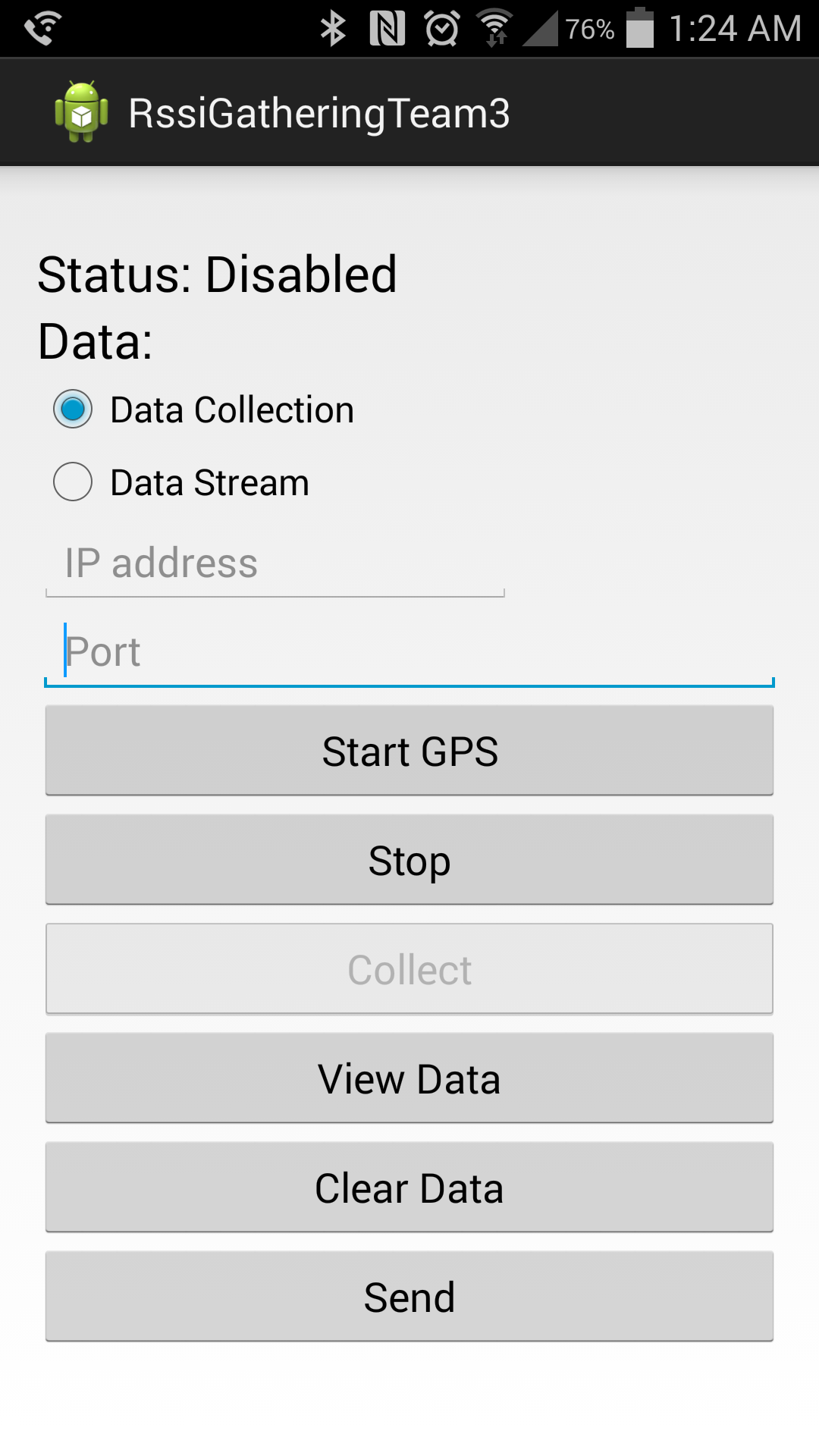
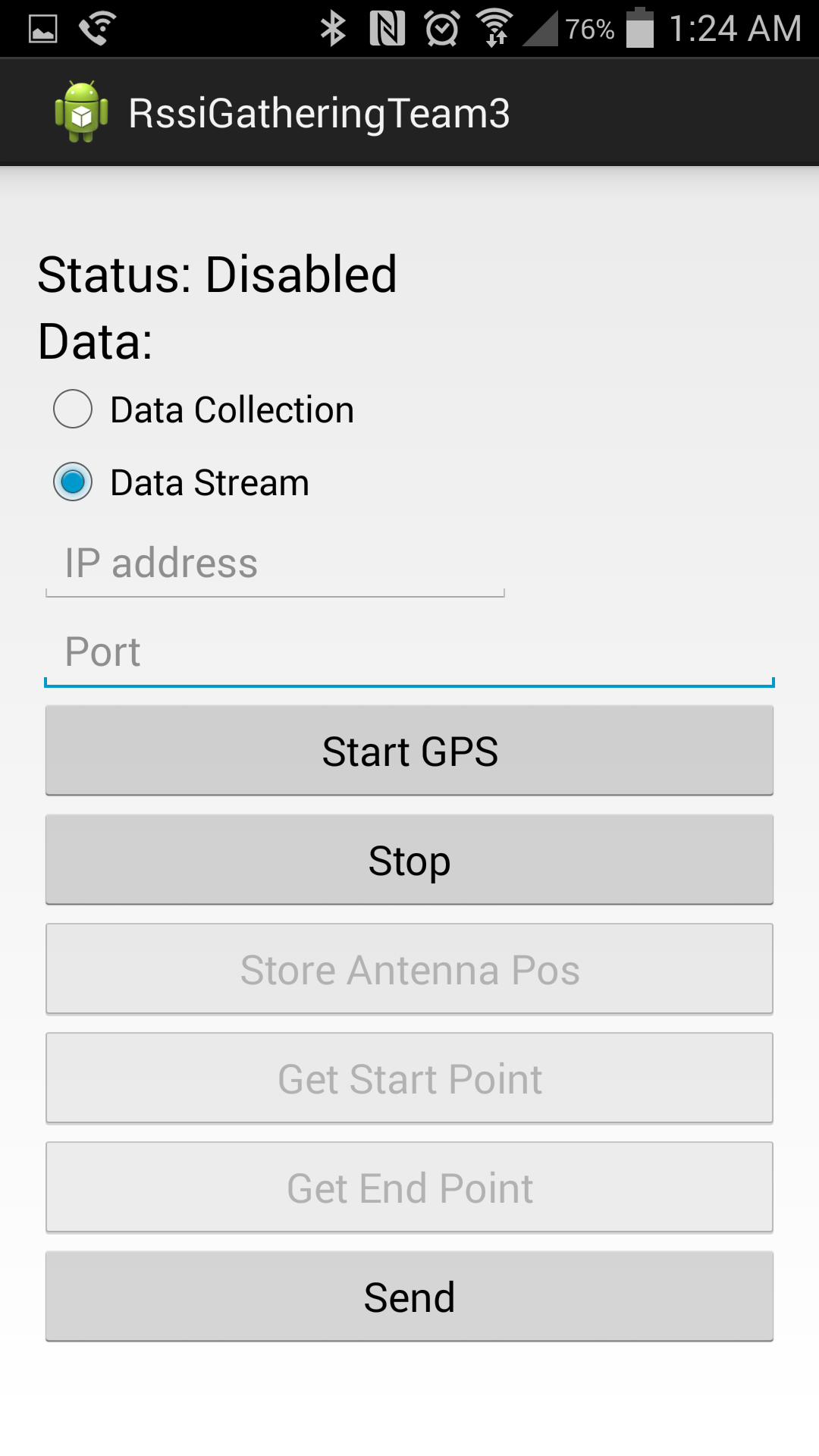
**Intro**

The goal of the project was to use knowledge built throughout the course through other projects in constructing a practical engineering system. This system would use our android device to direct an antenna that sends a wireless signal that we then measure on the same device. From here the app on the phone was designed to push to a java server which in turn pushes to a SQLite database. This server was also able to take the data from the phone and push to a Google Fusion Table for visualization of the database. Lastly, an algorithm was written to help with characterization of our antenna data. For this project clear roles were defined for each team member in order to complete the project: Miguel Alvarez (App development), Yuanxing Yin (Server development), Joao Aguiar (Motor hardware), Joseph Binder (Algorithm development), and Colin Brasher (Leader). In this report we will explain how the system was built as well as include applicable pseudocode to help show how the system was developed and implemented.

**Application**

In this project the application fulfills two roles: gathering data and streaming data.

The gathering portion is used to gather and store data before attempting to track the phone. This data is used to characterize the area in which the phone will be tracked afterwards. The points consist of gps coordinates (latitude and longitude) along with an rssi value.

This mode is represented by the first image. To gather data, the user must first press the Start GPS button. This will activate the gps and start looking for its position. Once a gps lock is acquired, the collect button will be enabled and the phone will update a latitude and longitude variable every second. If the collect button is pressed, the current rssi will be retrieved and it will be stored along with the latest latitude and longitude points.

The points for this mode are stored in a sqlite database on the phone and can also be sent to a server by inputting the servers IP address and port number. The rssi values in the sqlite database can be previewed on the phone using the “View Data” button.

The streaming portion of the app is used for tracking. The data points in this mode consist of antenna position , current position, and rssi. The rssi however is not used for the actual tracking but simply serves as a measurement tool to evaluate the effectiveness of the antenna.

Like the first mode, the “Start GPS” button activates the gps and waits to get a gps lock. Once it does, it will enable the “Store Antenna Pos”, “Get Start Point”, and “Get End Point” buttons. The phone will also begin updating a latitude and longitude variable every second. To begin tracking, the phone must be held right over the antenna position and the “Store Antenna Position” button is pressed. This will take the latest latitude and longitude points and store them to antenna position variables. Once these values are stored, the app will begin updating another set of variables for current position every three seconds. Every time these points are updated, an array of 5 doubles is sent to the server which. The data in the array is as follows [antenna latitude, antenna longitude, current latitude, current longitude, rssi].

With this data, the server can now calculate the angle of the phone relative to true north and adjust its own orientation to point at it.

The “Get Start Point” and “Get End Point” buttons were previously used for manual point sending but are no longer used since the app sends points automatically every three seconds.

The “Stop” button in each of these modes will halt location updates and disable the gps as well as the buttons requiring location data to prevent gathering of bad data.

**Server:**

1. Fusion Table: We import all jar packages needed and correctly configure credential json file. We create a class called “uploadfusiontable”. There are two methods in this class: one is createTable(para...), which is for creating a new fusion table in google drive and then inserting data, and another one is update(para…),which is for inserting data into an existing fusion table in google docs.

2. Server structure:

Besides the main class, there are 4 other classes contained in the server. Algorithm.class is used to implement algorithm on the collected data to reconstruct RSSI map. CreateTable.class is used to create a new table in local SQLite database. TestClass.class is a class for testing the motors ability to spin the cantenna. Uploadfusiontable.class is used to programmatically upload our data to Google Fusion Table. General functions of our server in pseudocode:

BEGIN

arg ← keyboard input

IF (arg equals to “1”) THEN

Server retrieves data from local database, runs the algorithm and upload results to fusion table

ELSE IF( arg equals to “2”) THEN

Server starts “Data Stream Mode”, receiving data from App every 3 seconds, stores them in local database and drives the cantenna to point to phone location.

ELSE IF(arg equals to “3”) THEN

Server starts “Data Collection Mode”, receiving data that is manually collected by the android app. It then stores them in local database.

ELSE IF(arg equals to “4”) THEN

Server will upload the data collected by Data Stream Mode to fusion table.

ELSE IF(arg equals to “5”) THEN

Server stops

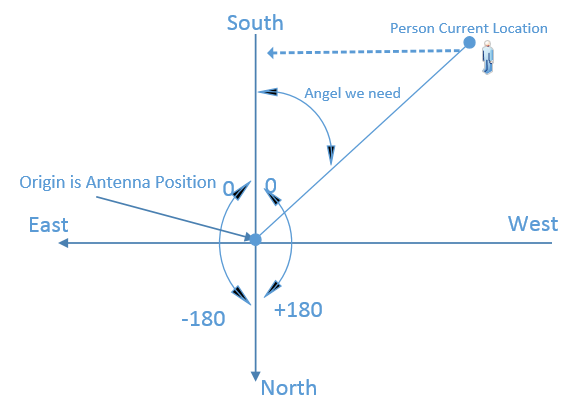
ELSE

Invalid input, please re-type.

ENDIF

END

3. Cantenna Tracking

At the beginning of test, we always set our Cantenna pointing to true South. Then use trigonometric functions to calculate the angle we need.

pseudocode:

BEGIN

latCurrent ← Current point’s latitude

lngCurrent ← Current point’s longitude

latAntenna ← Antenna’s latitude

lngAntenna ← Antenna’s longitude

nume ← (latCurrent – latAntenna)^2

deno ← (latCurrent – latAntenna)^2 +(lngCurrent – lngAntenna)^2

basicAngel ← arcsin(sqrt(nume/deno))

IF ((latCurrent <= latAntenna) and (lngCurrent >= lngAntenna)) THEN

Result← -basicAngel

ELSE IF ((latCurrent <= latAntenna) and (lngCurrent <= lngAntenna)) THEN

Result← basicAngel

ELSE IF ((latCurrent >= latAntenna) and (lngCurrent >= lngAntenna)) THEN

Result ← -180 + basicAngel

ELSE

Result ← 180 - basicAngel

END IF

END

**Algorithm**

Upon reaching step two and three of the project, it was necessary to develop a means to read between the lines with the data recorded and what it implies. In order to see the larger picture, an algorithm needs to be able to expand the data to encompass the full area. The first design plan involved linear interpolation, but this design plan failed when it came to the edges of the designated area. In order to account for this, the design was modified to estimate the locations of nearby AP’s and gradually lose signal intensity at a rate based on the other points given.

This method, while not quite as accurate as to exact locations of coverage drop between points has far more precision at locations near AP’s where linear interpolation doesn’t go below the lowest value, and also has the capacity to deliver values outside of the field of interpolation. For these reasons, this algorithm was designed to minimize the error of the points, but moreover far more accurately fill in the surrounding areas. Now in a real environment, the measurements can be affected by other nearby signals and obstacles. The algorithm designed is not the best at picking up the locations but instead has the tradeoff of being able to recognize additional AP’s so it can gain additional accuracy with scenarios that involve picking up another access point’s signal.

For the purposes of part two and displaying an accurate portrayal of an area, the algorithm performs moderately with a normalized error of ~ 5 RSSI, considering it’s lacking abilities in locating dropping signal regions, this error is fairly reasonable. The benefits to this algorithm type though are seen in part three of the project. The algorithm was tasked with evaluating the RSSI against what would be seen as an ideal signal. The designed algorithm very accurately portrays an ideal signal. Therefore after updating the algorithm to quantize it’s output to the input data points, the algorithm creates a reasonable ideal data set. This comparison can be seen in Figures 1 and 2. Figure 1 portrays a dataset taken, and Figure 2 is the ideal data set computed by the algorithm. As shown below, the walked dataset is far less smooth, as obstacles prevent an ideal reading. This is most visible in the southern aspects of the figures as there were the most obstacles located there. It is also worth noting that the larger groups of red are also prominent in locations with more points taken. This is unavoidable since in any real test run, the points cannot be perfectly spaced out, even with the appropriate timing, things like traffic and signal loss prevent an ideal dataset for the fusion tables to create.

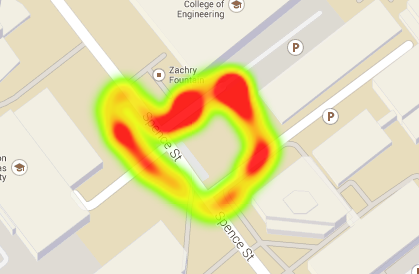
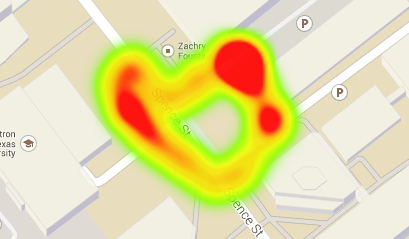
 

Figure 1: Original data Figure 2: Ideal dataset computed by algorithm

**Motor**

For this project a step motor in the bipolar configuration, the SD02B as the motor driver and the teensy board to do the control of the driver were used. The SD02B board generated the set of pulses needed to activate the motor and the control signal that indicated that an increment in the steps should be performed was done through PWM.

To generate the PWM the PWM pin from the teensy board was used; which allowed the configuration of frequency and the duty cycle. After trying some different frequency and duty cycle, the best combination found was frequency of 100Hz and duty cycle of 7.85%. To control when the motor should turn, the enable pin of the SD02B was set high during the time needed to generate the number of steps that the motor should turn.

As the cantenna would have the a cord attached, the teensy should not allow that angles greater than 360º be entered. To implement this, it was defined that the angles would be in the range of -180 to 180. This value means the position to which the motor should turn, not how much it should turn. The teensy’s job is to keep track of the position and calculate first the difference between the input angle and the position of the motor. Second the teensy calculates how many steps the motor should move and the direction of the steps. To prevent precision lost encountered in the first tests, a sensibility of 5 degrees was used. This was so the motor would not turn if the difference between angles were less than 5º.

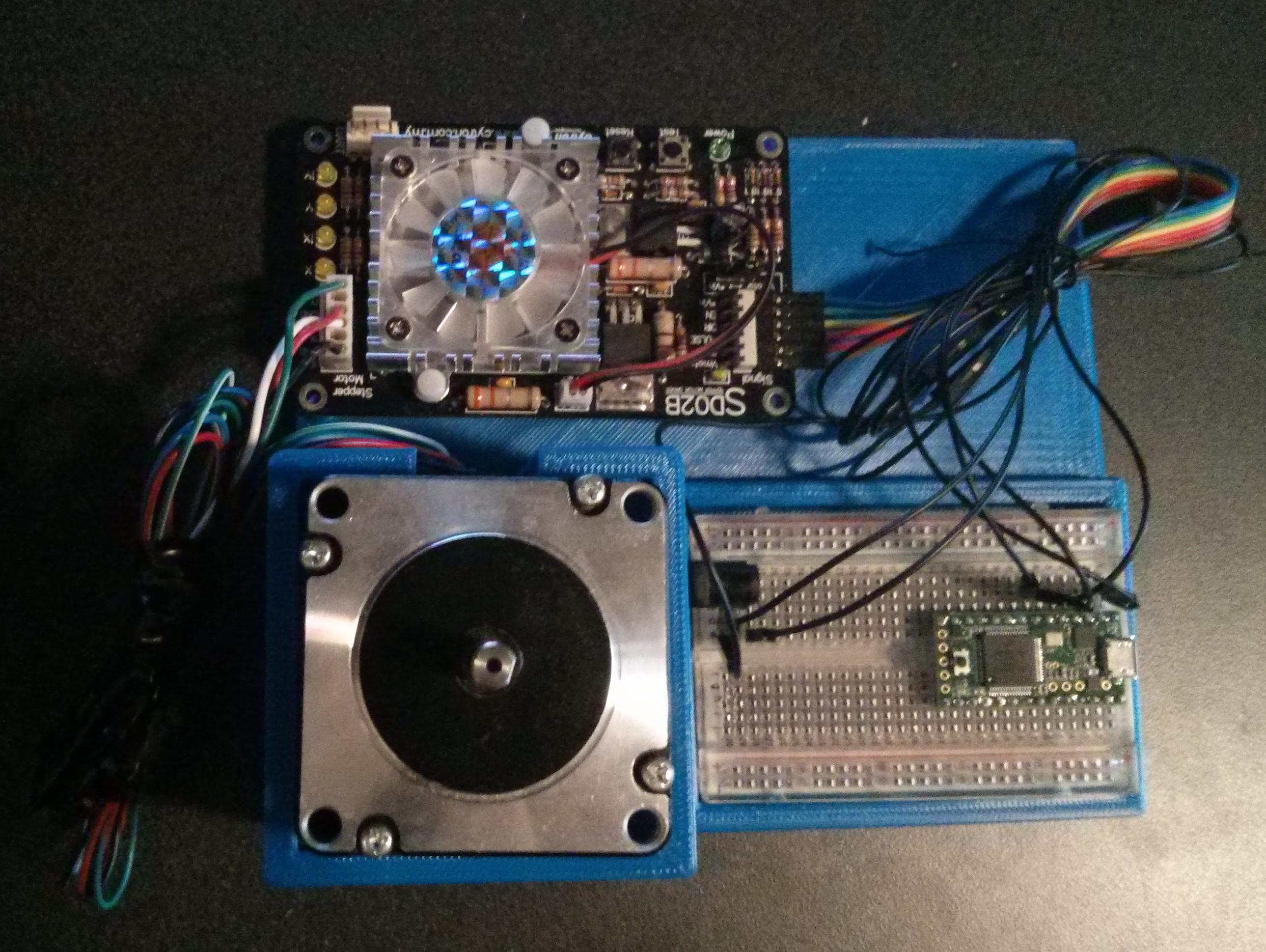


Figure 3: Mount without the canantenna

**Motor Interface**

For the second phase of the project the motor should interface with the server. From the server it receives the angle to which it should point. This implementation was done using the serial interface of the teensy board, which used the USB cable to emulate a serial port in the computer.

The server should send the angle in character by character, since the communication transfers 8-bit at a time and it was easier to decode if the data came as character than sending integers.

**Conclusion**

Overall, our system was successful when it was fully implemented with our antenna able to track our app allowing us to get reasonable RSSI as we traveled the selected route. Then we were able to push to a SQLite database with our data allowing us to parse the data. This parsed data was then rerun through the server where we reproduce the expected characterization of the ideal antenna. This data allowed us to have a quantifiable way to compare our data and see how well our system performed.