6CCS3PRJ Final Year Individual Project — WiFi Positioning Systems

Final Project Report

Author: Baldeep Gill

Supervisor: Andrew Coles

Student ID: 21034937

April 16, 2024

Abstract

GPS is usually used for navigation at the street and building level and has revolutionised how we travel. However, these same signals are not accurate or reliable enough to provide navigation and other utilities at the room level. In order to provide a more accurate and robust system, alternative signals are reviewed and evaluated as an alternative to GPS. A proof-of-concept positioning system that uses WiFi access points is then specified and implemented, making use of machine learning classification algorithms in an unconventional manner. This positioning system is then evaluated for its accuracy and how well it meets the targets that have been set.

Originality Avowal

Acknowledgements

I would like to give a special thanks to my supervisor, Dr. Andrew Coles, for his invaluable guidance and support throughout the project. I could not have completed this report without his help.

Contents

Chapter 1

Introduction

Due to the increased adoption of smart device across the world, the demand for the digitisation of certain tasks or processes has increased. One of the most notable examples of this can be seen in navigation: before the gradual introduction of GPS on mobile devices in the mid to late 2000s, the main method of navigation was via large physical maps and reading of street signs. There is a similar navigational challenge for indoor locations, too; large buildings with complex layouts can be confusing and hard to navigate through, however, unlike at the street level, there is usually a lack of maps, signage, or other utilities available to the user. Over time, positioning technologies have been refined to the point that it is commonplace to open up an app on your phone and have your location accurate to within 5 metres [11][27], whereas, there is no common digital utility for providing navigation or location for an indoor environment. Navigation through an especially complex building layout can be vastly improved by providing maps or floor plans [22], therefore, just like paper maps were largely replaced by GPS navigation, there is merit in considering a mobile application that allows convenient access to these maps and provides routing between locations.

In order to get anywhere near such a product, however, the problem of how to get the user's location needs to be solved. GPS signals are not reliable enough indoors and do not provide enough accuracy at the room level. Therefore, alternative positioning systems need to be considered.

The potential applications of indoor-focused positioning systems consist of a wide range of uses. As just mentioned, an Indoor Positioning System (IPS) can be used to solve the difficulties of navigating through a large building; a survey of the University of Genoa Hospital showed that both staff and visitors faced challenges in navigating through the building [3]. IPSs are very versatile and can be used for more than navigation: they can be used for asset tracking such as library books [17], the can be used to provide interactive museum exhibits [26], or they can be used to track user shopping habits [18].

1.1 Project Aims

The objective of this project is to evaluate the available alternatives to GPS based positioning, particularly those alternatives that make use of WiFi access points. A proof-of-concept application will be specified, designed and implemented to demonstrate how such a positioning system would work in a realistic environment and, perhaps, show how some of the applications listed above could be achieved. Several experiments will also be conducted on the positioning system to act as further demonstration of the positioning system and the feasibility of using such positioning systems in a commercial setting will be evaluated.

1.2 Report Structure

The report will first begin with a review of the background surrounding positioning systems as a whole. We will discuss alternatives to GPS and how some of those alternatives work in detail. Examples of existing work will also be discussed and evaluated to determine the best system to implement.

Having chosen the sort of positioning system to implement, application requirements and a specification will be formalised so that clear targets for the app are set. Following this, a design chapter will discuss how the code will be laid out as well as specifying a plan on how the application will ultimately take form.

Next, an implementation chapter covers the chronological process of creating the app. This chapter will include detail on how critical modules of code were implemented, as well as any issues faced during development.

Finally, evaluation and conclusion chapters will determine how well the positioning system functions at its role of providing an accurate location estimate. The software as a whole will also be evaluated against the requirements that were set to see where we succeeded and where improvements can be made.

Chapter 2

Background

2.1 Positioning Systems

The classic positioning technique people think about would be, of course, GPS (Global Positioning System). GPS makes use of 31 satellites in Earth orbit to determine the position of a GPS device. It does this by calculating the distance between itself and at least four GPS satellites using the speed of light and the time delay taken for the radio signal to reach the device. Once these four distances have been established, the device can use trilateration to determine its position up to 5 metres [11].

Despite the widespread use of GPS for accurate positioning, there is one main drawback in considering it's use in indoor positioning; the signal can be greatly degraded by several factors. These are: multipathing caused by signal reflections; obstructions blocking the line-of-sight to a satellite; or high levels of non-line-of-sight propagation [5]. These effects greatly impact the accuracy of the reported position to 15 – 20 metres and, in some cases, as much as 40 metres [16]. Therefore, in order to provide an accurate location service indoors, alternative methods need to be employed, of which, there are several.

2.1.1 Infrared (IR)

Positioning systems using the infrared portion of the electromagnetic spectrum are common and can provide high levels of accuracy. IR based systems, however, are only suitable at short ranges and only when there is direct line-of-sight since physical obstructions or interference from sunlight or fluorescent light can cause degradation in signal [6].

Traditional IR techniques involve the use of transmitters or tags placed around a target area which are uniquely labelled to help with determining location. There are many different implementations of this methodology: one such example is the Active Badge [29] system, which makes use of tags carried by office staff and receivers placed around the workplace to track the locations of staff. An extension to this can be seen with the proposed Infrared Indoor Scout [1], which used cameras fitted in the ceiling to cover a larger area (specifically a lecture hall) to locate the positions of infrared emitting tags. This paper demonstrates a decrease in accuracy compared to examples such as Active Badge as the coverage area is increased.

IR positioning systems boast extreme precision, however, their feasibility relies on numerous factors, the major one being the expected amount of obstruction between IR emitters and receivers. As mentioned, IR technologies are only feasible in direct line-of-sight situations, rendering it useless if any obstruction is to be expected. Another drawback of the technology is the cost of hardware implementation; despite IR tags being cheap, the cost to retrofit and wire receivers into existing infrastructure, along with the number of tags required in order to achieve good coverage of a large or tall space, and requiring staff or visitors to carry and ensure that a tag is always visible, makes utilising IR technology difficult [13].

2.1.2 WiFi

Just like IR, WiFi positioning systems require beacons to be placed in an environment in order to provide location data, however, WiFi based positioning systems benefit from the abundance of preexisting access points (APs) already integrated into public spaces; installing additional hardware is often unnecessary, reducing the overall cost of implementing a positioning system. The majority of the general public also now carry a WiFi-enabled device, requiring next to no extra effort for a potential user to leverage a WiFi positioning system. WiFi based systems are often less accurate compared to systems based on other signals but, despite this, they are the only ones that can still provide reasonable accuracy despite degradation from obstructions or non line-of-sight propagation.

There are several different ways that WiFi signals can be used to implement a positioning system. The most common method is one based on fingerprints where an environment is scanned and the signal strengths of wireless access points is recorded at multiple determined locations. In order to locate a device, a scan is taken and compared to the database of recorded scans and an algorithm is used to estimate the closest match(es) and, therefore, the location that the scan was taken. Fingerprinting is a relatively simple method to implement, however, the accuracy of the system is dependent on the number of locations scans were taken at and the number of scans taken at each position, resulting in a time-consuming and tedious process before any system can be implemented.

There are alternative techniques, however, that completely forgo the need for manual mapping of an environment, namely: Time of Flight (ToF) and Angle of Arrival (AoA). Time of Flight methods such as the one presented by Schauer *et al.* in [23] use the time it takes for a radio signal to travel the distance to a device to calculate a position. This is the same concept used by GPS, however, employing this method requires precise synchronisation of clocks, something that wireless access points do not facilitate. In order to overcome this, as Schauer *et al.* suggests, the round trip delay is used instead to calculate the time taken.

Angle of Arrival techniques, such as *ArrayTrack* presented by Xiong and Jamieson in [30], measure the angle of incidence when communicating with wireless access points and utilise triangulation techniques in order to locate a device. While this technique proves to be highly accurate, the accuracy heavily relies on the number of physical antennas present on access points; *ArrayTrack* presents an implementation of AoA with a median accuracy of 23 centimetres, however, used FPGA-based access points with a higher number of antennas than what is typically found. Recreating such results in a public area would require the retrofitting of access points in that area, a long and usually costly process. Another consideration of this technique is the hardware required by any potential user; *ArrayTrack* uses specialist network hardware with additional antennas as client devices, with hardware specifications far exceeding those found in a regular smartphone, making it difficult to replicate in a commercial setting.

2.1.3 Bluetooth

Bluetooth is a wireless standard that has become ubiquitous on personal devices such as phones or laptops, making it another promising candidate for use in an indoor positioning system. Bluetooth can be considered as a lower-power version of WiFi; it occupies the same 2.4 GHz frequency band as WiFi does but only works at a smaller range, up to 15 metres for Bluetooth compared to a range of up to 100 metres for WiFi [19]. Since Bluetooth works similarly to WiFi, however, implementations of positioning systems are also similar; Bai *et al.* present a system that could be used to track the location of the elderly using Raspberry Pis and Bluetooth beacons using fingerprinting and trilateration techniques [4], for example. Time of Flight techniques as described in the WiFi section are difficult to implement due to a lack of precise time measurement and synchronisation opportunities in the Bluetooth standard. The same can be said for Angle of Arrival based methods; the arrays of directional antennas needed for such methods are rarely present for use in Bluetooth devices[28].

Another consideration with Bluetooth-based positioning is, once again, the requirement of placing beacons in a target environment. While such technology is relatively cheap, the lower range of Bluetooth compared to WiFi ends up being a limiting factor to the size of an area that can be reasonably mapped using Bluetooth. The number of beacons that will have to be placed to cover the same area will be far higher than the number required for WiFi, plus, trying to cover large open areas might result in poor location accuracy due to the short range of Bluetooth. Finally, due to sharing the same frequency band as WiFi and its relatively low signal strength, Bluetooth may be susceptible to interference, making choosing Bluetooth in high traffic areas unwise.

Based on the information presented, making use of WiFi in an indoor positioning system would be the ideal choice considering setup costs, hardware, implementation and ease of use to a potential user. Using WiFi would eliminate any cost in buying and setting up hardware; there is an abundance of WiFi Access Points in public areas and spaces, and all of these can be used to aid the positioning process. If the system is intended to be used by, for example, a visitor to a building, then the chosen method of implementing the positioning system should ensure that access to the system is as simple as possible. WiFi-based systems lend themselves very well to this specific use case, with the majority of people having access to WiFi-enabled smart devices, there should be no need to organise any extra hardware in order for users to access the system. The only limitation to targeting user devices is the potential for different accuracies due to differences in phone hardware. Zandbergen found a negligible difference in the location accuracy on two different devices [31], however this can be assessed further during implementation of a prototype.

2.2 Time of Flight-based Positioning

Time of Flight systems are based on the same principles as GPS. A receiver device calculates the distance between itself and emitter base stations to determine the location of the device. While this method does not require an extensive amount of work to setup like fingerprinting does, there are still some prerequisites: the location of each base station (in this case WiFi access points) needs to be known and recorded so there is some reference to where the distance is being measured from; the clocks in each access point and the receiver device are precisely synchronised to ensure correct distances are calculated (GPS is accurate to 100 nanoseconds

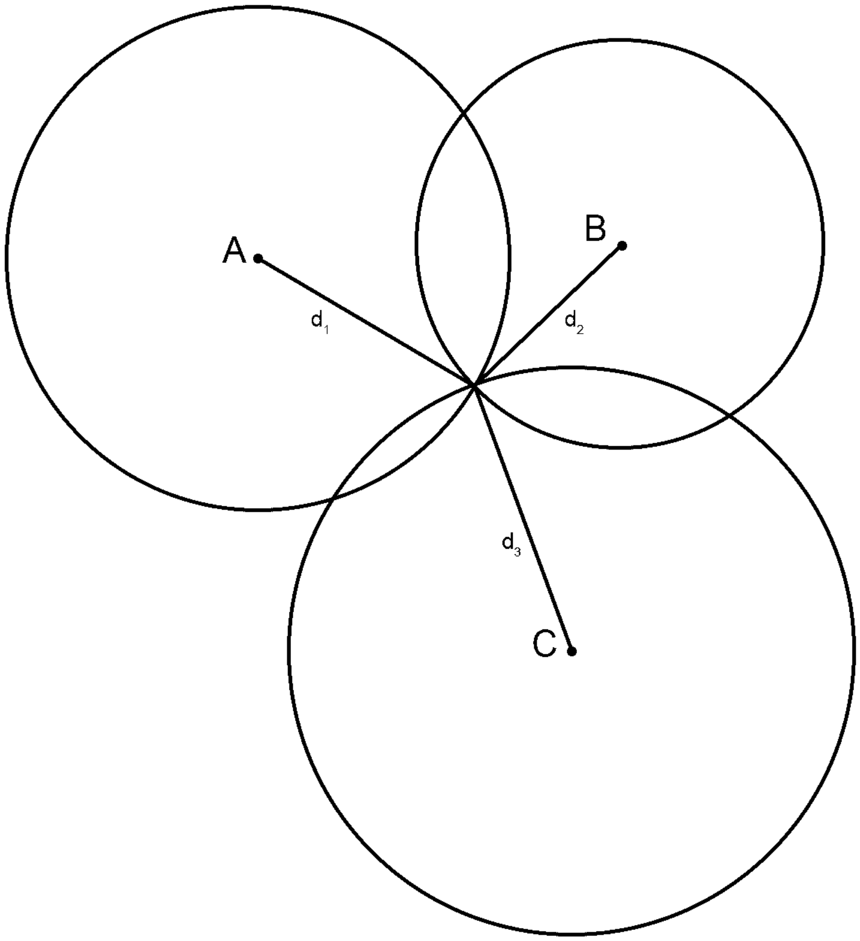


Figure 2.1: A 2-dimensional representation of the trilateration technique used with GPS and Time of Flight techniques

[12]); and timestamps must be included in packets exchanged between access points and a device.

When a signal is sent from an access point to a device, the timestamp, t_1, is recorded. The time it arrives at the device is then t_2. Assuming constant velocity of the radio signal travelling at the speed of light, c, the distance is given as:

$$d_i = c \cdot (t_2 - t_1) \tag{2.1}$$

where d_i represents the distance between the measuring device and access point i. Trilateration can then be used to find the position of the user, as shown in Figure 2.1. Circles of radius d_i can be determined around each access point, i, and the intersection point or area between three or more of these circles will represent the location of the device.

As mentioned before, precise accuracy in timing is required: the speed of light travels at 299,792,458 m/s and the time taken for an electromagnetic wave to travel just 30 metres is 0.0000001 seconds (equivalent to 100 nanoseconds). This degree of accuracy is not found in consumer-level hardware. Therefore, in a realistic scenario, specialist hardware will be required in both the surrounding infrastructure as well as for the user in order to properly make use of such a system.

Improvements to the system can be made by using the round trip delay instead of trying to determine the time of flight. This then forgoes the need for clock synchronisation as the responsibility of timing is placed on the client device. As presented by Schauer *et al.* in [23], the time of flight can be determined by measuring the time difference between sending a signal to an access point and receiving an acknowledgement back from the access point and then dividing that difference by two. Timing of a high degree of precision is still required in this case, so timestamps taken from the CPU clock was used which allowed for nanosecond precision.

2.3 Angle of Arrival-based Positioning

Angle of Arrival techniques use the angles and intersection of at least two signal vectors to locate a device. The basic principle of AoA uses the fact that a radio signal will travel slightly different distances to reach each of the antennae in an array, this is known as phase difference. This phase difference can be used to determine the angle of the incident radio signal. With this data from at least two access points, an intersection of the signal vectors can be used to determine the position of a device. Figures 2.2 and 2.3 visualise this concept.

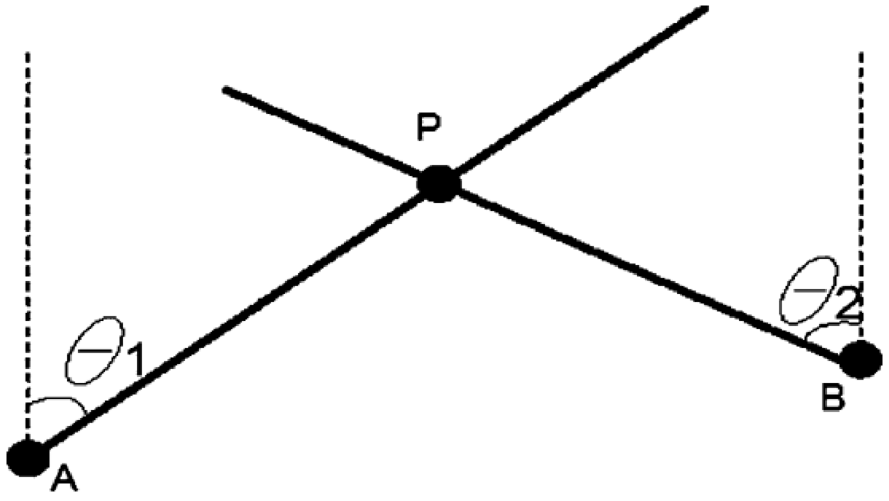


Figure 2.2: Angle of Arrival technique [19]

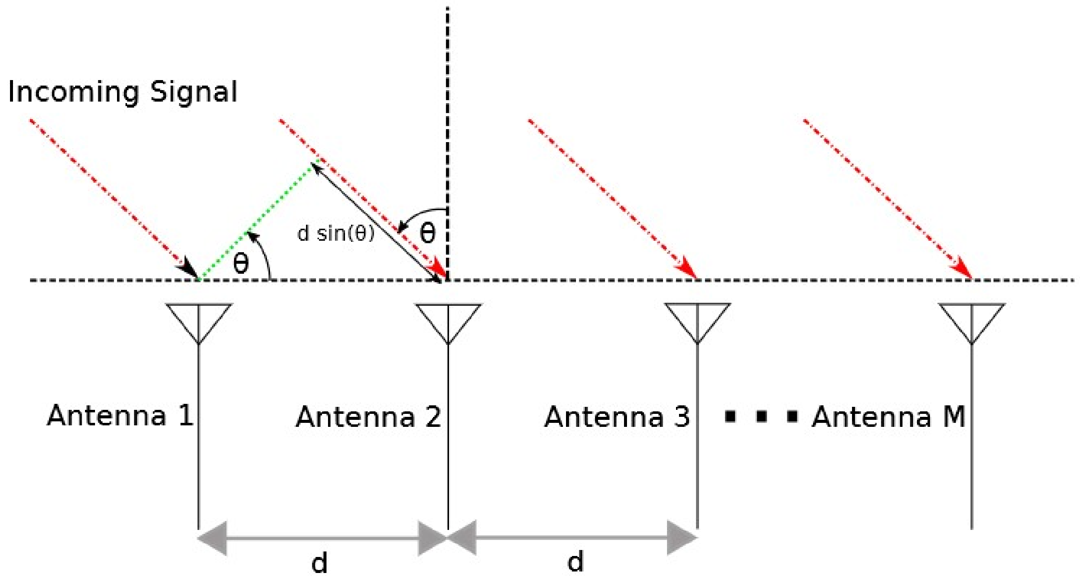


Figure 2.3: Phase difference is the result of the extra $d \sin(\theta)$ distance that the wave incident at antenna 2 has to travel compared to the wave incident at antenna 1 [25]

Some AoA techniques have required the same precision in timing and clock synchronisation as required with ToF; a navigation system called *TripNav* presented by Amundson *et al.* had such requirements but achieved an accuracy 0.95 metres while moving [2]; Chen *et al.* achieved an accuracy of 3 metres using an antenna array from consumer hardware which required clock synchronisation [7].

In order to measure the phase difference, $\Delta\phi$ of an incident wave between two antennae, the following equation can be used:

$$\Delta\phi = -2\pi \frac{d\sin(\theta)}{\lambda} \tag{2.2}$$

where, d is the distance between the two antennae, θ is the AoA, and λ is the wavelength of the wave.

Given Equation 2.2, θ can be calculated using the following equation:

$$\theta = \arcsin\left(\frac{\Delta\phi \cdot \lambda}{-2\pi d}\right) \tag{2.3}$$

This system is, however, a simple approach and is only relevant for a single wave. In a realistic setting, an incident signal will be composed of a multitude of waves affected by reflections, attenuation and time delays [25]. This is the multipath effect and will render the previously mentioned method ineffective for use in indoor positioning. To solve this, Schmidt presents the MUltiple SIgnal Classification (MUSIC) system in order to solve this problem [24]. MUSIC needs more antennae than signal paths; the MUSIC system can resolve at most $M-1$ signal paths, with M being the number of antennae, meaning that an environment that produces a large amount of the multipath effect may prove using AoA techniques difficult as cost and impracticability increases with the number of antennae.

2.4 Fingerprint-based Positioning

Fingerprint-based positioning consists of two parts: an offline phase and an online phase. The offline phase involves the mapping of an environment by recording the signal strengths at multiple locations around the environment. This goes to form a radio map which is then used during the online phase. The online phase is where actual positioning takes place; a scan is taken at an unknown location and is compared with the scans in the radio map, resulting in an estimated position.

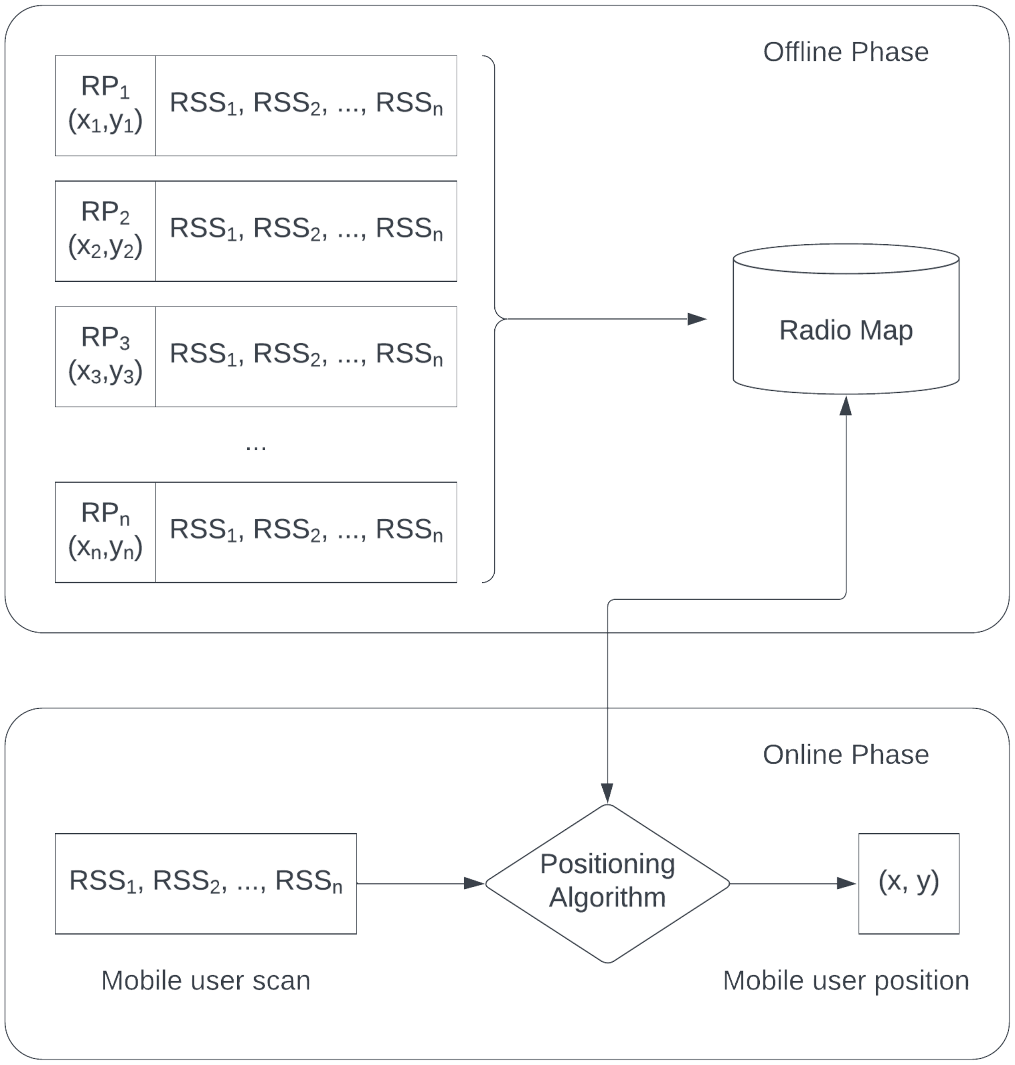


Figure 2.4: Fingerprint based positioning [5]

2.4.1 Offline Phase

As mentioned previously, the offline phase consists of mapping the target area for Access Points. The area will be split up into a grid, with each square on the grid representing a Reference Point (RP). At each RP, a scan is taken of all the 'visible' Access Points at that location and the signal strengths for each AP are saved to a radio map. Received Signal Strength Indicators (RSSI or just signal strength) values for the same AP can vary from one measurement to the next, therefore, multiple recordings should be taken at each RP and the signal strength values averaged in order to mitigate against fluctuations.

RSSI values can also vary depending on the orientation of the device and the user. A measurement taken with the user's position in between an AP and the fingerprinting device can reduce the signal strength by as much as 5 dB [21] compared to the user being on the opposite side of the device due to attenuation effects caused by the human body. In order to mitigate against this effect, fingerprints taken at each RP should also be taken in multiple different orientations.

The positioning system also relies on the geometry and the layout of the mapped area remaining consistent. This is due to objects such as furniture, walls and people causing attenuation to the signal. Any changes to the layout are likely to change the fingerprint of the area and negatively impact the positioning accuracy, with especially severe cases requiring a reconstruction of the radio map.

2.4.2 Online Phase

During the online phase, a scan will be taken at the device's location, which will then be compared to the fingerprints in the radio map. There are several machine learning algorithms that have been used before, and we will cover several of them now.

K-nearest neighbour

The KNN algorithm is the most commonly used algorithm used with fingerprint-based positioning and typically uses the Euclidean distance between the unknown fingerprint and every scan in the radio map to produce a location estimate [20].

Given a radio map with n recorded access points, a radio map vector, $R = \{(q_1, r_1), ..., (q_n, r_n)\}$ where q_i is the position of the RP of the form (x_i, y_i) and r_i is a list of signal strengths at recorded at RP q_i, and the unknown fingerprint, $F = \{f_1, f_2, ..., f_m\}$, where each f_i is the signal strengths recorded at the unknown location, the Euclidean distance, d, between vectors R and F is given by:

$$d_i = |R_i - F| = \sqrt{\sum_{i=1}^{n} (r_i - f)^2} \tag{2.4}$$

The value of k determines the number of nearest reference points taken into consideration, with $k = 1$ effectively turning the algorithm into a lookup table, returning the estimated location as the point of the nearest Reference Point. Choosing a value of k larger than 1 would simply have the algorithm return the average of the positions of the selected nearest neighbours.

Weighted KNN

A simple addition to the KNN algorithm is adding a weight to each of the k chosen Reference Points. The weights are determined by Euclidean distance to each Reference Point and so aims to weigh those points that are closer to the unknown fingerprint more heavily in the final position calculation than those that are further away.

Given the k nearest points, $Q = \{(q_1, r_1), (q_2, r_2), ..., (q_k, r_k)\}$, the position of the unknown location is given as:

$$q = \sum_{i=1}^{k} \frac{w_i \cdot q_i}{\sum_{j=1}^{k} w_j} \tag{2.5}$$

Where the weight w is given as the inverse of the Euclidean distance between RSSI r_i from the RP and the corresponding RSSI from the unknown fingerprint r_f:

$$w_i = d^{-1}_{(r_i, r_f)} \tag{2.6}$$

2.4.3 Further Improvements

Choosing k

Increasing the value of k above 1 does not necessarily guarantee an improvement in location accuracy. As shown by Hu *et al.* in [14], increasing the value of k up to 20 showed an increase in the positioning error, but also showed that different test points required different values of k in order to provide the most accurate results. The paper in [14] then went on to present a self-adaptive WKNN algorithm (SAWKNN) in which the value of k could be dynamically set depending on the location to provide the most accurate results.

Clustering

The positioning system can be further improved, especially for those systems with a large number of reference points in its radio map. Clustering is a machine learning process that separates data into groups by similarity. The idea is then that each group is identified by some value that best represents the data for its respective group. Its use in fingerprint-based positioning is to create some number of clusters of fingerprints that are similar to each other and use this data as the radio map. The clustered radio map can then be used to speed up the online phase; the unknown fingerprint will only be compared to the fingerprints in the closest matching cluster(s), reducing the number of comparisons performed, therefore, improving computation speed and efficiency.

There are two main methods to clustering, as presented by Cramariuc *et al.* in [8]: the first is clustering based on the distance of two fingerprints. For this, K-Means Clustering is the preferred choice, with the value of K setting the number of clusters used with the Euclidean distance measure in Eq. 2.4. During computation, the closest cluster and its neighbours are used to determine the estimated position to provide accurate positioning at the edges of a cluster. The second way to cluster fingerprints is by signal strength, or RSSI values. Here, Cramariuc finds that Affinity Propagation Clustering and K-Medians Clustering produces the best results.

Distance Measures

Another area of the system that can be varied and investigated is the distance measure used as part of the online phase. The Euclidean distance metric as presented in 2.4 is the one that is most commonly used in implementations of KNN, however, it is not the only one available. A study done by Duong-Bao *et al.* in [9] compared Manhattan distance, Minkowski distance, Canberra distance, and Chi-Squared distance to the traditionally used Euclidean distance. The study found that WKNN implemented with Chi-Squared distance had, in general, the smallest mean error even when changing three independent variables: environmental conditions, number of access points, and grid spacing. The study also showed that Chi-Squared performed substantially better than the other distance measures when increasing grid spacing and therefore decreasing the number of RPs, making it an important one to consider when implementing and testing. Table 2.1 shows the equations for the four extra distance measures.

Distance Measure	Equation
Manhattan Distance	$d_i = \sum_{i=1}^{n} \lvert r_i - f \rvert$
Minkowski Distance	$d_i = \sqrt[q]{\sum_{i=1}^{n} (r_i - f)^q}$
Canberra Distance	$d_i = \sum_{i=1}^{n} \frac{\lvert r_i - f \rvert}{\lvert r_i + f \rvert}$
Chi-Squared Distance	$d_i = \sum_{i=1}^{n} \frac{(r_i - f)^2}{\lvert r_i + f \rvert}$

Table 2.1: Alternative Distance Measures

Chapter 3

Requirements & Specification

From the three implementations of WiFi positioning mentioned in the previous chapter, fingerprint based positioning is the one most suited for the task: it is the only method that can be wholly implemented using regular consumer-grade hardware, making it the most accessible. The purpose of this software, then, is to implement a fingerprint based positioning system using a previously recorded radio map of public WiFi access points. The software will, of course, provide functionality in order to construct said radio map and the software will then use this positioning system to provide users with a location service in which users will be able to determine their current location and navigate through the mapped building.

3.1 Requirements

The requirements for the project can be separated into two: functional and non-functional requirements. Functional requirements specify what the software should do in terms of it's functions and capabilities. Non-functional requirements cover aspects that determine the quality of the software, such as performance or usability.

3.1.1 Functional Requirements

F1: The program should be able to take fingerprints of a user-defined location to add to the radio map.

- The program should ask the user to input their current location.
- The program should scan the user's current location for access points.
- The program will save the resulting fingerprint to the radio map to incorporate it in future location measurements.

F2: The program should utilise WiFi scans to determine the device's location to within 15 metres.

F3: The program should allow for different implemented positioning algorithms to be used to allow for testing.

F4: The program should obtain the necessary permissions from the user to allow for accurate positioning.

F5: The program should display the user's location according to the radio map.

3.1.2 Non-Functional Requirements

N1: During the offline phase, scans taken at each RP should be taken over a 30 second period .

N2: The user should not require any extra equipment besides their phone to interact with the system.

N3: The program should update it's estimated position of the device every 15 seconds.

N4: The program should support Android devices running a minimum of Android 8.

N5: Users should be able to easily interact with the application.

3.2 Specification

The specification states how each requirement will be satisfied in the implemented application. Functional requirements start with the letter 'F', non-functional requirements start with the letter 'N'. Requirement numbers will match up the numbers used to specify the requirements.

3.2.1 Functional

Table 3.1 specifies how each functional requirement will be met.

3.2.2 Non-Functional

Table 3.2 specifies how each functional requirement will be met.

Requirement Number	Requirement	Specification	Importance
F1	The program should be able to take fingerprints and associate them with a reference point	Save fingerprints as MAC address signal strength pairs. Do not allow users to take scans before inputting a grid coordinate.	High
F2	The program should utilise WiFi scans to determine location to within 15 metres.	Implemented positioning algorithms will use radio maps created by the user. Performance metric will be assessed in Section 7.	High
F3	The program should allow for different positioning algorithms to be used.	Implement KNN & WKNN positioning algorithms as their own modules so that they can easily be interchanged.	Medium
F4	The program should obtain the necessary permissions from the user.	Place permission checks before performing any functions that would require the users permission (e.g. WiFi scanning).	Low
F5	The program should display the user's location according to the radio map.	Positioning screen will show the coordinate according to the position estimate.	High

Table 3.1: Specification on how each functional requirement will be met

Requirement Number	Requirement	Specification	Importance
N1	Fingerprint scans should be taken over a 30 second period.	Scan throttling will be disabled for the sake of the investigation which will allow for a high enough rate of scanning over the 30 second period. Pressing the scan button will trigger the app to take scans over a 30 second period, prompting the user when they can continue.	Medium
N2	The user should not require any extra equipment besides their smartphone.	Fingerprint-based positioning does not require the user to posses any extra hardware. Both public and private access points can be used to map a location requiring no extra cost for someone to setup their own system.	High
N3	The program should update its estimated position every 15 seconds.	The 15 second update period will be met with a timer.	Low
N4	The program should support Android devices running a minimum of Android 12.	Android 11 is no longer officially supported by Android since February 2024. The app will be developed in Android Studio and, using its emulation tools, will ensure that the app will support devices on Android 12 onwards.	Low
N5	Users should be able to easily interact with the app.	Performance metric will be assessed in Section 7. This metric is less important due to the research nature of the project.	Low

Table 3.2: Specification on how each non-functional requirement will be met

Chapter 4

Design

This section will cover how the general process is to be broken down and implemented as a cohesive application. The general process on how fingerprinting works has been covered already, instead, a general overview of the system components and environment will first be outlined, followed by details of how the main modules of the codebase will be implemented.

4.1 System Structure

4.1.1 Positioning Device

The positioning device refers to the client device that will be used to apply and test the positioning algorithms. In this case, the term refers to the smartphone that will be used to map an environment using fingerprints and perform positioning calculations as per the currently implemented positioning algorithm. As mentioned in the Requirements and Specification, a smartphone using Android will be used. Since there is a need for the system to be as openly accessible as possible, only consumer-level hardware may be used, which ensures that the system as a whole can be tested and evaluated from a representative viewpoint; that of the users'.

4.1.2 Access Points

Access Points can be WiFi routers, computers or devices sharing a hotspot, or any other device with a Network Interface Card (NIC) that is broadcasting and will allow connections, such as a printer. WiFi networks are often secured with a password before a user is allowed access to the network, this, however, is not a hindrance to a positioning system; Access Points will still broadcast their identifying information before connecting. Every AP can be identified by their

Service Set Identifier (SSID), which identifies the network that you are trying to connect to, for example, '*eduroam*' or '*McDonald's Free WiFi*'. SSIDs are **not** unique. The other identifying piece of information is the MAC address (also referred to as Basic Service Set Identifier or BSSID), which is a **unique** 12 digit hexadecimal number assigned to each NIC and will identify the device that is being connected to. A network can consist of several APs, meaning each AP in the network will broadcast with the same SSID, but will each be individually identified by their own unique MAC address.

4.1.3 Radio Map

The radio map is effectively a database, storing Reference Points and the fingerprints associated with those RPs. How a room or a building is broken down into its RPs are defined by the developer or admin of the positioning system. Typically RPs are representations of a coordinate where the environment has been split up into a grid beforehand. When taking fingerprints at each Reference Point, two pieces of data will be saved; the MAC address of a wireless access points and the signal strength of that AP. A list of these MAC address signal strength pairs will be saved to represent all of the visible APs at an RP and a list of RPs will make up a radio map. The MAC addresses will be saved so that, in the case that there is only a subset of the visible APs in the unknown fingerprint, the same APs can be compared and the rest dropped. Since a MAC address is a fixed, unique and permanent identifier for a single wireless transmitter, this is the most reliable method to identify each unique AP.

In implementation, the radio map will be an SQLite database in Third Normal Form since the number of scans we will receive at each reference point will be variable. One table will store RP coordinates and another table will store scan data such as MAC address and signal strength. Each scan will have its own primary key but will also use the primary key of the reference point table as a foreign key to represent the reference point the scan was taken from. Being in Third Normal Form, the possibility of duplicates in the table will be eliminated as trying to save the same address as another with a signal strength will either be for a different RP which is acceptable or for an update for the same RP.

Figure 4.1 shows a sequence diagram of how the three system elements will be used to enable a positioning system.

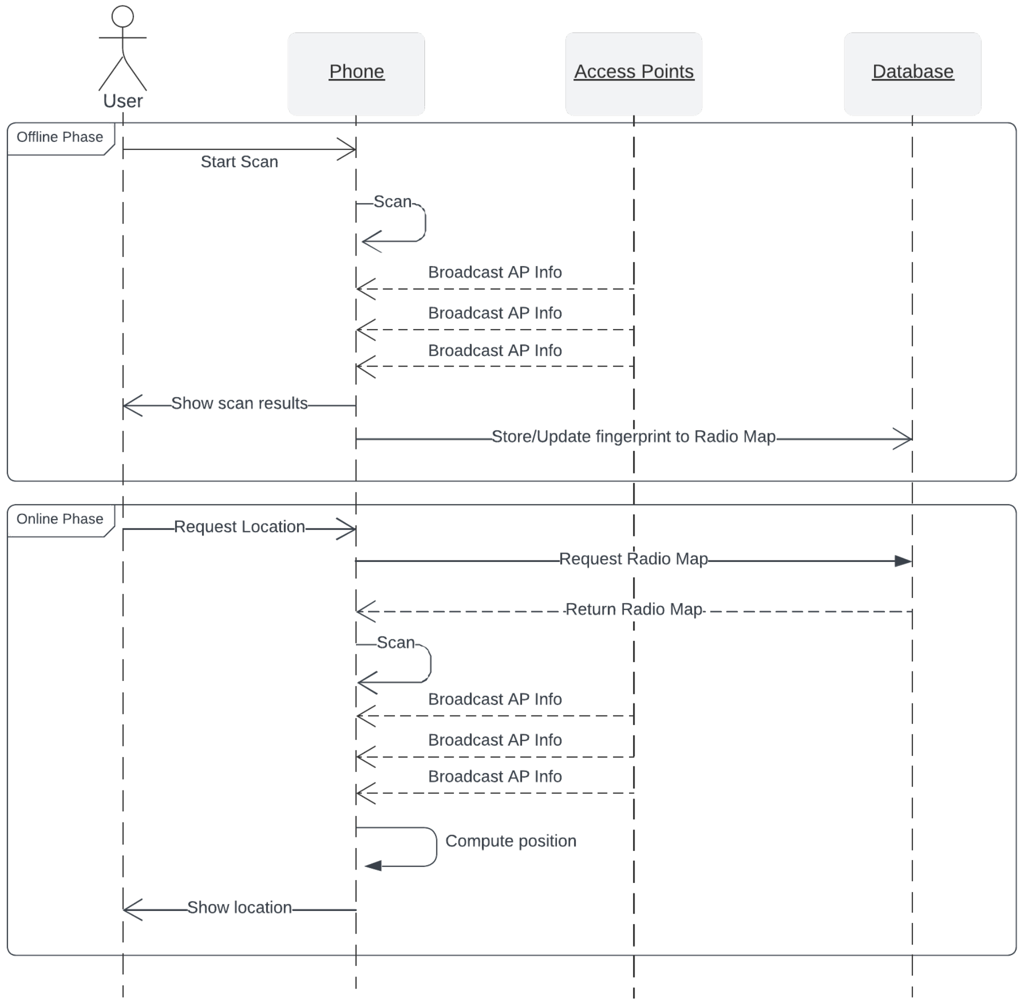


Figure 4.1: Positioning System Sequence Diagram

4.2 Code Structure

To allow for a high degree of variability in how the system performs, namely the positioning algorithms or the distance measures used, the app will be designed to be as modular as possible.

4.2.1 Development Environment

Since the platform targeted is Android, the obvious development environment to use would be Android Studio[1]. It is the recommended Integrated Development Environment (IDE) to use when creating Android apps and is bundled with features tailored for Android development such as API integration, user interface preview or device emulation. The IDE comes with native

[1]https://developer.android.com/studio

support for building a user interface, allowing for the process to be streamlined so that the main focus can be the actual functionality of the app. The logic of the app will be coded using Kotlin, a modern programming language that is officially supported by Google and Android and is completely interoperable with Java since Kotlin compiles to the JVM. Kotlin essentially extends Java with type inferencing, requiring the developer to explicitly state whether a variable can be null or not, reducing the chance of Null Pointer Exceptions. It also pushes developers to code in a more immutable style which is ideal for an asynchronous environment such as mobile development and results in approximately 40% less code than its equivalent in Java [15].

4.2.2 Scanning

Both the online and offline phase of the system will require a way to access the system hardware and perform a scan for wireless access points, so this module must be designed in a way that is reusable. WiFi scanning in Android is an asynchronous process. As of Android 10, the process requires a multitude of permissions from the user before scanning can take place, so there will be checks in place that will ask the user for these permissions if they haven't been authorised already. Another limitation to scanning is scan frequency throttling, which is set at a maximum of four scans in a two minute period as of Android 9. This throttling limitation can be removed by enabling developer tools on the device which is what will be done for the sake of this paper.

Once a scan has been requested, we can then view the scan results. These are not necessarily the latest scans; in the case of a scan failure, for whatever reason, the most recent available scan results will be returned. The results returned by Android are a list of type `ScanResult`[2]. A `ScanResult` object stores various information about the access point and the scan itself, however, the two important member fields needed are `level` and `BSSID`, which represent the signal strength and the MAC address, respectively, the two main details needed for fingerprint positioning.

The use case diagram in Figure 4.2 defines how two types of users will rely on a WiFi scanning module.

Regular users are those who would use the app for it's navigational features and are modelled as the actor 'App user'. 'Administrator' users are those who would be involved in determining Reference Points in the floor plan and recording fingerprints at each RP, creating the radio map.

Use case *Locate user position* displays the map and the user's location. The actual process

[2]https://developer.android.com/reference/android/net/wifi/ScanResult

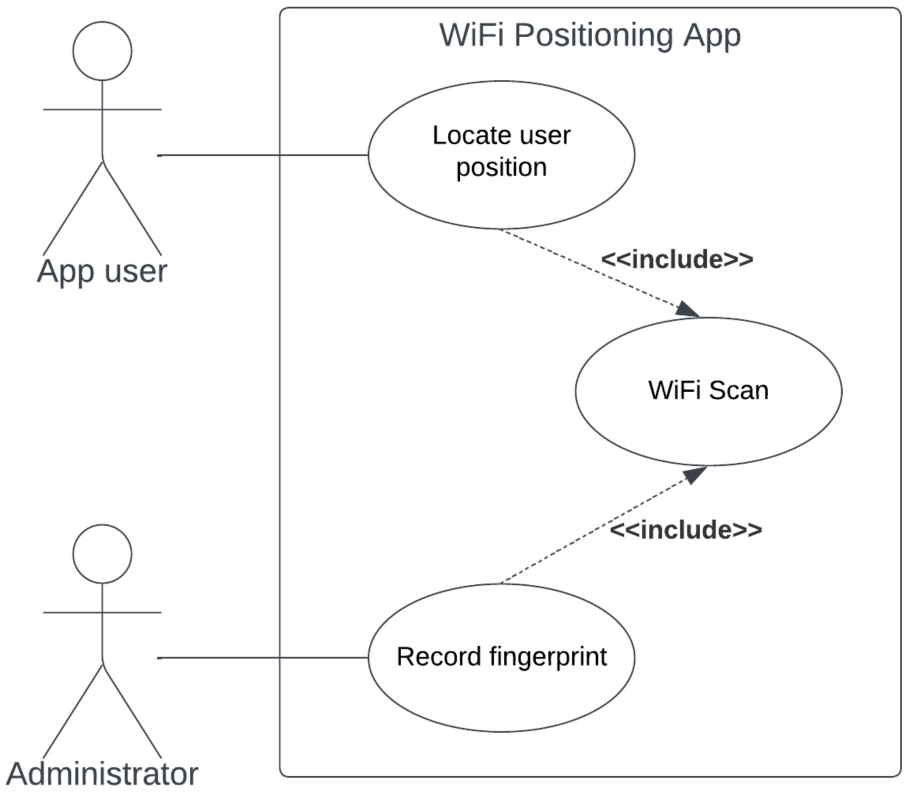


Figure 4.2: Positioning System Use Case Diagram

of locating the user's position is an automated process and is therefore always active as soon and as long as the application is open. This use case represents taking a scan from the user's current location, and performing the selected positioning algorithm to compare the fingerprint to those in the radio map.

Use case *Record fingerprint* is for use during the offline phase and involves the process of generating a fingerprint for a given Reference Point and then updating the radio map with the given fingerprint.

While the distinction between the two users here is useful in determining what functions the application needs to provide, any implementations of the application would merge the two users for the sake of ease of use while investigating different positioning algorithms. That is, both functions of locating the device and recording extra fingerprints will be available at the same time so that investigation of both roles can take place.

4.2.3 Positioning Algorithms

As mentioned before, the system will be using a fingerprint-based positioning model. For the sake of testing, several different algorithms will be implemented. Firstly, both KNN and WKNN will be implemented with the traditional Euclidean algorithm to find the positional accuracy gain when moving to a slightly more sophisticated algorithm. At the same time, the value of k will be adjusted to find the best possible positioning accuracy. Next would be to test the differences in accuracy when changing the Euclidean distance measure for at least two of the four other measures mentioned in Table 2.1. Since there is a large amount of variability required, both positioning algorithms (KNN and WKNN) will be implemented as part of a modular design so that the ability to switch between the two algorithms will be easy and adding other algorithms in the future is simple, too. Furthermore, the different distance measures will be implemented as their own separate functions so that comparing between them will only require passing a different function as a parameter.

4.2.4 Front End

The front end or the user interface for the application will entirely involve two different views: one for constructing the radio map as part of the offline phase, and the other representing the online phase, locating the user device. The app will use a taskbar to switch between the two different views, allowing for easy navigation between the two modes. The entire front end will be created using the Views[3] system, which is a legacy Android UI toolkit but is very mature and intuitive to use.

Scanning

The most important interaction between the app and the user when constructing the radio map will be determining the position of a scan. The app will not be able to determine the position by itself during the offline phase and so will require the user to enter the coordinates that they are stood in before allowing any scans to take place. There will be two inputs at the top of the screen, one for an x-coordinate, the other for a y-coordinate, and a scan button at the bottom that will only be enabled once both coordinates are provided. To provide some feedback to the user, the results of a scan will be displayed as the body of this view.

[3]https://developer.android.com/develop/ui/views/layout/declaring-layout

Positioning

For the part of the app responsible for positioning the device, the main body of the screen will consist of a text box that will display the estimated position. During the locating (online) phase, the app will use the radio map and will begin taking scans. Any positioning algorithm implemented should return a valid map coordinate as the estimated user location so that this coordinate can be displayed on the screen. The app should continue to scan and update the location on the map until the user returns to the fingerprinting screen.

Chapter 5

Implementation

Development of the application was done using an agile methodology. This development style prioritises creating prototypes of an application, with each iteration making improvements on the last. This allows for changes to be easily made in the middle of the development process, as each iteration of the product can be tested and evaluated in whether the current direction works or not.

Designing the system in modules also lends itself to an agile development method as each module can be developed and iterated upon by itself until it is ready for integration with the rest of the system. Testing and evaluation of each new module is simple due to the intentional loose coupling between modules which also facilitates the intention of improving upon each iteration.

Another benefit of this approach is that testing can be done at the end of each iteration, ensuring that the platform is solid for the next development cycle. In this case, each iteration roughly equates to a module and these modules can then each be individually tested. Writing the program in Kotlin provides another advantage in testing in that tedious tests such as checking for any undefined behaviour with null pointers do not need to be written, thanks to its null safety features. Many Android constructs also feature null safeties, providing defined and non-fatal behaviour when coming across unexpected null pointers.

5.1 WiFi Scanning

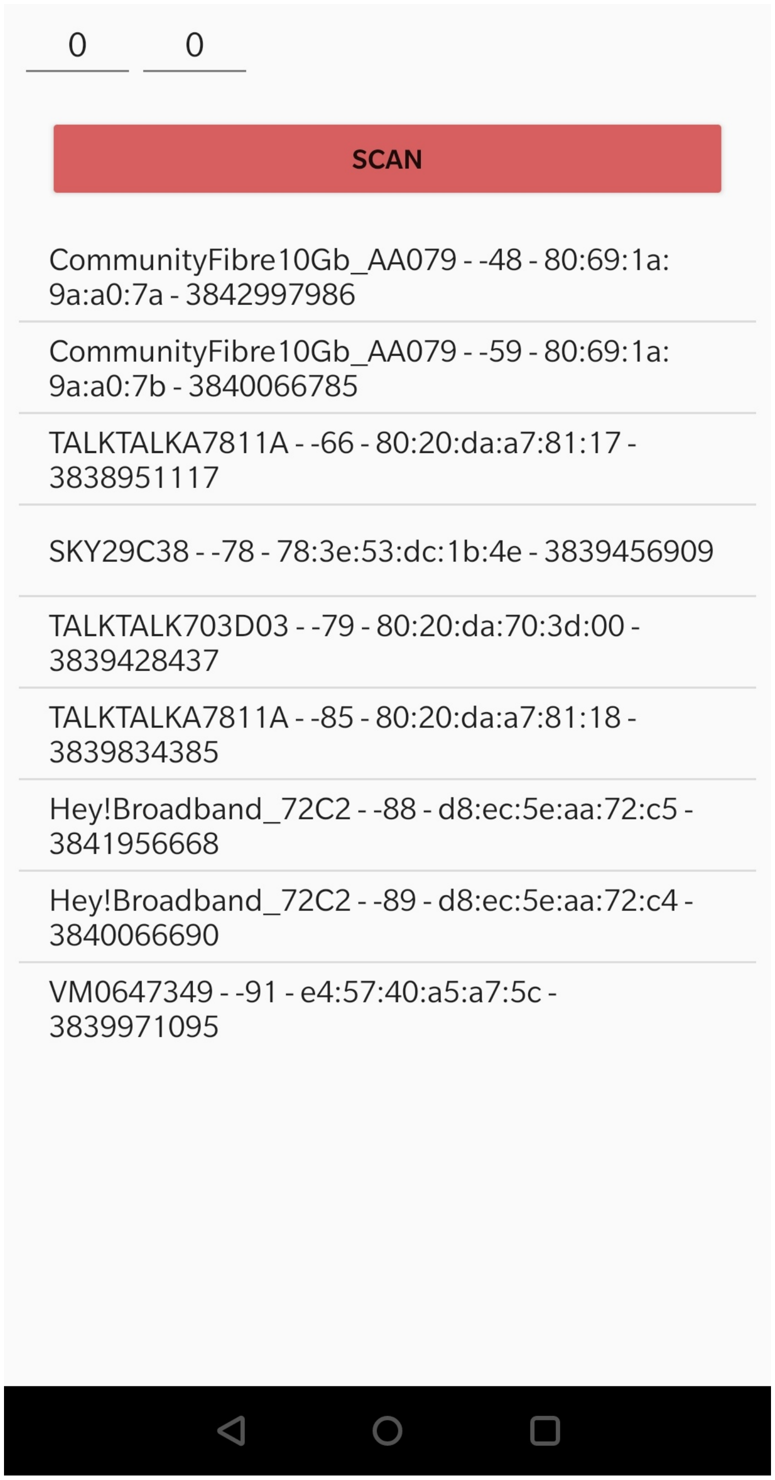
The first thing to be implemented was functionality for scanning for WiFi Access Points. The entire system is based upon scanning for APs so this module needs to be reusable for the implementations of both the online and offline phase.

The initial version of the scanning system consisted of a scan button at the top of the screen with the rest of the space being empty. Pressing the button would trigger the device to perform a WiFi scan and the empty space below the button would become populated with information of the access points that were picked up from the scan. Details include: SSID of the access point, MAC address of the access point and the signal strength of that AP.

The main aim for this version of the app was to determine the best method for scanning for WiFi. In order to trigger a scan on Android, the `startScan()` method is called on an instance of `WifiManager`. WiFi scans are an asynchronous process, so scan results are not returned back immediately. In order to process the scan results, a `BroadcastReceiver` object needs to be registered with the system along with an `Intent` filter set to `SCAN_RESULTS_AVAILABLE_ACTION`. `Intent`s can be simply thought of as boolean flags that are set whenever a corresponding action occurs. Registering the receiver with this intent will register the `onReceive()` method of `BroadcastReceiver` as a callback method when the scan results available flag is set. If we want to do our own processing of the received scan results, the `onReceive()` method needs to be overridden. Therefore, we extend the `BroadcastReceiver` class with our own `WifiReceiver` class in order to provide our own implementation of the `onReceive()` method. With this, we get access to the data resulting from a WiFi scan and our method gets called automatically whenever scan results are available.

As was briefly mentioned, scan results come in a list of the class `ScanResult`. This class is a wrapper that holds the information for one of the access points recorded in the scan, storing information such as identifiers like the SSID or MAC address, channel width and frequency information and signal strength, among other things. In this initial implementation of the app, our version of the `onReceive()` method takes this list of results and extracts the SSID, MAC address and the signal strength and formats this data into a string. This new list is then used with a `ListView` UI element to display the list of results to the user.

The next consideration was user input. When scanning an environment to construct a radio map, the position of that scan must also be saved along with it. With no way to determine the position of the user without first constructing the radio map, the user must input their own location. In our system, positions are represented by a coordinate system, an x value and a y value. How the environment is mapped to a coordinate system is up to the developers prerogative, however, a coordinate will need to be input regardless. To facilitate this, two `EditText` boxes were placed at the top of the screen. These boxes allow a user to input some



CommunityFibre10Gb_AA079 - -48 - 80:69:1a:9a:a0:7a - 3842997986

CommunityFibre10Gb_AA079 - -59 - 80:69:1a:9a:a0:7b - 3840066785

TALKTALKA7811A - -66 - 80:20:da:a7:81:17 - 3838951117

SKY29C38 - -78 - 78:3e:53:dc:1b:4e - 3839456909

TALKTALK703D03 - -79 - 80:20:da:70:3d:00 - 3839428437

TALKTALKA7811A - -85 - 80:20:da:a7:81:18 - 3839834385

Hey!Broadband_72C2 - -88 - d8:ec:5e:aa:72:c5 - 3841956668

Hey!Broadband_72C2 - -89 - d8:ec:5e:aa:72:c4 - 3840066690

VM0647349 - -91 - e4:57:40:a5:a7:5c - 3839971095

Figure 5.1: WiFi scanner with a list of scan results showing the SSID, signal strength (dBm), MAC address and timestamp (time in milliseconds since boot) for each AP

data and this data can be read through accessor methods. Since a fingerprint cannot be placed into a database without a position for the fingerprint, a check is placed on both boxes through the use of the a listener. The listener is executed when text in the box is edited and, in this case, only enables the scan button when both boxes are populated with data, ensuring that every scan has a corresponding coordinate to go with it. Furthermore, since coordinates are numerical values only, the boxes have the `inputType` parameter set as `number`. When a user taps on one of the boxes to edit the text in it, Android will only allow for a numerical input and display a numerical only keyboard. These two coordinate values can be accessed through code by simple accessor methods once the two `EditText` objects are instantiated.

At the end of this iteration of the program, the user was able to input two numerical values corresponding to the coordinates of the reference point that they were stood at, the user could press a button which triggered a WiFi scan and, within a few seconds, had a list of all the access points detected by the device display on the screen. This can be seen in Figure 5.1. Since a database mechanism had not been implemented yet, a small dialog box at the bottom of the screen would display the input position to ensure that the program was reading the values correctly.

5.2 Database

The next logical step in implementing the system was the database. The results from the scans needed to be saved so they could be accessed during the online positioning phase. As mentioned in the Design chapter, a two-table SQLite database is being used to store reference points and scans.

5.2.1 DbHelper

The database is implemented by extending the `SQLiteOpenHelper` class provided by Android with our own `DbHelper` class. The class inherits methods that assist with creating, opening and maintaining an `SQLiteDatabase` object that is saved to the app's private storage space. Extending the `SQLiteOpenHelper` class requires implementing two abstract methods: `onCreate()` and `onUpgrade()`. `onCreate()` is called when the database is first created and so this implementation executes two SQL statements that create tables *ref_points* and *scans*. Table *ref_points* has columns ID, xCoord and yCoord. It saves each reference point coordinate with its own unique primary key. This key can then be used to easily identify RP a scan belongs to. Table *scans* has columns ID, RPID, address and level. This table saves each individual scan as its own row in the table. Column RPID is a foreign key from *ref_points* and references its primary key. This helps differentiate scans with the same MAC address by the RP the scan was taken at. The second abstract method, `onUpgrade()`, specifies what to do when the database is upgraded, in our case, we delete the tables and recreate them.

The rest of `DbHelper` contains functions that will be used by the rest of the app to save, retrieve and update data in the database. To get data from the database, the `query()` function is used; this function takes in arguments such as table name, columns to return, and selection criteria to construct an SQL query and returns the corresponding rows as a `Cursor` object. A cursor object acts as an iterator over the returned rows, providing methods to navigate through rows of data or to return the data in a certain column for the row currently selected.

```
val cursor = db.query(
    /* table = */ "scan",
    /* columns = */ arrayOf(SCAN_ID),
    /* selection = */ "$SCAN_RP_ID = ? AND $SCAN_ADDRESS = ?",
    /* selectionArgs = */ arrayOf(rpid.toString(), address),
    /* groupBy = */ null,
    /* having = */ null,
    /* orderBy = */ null
)
```

Listing 5.1: A query function getting the SCAN_IDs of all rows where SCAN_RP_ID = rpid and SCAN_ADDRESS = address

Inserting data uses the `insert()` function which requires the name of the table that data is being inserted into and the values being inserted into the table. The `insert()` function returns the ID number of the new row that has just been inserted. While not particularly useful in our case, this still provides some feedback that the operation has occurred without error. The main advantage with using the functions provided by the `SQLiteOpenHelper` class over creating our own implementation of an SQL helper class is that operations performed on the database are done so safely with transactions enabled. While it cannot stop the developer from writing code that accidentally deletes the database, any errors that occur during SQL execution do not risk corruption. The helper class saves us time from implementing our own safe functions and allows us to perform the actual operations we need to without much worry.

The public functions that interact with the database are: `deleteAll()`, which deletes all the *rows* in the database; `getReferenceId()`, which takes in x and y values as parameters and returns the key ID of the corresponding row; `getAllReferencePoints()`, which returns a mapping of each RP ID to its corresponding coordinate as a tuple; `getAllScanResults()`, which returns a mapping from each reference point ID to all the scans associated with that RP, with the scans being represented as their own map from a String (MAC address) to an Integer (signal strength); `addNewReference()`, which adds a new reference point to table *ref_points* and returns the key of the new row; and `updateOrAddNewScan()`, which adds a new scan to table *scan* or replaces the values in the table if there is already an entry for the reference point and MAC address pair, returning the index of the row.

All of these functions require an `SQLiteDatabase` object passed in as parameter so that queries or insert operations can take place. This database object can be created by calling the function `getWritableDatabase()`. However, opening the database from closed is an expensive process; the database should remain open for as long as feasibly possible. To do this, a database object is created when the application is first initialised in the main activity and is threaded through the entire application so that the same instance is always being used.

5.2.2 Database Integration

In order to integrate database functionality with the scanning system, some changes had to be made to the previous design. Firstly, scan result data was only available inside the `WifiReceiver` class, whereas, the database instance was only available in the main activity class. Since receiving scan results is an asynchronous process, the receiver class needed some way to get the data back to its caller. The way this is done in this implementation is by requiring an abstract callback method as parameter to the `WifiReceiver` class. This method can then be implemented as an anonymous function by the caller, allowing for access to the scan data in the calling class.

```kotlin
wifiReceiver = WifiReceiver(wifiManager, object: ScanCallBack {
    override fun addScansCallback(results: List<ScanResult>) {
        calculatePosition(results)
    }
})
```

Listing 5.2: Anonymous callback function being passed as argument to WifiReceiver in the positioning fragment

The second change was to wait until multiple scans had been completed before adding a scan to the database. To do this, scans were launched from inside a coroutine, a lightweight version of a Thread supplied by Kotlin, which looped until the desired number of scans had been completed. To get meaningful results and to allow for sufficient processing time, a total of 4 scans are taken with an interval of 7 seconds between each scan, however these values can be adjusted by changing some instance variables. Testing showed that having too little of a delay between each scan (roughly anything below 5 seconds) would result in not all the scans being completed which would affect logic later down the line which was expecting 4 scans to have been completed and this is likely due to the asynchronous nature of requesting and performing WiFi scans.

5.3 Positioning

The final main feature that needed implementing was the positioning phase. In terms of user interface, the app would only require two screens for the offline and online phases respectively.

The screen for the offline scanning phase had already been implemented and did not need any major modifications. Therefore, to facilitate switching between two screens, the scanning functionality was moved to its own separate class as a `Fragment` and implementing the new positioning screen as its own `Fragment`. Fragments work as their own self contained segment with their own logic and interfaces, allowing us to switch between different fragments easily. The main user interface component then became a container that holds a fragment and a navigation bar at the bottom allows switching between the two fragments. A listener is used to wait for user interaction with the navigation bar and determines which fragment to display by the ID of the menu icon selected.

The logic of the positioning screen remains simple. A button allows the user to initiate a scan of their position by reusing the `WifiReceiver` class implemented for scanning functionality. This class had to be modified slightly since the receiver class took a `ListView` item as parameter. The positioning screen does not display the scans or use a `ListView` object, therefore, the parameter was removed and the responsibility of displaying the list is left to the callback method that is individually defined by the calling class. This serves to provide another layer of modularity, allowing other classes to freely use the receiver without needing to pass an arbitrary UI element as argument. The results of the scan is returned via the callback method passed to the receiver and, in the case of the positioning fragment, calls the `calculatePosition()` method with this scan data. Additional UI elements such as a dropdown box and a toggle switch allow users to choose the distance measure used as well as toggle between KNN and WKNN algorithms respectively.

The algorithms for calculating the position of the user is contained within its own class: `Positioning`. Since we aim to also test between different distance measures, another class, `Measures` is defined as an Enumeration. These enums implement their own versions of an abstract method to calculate the distance between two given fingerprints. The measures implemented are Euclidean, Manhattan and Chi-Squared distances. The algorithms for these measures can be seen in Equation 2.4 and Table 2.1. Creating an enum for different distance measures allows us to simply pass the measure we want to use as an argument to a function which can then use it to call the function to find distance. Having both of these classes separate from the rest of the code also allows for easier testing. Instead of having to rewrite code the positioning code that is hardwired in the body of an Android Activity, we can instead create an instance of a Positioning class and pass some test data without even needing to build and launch the app onto a mobile phone.

Finally, when the `calculatePosition()` method is called, the unseen fingerprint, the chosen distance measure, a map of reference point IDs to their coordinates and a flag to switch between KNN and WKNN are passed as arguments. The function then calculates the position given these arguments and returns the estimated location of the user as coordinates. This coordinate is then printed to the screen. Figure 5.2 shows a screenshot of the positioning screen.

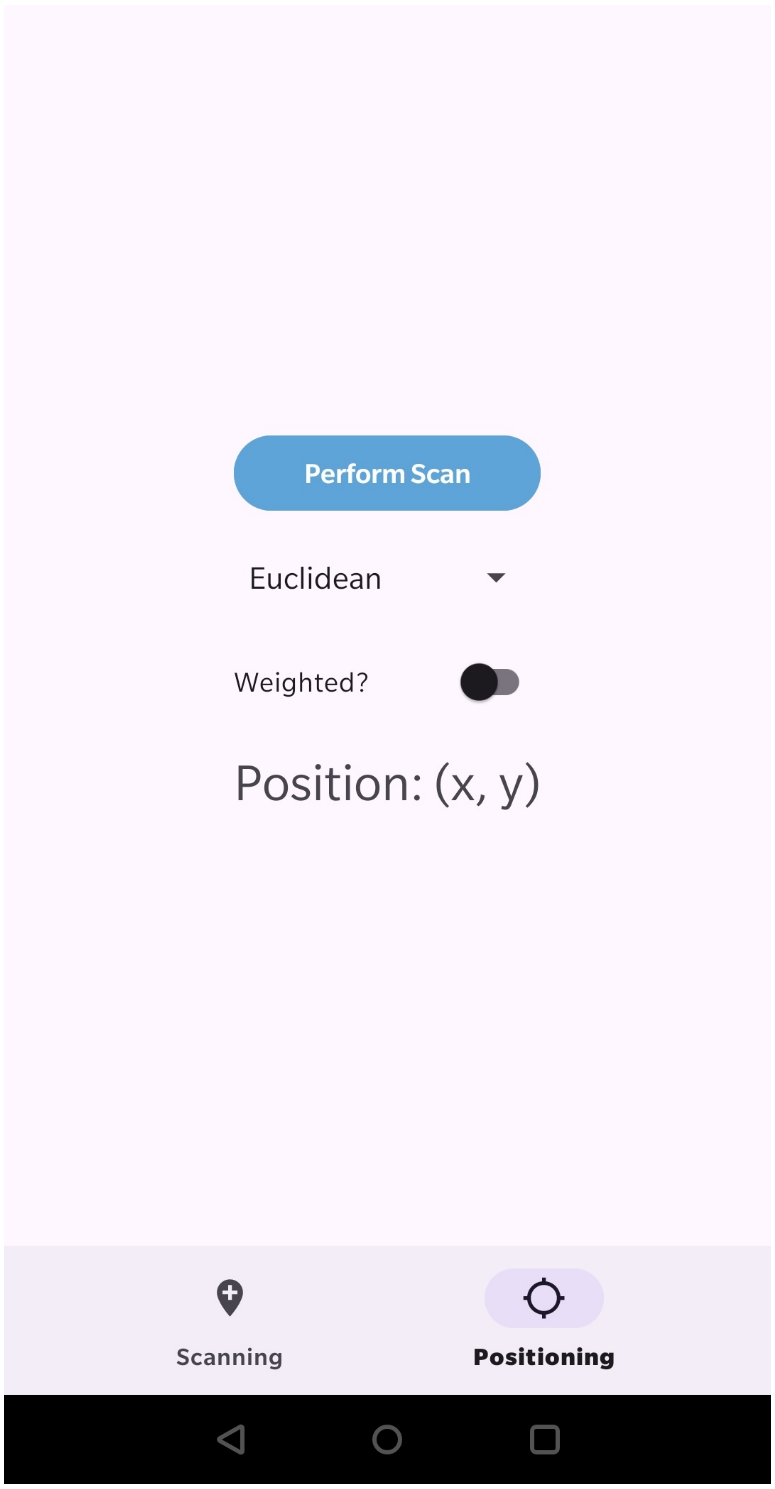


Figure 5.2: WiFi positioning screen. Users can select from the currently implemented distance measures (Euclidean is currently selected). The toggle switches between WKNN and KNN (KNN currently selected in figure).

Chapter 6

Legal, Social, Ethical and Professional Issues

6.1 BCS Code of Conduct

The British Computing Society Code of Conduct[1] is a set of rules and standards that all of its members are expected to abide by. They set out how members are to behave when working in computing and all members involved in this project have strived to comply with these standards. While only a select few of the guidelines apply to this project, any use of third party code, intellectual property or work not produced by me has been declared and accredited to the responsible individual(s).

6.2 Privacy & Data Security

While the current implementation of the application has been done so as a proof-of-concept of an indoor positioning system, there are still valid concerns and issues that need to be considered, most of them pertaining to a user's privacy and security of data. The main issue with any application concerned with a user's location is how such data is being used and if the data is accessible by any unauthorised third party. In its current state, the application data is saved in its own private storage area; this is upheld and maintained by the Android operating system, ensuring that user data remains secure. The source code for the application is open-source and freely available for scrutiny to make certain that the data is being used solely for providing a

[1] https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf

location service and that user data integrity is maintained and since user data remains entirely on the user’s device, there are no concerns in meeting GDPR standards.

A reasonable extension to the system would be to move fingerprint data to a server. This would be a more realistic use case for the system, as it is intended for use by multiple people, such as visitors to a building. Fingerprints made by users could also be feasibly used to contribute to the radio map, turning the collection process into a crowd-sourced effort. If this were to be the case, the user must be aware of what data is being captured and how it is being used. In this scenario, it is paramount to ensure that only the data needed from a user’s device is taken; any extra identifying data is considered a breach of user privacy.

Chapter 7

Results/Evaluation

There are a few different combinations of variables that need to be tested. Firstly, we will compare the positioning errors between using KNN and WKNN. The differences between the three distance measures we have implemented will also need to be evaluated and so there are a total of six unique methods that can be used to position the device. Finally, we will test the application against the requirements we set out to achieve in Chapter 3.

7.1 Positioning Accuracy

The first set of tests were performed on King's campus in one of the computer labs. The device used to conduct the test was a OnePlus 5T running Android 10. The lab was split up into twelve sectors, each roughly four metres apart. At each of these Reference Points, the app was used to take scans. As mentioned in the implementation phase, four scans are taken at seven second intervals, making a total of forty-eight scans. Each scan was also taken with the phone pointing in at least two different directions. Three Test Points were also determined and these points were used to test the six different combinations of positioning algorithm we could use and a value of K was arbitrarily chosen as 3. Figure 7.1 shows how the room was split up for the twelve Reference Points as well as the locations of the three chosen Test Points.

Figure 7.2 shows two graphs, both a top-down representation of the room. Three scans were taken for each of the six different positioning algorithm combinations at each of the three Test Points. The figure shows all of the predicted position coordinates returned by the application. The main characteristic seen in both the KNN and WKNN graphs is the fact that the positioning algorithm is weaker when trying to locate at the very edge of the mapped area.

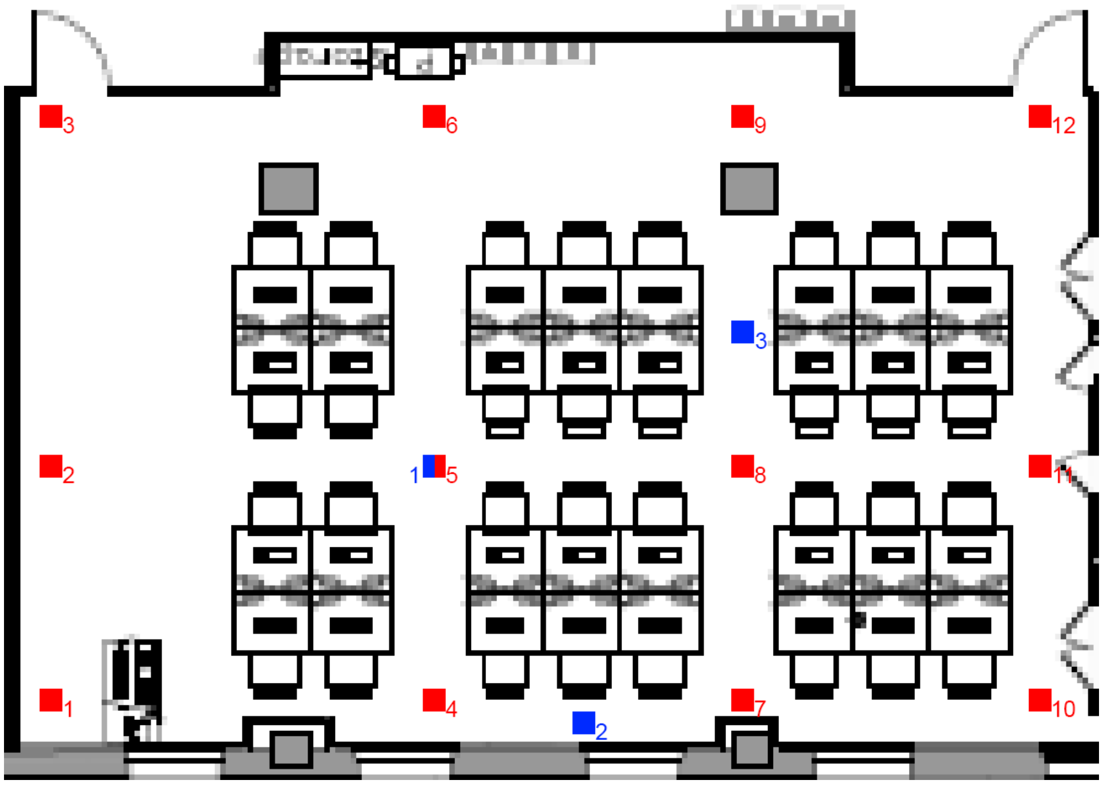


Figure 7.1: Map of computer lab used for testing. Red squares denote the locations of each Reference Point, with RP1 at coordinates (0, 0) and RP12 at (3, 2). Blue squares denote Test Points: TP1 – (1, 1), TP2 – (1.5, 0), TP3 – (2, 1.5). Each RP is roughly 4 metres apart. Room dimensions are roughly 14 metres by 9 metres.

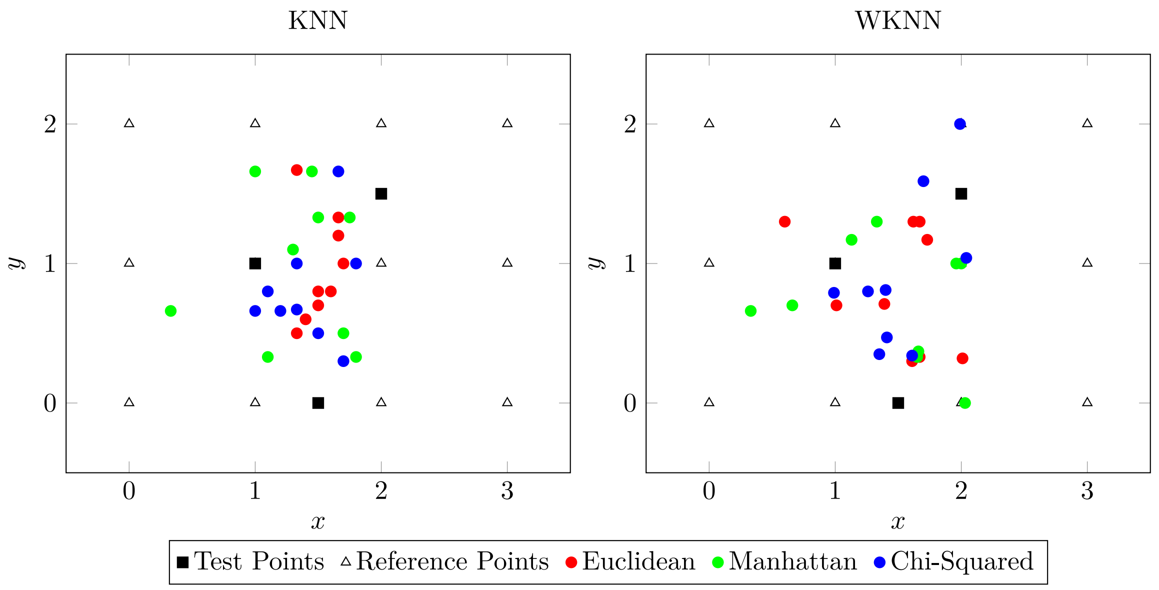


Figure 7.2: Graphs showing predicted positions returned by KNN & WKNN using different distance measures. Test Points numbered from left to right with TP1 at (1, 1).

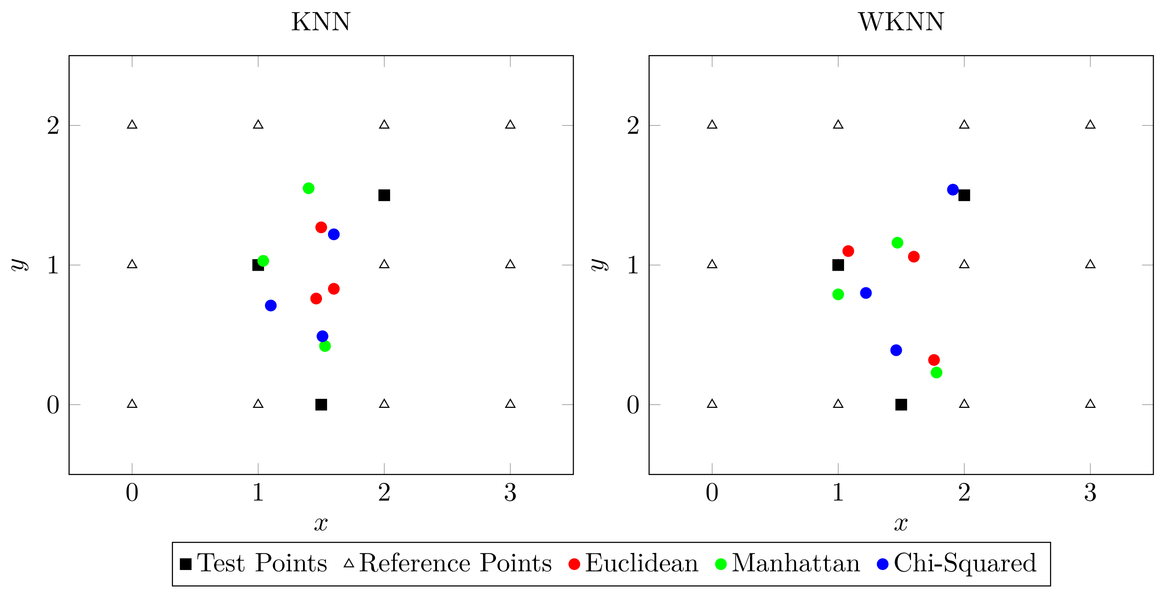


Figure 7.3: Graphs showing the average of the three predicted positions for each distance measure.

Positions returned when at Test Point $(1.5, 0)$, for example, mostly only have positive values of $y > 0.3$. Since there is no data in the negative y direction, it is not possible for the positioning algorithm to return such a value. Comparing between KNN and WKNN in general, we see that KNN produces results that could almost be described as random. WKNN tends to return results that are more grouped around each Test Point, despite of how loose that grouping might be.

More notably, however, taking the averages of the positions returned by each algorithm nets estimated locations that are much more accurate as can be seen in Figure 7.3. From this we see that, for KNN, the Chi-Squared and Manhattan distance measures perform better than the traditionally used Euclidean distance measure. This sentiment is repeated for the WKNN result, albeit Euclidean performs much better in this case – about on par with Manhattan. The position estimates seen in Figure 7.2 are each from one scan; Figure 7.3 shows that using the average of multiple scans will provide us with a much more accurate reading.

Figure 7.4 shows the average position errors for each combination of positioning algorithm including the new averaged positions that we have just seen. Out of the single scan estimated positions (solid lines), we see that Chi-Squared performs the best, with Manhattan and Euclidean performing about the same, with errors fluctuating depending on the scenario. However, when introducing the averaged data (dashed), Chi-Squared still performs well, but there is little to no improvement over its first set of results. Manhattan distance, on the other hand, has its

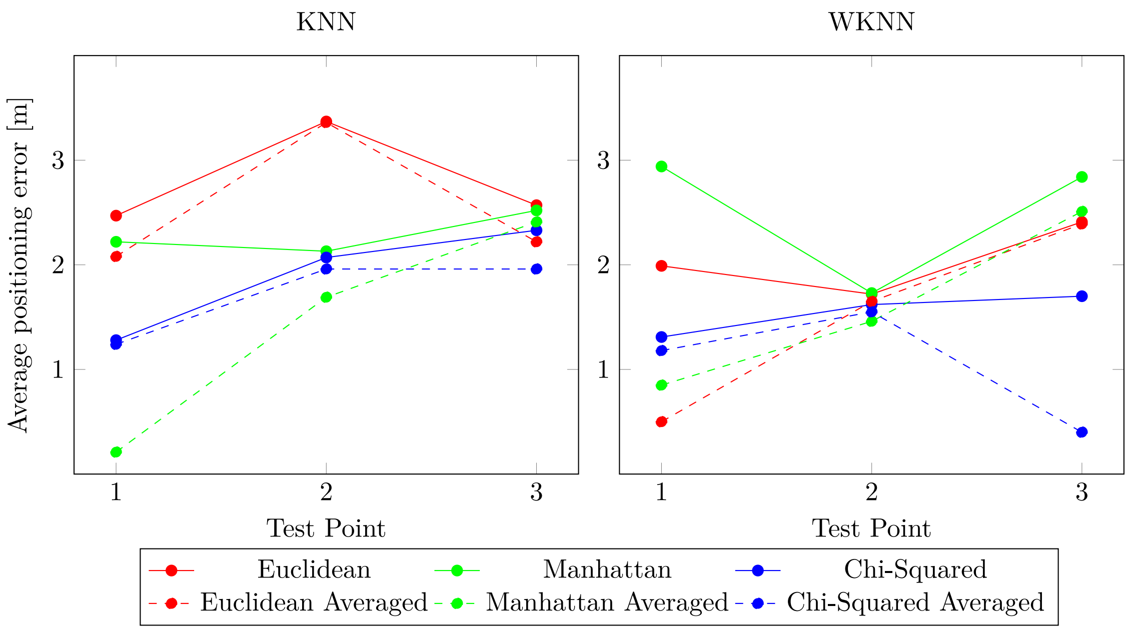


Figure 7.4: Graphs showing average positioning error at each Test Point with KNN & WKNN using different distance measures.

errors improve drastically. Whether or not this is a tangible improvement of accuracy will have to be tested with a second set of data. Comparing between KNN and WKNN, we can see that WKNN consistently performs better than KNN, with results for TP2 converging around the 1.6 metre mark.

7.1.1 Fingerprint Dataset

In order to further test the differences between our three distance measures, we will use another set of data. This dataset has been kindly made freely available by Duong-Bao *et al.* in [10]. Over the course of four months, the researchers constructed a dataset of 205 Reference Points, with 100 scans taken at each RP. The researchers performed accuracy tests of their own, comparing the performance of KNN and WKNN. They too found that WKNN performed slightly better than KNN. The distance measure or the specific algorithm they used is not specified in the paper, however, it is safe to assume that the conventional Euclidean distance measure was used. Testing of the dataset was done using just the WKNN algorithm this time; we have seen that, in general, it does provide accuracy improvements over KNN. A single test point was used for this example and the value of K was also tested from 2 to 14.

Figure 7.5 shows how the positioning error varies as K increases. It also shows the positioning accuracies for the three distance measures. While conclusions such as Euclidean distance

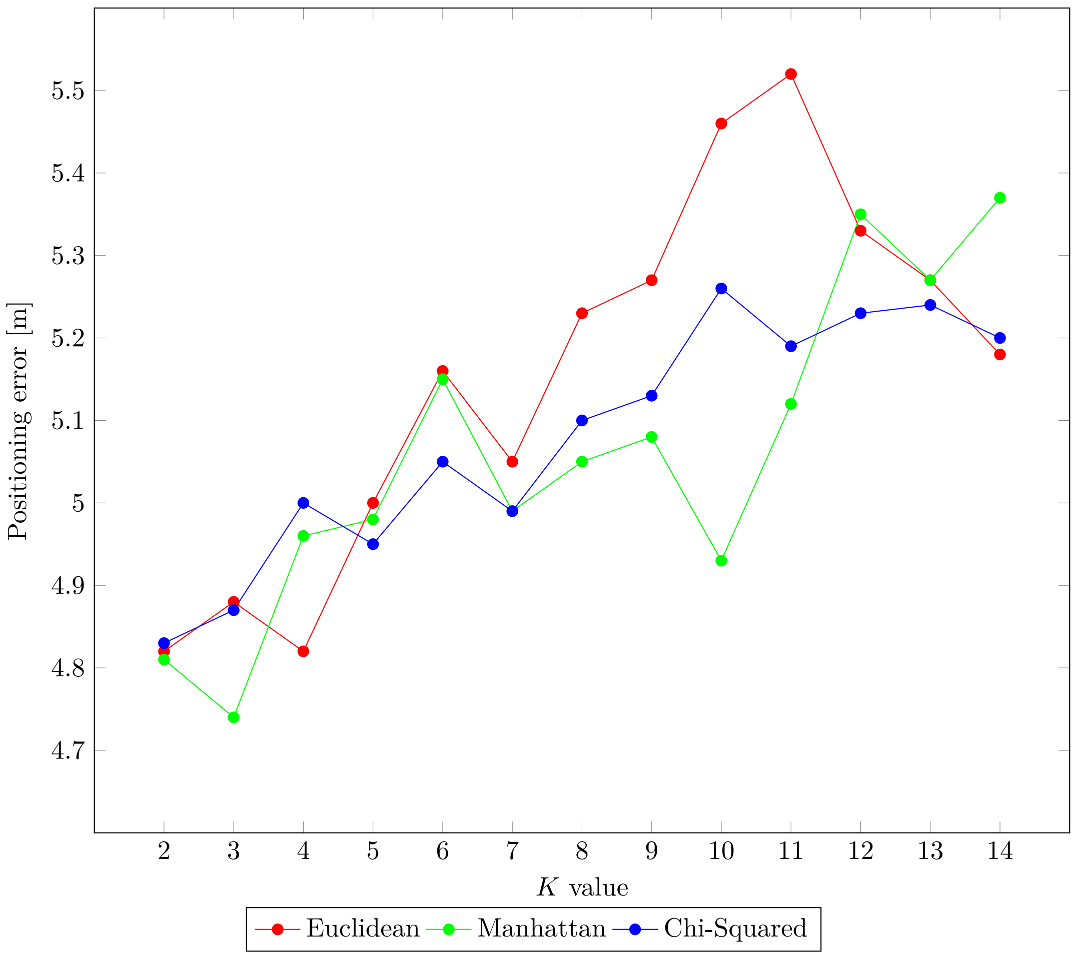


Figure 7.5: Graph to show how positioning accuracy varies over different values of K and with different distance measures.

performs the worst and Manhattan performs the best, the biggest difference between two different measures for the same K is roughly 0.6 metres. This level of difference is not significant enough to draw a conclusion. The reason for such a small difference in accuracy could potentially be due to the sheer volume of scans that were used to construct the radio map. With such a large amount of data, the naturally occurring variability of scan results is averaged out, resulting in similar performance for all three measures. Another explanation for the similarity of results is the layout of the RP map. There is just a 0.5 metre gap between each RP, meaning that many RPs in an area end up with similar data. This further reduces the variability that would be found in scans, bringing the scans closer together.

One thing that this graph does show, however, is the relation between K and positioning accuracy. There is clearly a positive correlation with increasing K and an increasing positioning error. Increasing K results in the algorithm introducing more and more arbitrary reference points, increasing the positioning error.

7.2 Requirements Testing

Here we will go through the requirements we defined in Section 3. It is important to check back and evaluate how well we have met each requirement to be able to determine how successful the project has been.

7.2.1 Functional Requirements

F1: The program should be able to take fingerprints and associate them with a reference point. This requirement has been met. As described in the Implementation section, an SQLite database is being used to save and store reference point locations as well as scan results. The use of a foreign key in the scan table allows us to associate each scan with the reference point it was taken at.

F2: The program should utilise WiFi scans to determine location within 15 metres. This requirement has been surpassed quite a bit. An initial accuracy requirement of 15 metres was set as that would have been sufficient to provide room-level accuracy, accomplishing our goal. As can be seen in Figure 7.4, the application has achieved positioning errors up to a maximum of 3 metres and, in some cases, as low as 0.2 metres.

F3: The program should allow for different positioning algorithms to be used. The positioning screen features a toggle that allows for switching between KNN and WKNN. The screen also features a dropdown menu that contains a list the three currently implemented distance measures. If others were to be implemented, they could simply be defined and added to the dropdown. Figure 5.2 shows the UI allowing for a user to choose the positioning algorithm. This requirement has been met.

F4: The program should obtain the necessary permissions from the user. Obtaining permissions is not something that was covered but it is indeed an important feature to have. Android does not allow for scans to be performed without the necessary permissions having been granted already. In this case the app does not properly check that the correct permissions are granted, failing this requirement point. Not having met this requirement does not affect the outcomes of this project, however, but should this project ever be considered for commercial release, this feature must be working.

F5: The program should display the user's location according to the radio map. When the user taps on the scan button on the positioning screen, the text field at the bottom of the view is updated within a matter of seconds with the position estimate. As mentioned, this position estimate has been shown to be accurate within 3 metres.

7.2.2 Non-Functional Requirements

N1: Fingerprint scans should be taken over a 30 second period. The app in its current state takes 4 scans of the environment with 7 second intervals, resulting in a total of 21 seconds to fully scan a reference point. The time was chosen somewhat arbitrarily; the main intent was that multiple scans are used at each RP instead of a single one to remove some of the fluctuation in signal strength from one moment to the next. This requirement has been met.

N2: The user should not require any extra equipment besides their smartphone. This requirement has been met. The entire functionality for creating the radio map and then also using it to generate a position estimate is contained wholly in a single app. No extra software or hardware is needed to use the system.

N3: The program should update its estimated position every 15 seconds. Like F4, this requirement pertains more to a commercial implementation of such a system. The user can update the position estimate whenever they want by pressed the position button again, however, no continuous update to the position is implemented as it would not facilitate the research being conducted in this project in any way. That is not to say that such a feature is not possible, this feature somewhat exists already in the fact that multiple scans are taken when creating the radio map.

N4: The program should support Android devices running a minimum of Android 12. The app supports a minimum of Android SDK 24 which is Android 7. Testing of the application has taken place entirely on an Android device running Android 10. This requirement has been met.

N5: Users should be able to easily interact with the app. As mentioned in the specification, this requirement is of less importance due to the nature of the project. The user experience was not a priority and the implementation of the UI was focused for testing first. That being said, from a subjective standpoint, the UI is not cluttered, every interactive item is clearly defined in its function and the app performs smoothly.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

The literature review at the start of this project covered various possible alternatives to providing a location service without the need for GPS, many of which already had real-life applications or solid proof-of-concept ideas. One of the key points learnt having undertaken this project is actually how simple but powerful fingerprint-based positioning can be. Even with a simple classification algorithm such as KNN, we were able to acquire a position estimate to within 3 metres, achieving a similar level of accuracy to GPS with just one scan and only twelve reference points. In addition to this, the system can be made much more sophisticated with relatively simple additions; simply switching from KNN to WKNN reduced our maximum positioning errors by roughly one metre and tightened the spread of results. The entire system as a whole has been shown to be highly flexible, providing great results in two different settings.

8.2 Future Work

There are still some improvements that can be made to the application as it is. In its current state, the `onReceive()` callback function is called whenever a WiFi scan is completed system-wide. This can lead to very rare cases where the logic of the app momentarily breaks during the offline phase, causing two separate scanning events to occur simultaneously. This can be resolved through more rigorous checks of the data, such as checking the timestamps of incoming scan results.

As has been mentioned several times, the project has been designed and implemented to be as modular as possible. This means adding extra features such as another distance measure is as simple as defining it in the `Measures` enum class. Implementing an even more complex version of a NN algorithm, such as the Self Adapting WKNN [14] can also be easily done by extending the `Positioning` class and overriding the `calculatePosition()` function.

In a more general sense, if this app was to be extended, the most realistic next step would be moving the radio map to a server. This would then allow multiple users to interact with the positioning system. In order for there to be some actual utility to the user, the app should be extended so that it uses the user location to provide a navigation service. By being able to determine where the user is located, a pathfinding algorithm is able to determine the best path to whichever destination; a satnav for a hospital, conference centre or university campus.

The positioning system need not be used for navigation either; a system could have beacons placed on objects of importance and, using the positioning system, ping to a central server its location in case anyone was looking for it. Such a system could prove to be useful in a product management scenario such as in a warehouse or a laboratory.

The main drawback to fingerprint positioning is the extremely tedious process of constructing a radio map. Therefore, methods on how this process can be streamlined could also be looked into. If the radio map has been moved to a server, a potential solution would be to use user scans (that is, users who aren't the administrator(s)) to add to a radio map and refine it, effectively crowd-sourcing the task of contributing to the radio map. This method would need some way of having the user confirm their location, such as scanning a QR code at a known position and then taking a scan. Another possible solution to this problem would be through the use of an autonomous drone with WiFi scanning capabilities but, perhaps, that would take the scope of the project away from positioning systems and towards machine learning exploration strategies.

Overall, designing and implementing the fingerprint-based positioning system in the flexible and modular way that it has allows for a wide range of different use cases and applications.

References

[1] E. Aitenbichler and M. Muhlhauser. An ir local positioning system for smart items and devices. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 334–339, 2003.

[2] Isaac Amundson, Janos Sallai, Xenofon Koutsoukos, and Akos Ledeczi. Mobile sensor waypoint navigation via rf-based angle of arrival localization. *International Journal of Distributed Sensor Networks*, 8(7):842107, 2012.

[3] Grigorios Anagnostopoulos. *Addressing crucial issues of indoor positioning systems.* PhD thesis, University of Geneva, Switzerland, 2017.

[4] Lu Bai, Fabio Ciravegna, Raymond Bond, and Maurice Mulvenna. A low cost indoor positioning system using bluetooth low energy. *IEEE Access*, 8:136858–136871, 2020.

[5] Chaimaa BASRI and Ahmed El Khadimi. Survey on indoor localization system and recent advances of wifi fingerprinting technique. In *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*, pages 253–259, 2016.

[6] Roberto Casas, David Cuartielles, Alvaro Marco, Hector J. Gracia, and Jorge L. Falco. Hidden issues in deploying an indoor location system. *IEEE Pervasive Computing*, 6(2):62–69, 2007.

[7] Hsieh-Chung Chen, Tsung-Han Lin, H. T. Kung, Chit-Kwan Lin, and Youngjune Gwon. Determining rf angle of arrival using cots antenna arrays: A field evaluation. In *MILCOM 2012 - 2012 IEEE Military Communications Conference*, pages 1–6, 2012.

[8] Andrei Cramariuc, Heikki Huttunen, and Elena Simona Lohan. Clustering benefits in mobile-centric wifi positioning in multi-floor buildings. In *2016 International Conference on Localization and GNSS (ICL-GNSS)*, pages 1–6, 2016.

[9] Ninh Duong-Bao, Jing He, Luong Nguyen Thi, and Khanh Nguyen-Huu. Analysis of distance measures for wifi-based indoor positioning in different settings. In *2022 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, pages 1–7, 2022.

[10] Ninh Duong-Bao, Jing He, Trung Vu-Thanh, Luong Nguyen Thi, Le Do Thi, and Khanh Nguyen-Huu. A multi-condition wifi fingerprinting dataset for indoor positioning. In Ngoc Hoang Thanh Dang, Yu-Dong Zhang, João Manuel R. S. Tavares, and Bo-Hao Chen, editors, *Artificial Intelligence in Data and Big Data Processing*, pages 601–613, Cham, 2022. Springer International Publishing.

[11] National Coordination Office for Space-Based Positioning Navigation & Timing. Gps accuracy, 2022. `https://www.gps.gov/systems/gps/performance/accuracy` [Accessed: December 2023].

[12] National Coordination Office for Space-Based Positioning Navigation & Timing. Gps timing, 2022. `https://www.gps.gov/applications/timing/` [Accessed: March 2024].

[13] Yanying Gu, Anthony Lo, and Ignas Niemegeers. A survey of indoor positioning systems for wireless personal networks. *IEEE Communications Surveys & Tutorials*, 11(1):13–32, 2009.

[14] Jiusong Hu, Dawei Liu, Zhi Yan, and Hongli Liu. Experimental analysis on weight K -nearest neighbor indoor fingerprint positioning. *IEEE Internet of Things Journal*, 6(1):891–897, 2019.

[15] JetBrains. Kotlin faqs. `https://kotlinlang.org/docs/faq.html#what-advantages-does-kotlin-give-me-over-the-java-programming-language` [Accessed: April 2024].

[16] Mikkel Baun Kjærgaard, Henrik Blunck, Torben Godsk, Thomas Toftkjær, Dan Lund Christensen, and Kaj Grønbæk. Indoor positioning using gps revisited. In Patrik Floréen, Antonio Krüger, and Mirjana Spasojevic, editors, *Pervasive Computing*, pages 38–56, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[17] Jianpo Li, Jun Wang, Fuxin Liu, Songjun Pan, Cong Zheng, Baochun Mu, and Ziqi Dong. The design of rfid localization system for library books. In *Recent Advances in Intelligent*

Information Hiding and Multimedia Signal Processing: Proceeding of the Fourteenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, November, 26-28, 2018, Sendai, Japan, Volume 1 14, pages 68–74. Springer, 2019.

[18] Weichao Liang, Zhiang Wu, Jie Cao, and Jun Gu. Understanding customer behavior in shopping mall from indoor tracking data. In *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*, pages 648–653. IEEE, 2018.

[19] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1067–1080, 2007.

[20] Zixiang Ma, Bang Wu, and Stefan Poslad. A wifi rssi ranking fingerprint positioning system and its application to indoor activities of daily living recognition. *International Journal of Distributed Sensor Networks*, 15(4):1550147719837916, 2019.

[21] E. Mok and G. Retscher. Location determination using wifi fingerprinting versus wifi trilateration. *Journal of Location Based Services*, 1(2):145–159, 2007.

[22] Michael J. O'Neill. Effects of signage and floor plan configuration on wayfinding accuracy. *Environment and Behavior*, 23(5):553–574, 1991.

[23] Lorenz Schauer, Florian Dorfmeister, and Marco Maier. Potentials and limitations of wifi-positioning using time-of-flight. In *International Conference on Indoor Positioning and Indoor Navigation*, pages 1–9, 2013.

[24] R. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Transactions on Antennas and Propagation*, 34(3):276–280, 1986.

[25] Martin Schüssel. Angle of arrival estimation using wifi and smartphones. In *Proceedings of the International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, volume 4, page 7, 2016.

[26] Petros Spachos and Konstantinos N Plataniotis. Ble beacons for indoor positioning at an interactive iot-based smart museum. *IEEE Systems Journal*, 14(3):3483–3493, 2020.

[27] Frank van Diggelen and Per Enge. The world's first gps mooc and worldwide laboratory using smartphones. *Proceedings of the 28th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2015*, pages 361–369, 2015.

[28] Yapeng Wang, Xu Yang, Yutian Zhao, Yue Liu, and Laurie Cuthbert. Bluetooth positioning using rssi and triangulation methods. In *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, pages 837–842, 2013.

[29] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, jan 1992.

[30] Jie Xiong and Kyle Jamieson. ArrayTrack: A Fine-Grained indoor location system. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 71–84, Lombard, IL, April 2013. USENIX Association.

[31] Paul A. Zandbergen. Comparison of wifi positioning on two mobile devices. *Journal of Location Based Services*, 6(1):35–50, 2012.