Scotty Maps

Jeff Chen, Ifeanyi Ene, Weelie Guo

Department of Electrical and Computer Engineering, Carnegie Mellon University

Abstract—**Scotty Maps is a system capable of providing highly accurate indoor localization and navigation services to students at Carnegie Mellon University. This project utilizes a network of ultrawideband transceivers to localize the user within academic buildings on campus. Our web application displays the user's location and provides navigational instructions to the best path to the student's destination. We have created a novel, inexpensive, and scalable system offering localization accuracy of less than 0.2 meters, a frequency of 10.1 Hz, and a total maximum delay of 840 ms.**

Index Terms—**A* Search, IMU, Indoor Localization, Mapping, Multilateration, Navigation Systems, Nelder-Mead, Two-Way Ranging, Ultrawideband**

I. Introduction

Navigation systems have become a ubiquitous part of our lives, revolutionizing how we move in the world. GPS can be leveraged to provide directions in nearly any outdoor environment, allowing navigation systems to offer unparalleled convenience and precision. However, navigational systems do not work well indoors due to GPS signals being stopped by walls. Hence, an indoor navigation system would take our daily navigational needs even further, providing seamless guidance within complicated buildings.

For students, the benefit of indoor navigational systems is especially pronounced. New students and faculty at CMU may have particularly significant difficulties in finding rooms in the academic buildings on campus. Trying to find a room in an unfamiliar building can both stress students and waste their time. An indoor navigation system would assist students in confidently finding their destinations, minimizing the frustrations that come along with getting lost in a new place.

Scotty Maps aims to solve the problem of students becoming lost in CMU buildings by creating a localization system and navigation app to guide students directly to their desired rooms. All students will need is a tag they can put in their backpacks. Then, once they are in a building, they can use their phone to access our web application, which will provide them with directions to get to their destination.

Indoor navigation systems are not wholly a new idea. Several companies have solutions regarding implementing indoor localization systems [9]. However, these systems can be complicated to deploy and maintain, requiring specialized IT support that can make them prohibitively expensive for a college campus. Instead, Scotty Maps will make a solution tailored specifically to the unique needs of the CMU campus by creating an affordable and scalable solution, focusing primarily on navigation within academic buildings to streamline the student experience.

II. Use-Case Requirements

Scotty Maps is designed primarily to achieve excellent localization accuracy, along with providing highly useful directions for the user to follow. However, along with these goals, this project also makes a priority of being accessible, and unobtrusive, while also protecting the privacy rights of students who use the device.

Our project has several use case requirements, the most important being highly accurate localization of up to an accuracy of one meter. One meter is typically the width of most doorways, and we believe that this localization resolution is sufficient for preventing students from getting lost in buildings. Then, we wanted the localization system to function responsively, having an update frequency for the student's location of at least 2 Hz. Assuming the student is moving at a walking pace, this frequency is sufficient to accurately maintain the real-time location of the student, as students will move less than a meter in the 500 ms.

There are also several physical characteristics of what the final product should consist of. We wanted the portable tag the user is carrying to have a battery life of at least 4 hours, as we think 4 hours is approximately the maximum amount of time we expect a student might spend walking around indoors on a given day. Next, the size of the tag should not be excessively large either, as we would like the tag to fit in a user's backpack. Hence, the device size should be less than 2 liters, the size of a laptop. Additionally, the overall price of the tag should not be too expensive for students to purchase, at a cost of less than $100. Finally, the cost of installing the localization system inside a building should be reasonable, at around $50 per hallway corner, or approximately $300 for a floor of a typical building. Extrapolating this to a typical four-story building gives a cost of $1200 per building, or approximately $50,000 for all the ~40 academic buildings on the Carnegie Mellon campus.

Due to the system constantly tracking the students as they are moving around indoors, respecting the privacy rights of the students becomes paramount. Security measures such as a login system should be implemented such that bad actors will be unable to access the system and view the locations of other students. Furthermore, our system should avoid storing historical data of student movements, as it should only use relevant information on the current location of the student to provide navigation information.

III. Architecture and/or Principle of Operation

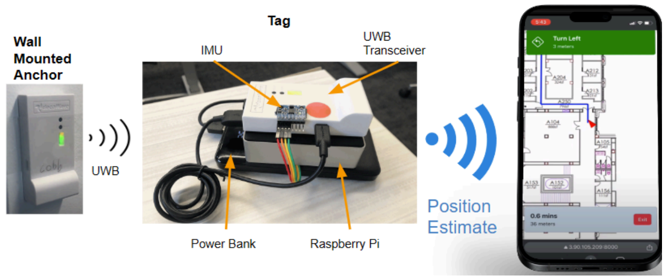


Fig. 1. A high-level view of our system architecture for finding the user's location and providing them with directions.

As seen in Figure 1, the high-level overview of our system consists of a network of UWB transceivers, which function as "anchors". These anchors are fixed in various locations in a building. The student carries around a "tag". Once the tag is in the vicinity of the anchors, the tag will communicate with the anchors to figure out the distances between it and each of the anchors. With the distances, it will then run localization algorithms to determine the device's location. This information is communicated over Wi-Fi to our server, which hosts our web application. Users can utilize a browser on their phones to access the server and view the webapp, which displays their current location in the building, as well as the directions they need to follow to get to their destination.

Our system block diagram is shown in Figure 2. Only a single anchor is displayed at the left, though we are able to have 11 total, fixed in various positions around a building for better connectivity throughout the area of the building. These anchors are constantly listening to UWB packets, sent by the tag. Upon receiving these packets, the anchors respond to initiate a communication protocol with UWB to determine the distance between it and the tag.

A number of changes have been introduced since our design report. The trilateration algorithm is replaced by multilateration with Nelder-Mead algorithms to get higher accuracy and more stability. The connection between DWM1001 and RPi 4 is changed from SPI to UART for convenience and faster reading of inputs. We are now using a WebSocket connection between the tag and the server.

The tag has the functionality to communicate with all of the anchors present and within range in our network. It runs the localization algorithms necessary for localizing our device. Additionally, there is also an IMU attached to the tag, which allows us to find the user's orientation, so we can know which direction they are currently facing within the building. The IMU can also be leveraged to help provide an estimation of the direction the user is headed in, which can be used in tandem with our UWB localization to better refine our location. The tag can send all the necessary localization information over WebSockets to our server.

Our server consists of a Django server that is hosted on AWS. The server contains a Django web application, which is the primary form of contact between the user and the localization device. The user is able to input locations they wish to go to via the webapp, and then the webapp will

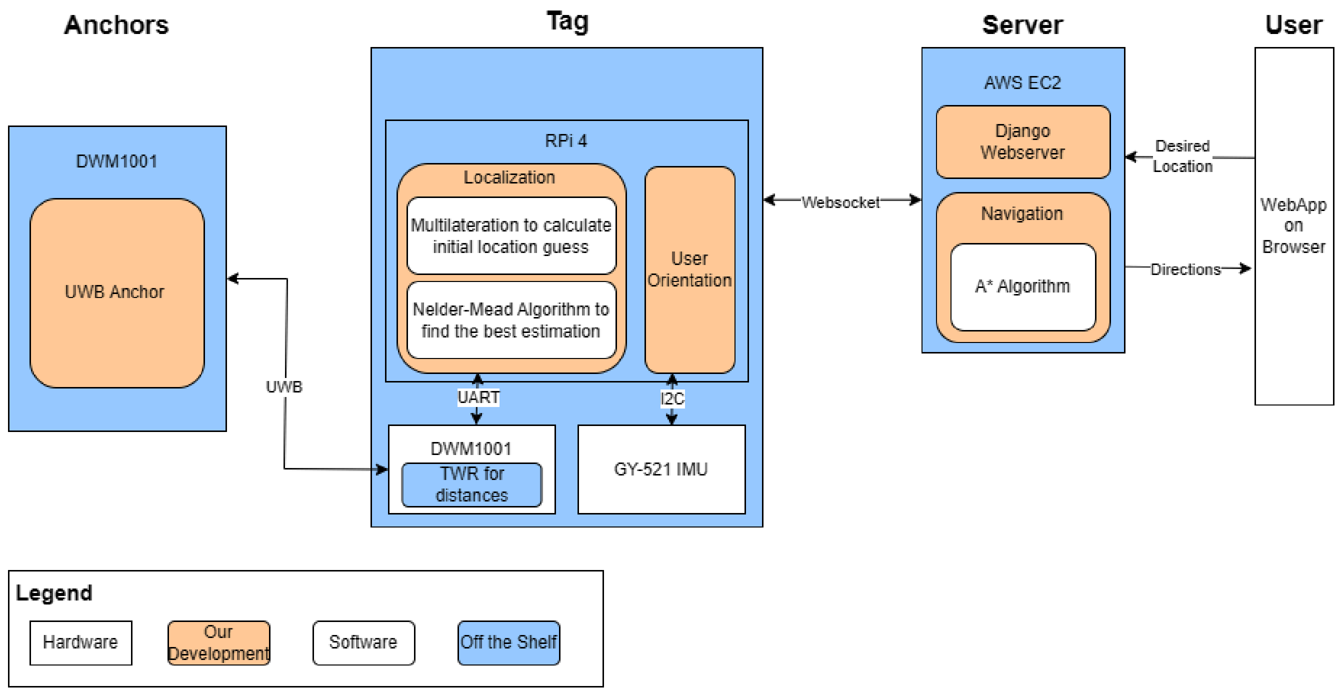


Fig. 2. System block diagram showing the overall layout of our anchors, tag, server, and user interface.

use that information to find navigation instructions the user can use to get to their destination. While in navigation, the user is able to input feedback into the system by their current position (provided to the server by the tag). Our server is able to keep track of the location of the user and display that information for the user to see in the web application.

Throughout our project, we applied many principles of engineering. Algorithms were very important to accomplish what we wanted to do, such as multilateration for localization purposes and A* pathfinding for navigational purposes. Our system architecture was also very important to ensure that the server was able to interface with both the tag and a user sending it a large variety of inputs. Finally, we focused on improving the user experience, making sure our navigation provided intuitive directions and clear guidance on where the user should go.

We also leveraged several scientific principles. As we were dealing with UWB sensors, we needed at least an elementary understanding of how they would travel through the air, such as how their electromagnetism might affect how they travel, reflect, diffract, or interfere whilst inside of a building. Additionally, an understanding of computer science was exercised. Being able to host a server with low latency to both the tag and the user was crucial in decreasing the latency the user experienced with their tag device.

Finally, there were also several principles of mathematics used in our final project, such as geometry, linear algebra, statistics, and calculus. Our algorithms strongly consider geometry and trigonometry for our multilateration to function correctly. The Nelder-Mead algorithm is purely geometric. It utilizes the coordinates information and iteratively shrinks the geometry formed by vertices to get the final estimation of locations. We use a significant amount of linear algebra, such as having a coordinate system to be able to successfully map the user, as well as using the least squares estimation to minimize our error. In terms of statistics, we use several filtering techniques such as low pass filters to help decrease localization inaccuracies from imprecision in our UWB distances. Finally, calculus is used for our IMU, where we use integration to get the cumulative rotation found by our gyroscope.

IV. Design Requirements

A. *Ultra-Wideband Accuracy*

The most important use case requirement for our design to achieve is high localization accuracy. To achieve an average accuracy of less than one meter, our distance measurements need to be even more accurate than this. Under the assumption we are using ultrawideband sensors and computing location with multilateration in a hallway, small errors can lead to much larger ones due to the small, acute angles formed by the anchors and our position. Calculating the error of a multilateration is not straightforward due to the many factors that impact the result. However, we can simulate the impact of errors, having chosen the multilateration algorithm (as detailed in Section VI.C). Assuming a user is standing anywhere in a typical 25-meter x 2-meter hallway, we find that a maximum error of 0.23 meters in a distance measurement causes the resulting localization accuracy to change by approximately one meter. Hence, we would like a distance measuring accuracy of 0.23 meters.

To achieve a 0.23-meter distance measuring accuracy, we use UWB and the TWR TOA protocol to determine the distances between our UWB devices. This protocol relies upon a tag sending a message to an anchor before the anchor responds with a reply. The DW1000 chip on the UWB board is then able to calculate the time of flight (TOF) using (1), where t_1 is the start time of the initiator, t_2 is the time at which the initiator receives the response from the responder, and t_{reply} is the response time from the sensor.

$$TOF = \frac{t_2 - t_1 - t_{reply}}{2} \quad (1)$$

The propagation speed of UWB signals through air is equal to the speed of light. Hence, we can solve for the resolution the DW1000 should have to be able to measure differences in location by using (2).

$$Distance = c\frac{(t_2 - t_1 - t_{reply})}{2} \quad (2)$$

We can assume that t_{reply} is a fixed constant based on the delay of the sum of the anchor's antenna and the anchor's processing speed. While solving for the propagation time of the signal, this can be safely ignored. By using a distance of 1 meter and a speed of light of 3.0*10^8, we find that the DW1000 should have a time resolution of at least 6.67 nanoseconds to be able to differentiate distances of one meter. Unsurprisingly for a device with accuracy claims of 0.1 meters, this is the case, as the DW1000 has a time resolution of approximately 15.65 picoseconds [1], implying that our UWB boards should be capable of reaching the accuracy necessary for our application.

B. *Responsive Tracking*

We would like to be able to sample the user's location at a frequency of at least 2 Hz to be able to capture their location in real-time. To do this, our tag needs to obtain distances from the anchors as well as run the localization algorithms within 500 ms. We considered it likely the communication protocols would be the most time-consuming portion. First, we need to be able to communicate with multiple UWB devices simultaneously. To obtain accurate distances between the tags and the anchors, several samples of distances should be obtained between the same tag and anchor. The TWR TOA protocol takes approximately 1.2 ms between a tag and anchor when the tag is able to discover an anchor [1]. When considering a larger network of at least 10 anchors and assuming the tag will sample each anchor at least 5 times pers second, we note that the overall localization time is reasonably quick at less than 100 ms and should not bottleneck our overall localization speed. The rest of the computation relies on our localization algorithms. These will be run on an RPi 4, which should be sufficiently powerful to run the algorithms well within the required time. Altogether, these results gave us confidence that we should be able to meet the timing requirements for our project.

C. Power Requirements

Our tag should have a battery life of at least 4 hours, as we assume that this is approximately the maximum time a student will spend walking indoors in a given day. Assuming the power consumption of the DWM-1001 development board is approximately 82 mA at 3.5 V when it is sending data, and noting that the power consumption of the RPi is around 600 mA at 5 V, we can use (3) to estimate our average power consumption when constantly transmitting and processing data is around 2.8 W.

$$p = iv \tag{3}$$

From these results, we find that our battery supply for the tag, to last for up to 4 hours of operation, should be at least 12 Wh. Any power supply with approximately 2.5 A of capacity at 5 V reaches this desired capacity should be sufficient for our device to maintain a full-day battery life.

D. Range of Ultrawideband Transceivers

We would like the installation of our UWB anchors to not be excessively expensive, at a cost of approximately $50 for a corner of a hallway. To do this, our transceivers should span the distance of most hallways (with longer hallways such as those in Wean Hall being divided into two). Given this constraint, we note that many hallways in academic buildings at CMU have hallways that span approximately 25 meters before meeting an "intersection" or a wall. The lengths of hallways in academic buildings are all different and there are many exceptions to this rule; however, from our experiences measuring the lengths of various hallways, we believed this distance to be typically sufficient. Hence, to keep the costs of our Ultra-wideband receivers on the lower end, we would like our UWB transceivers to have 25 meters of range while in an indoor environment–the advertised distances of UWB transceivers are typically greater than this, though the advertised distances are usually from an idealized outdoor environment.

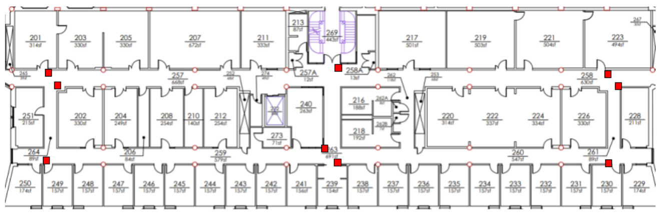


Fig. 3. Example locations of anchors, in red boxes, to map out floor 2 of ReH

Figure 3 displays an example of how anchors could be installed inside floor two of the Roberts Engineering Hall to map out the major walking areas, such that we have at least three anchors with direct line of sight in each hallway for multilateration (hence the occasional doubling of anchors). Despite the horizontal hallways spanning further than 25 meters, the edges of the main hallways are broken up by vertical hallways into lengths of less than 25 meters. By installing sets of anchors in these corners, we can map out the floor.

V. Design Trade Studies

A. Wireless Technology

One of the most important areas of contention when we came up with a solution was the technology we wanted to use for indoor localization. The choice for selecting a technology was motivated by our use case requirements, as we wanted to create a system with highly accurate tracking, while still keeping our system costs relatively low. The main wireless technologies we researched were Wi-Fi, BLE, and UWB, as these were all involved in previous research in indoor localization. We wanted to focus primarily on determining a technology's localization accuracy, then the range of the devices, before also considering the propensity of the signals to be subject to interference, and finally, cost.

Table I. Wireless Technologies Tradeoffs

Technology	Characteristics		
	Accuracy	*Indoor Range*	*Infrastructure Price*
Wi-Fi	3 m	50 m	$0
BLE 5.1	1 m	100 m	$42
UWB	0.3 m	100 m	$75

Table 1 summarizes some of our general findings regarding the characteristics of the technologies we were interested in exploring. We found the accuracy of technologies utilizing results from previous projects and research. Wi-Fi has the worst accuracy at approximately 3 meters [2], while Bluetooth has an accuracy of around 1 meter [3], with UWB having the highest accuracy of around 0.3 meters [4]. The indoor range of these technologies varies slightly with utilizing different devices or the assumption of the number of obstructions in an indoor environment, though we found BLE 5.1 and UWB to typically have higher ranges than Wi-Fi. Finally, we estimated the infrastructure prices of using these devices along a single 50-meter-long hallway, assuming we needed 3 anchors for BLE and UWB (to use multilateration) and 6 anchors for Wi-Fi (due to the difference in range).

Initially, we assumed Wi-Fi would be a good technology to use. Due to the ubiquitous nature of the internet, Wireless Access Points (WAPs) are already built into the infrastructure of buildings. WAPs typically have a high range (up to 50 meters indoors) and are also typically installed in a fashion such that there is some overlap between each of them, allowing us to triangulate the user. Utilizing these WAPs for distance localization could reduce the cost of implementing our system in a building. Additionally, there was a lot of published literature regarding utilizing Wi-Fi in indoor localization. However, previous research has shown the accuracy of Wi-Fi systems to be lacking accuracy; typically they struggle to become accurate to within 3 meters [2]. Wi-Fi inherently falls short because the technology was not built to be an indoor localization technology. Wi-Fi finds the distance between devices with RSSI, which lacks resolution, as well as faltering in the face of obstructions, such as walls, rendering Wi-Fi subject to interference.

BLE was another technology we researched. In terms of localization, it functions quite similarly with respect to Wi-Fi, also utilizing RSSI for distance calculation. Our research found that the newer BLE 5.1 version was capable of localization accuracy to around a meter with a direct line of sight, which is a level of localization accuracy that falls slightly short of our goal of less than one meter. The range of BLE 5.1 is greater than that of Wi-Fi at around 100 meters; however, a localization system would require the installation of a large number of these devices. Assuming each device consists of a transceiver and a microcontroller, we find that each device costs around $14, or a cost of $42 for localization in a hallway. Finally, due to Bluetooth also relying on RSSI, it also suffers from being prone to interference from obstructions.

The third technology we researched was UWB. UWB seems to have significant advantages in many aspects that make it more accurate in localization. Importantly, UWB is different from narrow-band systems such as Wi-Fi and Bluetooth as its signals have better multipath resolution, such that UWB signals remain distinct while narrowband signals might overlap in a multipath environment [4]. Hence, when it comes to determining distance, UWB can rely on protocols other than RSSI, such as Time-of-Arrival, which ultimately leads to a significantly higher accuracy of less than 0.3 meters, which is sufficient for our localization system. Our investigation of the range of UWB transceivers showed they were capable of a range of around 100 meters, high enough to cover longer hallways. Additionally, due to UWB being transmitted over multiple frequency bands, it is also less prone to interference than either Wi-Fi or BLE (both of which transmit on the 2.4 GHz band). However, the cost of a UWB system appears to be an issue, primarily due to the newer nature of the technology, and fewer options for transceivers. These devices have a per-device cost of around $25 [10], which makes localization within a hallway to be around $75. Although the prices for UWB devices are higher than other technologies, the advantages they have in accuracy and providing solid range make them the suitable technology for this project.

B. Ultrawideband Board

The choice of which UWB chip would best fit our application ended up being a rather simple choice due to the limited options available for cost-effective UWB solutions. Some manufacturers such as Microchip Technology only sell UWB chips. However, incorporating an UWB into a PCB was beyond our expertise, so we looked for developer boards with a UWB chip and antenna instead.

We examined UWB developer boards from NXP, SPARK Microsystems, and Qorvo, with the results summarized in Table 2. Boards from NXP and SPARK were relatively expensive, costing several hundred dollars for each board, so we opted towards Qorvo, which happens to have significantly cheaper developer boards.

Table II. UWB Development Board Cost

Device	Unit Price
SPARK SR1010	$999
NXP NCJ29D5	$129
Custom Qorvo DWM1000 PCB	$27
Qorvo DWM1001-Dev	$25

a.

Qorvo sells two varieties of chips, the DW1000 and the DW3000. The latter is the newer, more expensive variant offering slightly better energy efficiency, though qualities such as the range or accuracy are rated as being the same as the DW1000. Qorvo sells multiple packages of their DW1000 chip. One is the DWM100, which simply consists of a PCB with a DW1000 as well as an UWB antenna. We looked into designing PCBs for housing a DWM1000, as well as a microcontroller to control the transceiver, and we found that this would have a unit price of around $27. However, Qorvo also sells DWM1001-Dev boards that encompass the functionality our custom PCB solution would resolve at a lower price of $25 per device. Due to these reasons, we ultimately settled on using the DWM1001-Dev development kits.

C. Choice of Gateway Device

The DWM1001-Dev board we selected as our UWB transceivers is controlled by an nrf52832 microcontroller, which does not inherently have Wi-Fi connectivity. Therefore, we need a way to convert the device into a gateway for the tag to post to the server to update the state of the user's position. Two options we explored to include this functionality are the ESP8266 and the RPi 4. The ESP8266 has the benefit of being a cheaper device at around $8, compared to the RPi 4's higher price of $62. The ESP8266 also has a smaller power consumption while being smaller than the RPi. However, the RPi 4 has the primary advantage of having a massively higher amount of computing power available. With the higher compute, we were more confident the device could handle running our localization algorithms in the required amount of time in tandem with the extra processing needed to acquire data from our IMU and make location predictions with it.

D. Handling Multiple Transceivers

There will be situations where the tag can receive signals from more than three different anchors. To deal with these situations, there are two different approaches. One is to change our algorithm from trilateration to multilateration, and the other one is to pick the anchors that are closest to the tag. The advantage of multilateral positioning is that it will create more precise position estimates, due to the errors from many anchors destructively interfering with each other. However, we reasoned that the distances would not be accurate enough for anchors that are further away from the tag, especially in the case where the signal from the anchor is blocked by a wall. Ultimately, we decided to go with multilateration, as no matter what errors build up across all the anchor networks, they tend to cancel each other out when combined to estimate the position.

E. Frontend Choice

After deciding on using UWB as our localization technology and determining the components that would compose the device, another consideration was the most optimal frontend technology that could be used to communicate information effectively to the user. On this front, there were several viable options. We could design an app for mobile devices, a Python application, or a Django web application.

Implementing a mobile application is somewhat of a logical option, due to the plethora of other navigation apps that many people already use on their phones. Hence, we agreed this option would produce an application that would be fairly intuitive for users to use. However, the members of our group have fairly limited experience with mobile development, and we decided it would be excessively challenging to pursue this approach. Another option was to take advantage of the computing power of the RPi to host a graphical Python program. By adding a screen, we would be able to display directions while running the localization algorithms, which would keep all of the computing localized on one device. However, this option has the issue of leading to a worse user experience, as we would need to also come up with a user interface and work out an optimal method for considering user input.

We ultimately chose to use a Django webserver due to our previous expertise in developing Django apps. Django applications tend to be particularly flexible, allowing us to design an application that could be used on any mobile device or laptop. By having the webserver handle requests, we would be able to easily have a sufficient amount of connectivity between the user interface and their physical tracker.

VI. System Implementation

A. Anchor Implementation

From our block diagram in Figure 2, the anchors of our design consist of the DWM1001-Dev boards. These boards have a DWM1000 module which has functionality as the UWB transceiver. We configured these boards to function as anchors, such that they will always be listening for UWB messages, and be ready at any time to respond to requests from the tag to initiate an exchange of Two-Way Ranging (TWR) Time of Arrival (TOA) data to calculate the distance between the tag and the anchor. We set them with specific IDs such that the tag and the anchors are in the same system network. Thus, the tag is able to figure out the anchors that it can communicate with. Each of the anchors Li-Po battery powering it (though in a proper deployment it would probably be better to use stationary wall power), with the device itself being mounted on the wall with double-sided adhesive. Placing the anchor approximately five feet from the floor yielded the best range and accuracy, due to it being the height that a person might hold it at.

B. Tag Implementation

The tag also consists of a DWM1001-Dev board, except it is also considered a gateway device. The DWM1001-Dev board is controlled by an nrf52832 microcontroller, which does not have built-in Wi-Fi, hence, we connected the development board to a RPi 4 over USB. The RPi constantly communicates over USB to request readings from DWM1001-Dev. This DWM1000 chip runs multiple threads, each one performing Two-Way Ranging (TWR) with an anchor to get the distance between itself and the anchor. Once it has established these distances between itself and the various anchors, it then transfers the distance data to RPi through serial port.

The RPi has multiple threads as well; one collects the data by reading for the most recent serial input from the DWM1001, another runs localization algorithms such as multilateration, and the main thread runs in a loop to send positional information to the server. The tag is also connected to an IMU. By communicating with the IMU over SPI, the tag is able to use algorithms to find the orientation of the user. Then, the RPi sends all of the localization information to the webserver by utilizing the Python WebSockets library to be able to form a connection with the channel on the server. The tag device is powered by a 5V power bank, plugged into the RPi, while the DWM1001-Dev board receives power from its USB connection to the RPi.

C. Localization Algorithms Implementation

The most important algorithm that is used for localization is the multilateration algorithm with four anchors. Multilateration is the use of distances (or "ranges") for determining the unknown position coordinates of a point of interest [5]. Based on measurement of the two-way ranging (TWRs) between anchors and the tag, we could get the distances between the tag and the surrounding anchors. A point (x, y) on the Cartesian plane lies on a circle of radius r_1 centered at (c_x, c_y) if and only if is a solution to (4).

$$(x - c_x)^2 + (y - c_y)^2 = r_1^2 \tag{4}$$

With the same reasoning, we can derive equations for the circles generated by the three different anchors. Since each one has its own position, we can express their positions with $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$. The radius of the circles the form can be expressed as r_1, r_2, r_3, r_4. The problem of multilateration is solved mathematically by finding the point $P = (x, y)$ that simultaneously satisfies these equations of these circles.

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2 \tag{5}$$

$$(x - x_2)^2 + (y - y_2)^2 = r_2^2 \tag{6}$$

$$(x - x_3)^2 + (y - y_3)^2 = r_3^2 \tag{7}$$

$$(x - x_4)^2 + (y - y_4)^2 = r_4^2 \quad (8)$$

Equations (5), (6), (7), and (8) are based on the assumptions that the circles that the anchors generate meet at a single point, and the signals of the anchors can form perfect circles. However, in real life, there could always be cases when the circles don't meet at a single point, and the signals can form perfect circles. Thus, we need to find the point that best approximates the tag position. Given a position X, we can estimate how well it replaces the ideal precise location P. We can do this simply by calculating its distance from each anchor. If those distances perfectly match with their respective distances, then X is indeed P. The more X deviates from these distances, the further it is deviated from P. In this case, we can treat multilateration as an optimization problem. The goal is to find the point X that minimizes the error from X to P. Suppose the deviation from the ideal distances d_1, d_2, d_3, d_4 with the actual distances between the tag and the anchors are e_1, e_2, e_3, e_4, and the anchor's locations are L_1, L_2, L_3, L_4. We need to find the position X that minimize the following equation sets.

$$e1 = d_1 - dist(X - L_1) \quad (9)$$

$$e2 = d_2 - dist(X - L_2) \quad (10)$$

$$e3 = d_3 - dist(X - L_3) \quad (11)$$

$$e4 = d_4 - dist(X - L_4) \quad (12)$$

We can use the mean squared error to express the overall deviations from the point X to the ideal location of the current position.

$$\frac{\Sigma(d_i - dist(X - L_i))^2}{N} \quad (13)$$

To minimize the error, we used the Nelder-Mead algorithm [6] to get the final estimation X of the actual location. To make the computation process faster, we computed an initial guess based on the locations of the anchors L_i and the distances from the anchors to the tag d_i. Let the sum of the distances be S.

$$S = \Sigma d_i \quad (14)$$

To facilitate the computation process, an initial guess of the estimated location is calculated. The initial guess of our position X_0 is calculated by (15).

$$X_0 = \Sigma \frac{(N-1)S}{S - d_i} L_i \quad (15)$$

The Nelder-Mead algorithm takes the initial guess of our position along with the positions of the anchors and the distances between the anchors and the tag to get the final estimation of the location.

D. *Data Filtering Implementation*

The distance measurements that come from the tag can be imprecise, varying by up to 1-2 m while not moving. This can cause our distance prediction to be substantially incorrect (off by 8-9 m at times). To combat this, we implemented a low pass filter to discard high frequency fluctuations in our input data. The filter works by taking the last measurement of the user's distance from a particular anchor, and calculating how far away from that radius the user could be at the current moment, based on how long ago this last valid measurement was taken. We do this by assuming that the radius can change no faster than 2 m/s (slightly above average walking speed). So our condition for a valid measurement comes to (16), where r' is the new distance reading, r is the old one, and dt is the time since the last valid reading..

$$abs(r' - r) < 2\, m/s \times \Delta t \quad (16)$$

If the new distance measurement doesn't satisfy this constraint, we declare it invalid and discard it. After implementing this change, we did see considerable improvement in the accuracy of the system. However, sometimes even small errors (of ~1 m) in each of the anchor's readings were enough to throw off our final position estimate by up to 4-5 m. To remedy this, we implemented the same filtering for the position estimate itself, and this greatly increased the accuracy of the system, bringing errors down to ~1 m.

E. *Orientation Estimation Implementation*

For our orientation display, we need only the angle of the user about the gravity vector (like the angle given by a compass). However, while our IMU has an accelerometer with the ability to fairly accurately estimate its angles of tilt along the other two axes of rotation, it cannot return its rotation about the gravity vector, as the accelerometer works by reporting the direction of the gravity vector itself, which of course doesn't change as one rotates in the "compass plane". Our IMU also has a gyroscope that can retrieve all 3 axes of rotation, but it is only able to determine the relative rotation from a known starting point. For this reason, we employed a sensor fusion to maintain a quickly updated estimate of the user's orientation. Every time we receive a new position estimate that is sufficiently far away (2 m) from the last one used to estimate our orientation, we take the angle of the vector from the previous position estimate to the current one. This angle serves as our ground truth orientation estimate. This update is slow, however. To fill in the downtime, we employ the IMU, rotating our ground truth by our relative rotation around the gravity vector given by the accelerometer, until a new ground truth estimate arrives.

F. *Web Application Implementation*

The web application is built using Django and is hosted on EC2. It powers the web application as well as the API endpoints necessary to refresh data. The user interfaces with the webserver, accessing the webapp using the browser on their phone or laptop. When the user first opens the browser, they will be directed to create an account. The account creation process associates a private tag identifier (similar to a MAC address) with the user to create a private connection between the user and the tag device. The home page of the application displays a map centered over the CMU campus, along with some pins the user can click to access for further information of each of the academic buildings. The user will be able to pan around the map and view different portions of the campus as necessary. An image of this screen is displayed in Figure 4.

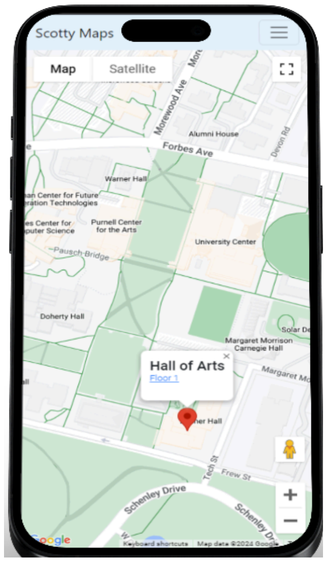


Fig. 4. The home screen of our application.

Once the student arrives in a building, they can select a pin of the building they would like to navigate in, which will cause the web application to open up a map of the building. A mockup of this view is displayed in Figure 5. The student is able to enter the room number they would like to go to as their destination. The webserver processes this information and runs the navigation algorithm, using the A* pathfinding algorithm for finding the closest path between the student and their destination, as well as providing them with written directions to aid them with their travel. As the student moves, their position (as displayed on the webapp) is constantly updated by using Javascript and occasional AJAX calls to the webserver to ask for data, showing the student's current progress. Javascript will also be used to draw the desired path the student should take to move forward.

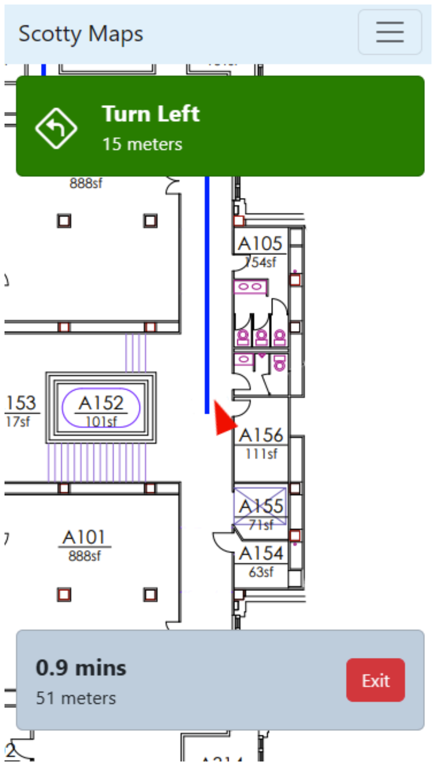


Fig. 5. The navigation view of our application.

G. *Communication From Tag to Server*

Our original idea was to include an API endpoint for the tag to constantly post updates to the server for the user's position. This was a very simple solution that relied only on Django. The performance of getting a distance update was on average within our design requirements of around 250 ms. However, the heavy load of packets occasionally led to slowdowns.

We eventually opted towards using WebSockets, utilizing the Django Channels library, and offloading much of the server load from handling location updates to a separate Redis server. Through our testing, the change to WebSockets was worth the effort; we found our latency to drop to around 67 ms, as well as the occasional slowdowns to cease.

H. *Mapping Process*

The mapping process was crucial to obtain measurements of each floor plan useful in testing so that we could synchronize localization estimates from the localization system with the actual position on a map in the user's browser. To this end, we needed to repeat this mapping process for each new area we wanted to test the localization system, as well for the final demonstration in the Wiegand Gym, so tools and protocols were developed to make this process easier and less prone to errors.

Mapping begins with using a laser measuring tool to find the lengths and widths of the hallways of a floor. A (0,0) point is designated in the corner of the building, and the anchors are placed relatively to this position. One of our anchors is configured as an initiator so that it can try communicating with other anchors to form a network. Our testing of the maximum range of anchors leads to us placing them in increments of 25 meters.

Our results from our physical mapping process is compared to the map of the building (obtained from CMU's online repository of building plans [6]), where we find the pixel locations of the origin and the offset. After demarcating the walkable areas by marking the starting and ending positions of hallways, a script converts the image into a two-dimensional graph with boolean values representing walkable areas and stores it as a JSON file.

Finally, the pixel coordinates of each room are marked in a spreadsheet. Once this is done, all of the collected information is exported into our server's database. With everything put together, the server can run our implementation of the A* pathfinding algorithm to efficiently find the shortest path between points. For the purposes of our demo, we are not mapping stairs or elevators, as we will treat every floor as a separate self-contained map.

I. Directions Implementation

When the user selects a room they want to navigate towards, an asynchronous POST request is sent to the Django server's navigation API endpoint. This endpoint allows the server to run two Breadth First Searches from the start and end points inputted to find the nearest points in the graph. Then, the shortest path is computed using the A* pathfinding algorithm. Then, it computes that path into individual line segments by determining when they change direction. Examining pairs of adjacent line segments provides us with the direction the user needs to turn. Then, the server compiles the path that should be drawn on the user's browser, as well as generating an icon that should be used to provide a visual indication, and sends the response back to the client.

The client parses the server's response, drawing the path on a canvas element placed over the map. It also calculates the length of the path, converting it into meters, before populating the header and footer with those values. To decrease the load on the server, the navigation API endpoint is called sparingly. The browser plays a role in contributing to calculating the navigation path, automatically aligning the path's end with the user's position as they walk down the hallway. As the path grows or shortens, the metrics regarding the time remaining or distance remaining correspondingly change.

Once the user is within two meters of the turn, the instructions update to display the next turn they should make. If they are at their destination, we will have the navigation mode display this by pausing navigation. Additionally, rerouting is also done as necessary. If the user strays too far off the path (at least 5 meters away), the application will assume they have gone into a different hallway, and will end up calling the navigation API to provide an updated path.

VII. Test, Verification and Validation

A. Results for DWM1001-Dev Range

To ensure we have sufficient connectivity inside of a building while keeping our costs down, we need to find the maximum distance at which two DWM1001 boards can communicate with each other. Our goal for the maximum range is at least 25 meters to meet the range discussed in the design requirements. We used a laser measuring tool to measure the distance between the two devices. We tested the range with walls blocking and without walls. For the situation without walls, the maximum distance between two DWM1001 devices is 35 meters. And for the situation with walls, the maximum distance between two DWM1001 devices is 26.5 meters. Beyond either of these distances, the connection became substantially worse. Both of the results fulfill the design requirements.

B. Results for UWB Measurement Accuracy

Our target goal for the accuracy of our UWB measurements was to be <0.23 meters to make sure that we would be able to have sufficient accuracy for our multilateration to function with < 1 meter error.

Our test involved us rolling a measuring tape between an anchor and a tag before comparing the distance seen on the tape with that measured by UWB. This was done in an indoor environment to simulate actual usage. The results are shown in Figure and reveal that the average difference is around 0.15 meters. This indicates that the accuracy of our localization system is sufficient for accurate localization and passes the test. This affirms our calculations from the design requirements where we predicted this could be achieved in theory.

C. Results for UWB Measurement Frequency

One of the goals for meeting a high update frequency for a smooth tracking experience is to have a fast distance update speed for our tag and its network of anchors, our goal being an update speed of at least 2 Hz. We measured the frequency at which our tag can get new distance values and found that this result is at 10.1 Hz. Hence, these results reveal that we are capable of providing very frequent updates to the user's position, as long as other factors are not bottlenecking the localization process.

D. Results for Localization Algorithm Latency

Tying into our previous test with the UWB measurement frequency, we want to make sure that none of the algorithms for the calculation of localization are bottlenecking our system. We find that it takes the Raspberry Pi a very fast average of 20 ms to run our multilateration algorithm. Additionally, the overhead of actually sending the data to the server is only 5 ms, causing the total computation time to take around 25 ms. This indicates that obtaining UWB distances is our bottleneck. Furthermore, because the localization algorithms are running on a different thread than the one obtaining the UWB distances, our localization update frequency is the same as our UWB measurement frequency.

E. Results for Localization Accuracy

To test the accuracy of our localization system, we stood at random positions within the hallways of the building and used a measuring tape to measure our distance from two walls to find our location in 2 dimensions. An estimated location from the localization system was obtained by using the pixel offsets found in the web application and converting them into meters. The difference between the actual position and the estimated position was calculated as the error. The average error should be less than 1 meter to fulfill the design requirements for accurate localization.

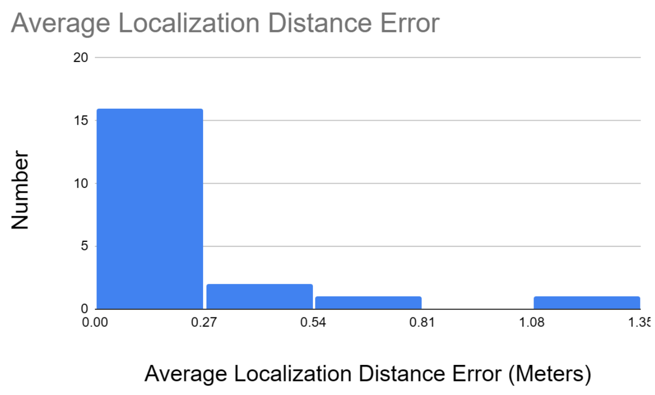


Fig. 6. Average Localization Distance Error

The results of our test are shown in Figure 6. We were able to find that the average error was 0.20 meters, which meets our use case requirements. The UWB measurement accuracy test meeting its design requirements contributed to helping this one pass, as we found from the design requirements discussion.

F. *Results for Localization Precision*

As a part of our localization system's accuracy, we wanted to avoid massive jumps in the user position and limit these to 0.5 meters. To test the precision of our localization system, we stood in a location and measured the distance of jumps we would see in the localization system. We notice the maximum fluctuations were approximately 2.1 meters, which was larger than the 0.5 meter goal. Because of this, we designed a filter for the final position estimate, that uses the change in estimated position over time to calculate our velocity, and rejects data points that would imply a velocity higher than some maximum (which we specified to be 2 m/s).

G. *Results for Heading Accuracy*

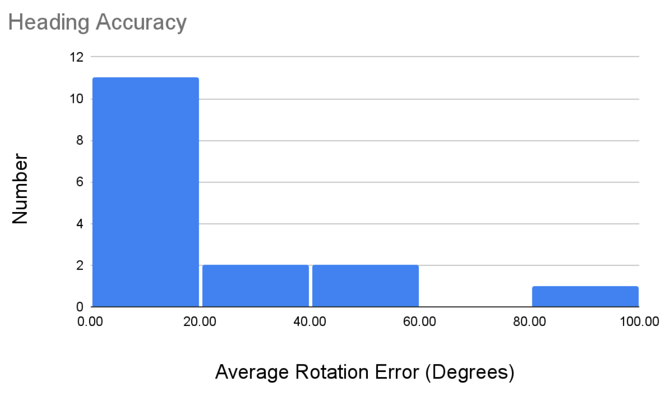


Fig. 7. Average Rotation Error

From our use case requirements, we want the heading of the user to be accurately represented on the map to provide a better sense of where they should turn. To test this, we walked along the hallway, rotating around, before stopping at parallel angles with the hallway (in real life) before comparing the result to that of what was seen on the browser. We took the difference between that angle and the actual angle. The average result over 15 trials reveals the average difference in angle is 20 degrees (Figure 7). This aligns with our goal of 20 degrees.

H. *Results for Navigation Optimality*

To ensure the optimality of the travel paths proffered by our navigation system, we used the A* pathfinding algorithm on a map of the building floor plan that has been divided into a grid and has had rooms and other obstructions marked as untraversable areas. The A* pathfinding algorithm mathematically guarantees an optimal travel path, so we need only verify that our implementation of the A* pathfinding algorithm is capable of always finding the shortest path.

This test was achieved by comparing the result of our A* implementation with a known BFS solution. We tested it on a collection of 50 randomly selected starting and ending positions of the map and made sure the path generated by the A* algorithm was always equal to the length of the BFS path. This test passed, indicating the A* algorithm was able to always find the shortest path.

I. *Results for Total Latency of System*

Testing the total latency of our system from user movement to the position being updated on the user interface is important to ensure that the user's position does not lag too far behind. We wanted the maximum latency under normal Wi-Fi conditions to be under two seconds.

Our tests consisted of walking along the hallways and recording videos of the map on the user's phone and them walking. We rewatched the recordings to find the differences in the user passing landmarks in real life versus them being reflected on the phone screen. These differences represented the total latency of our system. After taking measurements in different hallways, we averaged the measured latencies to get the average latency. The maximum result was 840 ms, which met the use case requirements.

J. *Results for Tag Device Characteristics*

To verify that our tag's battery life lasts long enough to meet our requirements, we started a navigation session with the device, and left it running. The device was still powered on after 4 hours, confirming that our battery life lasts long enough to aid a student throughout the day as they are walking indoors. Our testing actually showed that the device is capable of lasting around 10 hours when powered by the power bank, substantially exceeding our use case requirement.

We also measured the dimensions of the final package of the tag to ensure that the volume is less than 2 liters in size, ensuring it is not too much of a burden to carry around. We measure the final size of the device to have a footprint of approximately 0.49 liters, leading to a compact design that meets our use case requirement.

Finally, the cost of the tag device is $97, which meets our use case goal of the tag device being within $100. Via our testing of actual system metrics, as well as testing in various floors of buildings, we have found our infrastructure cost to be $38 per hallway corner, which meets our original use case requirement of $50 per hallway corner.

K. Results Quality of Directions & User Experience

To validate that the directions that we deliver to the user are understandable and that our user interface is easy to use, we resolved to pilot the app with several potential clients, these tests being carried out whilst testing our navigation system. The plan was that we will give our clients the tag device, and tell them a room in the building to navigate to. They will then have to enter this destination into the app and follow the directions it gives until they reach this room. After each trial, we ask them for qualitative feedback and a rating out of 5 based on the user experience of the app, as well as the quality of the instructions provided. Using their feedback, we work on improvements to any features the clients had issues with. And then we execute this feedback until we get at least 4 out of 5 on average.

While testing our navigation system, we recruited four students to navigate to an office on the second floor of Roberts Engineering Hall. All of these students were previously unfamiliar with the layout of the building. We provided them with a room to navigate towards with our application. The users attempted to walk to the room. At the end of their navigation session, we questioned their experience to obtain feedback. Ultimately, we were able to get a 4.5/5.0 rating on the usefulness of the system. The users indicated they thought that the instructions were useful and that they appreciated the propensity of the system to provide navigation times. These results provide validation that our product is capable of achieving its purpose.

VIII. Project Management

A. Schedule

Our schedule is displayed in Figure 8. There are substantial changes from this schedule from our original schedule. As we obtained a better understanding of the devices we were going to use, we correspondingly updated the schedule to account for those differences. Additionally, we were fairly cautious about scheduling excessive amounts of work per week, so we left two weeks of slack at the end of the semester. As the semester progressed, we updated our weekly schedules to better account for where we were, and set more reasonable and specific goals for the following weeks. However, despite the number of changes, the core of the schedule remained the same, where we had concurrent development of the localization and navigation systems.

B. Team Member Responsibilities

Jeff Chen was responsible for the web application portion of the project. He set up the web server and built the app to run in the browser of a mobile phone. He also was responsible for designing the entire user experience, including destination settings, updating the real-time map (with the user's position marked on it), and providing navigation routing and natural language directions. He worked with Weelie to build the communication endpoints to support the tag's communication with the server. Due to his work with including maps on the server, he ended up being in charge of mapping testing areas and developing the tools for improving the mapping process.

Weelie Guo was in charge of the anchor devices. This mainly involved programming the anchor devices to provide the right data to the tag device as it is needed. He also worked with Jeff on the mapping side of the project, planning where to place the anchors on the map in order to maximize localization fidelity. Additionally, he worked with Ifeanyi on the tag device, writing the trilateration algorithm for turning distance measurements into position estimates.

Ifeanyi Ene handled the tag device. He was responsible for designing and building the device that pairs with the user's smartphone to aid in navigation. He worked with Weelie to implement the localization routine, optimizing it with filters to improve the accuracy of results. He also wrote the code for the IMU to be able to estimate the user's orientation.

C. Bill of Materials and Budget

See Table III for the full Bill of Materials.

D. Risk Management

Our initial risks we outlined in our design report related to uncertainty towards programming and interfacing with the DWM1001 UWB transceiver. Due to a lack of experience in interfacing with the nrf52832 microcontroller, we were unsure if they would be suitable for our application. This also led to some uncertainty with our budget: there was always a chance that pivoting to a different device could be costly, or that we might need more transceivers than we initially assumed. Due to this risk, our primary goal at the beginning of the project was to tackle the issue with interfacing with the chips and start programming them. We quickly found that the support for programming the boards was fairly robust and that we did not need to make any adjustments to opt for a different option. Furthermore, once we got distance calculations, we needed to do a substantial amount of initial testing to verify they would be suitable for scalability.

We found many difficulties in planning our original schedule for the semester. Planning around nine weeks ahead revealed many difficulties as we were unsure of how long different tasks would take, or what tasks would even be necessary to complete later on in the semester. To alleviate the risk of running out of time near the end of the semester, our original schedule had two weeks of slack time. On the other hand, to make sure that we were on track and always had something we were striving to accomplish, we made it a point during our weekly meetings to discuss what we were planning on achieving in the immediate weeks (which ultimately led to many revisions of our schedule). Ultimately, this strategy ended up being successful.

Throughout the process of designing our localization system, we found a number of issues that could cause us to miss some of our original use case goals. Among these was an issue of position estimation latency. We found the tag was getting distance updates from its anchors at a slow rate of around once every few seconds. While we worked on resolving this issue, we also worked on developing a fallback method using our IMU. We attempted to use the IMU to integrate the user's position in space while using the UWB-based position estimate to adjust this position gradually over time using a complementary filter. Later on in the semester, we were able to resolve the original issues we had with the distance update frequency, which ended up rendering much of the work done on the IMU as obsolete. In hindsight, this was a relatively extreme risk mitigation. Development time spent on two split ventures slowed down development for both, though it was a sacrifice we were willing to make for a fallback plan.

IX. Ethical Issues

Our system handles location data, which is very sensitive, and even moves it over the web. If this data could be acquired by the wrong third party, it could lead to stalking and general unwanted surveillance. To combat this concern, we have locked access to the system behind user profiles. This way, a user's location data is available only to them and anyone else they share their password with. The server itself also does not save passwords, saving hashes instead, adding further security to the user's private data.

One possible edge case we are missing for our application is accessibility for people who are vision impaired. Our application requires a screen to read. To improve accessibility, we can include features such as text-to-speech and speech-to-text to allow these users to be able to interact with the different menus and listen to the instructions the device provides them.

Scotty Maps not only has a highly accurate localization system and helpful navigation system, but we have also developed many tools for implementing the system in a wide range of environments. If the system were to be applied in more environments outside of just schools, it could provide a substantial aid for people navigating in large, unfamiliar indoor environments such as shopping malls or airports. The widespread adoption of indoor navigation in these areas could help people become more comfortable navigating them and decrease the time they spend lost.

X. Related Work

The indoor localization aspect of our project is similar to the Inexpensive Sports RTLS System capstone project that was done in Spring 2020. This team also used DWM1001 development boards to do indoor localization. Instead of using TWR TOA for localization purposes, this group used TDOA instead.

Additionally, in Spring 2023, a capstone team developed a project called "WiSpider", which also used wireless signals to localize devices within buildings. However, they used Wi-Fi signals rather than UWB. Furthermore, they used pre-installed CMU Wi-Fi access points rather than building their own anchor devices [8].

We designed a navigation system interface heavily inspired by many extremely popular navigation apps such as Google Maps, Apple Maps, and Waze. However, instead of a focus on mapping the globe, we will instead focus on mapping out CMU academic buildings.

There is also a company named Sewio that offers a similar UWB-based tagging and indoor localization service, but based on the size of the deployment (if we use all ~40 academic buildings on the CMU campus), their system can easily run into the hundreds of thousands of dollars, making it much less accessible [9].

XI. Summary

In conclusion, we were able to successfully develop a project that solves the use case of students navigating indoors within CMU buildings. Our system was able to meet all design specifications. Users can experience a low latency, long battery life, accurate localization service with navigation options and clear directions. See Section VII for a detailed breakdown of the system's performance.

As for the limits of the system, the most notable would be the assumption that the user is never moving faster than 2 m/s. While this is a rather comfortable walking speed, it doesn't work for cases where the user is briskly walking to their destination. Additionally, the DWM1001 specification states that a UWB anchor network can support up to 15 tags for the TWR protocol while maintaining its current update frequency due to bandwidth. Hence, beyond 15 users on a single floor, the location update frequency will drop from 10 Hz to 1 Hz.

Through completing this project, we have learned about UWB and the different protocols that can be used to obtain distance estimates from them for localization. The primary thing we did not predict going into the project was the imprecision of the distance readings. Any student in a subsequent semester wanting to adapt or improve upon this project should do their best to have their system not rely primarily on these distance readings.

Glossary of Acronyms

BLE – Bluetooth Low Energy
RPi – Raspberry Pi
RSSI – Received Signal Strength Indicator
TDOA – Time Difference of Arrival
TOA – Time of Arrival
TWR – Two-Way Ranging
UWB – Ultra Wideband
IMU – Inertial Measurement Unit

References

[1] DecaWave. 2014. The implementation of two-way ranging with the DW1000.

[2] Yuntian Brian Bai, Suqin Wu, Guenther Retscher, Allison Kealy, Lucas Holden, Martin Tomko, Aekarin Borriak, Bin Hu, Hong Ren Wu, and Kefei Zhang. 2014. "A new method for improving Wi-Fi-based indoor positioning accuracy", Journal of Location Based Services, 8:3, 135-147, DOI: 10.1080/17489725.2014.977362

[3] Ramirez, Ramiro, Chien-Yi Huang, Che-An Liao, Po-Ting Lin, Hsin-Wei Lin, and Shu-Hao Liang. 2021. "A Practice of BLE RSSI Measurement for Indoor Positioning" Sensors 21, no. 15: 5181. https://doi.org/10.3390/s21155181

[4] Mazhar, F., Khan, M.G. & Sällberg, B. Precise Indoor Positioning Using UWB: A Review of Methods, Algorithms and Implementations. Wireless Pers Commun 97, 4467–4491 (2017). https://doi.org/10.1007/s11277-017-4734-x

[5] F. Thomas and L. Ros, "Revisiting trilateration for robot localization," in IEEE Transactions on Robotics, vol. 21, no. 1, pp. 93-101, Feb. 2005, doi: 10.1109/TRO.2004.833793.

[6] Gao, F., Han, L. Implementing the Nelder-Mead simplex algorithm with adaptive parameters. *Comput Optim Appl* 51, 259–277 (2012). https://doi.org/10.1007/s10589-010-9329-3

[7] Building Floor Plans and Room Information. Carnegie Mellon University. (n.d.). https://www.cmu.edu/finance/property-space/floorplan-room/index.html

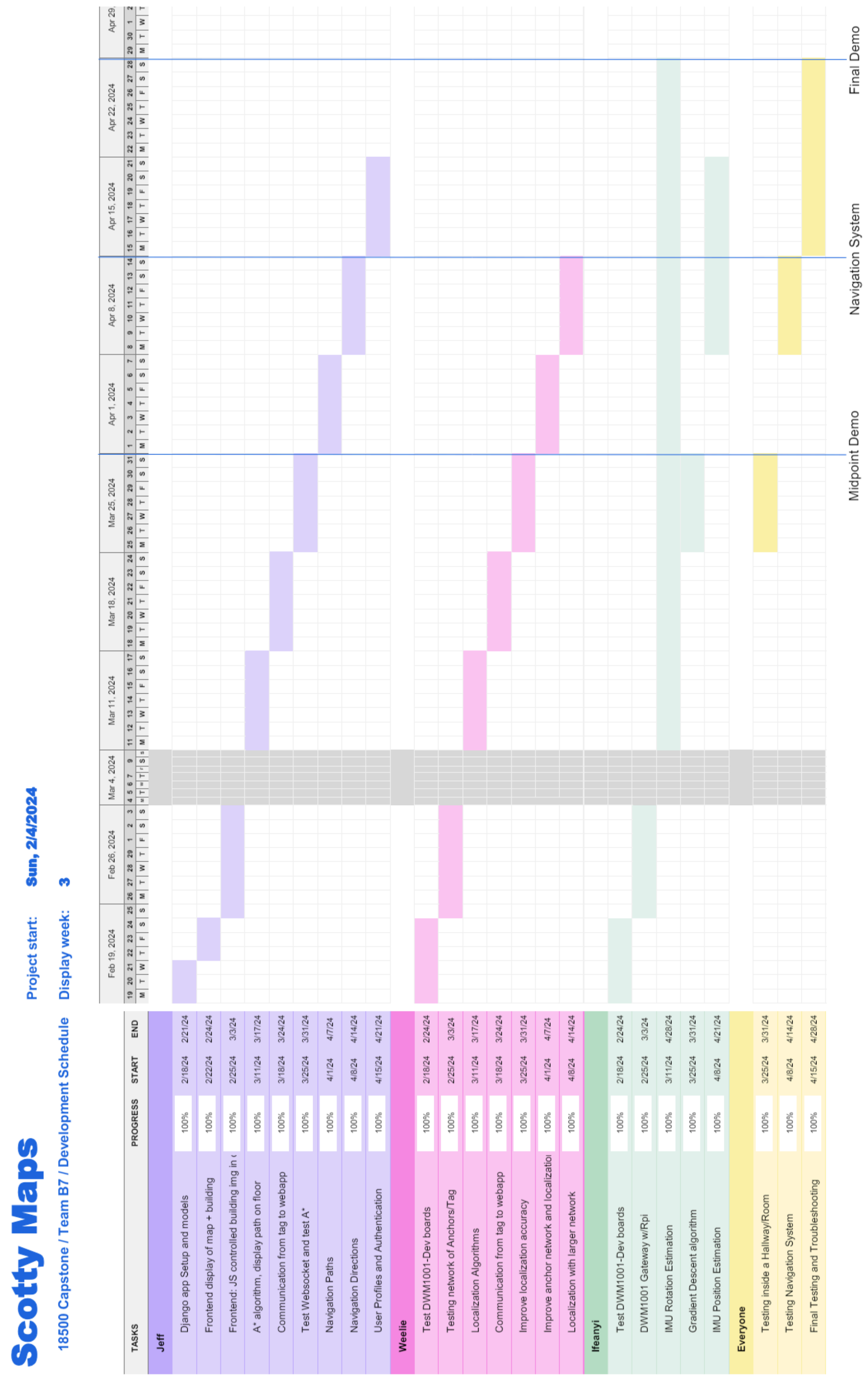
[8] Anish Singhani, Thomas Horton King, Ethan Oh. 2023. "WiSpider", CMU ECE Capstone, Spring 2023. https://course.ece.cmu.edu/~ece500/projects/s23-teamc2/wp-content/uploads/sites/238/2023/05/Team_C2_Oh_Singhani_Horton-King_final_report.pdf

[9] Sewio corporate website. https://www.sewio.net/

[10] Qorvo MDEK1001 product listing. https://store.qorvo.com/products/detail/mdek1001-qorvo/681952/

Table III. Bill of Materials

Description	***Model Number***	***Manufacturer***	***Quantity***	***Cost***
Raspberry Pi	4	Raspberry Pi Foundation	1	Inventory
IMU	GY-521	HiLetGo	1	$3.33
UWB Development Boards	DWM1001-Dev	Qorvo	12	Inventory
Li-Ion Batteries for DWM1001	16340	CWUU	12	$2.50
Battery Pack for Tag	10000mAh	Charmast	1	$20.00
Laser Measuring Tool*	LX-201	LEXIVON	1	$50.00
Adhesive Strips*	Heavy Duty	3M	1	$6.00
Li-Ion Battery Charger*	16340	GRACETOP	1	$10.00
Total Cost				**$119.33**
*** = Added after Design Report**				



TASKS	PROGRESS	START	END
Jeff			
Django app Setup and models	100%	2/18/24	2/21/24
Frontend display of map + building	100%	2/22/24	2/24/24
Frontend: JS controlled building img in (	100%	2/25/24	3/3/24
A* algorithm, display path on floor	100%	3/11/24	3/17/24
Communication from tag to webapp	100%	3/18/24	3/24/24
Test Websocket and test A*	100%	3/25/24	3/31/24
Navigation Paths	100%	4/1/24	4/7/24
Navigation Directions	100%	4/8/24	4/14/24
User Profiles and Authentication	100%	4/15/24	4/21/24
Weelie			
Test DWM1001-Dev boards	100%	2/18/24	2/24/24
Testing network of Anchors/Tag	100%	2/25/24	3/3/24
Localization Algorithms	100%	3/11/24	3/17/24
Communication from tag to webapp	100%	3/18/24	3/24/24
Improve localization accuracy	100%	3/25/24	3/31/24
Improve anchor network and localizatio	100%	4/1/24	4/7/24
Localization with larger network	100%	4/8/24	4/14/24
Ifeanyi			
Test DWM1001-Dev boards	100%	2/18/24	2/24/24
DWM1001 Gateway w/Rpi	100%	2/25/24	3/3/24
IMU Rotation Estimation	100%	3/11/24	4/28/24
Gradient Descent algorithm	100%	3/25/24	3/31/24
IMU Position Estimation	100%	4/8/24	4/21/24
Everyone			
Testing inside a Hallway/Room	100%	3/25/24	3/31/24
Testing Navigation System	100%	4/8/24	4/14/24
Final Testing and Troubleshooting	100%	4/15/24	4/28/24

Fig. 8. Gantt chart with our planned schedule.