CM3203: Individual Project

Final Report

Localising a device in a building using known access points

Ieuan Griffiths

Supervisor: Professor Stuart Allen

Moderator: Dr Matthew Morgan

CONTENTS

1 INTRODUCTION

At its core, this project aims to make navigating around Queens Buildings easier to do. This will be accomplished by using RSSI data from nearby Wi-Fi access points to infer a location within them. A key goal of this project is to be able to display the location of a device on a digital map, with some degree of accuracy.

The intended audience for the project would be new students to the school, who have little experience navigating around the building. Existing maps can be complex and hard to navigate, and don't provide real-time updates of location. Motivation from this project stemmed from a personal experience of getting lost in the building late at night, and struggling to navigate the building using physical or virtual maps.

2 BACKGROUND

2.1 PROJECT CONTEXT & STAKEHOLDERS

GPS-based navigation systems are widely used across all sectors, with huge public use both through specialised GPS navigation devices and through smartphone applications. As GPS uses satellites in orbit, it is necessary to have line-of-sight to the sky for good accuracy. If line-of-site cannot be maintained, signals can quickly become degraded as they travel through walls and ceilings. This signal degradation can result in excessive noise, and ultimately very poor precision. This is also the case in areas with poor visibility, such as forests and some city areas.

A good example of this is St David's Shopping Centre in Cardiff. This building covers over 129,500 m^2 of land, and makes up 39% of the retail floor space in the city. This building is covered by a roof throughout, and as such would likely struggle from poor GPS precision. As people become reliant on modern mapping and navigation systems, the need for similar indoor mapping systems that function in a similar manner has become necessary. Google has started to include indoor maps for major buildings in its 'Google Maps' software, however the approach it uses for improving accuracy of GPS for these indoor maps isn't available publically.

The applications of a method of indoor localisation are very broad, and could affect many industries in a significant way. Office buildings, resorts, warehouses, hospitals and schools could all make use of this technology, particularly with the use of data mining tools to analyse data. Resorts could build virtual maps and then use the data to understand how certain people move throughout the buildings, and warehouses could use them to direct warehouse staff to products and racks more effectively.

A particularly interesting application is that of hospitals, where installations of equipment is likely to be an expensive and drawn out process due to approvals and testing. By utilising existing Wi-Fi access points, staff could simply use a virtual map to navigate the hospital – potentially reducing travel time between wards and regions of the hospital. Information on where staff are in building could then potentially be used to

identify the closest doctor / nurse to call should an emergency arise in a certain area. This data could then be mined to identify efficiency improvements for spreading staff across the hospital.

Based on the focus of this project, key stakeholders within the University consist mainly of students – lectures can frequently change destination, sometimes at last minute. Students may be unable to locate an unfamiliar room within the building in a timely manner, costing them learning opportunities and potentially leading to unnecessary absence where the room cannot be found. Faculty may need to work in a variety of different rooms throughout the week, with meetings and teaching potentially being conducted in unfamiliar areas. A digital map could potentially help them to navigate the building with greater ease, and improve efficiency.

2.2 CONSTRAINTS

I will focus on navigating around on a small area of Queens Buildings, rather than the entire building. This should provide a good proof of concept for the technique, without wasting time testing over a very large area of the building. We will also seek to obtain a 2D location for a section of the building, rather than trying to work out which floor the device is on. Whilst this should be reasonably simple to accomplish using built in accelerometer data to obtain the altitude or grouping the APs into floors, this would again detract from the focus of the project, and require a significantly increased amount of testing. In terms of accuracy, the project is likely to produce only coarse location data. The reasoning for this is explained further on in the report, but is largely due to noisy data and reflected signals.

This project will only focus on a single test device, allowing for future scope to test the technique on a variety of devices and investigate the technical challenges that arise from this.

A very basic plan for carrying out the project is as follows:

- Implement a method of obtaining raw RSSI data for nearby access points
- Visualise and analyse this data to better understand the noise and fluctuations
- Obtain or devise a method to convert a RSSI value into an estimate of distance from the device to the AP
- Filter or smooth the distance data to make it less noisy
- Use the locations of APs and respective distances to trilaterate the device location

2.3 ASSUMPTIONS

For the purposes of this project, I will assume that all mobile devices can provide useful (i.e. raw in dB / dBm) RSSI data that we can access at application level. I will assume that all access points are fixed in location, and are always online with a constant Tx Power level. I will also assume that each mobile device has the same antenna, with the

same sensitivity. These assumptions may need to be adjusted as the project goes on, and this is detailed later in the report.

2.4 SUCCESS CRITERIA

To aid with evaluating the project, I have chosen 3 success criteria for the project. These are:

1. For the accuracy of the finished to system to be within 5m of the actual distance.
2. For the system to run from a mobile device, without any additional physical hardware necessary
3. For the system to be able to handle moving devices as well as stationary ones

2.5 EXISTING ALTERNATIVE SOLUTIONS

2.5.1 ACTIVE RFID TAGS

Passive RFID tags are used extensively throughout many industries, due to their low cost and fast read times. A user can hold a reader (a phone, or a specialised device) within around 30cm of the tag, and retrieve a small amount of data stored on it. It is possible to obtain the distance from the device to the reader through calculating round trip times, and working out a signal strength from that data. Due to the short read distance, this wouldn't be applicable to this problem. However, active RFID tags differ from passive tags due to a constant power source, rather than one powered by the reader. These tags generally have a much higher read distance which could make them a viable option, with many of these tags placed frequently throughout a building. However, these tags are significantly more expensive (around 10x according to industry sources) than passive tags, so this could be cost prohibitive to cover an entire building.

2.5.2 IBEACONS

iBeacons work in much the same way as Active RFID tags, but have the added advantage of transmitting dynamic data such as distance and power rather than a static chunk of data. iBeacons seem to be in a similar price range to active RFID tags, and therefore suffer the same cost-prohibitive issues. It is also important to note that these devices no longer seem to provide granular distance data, instead offering more discrete versions of distance, e.g. 'near' or 'far'. This would likely make any sort of accuracy impossible to achieve, depending on what sort of interval this data would change at.

2.5.3 BLE BEACONS

Existing solutions in the form of BLE [Bluetooth Low-Energy] tags have been widely explored in academia and in practice. Much like the former solutions, this requires a large investment for the organisation both in upfront installation & purchase costs, and ongoing maintenance. These tags work in much the same way as APs in that they send out beacons at regular intervals, from which we can obtain a signal strength. These beacons are specialised devices that specifically provide proximity data, and fit the

needs of the project very well – however utilising existing network infrastructure would be significantly for suitable for many buildings.

2.5.4 On-Site GPS Repeaters

An interesting solution in receiving GPS signals via antenna and repeating this across a building has been developed by a US company, which appears to be very popular across industry. It is notable however that this is designed for open warehouse spaces rather than that of an office block, with several repeaters likely required for a complex environment. It is likely that the cost would be a significant expense for an organisation where indoor location positioning isn't essential, and as such doesn't provide a viable solution to the problem.

3 APPROACH

3.1 INITIAL APPROACH

3.1.1 PLAN OF APPROACH

My initial approach was to use a mobile device to scan for nearby WIFI access points, and then feed the data from these scans into an app. This data would include a BSSID for the access point and an RSSI value for signal strength. This data would then be fed back to a server, where some sort of filtering would take place to filter out the noise. This server script would then select three known access points from the filtered data with the strongest RSSI, and calculate the distance for each one. Based on this distance data and the information stored about the location of each access point, a location for the device could then be estimated and returned to the app (See Figure 1). The app would then display the location on a map, based on the coordinates provided. The app would continuously scan for WIFI networks, feeding this data in the background to the server. This would then allow the location to be updated continuously, with accuracy of location likely improving with more data fed into the filter.

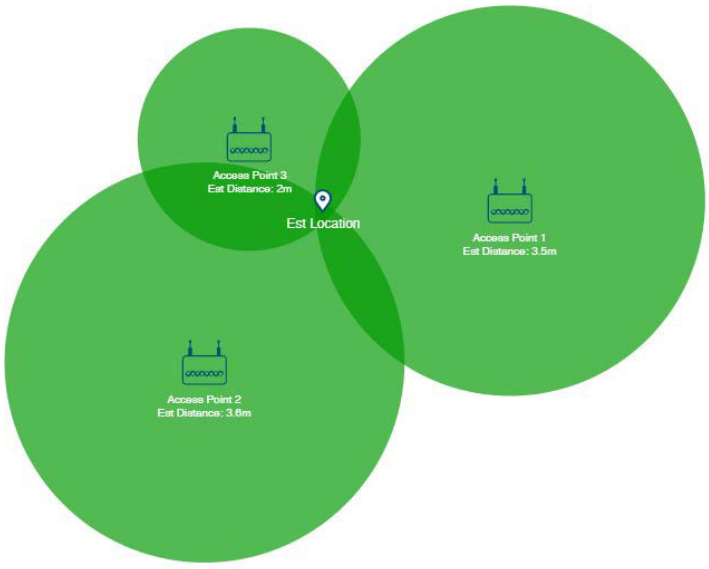


Figure 1 – Example of proposed location finding through trilateration given 3 known access points

3.1.2 IMPLEMENTATION

To gather initial data, I developed a simple application using the Ionic Framework which scanned for nearby WIFI networks. I utilised the *WifiWizard2* library, which exposed simple methods for obtaining this data. Despite many adjustments to the code, it ultimately transpired that iOS requires special permission and approval for any app wishing to scan for WIFI networks [1]. In addition, it appears that there is no way to do this in the background without user actions. Users would need to navigate to the setting

app to scan, with the limited data provided to the app when the user returned to it. This led me to focus on Android, which did allow this functionality to be implemented.

The library allows for scans to be requested as frequently as necessary, so I initially obtained this information every 50ms for 5 minutes. I then dumped this data out as JSON to allow me to manually transfer it for analysis. Viewing the data, I noticed a significant quantity of duplicated readings close together. This would indicate a stable RSSI signal (which would be very unexpected), however upon further inspection of the data an epoch-like timestamp is attached to each scan result with the time it was last updated. Checking this timestamp, I noticed very few of the scan results were actually unique – it appears that a cached result is returned for repeated requests within a certain period of time. Google confirms this in the documentation for *WifiManager,* the API used in the *WifiWizard2* library to return WIFI scanning data. Foreground apps can only obtain new data 4 times in a 2-minute period (once every 30s) and background apps may only obtain new data every 30 minutes [2]. This heavily reduces the practicality of obtaining large amounts of data to filter, however I decided to collect as many results as I could over a longer period of time in order to produce some sort of meaningful dataset that I could then use to obtain a location from.

For early testing a Jupyter notebook with pandas was used to analyse and manipulate data, in order to keep a record of problem solving throughout the project. After loading the scan data from a json file into a Pandas frame, the RSSI data is visualised with a scatter plot. As shown in Figure 2, the RSSI data is spread across the -40 to -32 levels. A histogram of this data shows the frequency of certain signal levels, with no clear level at which it seems strongest. However, simply taking the most common signal level isn't enough to provide a reliable stable value – some sort of filtering must be employed. After research, I opted for a Kalman Filter due to the availability of Python libraries to simplify implementation, and for the lack of resources it requires to work. Updating the filter doesn't require a history of all the data that has run through the system, only the new result and the previous ones. Given the large amount of data the server could be handling for one client, it made it sense to minimise the amount of resources the filter uses so that it can scale up for many users. I used a Python port of the KalmanJS library[1], which simplified the process significantly. For the purposes of this project the Kalman Filter is seen as a 'black box', and the mathematics behind it won't be explored. The filter is just a mechanism to make the RSSI data less noisy. Figure 3 shows a segment of the data before filtering, and after filtering. Note the significant reduction in variation between readings, with each result taking into account the prior when smoothing.

[1] KalmanJs - https://github.com/wouterbulten/kalmanjs

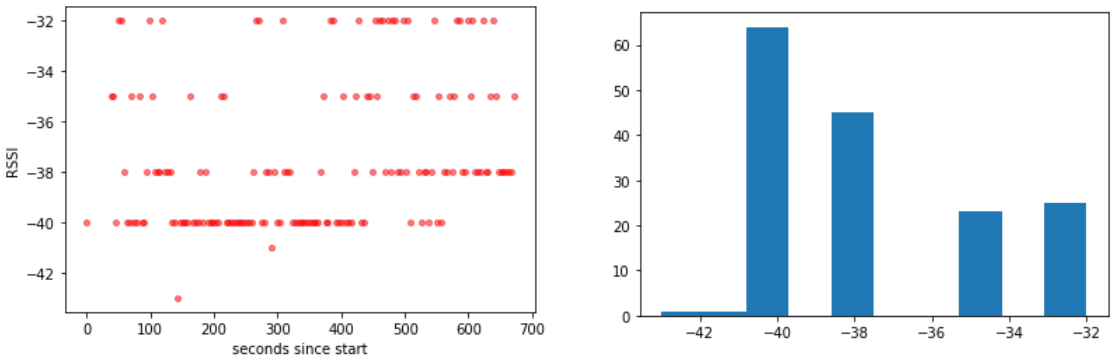


Figure 2.1 and 2.2 – RSSI data scattered across time, and a histogram of the same data

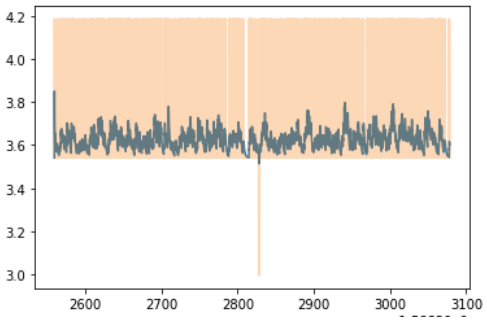


Figure 3 – RSSI data before and after filtering. Orange – unfiltered, Blue - filtered

3.1.3 ISSUES WITH APPROACH

After experimenting with data collected at several intervals, I found that the device would often not provide consistent enough data between readings. Whilst one scan at 1m may suggest a filtered RSSI of -40, another may jump to -50 or drop to -30 on different occasions. Whilst this could potentially be accounted for with extensive readings to calculate a more accurate figure, a major issue lies with the device giving readings of a stronger signal at a further distance. At 2m a reading of -50 may be achieved, but 3m may give a reading of -45 – indicating that the device must be closer. There are many possible reasons for this, with a strong possibility being the access point dynamically increasing and decreasing power based on usage. Given the slow rate that results are obtained at (10 minutes for 20 readings) there is potential that the power may be altered during this time in such a way as to dramatically affect the RSSI.

With the throttling of scan data proving a major hurdle for the project, it became clear that an alternative approach was required to allow the project to achieve some meaningful results.

3.2 REVISED APPROACH

3.2.1 ISSUES RELATING TO CORONAVIRUS

As the preparation for a revised approach was underway, the Covid-19 pandemic started to disrupt the project. With the core of the project relating to navigating around a specific busy building, it was unlikely that the project could achieve the desired goal of accurately navigating around the building using existing infrastructure. Due to this, a decision was made to revise the project slightly. Instead of navigating around a section of Queens Buildings, the aim would be to identify the location between 3 access points in a section of my house. Whilst this is significantly less impressive than the original project aim, it still allows for a proof of concept to be demonstrated, and for accuracy to be evaluated.

This adjustment introduced a new problem – the lack of access points. I obtained several access points cheaply, which would be placed around the home to broadcast a network. However, due to cost constraints all of these were purchased second-hand, and at minimal cost. This meant that none of the access points were the same, with each one having different transmission power and hardware specifications. Many of these access points were locked-down CPE routers, which meant that very few settings could be adjusted and very little information existed openly for them.

3.2.2 UPDATED HARDWARE

After considering ideas for how signal levels could be evaluated, I arrived at utilising a Raspberry Pi with an antenna attached to collect data. By using an external WIFI adaptor with an antenna, I hoped to be able to obtain higher quality and more consistent data than using the built-in antenna in a mobile phone. The choice of using a raspberry pi was purely based on the convenience of the device running Debian Linux, and being compatible with many of the existing network testing tools (TShark, aircrack-ng etc). Whilst the device would be able to work wirelessly[2], I connected to a laptop via Ethernet and a USB cable for power and connectivity purposes whilst testing. This provided a stable and simple setup for testing, and minimised the time spent setting up the new hardware.

3.2.3 COLLECTING RSSI DATA

Initially, I considered using the `iwlist` command-line utility to scan and retrieve information about nearby access points. However, upon testing this command took just over 2 seconds to execute, meaning a maximum of 30 scans/minute. Whilst this is a considerable increase from the throttling on android, it's still a bit too low for the high volume of data I'm looking for.

In the hope of viewing low-level data from the scans and increasing the scan frequency, I tried airodump-ng, part of the Airocrack-ng suite of tools. This tool was considerably

---

[2] Whilst the device does require power and a network connection, this could be achieved using an additional WIFI adaptor to connect to the internet, and a battery or power bank to provide power

faster than expected, capturing updating several times a second. This worked well and allows me to view real-time RSSI data from nearby access points from the terminal, and showed a beacon count of how many times the signal had been updated. However, this only allowed me to export data in .cap format for use in the other tools in the Aircrack-ng suite. Other than somehow scraping the data from stdin, this didn't give me any way to obtain the realtime RSSI data I required.

After some research and pointers, I found that tshark offers the ability to capture 802.11 frame data and export it in a more useful format (such as .csv). I captured data over a period of several minutes, and loaded the resulted dump into wireshark for analysing to see what data it contained. After filtering for `type.type_subtype = Beacon` to only view beacon packets, I was able to look through them to see if they contained any useful data. I found that they contained a value for RSSI under `radiotap.dbm_signal`, which fitted my needs perfectly. Each frame also included an epoch time under `frame.time_epoch`, so I was able to obtain the time the beacon was sent with some accuracy. See Appendix 1 for the contents of the beacon frame. It seems that tshark was able to capture a significant amount of beacon frames in a short space of time – around xxx in the y minutes I ran the command for. This was a huge increase compared to all of the prior approaches, and should provide the necessary volume of data.

Whilst viewing this data in wire shark was useful for exploring the frames, I needed to export data in a format that I could then load into a Pandas frame. Tshark has a built in utility for exporting to CSV, so I utilised this with capture filters to write the BSSID, epoch time and RSSI to a file whilst scanning. The command used and sample of the output is shown in Appendix 2.

After loading this data into Pandas and generating the same visualisations as used previously, I found that the data seemed to more strongly favour a single RSSI level, as evidenced by the darker areas of the scatter graph. This is a good indicator that the new hardware is generating higher quality data, and having a significantly larger amount of data to work from in a shorter period of scanning seems to contribute to this.

3.2.4 Converting RSSI to Distance

I opted to use the simplified equation found in the 'Indoor Positioning Algorithm Based on the Improved RSSI Distance Model' paper. [3] This equation is as follows:

$$\overline{RSSI} = -10n \log(d) + \bar{A}$$

Where $\overline{RSSI}$ is the estimated RSSI, n is a parameter for signal degradation that depends on the environment, d is the distance between device and AP, and $\bar{A}$ is the average measured RSSI at 1m from the AP. Rearranging, we get:

$$d = 10^{(\frac{\bar{A}-RSSI}{10n})}$$

Based on this formula, I needed to choose a suitable value for n and calculate $\bar{A}$. Many implementations of this formula use 2 as a value for n, and so this is what I initially used to calculate the distance. $\bar{A}$ was simply the average of a large (I used 2000+) sample of RSSI readings @ 1m. Due to each of my routers having different configurations, power

levels and hardware, I decided to measure the $\bar{A}$ value for each of the routers individually. I found a non-negligible difference between these readings, indicating that each router may give very different RSSI values at greater distances.

Initially I tested this formula using data collected at 2m, but received very unreliable results. After trying to manipulate values for $\bar{A}$ in the formula, I compared the RSSI values between tests, and found that they fluctuated significantly. After trying the experiment again at greater distances, I found a stronger correlation between RSSI and distance. This is shown in Figure 4.1. Whilst the distance was now increasing with a weaker RSSI signal, it didn't seem very accurate. I found that when trying to find the distance to access points, the device wouldn't sometimes show one correct value for distance, but upon changing location, this would have an incorrect value once again. I once again tried to manipulate the $\bar{A}$ value in order to achieve more accurate results, but this only caused the data to shift by a constant value. After a significant amount of experimentation, I finally remembered that the n value can be adjusted to account for signal degradation. Increasing this to 3 greatly improved the accuracy of the distance conversion. See Figure 4.2 for the comparison.

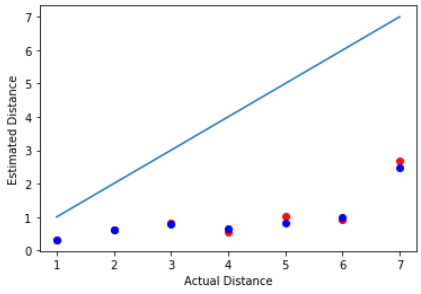


Figure 4.1

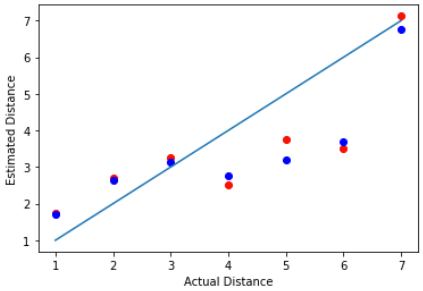


Figure 4.2

3.2.5 Estimating Device Location Theory

Now that I was able to obtain the distance from the device to a number of nearby access points with some accuracy, I needed to then use this data to estimate the location of the device. Initially, this seemed like a reasonably simple task. I intended to treat each access point as a circle, with the centre (x_1, y_1) being the location of the access point and the radius r being the estimated distance. This can be represented by the formula

$$(x - x_1)^2 + (y - y_1)^2 = r^2$$

By using this equation for each of the circles and solving the equations simultaneously, it should be possible to obtain a shared point of intersection a between all 3 circles (see Figure 5.1). However, this requires all of the radii to be very accurate and offers no margin for error. Should one of the radii (distance from AP) be incorrect by any margin, the three circles would no longer intersect at a common point [4] (see Figure 5.2)

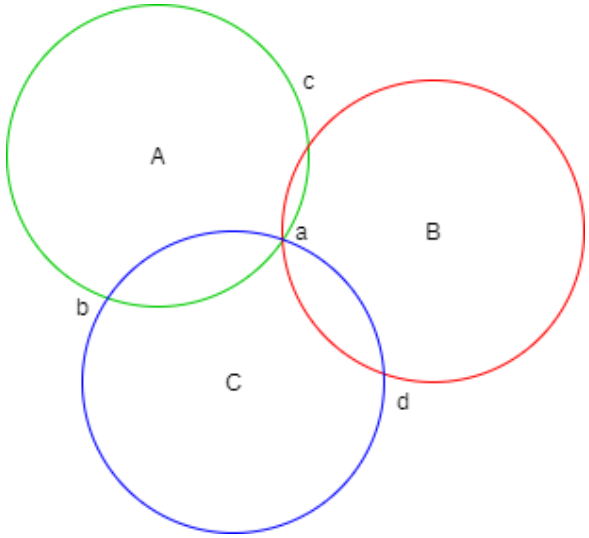


Figure 5.1

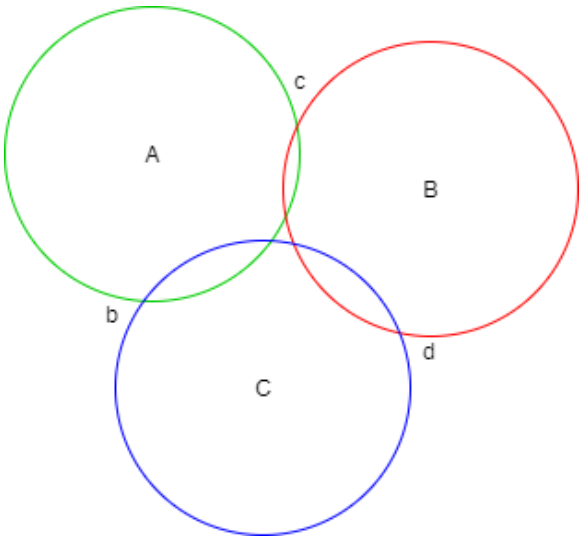


Figure 5.2

This creates a significant problem, as I know that despite filtering, the data still contains some noise and variation. Rather than try to somehow remove all error from the data, I decided to accept that the data would never be perfectly accurate, and consider this in my approach to estimating the location. I began looking at all of the possible arrangements of 3 circles. There are 14 possible arrangements, assuming that they do not all meet at a point (as was the case in Figure 5.1). These 14 arrangements are shown in Appendix 3 for reference. I initially set about writing an algorithm to find points of intersection and to find overlaps between circles, but given the variety of ways these can be arranged this proved a fruitless endeavour. I eventually considered the problem from a geometric point of view rather than a mathematical one. I focussed on the diagram shown in Figure 6. This shows three circles overlapping rather than touching, with the intersection of all circles bounded by points of intersection a, b, c. This intersection forms a complex shape with curved edges; however, this can be simplified into a triangle abc. Given all shapes overlap here, we know that the device lies somewhere within it. From this triangle, we can then obtain a centroid point, which will serve as the best estimate of user location.

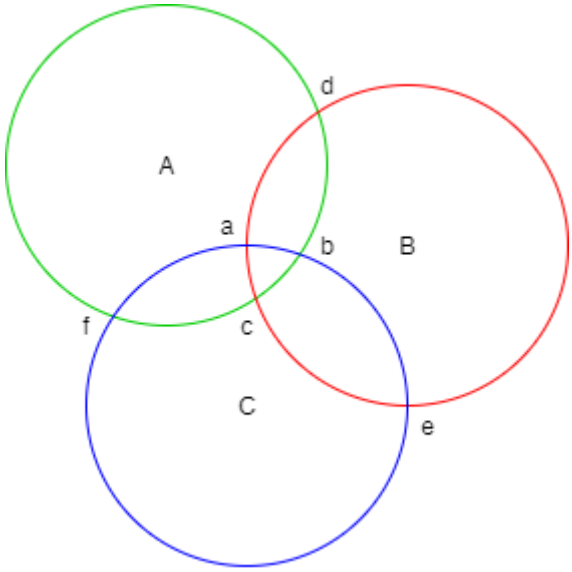


Figure 6 – Three access points overlapping with points of intersection at a, b, c, d, e, f

This centroid isn't the exact location of the user, we just know that the user lies somewhere within the complex shape. In order to determine how accurate this location is, we can use the area of the complex shape. This can be complex to obtain, with solutions including quadtree approximations and random sampling through Monte Carlo simulations. [5] Fortunately, many geometric libraries are able to do this with ease. From the area, we can convert this into a circle at the centroid, with radius $r = \sqrt{\frac{A}{\pi}}$.

This radius is equal to the uncertainty of the location based on the centroid. It is important to note that this accuracy does not include errors stemming from the RSSI data or distance approximations, it is merely a calculation of the uncertainty introduced by assuming the device is in the middle of the intersection.

This calculation works very well for the exact arrangement of circles shown in Figure 7, however there are still 13 other arrangements to contend with. This arrangement is the only one where we can state with any reasonable certainty that the user lies within an intersection, as all other arrangements would ignore at least one of the circles. Whilst it would be possible to calculate a midpoint between two points of intersection should only two circles overlap, this doesn't include the distance from the 3rd access point, potentially placing the user in the wrong place. It is for this reason that I decided to only estimate the device location using a 3 three circle overlap, and to manipulate all other arrangements until they follow this rule. This is done by iteratively increasing the size of the radius by 25% until a region is intersection between all three is identified. Whilst this decreases accuracy on each iteration, we can assume that the distances given by the access point were not accurate in the first place, due to the lack of overlap. By increasing the size of the radius by a percentage rather than a constant static value (such as adding 1m each time), each circle increases in size relative to its original size, retaining the ratio between circles. Once the 3 circles overlap, the location can be determined and accuracy calculated as above.

3.2.6 Implementing Device Location Estimates

I initially focussed on plotting circles with the calculated distance from AP as the radius. Matplotlib has a built-in patches library for drawing shapes, so this was relatively simple to do. Initial results proved very positive, with the actual device location residing in the overlap area as expected. This is shown below, in figure 7.1.

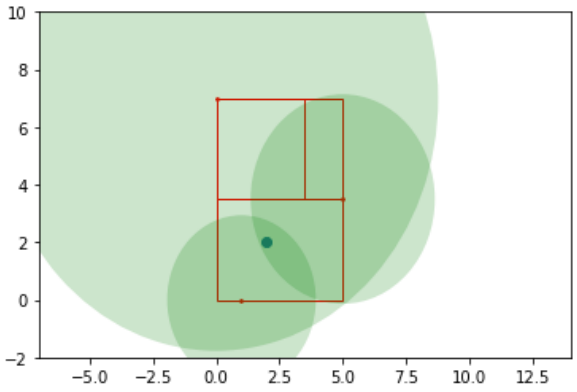


Figure 7.1 – The actual device location is shown as the blue dot. The rooms are outlined in red, with the access points indicated by the small red dots on the walls.

To implement this location estimate, I used the library 'shapely'. This library includes classes for many geometric shapes, and the spatial relations between them. For each of the access points, I created a `Point` object with the coordinates of each one. Shapely doesn't include a circle as a geometric shape, but we can add a buffer of length r to account for the radius, and achieve the same thing. For each access point, I added the buffer of the measured distance from AP. This creates 3 circles on a 2D plane, which we can now test to see if they overlap. For testing purposes, matplotlib was used to plot each circle as a PolygonPatch. The output of this shown below in Figure 7.2.

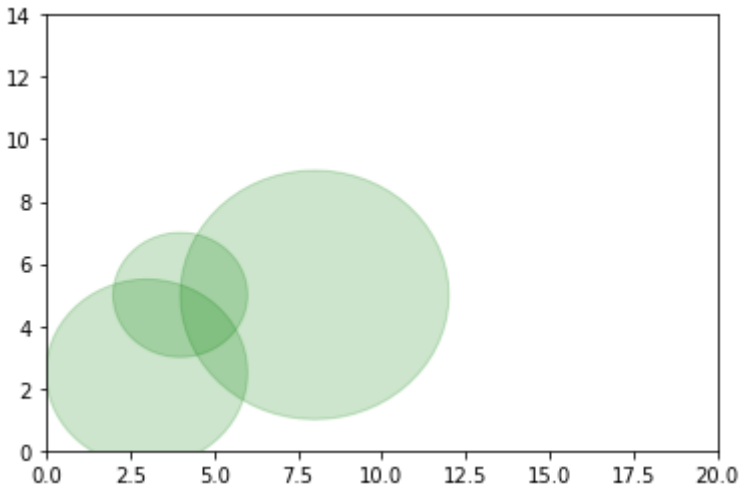


Figure 7

In order to test whether these three circles overlap, we simply calculate the intersection between two of the circles (e.g. Circle_1 and Circle_2) and then check if this new region intersects with the third circle. This is shown below:

```
all_intersect = Circle_1.intersection(Circle_2).intersects(Circle_3)
```

If this is true, we can then simply calculate the region of intersection between all circles by using the intersection method on each one, and then simply output the centroid using the centroid property. This is shown below:

```
intersection = Circle_1.intersection(Circle_2).intersection(Circle_3)

centroid = intersection.centroid.coords
```

Should the three circles not intersect, we simply iterate through each one increasing the radius by 25% until the three-intersection test has been met. This creates a rippling pattern spreading out from each access point, and due to the relatively small number of iterations required, is almost instantaneous. The step of 25% could be reduced to provide better accuracy, as this would result in a smaller intersection area should all 3 be very far apart. However, if all three were far apart to start with, the accuracy would be low as all 3 *should* always overlap in theory. This is shown in Figure 8.

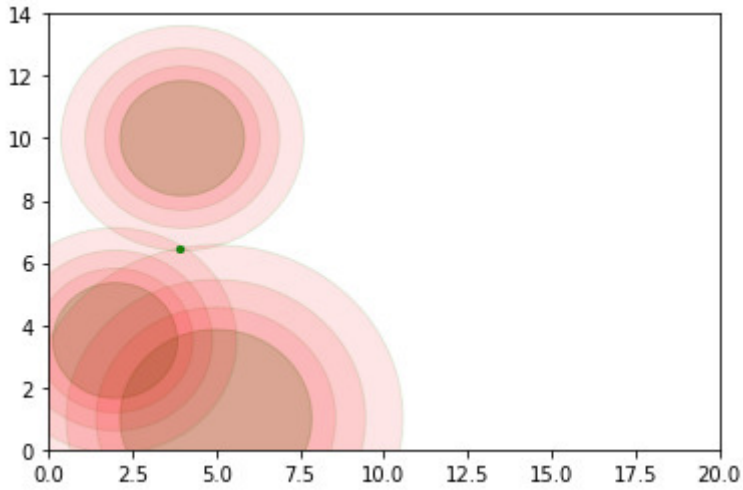


Figure 8.1

The accuracy based on the area of intersection can be trivially obtained using `intersection.area` and the equation found in section 3.2.6.

Using the same data as the example in Figure 7.1, I applied the algorithm to it to try and find the location. The output is shown in Figure 8.2, with a final estimated location of (2.66,1.45). This is only 0.85m away from the actual location at (2,2) which is a very good estimate

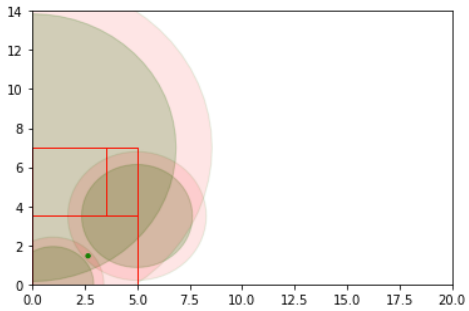


Figure 8.2 – Estimated device location shown as blue dot

3.2.7 Moving From Testbed to Server

In the previous sections I was able to obtain a location for a device using scan data collected into a csv file, which was then imported into a jupyter notebook and analysed. Whilst this works very well for testing purposes, this wouldn't fulfil the project aims by itself. For the location estimation to be automatic, I needed to write a simple server to automatically perform part of the code on demand. For this, I used Flask due to its simplicity and low overheads. I also changed the access point data from a dictionary with many sub-keys to a class system. This produced significantly cleaner code and allows me to abstract away from the complexities of certain parts of the code.

I created 3 simple endpoints, as shown in the table below:

Endpoint	Purpose	Input / Output
/scan-data [**POST**]	To input RSSI data into the server, collected from access points.	**Input**: JSON array of dictionaries with keys 'timestamp', 'bssid' and 'level'
/get-location **[GET]**	To obtain a location based on the data the server has received	**Output**: JSON dictionary with keys 'location' and 'accuracy'. 'location' is a tuple of the form (x, y).
/data-collection-mode **[GET] [POST]**	To control whether the device collects data or not. This is specific to using a headless raspberry pi, as I'm unable to control it unless via SSH, but with other devices the data collection mode could be controlled by the user.	**Input & Output**: Boolean value on whether to collect data or not.

On the device, a python script checks the data collection mode on the server every two seconds, before sleeping and uploading the data collected in that period if there is any. If the mode has only just been switched on, the script spawns a tshark process in the background dumping data to a CSV file. The script then iterates through the rows of this file every two seconds, checking for any that it hasn't read yet. If it finds any, it then processes this data, ensures it's complete and then adds it to a list of data to be sent to the server.

Should the data collection mode be switched off, the script would then terminate the process and check every 2 seconds until it's switched back on. A flowchart for this script is shown in Appendix 4.

3.2.8 Displaying Data on Map

For displaying the location, I chose to use a simple map made of rectangles drawn on a canvas. A canvas can be a very powerful element, so more complex maps could also be implemented if necessary. A simple JavaScript interface can be used to draw on the canvas, which allows us to simply iterate through the coordinates of each room, and draw out the rectangles. One negative part of the canvas JavaScript module is the lack of scaling features. The only available unit is pixels, so this means that every position and distance used previously in m has to be scaled up to fill the screen. This is easy to solve, but tedious to implement.

Once the rooms were drawn, I needed to obtain the current location of the user. The flask server implemented previously exposes a very simple API, with a get location endpoint. I used an XHR request to fetch this data in the background, and then drew this on the canvas. This produces a crude, but workable map with a clear reference to the user's current location. Given more time this could be made more user-friendly and complex, but for a simple proof of concept it works well. See Figure 9 below for an example of two rooms with the user towards the centre of the northmost room.

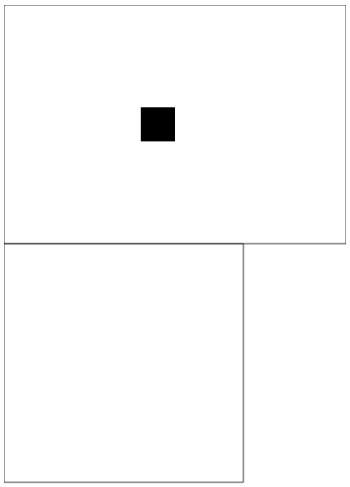


Figure 9

4 EVALUATION & REFLECTION ON LEARNING

Whilst the project did give some interesting data, and provided a reasonably accurate location depending on rooms, I don't believe it was very successful. I set out a rigid project plan at the start, which whilst helpful in the early stages, quickly became redundant due to certain stages taking significantly longer than others. This wasn't helped by the project complexity increasing by changing to a Raspberry Pi when code had already been written for a mobile phone, and also the added problem of using various old routers to navigate from. My original plan was based around only needing to account for a single mobile phone, which led to an unfortunate increase in workload when the Coronavirus pandemic forced this project to change. The majority of the project time was spent trying to obtain better data, trying to account for routers with varying power levels and some not really working at all. This all led to a last-minute push to produce some sort of prototype that would meet the project aims, but in many ways, it falls short of this.

Despite the project not meeting my original expectations, it did provide many learning experiences along the way. In particular, working with circle geometry proved surprisingly difficult, and many days were spent trying to devise some sort of algorithm to detect which of the 14 arrangements they were in. This project also provided a great introduction to noise filtering techniques, and specifically the libraries that can be used to work with statistics / filtering.

5 FUTURE WORK

Whilst this technique has been widely explored and improved upon, some future research may still remain. There is potential for machine learning to be used with large datasets, for the purposes of RSSI-based trilateration. A key purpose here would be noise reduction, where given a large enough training dataset it may be possible to largely remove noise from the data. This could be used in combination with Bluetooth Low Energy proximity to calculate how close several devices are to each other. If several people were standing next to each other in the corridor, both sets of data could be used to calculate a location. This would be particularly interesting given that Bluetooth 5.1 has a new feature called 'Radio Direction Finding', which allows a device to not only work out the proximity to the beacon / broadcaster, but also find out the direction in which the signal has come from. This could allow for a 'mesh' of Bluetooth devices to be created, where each device uses WIFI-based RSSI to find a rough location, and then uses proximity and direction of BLE signals to narrow the location down even further.

It would also be interesting to see whether external data such as CCTV could be used to identify the location of the device in combination with RSSI. If a fixed-position camera could identify the location of several devices / people, then this could be used in conjunction with RSSI to build a 'heatmap' of an area, showing signal strength across a room or corridor. Given enough people walking through an area with WIFI-enabled devices over a long period of time, it should be possible to build some sort of ML system to adjust the RSSI accordingly.

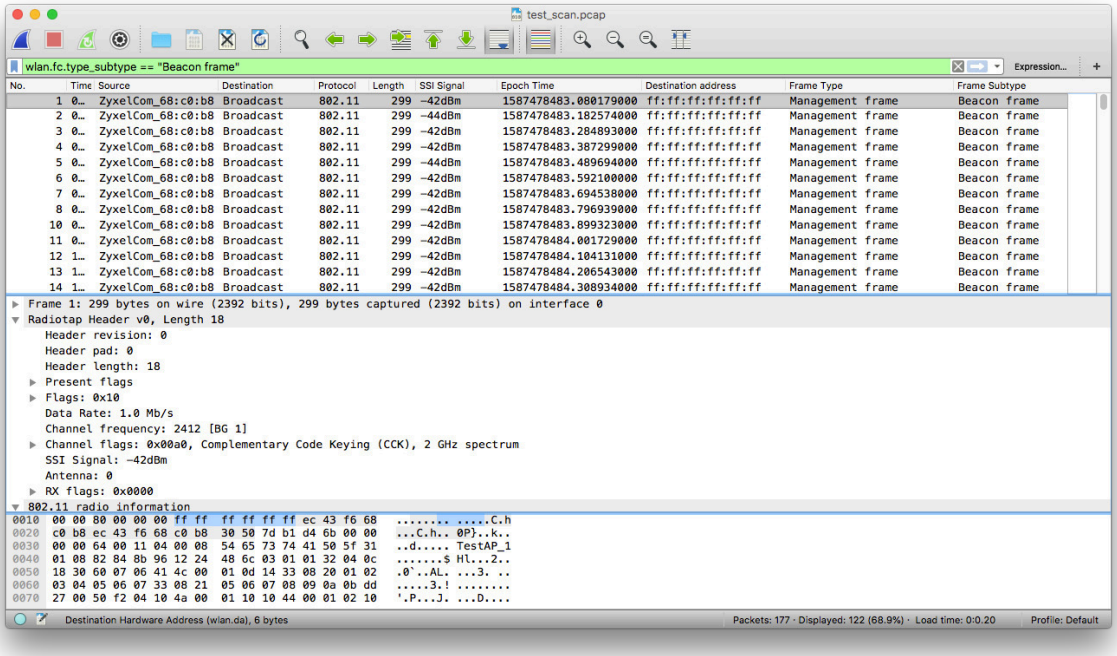
6 CONCLUSION

Whilst the project wasn't overly successful as a whole, it did provide some interesting learning opportunities along the way. Given enough data the system did estimate a reasonable location, and the algorithm using circle intersections seems to work very well. Given the original plan, this project may have worked out slightly more successful, however this was unfortunately not the case.

This project shows that whilst RSSI data can provide a reasonable estimation of location given enough data and good conditions, it isn't reliable enough to be used as an effective method of navigation. A large amount of the project also relies on the device being static in open space (such as placed on a table), and does not account for a person blocking the device by standing in front of it. Many devices used for navigation would be held whilst navigating, which is likely to impact the rough estimate of location significantly.

6.1 APPENDICIES

Appendix 1 – Contents of a beacon frame



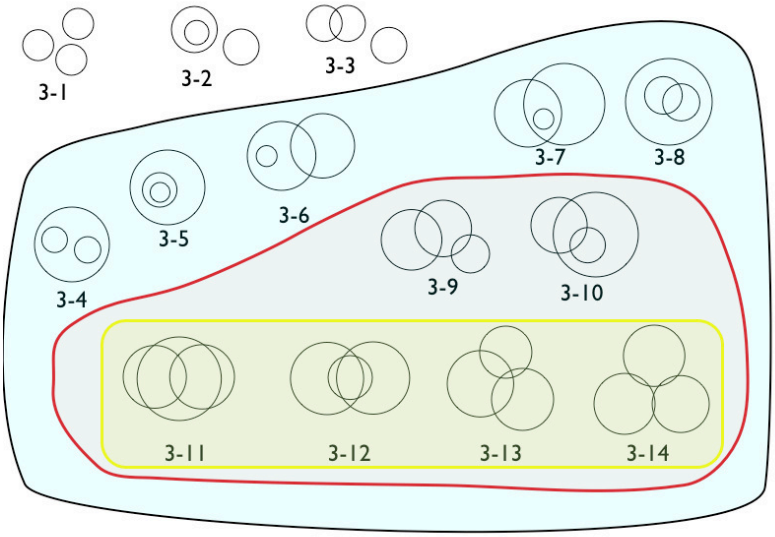
Appendix 2 – Tshark command & output sample

```
tshark -i wlan0 -f "type mgt subtype beacon" -T fields -e frame.time_epoch -e
wlan.bssid -e radiotap.dbm_antsignal -E header=n -E separator=, -E quote=d >
test_file.csv
```

```
"1588340422.137548000","ec:43:f6:68:c0:b8","-42"
"1588340422.239920000","ec:43:f6:68:c0:b8","-42"
"1588340422.342312000","ec:43:f6:68:c0:b8","-44"
"1588340422.444716000","ec:43:f6:68:c0:b8","-44"
"1588340422.547113000","ec:43:f6:68:c0:b8","-42"
"1588340422.649533000","ec:43:f6:68:c0:b8","-42"
"1588340422.751939000","ec:43:f6:68:c0:b8","-42"
"1588340422.854359000","ec:43:f6:68:c0:b8","-42"
"1588340422.956744000","ec:43:f6:68:c0:b8","-42"
"1588340423.059081000","ec:43:f6:68:c0:b8","-42"
"1588340423.161477000","ec:43:f6:68:c0:b8","-42"
"1588340423.263882000","ec:43:f6:68:c0:b8","-42"
"1588340423.366286000","ec:43:f6:68:c0:b8","-42"
"1588340423.468682000","ec:43:f6:68:c0:b8","-42"
"1588340423.571105000","ec:43:f6:68:c0:b8","-44"
"1588340423.673491000","ec:43:f6:68:c0:b8","-42"
"1588340423.775892000","ec:43:f6:68:c0:b8","-44"
"1588340423.878303000","ec:43:f6:68:c0:b8","-42"
"1588340423.980709000","ec:43:f6:68:c0:b8","-42"
"1588340424.083114000","ec:43:f6:68:c0:b8","-44"
"1588340424.185499000","ec:43:f6:68:c0:b8","-40"
"1588340424.287900000","ec:43:f6:68:c0:b8","-42"
"1588340424.390305000","ec:43:f6:68:c0:b8","-42"
"1588340424.492703000","ec:43:f6:68:c0:b8","-42"
```

Appendix 3 - Figure 6 – The 14 possible arrangements of 3 circles. [6]

This image illustrates all 14 arrangements, and is the only well-respected and documented source on this topic easily available.



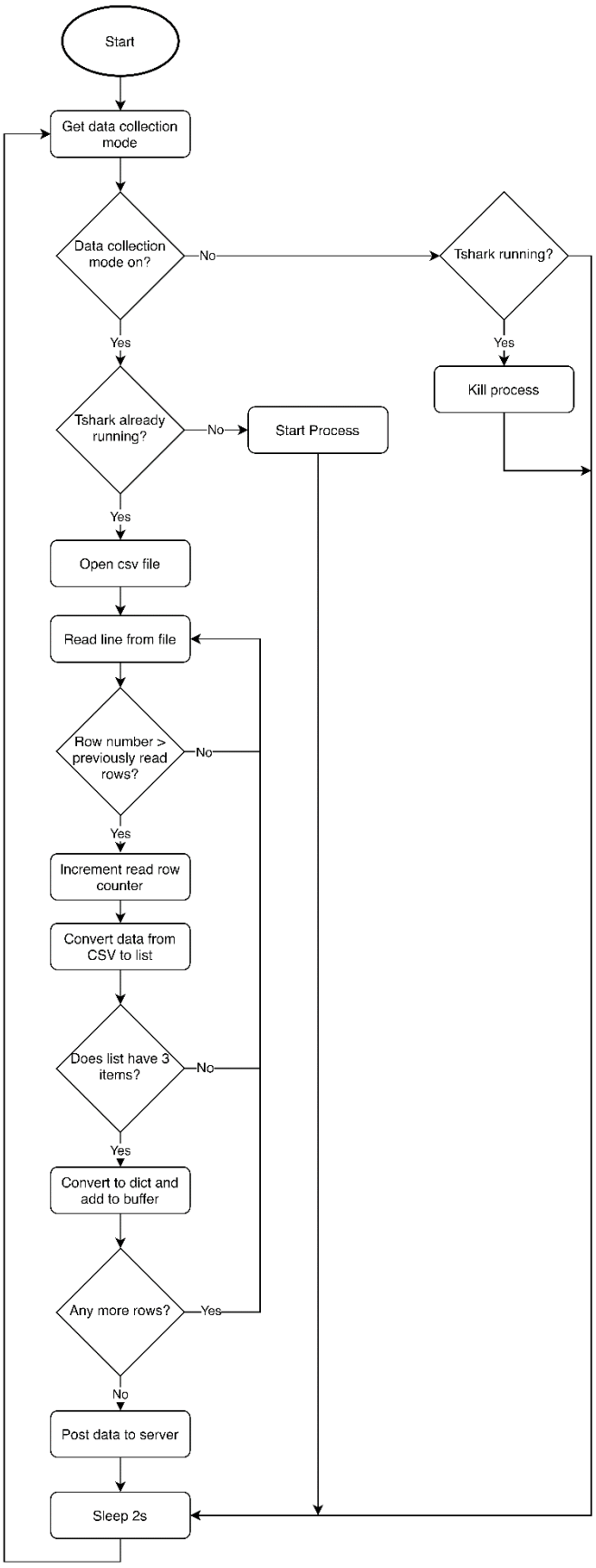
The 14 free arrangements of three circles (A250001), including:
A250001

: the 11 arrangements whose union of discs is connected (A_____)
A275923

: the 6 arrangements whose union of circles is connected (A275923)
A275924

: the 4 arrangements where every circle intersects every other circle (A_____)
A288554

Appendix 4


Start
Get data collection mode
Data collection mode on?
No
Tshark running?
Yes
Yes
Kill process
Tshark already running?
No
Start Process
Yes
Open csv file
Read line from file
Row number > previously read rows?
No
Yes
Increment read row counter
Convert data from CSV to list
Does list have 3 items?
No
Yes
Convert to dict and add to buffer
Any more rows?
Yes
No
Post data to server
Sleep 2s

7 REFERENCES

[1] Apple Inc, “Apple Technical Q&A QA1942,” 14 August 2017. [Online]. Available: https://developer.apple.com/library/archive/qa/qa1942/_index.html. [Accessed 10 May 2020].

[2] Google, “Wi-Fi scanning overview,” 1 May 2020. [Online]. Available: https://developer.android.com/guide/topics/connectivity/wifi-scan. [Accessed 10 May 2020].

[3] G. Li, E. Geng, Z. Ye, Y. Xu, J. Lin and Y. Pang, “Indoor Positioning Algorithm Based on the Improved,” *Sensors,* vol. 18, no. 9, p. 2820, 2018.

[4] M. I. M. Ismail, R. A. Dzyauddin, S. Samsul, N. A. Azmi4, Y. Yamada, M. F. M. Yakub and N. A. B. A. Salleh, “An RSSI-based Wireless Sensor Node Localisation using Trilateration and Multilateration Methods for Outdoor Environment,” 2019.

[5] B. Frederickson, “Calculating the intersection area of 3+ circles,” 19 November 2013. [Online]. Available: https://www.benfrederickson.com/calculating-the-intersection-of-3-or-more-circles/. [Accessed 14 May 2020].

[6] J. Wild, “Number of arrangements of n circles in the affine plane,” 16 May 2014. [Online]. Available: https://oeis.org/A250001. [Accessed 10 July 2020].