Indoor Proximity-based Advertising using Bluetooth Beacons

Finn Zhan Chen
School of Informatics, University of Edinburgh
Edinburgh, United Kingdom
finnzhanchen@gmail.com

ABSTRACT

Bluetooth is a perfect alternative technology for indoor localisation and is highly accurate, cost-effective, can be installed with minimal effort, and supported by many operating systems and devices. Indoor localisation proposes an immense value proposition to retails stores. In this paper, an indoor localisation prototype is built based on Bluetooth beacons' signals. The accuracy of the prototype is evaluated to determine its possible real-world applications, such as location-based advertisements in a shopping mall or supermarket environment.

1 INTRODUCTION

This paper focuses on the full-stack implementation of an indoor localisation prototype entirely based on Bluetooth signals that can estimate the location of an Internet of Things (IoT) device within an indoor environment. Firmware in the IoT device was developed to scan the Bluetooth signals continuously from a selected set of Bluetooth beacons. An Android app is developed to subscribe to the IoT device via Bluetooth and it acts as the gateway between the IoT device and the Cloud [14]. Data received on the gateway are decoded and are sent to the Cloud alongside with a timestamp via WiFi. The Cloud contains a storage of collected data, computed data, data processing scripts and Cloud visualisation dashboard. Localisation scripts are developed to estimate the location of the IoT device given the collected data and results are saved in the Cloud. A real-time visualisation tool was developed for Android. Furthermore, a Cloud visualisation dashboard was developed to visualise the localisation results. Tests were designed and used to evaluate the performance of the indoor localisation prototype, and based on the performance obtained, business use cases for the prototype are discussed.

2 DATA COLLECTION AND CLOUD STORAGE

The prototype environment (Appleton Tower Level 5, School of Informatics, University of Edinburgh) has been pre-populated with 10 non-uniformly distributed Bluetooth beacons shown in Figure 1.

In this prototype, the only data used to estimate the indoor location of the IoT device is the MAC address and Received Signal Strength Indicator (RSSI) of the Bluetooth beacons.

Firmware in the IoT device was developed to collect MAC address and RSSI of lab Bluetooth beacons. An Android app was developed to subscribe to the IoT device and receive notifications about value updates via Bluetooth. The phone running the Android app acts as a gateway (a network node that connects two networks using different protocols together) and posts the data to the Cloud [14] via WiFi.

2.1 Firmware

Firmware for an IoT device, Nordic nRF51, was developed in MBED OS using C++ to scan RSSI and MAC address of lab beacons and update the subscribed Bluetooth device (the phone) via notifications.

Examples [8] [9] were used to familiarise with Bluetooth advertisement callbacks, the Generic Attribute Profile (GATT) server and setting up custom services and characteristics. GATT establishes in detail how to exchange all profile and user data over a BLE connection [13].

2.1.1 Custom Services and Characteristics. GATT services are collections of characteristics and relationships to other services that encapsulate the behaviour of part of a device. GATT characteristics are defined attribute types that contain a single logical value [13]. A custom service and a custom characteristic have been set up.

The custom characteristic contains an array of size 7. Each element is an unsigned integer of length 1 byte. MAC address is saved in the first 6 bytes and RSSI value is saved at the last byte. Combining both data into one characteristic is more efficient as the IoT device only needs to update one characteristic rather than two characteristics.

The custom characteristic has the property of "Notify" [6] so it permits notifications of a characteristic value without acknowledgment from a subscribed device (in this case, the Android app gateway). "Notify" is preferred over property "Indicate" because notifications enable a higher transfer rate.

2.1.2 Bluetooth Scanning. The IoT device continuously scans for Bluetooth devices nearby and the received advertisements are filtered by the installed lab beacon MAC addresses. Continuous scanning ensures more data points are collected for the localisation algorithm in section 4. This is achieved by setting the scan window and scan interval to the same value. More data points result in more accurate localisation.

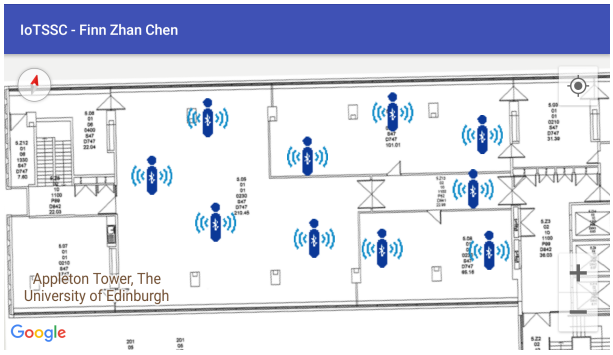

Figure 1: Lab beacons on the floor plan

For every lab beacon advertisement received, the custom characteristic is updated to the received lab beacon's MAC address and received RSSI, and the subscribed Bluetooth device is notified.

Due to the inconsistent signals of lab beacons and interference, some beacons are harder to detect even if the IoT device is close to the beacon. This causes fewer data points for the localisation algorithm to provide an accurate location estimation. In response to the latter, the data time window for which the RSSIs are used for the localisation algorithm is increased to minimise the effect. This is an external challenge and cannot be controlled during the development of the prototype.

2.1.3 Additional Sensors. Additional sensors could be added to the IoT device to provide more accurate localisation. However, due working alone the author decided to finish the full-stack implementation first.

2.2 Gateway Communication

2.2.1 Android App as a Gateway. An Android App has been developed using the template from [3] to subscribe to a Bluetooth device. Then, this template is further customised by the author and acts as a gateway between the IoT device and the Cloud. The app is subscribed to the custom characteristic of the IoT device and is notified of every new value update. For every notification from the IoT device, the app creates a timestamp and decodes the received MAC address and RSSI into strings. These three key-value pairs are posted to the Cloud via REST API with the appropriate authorisation headers [2].

2.2.2 AES-128 Encryption. Security measures have been considered to avoid data breach. Symmetric encryption algorithm such as Advanced Encryption Standard (AES) 128 bits [7] could be implemented for the prototype. AES-128 is less computationally intensive than AES-192 and AES-256 so it would be preferred for a battery sensitive application such as indoor localisation.

There are 2 stages where encryption could be used: the communication between the IoT device and the gateway, and the communication between the gateway and the Cloud. The former would be impractical if the battery life of the IoT device is a big factor as encryption is computationally heavy and reduces battery life. On the other hand, the latter is a suitable scenario for applying encryption.

At the gateway, a private key could be used to encrypt the timestamp, received MAC address and RSSI, and then post these to the Cloud via REST API. When the encrypted data needs to be used, the data on the Cloud can be decrypted using the same private key during the execution of a script and be used for estimating the location of the device.

Unfortunately, due to working alone and time constraint, encryption was not implemented in the prototype, but this is a crucial feature for future work.

2.3 Cloud Storage

The Cloud is used to store, visualise and process data. The Cloud contains:

(1) Collection of collected data: Save data (timestamp, MAC address, and RSSI) from the gateway, and are input for the localisation algorithm.
(2) Collection of estimated locations: Save output (timestamp, latitude, and longitude) from the localisation algorithm.
(3) Cloud visualisation dashboard: Visualisation of the estimated locations.
(4) Real-time localisation Python script: Computes the latest location given the collected RSSIs in the past y seconds where y is the data time window. All beacons in the data time window are considered "discovered beacons" and their proximities are outputted.
(5) Batch processing Python script for localisation: Run the localisation algorithm every x seconds for the RSSIs collected in the past y seconds at the time of the localisation algorithm where x is the algorithm time interval and y is the data time window. Results are saved in the collection of estimated locations.

Collected data are accessed by both Python scripts for localisation. When the Cloud visualisation dashboard is accessed, the batch processing script is called. Then, the Cloud dashboard retrieves the collection of estimated locations and visualise them. The real-time localisation script is called by a real-time visualisation Android app, and Android app visualise the latest estimated location and the discovered beacons' proximity.

3 THE LOCALISATION ALGORITHM

Localisation is the estimation of latitude and longitude of the IoT device. Two Python scripts have been developed to localise the IoT device by using the MAC addresses and RSSIs from the data collection.

Algorithm time interval is the time interval between each consecutive run of the localisation algorithm.

Data time window is the past time window in which the collected data are used for localisation. The higher the data time window, the more data points are available. Note that a higher data time window does not necessarily result in higher accuracy. This is used to discover beacons and their past RSSIs within the data time window to compute their proximity to the IoT device.

The real-time localisation script finds the newest position of the IoT device using the collected data in the data time window at the time of localisation. Also, all discovered beacons' proximity to the IoT device are outputted. This is used for real-time localisation visualisation in section 4.1 and manual RSSI calibration in section 3.1.2.

The other is a batch processing script that computes the path by running the localisation script iteratively for every predefined algorithm time interval on the entire collection of timestamps, RSSIs and MAC addresses given a data time window. Each successful localisation is posted to the Cloud.

The localisation is achieved using the lateration method which is based on knowledge of reference point positions and the distances to them [15].

3.1 Proximity to beacons

Proximity to beacon is crucial for the localisation algorithm and can be found using the distance estimation model proposed by Texas Instruments [4]:

$$RSSI = -(10 \times n) \log_{10} d + A \tag{1}$$

In Equation 1, RSSI is the radio signal strength indicator in dBm, n is the signal propagation exponent, d is the proximity to beacon, and A is a reference RSSI (the RSSI value measured when the IoT device is 1 metre away from the beacon). In free space, n is equal to 2.

3.1.1 Finding reference RSSI. The reference RSSI (A) from Equation 1 have been calculated by collecting over 50 RSSIs at roughly 1 metre from each beacon. The noises from the collected RSSIs has been filtered by removing the outliers using [5], and the remaining RSSIs are averaged to find the reference RSSI.

However, the proximity distance calculated is not accurate. An Android App from section 4.1 was been developed to visualise the proximity calculated from the equation in real time. The problem was that the reference RSSI value was too large, thus requiring calibration.

3.1.2 Manually Calibrating reference RSSI. Manual adjustments to the reference RSSI were made based on the visualisation of the calculated proximity on the Android App from section 4.1. The calculated proximity is visualised on the app as shown in Figure 2 and is compared to the desired proximity. The desired proximity is the distance between the IoT device location on the map and the target beacon which could be estimated using Haversine method [12] by inputting the ground truth location of the beacon and location of the point using [11].

The reference RSSI is adjusted manually such that the difference between the calculated proximity and the desired proximity is minimal. However, a manual adjustment will not result in a global minimum thus an improvement to this process would be to apply an optimisation algorithm. However, due to time constraint and working alone, this was not feasible.

Note that during this manual adjustment, the points are chosen on the map such that there is minimal interference to the target beacon. Manually calibration has been rigorous and tedious. This was by far the most time-consuming and crucial part to ensure an accurate location estimation from RSSI values.

The final reference RSSI value used in the localisation algorithm is displayed in Table 1. The large difference between the reference RSSI before and after manual calibration might be due to the fact that the beacons are not on the same level of height as the IoT device and human errors thus resulting wrong RSSI value estimate at 1 metre in section 3.1.1.

3.1.3 Calculating proximity. Equation 1 can be rearranged into:

$$d = 10^{\frac{A-RSSI}{10\times 2}} \tag{2}$$

RSSI from Equation 2 is the average of the past RSSIs collected from the same beacon within the data time window. Noises from the RSSIs are reduced by removing the outliers using [5]. However, a better and more complex noise removal model such as Kalman filter [1] could be used. However, due time constraint and working alone, the author decided to finish the full-stack implementation first.

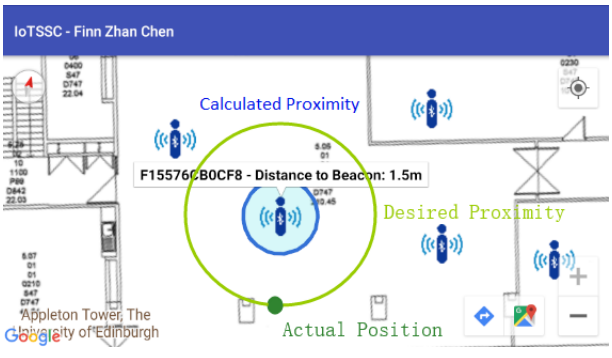

Figure 2: Visualisation of a beacon's calculated proximity and desired proximity to the IoT device

3.2 The Algorithm

The lateration algorithm calculates the estimated location of the IoT device given the exact location of beacons and beacons' proximity to IoT device. Each beacon has a circle with radius equals to its proximity to the IoT device. The lateration makes uses of the intersecting points of the circles to interpolate the location.

3.2.1 3-Dimensional Trilateration. Trilateration [10] has been a popular method to interpolate locations. An existing Python library for trilateration was found [16]. The 3-dimensional space version of this trilateration package (based geographic coordinate system: latitude and longitude) was experimented.

The limitation of this 3D trilateration was that it requires:

(1) exactly 3 beacons, no more or less. This causes localisation to fail in the scenario in Figure 3.
(2) their circles (the proximity to IoT device) must have mutual intersecting points. This causes localisation to fail in the scenario in Figure 4.

Table 1: Reference RSSI (in dBm) before and after manual calibration.

Beacon MAC address	RSSI at 1 metre	After manual calibration
ED23C0D875CD	-97.25	-84
E7311A8EB6D7	-95.97	-80
C7BC919B2D17	-66.78	-56.5
EC75A5ED8851	-90.24	-83
FE12DEF2C943	-89.56	-84.5
C03B5CFA00B8	-53.18	-50
E0B83A2F802A	-99.89	-86
F15576CB0CF8	-96.00	-89
F17FB178EA3D	-88.39	-85
FD8185988862	-95.03	-85

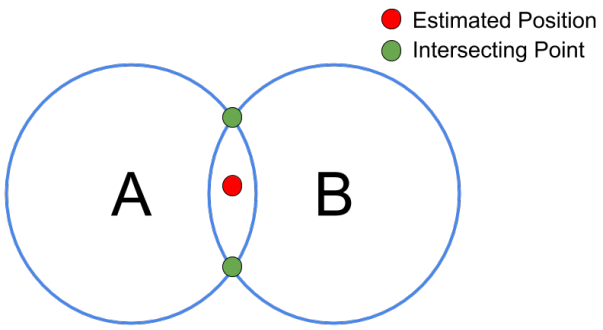

Figure 3: 2 circles with intersecting points.

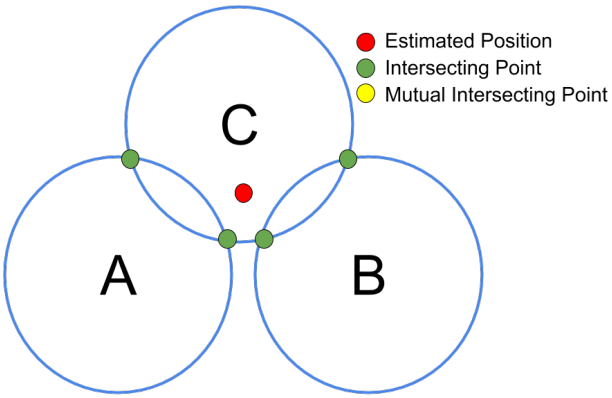

Figure 4: 3 circles with no mutual intersecting point

In the prototype, the number discovered beacons in the data time window vary greatly from 1 to 6 which makes the 3D trilateration impractical in interpolating the IoT device's position.

If there are more than 3 beacons, a solution to requirement 1 would be to apply a filtering criterion to choose exactly 3 beacons out of all the discovered beacons. The experimental criterion was to find 3 beacons with the closest proximity to IoT device, but this was proven impractical in the real-time Android app visualisation. Requirement 1 limits the number of beacons used in localisation which limits accuracy. Requirement 2 causes localisation to fail because the proximity to IoT device is often shorter than the true value, and thus no mutual intersecting points. Therefore, an alternative was pursued.

3.2.2 2-Dimensional Trilateration. The 2-dimensional space trilateration (based on cartesian coordinate) from the same package [16] was experimented. A Cartesian coordinate system is required for this implementation so the beacons' geographic coordinates have been converted into Universal Transverse Mercator (UTM) coordinates whose unit is in metre. This 2D version has a main advantage over the 3D version. The algorithm works when there are 2 or more beacons because it assumes the center of all mutual intersecting points is the estimated location, and therefore it can successfully estimate location in the scenario in Figure 3. The estimated position is converted back into latitude and longitude to

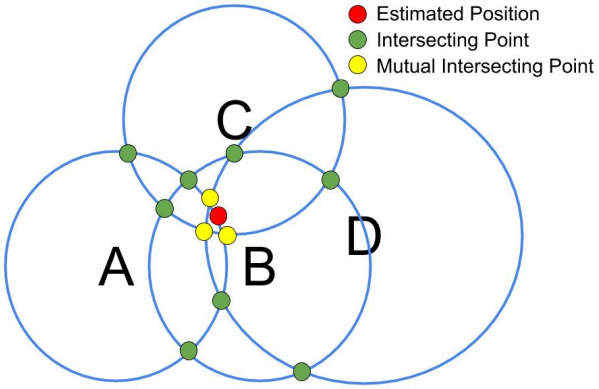

Figure 5: 4 circles with mutual intersecting points and normal intersecting points.

be saved on the Cloud and could be visualised on the Android app and the Cloud dashboard in section 4.

Similar to 3D trilateration, a weakness of this algorithm is that it requires mutual intersection points from all beacons' circle. Therefore, this algorithm fails to estimate location in the scenario in Figure 4

3.2.3 Generalised 2-Dimensional Lateration. A sensible solution would be to find the center of all intersecting points, and not exclusively mutual intersecting points. Therefore, the code in the 2D trilateration has been heavily modified by the author to suit the prototype's needs. As a result, the scenario shown in Figure 3 and Figure 4 the localisation algorithm have successfully estimated the location which is the mean x and y coordinates of all intersecting points.

To further improve the accuracy, the mutual intersecting points of all circles should not be treated the same as normal intersecting points as shown in Figure 5. Therefore, a weight-based approached has been adapted. This means that the mutual intersecting points weigh more than normal intersecting points. The weight of the mutual and normal intersecting point is set to 2 and 1 respectively. Although the weight can be easily adjusted, 2:1 ratio leads to more successful estimation and fewer outlier estimation as experimented in section 5. However, an optimisation algorithm could be developed to find the best weight ratio, but due to time constraint and working alone, this was not feasible.

3.2.4 Filtering discovered beacons. All discovered beacons with proximity to IoT device bigger than 10 metres were not taken into account in the lateration algorithm because of the further the distance the more unreliable the RSSI signal. This reduces noises and improve accuracy.

3.2.5 Last Estimated location. Last estimated location is used in the localisation algorithm in interpolating the IoT device location regardless of how many beacons are discovered in the data time window. The algorithm retrieves the last estimated location and if it is within the past 5 seconds at the time of localisation then, it is treated as an individual beacon with radius equals to the time elapsed in seconds multiply by 0.5. This assumes that the IoT device

moves at 0.5 metres per second. A more realistic speed could be estimated using additional sensors such as an accelerometer to measure speed.

3.2.6 Result Optimisation. The estimated location, P, is input to an optimisation algorithm [17] that finds point X which reduces the mean squared errors of the difference between each beacon's proximity to point P and their true distance to P (found using Haversine method[12]). Given a point X, if X perfectly matches with P, then X is P. If not, then X that minimises the error function is the more accurate estimation than P. X is returned and is converted back into geographic coordinates and saved on a Cloud with the timestamp when the algorithm is run.

Note that in the code submitted, only the "*_experiment.py" script contains this optimisation implemented because the Cloud did not have the necessary packages installed.

4 VISUALISATION

2 visualisation tools have been developed to visualise the results of the localisation algorithm.

4.1 An Android App for real-time visualisation

An Android app has been developed to visualise the results of the localisation algorithm in real time. The app calls an API endpoint in the Cloud which returns the geographic coordinates of the latest location alongside the MAC addresses and proximity of discovered beacons. As shown in Figure 6, the app uses Google Maps API to visualise:

(1) the positions of beacons (Bluetooth icons).
(2) beacon's proximity to IoT device (blue circles).
(3) the estimated location (red dot).

This app has mainly been used to manually calibrate the reference RSSI in section 3.1.2 and visualising the result from localisation in almost real time (API calls have delays).

4.2 Cloud Visualisation Dashboard

A Cloud visualisation dashboard was developed using HTML, JavaScript, Google Maps API. When the index homepage is accessed, an API endpoint which points to the batch processing script is called. The results from the script are uploaded to the Cloud. The dashboard retrieves the location estimations and plots them on Google Maps. There are 4 visualisations:

4.2.1 The Home Page. The homepage shows all estimations computed from the batch processing script on the Cloud as markers on the Google Map. Each marker has their title set to their timestamp. This page also has 3 buttons to navigate to the other 3 visualisations.

4.2.2 Path Trace. Path trace plots the estimated locations sequentially one by one. When the next one is plotted, the previous one is removed. This repeats until no more estimations are left. The timestamp of the estimation is shown on the menu and speed can be adjusted from the menu. This helps the author to visualise a live journey of the estimated locations.

4.2.3 Path Reconstruction. Like path trace, path reconstruction plots computed estimations sequentially for an adjustable time interval. Unlike path trace, the plotted markers will remain. This visualisation assists in evaluating the accuracy of the path of the moving IoT device.

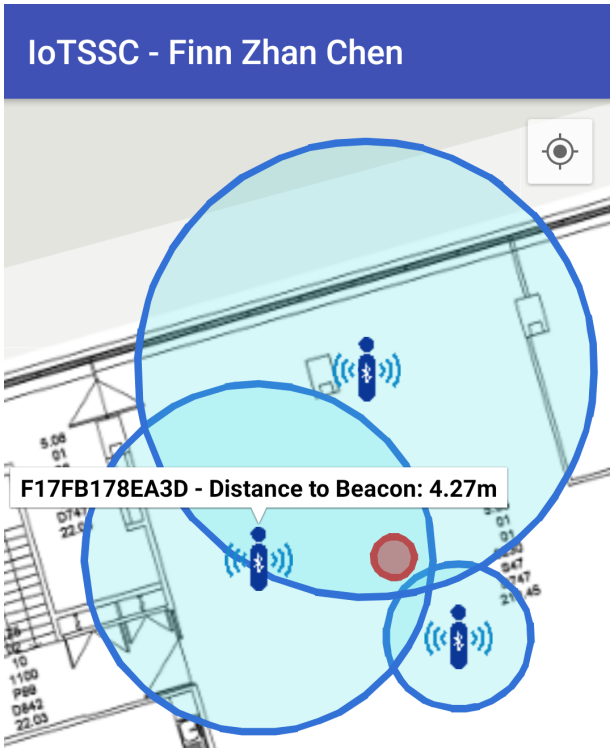

Figure 6: Android App for real-time visualisation.

4.2.4 Heatmap. This plots a heatmap of the estimated locations. The denser an area is, the stronger the colour is. This visualisation helps the author in evaluating areas on the map that are rarely the result of the localisation algorithm.

5 PERFORMANCE EVALUATION

Location accuracy is the key performance indicator. The evaluation is heavily focused on stationary points tests rather than continuous moving path tests due to easier quantitative evaluation. During the collection of the data in the path, the IoT device is facing up and is around 1 metre above ground.

5.1 Stationary Test Set

In this test, shown in Figure 7, 15 evenly distributed points have been selected on the floor map whose ground truth geographic coordinate have been collected using [11]. The path started on point 0 and finished on point 14. Data is collected for 30 seconds at each point and their starting and finishing timestamps are also recorded as a reference for the evaluation scripts ("*_experiment.py").

For each test point, the number of successful location estimation within the 30 seconds period is recorded. The mean distance error of the estimated location to the true location has been calculated using the Haversine method [12]. Then, the overall accuracy is calculated.

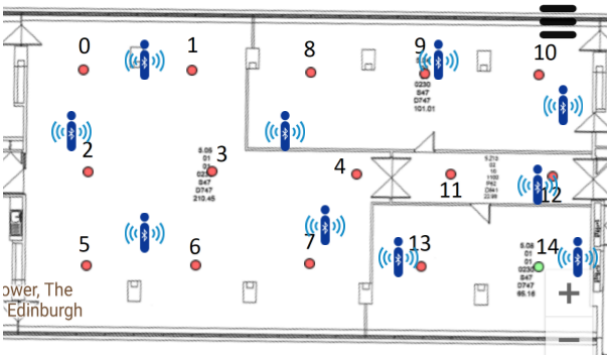

Figure 7: Stationary points test set.

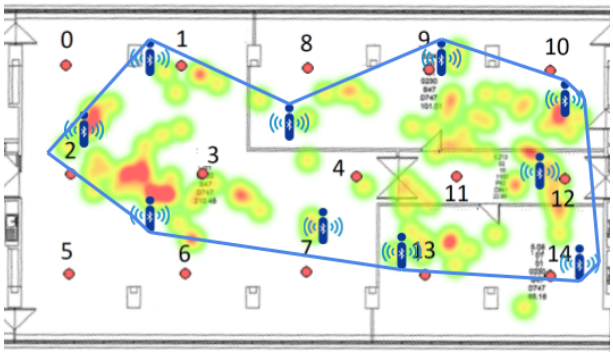

Figure 8: Visualisation of the test result.

The parameters of the localisation algorithm have been experimented to maximise the following in order of importance:

(1) overall accuracy.
(2) number of successful estimation.
(3) individual test point accuracy.

5.2 Test Results

After some experimentation of the adjustable parameters, the data time window is set to 4.5s, the algorithm time interval is set to 3s (there should be a maximum of around 10 estimations per test points). The results are displayed in Table 2.

As clearly shown in Figure 8, the localisation algorithm is heavily dependent on the Bluetooth signals from the beacons and the estimated location from the algorithm is limited to the blue lines. This could be solved by adding additional Bluetooth beacons. Inputs from additional sensors would also improve accuracy at places where Bluetooth signals are not available. However, due to time constraint and working alone, this was not feasible.

Table 2: Test result for each test point.

Test Point	Number of Estimations	Mean Distance Error (metres)
0	8	3.1570353838851464
1	5	1.0454290095323688
2	6	3.867109226134572
3	3	4.209911370221693
4	6	2.550590978083412
5	6	4.659396844781046
6	3	2.641241776631099
7	10	5.735763376893904
8	6	3.553800402363198
9	7	5.096912579179528
10	11	3.244255123105696
11	9	2.6440261327014656
12	8	1.3556790824841536
13	4	2.233608023472335
14	6	4.753636728148626

The localisation algorithm on average provides an accuracy of 3.38 metres. The results of the test points provide 1 to 6 metres accuracy depending on environmental factors such as walls, furnitures, and distribution of the Bluetooth beacons.

6 CONCLUSIONS

The feasibility of the indoor localisation prototype in real world applications depends on its localisation accuracy. Although Bluetooth signals were attenuated due to environmental factors, a localisation accuracy of 3.38 metres proves to be useful in some business scenarios. For example, proximity-based advertising in shopping malls where stores are far apart (around 5 to 20 metres) or category advertising (for example, meat and beverage section) in supermarket are feasible.

A weakness of an indoor localisation system that solely relies on Bluetooth signals is that the estimated localisations are limited to areas covered by Bluetooth beacons. This problem could be solved by adding more equally distributed beacons which are cost-effective and highly scalable.

Parameters for the algorithm such as data time window and algorithm time interval have been optimised for stationary points, so these parameters should also be experimented to find the optimal parameters for continuous moving paths.

7 FUTURE SCOPE

Additional sensors could be added to improve accuracy and mitigate the localisation algorithm's dependency on Bluetooth signals. A better filter to remove noises from RSSI signals than outlier removal would be Kalman Filter.

Optimisation algorithms could be developed to find the optimal reference RSSI and weights for the mutual and normal intersecting points to maximise accuracy. Encryption algorithm could also be used to make data more secure and avoid data breach.

Combining location data with users' shopping profiles will enable more accurate and timely advertisements for items nearby, as well as informing users of any coupons they may have to use. Further, gathering the individual paths of the customers will help in designing mall and store layout and future campaigns.

REFERENCES

[1] Wouter Bulten. 2015. Kalman filters explained: Removing noise from RSSI signals. https://www.wouterbulten.nl/blog/tech/kalman-filters-explained-removing-noise-from-rssi-signals/

[2] Android Developer. 2018. Volley Overview. https://developer.android.com/training/volley/

[3] Google Developer. 2017. Android BluetoothLeGatt Sample. https://github.com/googlesamples/android-BluetoothLeGatt

[4] Qian Dong and Waltenegus Dargie. 2012. Evaluation of the reliability of RSSI for indoor localization. In *Wireless Communications in Unusual and Confined Areas (ICWCUCA), 2012 International Conference on*. IEEE, 1–6.

[5] Punit Jajodia. 2018. Removing Outliers Using Standard Deviation in Python. https://www.kdnuggets.com/2017/02/removing-outliers-standard-deviation-python.html

[6] MartinBL. 2016. Bluetooth low energy Characteristics, a beginner's tutorial. https://devzone.nordicsemi.com/tutorials/b/bluetooth-low-energy/posts/ble-characteristics-a-beginners-tutorial

[7] Frederic P Miller, Agnes F Vandome, and John McBrewster. 2009. Advanced encryption standard. (2009).

[8] Arm Mbed OS. 2015. BLE GATT Example. https://os.mbed.com/teams/Bluetooth-Low-Energy/code/BLE_GATT_Example/

[9] Arm Mbed OS. 2016. BLE LED Blinker. https://os.mbed.com/teams/mbed-os-examples/code/mbed-os-example-ble-LEDBlinker/

[10] Onkar Pathak, Pratik Palaskar, Rajesh Palkar, and Mayur Tawari. 2014. Wi-Fi Indoor Positioning System Based on RSSI Measurements from Wi-Fi Access PointsâĂŤA Trilateration Approach. *International Journal of Scientific & Engineering Research* 5, 4 (2014), 1234.

[11] Valentin Radu. 2018. LSR DataCollection. https://github.com/vradu10/LSR_DataCollection

[12] C Carl Robusto. 1957. The cosine-haversine formula. *The American Mathematical Monthly* 64, 1 (1957), 38–40.

[13] Bluetooth SIG. 2018. Generic Attributes (GATT) Specifications. https://www.bluetooth.com/specifications/gatt

[14] Tom Spink. 2017. InfCloud. http://glenlivet.inf.ed.ac.uk:8080

[15] Jie Yang and Yingying Chen. 2009. Indoor localization using improved rss-based lateration methods. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE, 1–6.

[16] Jay Ying. 2015. The python version implementation for trilateration algorithm. https://github.com/noomrevlis/trilateration

[17] Alan Zucconi. 2017. Positioning and Trilateration - Optimisation Algorithm. https://www.alanzucconi.com/2017/03/13/positioning-and-trilateration/#part3