# Minimap

### Team North Star

|  |  |  |
| --- | --- | --- |
| Chris Carey | Benjin Dubishar | Josh Osgood |
| JP Mardelli |  |  |

### Project Sponsor

### Paul Mittan, Lockheed Martin

### Faculty Coach

Prof. John Loser

### Project Overview

Minimap is a system that allows for the mapping of venues like science fairs, conventions, etc. and tracking of system users on the created map. Paul Mitton from Lockheed Martin wanted a system that would deliver a high fidelity map of an event to users and also a means for tracking them with GPS-like accuracy and latency to display their positions on the map.  
  
The system consists of three main programs: Cartographer, which allows an event coordinator to annotate a map with table positions, physical barriers, and tracking hardware, Bridge, which hosts the map, team, and other client information, and the Android client, which grabs the information from the bridge and displays the user's map and relevant information, as well as dealing with tracking the users location. Bluetooth 4.0 LE beacons are used for tracking. Multilateration based off of signal strength is used to estimate where the user's device should currently be positioned on the event map.

### Basic Requirements - Benjin

Minimap is intended to be a complete, end-to-end indoor positioning suite. While Bluetooth 4.0 LE was selected to be implemented as the positioning technology of choice, the project is designed to work with any technology that supports the conversion of a signal from a point of known location to a distance. Minimap’s requirements predominantly fell into two categories: positioning and usability.

Past attempts at completing this project fell short either in accuracy and precision or in the update speed of the position. As such, Minimap had to satisfactorily fulfil both the continuous updating and “usable” accuracy requirements; the marker representing the users position had to track the user’s position closely so it could be used for navigation.

The primary usability requirement was that novice users be able to set up the system with minimal effort and in less than 30 minutes. Preparation for running the positioning system included the creation of a metadata map, which included the locations of beacons. This metadata map had to be distributed to the connected client devices (users), who then needed to be organized into teams from the administration console. The application in which this administration console and the server existed had to ingest the most recent positions from users and distribute the full set back to everyone.

Minimap comprises three apps (two server-side, one on the client) that work in concert, but other than that, it does not interface with any other systems.

### Constraints

### There were a number of constraints that the team dealt with during this project. Due to the hardware technology aspect of our project, the two largest constraints were time and money. Time was constrained because of the time required to determine a technology for location positioning through a trade study as well as the time required for devices to be ordered and shipped. This reduced the time available for full-scale development of the system. Money constrained the team by limiting the amount of devices available for our project, especially since it was important to prevent the sponsor from investing into a technology that might not meet project required. The team selected the Android platform in part due to an inability of iOS to read Wifi signal strength, which at the time was a potential technology solution. The tracking algorithm required a tradeoff between accuracy and latency due to the infrequent signal strength readings as detailed in the system design section of this report.

### Development Process

*Team Roles*

Chris Carey - Development Coordinator

Benjin Dubishar - Sponsor Communicator

Josh Osgood - Website Coordinator

JP Mardelli - Testing Coordinator

*Process Methodology*

Based on the needs of the project sponsor, an agile process was to be used for the development phase. Upon further consideration of the system constraints and considerable risks, a spiral methodology with two-week sprints was approved and adopted for the development phase. With multiple hardware-based unknowns, each two-week sprint focused on overcoming, eliminating, or diminishing a major risk. The greatest risk of the system was that the method of determining the distance to a beacon was inaccurate. This risk would exist whether or not the beacon was replaced with a Wifi router, an Ultrasonic emitter, or some other ranging technology compatible with consumer mobile devices. With multiple sources ranging from academic papers to developer forums reporting a wide range of success or failure with each system, this risk could only be overcome by implementing the system. Without the time or budget to attempt to implement multiple technologies, or the domain expertise to implement them concurrently, once Bluetooth technology was selected, the risk exposure was set. Each two-week sprint aimed to reduce the risk of inaccuracy, as much as the hardware technology allowed.

This risk of inaccuracy was split into smaller risks from which two-week sprint goals were derived from, e.g. continuously reading RSSI values, reading RSSI values at long ranges, converting RSSI values to distances, etc. A parallel process existed with other risks throughout the system, including establishing server-client communication and drawing a complex map using the Google Maps Android API.

*Communication and Scheduling*

As the development phase ramped up, the sponsor and team determined that a more formal schedule was needed for planning and estimation. The team drafted out individual tasks for the next three sprints (rather than just the next sprint), complete with story points. The story point valuations enabled the team to assign an appropriate amount of work for each sprint, and to also shift work from various tracks, i.e. client and control tasks.

The team held a weekly status meeting with the sponsor which primarily focused on the completed and planned work for each team member. These individual reports were self-reported and were typically general as they did not reference specific assigned tasks. After formally drafting out individual tasks, these individual reports referenced these tasks by name and story point values. As a whole, each sponsor meeting involved a sprint status report in which team members reported tasks as completed, underway, not started, or blocked. This color-coded report quickly communicated to the sponsor the current progress of the team as a whole.

### Project Schedule: Planned and Actual

The project was incorporated into two major phases: a research phase and a development phase. Prior to the research phase, a period of several weeks was used for planning, requirements elicitation, and system design. The planning period involved many team organization tasks such as team role selection, meeting time determination, code repository creation, and process tool initialization. Requirements elicitation was conducted over a period of several weeks which involved formulating questions, email Q&A dialogues, and an in-person client interview. A system design period of two weeks followed in which a general architecture design was drafted which demonstrated the various required software and hardware components and connections. This system design was refined several times after being reviewed by the client.

With enough information about the project and system design, it was at this point that a process methodology and project schedule was formed. The research phase was to be conducted over a period of 4-6 weeks in November, concluding with a trade study document and technology selection recommendation in December.

Information regarding the specifics of the development phase would not be available until the research phase was completed and an indoor localization technology was selected and approved. Without this information, a schedule for the development phase could not be drafted until December.

The development phase schedule was organized into two-week sprints in accordance with the adopted Spiral methodology. Sprint details and tasks were determined for the next three sprints only. Planning sprint details beyond six weeks did not prove to be useful as the results and findings of preceding sprints typically affected the prioritization and scheduling of sprints several weeks out.

Below is a table presenting the actual phases and sprints through the project.

|  |  |
| --- | --- |
| **Weeks** | **Goals** |
| 1 - 4 | Requirements Elicitation and System Design |
| 5 - 9 | (Research Phase) Trade Study and Technology Selections |
| 10 | UI Mockups |

|  |  |  |  |
| --- | --- | --- | --- |
| **Weeks** | **Client Positioning** | **Client Map** | **Control** |
| 11 - 12 | N/A | UI Scaffolding | Classes and Schema |
| 13 - 14 | Bluetooth RSSI Readings | Beacons on Map | Map Ingestion |
| 2 - 3 | Beacon Distances | Get Server JSON | Service/UI Integration |
| 4 - 5 | Geometric Trilateration | Get Server JSON | Control Refactoring |
| 6 - 7 | Trilateration via Least-Squares | Generic Map | API Endpoints |
| 8 - 9 | GATT Readings | Custom Map | Cartographer UI |
| 10 - 11 | Map Environment Contribution | Itinerary on Map | Itinerary Creation UI |

|  |  |
| --- | --- |
| **Weeks** | **Goals** |
| 12 - 13 | Integration, Testing, Deployment, and Support |
| 14 | Deployment Assessment |

*Deviations and Adjustments*

Below is the original Fall Semester schedule (post-Requirements Elicitation)

|  |  |  |
| --- | --- | --- |
| **Weeks** | **Client** | **Control** |
| 5 - 6 | Android Skeleton | Technology Selection |
| 7 - 8 | Wi-Fi Triangulation Exploration | - |
| 9 - 10 | Other Technology Exploration | - |
| 10 - 15 | Open Space for Development | Open Space for Development |

It was initially believed that various indoor localization technologies would be explored through implementation in a two-week period. However, without access to the necessary hardware to explore alternative options (Bluetooth, Ultrasound), the schedule was adjusted to reflect a period in which research would be conducted through observations of academic research papers and implemented projects, rather than implementing the project in-house. This shifted the schedule to focus on the Trade Study document deliverable in which all technologies were compared at once, rather than a schedule in which each technology was implemented in a two-week period after which results would be reported.

Below are the original goals for Spring Semester sprint, presented for only those which differed:

|  |  |  |  |
| --- | --- | --- | --- |
| **Weeks** | **Client Positioning** | **Client Map** | **Control** |
| 8 - 9 | Accelerometer, Weighted Least-Squares | - | - |

Before it was discovered that Bluetooth RSSI readings could be read at a period of 10-50 milliseconds by connecting to a beacon’s GATT profile, the plan for improving client positioning involved two additional approaches. The first approach involved using accelerometer magnitude readings to dynamically adjust the amount of averaging used, in order to improve the balance between accuracy and frequency of location updates. The second approach was the addition of weights to the non-linear least squares estimation method for multilateration, which would weight the contribution of each beacon based on its distance, general accuracy, and other possible factors.

When it was discovered in week 8 that connecting to Bluetooth beacon’s GATT profile could improve the scan frequency from 4000 milliseconds to 10 milliseconds, implementing this in the system became the highest priority client positioning task. However, the method proved unstable through the combination of the Android 4.4 SDK, Bluetooth 4.0 specification, and Stick-N-Find Bluetooth beacons (detailed in the System Design section). The goal of implementing GATT readings was dropped at the end of week 9. With only one sprint remaining, implementing accelerometer contributions and weighted least squares estimation was forced to be removed from the schedule, in order to focus on integration between the indoor positioning results and the map.

### System Design

Control



The control architecture can be conceptually divided into three major packages: the application core, the server, and the GUI.

*Core*

The core application, which maintains the system state, adheres to a layered structural architecture - more specifically, a lightweight service-oriented architecture.

The data access layer - the deepest layer of the system - defines the properties of all core data types, including users, teams, maps, and their components. Additionally, it defines an interface for manipulating a repository of these data types. By supporting standard CRUD operations, the data access layer adheres to a data-centric architecture that would be compatible with any standard database. With that said, no requirement of the system is dependent on database persistence - everything can be stored in working memory. The data access layer is instead connected to a local data repository, thereby avoiding the operational cost of file I/O while simultaneously reducing external dependencies.

Above the data access layer is the service layer, which acts as a public interface to the system state. A singleton object within this layer maintains references to data managers, ensuring that any application on the memory stack is manipulating the same system. Additionally, this layer is responsible for verifying that all interactions adhere to the few logical constraints that apply to the system.

*Server*

The server implements a RESTful web interface, allowing remote users to interact with the system over HTTP. This interface maps 1:1 with the functionality of the service layer, ensuring that anything that can be done programmatically through the application core can also be done remotely. This is possible due to the system’s simple data-centric architecture, which REST was specifically designed to support.

To achieve the implementation of the RESTful interface, the server project was built using the ASP.Net Web API 2 framework. Additionally, the Open Web Interface for .NET (OWIN) framework was used to create a simple, embedded server. This decision was made in order to support a non-functional requirement of the system - it should be easy to install, configure, and execute. As an added benefit, the server processes can share the memory stack with the core application; message routing is effectively a non-issue.

Control GUIs

The UIs were constructed using Windows Presentation Foundation elements, which allows for both programs to run on any modern Windows installation, while maintaining native look and feel. The selection of WPF as our UI framework also kept the entire suite highly performant (unlike the GTK+ and Java’s Swing frameworks) because all components necessary for the GUI were loaded with the operating system at bootup.

*Bridge GUI*

The Bridge program in the Control suite handles the runtime logic of the system, including position syncing and team organization. Because the Core layer maintains state, data, and logic, the UI package is simply a collection of XAML files describing the layout and data-binding parameters of the UIs and their respective code-behind files, which contain event listeners.

To support performant display of extremely high number of elements (which ultimately ended up being unnecessary due to other optimizations), a type of drawing canvas out of Microsoft Research called “ZoomableCanvas” was used. Zoomable Canvas supports sparse drawing of many objects to the apparent effect of everything being drawn at once, rather than the usual “top-down” loading effect.

*Cartographer GUI - Benjin*

Cartographer is architecturally less-complex than Bridge due to the lack of a standalone server, however the drawing aspect introduces an entirely new design. Cartographer was adapted from an open-sourced paint program built in WPF, with appropriate modifications to tools functionality and the GUI.

Cartopher is organized into a GUI layer, a utility layer with the library that provides tools, drawing, and canvas functionality, and the common Core layer that shares data-types with Bridge and underlying server. The GUI layer has the standard XAML+code-behind structure, while the Utilities library is divided between tools used by the GUI for selection, drawing, and modification, the shapes themselves, and the graphical representations of the shapes.

Indoor Localization

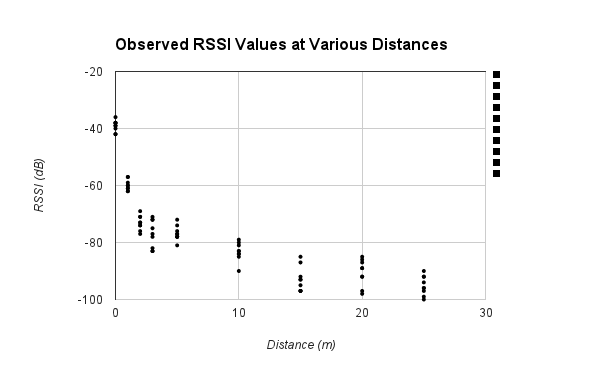
*Overview*

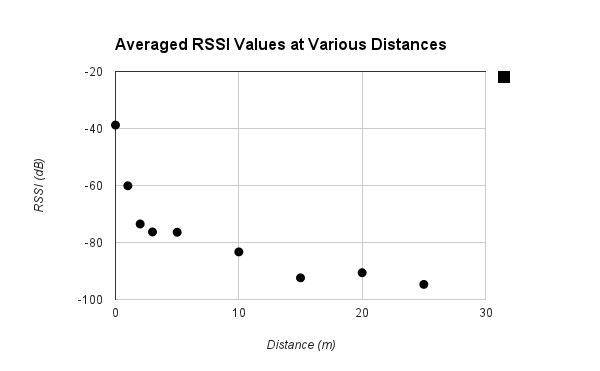
The determination of a position of a device with respect to its indoor environment, i.e. indoor localization, is accomplished through Bluetooth RSSI multilateration. Trilateration is the process of using the distances between the observer and three known points to calculate the position of the observer. Multilateration is the same process, but using more than three known points due to the inaccuracy of the estimated distances between the observer and known points. As the number of known points increases, the precision and accuracy of the position calculated through multilateration increases.

*Bluetooth Received Signal Strength Indication (RSSI) Limitations*

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

Received Signal Strength Indication (RSSI) correlates logarithmically to distance, but various environmental and hardware factors affect the signal propagation. This leads to inaccuracy of distances calculated from a single RSSI reading. Accuracy is improved if the distance is calculated from an average of multiple RSSI readings. However, the effectiveness of this accuracy with the tested hardware diminishes at about 10 meters, to the point at which the difference between RSSI readings at 10 and 20 meters become difficult to discern.



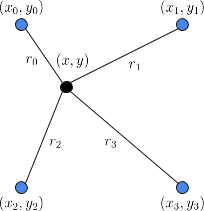


The Android 4.4 SDK and Nexus 7 device tested required about four seconds to acquire RSSI readings from all nearby beacons using a Bluetooth Low Energy (BLE) scan. It was possible to acquire readings from up to four beacons roughly every 10-50 milliseconds after connecting and reading the GATT profile of the known beacons. These GATT connections, however, were unstable and frequently disconnected without the ability to reconnect. These disconnections came without warning or any appropriate error message beside a generic GATT\_ERROR code. Though more frequent RSSI readings would have improved the precision and response time of the indoor localization algorithm by a significant factor, the system could not rely on an unstable and undiagnosable hardware and software system. It is hoped that improvements in hardware and software specifically relating to Bluetooth 4.0, the Android SDK, and Android Bluetooth 4.0 devices will enable this precision in the near future.

Given the limited number of RSSI readings that can be scanned in a short period of time, combined with the prevalence of inaccurate RSSI correlations, a fast averaging method with both outlier filtering and responsiveness needed to be chosen. At first, the mean-shift algorithm was implemented, but such a method required more data points for effectiveness; it offered either accuracy or responsiveness, but not both. A simple sliding median calculation proved to suit these needs most effectively.

*Multilateration Implementation*

Given the positions of and estimated distances between the observer and multiple beacons, the problem to solve is an overdetermined system of distance equations. Various projects have found success with using a non-linear least squares estimation method (NLLSE) method. [1]. The method was implemented using the Gauss-Newton method which explained below:

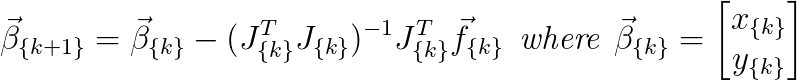


The distance equation of each beacon is used to write the equation to minimize through NLLSE: the difference between the calculated distance and the estimated distance to each beacon.

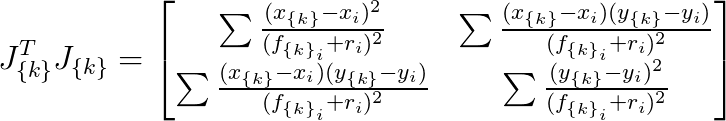
This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

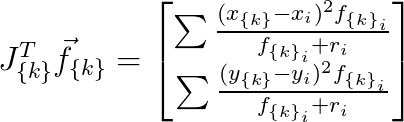
This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

The Gauss-Newton Algorithm involves taking the best guess for the optimal solutionThis is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.and iteratively improving it by using the minimization functionThis is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.and its Jacobian matrix [2].



The derived values of the Jacobian matrix components of the Gauss-Newton Algorithm are given below for the reader’s convenience:





The resulting value This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.is the optimal solution for the location of the observer. The accuracy of the solution solely depends on the accuracy of the estimated distances. Given estimated distances completely free of error, the resulting position would equally be free of error through this method.

### Process and Product Metrics

*Process Metrics*

In the initial project plan, the team identified four process-based metrics that could be employed in order to keep the project on schedule.

1. Hours Worked
2. Team Velocity
3. Bugs Reported/Fixed
4. Average Bug Fix Time

The first two metrics were related in that they measured the team’s availability and rate of development, which were both helpful in sprint planning. Hours were logged into a spreadsheet and posted as a bar graph on the team website. By comparing these values to the estimated story points of completed tasks, the team was able to better estimate the time that would be required to complete a particular task. Much of the team’s work in the first semester was based on research, however, and an accurate measure of velocity could not be established until well into the second semester. Because there would be so little time to benefit from this metric, the team eventually stopped estimating task effort altogether. Instead, an attitude of “it’ll take as long as it has to” was adopted, causing work hours to fluctuate with the difficulty of the assigned task.

The last two metrics dealt with bug fixes, which would have been helpful in measuring the “correctness” of code and the team’s ability to fix errors. Like velocity, these metrics were made obsolete by the constraints of the development schedule. Each component of the system was developed independently, and integration did not occur until the final weeks of development. By the time there was an opportunity to test and discover faults/errors, there was not enough time remaining to warrant the measuring of any metrics.

*Product Metrics*

Several product-based metrics were also identified in the project plan, which could be employed in order to measure the evolving quality of the system.

1. Range of Error (Accuracy)
2. Test Coverage
3. Device Coverage

Of these three metrics, only the first was actually employed. While test coverage was considered to be a desirable metric for this project, the hurriedness of the implementation phase made dedicated unit testing difficult to fit into the schedule. Device coverage, on the other hand, was simply made obsolete by the fact that the team had decided to target the Nexus 7 tablet exclusively.

With regard to the development of the localization algorithm, accuracy was an extremely valuable metric to track. One of the major concerns of the sponsor was the accuracy of this algorithm. Each sprint report included a measure of the current system accuracy, which could be compared with prior measurements in order to show qualitative progress. This was valuable for both the team and the sponsor, as it allowed both parties to come to realistic expectations about the limitations of the system. When that limitation was finally realized, no one was particularly surprised.

*Summary*

In summary, the team did not employ as many metrics as had been originally intended. This was due in large part to the constraints of the development process, which required a large research component and spared relatively little time for implementation and testing. The metrics that were collected did result in a greater visibility of overall development progress to the sponsor.

### Product State at Time of Delivery - Benjin

The delivered product is very nearly feature-complete, only missing “gravy” functionality. It can be (and has been) deployed in the field as is. In the initial spec, three apps were planned as the Control suite - Cartographer, Bridge, and Itinerary Creator - however as the project progressed, it made more sense to combine the latter two, as they’d both need to operate at runtime, rather than beforehand.

The working end-to-end flow begins with Cartographer which constructs new maps and edits existing files. This file is then opened in Bridge, where the user can also manage teams and users. Once available in Bridge, the clients can connect with a username, download the map, and start using the system to locate themselves.

Itineraries are fully-supported in the client and in the backend of Bridge, but complications arising from the choice of drawing technology for Bridge made it more difficult than time allowed to tie the GUI in. Stemming from the same issue, user positions also don’t animate at runtime on Bridge, though they continue to update and broadcast on all clients connected.

While the system updates position very quickly (up to multiple times per second), the accuracy left some to be desired due to the high degree of fluctuation in the signal strength. Due to this unexpected finding, a secondary positioning technique that wasn’t originally planned was added. This addition simply highlighted the nearest beacon to give the user a general sense of location rather than attempting to pinpoint an exact coordinate.

### Project Reflection

*Development Process*

Overall, team members found it difficult to frequently reference formally defined tasks in individual reports to the sponsor. First, the results of previous tasks often created new, unplanned tasks that required team and sponsor approval. One example was the discovery of the faster RSSI scanning GATT method part-way through an unrelated sprint. Second, many tasks were not formal development tasks with assigned story point values. Weekly tests were conducted in order to validate tasks, but those tests were designed on the basis of what tasks were accomplished as well as what indoor space was available for testing. Integration tasks were also not formally assigned nor given story point values. Lastly, individual reports often referenced unrelated work such as preparation for SE deliverables, sponsor deliverables, or development environment configuration. The percentage of time spent weekly on pure development tasks fluctuated week-to-week such that individual reports referenced them sparingly. This made it difficult to measure week-to-week effort contributed to the project by each team member.

These informally referenced non-development tasks should have been defined in an external document (like formal development tasks) in order to aid the team with scheduling. These tasks could have also been estimated in story points as they arose, in order to ensure that each team member was still committing an appropriate amount of work week-to-week.

*Research*

Upon completion of the trade study in the first term, the team was fairly confident that an appropriate technology had been selected that would address the issues of previous attempts. In practice, these Bluetooth beacons were not nearly as accurate or consistent as had been hoped. Despite this, the team remains confident that the best decision was made based on the information available. In truth, no existing technology within the budget of this project has been made to do what is required for real-time positioning. In the team’s defense, this was realized just a few weeks into the second semester. Had this been a project in industry, there would have been talks concerning a re-evaluation of objectives, development constraints, or even potential project termination. With the devices purchased and nothing to lose, however, the team decided to push forward and give it their best shot (while managing expectations).

In hindsight, it is unwise to rely on whitepapers and similar documents for anything beyond theoretical possibility. A project with a research component of this variety - where there is not yet an established method of accomplishing the task - requires extensive testing with real hardware, a controlled environment, and realistic expectations. A team tasked with producing a proof-of-concept positioning algorithm might be successful, as should a team tasked with creating a working system around it. Hoping to accomplish both in parallel is more than likely to fail.

### References

[1] Murphy Jr, William S. "Determination of a position using approximate distances and trilateration." Colorado School of Mines (2007).

[2] Björck, Ake. Numerical methods for least squares problems. Siam, 1996.