Determining Optimal Antenna Configuration using RF landscape Modeling

ECEN 489 Final Project Report

By

Sam Carey
Mark Hardiman
Peng Li
Alexandre Lopes
Cameron Shaw

1. Introduction

The goal of this project was to provide maximum wireless signal strength for any location in a finite region by manipulating a directional antenna. To accomplish this, a cantenna was rotated in the horizontal plane by a stepper motor, controlled by a driver board, controlled by an Arduino microcontroller, controlled by a laptop. An Android phone relayed signal strength and GPS coordinates to a server running on the laptop via Wi-Fi. The server would receive this information, and, knowing its own GPS location, calculate the angle between itself and the user. The server would then send this angle to the microcontroller which would then rotate the antenna to point directly toward the user, thus maximizing signal strength.

2. Development

2.1 Antenna Characterization

In order to inform decisions regarding "cantenna" angle, the field characterization was measured in the first step of the project. Raw data consisted of RSSI values measured with a directional antenna at a constant radius at five degree intervals relative to the center cantenna. After two full circles, the data was processed via averaging of same-angle RSSI values as well as by applying a smoothing filter in order to avoid irregularities in the RSSI values. To acquire greater accuracy, the symmetry of the cantenna was used in order to essentially double the number of sampled data points per angle, e.g., the value measured at 45 degrees should be the same as the value at -45 degrees. Figure # shows the final characterization of the cantenna with normalized RSSI values to eliminate the dependency on the radius of the measurement circle.

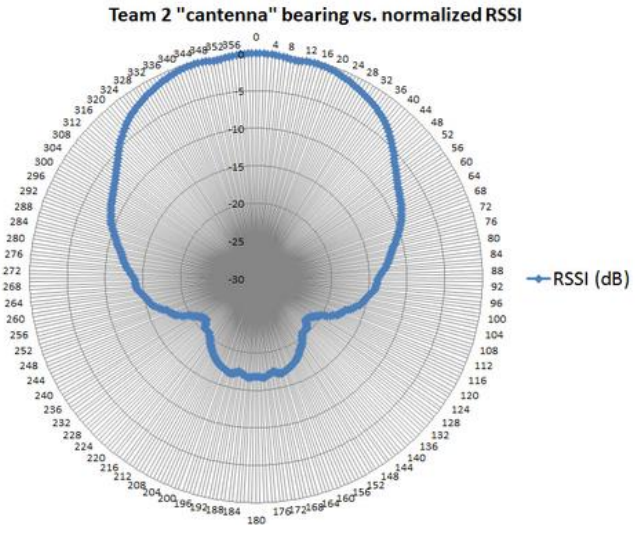

Figure 1: "Cantenna" radiation pattern.

2.2 Android Application

For this project the Android Application was used to get RSSI values from the Wi-Fi Network and GPS Location, store them into a SQLite database and send these information to a server. It was adapted from the first stage of the project to the last one, because for all parts of the project it was necessary to have RSSI values of a network.

First, it was defined that the user should be able to select the network he wanted to be connected to get the RSSI values of this network, so there is a space in the application where the user writes the name of the network he wants to connect. After clicking the "connect" button, the cellphone will be connected to this Wi-Fi network. It also has a space to check which networks were available on the moment the program started. After choosing the network, there

is a button that the user can press to acquire RSSI value from the selected network. Every time this button is pressed, the program also gets the GPS location of the cellphone and store both and send to the server. The connection to the server is made using datagram sockets to establish connection with the server. Therefore, for this first phase of the project, it was necessary to press the button to send the information to the server and to acquire data.

For the second phase of the project, some changes were made to the server. For purpose of testing the motor of the cantenna, the application needed to send the information to the server every X seconds. Therefore, the code was adapted for this situation and two extra buttons were added to the project. The first one was a pausing button. The user was able to pause the acquiring of RSSI values, location and connection and after this, he/she was able to continue every time he/she wanted. The second button, was used again for testing. Instead of sending the RSSI value and location, it was going to send a specific number that could be interpreted in the server in order to test the motor of the cantenna. For example, it was assigned that this button was going to send 1 as RSSI value, 1 as latitude and 1 as longitude, so, if the server receives this information, it can treat this information as a special case to do some special thing.

Finally, the automatic sending and acquiring information was removed from the application so that the user could define when to get RSSI values. It was also created a SQLite database in the cell phone to store the RSSI and location of all the acquired values. This database stores the information until a button is clicked. If the "delete" button is clicked, it will erase the database.

2.3 Server

The project utilizes a central Java server in order to coordinate information between the Android application and the hardware/motor setup, as well as to record data in an SQL database. The server runs on a laptop connected to the same network as the Android phone, and this laptop is connected via USB to a Teensy microcontroller in order to interface with the hardware. A serial communication protocol is set up to send command to the microcontroller from the server.

2.3.1 SQL database

Upon receiving RSSI and GPS data from the phone in a UDP datagram, the server calculates the required angle for the cantenna based on its GPS position, and passes this angle to the microcontroller. Additionally, the server makes an entry into a SQL table, which has columns for latitude, longitude, RSSI, and angle.

Furthermore, the server has a built-in protocol to follow in case no packet has been received recently, indicating the phone has lost connection to the network due to a bad cantenna angle. Once 20 seconds have elapsed with no packet received, the server begins to coordinate a scan in which the motor moves the cantenna in increments of 30 degrees until a packet is received again.

2.3.2 Real-time tracking system

A real time tracking system is implemented in our server. As the user walking, the cantenna is pointing to the user in real time.

To implement the tracking system, we choose true east as the 0 degree in the system coordinate. As we receive location data packet from the application, the server calculates the angle between the user direction and the true east. The angle will be sent to the microcontroller through the UART.

2.4 Hardware

The basic function for the microcontroller and step-motor system is to rotate the cantenna to the desired direction.

The first challenge in the system is the communication between Teensy microcontroller and the step-motor driver. We tried to use UART control port on the step-motor driver. However, no feedback was received as we sent signal to it. Then we tried the signal control panel and find out it is robust. There are three ports in the signal control port – EN, DIR and PULSE. EN controls the other two ports enable or disable; DIR controls direction set to clockwise or counterclockwise; PULSE controls the step of the rotating. Each pulse (from low to high) sent to PULSE will generate 0.18-degree rotation.

The second challenge is to set up a correct coordinate system. In our design, a defined geographical direction is selected as 0 degree in the microcontroller coordinate. For the clockwise semicircle, the angle decreases to -180 degree uniformly; for the counterclockwise, the angle increases to 180 degree uniformly. Note a real geographical direction need to be chosen in the server, and the cantenna need to be collated during setup of the system.

Finally our microcontroller and step-motor system works as we expected. We thank for the help from the other groups in building up the whole system.

2.4.1 3D-Printing

The cantenna is mounted onto the vertical shaft of the motor using 3D-printed parts: two clamps previously designed by Jake McKnight and an additional piece that joins the two together. This joint piece has a hole for the motor shaft that can be tightened onto the shaft with a setscrew. The 3D-printer was also used to create a base for the motor that reduced the risk of tipping during operation.

2.5 Processing and Algorithms

Given a list of latitude and longitude coordinates of an area surrounding a Wi-Fi antenna, one challenge was to predict the RSSI values that could be found with a receiver at those coordinates. These predictions would be based on another set of RSSI and GPS data gathered in the area. A multistage algorithm interpolated this data for an m by n grid containing the points to be predicted:

After being converted from spherical coordinates to Cartesian coordinates, the data points are used to reconstruct a surface continuous in X and Y, with Z representing RSSI. First, the points are fed into a Delaunay Triangulation function from a library, which forms a linear interpolation by drawing triangles between all the points. This Delaunay method chooses triangles in the XY plane such that no points lie within the circumcircle of another triangle, which generally avoids skinny triangles and results in smoother gradients.

This irregularly shaped, interpolated surface does not encompass the whole grid, so an extrapolation stage was implemented using successive relaxation. This works by sweeping through the grid, assigning to each point the average value of the points immediately surrounding it. Over the course of many sweeps, the interpolation surface is held constant while the values along its edge progressively "bleed" outward to the surrounding, variable points. Eventually, the outlying region converges to a smooth continuation of the inside interpolation.

Successive relaxation is employed once again to smooth the surface. This time, all points are left variable and averaged to surrounding points. Each sweep has the overall effect of flattening the surface, so the maximum and minimum RSSI values are recorded before each sweep and are used afterward to rescale the range of RSSI values. This preserves the general shape of the surface.

Lastly, the interpolated grid is converted back to spherical coordinates and used to assign RSSI values to the requested coordinates.

Algorithm Summary:

- Read RSSI/GPS data from database
- Convert GPS coordinates to a local Cartesian space
- Interpolate using triangulation
- Extrapolate borders using successive relaxation with interpolation held constant
- Loop: Smooth all, using successive relaxation, and restore scale
- Convert interpolation to Spherical coordinates
- Map requested GPS coordinates to RSSI interpolation

3. Results

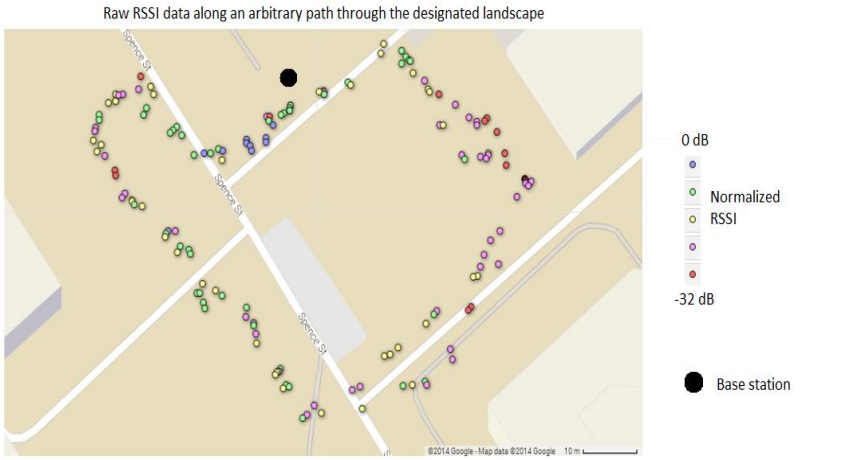

Figure 2: Color-coded raw RSSI data from full-system test.

Analysis of early field data indicates that no significant reflections exist in the target landscape that could be used to improve upon the basic line-of-sight scheme. To come to this conclusion, data points (GPS and RSSI) were taken through areas with no direct line-of-sight to the base station position, such as the northwest corner of the target area. For each location in this sub-areas, RSSI was recorded for five angles relative to the user's position: 0, +30, -30, +60, and -60 degrees, where 0 indicates the direct line-of-sight cantenna angle.

A final set of raw data was recorded using the direct line-of-sight scheme decided upon. Figure # above shows the raw RSSI data collected using the Android application while walking along a pre-designated route through the target landscape. Note the unexpectedly low RSSI values close to the base station position. This is likely due to inaccuracy in GPS reading given by the phone, resulting in incorrect angle calculations. Thus, the cantenna angle was not properly suited to the user's location, resulting in poor RSSI values while the cantenna pointed in the wrong direction.

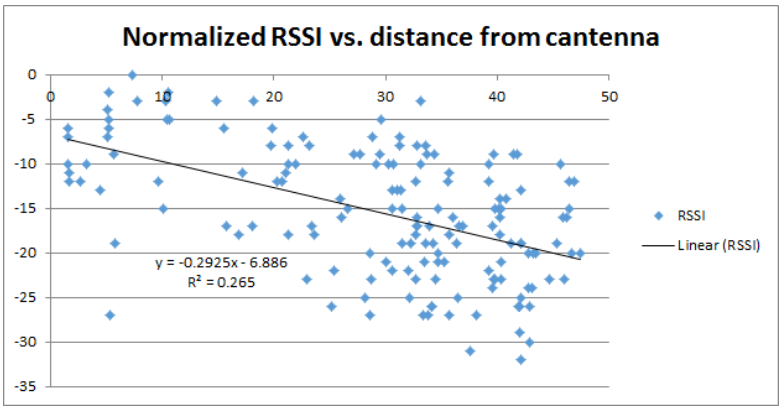

Figure 3: Best-fit line of RSSI vs. distance from base station to obtain model coefficients.

An attempt was made to use the final raw data set to tune a Friis transmission model in order to predict RSSI values along the path at any arbitrary point. The Friis transmission equation is given as

$$\frac{P_r}{P_t} \propto G_t G_r \left(\frac{\lambda}{R}\right)^n$$

for cluttered environments in which direct line-of-sight is not guaranteed. The trend line in Figure # is equivalent to the logarithmic representation of the above equation. However, n was expected to be greater than 2, which is the baseline for radiation falloff with distance. In the logarithmic equation, n is equal to the negative of the slope of the line, or 0.2925. This distance falls well below realistic values. This inaccuracy is likely due to the aforementioned problems with the close range data having artificially low values, which resulted in a much more gradual slope in the trend line. Due to these issues, no approximations of RSSI values were made using this model. In the future, the model should be tuned first in more controlled conditions rather than relying on data taken with the complexity of the full system.