Indoor Positioning System

Authors: **Li Puma, Juan Francisco (55824)**

Rodriguez Nicastro, Julian (55291)

de Rocheboüet, Diego (55821)

Supervisor: **Pablo Ignacio Fierens**

Final Project
Software Engineering

Instituto Tecnológico de Buenos Aires

This project was carried out as a part of the Software Engineering degree at ITBA.

Buenos Aires, Argentina
29 November 2019

Abstract

GPS and location services have been a breakthrough in recent history and have become a necessary part of our daily lives, but they haven't yet reached the point where we can be located inside buildings. This would be really useful, with multiple applications such as localization in big buildings, location-based advertising, location-assisted games, location-aware social networks, etc. For this reason, there has been research on the field to solve this problem and many possible approaches have been found. This project presents a starting point for one of these solutions using WiFi fingerprinting. The project consists of an API and two mobile applications that consume it, one for admins and another for users. The admin app is used to gather samples needed for WiFi fingerprinting in an offline stage and the user app is used to locate users based on WiFi scans on an online stage. This solution is then tested and the results are presented.

Keywords: Indoor Positioning System, WiFi positioning system, Fingerprinting

1. Introduction

In today's world, the service provided by the Global Positioning System (GPS) is ubiquitous. The system works especially well outdoors and enables accurate localization, speed measuring and timing, among other things. Its most common use case is street directions, which is possible thanks to massive amounts of street data as well as GPS' accuracy: numerous studies have found that GPS-enabled smartphones are typically accurate to within a 10-meter radius approximately (Van Diggelen and Enge, 2015; Zandbergen, 2009; Hess et al., 2012; Merry and Bettinger, 2019). However, the system is not infallible and GPS signals can be blocked by most objects, deteriorating location accuracy. This makes GPS unreliable for indoor positioning as roofs, walls and other objects obstruct signals. There are several commercial indoor positioning systems (IPS) available that overcome these shortcomings, such as *insoft GmbH*, *HERE* and *Proximi.io*. Many of these require the purchase of specialized equipment for localization to be sufficiently accurate, and this investment makes commercial IPS infeasible for individuals.

Another approach, which is the focus of this project, leverages another ubiquitous system: WiFi. By measuring the strength of received WiFi signals, a *radio map* can be constructed and later used to locate a user given the wireless Access Points (APs) around them. This technique is called *fingerprinting* and, thanks to the prevalence of access points, becomes a "free" solution in that it requires no extraordinary investment. Thanks to this, indoor location services could be provided both by businesses large and small as well as by individuals, and as such the global positioning system could further extend inside buildings.

The goal of this project is to develop an extensible and adaptable solution that implements this technique. With the purpose of reducing the access barriers as much as possible, it was decided to use smartphones as the localization agents. This makes sense because they are almost always carried on-person, they are even more common than WiFi access points and they are equipped with vast and sophisticated sensor suites, including GPS, WiFi and Bluetooth antennas among many other sensors. The project is composed of various applications which aid in different stages of the process. Initially, a building administrator registers the building on the system with data including coordinates, number of floors, and floor plans for every floor. Then, with the use of a mobile app, authorized users collect samples, which contain both a recorded fingerprint and an entered location (coordinates and floor), thus charting out the building's radio map. Once enough data has been collected, regular users, with a different mobile app, perform automated background WiFi scans which are periodically sent to a server. The server compares the fingerprint with those of recorded samples and returns a location computed from the fingerprints of the most similar samples, if any. The estimated location, including floor, is subsequently updated on the user's app.

It is not the intention of this project to introduce a novel algorithm with which to estimate location by comparing fingerprints, but rather to implement a known, well-performing one in the context of the applications discussed previously and compare the resulting accuracy against that reported by the authors of the algorithm. It is the intention for the implementation to be such that the algorithm's parameters, or the algorithm itself, can be easily changed and tested. With this, accurate and relevant location services could be provided to users walking up, down and throughout any building.

2. Problem Setting

2.1 WiFi

The ubiquity of WiFi (IEEE, 2016; Wi-Fi Alliance, 2019), as previously discussed, makes it a great tool to leverage in predicting the position of users. Each WiFi network is composed of one or more wireless access points that connect wireless devices within a local area network (LAN). These APs are usually routers, placed by homeowners or building administrators, and they provide the connection to the World Wide Web. Each of these devices can be uniquely identified by their hardware (MAC) address or by their Basic Service Set Identifier (BSSID), both which are 48-bit labels. It is also possible for a single AP to broadcast multiple networks, as is usually the case for home routers which have both 2.4GHz and 5GHz networks. In this case, even if both networks share the same name, they have different BSSIDs.

Devices are able to perform a wireless scan to discover networks they can connect to. Based on the distance and obstacles between the device and the networks' APs, as well as the capabilities of the device itself, different signal strengths will be detected for each of these networks. This measure, called the Received Signal Strength Indicator (RSSI), has no defined units or range (see IEEE, 2016, Table 20-1) so these are manufacturer-dependent (Lui et al., 2011), but generally speaking a higher number indicates better signal strength. In the case of the Android operating system, it is reported in dBm (Android Developers, 2019), and data collected in the project shows that its values range from -100 (no signal) to 0. Therefore, when a scan is performed the result is a list of APs with their corresponding RSSIs, called a *fingerprint*. It should be noted that when a scan is performed not only will APs be detected, but also any device that broadcasts a network such as printers, cellphones or personal computers. Because not all of these are fixed to a specific location, not every network can be used as reference for localization.

2.2 Previous Research

There is an abundance of indoor localization methods that have been proposed in the past. In this project, the methods of particular interest are those which can be performed entirely, or at least mostly, with the use of the sensor suites present in most smartphones today. Thanks to the complexity of the sensor suites found on smartphones, many methods can be implemented. Figure 1 displays a taxonomy of indoor localization methods. Dead reckoning works with a known initial position along with sensor data in order to calculate changes to that position. Sensors commonly used in dead reckoning are the accelerometer and gyroscope. Fingerprinting, as previously discussed, consists of building a database of sensor data with known locations, and then comparing new fingerprints with known ones to locate a device by fingerprint similarity. WiFi RSSI fingerprinting, a specific type of fingerprinting, is the technique chosen in this project.

Trilateration works by estimating distance to various wireless anchors (not necessarily WiFi APs, distance to wireless devices based on Bluetooth beacons could also be used for example). With distance measurements at a minimum of three separate known locations, a three-dimensional position relative to the anchors can be established. A popular way of measuring distance is through time of flight (ToF): by measuring the time it takes a signal to travel from source to receiver, and multiplying by light speed, distance can be obtained. The WiFi standard provides methods to estimate distance using ToF, and the latest versions of Android already make use of it. The estimated precision is in the order of 1 ns (30 cm), although the error can be greater for localization (Banin et al., 2016). Triangulation is similar, but works with angles instead of distances. If the angles at a minimum of two known locations for a smartphone are known, their location can be estimated.

Proximity estimation works on the assumption that the device's position is that of the beacon with highest signal strength. This provides only general proximity, but it is enough for several use cases such as advertising in indoor shopping malls. Last but not least, visual localization uses a phone's camera to recognize the environment or analyze light signals encoded within, such as LED pulses in ambient lighting, to estimate a location. All of these and other methods are discussed in more detail in Langlois et al. (2017).

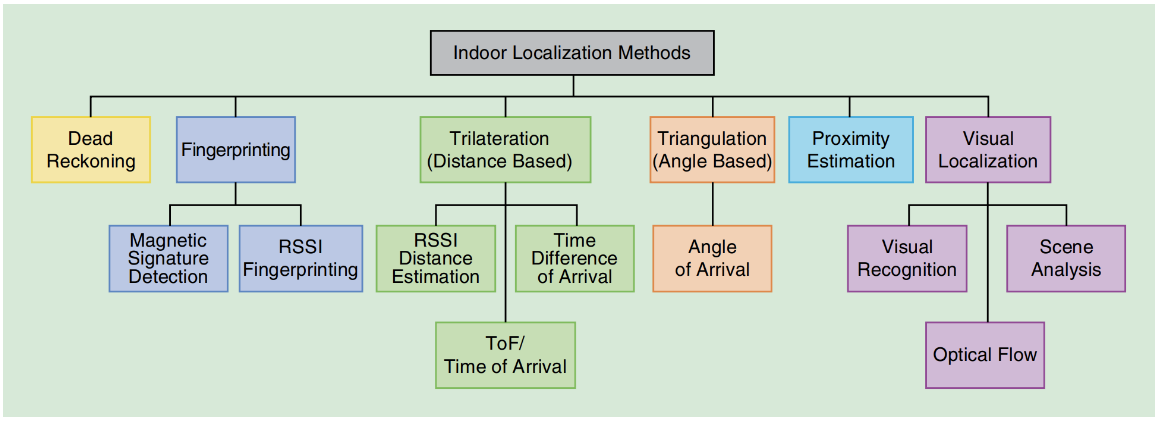


Figure 1: Taxonomy of indoor localization methods. Langlois et al. (2017)

The accuracy of localization depends on the sensors and the technique used. In the case of WiFi RSSI fingerprinting, Campeón et al. (2018a) have obtained a maximum accuracy in the order of 1 - 2 m. There are also studies that show that the accuracy of WiFi RSSI fingerprinting can be improved with the use of other sensors, such as the magnetometer

(Campeón et al., 2018b). The accuracies of various other methods are displayed in Langlois et al., Table 1.A.

Out of all the discussed localization methods, WiFi RSSI fingerprinting is the most popular. In order to compare performance of existing solutions, international competitions are held yearly where teams subject their algorithms to a benchmarking test are and evaluated on mean error. In the EvAAL competition at the 2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN) held in Canada, the RTLS@UM team obtained first place with their algorithm with a mean error of 6.20 m, median error 4.6 m, 100% building hit rate and 93.74% floor hit rate (see Moreira et al., 2015, Table XI). The algorithm presented by the team is the one that this project implements.

2.3 Positioning Algorithm

RTLS@UM presents multiple variants of their algorithm but for the purposes of this project the base variant, labeled A_1V_1 in (Moreira et al., 2015), was used. WiFi fingerprinting is based on recording a fingerprint and comparing it with a previously built radio map, which is essentially a list of fingerprints with their corresponding locations. By comparing, the algorithm can find the most similar fingerprints and compute the average of their associated locations. The algorithm thus receives as input both a fingerprint and the radio map and outputs the estimated location from where said fingerprint was measured. This is done in four main steps which can be seen in Figure 2 and which are now explained in detail.

In the first step, some preliminary calculations need to be done. The algorithm starts by taking the input fingerprint and sorting its APs by descending RSSI. Let fp_0 be this sorted input, which can be defined as a list of tuples $[(AP_0, RSSI_0), (AP_1, RSSI_1), ..., (AP_Q, RSSI_Q)]$ with Q being the number of APs in the fingerprint. This is also done for each of the fingerprints in the radio map R, with the goal of obtaining the best AP for each fingerprint (the one with the highest RSSI). As a last preliminary calculation, the similarity between fp_0 and all of the fingerprints in R is calculated. The similarity function is a variant of Manhattan distance (with shorter distance meaning greater similarity), defined in the paper as:

$$S(fp_1, fp_2) = \frac{1}{S} \times \sum_{i=1}^{S} |rssi_i^1 - rssi_i^2| - 2 \times C \tag{1}$$

where S is the total number of APs observed in the union of fp_1 and fp_2, and C is the number of APs that were observed in both fp_1 and fp_2. For missing APs, in fp_1 or fp_2, a default RSSI value is used.

As a second step, the algorithm decides the building where the fingerprint was taken. For this, a subset of R with all the fingerprints with AP_0 as their main AP is created. If this subset is not empty, the building is selected from this group by simple majority. If it is empty, the process is repeated for the next strongest AP in fp_0 (AP_1, then AP_2, etc.) until a building is chosen. Let the selected building be B. If no building is found, the algorithm ends and returns an empty location.

Once the building is selected, the algorithm determines the floor within B. This starts by creating R_1, a subset of R with all the samples that have B as their building. Following this, R_2 is created, a subset of R_1 with all the fingerprints where AP_0, AP_1, ..., or AP_M is the strongest AP, M being a parameter of the algorithm. If R_2's cardinality is less than N,

N being another parameter of the algorithm, then R_2 is overwritten to be R_1. The next step is to take the K_1 most similar samples from R_2 and, like the building, the floor F is selected by simple majority.

Finally, R_3, a subset of R_2 where the floor is F, is built. The algorithm takes the K_2 most similar samples from R_3 and calculates the centroid to obtain the result. This is done by averaging the x and y coordinates of the resulting set, each axis averaged separately. The returned value is the coordinates of the centroid, along with F and B.

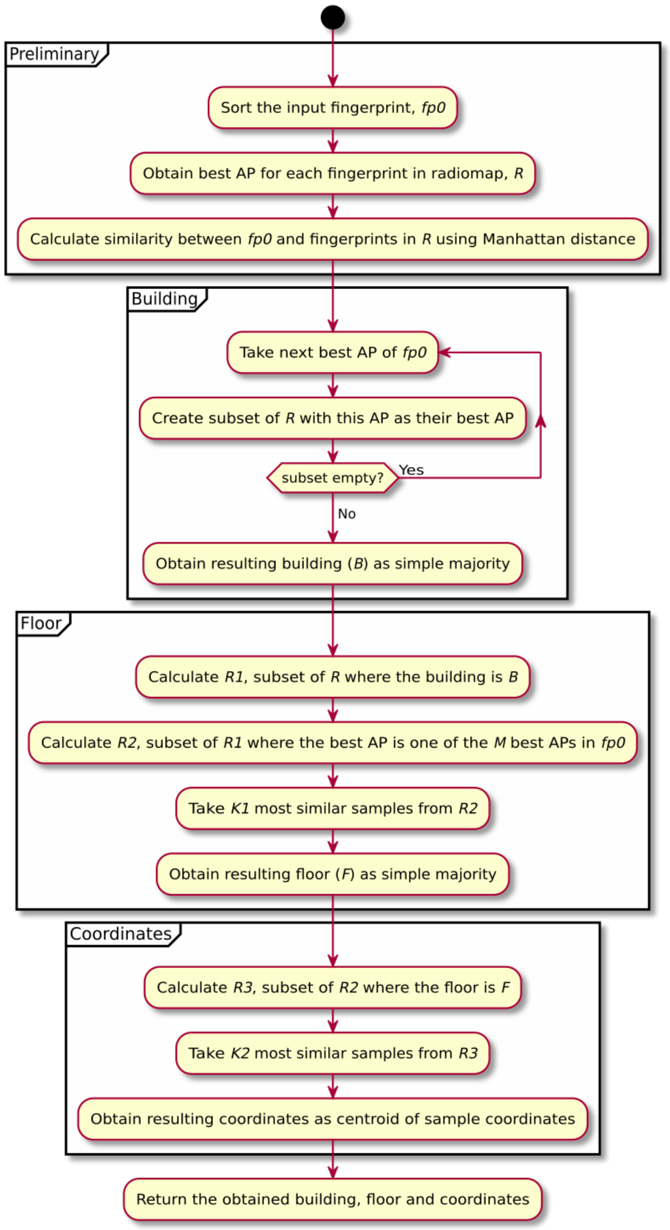

Preliminary
Sort the input fingerprint, *fp0*
Obtain best AP for each fingerprint in radiomap, *R*
Calculate similarity between *fp0* and fingerprints in *R* using Manhattan distance
Building
Take next best AP of *fp0*
Create subset of *R* with this AP as their best AP
subset empty?
Yes
No
Obtain resulting building (*B*) as simple majority
Floor
Calculate *R1*, subset of *R* where the building is *B*
Calculate *R2*, subset of *R1* where the best AP is one of the *M* best APs in *fp0*
Take *K1* most similar samples from *R2*
Obtain resulting floor (*F*) as simple majority
Coordinates
Calculate *R3*, subset of *R2* where the floor is *F*
Take *K2* most similar samples from *R3*
Obtain resulting coordinates as centroid of sample coordinates
Return the obtained building, floor and coordinates

Figure 2: Positioning algorithm diagram.

3. Implementation

In order to achieve our goal of developing an extensible and adaptable solution that implements the algorithm introduced by RTLS@UM, the proposed design had to cover two main flows: the administration flow, where building plans could be uploaded and fingerprints collected; and the user flow, where real-time positions would be shown on a cellphone using the algorithm. A solution with two mobile applications and a web server was proposed, the idea being to have one mobile application for each of the flows. On addition to this, a web app was also developed with the purpose of making algorithm testing easier. The high-level flow can be seen in the Figure 3, where the three main components appear. To the left is the admin app with the administration flow: for each sample, the administrators have to locate themselves, perform a scan and send the information to the server. In the middle, the web server that stores the data and performs the calculations. And to the right, the user app with the user flow: the recorded fingerprint is sent to the server, which replies with a location that is shown in the app. In the following sections each of the components and flows are discussed in further detail.

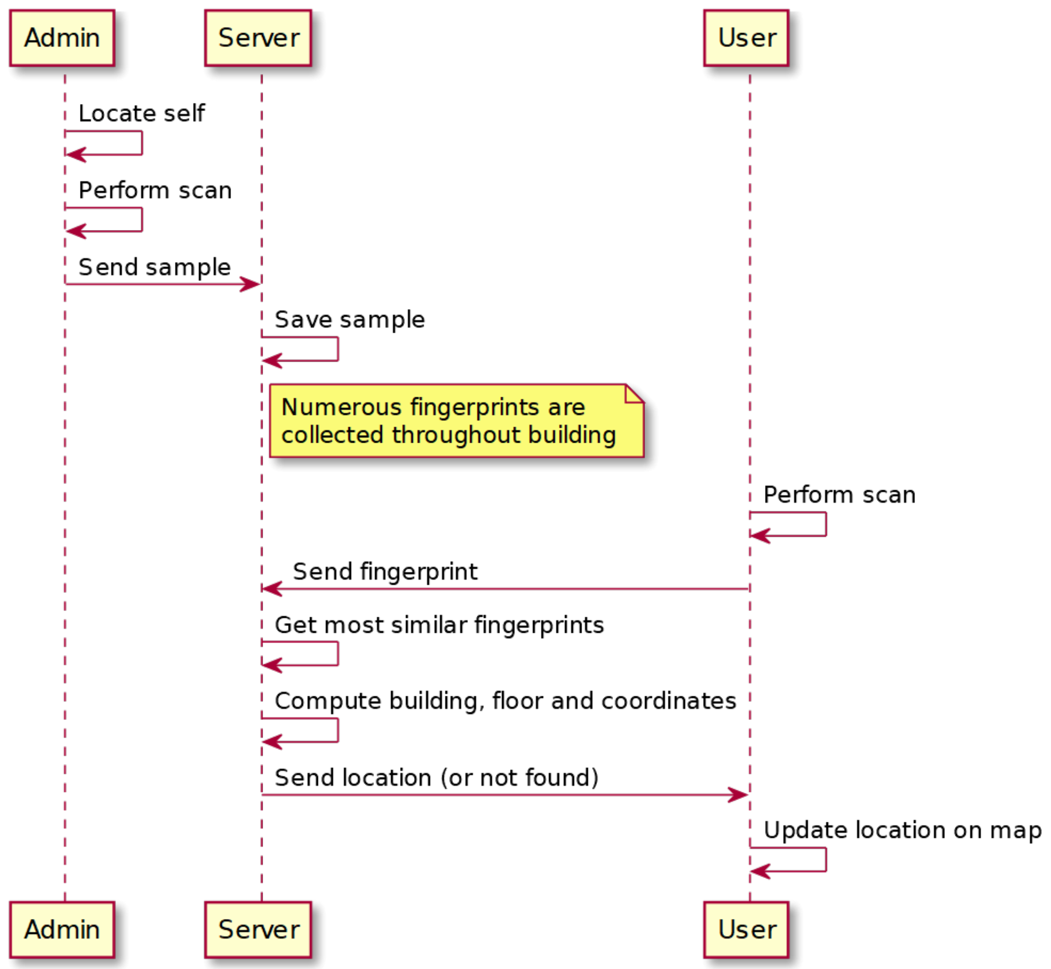

Admin
Server
User
Locate self
Perform scan
Send sample
Save sample
Numerous fingerprints are collected throughout building
Perform scan
Send fingerprint
Get most similar fingerprints
Compute building, floor and coordinates
Send location (or not found)
Update location on map
Admin
Server
User

Figure 3: High-level flow.

3.1 API + Database

In order to calculate the position, two options arose. The first one was to have the mobile phone perform all the calculations, by running the algorithm in the user app. This would mean that the app would need to store all the samples and query for updates at regular intervals to maintain a synchronized database. This first approach has the benefit that the location would not depend on neither the transport time to nor the calculation time within the server, but the trade-off is increased CPU and storage usage. It also means that if the administrator wants to update the algorithm or its parameters, this wouldn't propagate to users unless an update is downloaded. The decision was made to not over-encumber the user phones and instead take a different approach, with the server performing all the calculations and being in charge of storing all samples in a database.

The API is implemented in NodeJS using the Express web framework to serve requests and authenticate, using Mongoose to perform CRUD operations against a MongoDB instance. The database maintains a record of all registered buildings, their data (location, floors, floor plans, etc.) as well as fingerprints captured within them. Production instances of both the API and the database it manages are deployed in Heroku and are accessible under `https://pf-itba-spi.herokuapp.com/api-docs`. Most technologies were chosen because they are the easiest to get up and running with for web projects. MongoDB in particular was chosen over other relational database engines because its lack of schema is suitable for this use case, where arbitrary sensor data is collected by admins and used by users. Relational databases would require either frequent schema updates for new data types, or a "catch-all" column for arbitrary extra data which would bloat over time without taking advantage of the benefits provided by relational databases. Heroku, the deployment platform was selected due to its ease of use and familiarity among the project members. Changing the deployment platform would not require any more effort than setting it up and deploying the API there, since nothing is particularly tied to Heroku. Finally, NodeJS, the JavaScript runtime under which the API itself runs, was also used along with the Express framework due to its particular specialization for APIs, as well as its ease of access to external code. Many packages available from the Node Package Manager (NPM) registry provide out-of-the-box solutions for many issues such as database connection (*Mongoose*), authentication (*jsonwebtoken*), configuration (*dotenv-safe*), API documentation (*swagger*) and more. Furthermore, JavaScript's weak and dynamic typing makes it a perfect fit for working with arbitrary data such as samples.

To serve data and provide functionality multiple endpoints were created. First, there are CRUD endpoints (Create, Read, Update, and Delete) for our two REST resources: buildings and samples. Secondly, there is the location endpoint which receives a fingerprint and runs the positioning algorithm with it against all stored samples. Thirdly, there is the location filtering algorithm, which receives a sample ID and runs the positioning algorithm but excludes the specified sample to use it as input. In other words, it is as if the sample had never been recorded, and is instead a user-recorded fingerprint. This is done so the location output can be compared with the real location, also stored in the database, and is used to measure the error distance for said sample.

3.2 Admin App

The admin app is used to record and manage samples in registered buildings. It authenticates against the API with a cryptographic keypair (i.e. public/private key authentication) that gives it permission to get building information and also insert new samples in existing buildings. The admin flow can be seen in Figure 4, which is now explained. When the admin app starts, it requests the buildings with their floors to be able to show the corresponding floor plans. Once this is done, the administrator can start to upload samples, which is achieved in three steps. First, they go to the location where they want to add a new sample. Secondly, they select the corresponding building and floor. Then they use the floor plans to locate themselves, by moving the map using the touchscreen. Finally, once they have located themselves on the map, they initiate a scan and wait for it to complete and upload successfully. The sample is saved in the server's database and a new scan can be started.

The admin app also has access to existing samples, so as to visualize the existing radio map. Samples are all displayed as markers on the map of the currently selected floor, as can be seen in as seen in Figure 5a and Figure 5b. The administrator can tap on each of these markers to see more details about their corresponding fingerprints, or delete them if needed.

With this the admin app has the liberty to fully manage samples in buildings. It is worth noting that existing samples cannot be edited other than by directly accessing the database and modifying values. They can only be deleted, as previously discussed.

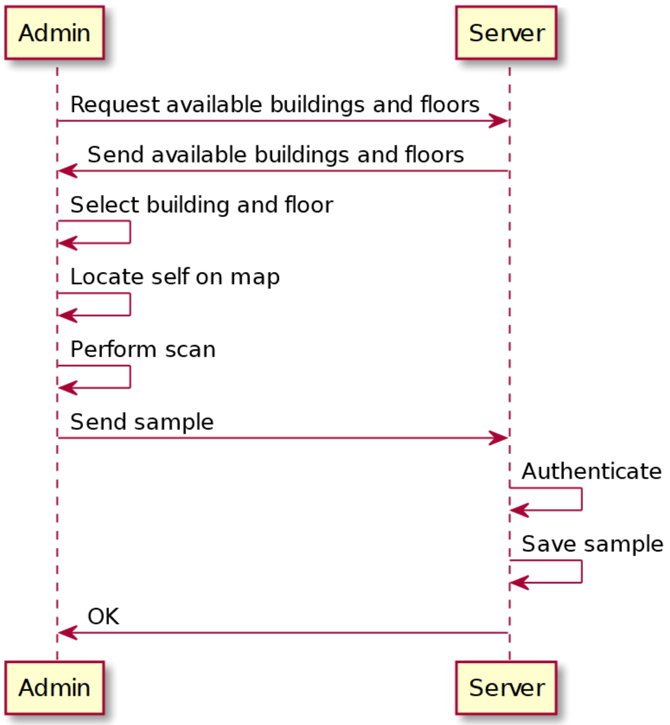
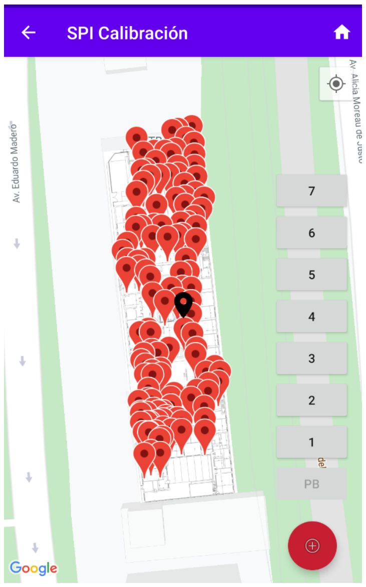
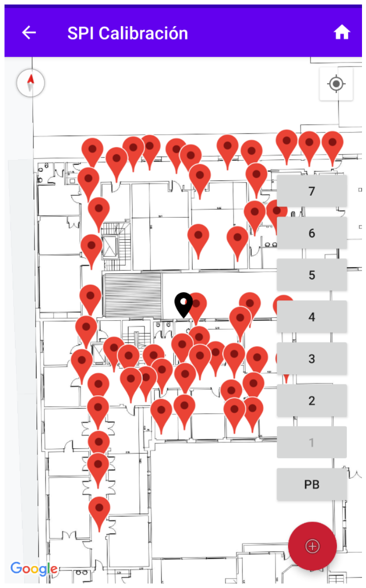

Admin
Server
Request available buildings and floors
Send available buildings and floors
Select building and floor
Locate self on map
Perform scan
Send sample
Authenticate
Save sample
OK
Admin
Server

Figure 4: Admin app flow.

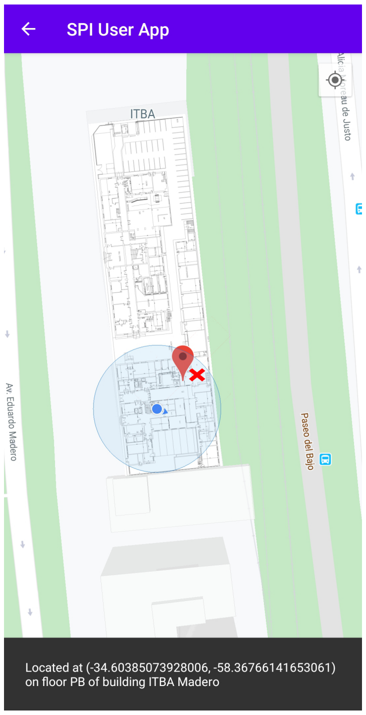


(a) Admin app screen with all the samples for the ground floor.



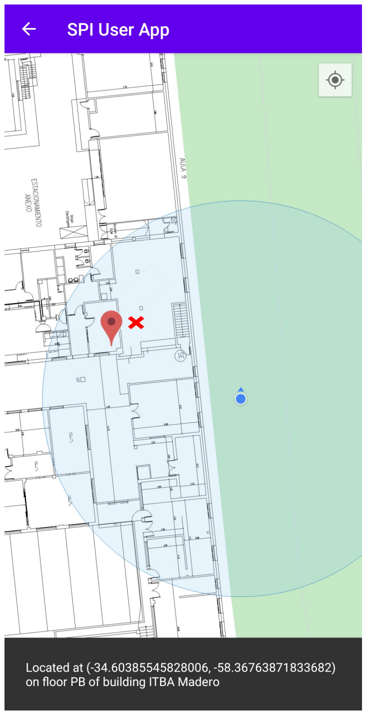
(b) Admin app screen with samples for the first floor, zoomed in.

Figure 5: Admin app screenshots.



Located at (-34.60385073928006, -58.36766141653061)
on floor PB of building ITBA Madero

(a) User app screenshot in ground floor.



Located at (-34.60385545828006, -58.36763871833682)
on floor PB of building ITBA Madero

(b) Zoomed-in user app screenshot in ground floor.

Figure 6: User app with system (blue dot), algorithm (marker) and real (cross) locations.

3.3 User App

The user app has the same base code as the admin app but instead of calibration, its main purpose is locating the user. Because the app would be for public use, it authenticates against the API with a different keypair that only gives it read access to buildings in order to download building info, including indoor maps and floor distribution. To maintain the user location updated, the location flow seen in Figure 7 is performed periodically. Due to limitations in Android, which will be explained in next section, this is done every 10 seconds for Android versions below 8 (Oreo) and every 30 seconds otherwise. First, the user's phone launches a WiFi scan request and hits a location request endpoint in the API with the scan results. The server compares the sample to the ones collected in the calibration process, locating the building, the floor and the coordinates (in that order). After finishing with the algorithm, the server replies with the calculated location (or lack thereof) back to the app. If the API replies with a valid location, the app updates the current location accordingly. This is done by showing the user location with a marker, updating the floor map and sending a toast with this information. This can be seen in Figure 6a and Figure 6b, where a red cross was added to show the real user's location. If not located, the app maintains its current location (if any) and shows an icon that indicates that the location is unknown. If the user device loses Internet connectivity, another icon is shown until connectivity is restored. The user app will continue to attempt location when it has a connection to the API.

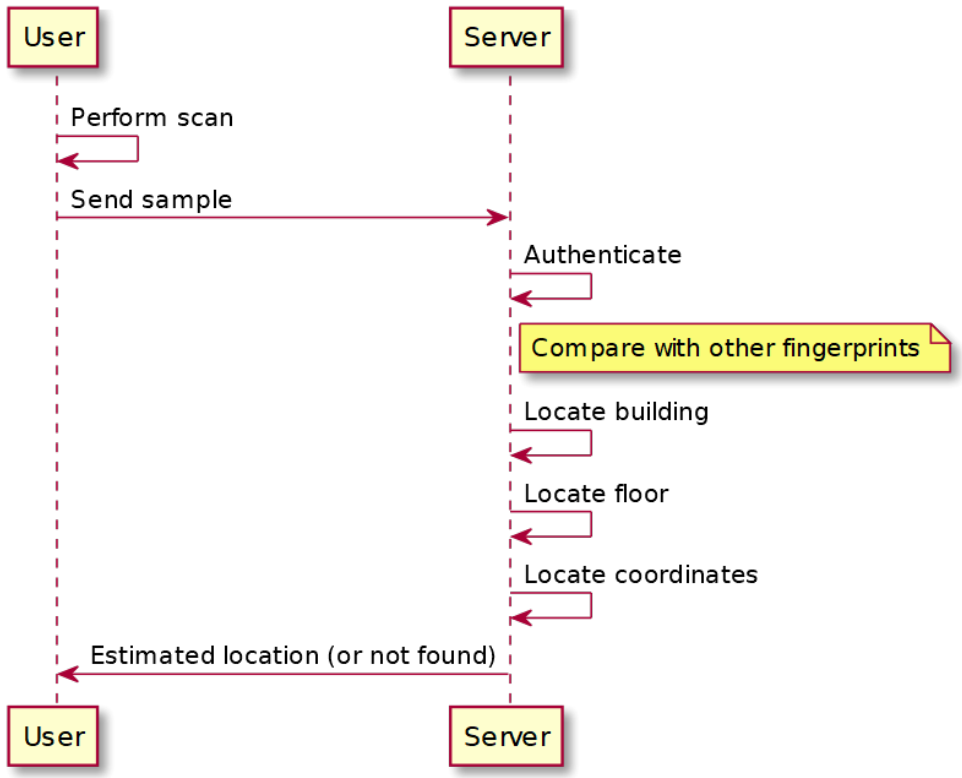


Figure 7: User app flow.

3.4 Web Admin/Tester

In addition to the components described above, a simple web app was developed as a quick non-mobile solution that allows visualizing the radio map, much like in the admin app, as well as testing the accuracy of the location algorithm, without having to be physically present at any particular building to send real fingerprints. The web app displays available samples just like in the admin mobile app, and when clicking on a marker the app hits the filtering endpoint in the API with the clicked sample. The API in turn runs the location filtering algorithm, which was previously discussed. The result (if any) is displayed in the web app as a marker with a different color to distinguish it from existing samples and compare. If location fails, no marker is shown.

3.5 OS choice

For the mobile applications, Android was chosen as the target platform. The main reason behind this decision was the high availability of hardware-handling libraries the SDK provides. Conversely, iOS was discarded because Apple's developer kit does not allow querying WiFi state, therefore making WiFi scanning apps infeasible (Eskimo, 2019, 2017). This left Android as the only viable option to implement the IPS in.

Flutter and React Native hybrid frameworks were considered but discarded since their only advantage over Android is the option to develop a single application to build and release for both Android and iOS simultaneously. But because in this case the iOS build would not be released for reasons explained above, neither alternative added any value. It is also worth noting that hybrid frameworks usually lack hardware-handling libraries, so a third-party one would have been needed. An initial research showed that such libraries are scarce and not always reliable. For these reasons, native development was ultimately chosen over hybrid development.

It should be noted that even though the Android SDK allows making network scans, as of Android API level 26 (Android 8) background WiFi scans have been severely restricted. Unless it is a system app, an app in foreground is capped at four scans every 2 minutes while an app in background is only allowed a single scan every half hour (Wang, 2019). This change in Android policy is recent and greatly limits the functionality of WiFi scan based applications. As of today there is no solution to the problem and there is no expectation for this decision to be reverted.

4. Results

All tests and samples were taken at the Madero building of the Instituto Tecnológico de Buenos Aires (ITBA) using a OnePlus 6T, a Samsung Galaxy S10 and an LG Q6. In total, 567 samples were collected. It should be noted that because both Madero buildings are treated as a single building in the floor plans, this was respected in the applications. And because, thanks to this, there is only a single building in all of the samples, results for building accuracy are not be presented in this section.

With the goal of finding the best parameter configuration, several tests were run while varying a single parameter at a time. These tests were done using the location filtering algorithm explained in section 3.1 for every collected sample, recording the error between the resulting position and the real position in the database. After each of these tests, the parameter that gave the best results was chosen and used for the subsequent tests. The modified parameters were the ones already presented with the algorithm in the section 2.3, which can be seen in Table 1.

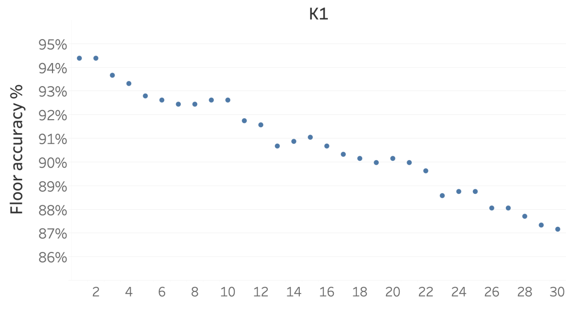
Table 1: Parameters of the positioning algorithm.

Parameter	Description
K_1	Maximum number of samples to use for the simple majority when calculating floor.
K_2	Maximum number of samples to use for the centroid when calculating coordinates.
N	Minimum number of samples to use in floor calculation before defaulting to all of the samples in the building, using R_1 as R_2.
M	The number of APs of each fingerprint to use when calculating R_2.
Default RSSI	RSSI value to use in fingerprint similarity calculation, when one of the them does not have a value for a certain AP present in the other.

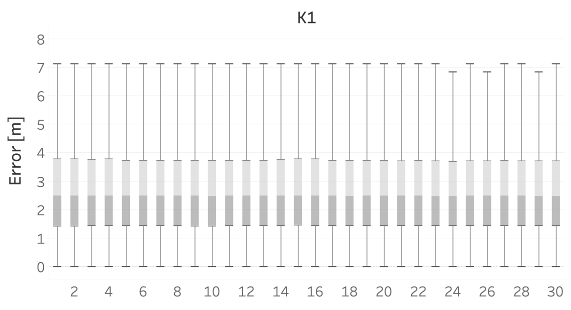
For the following analysis, horizontal error was computed for the cases where the location algorithm returned the correct floor. This is because if the floor is incorrect, the user won't care what position they are shown in, because the map will be incorrectly positioned anyway. This is also done because the algorithm only uses the samples of the floor it correctly predicts in coordinate calculation.

The starting values of the parameters were: $K_1 = 3$, $K_2 = 3$, $M = 3$ and $N = 3$. $RSSI$ = -90 was used in all of the tests, because this represents an AP with almost no signal at all. After each set of tests, each parameter was replaced with the best value found.

First K_1 was varied with $K_2 = 3$, $M = 3$ and $N = 3$. The results, displayed in Figure 8a, show an inverse proportion between K_1 and floor accuracy. As expected, K_1 does not significantly affect coordinate precision as seen in Figure 8b but takes an important role on the inference of the floor. Best average results were found for $K_1 = 1$. $K_1 = 2$ had the same results as $K_1 = 1$, this is because even in the case of a tie, floor calculation uses first value which is the one used for $K_1 = 1$. This effectively makes both values of K_1 the same.



(a) Correct floor percentage, varying K_1.



(b) Mean horizontal error, varying K_1.

Figure 8: Floor accuracy and horizontal error varying K_1.

With K_1's best candidate the next step was to find the best value for K_2. Parameter values used were $K_1 = 1$, $M = 3$ and $N = 3$. Because K_2 is used after obtaining the floor this parameter does not affect floor accuracy, but as seen in Figure 9, it greatly influences coordinate accuracy, with $K_2 = 2$ as the best fit.

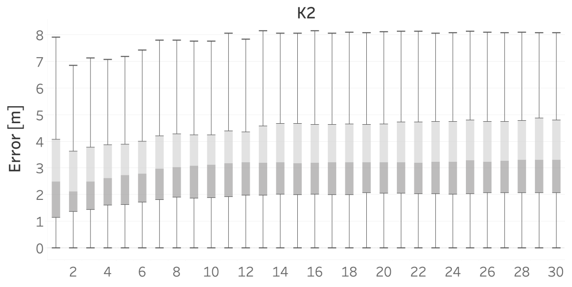
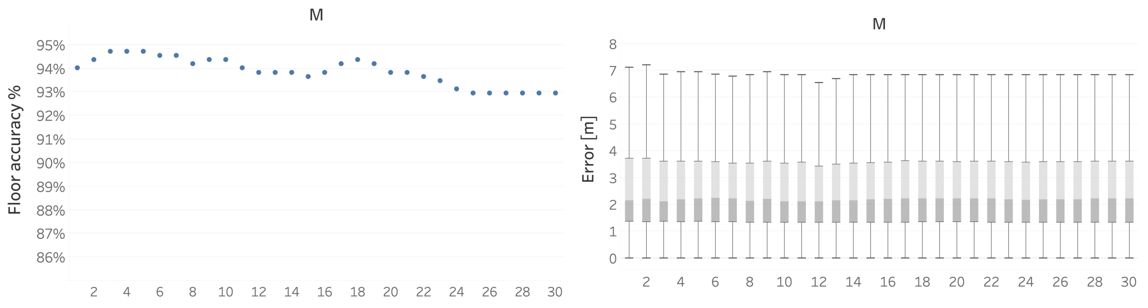


Figure 9: Horizontal error, varying K_2.

After finding K_2, M was next. Parameter values used were $K_1 = 1$, $K_2 = 2$ and $N = 3$. Floor precision was found to be best for values 3, 4 and 5, as shown in Figure 10a. $M = 4$ was chosen as the best candidate after analyzing horizontal error in Figure 10b.



(a) Correct floor percentage, varying M.

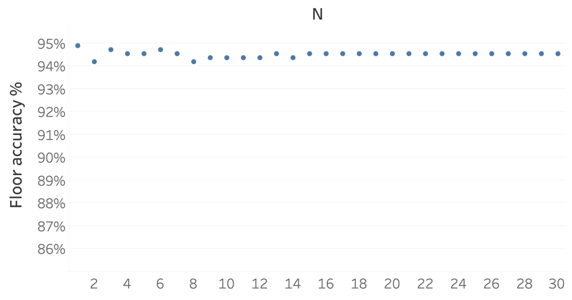
(b) Horizontal error, varying M.

Figure 10: Floor accuracy and horizontal error, varying M.

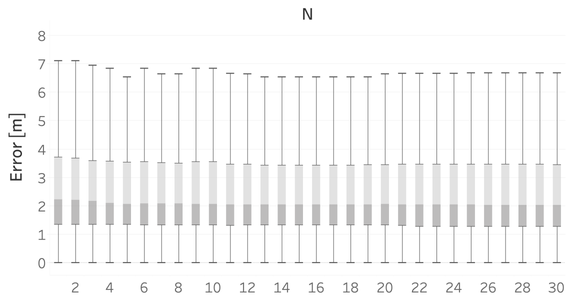
N was the last varied parameter with $K_1 = 1$, $K_2 = 2$ and $M = 4$. Although $N = 1$ has the greatest horizontal error (Figure 11b), its difference with the smallest one, $N = 15$, was just around 0.15 m. As before, floor precision was prioritized over horizontal precision, and so $N = 1$ was chosen.

After analyzing each set of runs, the best parameter configuration was to be the following: $K_1 = 1$, $K_2 = 2$, $M = 4$ and $N = 1$. This configuration's floor precision was 94.89% and its average horizontal error 2.87 m.

As stated before, GPS-enabled smartphones are typically accurate to within a 10-meter radius approximately. As shown in Figure 12, 99% of the time error is below 10 m and, 40% of the time below 2 m, showing that better accuracy is achieved when using this implementation over GPS alone. It is also important to note that GPS has less vertical accuracy than it does horizontal (Applied Research Laboratories, 2019), although regardless the system itself does not have enough information to estimate floor number since every building has



(a) Correct floor, varying N.



(b) Coordinates Error, varying N.

Figure 11: Floor accuracy and horizontal error varying N.

different starting height and individual floor height. Results show that this implementation does locate the current floor within the tested building with 94.89% accuracy.

Finally, these results can be compared with the ones obtained by RTLS@UM. In their paper they report 93.74% floor accuracy rate, while our implementation showed a similar value as stated above. Meanwhile, their horizontal error is shown to have a mean error of 6.20 m, but this is not comparable to these results mainly because RTLS@UM's algorithm adds an error penalty of 4 m when the floor was not located correctly while this project's implementation does not.

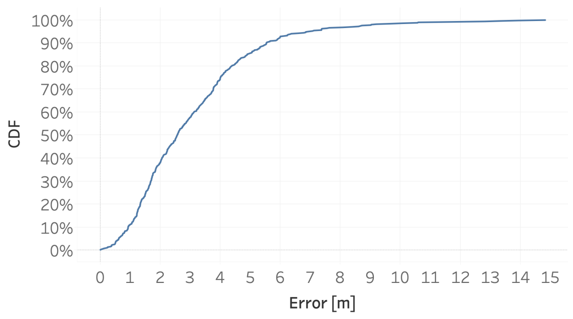


Figure 12: Cumulative distribution function coordinates error for best parameters: $K_1 = 1$, $K_2 = 2$, $M = 4$ and $N = 1$.

5. Conclusion

This was an interesting project that allowed for a first dive into IPS. Results were found to have a similar error to those of RTLS@UM, which means the solution works fairly well, although there is still a lot of room for improvement. Even if these errors are low from a scientific point of view, they have not yet arrived to a point where they are acceptable for the average user, which would need an error accurate to the meter with an almost perfect floor accuracy.

Future work on this project could involve improving the usability of the applications, as well as their functionality. To improve the quality of the user application, for example, the map camera could be exclusively controlled by the user, rather than being forced to center on the estimated location when found. It was not uncommon to be exploring the map all to be abruptly moved to a different location, potentially with a different zoom level. To make the experience smoother, the user app could be limited to simply show a marker on the estimated location, and override the My Location button to center on the marker rather than on the system estimated location. Additionally, the floor selector could be enabled to give the ability to the user of exploring the building, while highlighting the currently located floor. From the administration side, floor plans must be manually uploaded and aligned to the world map by trial and error, but an iteration of the web admin app could offer an interactive map uploader to simplify the process. These kinds of changes would greatly streamline the process for administrators, which would in turn make location services available more quickly to end users.

Another area that could be worked on is new features, like allowing administrators to associate names with each section of the building such as classroom numbers for universities, shop names for shopping malls, etc. This data could be displayed in the user app, or be used as search filters. If the layout of the building were deeply detailed, this could even enable pathfinding and routing between arbitrary points.

As final work, it would also be interesting to test more of the algorithms and ideas available for IPS. To combine them and take the best of each one, to make our own approach to problem with the goal of getting even better results.

6. Acknowledgements

As a final note, we would like to acknowledge all the people who helped us complete this work. First and foremost we want to give our thanks and appreciation to our tutor, Pablo Ignacio Fierens, without whom none of this would be possible. Secondly, we would like to express our gratitude towards Santiago Vallés, Director of Software Engineering Department, and Trebotic Jaqueline Solange, Architect of Infrastructure Department, who were invaluable in helping us acquire ITBA's floor plans. We would also like to extend our thanks to Valeria I. Soliani, Final Project Coordinator of Software Engineering Department, who patiently guided us through the process of the final project from start to finish. And finally, we want to thank our families and friends who supported us all of these years, and for that we are eternally grateful.

References

Android Developers, 2019. URL `https://developer.android.com/reference/android/net/wifi/ScanResult.html#level`. Visited on November 2019.

The University of Texas at Austin Applied Research Laboratories, 2019. URL `https://www.gps.gov/systems/gps/performance`.

Leor Banin, Uri Schatzberg, and Yuval Amizur. Wifi ftm and map information fusion for accurate positioning. In *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2016.

JP Grisales Campeón, S López, SR de Jesús Meleán, H Moldovan, DR Parisi, and PI Fierens. Indoor positioning based on rssi of wifi signals: how accurate can it be? In *2018 IEEE Biennial Congress of Argentina (ARGENCON)*, pages 1–8. IEEE, 2018a.

JP Grisales Campeón, Sebastián López, Sergio R de Jesús Meleán, Horatiu Moldovan, Daniel R Parisi, and Pablo Ignacio Fierens. Fusion of magnetic and wifi fingerprints for indoor positioning. In *2018 Congreso Argentino de Ciencias de la Informática y Desarrollos de Investigación (CACIDI)*, pages 1–5. IEEE, 2018b.

Eskimo. Re: List available wifi access points in the same network, 2017. URL `https://forums.developer.apple.com/message/276151#276151`. Visited on November 2019.

Eskimo. How to identify internet signal strength inside my custom application, 2019. URL `https://forums.developer.apple.com/thread/126001`. Visited on November 2019.

HERE. Indoor positioning systems & navigation, 2019. URL `https://www.here.com/products/tracking-positioning-solutions/indoor-positioning-systems`. Visited on November 2019.

Basil Hess, Armin Zamani Farahani, Fabian Tschirschnitz, and Felix von Reischach. Evaluation of fine-granular GPS tracking on smartphones. In *Proceedings of the First ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, pages 33–40. ACM, 2012.

IEEE. IEEE standard for information technology-telecommunications and information exchange between systems local and metropolitan area networks-specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2016*, pages 1–3534, Dec 2016. doi: 10.1109/IEEESTD.2016.7786995.

insoft GmbH. Solutions for real-time locating systems (RTLS) by infsoft, 2019. URL `https://www.infsoft.com`. Visited on November 2019.

Christopher Langlois, Saideep Tiku, and Sudeep Pasricha. Indoor localization with smartphones: Harnessing the sensor suite in your pocket. *IEEE Consumer Electronics Magazine*, 6(4):70–80, 2017.

Gough Lui, Thomas Gallagher, Binghao Li, Andrew G Dempster, and Chris Rizos. Differences in RSSI readings made by different Wi-Fi chipsets: A limitation of WLAN localization. In *2011 International Conference on Localization and GNSS (ICL-GNSS)*, pages 53–57. IEEE, 2011.

Krista Merry and Pete Bettinger. Smartphone GPS accuracy study in an urban environment. *PloS one*, 14(7), 2019.

Adriano Moreira, Maria João Nicolau, Filipe Meneses, and António Costa. Wi-Fi fingerprinting in the real world - RTLS@UM at the EvAAL competition. *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2015.

Proximi.io. Accurate indoor positioning, 2019. URL `https://proximi.io/accurate-indoor-positioning/`. Visited on November 2019.

Frank Van Diggelen and Per Enge. The world's first gps mooc and worldwide laboratory using smartphones. In *Proceedings of the 28th international technical meeting of the satellite division of the institute of navigation (ION GNSS+ 2015)*, pages 361–369, 2015. URL `https://www.gps.gov/systems/gps/performance/accuracy`. Visited on November 2019.

Jules Wang. Android started heavily throttling Wi-Fi scanning in Pie, Google confirms it's here to stay, 2019. URL `https://www.androidpolice.com/2019/07/10/android-started-heavily-throttling-wi-fi-scanning-in-pie-google-confirms-its-here-to-stay/`. Visited on November 2019.

Wi-Fi Alliance, 2019. URL `https://www.wi-fi.org`. Visited on November 2019.

Paul A Zandbergen. Accuracy of iPhone locations: A comparison of assisted GPS, WiFi and cellular positioning. *Transactions in GIS*, 13:5–25, 2009.