commercially. One improvement revolves around the components used to build the sensor. In this project, the quality of the components was fairly standard and readily available due to how cheap they were. Because of how cheap they were, they were not up to the commercial standards. For example, the Wi-Fi signal not being strong enough to access a point or the sensor not waterproof. One way of overcoming this method is to buy sturdy and strong components that are capable of fulfilling their functionality in the external environment.

Furthermore, the sensor would need to be enclosed in a waterproof case to withstand extreme weather conditions, such as rain and snow. But even more so, the design of the sensor should be built in a way that would be able to cater more parking bays per sensor. For example, with the current structure of the sensor outlined throughout this project it will only be able to cater for one parking bay, but if the sensor was enclosed in a cubical structure then it would theoretically be able to take care of potentially five parking bays, hence, being more cost efficient.

As well as that, more components could be added, such as a light dependent resistor to make the readings even more accurate. But even more so, utilise the Stein-Hart equation for a more accurate result in the thermistor reading.

Another future prospect would be to make this sensor fully commercial. In order to do that, I would have to have the databases clustered as this would make the architecture of the design more scalable so that it will be able to handle the immense amount of queries. Furthermore, SSL would have to be implemented on every aspect of communication; this means the communication from the sensor to the server and the server to the database must all be under the SSL protocol. This is not only for the sake of security, but also to keep peace in the mind for the stakeholders and clients.

Furthermore, if more time was permitted, a website would be developed. It would offer the same functionality as the app but this would expand to more users as currently the app is android exclusive. Therefore, people who use iOS,

windows or an android OS not supported by the app will benefit from the website. This would lead to an increase of users therefore it is important that the tech stack change to something more suitable for such demands. Tools such as Spring Integration would be used as this is an enterprise framework, hence, taking full advantage of enterprise design patterns. As well as using enterprise frameworks, AWS (amazon web services) would be used to ensure it is fully scalable to cope with the demand.

9.7 - Tackling the problem statement

This whole project revolved around tackling an ongoing issue drivers face on a daily basis: looking for parking bays. The solution outlined in this project does provide the necessary tools to fix this issue as it fulfils one of the core criteria in tackling this problem, which is acquiring real time data. By gaining real time data and delivering it to the user, the user is able to see what bays are available, hence, eradicating the need for a driver to continuously drive looking for a parking bay. Furthermore, the solution is presented in an app, therefore, it is necessary for it to be user friendly. By looking at competitor's apps and services that try and tackle the situation, this app was made by gathering their best features and incorporating it into one.

Even more, some of the resources used in this project have come from people who work in established companies relevant to their field of work, such as British Parking Association. As well as that, reaching out to competitors and asking them questions was also quite useful and was a good approach in getting information direct from the companies. Moreover, other resources were found in books, websites and in published papers. The RAC foundation, which contained the data about all the parking times, was very useful as it was used to aid the choice of what machine learning algorithm to implement. Despite it having a cornucopias amount of data, the data from the report used was outdated as it dates back to 2012.

Whilst this solution does use a sensor, as other solutions have outlined this approach may be costly and other potential solutions exist, such as using imagery and robotics. If more time was given, I would design and implement my changes and turn this prototype into a fully-fledged commercial product as there are certain characteristics that could not have been implemented due to time, resources and cost. Furthermore, being able to use queuing theory in the implementation would significantly help as it would give more information regarding how the parking bays are used and eventually incorporate the theory with machine learning algorithm.

Finally, as outlined in the introduction, taking the legal aspects of data sharing and data storing lightly can have dire consequences. As this prototype moves

to being a commercial product, the consideration of current laws regarding data would need to be taken into account, as well as, the social ethics of the sensor. Also from a data controller's point of view, how the data will be stored, and used, will also change as the customer base grows. This is due to the fact that customers will request more features of the product and in order to implement the features, it will inevitably involve around collecting users information and storing it (i.e. car brand, car model) therefore, a huge emphasis will be involved around the Data Protection Act 1998. Also, just how windmills are known to obscure the scenery (Scientific American, no date), the sensor might degrade the aesthetics of the pavement. Moreover, in terms of data, as the product grows so will the customer base. This is why it is essential to be as transparent as possible to the customers on how their data will be handled.

References

- Achak, M (2014). Understanding Erlang and Queuing Theory. Fcr. Available at <http://www.gofcr.com/understanding-erlang-and-queuing-theory/> [Accessed September 2017]
- Ada, L (No Date). Adafruit. Available From <https://learn.adafruit.com/thermistor/using-a-thermistor> [Accessed August 2017]
- Albertson J. (2017). Put it in park with new features in Google Maps. Google. Available From <https://www.blog.google/products/maps/put-it-park-new-features-google-maps/> [Accessed August 2017]
- Android: Import SSL certificate and use it to SSL connection (2011). Step by Step. Available From <http://transoceanic.blogspot.co.uk/2011/11/android-import-ssl-certificate-and-use.html> [Accessed March 2018]
- Anon (no date). Shattered. Shattered. Available at <https://shattered.io/> [Accessed March 2018].
- Anon, 2016. Cars on England's roads increase by almost 600,000 in a year. BBC News. Available at: <http://www.bbc.co.uk/news/uk-england-35312562> [Accessed January 7, 2018].
- AppyParking. (No Date). AppyParking homepage. Available From <http://www.appyparking.com/rta.html> [Accessed August 2017].
- Arduino Documentation. (No Date). Analog Input. Available From <https://www.arduino.cc/en/Tutorial/AnalogInput> [Accessed September 2017]
- Bates J and Leibling D. (2012). Spaced Out Perspectives on parking policy. RAC Foundation. Available From https://www.racfoundation.org/wp-content/uploads/2017/11/spaced_out-bates_leibling-jul12.pdf [Accessed December 2018]
- Black Box Testing. (No Date). Software Testing Fundamentals. Available From <http://softwaretestingfundamentals.com/black-box-testing/> [Accessed January 2018]
- Bronstein A. (2017). A Quick Introduction to K-Nearest Neighbors Algorithm. Medium. Available From <https://medium.com/@adi.bronstein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7> [Accessed February 2018]
- Brownlee J (2016). Logistic Regression for Machine Learning. Machine Learning Mastery. Available From <https://machinelearningmastery.com/logistic-regression-for-machine-learning/> [Accessed January 2018].
- Brownlee J. (2016) Logistic Regression Tutorial for Machine Learning. Machine Learning Mastery. Available From <https://machinelearningmastery.com/logistic-regression-tutorial-for-machine-learning/> [Accessed January 2018].
- Carles Anton-Haro (2011). Machine-to-machine (M2M) communications : architecture, performance and applications. Cambridge, England ; Waltham, Massachusetts ; Oxford, England. Woodhead Publishing.

- Chouhan, T. (2018). Question about an innovation to help car parks. [email]. Sent to Kathryn Griffith, 12 January.
- Cisco. (No Date). Introduction to Secure Sockets Layer. Available From <http://euro.ecom.cmu.edu/resources/elibrary/epay/SSL.pdf> [Accessed From April 2018]
- Driessen, V. (2010). A successful git branching model. [image]. Available from <http://nvie.com/posts/a-successful-git-branching-model/> [Accessed September 2017]
- eTutorials. (No Date). Asymmetric Encryption Explained. Available From <http://etutorials.org/Programming/Programming+.net+security/Part+III+.NET+Cryptogra phy/Chapter+15.+Asymmetric+Encryption/15.1+Asymmetric+Encryption+Explained/> [Accessed From March 2018]
- Garret, O., 2017. 10 Million Self-Driving Cars Will Hit The Road By 2020 -- Here's How To Profit. Forbes. Available at: <https://www.forbes.com/sites/oliviergarret/2017/03/03/10-million-self-driving-cars-will-hit-the-road-by-2020-heres-how-to-profit/#3c617ee37e50> [Accessed April 7, 2018].
- Granger P. (2017). Chrome 62 Means Time to Get Secure and Improve Your SEO and Trust. October 2017. Available From <https://www.globalsign.com/en/blog/chrome-62-update/> [Accessed S
- Hamada R.H.A, Patrick S, Justin D.D.D, Yap V.V. (2010). Vision-Based Automated Parking System. 10th International Conference on Information sciences, signal processing and their applications. Malaysia. 10-13 May 2010. 757-760.
- Harrison C. (2007). Harvard University Program on Survey Research. Available From https://psr.iq.harvard.edu/files/psr/files/PSRQuestionnaireTipSheet_0.pdf [Accessed March 2018]
- Kaliski. (No Date). The Mathematics of the RSA Public-Key Cryptosystem. Available From <http://www.mathaware.org/mam/06/Kaliski.pdf> [Accessed March 2018]
- Kaluža B. (2016). Machine Learning in Java. Packt Publishing. Available From <http://proquest.safaribooksonline.com.ezproxy.westminster.ac.uk/book/programming/machine-learning/9781784396589> [Accessed From December 2017]
- Konig M, Neumayr L. (2016). Users' resistance towards radical innovations: The case of the self-driving car. Elsevier. Available From <https://www.journals.elsevier.com/transportation-research-part-f-traffic-psychology-and-behaviour> [Accessed Jan 2018]
- Lavenuta G (No Date). AN EXPLANATION OF THE BETA AND STEINHART-HART EQUATIONS FOR REPRESENTING THE RESISTANCE VS. TEMPERATURE RELATIONSHIP IN NTC THERMISTOR MATERIALS. QTI Sensing Solutions. Available From <https://www.thermistor.com/sites/default/files/specsheets/Beta-vs-Steinhart-Hart-Equations.pdf> [Accessed September 2017]
- Lee D. (2018). CES 2018: Byton unveils futuristic 'truly smart' car. Available From <http://www.bbc.co.uk/news/technology-42599345> [Accessed From January 2018]

Mathew G. (2013). Drawing driving route directions between two locations using Google Directions in Google Map Android API V2. Knowledge by Experience. Available From <http://wptrafficanalyzer.in/blog/drawing-driving-route-directions-between-two-locations-using-google-directions-in-google-map-android-api-v2/> [Accessed January 2018]

Moore, C. 2017. OnePlus OxygenOS built-in analytics. Chris's Security and Tech Blog. Available From <https://www.chrisdcmoore.co.uk/post/oneplus-analytics/>. [Accessed February 2018].

Morbin, T. (2017). Anti-virus collects data without user permissions & uses commercially. SC Magazine UK. Available From <https://www.scmagazineuk.com/anti-virus-collects-data-without-user-permissions-uses-commercially/article/690001/> [Accessed February 2018].

Newman L. (2017). A SUPER-COMMON CRYPTO TOOL TURNS OUT TO BE SUPER-INSECURE. Available From <https://www.wired.com/2017/02/common-cryptographic-tool-turns-majorly-insecure/> [Accessed April 2018].

Petsch K, Dotzlauf P, Daubenspeck C, Duthie N and Dr. Mock A. (2012). Automated Parking Space Locator: RSM. ASEE North Central Section Conference. Ohio Northern University. 24 March 2012. 1-13

Popper B. (2017). Google announces over 2 billion monthly active devices on Android. The Verge. Available From <https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users> [Accessed December 2017]

Prodromou A. (2017). TLS/SSL Explained: TLS/SSL Terminology and Basics. May 2017. Available From <https://dzone.com/articles/tlsssl-terminology-and-basics> [Accessed April 2018].

RAC Foundation (2017). Local Authority Parking Operations Revenue Outturn for England. RAC Foundation. Available From https://www.racfoundation.org/assets/rac_foundation/content/downloadables/Local_Authority_Parking_Operations_Revenue_Outturn_for_England_2016-17_listed_by_size_of_surplus_and_alphabetically.pdf. [Accessed November 2017].

Rashid M. M, Musa A, Rahman A.M and Farahana N, Farhana A. (2012). Automatic Parking Management System and Parking Fee Collection Based on Number Plate Recognition. International Journal of Machine Learning and Computing. Vol. 2, No. 2. 93-98.

Robert M, Promet R and Drago P. (2015). APPLICABILITY OF INFORMATION TECHNOLOGIES IN PARKING AREA CAPACITY OPTIMIZATION. Computational Logistics: 6th International Conference, ICCL 2015. Netherlands. 23-25 September 2015. 143-151.

Sánchez F. (2013). Self-signed SSL acceptance on Android [Forum Post]. StackOverflow. Available From <https://stackoverflow.com/questions/1217141/self-signed-ssl-acceptance-on-android> [Accessed March 2018]

Schulz J. (2017). Our Newest Innovation: Ultrasonic Sensor Parking Availability Technology. Inrix. Available From <http://inrix.com/blog/2017/12/ultrasonic-sensor-parking-availability-technology/> [Accessed December 2017]

Security with HTTPS and SSL. (No Date). Android Development Guide. Available From <https://developer.android.com/training/articles/security-ssl.html> [Accessed March 2018]

Sohail F. (2014). Understanding the Tomcat NIO Connector and How to Configure It. DZone. Available From <https://dzone.com/articles/understanding-tomcat-nio> [Accessed March 2018]

SuperDataScience (2017). Logistic Regression Tutorial. YouTube. Available from <https://www.youtube.com/watch?v=nz-FrbAa8dY> [Accessed on October 2017].

Sztrik J. (2011). Basic Queuing Theory. University of Debrecen. Available At http://www.academia.edu/download/39151797/SOR_Main_Angol.pdf [Accessed October 2017]

UFLDL Stanford. (No Date). Optimization: Stochastic Gradient Descent. Available From <http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/> [Accessed On January 2018].

Understanding POJOs. (No Date). Spring. Available From <https://spring.io/understanding/POJO>. [Accessed September 2017]

Understanding Tomcat Connectors. (No Date). MuleSoft. Available From <https://www.mulesoft.com/tcat/tomcat-connectors>. [Accessed March 2017]

Weiss B. (2018). Trump-linked firm Cambridge Analytica collected personal information from 50 million Facebook users without permission. 17 March 2018. Available From <http://uk.businessinsider.com/cambridge-analytica-trump-firm-facebook-data-50-million-users-2018-3> [Accessed March 2018].

Whalley A. (2016). SHA-1 Certificates in Chrome. Available From <https://security.googleblog.com/2016/11/sha-1-certificates-in-chrome.html> [Accessed April 2018]

What is Agile? What is Scrum? (No Date). CPrime. Available From <https://www.cprime.com/resources/what-is-agile-what-is-scrum/> [Accessed September 2017]

Why is Nearest Neighbor a Lazy Algorithm? (No Date). Available From <https://sebastianraschka.com/faq/docs/lazy-knn.html> [Accessed February 2018]

Wiring. (No Date). Breadboard Overview. [image] Available From <http://wiring.org.co/learning/tutorials/breadboard/> [Accessed September 2017]

Zulkifli H. (No Date). Understanding Learning Rates and How It Improves Performance in Deep Learning. Towards Data Science. Available From <https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10> [Accessed February 2018]

Bibliography

Ametherm. (No Date). NTC Thermistor - Calculate The Temperature Coefficient. Available at: <https://www.ametherm.com/thermistor/ntc-thermistor-calculating-the-temperature-coefficient-of-a-thermistor> [Accessed September 2017].

Amit K Pandey. (2017). Queuing Theory Explained. YouTube. Available from <https://www.youtube.com/watch?v=xO0YWDCSpqk> [Accessed on October 2017].

Anon, (no date) Waiting Line Models. California State University, Sacramento. Available From <http://www.csus.edu/indiv/b/blakeh/mgmt/documents/opm101supplc.pdf> [Accessed 2018]

Eric Jesse. (2015). Queue Modeling Basics. YouTube. Available from <https://www.youtube.com/watch?v=Wgkcrtjrr7s> [Accessed on October 2017].

Hellström, H. (2013). What makes RSA secure by using prime numbers? [forum post] Crypto StackExchange. Available from <https://crypto.stackexchange.com/questions/10590/what-makes-rsa-secure-by-using-prime-numbers> [Accessed March 2018].

Matematicainformatica. (2018). Volley Android HTTPS self signed certificate. Math Info e Varie. Available at: <http://matematicainformatica.altervista.org/volley-android-https-self-signed-certificate/> [Accessed March. 2018]

Noureddin Sadawi. (2014). How K-Nearest Neighbors (kNN) Classifier Works. YouTube. Available from https://www.youtube.com/watch?v=v6278Cjf_qA [Accessed on January 2018].

plusAdmin (1997). Agner Krarup Erlang (1878 – 1929). Plus Magazine. Available at <https://plus.maths.org/content/agner-krarup-erlang-1878-1929> [Accessed September 2017].

Tarikh Chouhan, IPR (2018).

Thales Sehn Korting. (2014). How kNN algorithm works. YouTube. Available from <https://www.youtube.com/watch?v=UqYde-LULfs> [Accessed on January 2018]

Weisstein, Eric W. (no date). RSA Number -- from Wolfram MathWorld. Available at: <http://mathworld.wolfram.com/RSANumber.html> [Accessed March. 2018].

What is a breadboard? (No Date). Wiring. Available From <http://wiring.org.co/learning/tutorials/breadboard/> [Accessed Sepetember 2017]

What is Decision table in software testing? (No Date). ISTQB Exam Certification. Available From <http://istqbexamcertification.com/what-is-decision-table-in-software-testing/> [Accessed March 2018]

Appendices

Questionnaire created to gain insight of the app:

Parking Bay Questionnaire – functionality and usability

Ranging from 1 to 5, 1 being bad and 5 being excellent, rate the following:

1) Layout of the app

2) Functionality of the app

3) Performance of the app

4) Usability of the app

Whilst using the app, did you...

1) Wonder how data is being collected whilst using this app? YES/NO

2) Think about how safe the data is being stored? Yes/No
