

Antenna Alignment Augmented Reality Video Application

Final Year Project

Chris cornwall -12311016

15/04/2018

Contents

[Introduction 2](#_Toc510356773)

[Alignment of Highly Directional Microwave Antennas 2](#_Toc510356774)

[Augmented Reality 2](#_Toc510356775)

[Literature Review 2](#_Toc510356776)

[Description 3](#_Toc510356777)

[State of the Art Review 3](#_Toc510356778)

[Design 3](#_Toc510356779)

[Implementation 4](#_Toc510356780)

[Nativedetailscreen.js 5](#_Toc510356781)

[Marker.js 6](#_Toc510356782)

[Signal.js 6](#_Toc510356783)

[Radar.js 8](#_Toc510356784)

[Backend 8](#_Toc510356785)

[Evaluation 9](#_Toc510356786)

[Additional Requirements 11](#_Toc510356787)

[Conclusion 13](#_Toc510356788)

[Works Cited 13](#_Toc510356789)

# Introduction

## Alignment of Highly Directional Microwave Antennas

In recent years, there has been a steady growth of wireless infrastructure across the globe, fixed wireless is now an attractive solution to many businesses and homeowners, who, in previous years, may have had difficulty obtaining and maintaining a quality broadband connection.

To achieve these high speed fixed wireless connections, a radio signal is sent from a transmitter antenna to a receiver antenna via highly directional microwave antennas.

As the name suggests, these antennas are highly directional, meaning that the signal they transmit and receive performs well over relatively large distances. However, as a side effect of this increased directional power, they must be aligned in both the horizontal and vertical plane to a high degree of accuracy in order for the receiving antenna to obtain an adequate signal.

Typically, when an ISP technician installs a highly directional microwave antenna, they use a specialist piece of equipment, much like a multimeter, to read real-time signal data. Whilst this real-time information can be used to correctly align the antenna in both the horizontal and vertical planes, it does require some trial and error initially so that the antenna can be aligned in the correct general direction.

## Augmented Reality

Augmented reality can be defined as “An enhanced version of reality where live direct or indirect views of physical real-world environments are augmented with superimposed computer-generated images over a user's view of the real-world, thus enhancing one’s current perception of reality.”

# Description

“Antenna Alignment Augmented Reality Video Application (Smartphone or Drone derived Video). The goal is to develop a novel smartphone application that uses augmented reality techniques to help with the alignment of highly directional microwave dish antennas. High speed fixed wireless point to point links use very high gain and directional dish type antennas that have often had a very narrow beamwidth so precise alignment in both the horizontal and vertical planes can be very difficult, especially when working at height on a telecoms tower or rooftop. The application would be used to provide visual clues to help engineers with the initial alignment of the antenna as well as information on the expected signal strength based on the distance and power budget calculations. The video source could be the camera on a mobile phone or video being streamed live from a GPS enabled drone.”

# Literature Review

In an effort to conduct the relevant research for this project I identified the following areas as being particular areas of interest:

• Development of an Android application using Android Studio

• Development of an augmented reality android application using an appropriate SDK

• Collecting and using sensor data gathered from a smartphone

• Basic wireless link budget analysis

From my preliminary research I concluded that the most challenging aspect of this project was going to be the work relating to augmented reality. In particular, accurately displaying the position and orientation of an antenna, or indeed any object, presented a non-trivial technical challenge which required the manipulation of mobile sensor, camera and GPS data. With this in mind, I conducted my research with an emphasis on the implementation of augmented reality apps and with an aim to uncover the methodologies behind developing a highly accurate and user-friendly augmented reality application.

<http://dbis.eprints.uni-ulm.de/1522/1/PRY_JUS_17.pdf>

## The AREA Framework for Location-Based Smart Mobile Augmented Reality Applications - Rüdiger Pryss\*, Philip Geiger, Marc Schickler, Johannes Schobel, Manfred Reichert

This is a follow-up paper to a paper published in December 2015 by the same authors. In it, the design and implementation of a mobile augmented reality kernel known as AREA v2 (Augmented Reality Engine Application version 2) is described in great detail, alongside the challenges faced whilst implementing such a kernel and the solutions which were found.

AREA v2 is a location-based augmented reality kernel, which, like any other augmented reality application, faces domain specific challenges, namely: overlapping points of interest, OS-specific sensor data processing and the reliable display of points of interest markers.

The problem of overlapping points of interest is a common theme amongst many research papers on the topic of location-based augmented reality. Many of these papers suggest their own novel solutions to this problem, and this paper is no different. In it, the authors suggest using a clustering algorithm which takes preferences from the user and then clusters points of interest, based on these preferences, into one point which is represented by a marker on the screen. This marker can then be used as a gateway to the POIs contained within it, thus enabling the user to interact with POIs that may previously have been hidden.

With regard to OS-specific sensor data processing with an aim to providing accurate and reliable POI markers, this paper notes that “regarding iOS, sensor data of the gyroscope as well as the accelerometer are used, whereas for Android sensor data of the gyroscope, accelerometer and compass of the mobile device are used to position the virtual 3D camera correctly.” Whilst these nuances are naturally handled by any augmented reality SDK and are not necessarily present in every augmented reality engine, it is important to note this difference, as performance of an application may change based on the operating system and set of circumstances at play.

Following on from this, the paper delves into issues involving “jittery” POI markers and possible sources of interference. The authors note “when using (a) the values of the gyroscope for a user that (b) frequently changes the position of his Android smart mobile device, the POIs on the screen of his smart mobile device oscillate badly. To obtain better user experience, we smooth the gyroscope values by using the SLERP algorithm [16] (cf. Algorithm 1, Line 28). Second, the rotation vector provided by the Android mobile OS is very precise on one hand, but it is prone to (1) frequent position changes, (2) slow position changes, and (3) magnetic interference sources on the other”. Again, some of the above may be handled by the chosen augmented reality SDK but it is nonetheless important to take note of this so as to enable swift identification of such issues throughout the development process.

In summary, this paper has given me a deeper understanding of the low level processes performed by augmented reality engines and the challenges faced therein.

# State of the Art Review

Many technologies have been used throughout this project. Aside from the main programming languages which were used to create the application (Java, Javascript and HTML 5), other common technologies such as jQuery, PHP and MySQL were used alongside a less well-known technology – Wikitude SDK. Please see below for a brief summary of each of these technologies alongside and explanation of how they were applied to create Align AR.

## Java

Java is a general purpose object-oriented programming language. It is commonly used across a range of applications and is also the native language for Android applications (REF: Java). For this project, I have used Java to retrieve and manipulate sensor and GPS data which is then passed to the Wikitude SDK (more on this below).

## Javascript

Javascript is a highly dynamic, interpreted programming language. It is mainly used in conjunction with HTML and is the language which I have used to manipulate the GUI for my application.

## HTML 5

HTML 5, or Hypertext MarkUp Language is the standard markup language used in webpages across the globe. In relation to my project, HTML was used to render the GUI, which was then manipulated using Javascript as mentioned above.

## jQuery

jQuery is a Javascript library which can be used to simplify the styling and animation of HTML pages. For my project, I used jQuery to create the basic style of the app and also to handle some of the animation.

## PHP

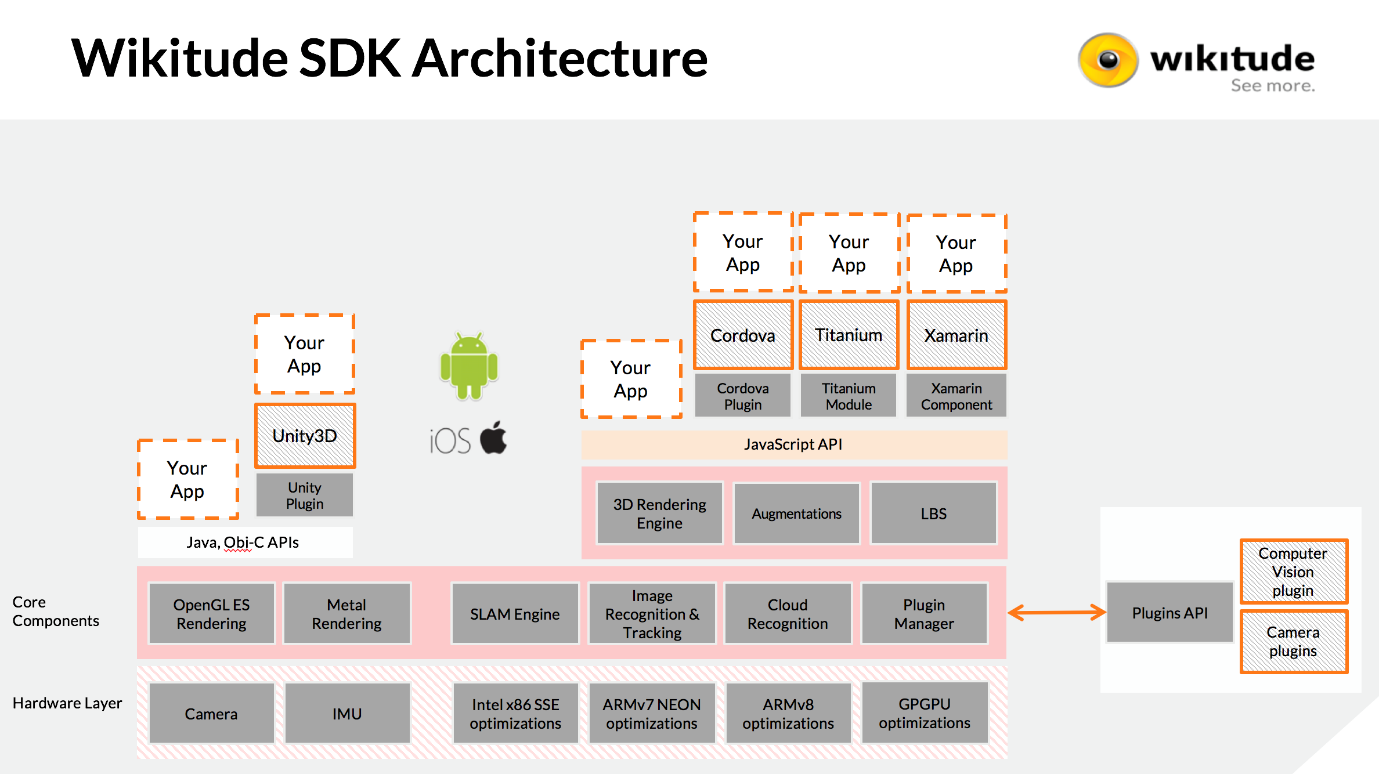
PHP is a server-side language which is mainly used for the manipulation of databases in conjunction with web applications. In this project, PHP was used to allow a user to add to, edit or delete from a MySQL database.

## MySQL

MySQL is a relational database management system which is open-source and widely used. For the purpose of this project I decided to use MySQL to store the locations and properties of each transmitting antenna.

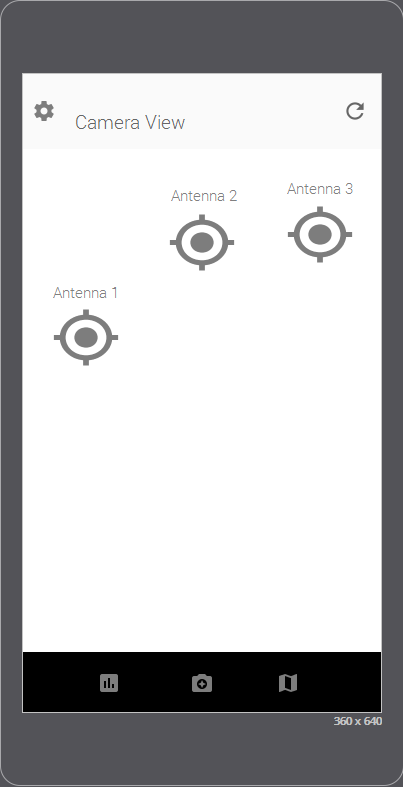
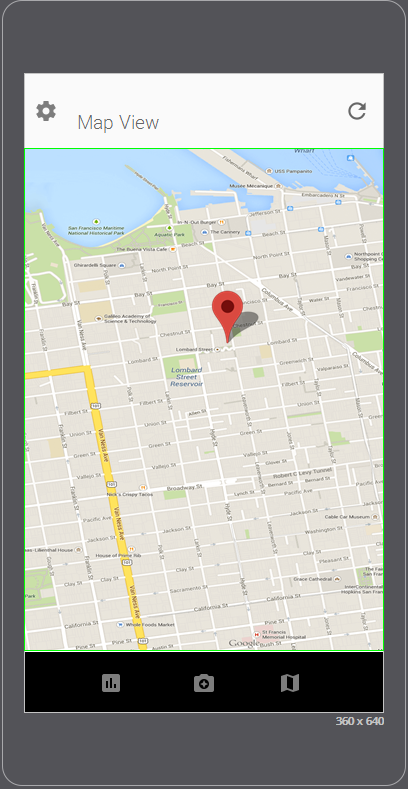
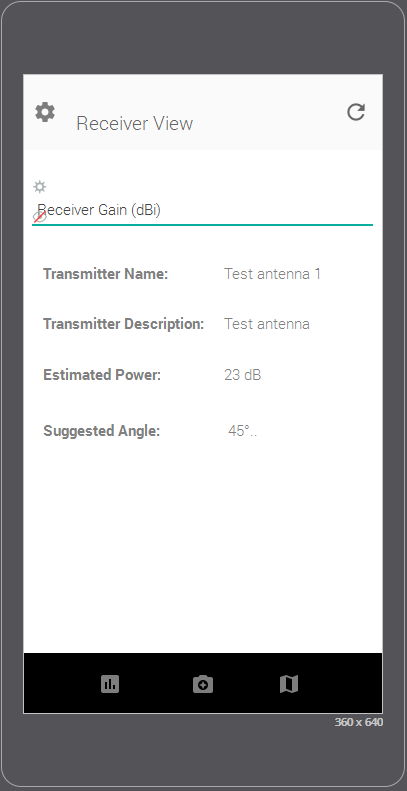
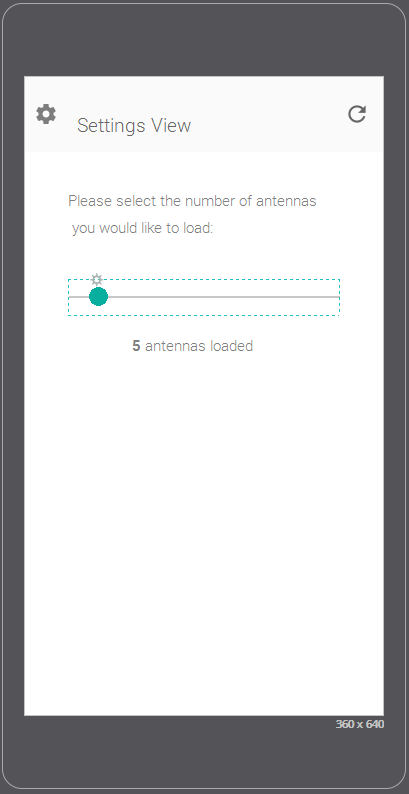
## Wikitude SDK

The Wikitude SDK is a cross-platform, mobile augmented-reality SDK which offers many different forms of augmented-reality such as geo-tagging, image-tracking and location-based. Wikitude is available as a plugin for many platforms including Unity and Cordova and its Javascript API can be used within Android Studio to develop augmented reality apps for Android. Alongside an abundance of features, Wikitude provides a vast amount of documentation which has proved invaluable throughout the implementation of this project.

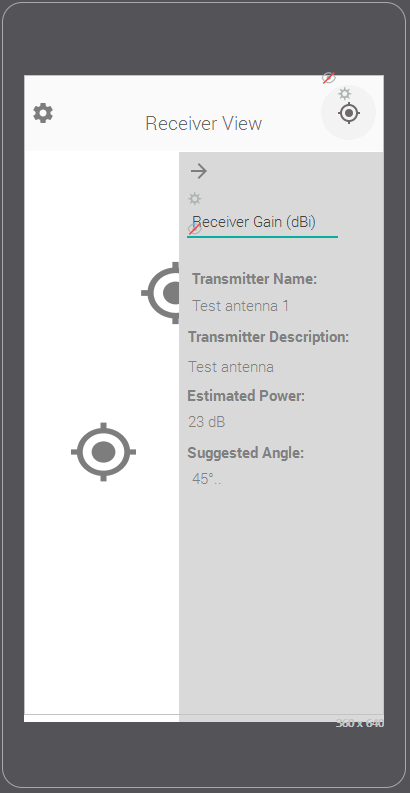
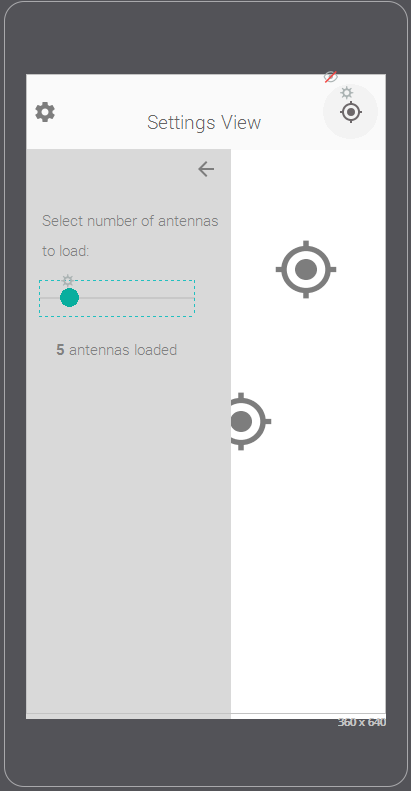


# Design

The design phase of any application can be challenging but it is particularly challenging in the realm of location-based augmented reality. This is because if there are many points of interest located around a user, then the screen can become over cluttered very quickly. Overcoming this challenge and ensuring a high level of usability were my main priorities during the design phase of this project. The following diagrams (fig. x, fig x1….) show my original design:

This design is simplistic and very user-friendly as it contains only 4 screens, each of which can be navigated to using one of the three buttons found along the action bar at the bottom and the settings icon located in the top right corner. To reload the antenna markers on the screen, the refresh symbol placed in the top right-hand corner of the screen can be pressed. This is all very intuitive which, of course, is the aim. However, after more consideration, I concluded that four completely different screens were unnecessary and that the above design could be condensed into one screen with multiple overlays partially, or fully covering the camera view, as needs be.

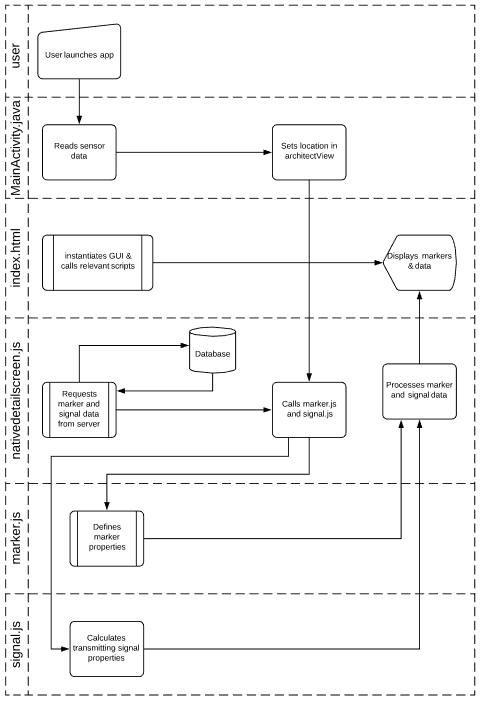
 

With this new design, users can input and see signal data simply by clicking on a desired marker, which will display an overlay where data can be input/viewed. Similarly, a user can click the gear icon to display an overlay where the number of antennas loaded to the screen can be adjusted. The map view remains much the same in this design, but is accessed through the location icon on the top right of the screen which then displays an overlay which covers the entire screen. In an effort to simplify the design, I have decided to remove the reload button and instead, let any click on the screen which is not on a button or marker cause a reload to happen.

# Implementation

Align AR is an android application whose core is written in Java which uses the Wikitude Javascript API to render augmented reality geo-objects to the screen.

In “MainActivity.java” the basis of the app is created, on which everything else is built. Within “MainActivity.java” an “architectView” is created. This architectView is part of the Wikitude Javascript API and is used to create a camera surface and to handle all sensor events (REF:<https://www.wikitude.com/external/doc/documentation/latest/android/setupguideandroid.html>). Using Google Play Services “fusedLocationProvider”, a user’s location is calculated and passed to the architectView as latitude, longitude and elevation (along with accuracy). Originally I used Android’s own location service to facilitate this functionality but after experiencing significant inaccuracies and very poor power usage statistics I decided to switch to Google’s fused location provider which has proved to be far more accurate and efficient. This improved performance is well documented (REF: Include play services vs android location reference). Apart from gathering sensor data, the Java code itself provides no functionality to the user. The entire front-end experience is built in Javascript and html, in what is commonly referred to as, the “ARchitect World”.



My ARchitect World is closely modelled off of the Wikitude example which can be found here (REF: INCLUDE LINK TO SAMPLE). In the example, the GUI is built in HTML and is manipulated using Javascript. Likewise, Align Ar is built using the same technique, but with a number of extra features as will be described further on.

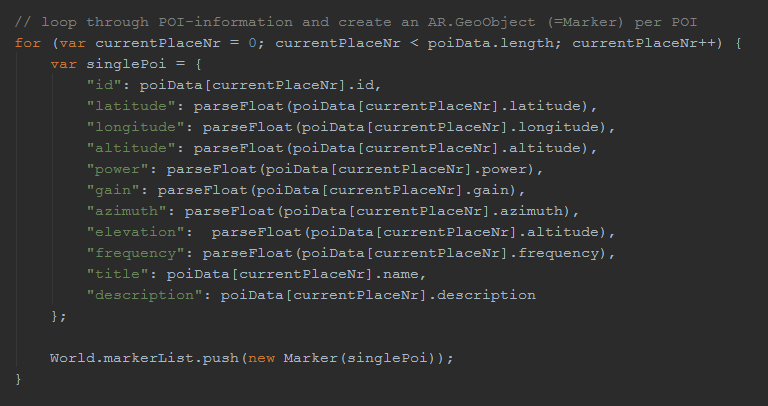
In “index.html”, a transparent html page is created over the camera view using jQuery’s “mobile-transparent-ui-overlay.css”. From this starting point, each Javascript file is loaded, a breakdown of the core functionality of each of these is as follows:

### Nativedetailscreen.js

Most of the front end display and user interaction is handled in “nativedetailscreen.js”, while specific marker, radar and signal properties/functionality are handled in their own scripts. In this script, a “world” variable is created which defines the AR world and the functions that control it. These functions directly influence the gui as seen by the user and include functionality to update the values of markers as users move between locations, change the amount of markers being displayed based on a user’s preference and handle the gui elements as panels are opened and closed. Some of the other functionality provided in “nativedetailscreen.js” is as follows:

#### loadPoisFromJsonData

As the name suggests, this function loads POIs (points of interest a.k.a antennas) from a json file stored on the danu6 server. Each POI contains information about a given antenna, including their latitude, longitude and elevation. Using these details, a new marker is created and added to the world’s marker list.



#### HideMarkers

This function is called when a user clicks on a marker and is used to hide all the other markers currently visible on the screen. In doing so, this prevents too much “clutter” from appearing on the screen and also helps to ensure that a user is not accidentally looking at the wrong marker.

#### InitMap & InitMapMarkers

These functions are used to initialize the Google Maps API and to place the antenna locations as markers on these maps.

### Marker.js

“marker.js” defines the properties and functionality of each individual marker. This includes the initial creation of the geo-object, the animations of the image drawables (including idle, active, green etc..) and the on-click triggers.

### Signal.js

The “signal.js” script is where all the necessary calculations are performed in order to calculate a number of properties for each signal being transmitted. A breakdown of the main functionality of signal.js is as follows:

#### Link Budget Analysis

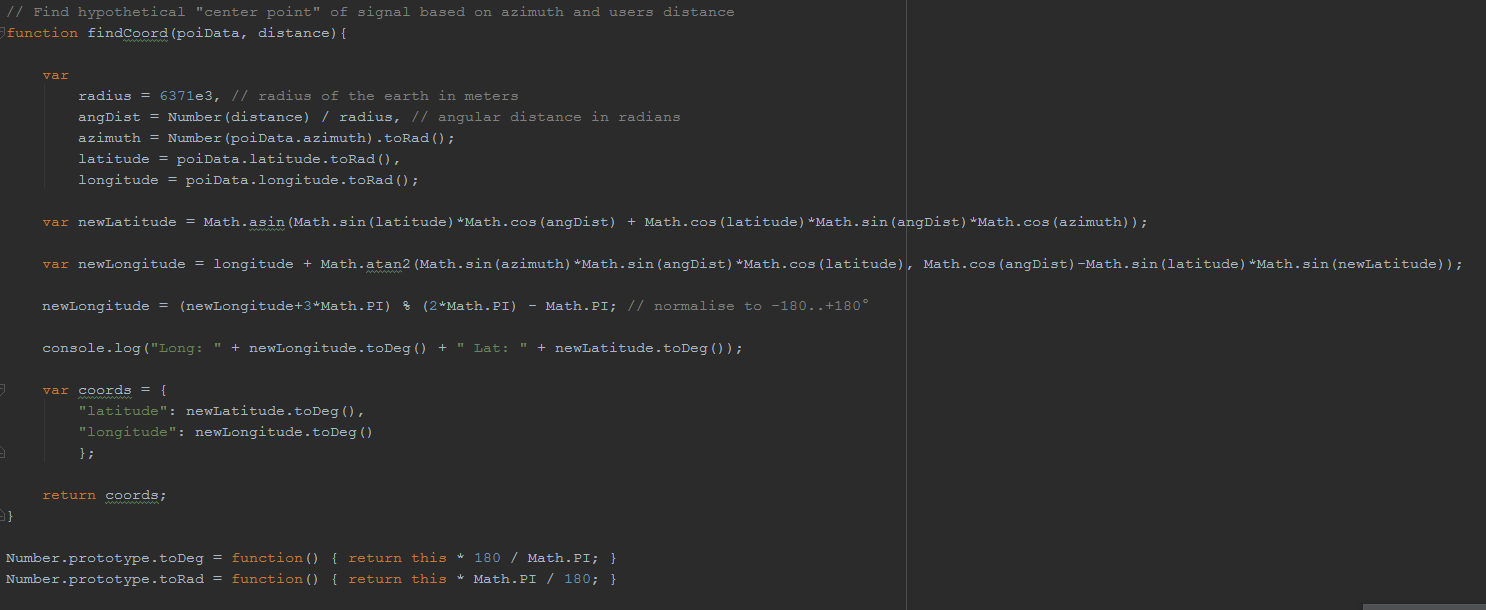
To calculate the link budget between the receiver and transmitter, signal.js reads the power (dBm), frequency (MHz) and gain (dBi) of the transmitter and perform a link budget analysis using the formulae described below (Formulae used section??).

#### Recommended Angle

Using the user’s elevation, the transmitter’s elevation and the distance between these two points, the recommended elevation angle (angle in the vertical plane) is calculated using simple trigonometry. Whilst this is most certainly an over-simplified solution, given that the aim is only to provide an estimated best angle, I believe this to be sufficient.

#### Sector Selection

In the event that a given mast has a number of sectors, all with different azimuths, it is important to identify the sector intended for an antenna at the users location. This is because a number of markers placed on the exact same spot on the screen provides obvious difficulties for users hoping to interact with these markers. To combat this problem, I have created a “sector filtering” function which is a function seen in nativedetailscreen.js and which uses a function called “findCoord” which can be found in signal.js. ”findCoord” first uses the location and azimuth of the transmitter, along with the distance from the user to calculate the co-ordinates of a point on a line originating from the transmitter, faced towards the azimuth, at the user’s distance. The sector filtering function found in nativedetailscreen.js finds these co-ordinates for each sector on a mast and then shows only the transmitter yielding the closest co-ordinates. Details of the algorithm used to calculate these co-ordinates can be seen below (formula/algorithm section).

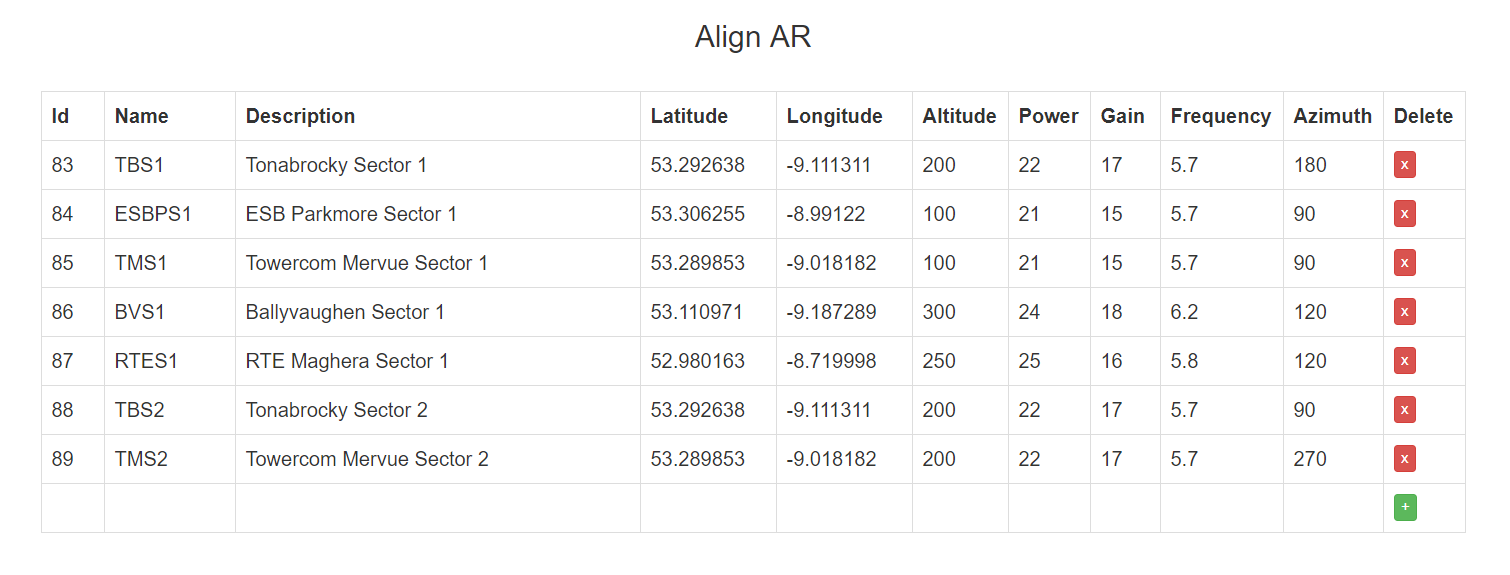


### Radar.js

“radar.js” is a simple script which handles the creation and animation of the radar which can be seen in the top left-hand side of the screen.

## Backend

In order to provide a useful service for ISPs I have added a simple GUI which can be used to input the location and properties of transmitting antennas. The page itself is written in HTML and Javascript and interacts with a MySQL database using PHP.



As can be seen in fig.xx above, this GUI consists of a simple “live table” which allows the user to view, edit and add antenna information conveniently.

Each interaction with the table is parsed into a MySQL query and sent over the wire to a database which is being hosted on one of the danu servers. In addition to this, each time the table is edited and the results sent to the database, a “select” query is used to retrieve the entire contents of the database which is then piped into a json file which is also stored on one of the danu servers. From here, the Android application is able to read the contents of the file and create the appropriate geo-objects as explained above.

In order to provide some level of authentication to this backend, I have added the Google login API, which allows a user to authenticate with their Google account credentials. This adds a convenient login method for a user and also allows me to keep track of who has logged in and at what time they have logged, thus enabling me to keep track of the changes a particular user has made.

\*\*\* PICTURE OF LOGIN WHEN IT LOOKS BETTER\*\*\*\*

# Evaluation

Whilst evaluating this project, I have reviewed the requirements which I and my supervisor agreed upon, and as are stated in my project definition document. The following is my evaluation of Align AR in regards to these requirements:

#### 1. The application should be compatible with all android devices where a recent version of android is installed

Align AR has been built with Android SDK version 15 (Android 4.0.3, 4.0.4) as its minimum compatible version and has been extensively tested with Android SDK version 26 (Android 8.0). The app has been tested on several different versions of Android ranging from Android 5.0 up to the new Android 8.1 and performs well on all. The app has also been tested on a range of different devices including both new and old Android phones and tables and has not experienced any major issues with performance or with the GUI.

#### 2. The application should have read-only access to a database containing the following information:

#### a. The GPS co-ordinates of relevant microwave broadband transmitters

#### b. The effective isotropic radiated power of these transmitters

This requirement makes up a key part of the functionality of Align AR. As has been described in the implementation section above, in order to draw each marker on the screen and calculate the necessary link budget, a json file is read from a secure server, which in turn, gathers its information from a MySQL database. The access which the app has to this json file is strictly read-only and as the app only interacts with the json file and not the database itself a possible security hole is avoided.

#### 3. The application should be able to use this data to calculate the necessary link budget

This requirement has also been completed and somewhat extended as the app also offers the user the opportunity to input not just the receiving antenna gain value, but also the connection loss and cable loss if they so wish. This offers a higher level of accuracy for the user.

#### 4. Using the device’s camera, the application should display markers on the screen indicating the location of the transmitting antennas

Again, this requirement makes up one of the core functionalities of the app and is also a requirement which I have expanded on, as will be explained further below.

#### 5. To help with the alignment of the antenna, the application should provide visual and/or audio cues to the user indicating the best vertical and horizontal alignment and the expected signal strength when aligned

The markers on the screen indicate the position of each transmitting antenna in both the horizontal and vertical axis, this enables the user to align the antenna visually before viewing the antenna details overlay which tells the user more about the transmitting antenna’s exact position and suggests a suitable elevation angle for the receiving antenna

## Additional Requirements

During the course or this project, my supervisor and I agreed upon a number of auxiliary requirements which are not set out in the project definition document. My evaluation of the project deliverable’s implementation of these requirements is as follows:

## 

#### A backend should be provided so that in a real-world scenario, ISPs can easily add, edit and delete antenna details on the fly.

Originally, the location and properties of each antenna was stored locally on the user’s device. This was convenient for development purposes but did not make the app very user friendly as to edit the details of any antenna involved manually editing a json file on a user’s device. To combat this, I have developed a backend (described above in implementation section) which authenticates a user and allows him/her to easily manage their database of antenna locations and properties. I’m very pleased with how this turned out as I believe it be extremely intuitive and user-friendly.

#### The application should be able to filter out sectors of a given mast so that only the appropriate antennas are displayed, taking into consideration the azimuth of each antenna and the user’s location.

This feature is important when taking into account how the app may be used in a real-world scenario. Typically, an ISP will have a number of antennas located on the one mast. This is because each of the antennas must send a signal which will travel a considerable distance and thus, must be highly directional. (REF: Reference about highly directional antennas). This presents a problem when attempting to display markers for each of these antennas as the markers will appear on top of each other. Also, It is also pointless to display information about an antenna which is not facing our user. It is because of these reasons that my supervisor and I decided to include a sector filter feature. This feature has been tested as described below (in testing section) and I am happy with its performance although I do feel that as a possible improvement, the algorithm used to filter out unwanted antennas could be improved. Currently, if the centre of the beam area of an antenna is closer to the user than the centre of the beam area of any other antenna on the same mast, then only that antenna’s marker will be displayed. (REF: beam area definition).This is sufficient for filtering out any unwanted antennas and does not pose much of a problem as an ISP technician will presumable know whether or not a transmitting antenna is pointing in the right direction before attempting to install a receiver. However, an improvement on this would be to perform the appropriate calculations in order to model the transmitting signal’s azimuth and elevation pattern and thus determine whether or not it is possible for the user to receive a signal from the chosen antenna.

#### Functionality should be provided to filter the number of markers displayed on the screen based on the distance between the user and these markers

A common problem with geo-location augmented reality applications is the over-cluttering of markers on the screen. (REF: some reference about augmented reality cluttering). To avoid this problem, a common solution is to filter out unwanted markers using some property. In Align AR, I have provided a simple slider to the GUI which allows a user to select the number of markers they would like displayed on the screen. As the user moves the slider to the left, markers which are the greatest distance away are removed, thus leaving it easier for a user to view and select appropriate markers. As an ISP may have several antennas all over the country, I believe this is a very useful feature which, whilst relatively simple to implement, greatly improves a user’s experience within the app.

#### A “map view” should be added so that the location of each antenna can be easily viewed by a user

This feature is more of a “nice-to-have” rather than a necessity, however, it does allow the user to view both a map and satellite view of their surrounding areas which, in turn, allows them to verify the location of each antenna to ensure that the markers, possibly inputted to the system by someone else, represent the true locations of the antennas.

# Conclusion

# Works Cited

**There are no sources in the current document.**