Design, Development, and Implementation OF Pinpoint: A Location Tracking Application

A Senior Project Submitted to the

Department of Computer Systems Engineering Technology

of the School of Engineering, Technology, and Management at the Oregon Institute of Technology

in partial fulfillment of the requirements for the Degree of

**Bachelor of Science**

Max Brandstetter

© June 2017

**Senior Project Approval Page**

The senior project of Max Brandstetter for the Bachelor of Science degree was accepted by the evaluation committee and the Department of Computer Systems Engineering Technology at the Oregon Institute of Technology.

COMMITTEE APPROVALS:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Wilsonville Program Director (Oregon Institute of Technology)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Reviewer 2 (Oregon Institute of Technology)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Reviewer 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Reviewer 4

**abstract**

The project, Pinpoint, is an Android application that uses Bluetooth devices called beacons to map and track a user within a room. The project itself is proof of concept an application that could similarly map and track users in malls, convention centers, and other large indoor areas. In addition, it aims to experiment with beacons, which have yet to see much public use.

The main features of Pinpoint are the map setup and the location tracking. Map setup encompasses a simple setup process, which instructs the user on selecting beacons, placing them, and scanning distances between the beacons. Following the process generates a map of the room, which can then be used to track the user's location in. The location tracking performs similarly. It measure the user's distance to each of the configured beacons, calculates the distances to each beacon, and then inputs coordinates and the distances to a trilateration calculator. The output is an approximate location, which is updated frequently to provide real-time tracking capabilities.

The project never reached completion, as I was unable to account for signal interference effectively. There were some steps that I took to counteract interference, such as collecting a data set from each beacon and filtering that of unusual values. However, due to the nature of Bluetooth signals and how trilateration works, even the slightest fluctuation can cause inaccuracy. The project is not a complete failure, as it works to some extent and was a learning experience for me.

**acknowledgements**

In completion of this project, I would like to acknowledge the professors at OIT for their commitment to their students, such as myself. The knowledge and lessons they passed on, including those outside of class, helped immensely throughout my education. In addition, I would like to acknowledge my family for their continual support. In particular, I want to acknowledge my older brother Taylor, who helped me brainstorm the project and encouraged me throughout the development process.

**List of Acronyms**

**BLE** Bluetooth Low Energy

**RSSI** Received Signal Strength Indicator

**UUID** Universally Unique Identifier

**dBm** Decibel-Milliwatts

**Summary Table of Contents**

Chapter 1 Introduction 8

Chapter 2 Background 13

Chapter 3 Functional Description 16

Chapter 4 Detailed Description 20

Chapter 5 Test Results & Validation 29

Chapter 6 Summary 34

**Table of Contents**

Chapter 1 Introduction 8

Overview 8

Product Description 8

Existing Products 9

Summary 11

Report Outline 12

Chapter 2 Background 13

Overview 13

Background Information 13

State of the Art 14

Summary 15

Chapter 3 Functional Description 16

Overview 16

Functional Description 16

Block Diagram 17

Summary 19

Chapter 4 Detailed Description 20

Overview 20

Detailed Description 20

Architecture 26

Summary 28

Chapter 5 Test Results & Validation 29

Overview 29

Test Plan 29

Test Results 30

Discussion 32

Summary 33

Chapter 6 Summary 34

Overview 34

Project Summary 34

Future Direction 34

Concluding Remarks 35

References 36

# Introduction

## Overview

This chapter provides a brief description of the project, alongside a comparison to similar products that have been developed. A report outline will be included at the end of the chapter to give an overview of the full report.

## Product Description

At this point, beacons have only seen use as a means of providing location context. To briefly elaborate, they can establish vicinity, but not location, and are therefore used as a means of monitoring traffic through certain areas and other similar tasks. This is partly due to restrictions inherent to how Bluetooth signals propagate.

Pinpoint aims to fill a gap in the Android application market by utilizing beacons in a more precise way. In particular, Pinpoint is a real-time mapping application that uses Eddystone Beacons to map a room and triangulate a user's location within that room. As the user moves around the room, so will the visual representation on the Android device.

The goal in developing Pinpoint was to cover a kind of niche that had yet to be addressed: maps and navigation in large buildings. To elaborate, most navigation in places like convention centers and malls is done manually, or rather there are no electronic means of navigating outside of static maps. With the technology available today, it seems unreasonable to go without a solution. By starting small and developing Pinpoint, I hoped to prove that it was possible to similarly map those building with beacons.

The same method is used to address both the problem of mapping and that of tracking within a room. To briefly explain, each beacon has an RSSI that can be measured by any Bluetooth compatible device. The RSSI varies based on the distance from the beacon, so by measuring and recording the RSSI at several specific distances, a power line regression formula can be made to predict distance. In doing so, other steps can be taken to map a room based on measurements, or triangulate a user's location similarly.

In addition, as a proof of a potentially larger application, Pinpoint is made up of two main parts directed at entirely different users. To simplify, the two users are administrative users who handle setup and other administrative tasks, and application users who simply use the location tracking capabilities. The first major part, map setup, is directed toward administrative users and should not involve application users at all. The second part, location tracking, is directed toward application users but realistically could be used by both users.

Upon completion, Pinpoint was meant to have the following characteristics or fill the following requirements:

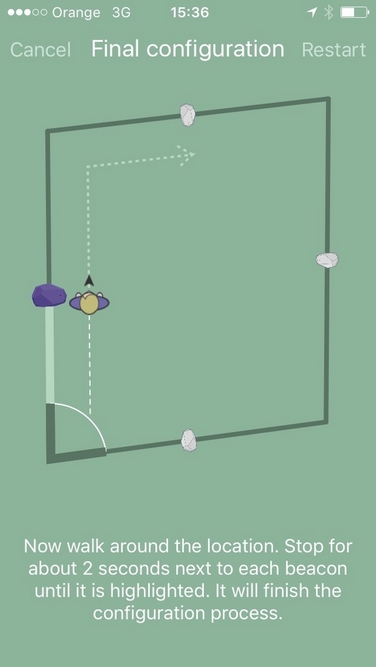
* Determine location based on data from beacons
* Convert location to visual data
* Predict distance based on RSSI
* Eliminate noise from beacon signals to prevent fluctuation
* Maintain a login system for storing settings and other information
* Contain a quick setup process for a beacon system and map

The project was meant to have other characteristics, but due to time constraints and issues finishing the characteristics listed above, they were removed. The above characteristics were completed for the most part, although it was not possible to eliminate all of the noise that Bluetooth signals can experience. This aspect is discussed in detail later in the report.

## Existing Products

The most similar products are Locatify, Lighthouse.IO, and Estimote. Currently, most applications that utilize beacons only do so to provide location context. In the case of Locatify [2] and Lighthouse.IO [9], the applications do precisely that. They can detect proximity and act as a platform to build on, but provide no legitimate location tracking capabilities. Although both Pinpoint and the other applications utilize beacons for location purposes, the result and use is widely different

Estimote, on the other hand, solves the same problem that I attempted to solve through Pinpoint. Using customized beacons, Estimote Indoor Location is capable of mapping a room and tracking users within it. The technicalities of how it works are not clear from any online sources, but the setup and known features work as follows.



**Figure 1. Map Setup [4]**

Before being able to track location, a user must first set up a map, much like with Pinpoint. A similar process is followed, in which the user places beacons at the center of each wall and moves along the perimeter of the room in order to construct the map. As with Pinpoint, this method of automatic mapping is not capable of mapping anything other than rectangular rooms [3]. However, the map can be manually created as well, allowing for flexibility in room shape and beacon placement, a feature that Pinpoint lacks.

Once the room is set up, any user with a compatible device will be tracked by the Estimote Cloud, so long as permission is given [3]. Essentially, all user location data is stored in cloud storage, and can be accessed to provide contextual location information, as Locatify and Lighthouse.IO did. Again, Pinpoint acts similarly, although since it is smaller scale, there is no cloud storage involved. In addition, while Pinpoint failed to track movement accurately, it seems that Estimote Indoor Location is capable of accurate tracking within one meter. The reasons behind Pinpoint's issues will be discussed later on in the report.



**Figure 2. Estimote Stickers [5]**

In addition to providing location information on users, Estimote Indoor Location also adds additional functionality through Stickers, which can be attached to physical objects as a means of providing additional location context [5]. For example, a sticker could be attached to a laptop, and if the laptop was ever lost, the information stored in cloud could be used to locate it. Stickers are tracked in this way by updating their location information whenever a user with a cloud compatible device passes it. So essentially, it stores the user's information as its own, but this gives constant context for its location in a room. This application of beacon technology is not something I considered and is completely absent from Pinpoint.

From what can be determined, Estimote Indoor Location has the same function of Pinpoint, although far more effectively. As such, it is definitely possible to use beacons for location tracking, although the most effective way to do that cannot be determined from what information Estimote provides.

## Summary

In itself, the completion of this project does not result in anything substantial or new. As such, considering Estimote's apparent success in doing what Pinpoint is meant to accomplish, Pinpoint does not hold much merit. That is, from a realistic or marketable standpoint, developing Pinpoint as a solo developer will not outdo a collaborative effort such as Estimote.

However, as a project, Pinpoint aims to work with beacons, which so far have seen little practical use, and only recently have hardware configurations such as Eddystone gone open source. Working on Pinpoint will therefore offer experience in many areas that I have yet to experience as a student and as a developer. Furthermore, since the technical details of Estimote are not known, it is not as if working on Pinpoint will be easy. The only things known are concepts, so while I could choose to mimic those concepts, it will be worthwhile to develop Pinpoint from square one and choose my own methods.

The merit of the project, then, is in personal growth, as working on Pinpoint will expose me to new technologies and allow me to work with platforms I have yet to use. In addition, the existence of applications such as Estimote Indoor Location proves that the concept is doable; I just need to determine how it is done.

## Report Outline

## The report is organized as follows:

* Background
  + Explain any background information required to understand the project
  + Overview of the State of the Art - how are beacons currently used?
* Functional Description
  + Provide a functional description and explain functional requirements
  + Explain block diagram of project
* Detailed Description
  + Expand on functional description in detail, providing more specifics on how the project operates
  + Discuss the organization and architecture of the project
* Test Results & Validation
  + Describe the test plan used for the system
  + Describe the results of the tests
  + Discuss the tests and explain the reasoning behind their use
* Summary
  + Provide a brief project summary
  + Discuss future direction for the project
  + Give concluding remarks

# Background

## Overview

This chapter provides background information regarding the project and the technologies it uses, in order to provide an understanding of technical details later on.

## Background Information

Beacons are the main component of Pinpoint, so in order to understand how Pinpoint works, it is important to understand how Beacons operate and what difficulties exist regarding their use. In addition, since trilateration is used to determine location, understanding the details behind that will help in understanding the design and workings of Pinpoint.

As previously mentioned, beacons are simply BLE devices. They broadcast an identifier via Bluetooth signals to nearby compatible devices, such as smartphones and tablets. To emphasize, beacons are a 1-way transmitter. That is, they do not receive information or interact with anything else, they simply send out a signal; it is up to applications and other devices to translate and utilize these signals.

The signal that beacons transmit is the crux of all operation and interaction, as it contains modifiable information that can uniquely identify the beacon, show its physical location, trigger location-based events, and more. This signal can also be modified in how it is transmitted by manually adjusting the broadcasting power or advertising interval. In addition, most beacons come with a factory-calibrated constant called Measured Power, which is the expected RSSI at 1 meter. This value is used in Pinpoint to create a distance estimating formula, among other things.

In the case of Pinpoint, not many of these features are utilized, as all distance measurement is done through RSSI. RSSI is not a piece of information that is transmitted, but rather a result of settings in the beacon. In particular, broadcasting power determines how strongly the Bluetooth signal propagates, and thus how the RSSI is read. Every Bluetooth device will have an RSSI, as it is merely an indicator of active signal strength.

This signal is not received at a constant value though. Just like any similar signal, Bluetooth signals can experience noise, or signal interference, as a result of various sources, such as Wi-Fi signals, reflective surfaces, other Bluetooth signals, and general human interference [1]. This kind of interference causes fluctuations in the RSSI of a beacon, which in the case of location tracking, causes distance measurements to fluctuate abnormally. As such, cancelling this noise becomes vital in order to maintain consistency in reading RSSI.

## State of the Art

It is difficult to accurately describe the state of the art in reference to beacons, as they have yet to see mainstream or widespread use. However, it is possible to generalize their use based on what applications do exist, as well as the trends that can be expected based on the technology itself.

One of the main uses of beacons at this point is in advertising, primarily in major retailers. Essentially, since beacons contain a UUID, applications can trigger specific events when certain UUID's are read. The event is fully customizable and is dependent on the application used, but advertisements would most likely take the form of a push notification. Of course, the user would be able to disable these notifications if they wanted, but it would be possible to encourage sales of certain products by notifying users in this way.

In addition, to acting as a method of advertisement, beacons in the same setup could be used to simply monitor user traffic through an area. By using something like the Estimote Cloud, it would be possible to log data whenever a user passed by a specific beacon [3]. Then, based on the history of data, retailers could rearrange products in order to balance the spread of traffic to other areas, or they could simply use the data to understand the most important products their customers purchase.

At this point, the only other main use is in providing location context for indoor navigation or other uses, which has been discussed briefly. In the cases of Estimote Indoor Location or Pinpoint, beacons can be used to track a user's movement through some mapped space. However, with the use of the Proximity Beacon API or similar services, it is also possible to provide contextual information [6]. To elaborate, beacons could be used to just indicate the floor a user was on and thus what could be found on that floor, or the approximate location of a user based on the nearest beacon. This has not seen extensive application yet, but could be immensely helpful in malls or large office buildings, as users would have a simplified quick reference when navigating.

## Summary

Beacons are simply Bluetooth devices that send out a signal, with additional data associated with that signal. Up until now, they have primarily been used for location context and proximity detection, which can work well alongside other data. However, they have the potential for application in other areas.

# Functional Description

## Overview

This chapter will provide a functional description of the project in order to provide insight to how users and other agents may interact with the application. The chapter also includes a block diagram, which provides a depiction of the overall system architecture.

## Functional Description

For functionality within the application, the following features are expected to complete:

* The user should be able to login to the application
  + User's without an account will be unable to move forward, but they can create a new account at any time
* The user should be able to view the map on the main screen using touch commands such as pinch-zooming, scrolling, and so on
* The user should be able to navigate between different elements in the application through a constantly accessible menu
  + The menu should not interfere with normal processes
* The user should be able to setup a new map if one does not exist, or if they want to replace the current one
* The user should be able to track their current location in the room, if a map exists
  + The user should also be able to start and stop the location tracking manually, although if active, the location tracking should update at a specified interval

The above features are based on assumptions of application use and the systems that will be used in the application.

Since this is an Android application, the application as a whole is built with modularity and flexibility in mind. All elements on all views or pages will resize based on the device in order to properly show everything. Touch screen capabilities are used where they fit, and all controls should be obvious in function to avoid confusion. This includes, but is not limited to, accessing the menu, logging in or logging out, navigating through the application, and so on. In addition, if elements are added later in the development process, they should be accessible via the menu and they should not directly interfere with any other processes.

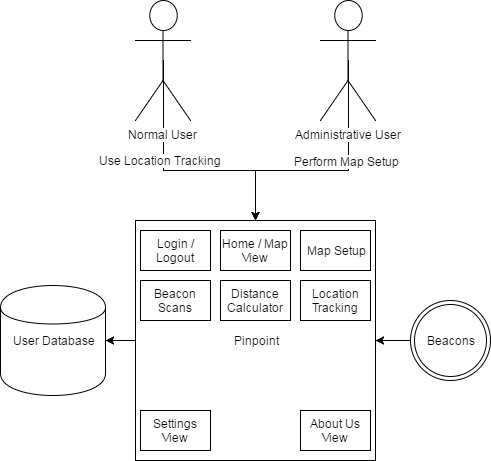
The core of the application is split between mapping a room and locating a user, so the majority of the functionality is located in those components. Mapping a room should be a simple setup process, much like a setup wizard for a desktop application. In short, the user can access a map setup option from anywhere in the application, and that will walk them through selecting what beacons to use, setting up the beacons, and scanning distances. This is done to avoid confusion in setting up the system and ensures that the process is done correctly in order to meet the application's assumptions.

Locating a user should be incredibly straightforward and require little user interaction. If a map is available, it should be shown on the screen, along with indicators for beacon placement in order to provide some reference when navigating the room. To start tracking, the user should only need to press a button, which will trigger all necessary processes to locate them in the room. Similarly, if they want to stop tracking, the process should be stoppable by pressing a button, or exiting the application in any manner.

All mapping and location tracking is done by scanning for the beacons linked to the current map and determining the user's distance to them. To reiterate, the beacon merely send out data, the user's device and the application are responsible for translating that information as needed. No user input outside of selecting the beacons should be required when determining distance or making other measurements.

## Block Diagram

The majority of Pinpoint's design was figured out as the project progressed, as there were no definite database or class interactions that needed to occur. In addition, many of the resources that connected objects were not used, so there is extraneous functionality in many places. As such, what follows is a simplified block diagram to illustrate major components of the project.



**Figure 3. Simple Block Diagram of Pinpoint**

Although relatively simple, Figure 3 visualizes the basic design of Pinpoint. In itself, Pinpoint is singular in purpose. It contains a few views as well as a login system which stores user data in a database, but the majority of the design is focused on support the map setup and location tracking components. As such, the program only communicates with the user database and the beacons, as they are the only parts or services that are needed.

As shown above the block representing Pinpoint in Figure 3 are the two users: normal users who simply utilize the location tracking, and administrative users who make sure the system is set up properly. In the end, Pinpoint combines accessibility of these users. However, if the project was to be developed further, these two users would be clearly separated, and more structure would be added to allow users to select maps based on what beacons are nearby.

Overall, the design of Pinpoint is fairly simple and follows the traditional design of Android applications. Each view has its own XML file and associated activity, which handles all functionality, and the intent (the active view) is passed between activities based on the controls implemented.

## Summary

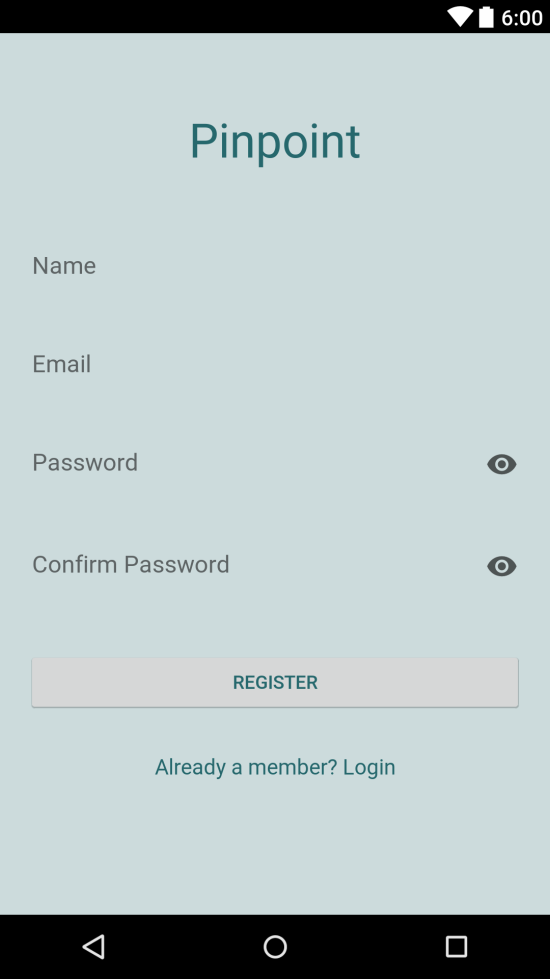
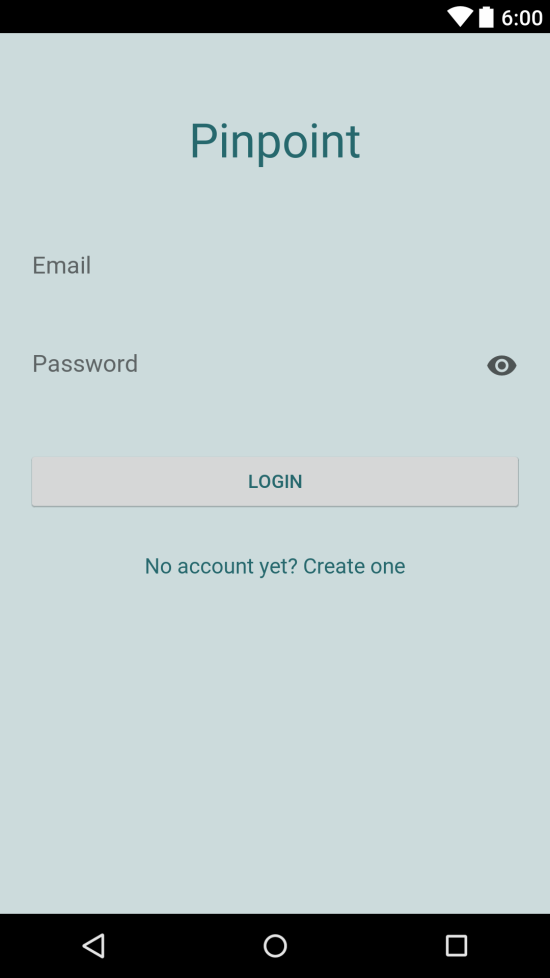
Pinpoint's function is primarily to be able to map a room and track a user within that room. However, since it is meant for public use, it also needs to be flexible and easily understandable. As such, it is relatively simple in terms of technical design, as it follows a standard Android model.

# Detailed Description

## Overview

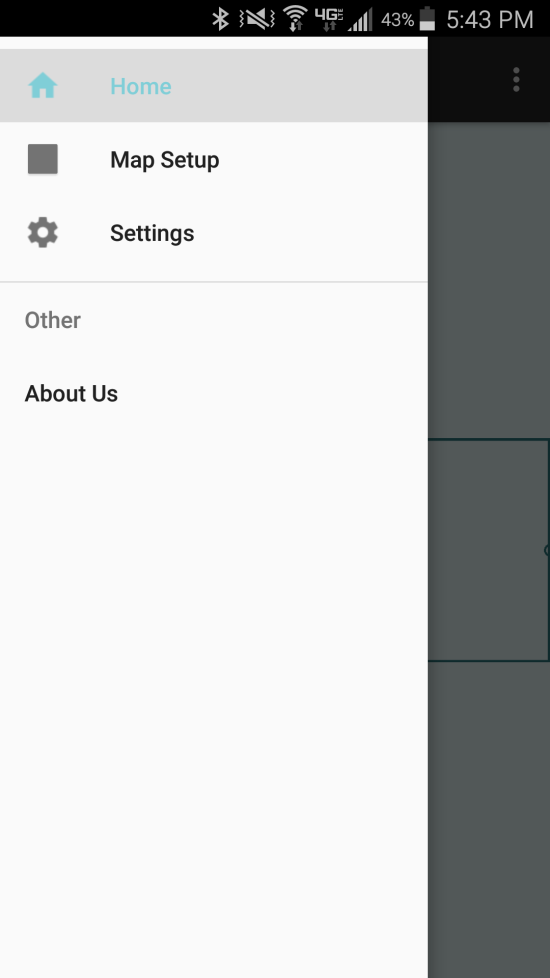
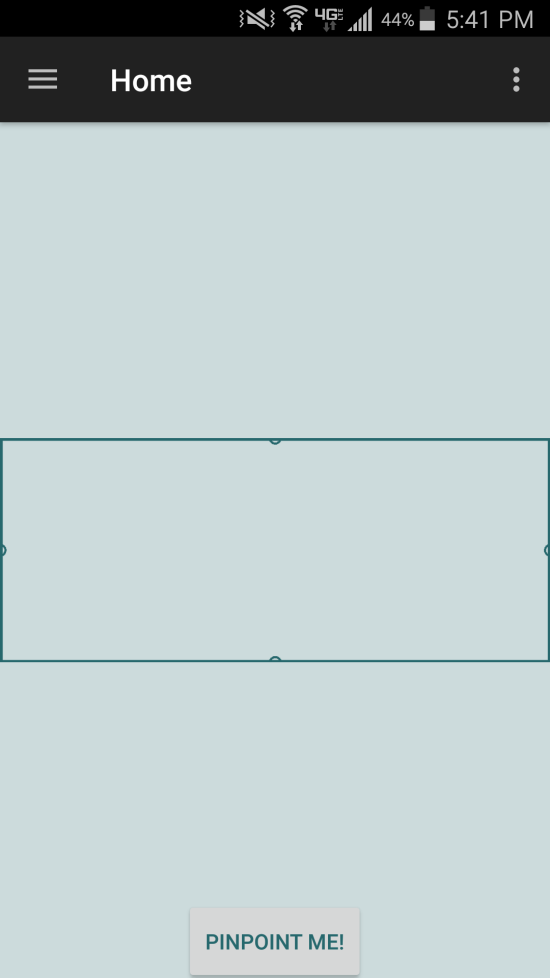
This chapter provides a detailed explanation of how the project functions, supported by diagrams and other design illustrations where applicable.

## Detailed Description



**Figure 4. Login (Left) and Register (Right) Views**

Whenever the application is first started, it opens to the login view, seen in Figure 4. This serves as a gateway to the main application, which can only be accessed by users with legitimate login information. If a user does not have an account, they can easily create a new one by following the links provided on the page. To register, a user simply needs a user name, an email, and a password, as shown in Figure 4. All of the fields are error checked to ensure that they are not empty and follow the necessary format, in the case of emails. Currently, all account information is stored locally in a SQLite database. This is primarily done for convenience, as I had difficulty finding an effective platform to deploy a cloud database system on; it would be fairly easy to deploy if I chose to, but it was unnecessary for the completion of this project. Once an account is created, the user is brought back to the login screen, where they can enter their information and continue.



**Figure 5. Home View (Left) and Menu (Right)**

The home view, seen in Figure 5, contains very few elements but allows the normal user to utilize all the functions of Pinpoint. The view itself is of the map, which once setup, will be capable of displaying the user's current location in the form of a cyan dot. Controls are provided at the bottom of the view which allows the user to start and stop the automatic tracking as they want (the specifics of location tracking will be discussed later on). The top right of the screen contains an icon which once pressed, allows a user to log out of the application and return to the login screen. To enter again, they must reenter their login information. On the top left of the screen is a menu icon, which once pressed will open a menu tray containing navigation elements for the "Home", "Map Setup", "Settings", and "About Us" views. The "Settings" and "About Us" views serve as placeholders for now and contain no unique information.

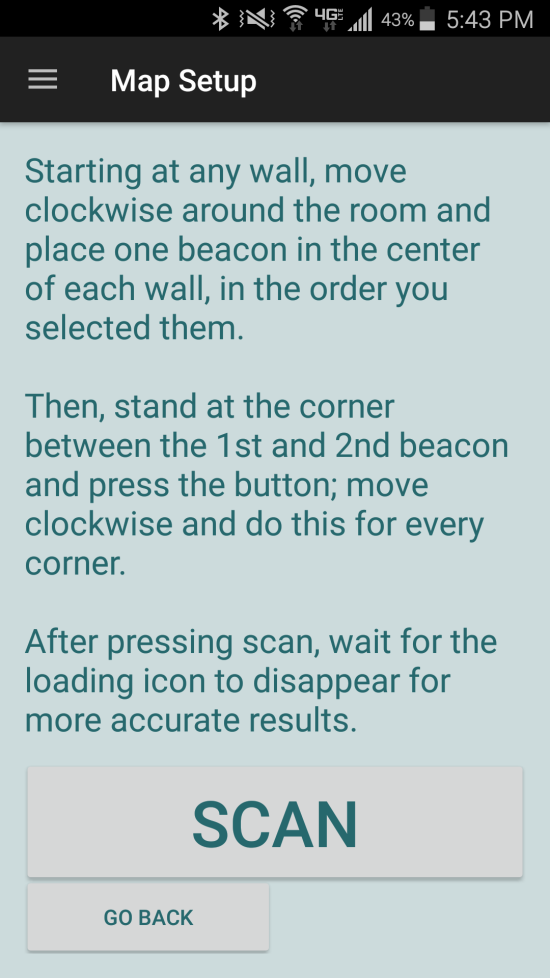
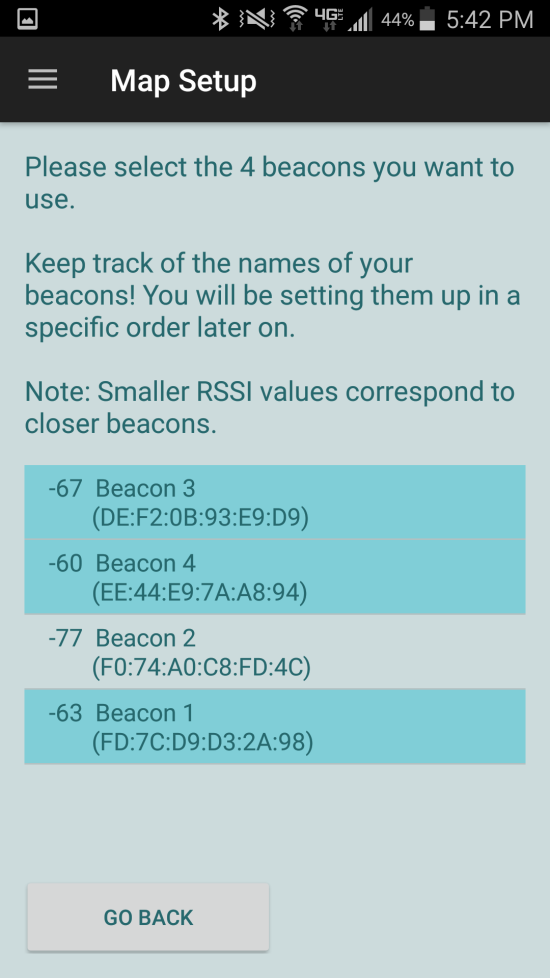
|  |  |  |
| --- | --- | --- |
| **Distance (meters)** | **RSSI (dBm)** | **Ratio (RSSI / -59)** |
| 0 | -27 | 0.457 |
| 1 | -59 | 1 |
| 3 | -67 | 1.136 |
| 5 | -71 | 1.203 |
| 10 | -79 | 1.339 |
| 15 | -84 | 1.424 |
| 20 | -87 | 1.475 |
| 30 | -92 | 1.559 |
| 40 | -94 | 1.593 |
| 50 | -100 | 1.695 |

**Table 1. Distance and RSSI Measurements**

Before addressing the specifics of the map setup and location tracking portions of the application, it is important to understand the core functionality behind the distance measurement used in both parts. As mentioned earlier in the report, a power regression formula is used in order to predict the distance to a beacon based on the RSSI read from that beacon. The formula follows the power regression model (y = A \* x^B + C), where distances and their corresponding RSSI measurements are used as inputs, as shown in Table 1. By establishing a ratio for each RSSI value, based on the RSSI value measured at one meter, we can determine the independent regression variables, and use those to determine A, B, and C, which are used in the power regression formula. To clarify, one meter was chosen as the value to zero on as it provided the most accuracy at distances up to 20 meters.

**Figure 6. Power Regression Formula for Distance**

The end result, shown in Figure 6, takes the ratio of current RSSI over RSSI at one meter as input, and returns accurate measurements of distance up to 15 meters, after which the accuracy degrades rapidly. In addition, since the formula is based on measurements at one meter, the predicted distance for distances less than one meter end up approaching a negative value. To account for this, any time a predicted distance is calculated to be less than one, the value for C is removed and the distance is recalculated. This is because the initial formula is zeroed at zero, so any calculation made without C will be a correct estimation for values less than one.



**Figure 7. Map Setup: Select Beacons (Left) and Scan Beacons (Right)**

With the distance formula established, map setup and location tracking can be more easily understood. Map setup itself is a fairly straightforward operation, and thorough instructions are provided throughout the process to assist the user. To start, the user simply needs to select four beacons to use from a list of nearby beacons, as shown in Figure 7. The list will only populate with Bluetooth devices with valid UUIDs, which eliminates most normal devices. The user cannot move to the next step until they have selected four beacons and if they try to select more than four beacons, they will be notified and no changes will be made to the list of selected beacons. To clarify, the user selects only four beacons because the application does not support irregular rooms, as it was found to be outside of the scope of the application.

Once the user selects four beacons, they move to the next part of setup, which has them place a beacon at the center of every wall and scan the distances between them. This is done by standing at the corner of the room between the first and second beacon placed, and pressing a "Scan" button, seen in Figure 7. A dialog will prevent further action while the device scans for the beacons.

During this time, the application is actually collecting around 20 scans from each beacon in order to attempt to account for noise experienced during setup. The value 20 was chosen arbitrarily as an amount of scans, but the goal was to increase accuracy while not causing ridiculously high scan times. The more this value increases, the more consistent the scans would become in theory. Regardless, once the scans have been collected, a 90% confidence interval is calculated and used to remove outlier RSSI values from the set of scans. Once finished, the remaining values are averaged and input to the distance formula. The goal in doing so was to account for sudden fluctuations in RSSI caused by people walking in front of beacons. Unfortunately, this method only helped somewhat in accounting for noise, so while it is an improvement over the initial method I had implemented, there is still too much fluctuation for the application to be accurate. This will be discussed in further detail in Chapter Five.

The above process is repeated for the corner between the second and third, third and fourth, and fourth and first beacons. When all scans are finished, the setup process is finished by averaging the opposite walls to find the width and length of the room. These values are used to generate a bitmap image, which is then saved for use by the user. The bitmap, along with any other drawn elements, are scaled using an expansion factor. That factor is based on the largest of width and length, divided by 1000; this results in the greatest dimension of the bitmap being 1000 pixels. In addition, upon completion of the distance calculations, the user is notified of the setup's completion and return to the home view, which now contains the map they generated.

Location tracking occurs in a similar way to the map setup, except instead of scanning from corners of the room, the user can start locating from anywhere in the room and the application will track them. Once they start tracking but touching the "Pinpoint me" button, seen in Figure 5, the device starts to collect 20 scans from each of the four surrounding beacons. Again, each set of scans is cleared of outliers and the resulting RSSI is input to the distance formula. The four distances are then entered into a trilateration calculator (obtained through an open source Java library [10]) alongside coordinates for each beacon.

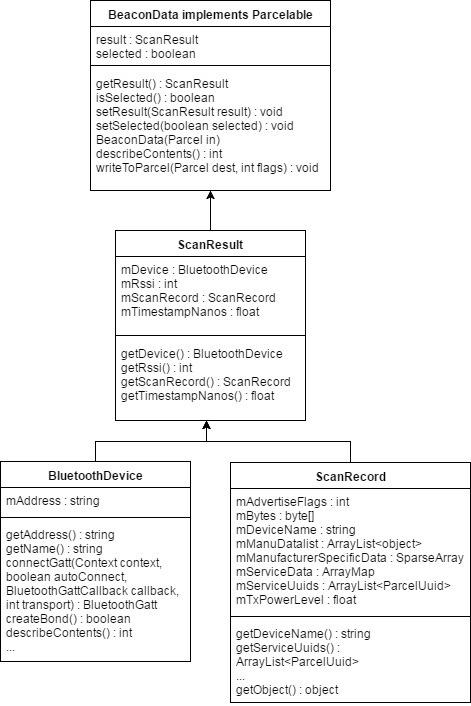
To explain the trilateration calculator briefly, it works by drawing a circle around each coordinate where the radius is the distance entered. The coordinates in this case represent the beacons. Once the circles are drawn, the calculator uses the Levenberg-Marquardt algorithm [8] to estimate the most likely location of the centroid, or center of the ellipse found between each of the four circles. The resulting point can then be drawn directly to the stored map as a circle, which scaled using the expansion factor mentioned earlier, and is loaded in place of the current map.

So long as the automated tracking is active, the above process occurs every second, except rather than gathering 20 new scans for each beacon, values are replaced as in a queue. Pinpoint works primarily with lists, meaning that to replicate a queue, the first value in the corresponding list is removed, all other values are re-indexed, and the new scan is added to the back whenever one is received. This results in quicker updates for the user's current location, as well as movement through the room being shown through gradual changes.

The only other functionality to note outside of the two main features is how maps persist and are saved, and how values are passed between intents or views. The way each operates is pretty standard in Android development, but stands out due to complexity. Essentially, whenever a map is created, it is physically saved to the device. The goal was to save to a database, but time did not allow for that to be done. In order to access the map again in cases where the user closes and reopens the application, the name of the map is saved to Shared Preferences. Shared Preferences allows us to associate data in the form of key/value pairs to an application, so unless the user deletes the application from their device, the preferences will remain unchanged. So, once the name of the map is saved, it simply needs to be pulled from the Shared Preferences for the application and loaded.

Although passing values between intents could be done in the same way, it is more efficient to create a bundle. Bundles allow us to similarly save key/value pairs, but rather than associating it with the application, it is associated with activities. As such, if the user were to close the application and reopen it, the bundle would be deleted. Bundles can be passed between intents, the values can be loaded, removed, or added to, or the bundle can be left alone if it is no longer needed. This is primarily used during map setup in order to pass data such as the selected beacons forward for use in scanning, among other things.

## Architecture

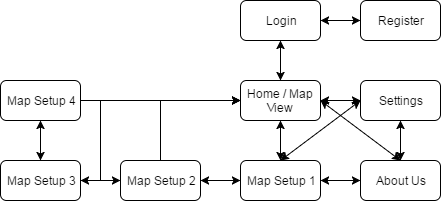


**Figure 8. BeaconData UML Diagram**

Figure 8 shows a UML diagram for the BeaconData class, which stores all important data received from a Bluetooth device. The class itself is fairly simple and mostly serves as maintaining single beacons rather than needing to remove and recreate them during updates. The object of interest in BeaconData is the ScanResult, which in turn contains a BluetoothDevice object, a ScanRecord object, and several helper fields and methods. A ScanRecord contains all the information that the application is capable of detecting and serves as the basis for most work with beacons within the application. Primarily, we are interested in the RSSI, but the BluetoothDevice and ScanRecord provide information that allows us to uniquely identify beacons from other devices. This helps with persistence and storage, which is discussed later on.

To reiterate, the purpose of the BeaconData class is to store beacons, so whenever a beacon is scanned, the resulting data is stored as a BeaconData object. Then, data from that beacon can be accessed and used to whatever capacity is needed. In the case of the distance calculator class discussed earlier, it will access each BeaconData object's ScanResult in order to get the RSSI. This is the most important bit of architecture used in Pinpoint and serves as the basis of most functionality, whether it is actual distance calculation or simply checking if a Bluetooth device is actually a beacon.

As mentioned earlier, all account information is stored locally in a SQLite database. Currently, the only information stored is the account itself, which includes the username, email, and password for any account. The goal was to eventually add settings to the application, which would be stored in the database alongside the account information, and loaded at runtime for a more customizable user experience. However, other components were given priority, so the login system and database only serves as a gateway to the rest of the application.



**Figure 9. Navigation Flow**

Figure 9 is a simple diagram depicting the navigation flow used in Pinpoint. Pressing the "Back" button on the Android device will return to the previous screen, unless trying to return to the Home View after logging out. There is no way to undo this navigation.

All operations start at Login and continue from there. From the Login, a user can login and move to the Home View, or move to Register and register a new account. Once on the Home View, users have access to the menu tray, which allows them to navigate between Home View, Settings, About Us, and Map Setup 1 as they want. They can also log out, which will return them to the Login. If a user enters Map Setup 1, they can continue with the map setup process, or return to the Home View without finishing. At any time during the map setup process, they can freely return to the previous step (i.e. Map Setup 3 to Map Setup 2).

## Summary

The method for distance calculation forms the core of both the map setup and location tracking features. In a perfect environment, it works effectively by approximating distance based on the RSSI of a beacon. There are other features of the design, which are briefly detailed, but they are not as significant. The architecture is relatively straightforward, although the primary class element, BeaconData, is slightly complex. It relies on inheritance and is the main object used when working with beacons, so it is important.

# Test Results & Validation

## Overview

This chapter gives an overview of the test plan and validation study used for the project. In addition, the chapter provides the results of those tests and discusses them for analysis.

## Test Plan

For this project, I had no main plan with regards to testing, as going into the project, I still was not entirely sure how everything would work. In the end, I avoided most methods of automated testing, such as unit tests, and relied on thorough manual testing as there were not many components to actually test. Primarily, this was needed because half of the project was just attempting to account for irregularity in the beacons, so testing manually in a consistent environment was the best way to get results.

For testing the map setup and location tracking, I setup the beacons in my garage (which measure about six meters by 7 meters) at the same points each time. To validate that the location tracking, and thus the map setup, was working, I checked my location on the device at the center of the room, in front of each beacon, and at each corner for every test. Occasionally, I would check my location at other points in the room, but if the previously listed locations were found consistently, then I considered the project to be functional.

Outside of testing the two primary features, other portions such as the login system or program navigation were validated by performing minor stress testing, checking for screen orientations, and so on. If a section was found to perform the same regardless of the above testing, then it was considered complete. Components outside of the map setup and location tracking were rarely modified after their initial completion, so additional testing was not needed. In addition, simple usability testing was done in order to check that the program was navigable and understandable to other users.

## Test Results

Despite my best efforts to make the map setup and location tracking both accurate and consistent, the program failed to reach completion in that regard. Although more of a discussion point, this serves as a precursor to the results that follow. Essentially, in performing the test for location tracking described at the start of this chapter, the program would consistently locate me if I was somewhere near the middle of the room. However, as I moved to the beacons and the corners of the room, the program would either continue to show me in the middle of the room, locate me outside of the bounds of the room opposite of where I was standing, or something similarly incorrect.

As mentioned, this was an expected issue, as Bluetooth signals fluctuate easily and can be interrupted by several sources. As such, much of the work I did in testing and continuing development was to prevent this. In gathering a large amount of data and normalizing it, I had hoped that the situation would be resolved. It did help, but the issue of direct interference by objects, especially for extended periods, remained unresolved.

|  |  |  |
| --- | --- | --- |
| **Interference Used** | **RSSI (dBm)** | **Calculated Distance (meters)** |
| None | -59 | 1 |
| Hand Covering Phone | -78 | 8.6 |
| Body Between Phone and Beacon | -81 | 10.7 |
| Other Bluetooth Devices Near Beacon | -76 | 7.4 |
| Sheet of Paper In Front of Phone | -64 | 2.3 |

**Table 2. Interference Sources' Effects on RSSI**

Table 2 shows some example measurements of RSSI at one meter away, along with the interference introduced in each measurement and the calculate distances based on the RSSI. As seen, even the slightest interference, such as a piece of paper held in front of the device, can cause the calculated distance to fluctuate by over one meter. This illustrates the importance of noise cancellation, especially in a system focused on precision. Again, gathering large amounts of data and averaging it helped to reduce the affect of the interference, assuming that there is only interference for short periods. However, in cases of constant interference, another approach must be taken, and I did not have the foresight to implement that early enough. The proper steps I should have taken will be discussed later in this chapter.

Essentially, the result is that the program correctly maps a room and locates a user in a perfect environment. When a source of interference is introduced, it will be handled so long as the interference is temporary and severe. Otherwise, the set of RSSI values used to determine distance becomes the fluctuated value itself. Furthermore, since the program uses trilateration to locate a user, any change in the distance to one beacon without a respective change in another confuses the system. This is why the most common fail scenario when scanning away from the center of the room was that I was still in the center of the room or far outside of it.

There were not many parts of the applications outside of the map setup and location tracking that required changes. The most important changes found in testing came about by testing device orientation (held vertically or held horizontally), and navigating through different sections. To explain, whenever the device's orientation changes, the current view and activity are reloaded. In most cases, this had little effect on the program. However, in parts of map setup or the home view, this completed removed the current progress. As such, I had to change specific activities and views to ensure that if the orientation changed, either the data was saved and loaded to retain progress, or changes to the orientation did not actually reload the page.

Other than that, a lot of crashed were found and resolved through pressing buttons outside of the context of their use. For instance, pressing the "Pinpoint me" button on the home view used to crash the application if the beacons could not be found or if there was no map. Or attempting to select more than four beacons during map setup used to similarly crash the application as the next part of setup expected only four. Both issues and others were easily found and resolved through this method of testing, although realistically, implementing unit tests or some other form of automated testing would have ensured that these problems do not resurface.

In performing usability testing, my main goal was to ensure that the application made sense to a user that had not seen it before. As such, I tended to test it with people that had not seen Pinpoint before. When testing, the testers were meant to complete two scenarios: setup a new map, and track their movement through a room. This was partly to test functionality, but also to ensure that the user had a smooth experience using the application and did not experience confusion in performing any of the tasks.

The results of usability testing were relatively consistent and encouraged several changes throughout the development cycle. For instance, during map setup when the user must scan the distance to each beacon, there was not a loading dialog initially. I suspected this would be an issue and testers would consistently move away from the spot they were scanning from despite instructions to do otherwise. Adding the loading dialog made it clear that the testers needed to wait and led to a more fluid user experience. Other than that, there were some small issues such as the presence of a "Pinpoint me" button on the home view, which does nothing if there is not a map to use. Again, clarifying its use through pop up dialogs in the case of misuse helped to guide the user without being too intrusive.

In addition to performing the tasks, I also had testers complete a short survey to determine any other problem areas that I may not have noticed through observation. The survey included questions such as "Was the application easy to navigate? If not, where did you have trouble?" and "If this was a real-world application, would you use Pinpoint? Why or why not?" Answers to these questions helped me determine what parts of the application needed improvement, without influencing the user's ideas. Overall, usability testing helped me to clarify aspects of the UI that I may not have noticed.

## Discussion

With regards to navigation and general application flow, the project meets the standards I had established. Pinpoint is bug free, as far as I can tell, and is a simple and easy to understand application. However, since the core of the application was focused on the idea of tracking location using beacons, the completion of the project was based on the completion of that portion. Based on the results of my tests, Pinpoint is not complete, as the map setup and location tracking portions are still too inaccurate.

As mentioned throughout the report, the main reason for this failure is in the nature of Bluetooth signals. Since the application is based on working with Bluetooth, I needed to handle noise correctly in order to complete the project. While I did handle noise to a certain extent but implementing outlier removal and making calculations based on a data set rather than a single element, it was not sufficient.

From what I understand, the most effective way to handle noise would have been to implement a system to detect when signals changed too severely. This is easy to imagine in some cases: if the RSSI from a beacon is measuring -60 dBm and suddenly spikes to -90 dBm, it can be inferred that the user did not just move 29 meters over less than a second (see Table 1 for RSSI/Distance measurements). In this case, we could get the difference between the correct value and the spiked value, and simply remove that from the current RSSI until it returns to normal. This works fine in theory, but still does not account for instances where the user is moving across the room. As such, even that is not a complete solution, as it is built on assumptions that will not always be true.

Another minor fix would be to increase the amount of data I collect before determining the distance to a beacon. It would likely increase accuracy and help account for some extended interference. Unfortunately, the cost would be drastically higher scan times, which would result in the location tracking being far from real-time, which was the goal in development.

Similarly, the system could be modified to allow more beacons to be setup in the same room. Hypothetically, adding more data points as reference for trilateration would increase the accuracy of estimates. The downside here is increased cost for simply mapping a room.

Other small fixes could help, such as implementing a Gaussian filter by removing a few maximum and minimum values from the data set, and using the average of the remaining elements. On the same line of thinking, I could implement a Kalman filter [7], which uses recursion to predict future sets based on the past and current values. It would be far more difficult to implement, but could have the greatest impact in handling noise for beacons. As far as altering the current system in Pinpoint, these are the only solutions I can figure out.

## Summary

This chapter provided an overview of the testing methods and validation requirements used in this project, as well as the results of those tests. The goal was to have accurate and consistent location tracking, but due to insufficient handling of signal interference, the project did not reach completion. There are some solutions that may help, but due to time constraints and lack of knowledge, they could not be implemented in time.

# Summary

## Overview

This chapter provides a summary of the project and development process. In addition, it discusses future direction for the project in case I or someone else attempts to make improvements to the project.

## Project Summary

In summary, Pinpoint is an Android application that aims to use Bluetooth beacons to map a room and track a user's location in that room. The project itself is a kind of proof of concept for a larger scale application that could similarly map and track users in malls, convention centers, and other large indoor areas. The goal in doing so is to fill a niche in the Android/iOS market that has not been filled: accurate electronic navigation for large buildings.

The project is focused primarily on the map setup and location tracking portions, although it contains more, because they form the core of the application. They work by calculating distance to a beacon, which is done using a power regression formula that converts RSSI to distance. The formula is mostly accurate, but since beacons merely emit a Bluetooth signal, there is also the issue of noise cancellation to deal with.

There were attempts to account for signal fluctuation caused by interference, such as collecting a data set rather than a single element and removing outliers from that set. However, this only solved part of the issue, as it did not account for interference that occurs over an extended period of time. As such, the project is not technically complete, as it does not locate a user accurately or consistently.

## Future Direction

Considering the existence of other products such as Estimote Indoor Location [3], I believe that the project is still doable, it just requires some changes. If I were to continue work on it, I can only see two major ways to fix the current issues and reach completion.

First, I could attempt to implement a Kalman filter [7], as discussed in Chapter Five. The Kalman filter uses recursion, along with past and present values, in order to create a predictive model. This predictive model helps account for noise since it is based on data from a longer period of time. This is done by minimizing covariance, or variability of two or more values, when certain conditions are met. It would be difficult to implement, as it is not a simple algorithm. However, it would definitely increase accuracy in comparison to the current system of noise cancellation.

Second, I could scrap my current system which is based on precision distancing and opt for a system entirely based on proximity detection. I am not sure if this could actually work, but I imagine that if I could determine set thresholds for being close, near, far, and so on from a beacon, then I could create a grid system that based on the beacon locations. Then, instead of getting precise distances to each beacon, I can base the approximate location of a user based on their approximate proximity to any number of beacons. iBeacon, which is the iOS compatible beacon format, is often used in this manner, so it seems doable although much more extensive than the first solution.

## Concluding Remarks

In itself, I do not feel that failing to reach completion on this project is negative. It would have been nice to complete the project as I imagined it, but there are benefits to the work I did complete. For instance, most of the technologies I worked with on this project including Android, Java, and Beacons, were technologies that I had no prior experience with. As such, I learned a lot about effective programming in Android and Java, as well as the features available to Android developers. With regards to beacons, I still do not understand enough, but it was interesting to use a newer technology that has not seen much use yet. Of course, that caused a lot of problems as there is not much information available regarding beacons, but it posed a unique challenge that helped me maintain interest in the project. In short, although the project is not complete by any means, I feel that it was an effective means of personal growth as a developer.

**References**

[1] A. Madhavapeddy & A. Tse. (2005). *A Study of Bluetooth Propagation Using Accurate Indoor Location Mapping* [Online]. Available: http://www.cl.cam.ac.uk/research/dtg/www/publications/public/acnt2/ubicomp2005-bluetooth.pdf

[2] Blaz. (2015, March 3). *Indoor Positioning Systems based on BLE Beacons - Basics* [Online]. Available: https://locatify.com/blog/indoor-positioning-systems-ble-beacons/

[3] Estimote. (2017). *Estimote Indoor Location* [Online]. Available: http://estimote.com/indoor/

[4] Estimote. (2017). *How to setup up a location in Estimote Indoor Location* [Online]. Available: https://community.estimote.com/hc/en-us/articles/203439046-How-to-set-up-a-location-in-Estimote-Indoor-Location-

[5] Estimote. (2017). *What are Estimote Stickers* [Online]. Available: https://community.estimote.com/hc/en-us/articles/203323543-What-are-Estimote-Stickers-

[6] Google. (2016, February 9). *Proximity Beacon API - Overview* [Online]. Available: https://developers.google.com/beacons/proximity/guides

[7] G. Welch & G. Bishop. (2006, July 24). *An Introduction to the Kalman Filter* [Online]. Available: https://www.cs.unc.edu/~welch/media/pdf/kalman\_intro.pdf

[8] H. Gavin. (2017, March 22). *The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems* [Online]. Available: http://people.duke.edu/~hpgavin/ce281/lm.pdf

[9] Lighthouse. (2017). *Indoor Location Technologies Compared - Beacons* [Online]. Available: https://lighthouse.io/indoor-location-technologies-compared/beacons/

[10] S. Wiedemann. (2017, May 9). *Lemmingapex Trilateration* [Online]. Available: https://github.com/lemmingapex/Trilateration