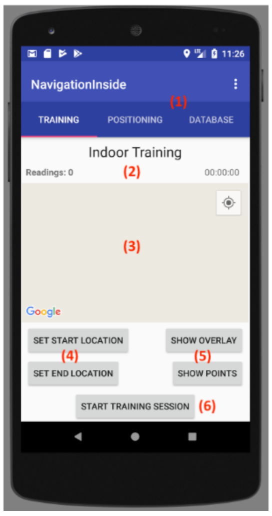
Indoor Navigation Mobile App

User Guide

App functionality and Interface

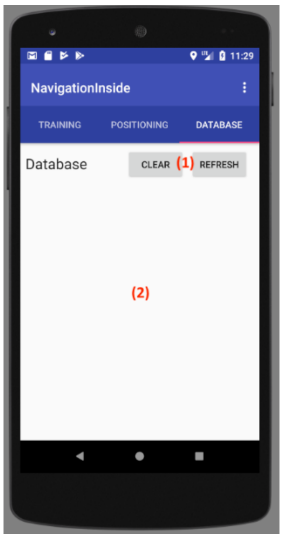
The *Indoor Navigation App* allows a user to find their location on a Google Map, using GPS information outside and a custom WiFi based triangulation system while indoors. This indoors system must first be trained, to collect some reference points within the desired buildings. The interface is split into three tabs which can be toggled between by swiping to the side of the screen or by tapping on the tab name in the action bar at the top of the interface.

Training Tab



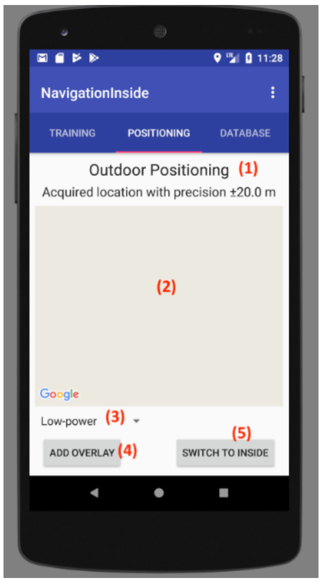
Each section of the app is selected with the tab bar in (1). The first tab allows for indoor '*reference points*' to be gathered. A map is displayed in the center of the screen at (3). To begin a training session, the user should first navigate in the Google Map to the location they wish to train. Next, the 'Set Start Location' button (4) should be pressed, followed by tapping the map in the desired position. The same process should then be followed to set the end position. A red line will be drawn between these two markers. This is the path which the user should walk along during the training session. For best accuracy, a constant pace should be followed as they walk between the points. A faster pace will result in fewer reference points. The 'Start Training Session' button (6) begins the session, and the UI timer and readings counter in (2) will begin to update. Once the user reaches the end they must press the 'Stop Training Session' button (also (6)). To assist with selecting the training path, the buttons at (5) can be used to toggle an indoor floorplan overlay onto the map or to display all existing recorded points.

Database Tab

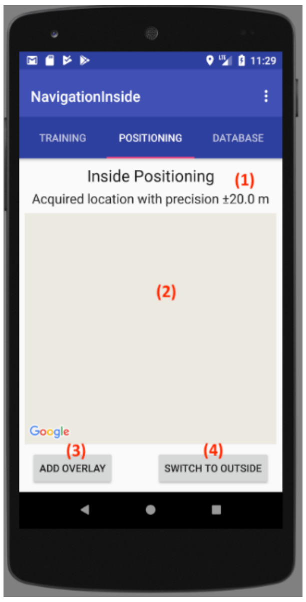


The third tab is useful for debugging the reference points. A list of all reference points with their position (latitude and longitude) and WiFi networks (BSSID and level) is shown in a scrollable text field in (2). Buttons in (1) allow the user to remove all existing points (Clear) or to manually reload the database (Refresh).

Positioning Tab



The core functionality of the app is found in the Positioning tab which starts in outdoors mode as shown in the image on the left. The button at (5) toggles to indoors mode (image on the right). In both cases, the user's location will be calculated and shown on the Google Map at (2) and the accuracy of the result will be described in (1). The indoor floorplan overlay can again be toggled using the 'Add overlay' button. In outdoors mode, a dropdown menu (3) allows for the level of accuracy to be adjusted. Higher accuracy requires more power so in some situations it may be useful to operate in a low-power mode. In indoors mode, the location shown will be the reference point which most closely matches the user's current conditions.



Potential Uses

Indoor venues could gather the training data and indoor floorplan information so that their potential customers/visitors could use this app to navigate around. Users can also add their own custom locations such as their homes by using the training tab themselves.

Loading and running the app

The app was tested on an *Archos 5* device running Android Jelly Bean 4.3.1. Updated *Google Play Services* are required by the app to use the Google Maps and Location features. It has been designed to support Android API levels 18-27. Android Studio 3.0.1 was used to develop the project.

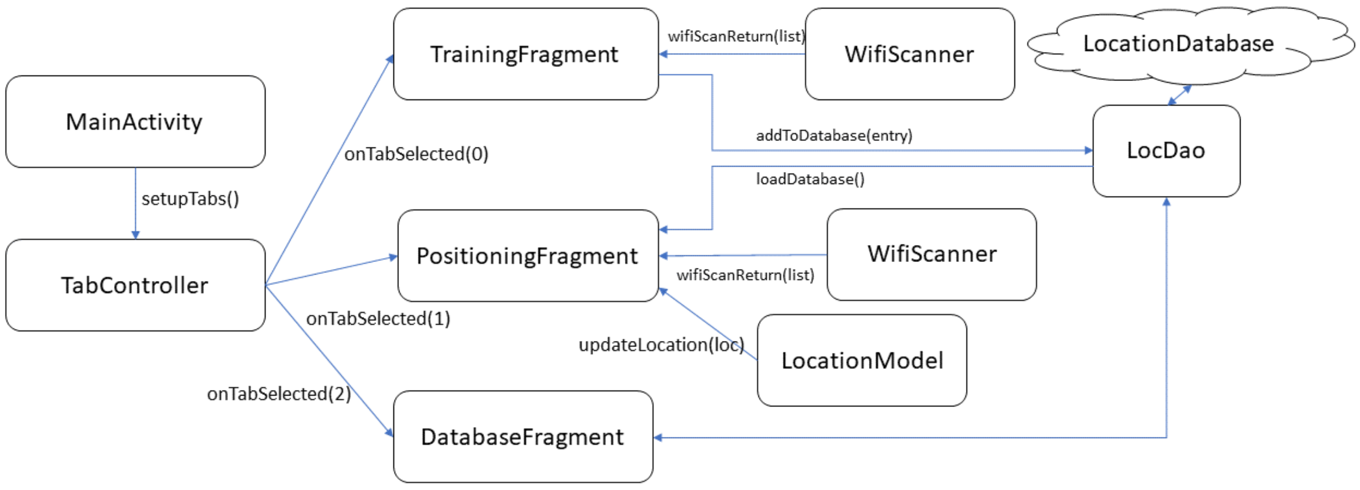
An APK file (**.apk**) of the app can be installed directly onto a mobile device, or the full project can be compiled using Android Studio to run the app through a software emulator. However, the critical WiFi and location features are difficult to emulate so this is not recommended.

Programmer's guide References

[1] – Room API, https://developer.android.com/training/data-storage/room/index.html
[2] – Google Maps API, https://developers.google.com/maps/documentation/android-api/start
[3] – Location Updates, https://developer.android.com/training/location/receive-location-updates.html

Programmer's guide

The application is highly modular and spread between several Java class files. An overview of the flow of information in the core operation of the application is given in the diagram below:



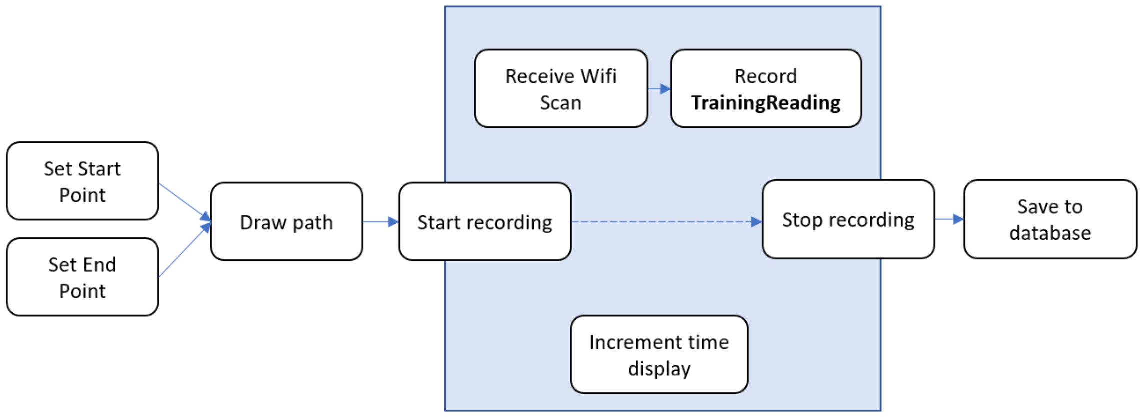
The core application Activity is defined in **MainActivity.java**. This handles checking that the user has provided all the necessary Location and WiFi permissions to the app with *checkAllPermissions()*. It also initializes the reference point database using the Room API [1] method *databaseBuilder()*. It then instantiates a new TabController (**TabController.java**) to use as the adapter for a ViewPager object which governs the three-tab interface. This loads the appropriate Fragment when the user selects a specific tab in the user interface. To support the Google Maps features, a developer API key is needed which should be added to the **AndroidManifest.xml** file [2]. All UI design was done visually to create a *ConstraintLayout* XML file for each Fragment.

Training

The first tab is for the training mode of the application. The implementation is in **TrainingFragment.java**. A GoogleMap is the core element of this tab, and it is instantiated within a MapView UI object [2]. The *MapsInitializer.initialize()* method sets up the Google Map and then the *getMapAsync* method requests the MapView to be filled by the GoogleMap once it is ready. The map is customized to allow the native location feature and the camera is set to Kings Buildings by default. A MapClickListener is added so that the start and end training points can be selected by tapping on the map. There are five Buttons which responds to being pressed using the *onClick* method of the *onClickListener* interface which this Fragment implements.

A WifiScanner is instantiated (**WifiScanner.java**) which uses the device WifiManager to implement a WifiScanReceiver. This sends the scan results back to the Fragment each time they are received but they are only used while a training session is being recorded.

Buttons to set the start/end point and to start/stop the recording session make up the primary function of recording training data. This operation is shown in the flowchart:



On pressing the set start/end point button the user can then press on the map which will create or move the relevant Marker to that location. If both exist then a red Polyline is drawn between them to show the training path.

The user should then press the 'Start recording' button to enter training mode. During this time, the app then listens for Wifi scan results and records them as a TrainingReading (**TrainingReading.java**) which couples the returned List of ScanResults with the time of the reading. An on-screen stopwatch is also continually updated using the Runnable method *stopWatchRun* and a *Handler* which executes it as *stopwatchHandler.postDelayed (stopWatchRun, 0).*

Upon a second press of the recording button, the collected TrainingReadings are iterated through and saved to the Room database. The location of each reading is found by interpolating between the start and end location using the relative time of the reading in the *stopStopwatch()* method. The Room database is accessed using the LocDao interface method *insertOne()* which runs in an Asynchronous task from the *addToDatabase()* method.

Two additional buttons provide other useful functionality. The toggle overlay button toggles a semi-transparent indoor floorplan map to be displayed over the correct locations in the Google Map. This is done with a GroundOverlay and the *addGroundOverlay* method of the Google Maps API. The toggle points button allows for all the existing reference points captured to be displayed as Markers on the map. This is done with the *addMarker()* method of the Google Maps API.

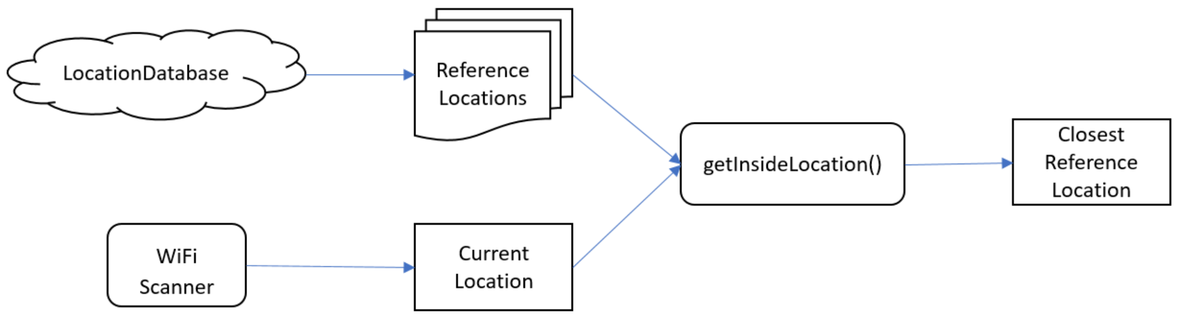
Positioning

This tab (**PositioningFragment.java**) allows the user to locate their current position using a Google Map. This map is similarly set up as in the Training Fragment. A button allows the floorplan overlay to be toggled as was the case in the Training Fragment. A second button toggles between GPS based outdoor positioning and the custom Wifi indoor triangulation.

For the outdoor positioning, a separate class, defined in **LocationModel.java** handles the use of the Android *FusedLocationProviderClient* API [3]. Location updates are requested using the *requestLocationUpdates()* method of the *FusedLocationProviderClient* API. These are then handled using a LocationCallback method which sends the new location back to the

PositioningFragment using the *updateLocation(pos)* method. The accuracy and minimum interval between updates can be set by the user to one of three tiers using the UI dropdown menu from a low-power, low-accuracy mode to a balanced one then up to the maximum accuracy. The *modifyLocationRequest()* method allows for these options to be tweaked. When a location is received by the Location Fragment, a Marker is placed upon the Google Map to show this with the *displayPosition()* method and the accuracy of this reading is reported to the user in a TextView above the map. The accuracy can be found by calling the *getAccuracy()* method on the new Location object.

Indoor positioning can be activated with the toggle button. This activates the listening for Wifi results from another **WifiScanner.java** object. As in the Training Fragment, the scanner reports back each time a scan completes with the *wifiScanReturn()* method. The received List of ScanResults are then compared with the database of recorded reference points in the critical method of the application *getInsideLocation()*:



This method is where the indoors positioning algorithm is implemented. The List of reference points is iterated through and each is assigned a score. The point with the highest score is set as the user's most likely location. To assign a score to a point, the three strongest Wifi BSSIDs are compared with the three strongest of the access point. If there is a match then the score is incremented by 100 minus the difference in dB level of the signal. Therefore, a closer match will yield a greater score. Some weighting was added to try to further tune the system. The strongest access point received will contribute more than the second strongest and that contributes more than the third. These weights are currently set to 100%, 60%, 30% which yielded good results in accurately identifying the user's location. Further work could be done to tune this algorithm for maximum performance by adjusting these weights or using more than the three strongest access points.

Database

For debugging, the third tab implements a basic scrollable TextView which displays the content of the Room database in String form. The implementation is contained within **DatabaseFragment.java**. Two Buttons are handled with onClickListeners to perform a deleteAll or loadAll operation from the Room database. These are again performed as Async Tasks. The resulting database information is formatted into a long String in the *PostExecute()* method of the load task so that it can be displayed in the scrollable TextView.