Highly Accurate Mobile Device Positioning based on Wi-Fi signals

Team Braintrust

<table>
<thead>
<tr>
<th>Brandon Pastuszek</th>
<th>Sam Gottfried</th>
<th>Joe Impellizzieri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ankur Patel</td>
<td>Jeff Seaman</td>
<td></td>
</tr>
</tbody>
</table>

Project Sponsor
ZOS Communications

Faculty Coach
Professor Raghu Reddy

Project Overview
The Wi-Fi-based device positioning system augments an existing system capable of determining a device's location to the highest accuracy possible. The system will receive sets of a client's visible Wi-Fi devices and their relative signal strengths. With this, it determines the device’s location by comparing the received sets to a series of historically visible Wi-Fi devices and their relative signal strengths at known locations. The location returned will contain a latitude, a longitude and an altitude and/or a meaningful label about the location. There will be a label-only location, such as a room number, when GPS coordinates for the historical readings are unavailable, which is a frequent occurrence inside buildings.

The system will be capable of detecting data sets that differ significantly from other data sets from the same client at the same location, preventing the system from using outlier data when determining location. It will also "learn" and "self-heal" by adding new Wi-Fi devices to the database after they have been consistently detected by clients from a known location and by ignoring in calculations Wi-Fi devices that are consistently no longer detected by clients at a known location.

Basic Requirements
Our system had four functional requirements. First, the system has to establish signal strength baselines via seeding baseline data from a wireless device. Someone is required to go around a room with a wireless device in a variety of patterns in order to seed the room’s wireless signature into the database. Secondly, the system has to establish wi-fi device locations via someone seeding access point information (latitude, longitude, altitude) into the database. This would be used by the weighted averages algorithm to try to locate the wireless device based on the position of the access point and the relative signal strength read by the wireless device. The third requirement is that the system has to locate an end user device. The device sends our system all of the wireless access points that it could see along with relative signal strengths, and our system then attempts to locate the end user device as accurately as possible. Lastly, the system is able to self-heal and self-learn when it encounters new access points.
The system will receive inputs from ZOS’s existing system, with seeding requests being directed to the seeding service and location requests going to the locationing service. Seeding requests will be in an XML format and will contain several readings captured by a wireless device. Readings will include signal strengths, which will each include access points and relative signal strengths. Access points will each include an SSID and a MAC address. If the seeding request is to seed a single access point, then GPS data, including latitude, longitude, and either a height descriptor or altitude, are also included. Location requests will also be in XML format and will contain a reading with all of the access points captured by the end user device. Readings will include signal strengths, which will each include access points and relative signal strengths. Access points will each include an SSID and a MAC address.

If the request is to the locationing service, the system will return either a description (room number or height descriptor) from the neural network or a GPS coordinate from the weighted averages algorithm, depending on which response has the highest confidence rating.

The only human operator characteristics affecting the requirements are the methods of seeding baseline data from an area. Depending on the size of the area and any obstructions, such as walls, the methods to seed baseline data may be very time intensive. ZOS, however, hasn’t placed any restriction on the methods of seeding a room, as long as they are documented.

**Constraints**

There were several constraints on resources, design and implementation. Team members only had 10-15 hours per week to devote to the project and had no budget. Since it’s easier for the project sponsor to scale web servers than database servers, a major design constraint was that database queries should consist only of requests for information and any calculations must be performed on the web server.

The implementation constraints include algorithm and technology. Due to patent issues, the “weighted averages” implementation to determine the GPS location of an end-user device must not use triangulation techniques, must not change the program flow based on the number of access points and must use the inferred location of an access point rather than the true location of an access point. Because the project sponsor uses MySQL, the implementation must be able to interact with a MySQL data source.
Development Process

The process used was a Modified Evolutionary Prototyping. Evolutionary Prototyping is used generally when requirements are unknown. As shown in Figure 1, the process would begin by defining a project scope, then cycling through a process of requirements gathering, design, development and client feedback. Eventually, the cycles end when the client is satisfied. At that point, the system is discarded and the now known requirements are used to develop a production version of the system.

![Traditional Evolutionary Prototyping Diagram](image-url)
For this system, the requirements were known but the implementation was unknown. As shown in Figure 2, the process begins with requirements gathering and high-level design. It then cycles through a process of algorithm design, implementation, sponsor feedback, then refinement or new research. Eventually, the cycles end when the sponsor is satisfied. At that point, the system is released to the project sponsor. For our 10-week time frame, three three-week evolutions (cycles) occurred.

**Modified Evolutionary Prototyping**

![Diagram of Modified Evolutionary Prototyping](image)

Figure 2
This process was approved by but not mandated by the sponsor. They deferred to the team to determine the most efficacious process for the project. Communication with the sponsor occurred toward the end of each evolution. An evolution ended with a demonstration of the developed algorithm to the sponsor. Then, the team and the sponsor discussed whether to improve on the algorithm or cease working on it. Additionally, work for the upcoming evolution was discussed.

Team roles were identified largely by initiative and expertise. For example, Brandon Pastuszek served as the contact for the sponsor, as well as the web service, seeding, database and tools developer. He is consistently on E-mail and has a wide-range of experience with Windows Communication Foundation (WCF) database-backed informational web services. Sam Gottfried served as Quality Assurance (QA) because of his experience on co-op working on QA, as well as QA roles on previous projects.

**Project Schedule: Planned and Actual**

The project schedule was developed to allow a sufficient amount of time for system requirements, specification, initial research, algorithm implementation, testing and delivery. The key activities and milestones were SRS completion, initial research, three evolutions and delivery.

The first quarter schedule was divided into three main components--Planning, Research and High-Level Design. The Planning phase consumed the first five weeks and consisted mainly of formalizing requirements and specifications. The research phase consumed weeks five through seven and consisted mainly of reading and discussing methods of fulfilling the requirements gathered in the previous five weeks. The High-Level Design phase consumer weeks eight and nine and consisted of creating the system architecture and design for the web service and common model objects. During the first quarter, all major milestones were met on-time.

Weeks 10 through 18 were divided into three three-week evolutions. The second quarter estimated vs. actual is shown in Figure 3 below. The first two evolutions ran a week longer each than expected. The third evolution, however, was only delayed by a week because the issues causing the second evolution delay were not dependent upon any features added during evolution three.
The first evolution was expected to complete the environment setup, database seeding functionality and basic neural network functionality. It was expected to run from 2/15 - 3/21 (not including finals week and break week). It actually ran from 2/15-3/28 (a one week delay) due to the underestimation of the time required to setup the environment and database connection.

Evolution two was expected to complete the neural network functionality and the weighted averages functionality. It was expected to run from 3/22 - 4/11 but actually ran 3/29 - 4/25. It began a week late due to complications in the first evolution but also ran a week late due to a slight alteration in the location flow.

The third and final evolution was expected to add self-learning and self-healing functionality to the system. It was expected to run from 4/12 - 5/2 but delays in evolution one caused the start of this evolution to be delayed by one week, consuming 4/19 - 5/2. Much of the functionality of this evolution was anticipated in the design of evolutions one and two, allowing it to still be completed in two weeks instead of three.
System Design

This system, for the most part, is a traditional informational system and utilizes a three-tier architecture (as shown in Figure 4). The three tiers are: Presentation, Logic and Data.

![Diagram of System Design](Image)

**Figure 4**

The Presentation Tier consists of all components visible to external systems. In this system, there is a single web service interface that is visible with several methods for seeding data and locating an end-user. Methods in this interface are no longer than a line or two long because they delegate calls to the appropriate component in the logic tier. The web service interface serves solely to relay messages in and out of the system.

The Logic Tier consists of two major components--algorithms and seeding. Calls from the web service interface to locate an end-user device are sent to a controller that then calls the appropriate algorithms to determine a location based on the provided data. If the web service interface was called to seed data, the seeding data is send to a different controller that validates the data then relays it to a database interface to persist for future use.

The Data Tier consists of the core model objects (AccessPoint, SignalStrength, Reading and Region) as well as an object-oriented interface to retrieve and update data in a relational MySQL database. All database modifications occur in transactions. In the event of an error during database updates or insertions, all changes are rolled back.
There were few design changes during development. One change was to include pre-populated baseline data, when appropriate, with the end-user location data as a convenience to the location algorithms. Other design changes included storing MAC addresses as strings instead of bytes (for convenience) and switching to higher accuracy data types for storage and representation.

**Process and Product Metrics**

We used the following process metrics:
- Number of estimated and actual hours individual team members have worked on tasks
- The teams cumulative average and estimated numbers of hours per week
- Deliverables accomplished on time vs late
- Volatility of requirements

We used the following project metrics:
- Average total time it takes to figure out a device’s location
- Time taken to gather Wi-Fi data for a room
- Average accuracy of device locations with ample amount of seeded historical data

We estimated 810.10 hours worth of work over the course of the project, and we spent 874.10 hours actually working on the project. This means that our estimates were about 93% accurate in estimating our time.

For the cumulative average and estimated number of hours per week, we have provided the following graph for estimated vs actual hours per week (through week 17). The major thing that was noticed was that there were underestimated time during the middle of each quarter and overestimated near the ends of the quarters. There was also a lot more time spent on the project during implementation than during any other time.
For deliverables on time vs late, we had 31 deliverables over two quarters, 5 of which were late. This means that only 16% of our deliverables were late. When they were late, they were usually only a week or so late and didn’t impact the project too much.

We did very well with requirements volatility. There was only 1 requirement change in 20 weeks of project time. The requirements change was simply being able to seed from a file, which only took about an hour and hardly affected the project.

In terms of speed, we can locate an end user device in under a second. To seed a room, we go around the perimeter with a wireless laptop and record 20 seconds, take a step, and then repeat until we have gone around the room 1 complete time. This takes about 6 minutes for a team room. We also leave the laptop in the center of the room and collect data for 5 minutes. The results from this combination have led to a pretty high level of accuracy as shown in the following graph.
Product State at Time of Delivery

At the end of this project, we have a system that fulfills the three major requirements of the project (the fourth, being self learning and self healing, wasn’t as high of a priority). We can seed baseline data from a wireless computer using the inSSIDer application provided by ZOS. We can also add GPS data to the access point in the database. We can then stand in a random spot in a room and send a reading to the system and expect a location back from the system that is highly accurate.

Both the neural network and weighted averages systems that we originally committed to are done and work correctly, and our design permits ZOS to add in any other algorithm solutions if they later decide to do so.

The only feature that was added that wasn’t planned was the ability to seed from a file stored locally on the server. A .gpx file with baseline data can be placed on the server or in a remote location that can be accessed from the server, and a request can be placed to the server with the location of the file. The file will then be parsed and seeded into the database for use by the algorithms.

Project Reflection

Our meetings with ZOS were mostly done via Skype and GoToMeeting. This was very convenient for both ZOS and the team, as neither had to spend time traveling to ZOS’s location or to RIT. We found our meetings were very effective remotely and we received positive feedback from them regarding this.

The major thing that we would have done if we had more time would have been to collect data from more rooms with different sizes. Our algorithms are very effective in the SE team rooms, but it’s hard to tell how effective they would be in a room with a different size or with more obstructions.