Internal Positioning System (IPS) Application For Android

Andre Bododea
s1350924

Part 1: User Guide

The purpose of this paper is to discuss an Indoor Positioning System (IPS) application designed for Android. GPS cannot function accurately within a building, therefore there is a strong need for IPS to take over tracking the user from the GPS once he steps within a building. This application attempts to solve this problem by using RSS scans from WiFi access points. The user must first train the floor plan of the building he/she is currently within. The RSS values for positions on the map are recorded in persistent memory via a database as the user walks around performing the training. Once enough points are collected, the user can then use the application in order to track themselves as they walk around the building.

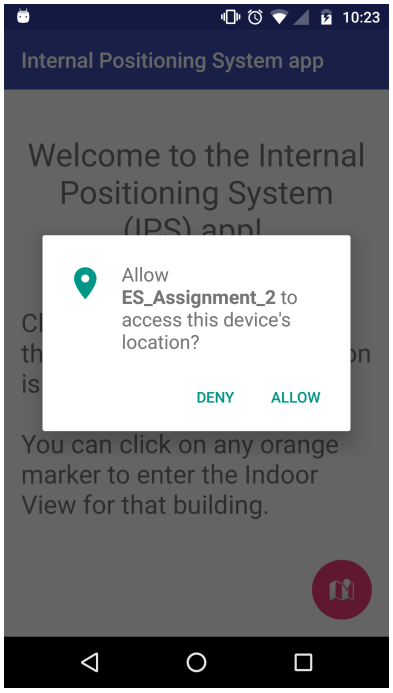

Figure 1: Requesting user permissions

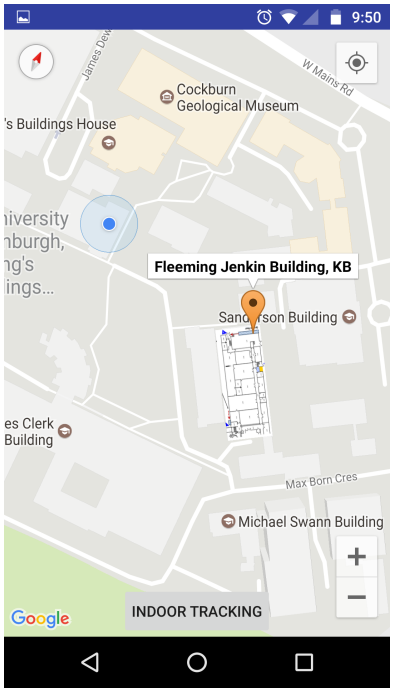

Figure 2: Current location on Google map view

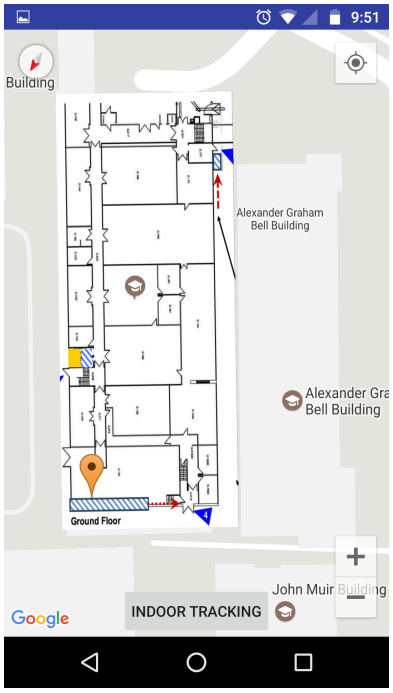

Figure 3: Building with floor plan overlayed

Immediately after installation, as the user opens the app a prompt opens asking them to enable location permissions (**Fig 1**). This permission is integral to the app's ability to function, as without it we cannot complete our scans. If the user does not accept, the app will not be launched. After this, the user is presented with a standard Google map. The user can press the location button in the top right-hand corner to zoom in on his/her current location (**Fig 2**). If this location is near a building with a floor plan overlayed, there will also be an option to click on an orange marker to enter the *indoor view* (**Fig 3**). Once they have reached this indoor view, training instructions for the floor plan for that building will be shown. The user is told to draw their desired path that they will then follow in order to complete the training of the database.

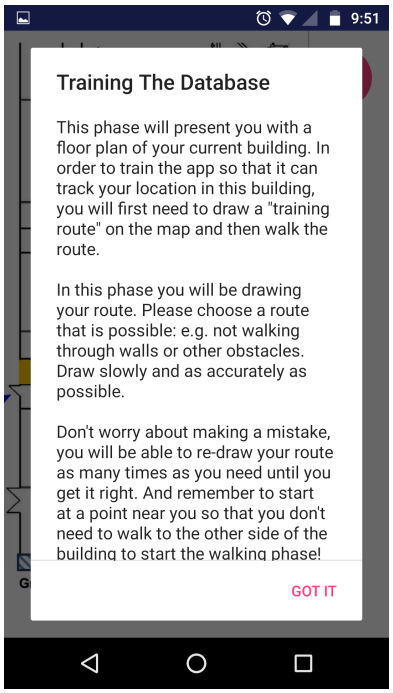

Figure 4: Training instructions once in indoor view

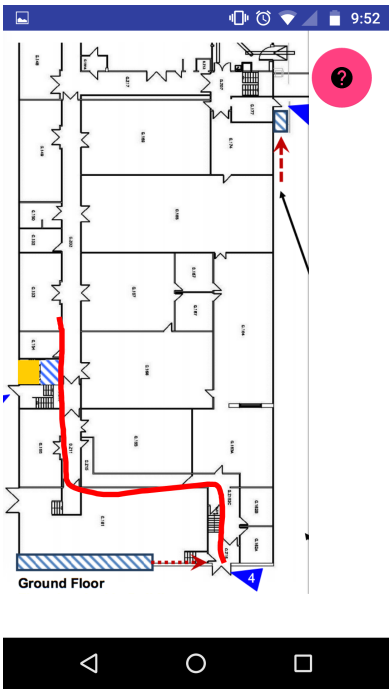

Figure 5: User drawing their chosen training path

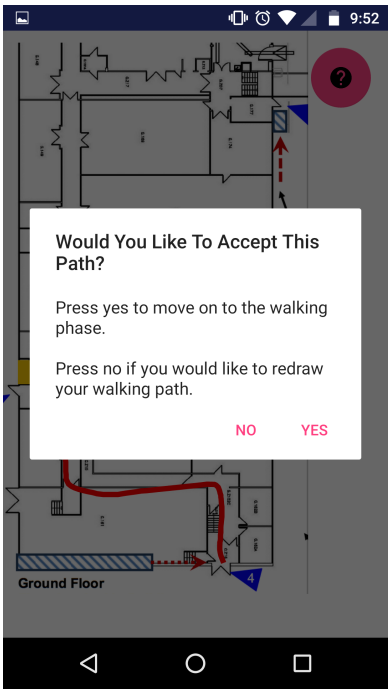

Figure 6: Prompt to accept the path

Once the user has drawn a path (**Fig 5**), he/she must then follow this path as an animated green dot walks them through the path (**Fig 8**). They must try to follow the green dot as closely as possible in order to accurately train the data set.

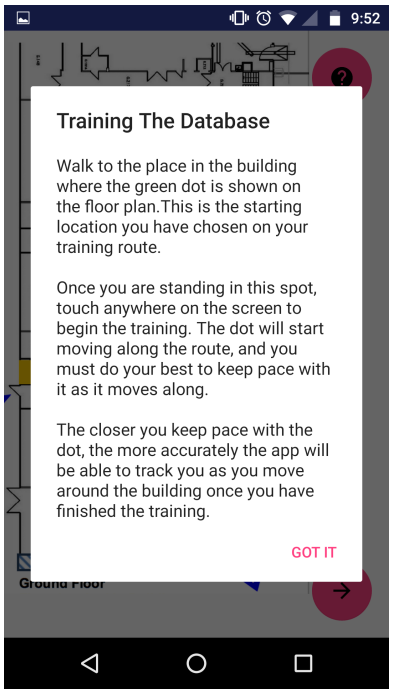

Figure 7: Training instructions for following the training animation

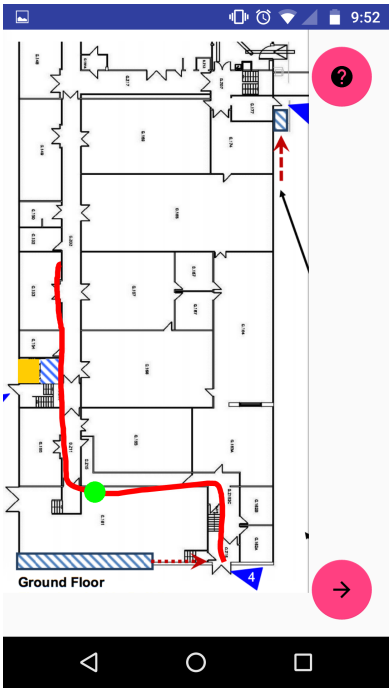

Figure 8: Training animation in progress, user is walking path

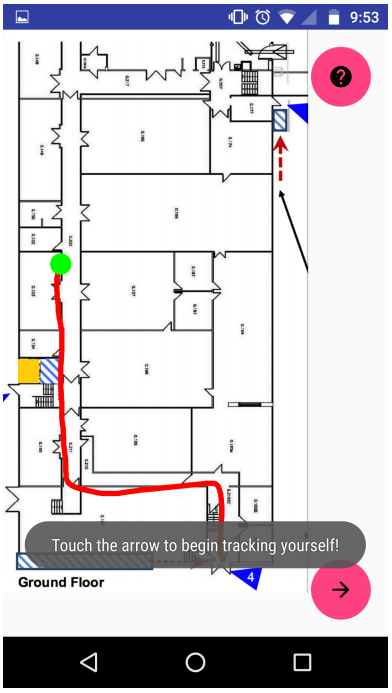

Figure 9: End of path reached, training is now finished

Once the user has drawn a path then physically walked that path in order to train the database on a set of scan points, he/she can now use the floor plan to track his/her position in the building *for the region trained in their path* (**Fig 10** & **11**). The app will attempt to find the closest trained point from the database, and

match it to the user's current position on the map via WiFi scans.

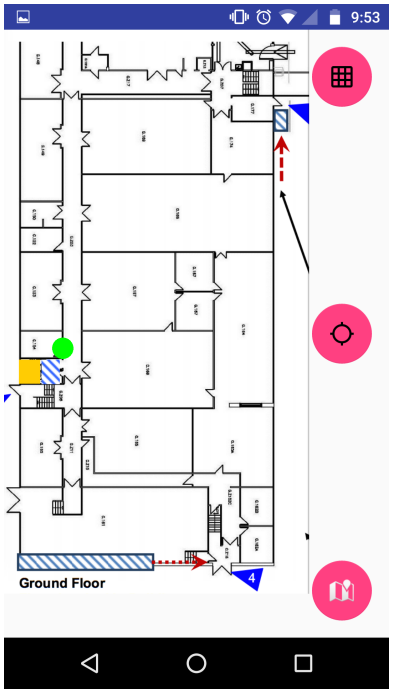

Figure 10: Tracking the user's position on the floor plan while moving around building

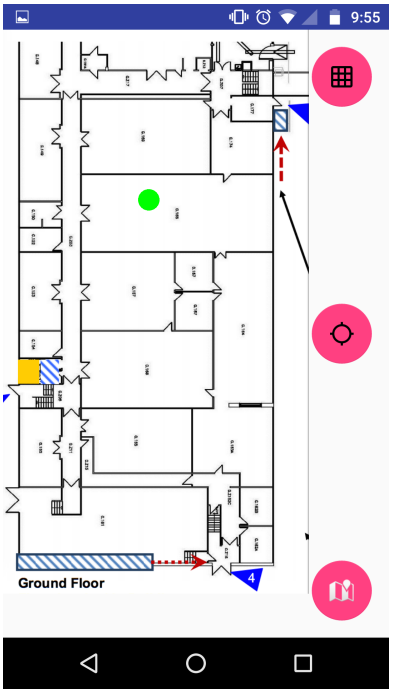

Figure 11: After another few sets of trainings, the user can have a wider range on the map

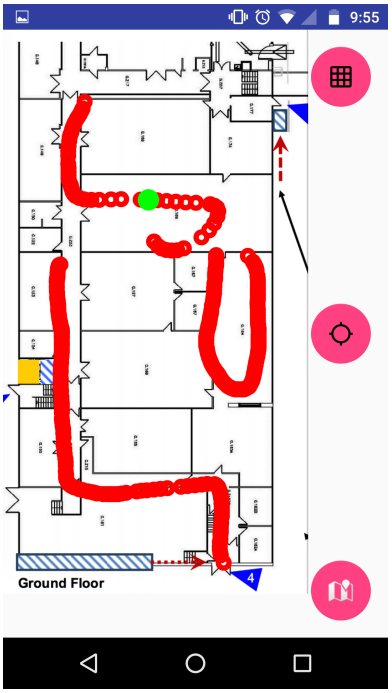

Figure 12: "Grid" button shows all previously recorded paths in the database

The user can record as many different paths as they want, and use the "grid" button in the upper right-hand corner to view all the paths already trained (**Fig 12**). This gives them an idea of what parts of the building tracking is available and what parts have yet to be trained. The button with the map icon in the lower right-hand corner takes us back to the map view from **Figs 2** & **3**.

With this simple, easy to use interface, along with an animated training system that allow users to train any building quickly and flexibly, this app requires no technical know-how and makes indoor positioning simple and accessible for casual users.

Part 2: Developer Guide

This application was designed to be *highly modular*. Each screen has its own Activity class, and important values are passed via Intent to the next Activity class in the call-order.

Techniques for localisation using radio signal can take advantage of numerous technologies - Bluetooth, WiFi, and cellular signals just to name a few [1]. Localisation via WiFi was chosen for the simple reason that WiFi hotspots are available in most public buildings, and no additional infrastructure setup is needed to work with them [2].

This app was designed to take advantage of WiFi access points, or more specifically RSS scans. The concept of triangulating via RSSI scans is certainly not a new or particularly complex one. It was explored at length as early as 2004 by Kaemarungsi et all [3], however advances in RSSI localisation algorithms continue to be made [4, 5] and continue to show it to be the most powerful and flexible tool for IPS available today.

Implementation: Sensors and APIs

Google Maps API

In order to be able to have such a beautiful and responsive map interface, we have implemented our outdoor map using Google Maps Android API [6]. Along with being a great app that allows tonnes of features such as pins and floor plan overlay (**Figs 2** & **3**), using the Google Maps API allows the added bonus of user being familiar with the layout and use of the map.

WifiManager

Once we are in our *inside view*, the most important tool we have is the WifiManager. This API allows us to manage all aspects of Wifi connectivity

The most important use for WifiManager in this app is getting nearby access point addresses and their corresponding signal strengths. The code below shows how we use the WifiManager to get these values for the location where the user is currently standing.

```
WifiManager wifiManager  = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
List<ScanResult> wifiList = wifiManager.getScanResults();

ArrayList<String> networkAddresses = new ArrayList<String>();
ArrayList<Integer> signalStrengths = new ArrayList<Integer>();

// Add the BSSIDs and corresponding signal strengths to the lists
for (ScanResult scanResult : wifiList) {
    networkAddresses.add(scanResult.BSSID);
    signalStrengths.add(Math.abs(scanResult.level));
}
```

Listing 1: Get BSSID values and corresponding signal strengths at current location via the WifiManager API

Once we have these values, they must be stored somewhere. In order to be used over the lifetime of the app, and not just for the current session, the data storage must be *persistent*. The best form of persistent storage for us in a small application situation such as this one is a database.

Database

Android ships with a powerful API for interfacing with SQL databases, called SQLite [7]. We design a database to store the values found from the WifiManager, stored for each point on the map. This database must store the point on the floor plan, denoted by an x-y coordinate pair. It must also store all the BSSIDs and corresponding signal strengths at that point on the floor plan.

PointID	xCoord	yCoord	BSSID	SignalStrength
1	1003.789018	7314.123744	00:08:C7:1B:8C:02	79
1	1003.789018	7314.123744	20:29:B1:1A:5A:03	86
1	1003.789018	7314.123744	02:28:A1:1A:22:31	61
2	2991.345891	3423.812379	12:08:C7:1C:8B:02	64
2	2991.345891	3423.812379	20:29:B1:1A:5A:01	72
3	7237.345891	1234.567891	21:61:C2:8B:5A:B1	40
.	.	.	.	.

Table 1: Format of our SQL table for storing coordinates, BSSIDs, and SignalStrengths

Each x-y point is associated with a unique PointID. This ensures no duplicates are chosen in querying the database when we are getting the nearest point based on the BSSID values. At each training point along the training route (the animation that the user is meant to walk along with), we query the WifiManager for all

available BSSIDs and their corresponding signal strengths. These values are then stored in a table such as the one above. We can then query the table at a time when we need to access these data points, such as finding the nearest recorded data point to the user's position as in **Figs 10** & **11**.

Finding The Nearest Point: K-nearest Neighbours

We have training data stored in the database as discussed in the previous section, however we still have not discussed how we access this data and how we use it when predicting the user's nearest position on the map via RSSI signal strength. We use a modified version of the popular k-nearest neighbours method to do this.

We begin by getting the BSSIDs and signal strengths of our current location as in Listing 1 above. Then we use these values, and cycle through the database using SQL commands to extract the entries with matching BSSIDs. Once we have extracted these entries we simply extract the corresponding signal strength, and compute the Euclidian distance between that signal strength and the strength at the current position.

By doing this over the entire set of points in the database, we can find the entry with the lowest mean distance between its signal strengths and the current signal strength across all BSSIDs. This point is considered the nearest point, and we simply use a query to extract the x and y values of the coordinates and then subsequently plot these values as a green dot on the floor plan (seen in **Figs 10** & **11**)

Conclusion

Overall, this application is robust to changing environments as WiFi is so widespread a technology. Further steps could involve inclusion of other data - EMF (Magnetic Field), Radio Frequency, Bluetooth. An interesting further step would be literally that - including a step counter to not only assist with accurate pacing of the user with the training animation, but to eventually replace the animation with a self-training step counter based implementation that trains as you walk around the building. All of these additional technologies will be considered for inclusion in version 2 of this application. As a first release, however, this IPS application can serve as a gentle introduction to the average user to IPS through the use of easy to use and understand animations guiding them through the process and even making it fun.

References

[1] Maxim Shchekotov. Indoor localization method based on wi-fi trilateration technique. In *Proceeding of the 16th conference of fruct association*, 2014.

[2] Alexey Kashevnik and Maxim Shchekotov. Comparative analysis of indoor positioning systems based on communications supported by smartphones. In *Proceedings of FRUCT conference*, pages 43–48, 2012.

[3] Kamol Kaemarungsi and Prashant Krishnamurthy. Modeling of indoor positioning systems based on location fingerprinting. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1012–1022. IEEE, 2004.

[4] F Alsehly, R Mohd Sabri, Z Sevak, and T Arslan. Improving indoor positioning accuracy through a wi-fi handover algorithm. In *International Technical Meeting, The University of Edinburgh, Scotland, UK*, pages 131–139, 2010.

[5] O Oguejiofor, V Okorogu, Abe Adewale, and B Osuesu. Outdoor localization system using rssi measurement of wireless sensor network. *International Journal of Innovative Technology and Exploring Engineering*, 2(2):1–6, 2013.

[6] Google. Google maps android api. `https://developers.google.com/maps/documentation/android-api/`.

[7] Google. Sqlite android api. `https://developer.android.com/reference/android/database/sqlite/package-summary.html`.