

Antenna Alignment Augmented Reality Video Application

Final Year Project

Chris cornwall -12311016

15/04/2018

Contents

[Introduction 2](#_Toc510113889)

[Alignment of Highly Directional Microwave Antennas 2](#_Toc510113890)

[Augmented Reality 2](#_Toc510113891)

[Literature Review 2](#_Toc510113892)

[Description 3](#_Toc510113893)

[State of the Art Review 3](#_Toc510113894)

[Design 3](#_Toc510113895)

[Implementation 4](#_Toc510113896)

[Nativedetailscreen.js 5](#_Toc510113897)

[Marker.js 6](#_Toc510113898)

[Signal.js 6](#_Toc510113899)

[Radar.js 7](#_Toc510113900)

[Evaluation 8](#_Toc510113901)

[Conclusion 8](#_Toc510113902)

[Works Cited 8](#_Toc510113903)

# 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.”

# Literature Review

With this as the backdrop to my project, I have conducted substantial research on the topics of directional antenna alignment, augmented reality and Android development. Please see below for a synopsis of this research.

# 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.”

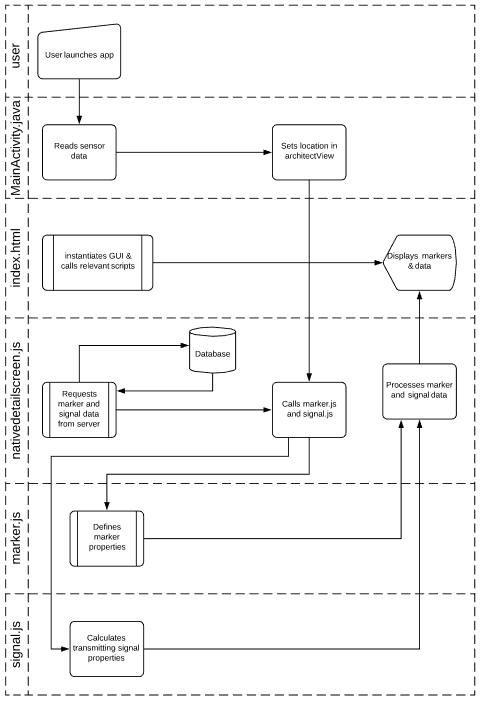
# State of the Art Review

# Design

# 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.

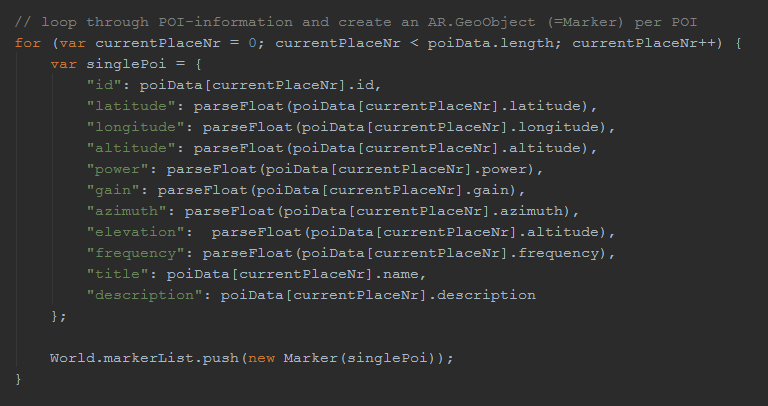
In “index.html”, a transparent html page is created over the camera view using jQuery’s “mobile-transparent-ui-overlay.css”. From this starting point, each Javascript file is loaded, a breakdown of the core functionality of each of these is as follows:

### 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

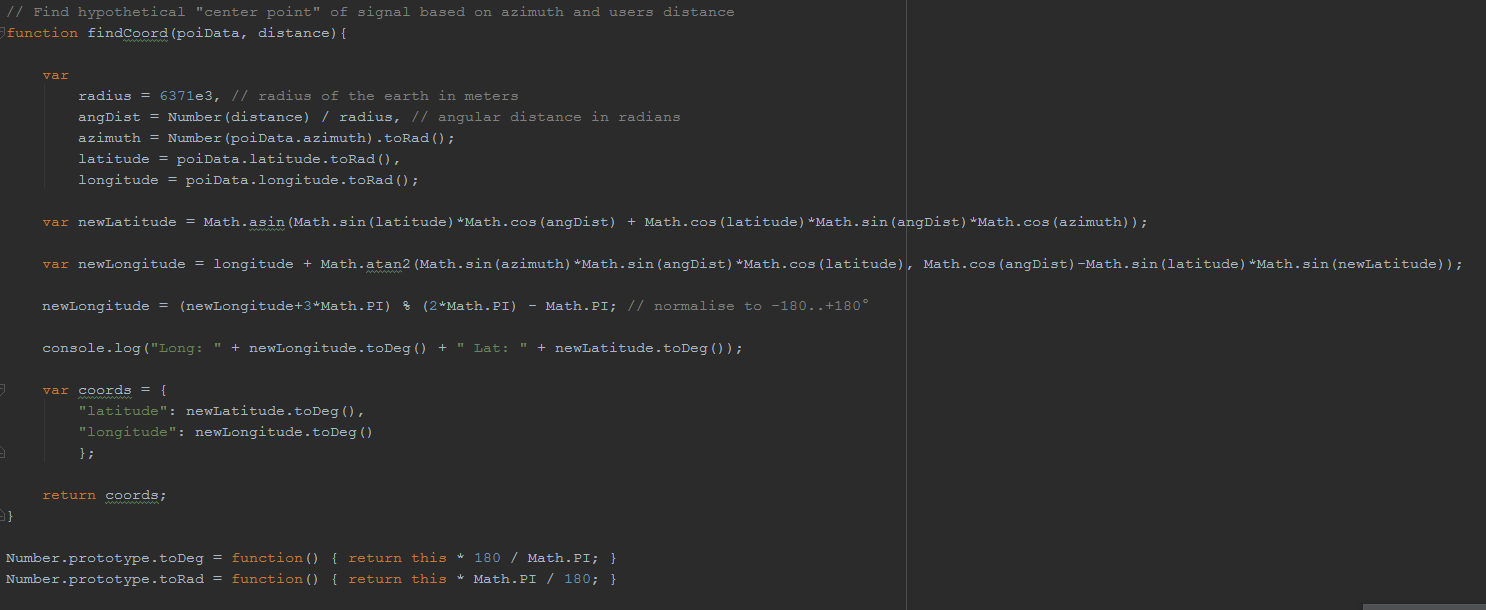
To calculate the link budget between the receiver and transmitter, signal.js reads the power (dBm), frequency (MHz) and gain (dBi) of the transmitter and perform a link budget analysis using the formulae described below (Formulae used section??).

#### 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.

# Evaluation

# Conclusion

# Works Cited

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