A ZONE-BASED, UNDERGROUND LOCALIZATION SYSTEM USING PASSIVE REVERSE RFID AND IMU TECHNOLOGIES

by

Robert D. Jones

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Electrical Engineering).

Golden, Colorado

Date ____________________

Signed: ____________________

Robert D. Jones

Signed: ____________________

Dr. Atef Z. Elsherbeni
Thesis Advisor

Golden, Colorado

Date ____________________

Signed: ____________________

Dr. Peter H. Aaen
Professor and Head
Department of Electrical Engineering

ABSTRACT

In order for miners to mitigate hazardous dust concentrations underground, they must first identify where in the mine the high concentrations occur. To do this, they wear dust monitoring units to track the amount of dust in the regions they travel and recording their locations by hand. This is extremely time consuming for large underground mines making an automated localization system a necessity. Underground localization systems typically use massive amounts of infrastructure such as active radio beacons or modulated light sources that require connection to power and communication infrastructures making them unfeasible for cost-effective and rapid deployment.

This thesis describes a wearable system that uses a combination of IMU dead reckoning, reverse passive RFID trilateration, and map matching to localize a user in both an indoor and underground environment. The only infrastructure required for the system to operate are clusters of passive RFID tags sparsely placed throughout the area. IMU dead reckoning localizes the user in between tag clusters while the RFID tag clusters reset the drift errors accrued by the IMU. Map matching projects the dead reckoned values onto a path, sacrificing a user's lateral distance from the path for a massive increase in accuracy. The system presented successfully localizes a user at Colorado School of Mine's Edgar experimental mine.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Radio Frequency Identification	RFID
Radio Frequency	RF
Inertial Measurement Unit	IMU
Degree of Freedom	DoF
Received Signal Strength Indicator	RSSi
Medium-Density Fibreboard	MDF
Universal Battery Eliminator Circuit	UBEC
Fast Fourier Transform	FFT
Inverse Fast Fourier Transform	IFFT
Look-Up-Table	LUT
Right-Hand Circularly Polarized	RHCP
Left-Hand Circularly Polarized	LHCP
Comma Separated Values	CSV
Decibels-milliwatt	dBm
Zone Categorization Algorithm	ZCA
Clockwise	CW
Counter-Clockwise	CCW
Micro-Electromechanical Systems	MEMS
Software Defined Radios	SDR
Integrated Development Environment	IDE

Virtual Private Network	VPN
Integrated Development and Learning Environment	IDLE
Angle-of-Arrival	AoA
Time-of-Arrival	ToA
Programmable Intelligent Computer	PIC
Global Navigation Satellite System	GNSS
Global Positioning System	GPS
Pedestrian Dead Reckoning	PDR
Bluetooth Low Energy	BLE
Ultra-wideband	UWB
Access Point	AP
Integrated Circuit	IC
Continuous Wave	CW

ACKNOWLEDGMENTS

I would like to thank Joseph Diener for all of the guidance and encouragement he has given me these past couple of years as well as making a fantastic antenna for the prototype and helping me with the IMU coding. Kyle Patel, thank you for all of your help in getting me started on this project and for all of the advice you have given me throughout my undergraduate and graduate time. Thank you Yiming Chen for all of the assistance in acquiring data during the Brown building tests. Rachel Lumnitzer and Mike McNulty, thank you for all of the help in setting up the many tags inside of Edgar mine. Also Mike, your quick lessons on bash scripting saved me a tremendous amount of headache in creating the acquisition code for the prototype. Dr. Jürgen Brune, thank you for your continuous suggestions over the course of the project as well arranging all of the trips to the Edgar mine. Dr. Atef Elsherbeni, thank you for being my advisor these past five years. You have provided me with so many opportunities that have enabled my success to a point I could have never dreamed of. Dr. Kate Remley, thank you for all of your support these past four years as well as teaching me how to be a NIST-grade metrologist. Also, thank you for your patience as I multi-tasked college, work, and graduate research. Last but not least, thank you to my parents and friends for all of the love, understanding, and support you all have given me the past couple of years.

The work is funded by the Alpha Foundation under contract number AFC820-50 and the National Institute for Occupational Safety and Health (NIOSH) under contract number 75D30119C05412.

To my family: Susan, Bob, and Andrew.

CHAPTER 1
INTRODUCTION

1.1 Motivation

Often traditional localization methods rely on GPS and other GNSS technologies in order to function, and hence are unusable in the GPS-denied environment. Routine operation of a mine can lead to harmful concentrations of dust and other particulates building up in isolated areas, which present a health risk to miners. Dust concentration hazards are especially harmful in coal mines as they can cause explosions such as the one that killed 29 miners in 2010, [1]. For coal dust mitigation, the miner sprays a water and rock dust mixer on top of the coal dust to render it inert [2] as well as use airflow routes through a mine to move the dust out [3]. In order to mitigate the dangerous dust build ups in a mine, the miner must be able to map where the dangerous dust concentrations are. Miners wear dust monitors to track the amount of dust they are exposed to as they patrol a mine. Mapping the dust concentration requires the position of the miner which is typically done by hand which is extraordinarily time consuming for large mines. There is a need for an affordable, robust, localization system that requires very little infrastructure to operate and can provide accurate positions to aid in mapping sensor readings.

The dust monitor used for the dust mapping is the PDM 3700 as described in [4]. This unit takes measurements once per minute and requires post-processing software to obtain the dust concentrations. Dust mapping is an active area of research that is also covered within the scope of the two grants that support the work of this thesis but dust mapping itself is not within the scope of this thesis. This thesis focuses on the miner localization system which has the primary purpose of providing the miner's position to the dust mapping software. The presented localization system does not control the dust monitor but will operate in parallel with it so that the user's position over time can be matched with the user's recorded dust concentration over time.

The localization system must have the following characteristics:

- The system must be body-wearable, restricted to only the helmet and trunk of the user. Foot-mounted solutions are not ideal due to contamination.
- The system must have resolution on the order of feet but not inch-level accuracy as the dust monitor only samples once a minute and the user can travel many feet during this time.
- The system cannot rely on any power or wireless communication infrastructure in the mine beyond passive infrastructure, such as RFID tags. The infrastructure installed must not require constant maintenance.
- The system cannot rely on constant contact with any communication infrastructure because mines easily disrupt wireless communications.
- The system must be scalable to a large environment.
- The system must be easy and fast to deploy.
- The system must be as low-cost as possible.

These requirements are very stringent and eliminate many of the previously proposed localization techniques as viable options.

1.2 Previous Solutions

The fields of indoor/underground localization are heavily saturated by decades of potential solutions for localizing people and vehicles. These solutions invoke a wide range of sensors including: RFID, ultra-wideband communication, Wi-Fi, Bluetooth, inertial position estimation, communication using light, magnetic field classification, mapping, sensor networks, and ultrasonic. This section will summarize a couple of the previous solutions to this challenge.

The vast majority of underground localization solutions are aimed not at creating a map of a sensor reading, but rather at real-time tracking of miners in a mine where their positions are transmitted back to a central hub. These solutions usually use wireless access points, zigbee sensor networks, RFID readers, or mesh networks to communicate with the miners. Solutions such as the one in [5] and [6] use Zigbee sensor nodes scattered throughout the mine to communicate with each other and localize miners that move in their proximity. Wifi fingerprinting is another possible solution as described in [7], where a miner will detect multiple wifi access points (AP) and based on the MAC address of the AP's and the received signal strength indicator (RSSi), be able to determine their most likely position. Another fingerprinting solution is presented in [8] where the author measured the RSSi of the leaky feeder lines at various positions to create a "radio-map". Using this map, users can match the pattern of RSSi values they receive as they advance through the mine to the "radio-map" to localize themselves. This technique had a resolution of 40 meters which is too coarse for dust monitoring. Visible light communication (VLC) as discussed in [9], [10], and [11] involves the use of lamps placed at the roof of the tunnel that act as over-the-air light transceivers sending optical signals to the miner's helmet which then it turns sends data back to the lamps. A similar solution to VLC is presented in [12], where the miner projects a unique shape at the top of their helmet onto blackboards placed throughout the mine where a camera network then detects this shape to determine where the user is. It is also popular to place RFID readers throughout a mine and place the tags on the miner to localize them, as described in [13]. All of these techniques require a tremendous amount of infrastructure to install into the mine and are therefore not viable solutions to the problem.

The work presented in [14] uses a combination of IMU and active beacon technologies to localize the miner. The active beacons are placed in different place in the mine and emit an ultrasonic as well as an RF signals. The miner wears a receiver that receives both of these signals, using the ultrasonic signal to determine distance from the user to the beacon via time-of-flight and using the RF signal to synchronize the beacon and the receiver in time. Once the distances from the receiver to three beacons are known, the localization will use a least-squares trilateration approach to determine the user's coordinate. It is assumed that all of the beacons will have their coordinates pre-determined at the time of their installation. The active radio beacon positions are fused with an IMU to get a sensor fused position. This work uses an embedded system to record the IMU and receiver measurements where data are then offloaded onto a computer running MATLAB to execute the post-processing algorithm. It was not clear from this work if the laboratory it was tested in was underground or just indoors. This work demonstrates a potential solution using active beacons, least-squares trilateration, and IMU data.

The work in [15] describes an indoor localization algorithm focusing solely on the IMU contained within a smartphone to perform pedestrian dead reckoning (PDR). They use a Kalman filter to fuse the accelerometer and gyroscope data to calculate the user's pitch angle over time. Using the pitch angle, they determine when the user took a step. The highest maximum amplitude of the pitch angle and the lowest minimum amplitude of the pitch angle, combined with a linear regression model is used to extract a user's step length. This step length is applied to the dead reckoning as a fixed value through the entire measurement. This work uses another Kalman filter to fuse the gyroscope and magnetometer readings to determine the user's heading. Once the step length, step occurrence, and user's heading are determined, the author presents a simple dead reckoning equation to determine the user's position. The tests performed were in an indoor environment where the system demonstrated good localization around simple paths with around 0.67 to 1 meter of error. This source represents a potential IMU-only solution to user localization.

The work in [16] discusses many different technologies that are commonly used in localization systems for underground mines. This overview discusses the drawbacks in inertial localization, Wi-Fi fingerprinting, passive RFID proximity localization, map-based positioning, very-low frequency radio wave navigation, and magnetic field positioning. Beyond this overview, the author tests four different technologies: Bluetooth low energy (BLE), magnetic field positioning, IMU dead reckoning with map matching, and ultra-wideband (UWB) in a small section of an underground mine tunnel. The author uses the BLE system to get the user's coordinate narrowed down to an area and then uses the IMU dead reckoning to finely estimate the user's position within that area. The IMU dead reckoning is determined using a step counter with a fixed step length estimate. To determine the distance from a BLE node to the user's location, a polynomial curve fit converts the received signal strength of the BLE node to distance. However, the variation of the received signal strength for a given distance almost has a spread that is over the entire range of possible RSSi values. If the BLE system is non-operational, the localization systems uses magnetic field data instead, comparing it to the entire mine's magnetic field database and seeing where the point of highest correlation is. The point of highest correlation will be roughly the area where the miner is. The major downside to magnetic field positioning is that there is a significant amount of pre-characterization work that goes into building the database and will give inaccurate readings if machinery or other metallic objects are moved around inside of the mine. At the end of the positioning, the algorithm uses map matching to align the localized path to the map. The testing was done in a small section of an underground mine with a very simple path. This source shows an underground localization solution that fuses three different types of sensors to get the user's position, as well as good general overview of all of the flaws of other technologies used in localization. Another significance of this work is the use of a polynomial curve fit to estimate the RSSi of the BLE to distance, even though it has tremendous variation around the fitted curve.

Passive RFID is not as popular in underground mines as active RFID or radio nodes are, but it is a solution that others have leveraged in the past. The work presented in [17] and [18] use passive RFID tags as ”landmarks” to map out a mine. The passive RFID tags were not used to directly localize the user inside of the mine but as anchors to stitch together the various map sections taken from their laser scanner data. Interestingly, they did not need to pre-record the 2D coordinates of the RFID tags, instead they just recorded the unique tag ID at the location they read them and then stitched together all of the various maps by using the landmarks as reference points. In [19], the researchers place passive RFID tags at intersections of the mine to assist the loader in autonomously driving through the mine. The reason why passive RFID has not been adopted into any localization solution is because the read range is thought to be too short, according to [20], and [21].

A further subset of passive RFID is whether the system performs tag localization, or reader localization. Tag localization in an underground or indoor environment is where many RFID readers are scattered throughout an area and will track the tag that is attached to a person. Reader localization, also called reverse RFID localization, is where tags are placed throughout an environment and the reader localizes itself off of the tag reads. In [22] the researchers present an indoor localization system that uses hundreds of tags placed all over a floor to localize a reader. The system presented in [23], uses multiple tag reads along with IMU measurements to determine the angle of arrival the reader is from the tag array. Knowing the angle of arrival information from a reader to a set of RFID tags is sufficient to localize the reader as described in [24]. In 2011, NIOSH describes in [25] the contract that they awarded to L3 communications for their underground reader tracking system, the ”Tru-Tracker Precision Location System,” but this used active UWB beacons that broadcasted their location. The overview paper by NIOSH in 2012, [26], theorizes the use of passive RFID localization inside of mines where the miner localizes themselves off of RFID tags and then transmits this position to a radio but at this time, no one had implemented a successful passive RFID reader localization system underground. To our knowledge, no other researchers have achieved this, making the system presented in this thesis the first successful passive reverse RFID localization system for mining personnel underground. None of the individual components of this thesis are novel: IMU dead reckoning, zone-based error correction, passive reverse RFID trilateration; but the combination of all of them in a complete system that was tested underground successfully is unique.

1.3 Radio Frequency Identification

The field of RFID technology has been around for over half a century, with many applications in object labeling, tracking, access control, and credit card commerce just to name a few. RFID spans a wide range of frequency bands such as the near field applications which reside in the 120-150 kHz and 13.56 MHz bands, and the far field applications which reside in the 433 MHz, 902-928 MHz, and 2450-5800 MHz bands [27]. In these bands, there are three different types of RFID tags: passive, semi-passive, and active. Passive tags harvest a portion of the energy off of the query signal of the reader, excite their inner integrated circuit to modify the query signal waveform, and then re-broadcast it back to the reader. Active RFID tags do the same thing except they do not harvest the incoming query signal but instead rely on an internal battery to emit their response. Semi-passive tags use battery power to operate the internal logic circuits but do not use it to amplify their response signal [27]. Passive tags are an attractive solution as they do not have batteries, have life spans greater than 20 years [27], but at the expense of relatively short read ranges. Active tags are an attractive solution as they have relatively long read ranges but with higher cost and a much shorter lifespan due to their battery. The tags used for this thesis are 915 MHz passive tags. RFID readers are designed in one of two ways: as a montostatic reader which uses a circulator and one antenna to perform simultaneous transmit and receive, or as a bistatic reader which uses two separate antennas instead. The readers used in this thesis are monostatic readers [27]. The RFID signals are restricted to a channel bandwidth of 500 kHz and a maximum data rate of 85 kbps. If multiple readers in the same environment are querying tags, other techniques such as multiplexing and frequency hopping will need to be used in order to prevent interference [27].

The complete tag read operation is described in detail in [27]. The reader first broadcasts a continuous wave (CW) query signal. This signal then travels over the air to the tag which then receives it, splits a portion of the signal through a rectification circuit, and then excites the tag's on-board integrated circuit (IC). Once turned on, the IC uses a switch to modulate its load between two different impedances to encode its ID and other information (depending on the tag's class) to the received signal. The impedance-modulated signal is then backscattered (sent back) to the reader. The complete read operation takes place on the order of milliseconds, so many tags can be read within a very short time period. The RFID readers use various protocols such as tag talk first (TTF) or reader talk first (RTF) to prevent interference between tag responses as well as between reader responses [27].

1.4 Proposed solution and Thesis Layout

A hybrid localization algorithm is developed based on incorporating RFID passive tags, IMU sensor readings, as well as map matching techniques. The IMU sensors produce reasonable estimates of a user's linear travelled distance but produces poor heading information at longer time scales due to small cumulative errors compounding into larger ones. In contrast, RFID tags can be used to produce localization of a user to a given area without the cumulative errors, although fine resolution is difficult to achieve. These different techniques produce superior accuracy when combined than any individually.

By placing a series of RFID passive tags inside a mine, zones are created that have *a priori* known positions that are used to reset the cumulative errors in IMU estimation. Multiple RFID tags are located in each zone to allow for the received signal strength (RSSi) based methods to produce a more accurate initial estimate of the user's location in a zone. The positional data can then be synced using the timestamps with other sensors the miner is wearing, such as a dust monitor, to create a map of a mining safety metric across the mine. The presented system uses a post-processing localization algorithm rather than a real-time positioning algorithm. Since the focus of this research was to map hazards in a mine, there is not an immediate need to turn this into a real-time algorithm. The developed system is optimized to be low-cost, robust, and accurate. By wearing the expensive reader, hundreds of cheap passive RFID tags can be placed throughout the mine with indifference to other communication infrastructure. This is inverse from other algorithms which use readers setup in various places of the environment and read tags on the users. This system does not require other infrastructure improvements beyond the tag installation making it easy to install. The system is robust because the reader needs to read at least one tag in a zone to be categorized and only three tags to determine the initial position. Constant tag reads are not required. Zones can be spaced far apart giving the user the ability to tradeoff positional accuracy with the number of tags required to divide up an environment. The use of more than the necessary number of tags in a zone, will allow the system to be operational even when mining hazards lead to the destruction of a couple of tags. Finally, the system exhibits greater accuracy over only IMU dead reckoning algorithms by the identification of zones using RFID tags and the application of the map matching process.

This thesis is organized into 4 chapters. Chapter 2 describes the data acquisition hardware and software used in the prototype. Chapter 3 describes the pre-characterization needed for the algorithm to work, as well as the post-processing MATLAB code for determining the user's position in the mine. Chapter 4 describes the experiments in Brown building and Edgar mine. Chapter 5 discusses the results and future work, and finally concludes the thesis.

CHAPTER 2
DATA ACQUISITION PROTOTYPE

2.1 Complete System Overview

The first requirement of the localization system is that it needed to be a body-wearable system that the miner could wear during their shift and not rely on any power or communications infrastructure inside the mine. Passive RFID tags are extremely cheap and require very little work to install into a mine so these were permissible infrastructure additions to the mine. In order to provide the data necessary for the localization algorithm, the system needed to record two main sets of data: human locomotion data of the user, and RFID tag readings. The last requirement of the system was that it needed to maximize the read range of the system to the RFID tags so the user could walk in both narrow corridors and large openings and still record RFID data. The final prototype that is described in this chapter used commercially available components, not including the antennas, that were easily operational at the expense of a compact system. The entire cost of the prototype system is $2,572. The breakdown of this cost is listed in Table A.1 in appendix A. Future improvements on this system will aim at making the system more compact and ideally fit on a miner's helmet or on a belt.

The prototype localization system uses two RFID readers, two custom-made antennas, a power supply circuit with a battery, an inertial measurement unit (IMU), and a Raspberry Pi. Figure 2.1 shows a block diagram of the prototype, and Figure 2.2 shows the final prototype, without the hardhat. The user controls the system through the Raspberry Pi which is connected to a touchscreen. Collected data are recorded to a Raspberry Pi, that is accessed at the end of a shift using USB or Wi-Fi to offload the data to a post-processing computer running MATLAB. The remainder of the chapter will discuss each of these components in detail as well as how they are controlled.

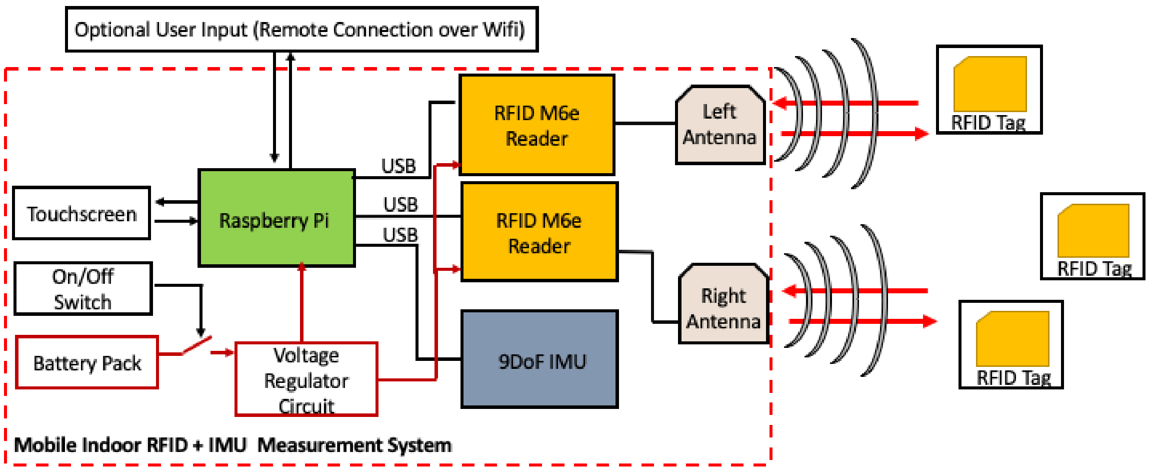


Figure 2.1: Hardware block diagram of the localization prototype.

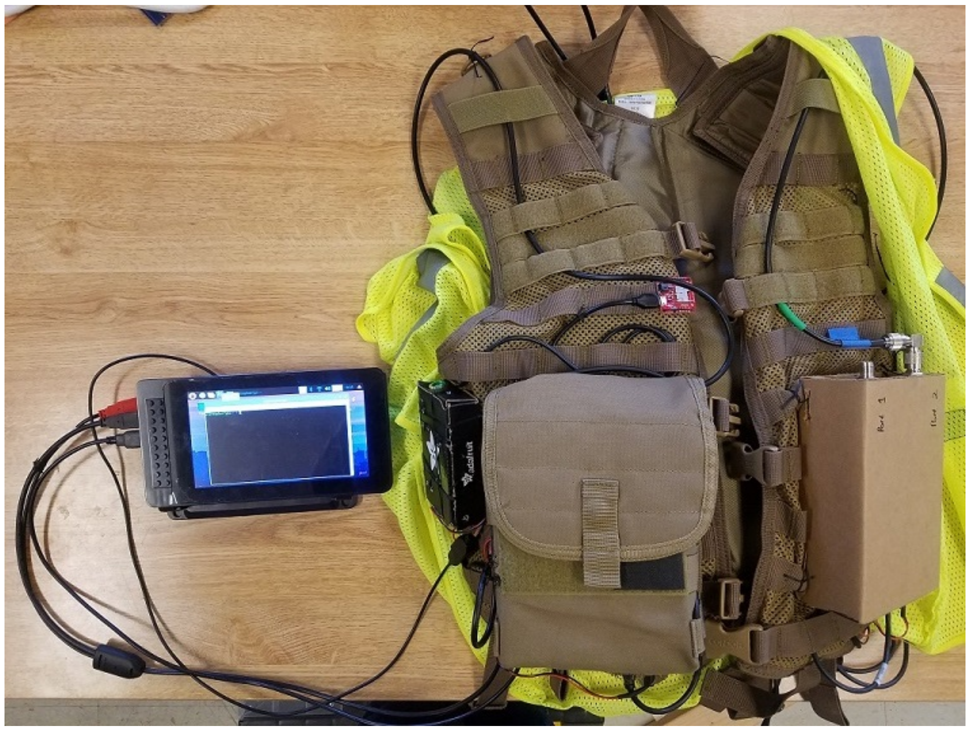


Figure 2.2: Complete wearable prototype. The vest contains two RFID readers, IMU, battery, and power circuit. The touchscreen contains the raspberry pi which connects to the power circuit, IMU, and RFID readers.

2.2 Inertial Measurement Unit

The IMU is an obvious choice to obtain human locomotion information and is used extensively in many other localization algorithms. The IMU chosen for the prototype is the Sparkfun Razor nine degree-of-freedom (DoF) sensor shown in Figure 2.3 and contains three accelerometers, three gyroscopes, and three magnetometers, one for the X,Y, and Z axis. The IMU uses a SAMD21 processor which is programmed using the Arduino IDE. The Arduino library to run the IMU is provided in [28]. It is mounted at chest level on a vest to record the user's trunk movement. Foot mount was avoided as the miners did not want to attach the device to their boots and head mount was avoided because the user needs freedom to look around as they are moving through a mine.

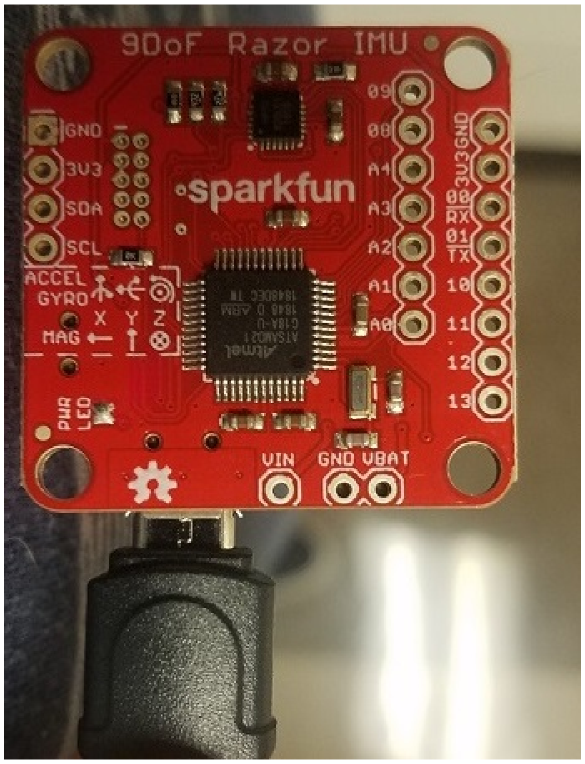


Figure 2.3: Sparkfun 9 DoF Inertial Measurement Unit.

The accelerometer is an micro-electromechanical (MEMS) sensor that works by recording the capacitive fluctuations from a plate suspended above the chip's surface held up by tiny microscopic springs as described in [29]. Changes in acceleration deflect the plate creating an output signal from the capacitors that is proportional to the applied acceleration. The units of the measured acceleration are of $\frac{m}{s^2}$.

The three axis gyroscopes are also MEMS sensors that operate similarly to the accelerometer in the sense that they use integrated capacitors to transduce the vibrations on the chip's microscopic spring-mass system to an analog value. This chip is described in more detail in [30]. The axis of the gyroscope are not described in X,Y, and Z but are instead described by the angular velocity along the axis of yaw, pitch, and roll. Using an airplane as an analogue, the yaw axis is the side-to-side turning of an airplane; the roll axis is the action where the airplane turns its left wing up right wing down, and vice versa; and the pitch axis is the up-and down orientation of the nose of the airplane to the Earth. For any system, the gyroscope can be oriented in such a way where the object's yaw axis is about the X, Y or Z axis or a combination of them. This prototype has the IMU oriented so that the Y-axis is the user's yaw axis, the X-axis is the user's pitch axis, and the Z-axis is the user's roll axis. See Figure 2.4 for an illustration of yaw, pitch, and roll pertaining to the miner. For this system, the yaw axis is the most important as it directly describes the user's heading. The output data from the gyroscope's yaw, pitch, roll axis are in units of $\frac{degrees}{s}$.

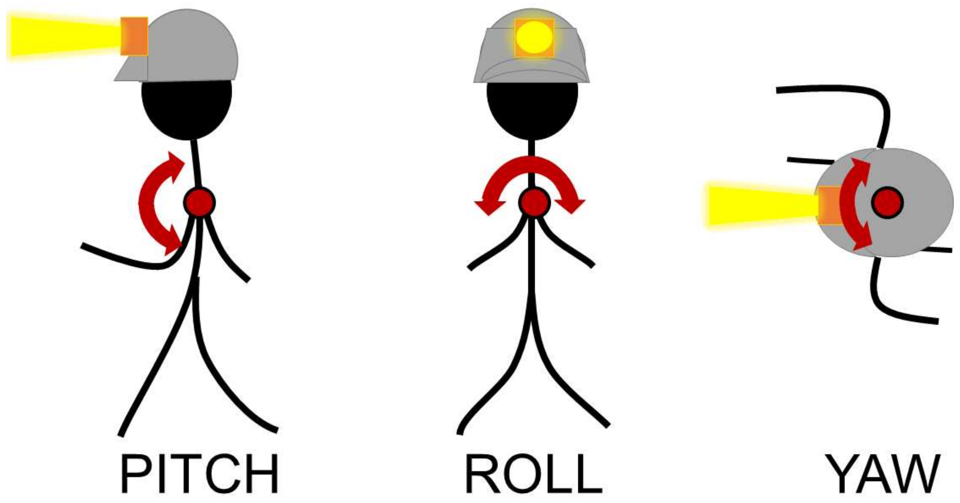
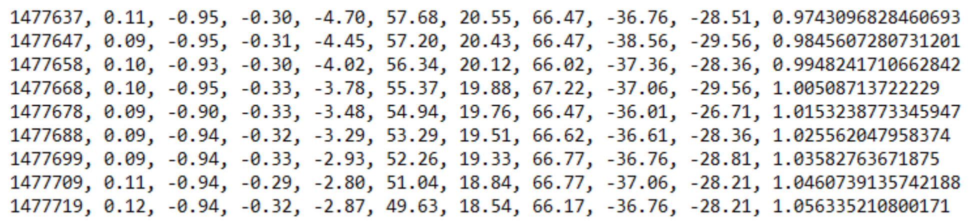


Figure 2.4: Illustration of pitch, roll, and yaw.

The three axis magnetometer records the magnetic field about each of the X,Y and Z axis, in units of Gauss, by monitoring the voltage changes across the magnetoresistors as described in [31]. The three axis magnetometer is not used in the localization algorithm but the data are always recorded by the prototype. The complex, metallic nature of the mine infrastructure, mining equipment, and variations of the ore composition can all throw off the magnetometer readings. The material in [32], and [33] propose solutions to the underground magnetometer challenges but these were not incorporated into the localization system presented here due to additional infrastructure requirements and the addition of more time-intensive pre-characterization tasks. Future work could be to incorporate a robust magnetometer algorithm into the localization system to improve the positioning estimate.

The prototype communicates with the Razor IMU over USB using the Raspberry Pi which monitors a serial port where the IMU data is streamed to. The Raspberry Pi sends ASCII characters to the IMU's on-board processor to control what information is being recording as well as when to stop and start the recording. A reference for all of the capabilities of the sensor can be found in [34]. This IMU can output the raw data from each sensor as well as the quaternions or Euler angles from the fusion of all three sensor readings. The localization system just uses the raw data from the three accelerometers and three gyroscopes and ignores the magnetometer data. The IMU outputs the raw sensor readings from all nine sensors which the Raspberry Pi stores in a text file in the format shown in Figure 2.5. The first column is the IMU system time in milliseconds; columns two through four are the accelerometer data in g's; columns five through seven are the gyroscope data in degrees per second; columns eight through ten are the magnetometer data in micro-Teslas; and the last column contains the Raspberry Pi's time stamps when the reading was received, in seconds.



```
1477637, 0.11, -0.95, -0.30, -4.70, 57.68, 20.55, 66.47, -36.76, -28.51, 0.9743096828460693
1477647, 0.09, -0.95, -0.31, -4.45, 57.20, 20.43, 66.47, -38.56, -29.56, 0.9845607280731201
1477658, 0.10, -0.93, -0.30, -4.02, 56.34, 20.12, 66.02, -37.36, -28.36, 0.9948241710662842
1477668, 0.10, -0.95, -0.33, -3.78, 55.37, 19.88, 67.22, -37.06, -29.56, 1.00508713722229
1477678, 0.09, -0.90, -0.33, -3.48, 54.94, 19.76, 66.47, -36.01, -26.71, 1.0153238773345947
1477688, 0.09, -0.94, -0.32, -3.29, 53.29, 19.51, 66.62, -36.61, -28.36, 1.025562047958374
1477699, 0.09, -0.94, -0.33, -2.93, 52.26, 19.33, 66.77, -36.76, -28.81, 1.03582763671875
1477709, 0.11, -0.94, -0.29, -2.80, 51.04, 18.84, 66.77, -37.06, -28.21, 1.0460739135742188
1477719, 0.12, -0.94, -0.32, -2.87, 49.63, 18.54, 66.17, -36.76, -28.21, 1.056335210800171
```

Figure 2.5: Sample IMU Data. Column 1 is the IMU time in milliseconds. Columns 2-4 are the accelerometer data in g's . Columns 5-7 are the gyroscope data in degrees per second. Columns 7-9 are the magnetometer data in micro-Teslas. Column 10 is the Raspberry Pi's time in seconds.

In this prototype, the IMU's accelerometers are used as a step counter and the gyroscopes are used to track heading. Together, these two pieces of information allow the localization algorithm to dead reckon the user. Currently, this dead reckoning only works when miners are moving in the mine on-foot. This solution will not work if the miners are traveling in a vehicle, however this dead reckoning system could be replaced with a vehicular dead reckoning system for this case. Vehicular dead reckoning systems are typically easier and more accurate to implement as vehicles have fewer degrees of freedom of movement than a person and also have more sensors, such as encoders on the wheel that can give distance estimates. It is well known, [35], that IMU dead reckoning accumulates errors very quickly and different techniques are used in the sensor community to reign in these errors. The localization system presented relies on the RFID data to accomplish this.

2.3 RFID Reader

The prototype system uses two ThingMagic Mercury M6e development boards as readers to communicate with the RFID tags. One of these readers is shown in Figure 2.6. These devices have a maximum output power of 30 dBm (decibels-milliwatt) but this can be set lower if the power is too high. Each of these readers can connect to two transmit/receive antennas but from extensive testing, using one reader to operate using two antennas is not as reliable as using one reader for each antenna. These readers are controlled via Raspberry Pi over a USB connection but the power is supplied using the power supply circuit as they require a significant amount of power to operate. The Raspberry Pi uses the Python library provided in [36] to setup the readers as well as to extract data from them. The operation of the Python functions is discussed in section 2.8.1.

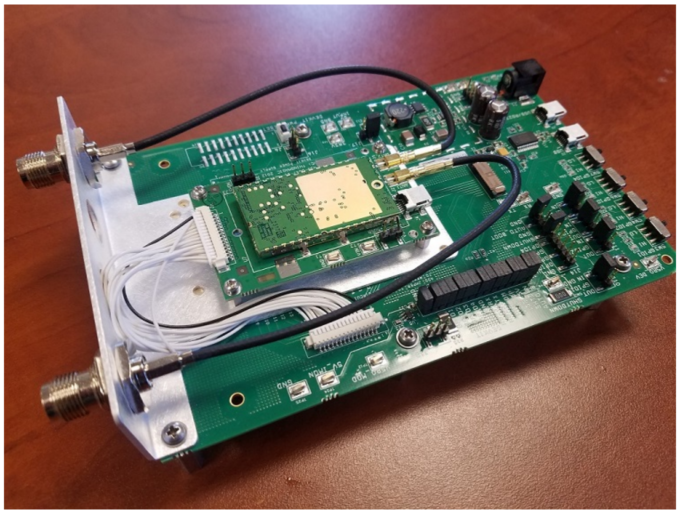
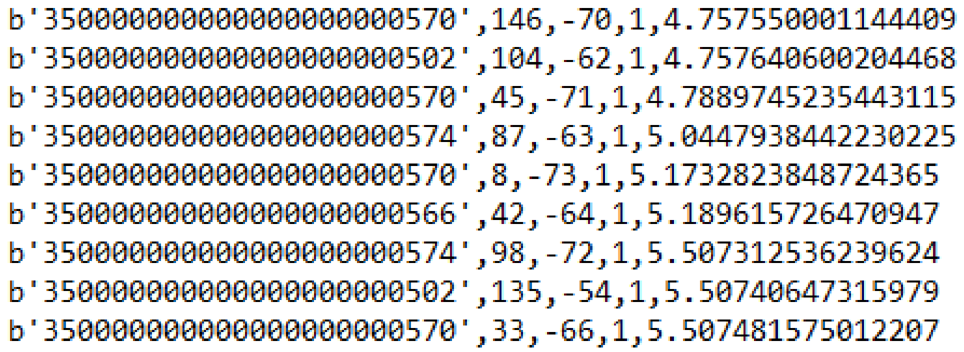


Figure 2.6: ThingMagic M6e Reader development board.

Typically, RFID readers must conform to the frequency band of the region as well as hop its operating frequency continuously. The readers in this prototype are locked to 915 MHz to ensure that both the transmit power and received signal phase are not changed by the frequency hopping, only by the tag reading changes in the environment. This is an acceptable solution only inside of a mine. Outside of the mine, fixing the frequency violates

FCC regulations as readers must frequency hop in order to not interfere with other readers. Per the Electronic Code of Federal Regulations Title 47, Chapter 1, Part 15 Subpart C, 15.211, we are allowed to setup radio systems underground that can operate at whatever frequency deemed necessary, as long as the signal is contained in the mine.

A snippet of the output data from the reader is shown in Figure 2.7 for a series of tag reads. The first column is the tag ID; the second column is the phase of the received signal in degrees; the third column is the received signal strength (RSSi) from the tag read in dBm; the fourth column is the reader number that received the signal (whether the right antenna reader or left antenna reader detected the tag); and the fifth column is the time stamp of the Raspberry Pi system time when the reading occurred in seconds. The RFID tags used in all experiments had very simple tag IDs where only the last three digits of the ID would change based on which tag it was. This made post-processing the RFID data easy as only the last three digits of the tag ID were all the localization algorithm needed to categorize the tag.



```
b'350000000000000000000570',146,-70,1,4.757550001144409
b'350000000000000000000502',104,-62,1,4.757640600204468
b'350000000000000000000570',45,-71,1,4.7889745235443115
b'350000000000000000000574',87,-63,1,5.0447938442230225
b'350000000000000000000570',8,-73,1,5.1732823848724365
b'350000000000000000000566',42,-64,1,5.189615726470947
b'350000000000000000000574',98,-72,1,5.507312536239624
b'350000000000000000000502',135,-54,1,5.50740647315979
b'350000000000000000000570',33,-66,1,5.507481575012207
```

Figure 2.7: RFID data from one reader. Column 1 is the tag ID. Column 2 is the phase of the received tag response in degrees. Column 3 is the RSSi of the received tag response in dBm. Column 4 is the reader number. Column 5 is the Raspberry Pi's time in seconds.

2.4 RFID Tags

The RFID tags are not located on the user but rather on the walls of the mine. These passive RFID tags are placed in clusters throughout the mine to demarcate the start of a zone as well as provide the initial position of the user as they pass into a zone. Figure 2.8 shows a picture of one of the RFID tags that was used in the mine. All of the tags that were placed in Edgar mine were oriented vertically. This tag is a simple linearly polarized tag that operates at 915 MHz and lists its tag ID on the face of it. The radiation pattern of the RFID tag is roughly that of a dipole making it heavily influenced by objects in its radial vicinity but not at the top and bottom of it due to the null in the pattern. If these tags were placed horizontally, the localization system would not be able to communicate with the tag if the user passed into the nulls of the tag. The vertical orientation ensures that the user always has the possibility of communication no matter the radial they are from the tag. The drawback of the vertical orientation is that it is influenced by the walls of the mine which reside well in the near-field of the antenna. The offset of the tag from the wall creates constructive or destructive interference which could be different for every azimuth angle. The tags are attached to small medium-density fibreboard planks to provide structural integrity and are attached to the walls of the mine using 1/4" anchors.

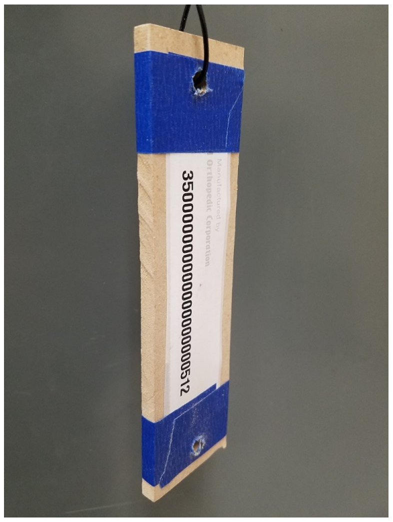
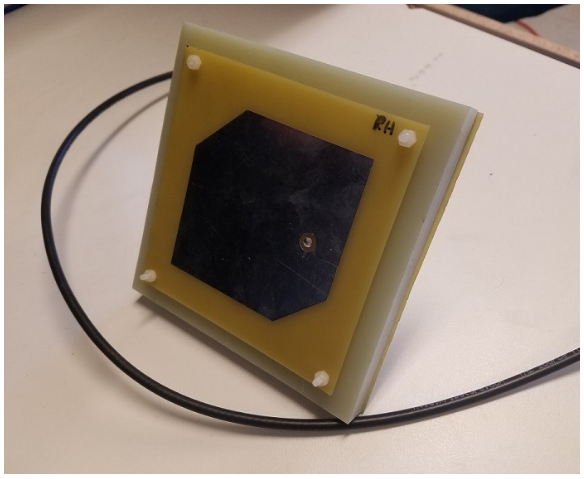


Figure 2.8: Linearly polarized RFID tag with thin medium-density fibreboard (MDF) backing.

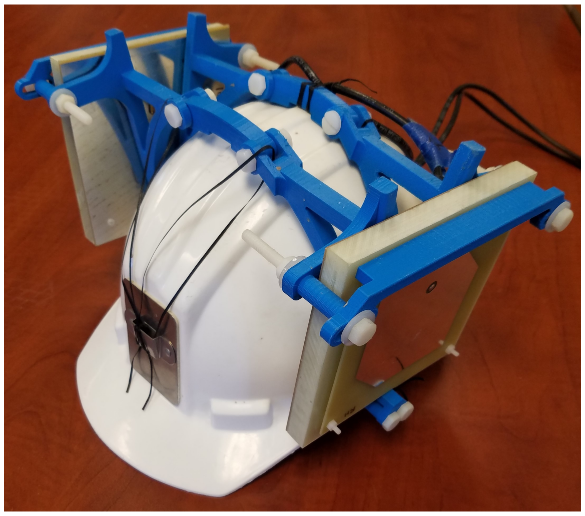
There are two types of RFID tags: active and passive. Active tags can be detected over much longer distances than passive tags however they must require recharge after a year if they use a battery or they require additional infrastructure in the environment to provide power. Passive tags were chosen for the localization system as large read ranges were unnecessary, the tags were inexpensive in large quantities, they do not have an expiration, and they did not need any additional infrastructure to operate. Both tag types have a chip connected to an antenna that transmit their IDs when excited. The most important metric for passive RFID tags is the received signal strength from the reader which is a combination of path loss, polarization, mismatch between the tag's chip and antenna, tag antenna gain, and efficiency. The assumption for passive tags is that as long as the tag has enough received power from the query signal to operate, the reader will have a sensitive enough receiver to detect the tag's transmitted response. Tag-to-tag collision is unlikely for clusters of passive RFID tags as they generate a random time offset on receipt of the query signal that is applied before they transmit their ID, ensuring that they are not communicating at the same time as another tag. Multi-path reflections in the mine could still cause a tag to interfere with itself or other tags.

2.5 Circularly Polarized Antennas and Mounts

The RFID antenna originally used was the Alien ALR-9611-CR antenna which has about 6 dB of gain and is circularly polarized but was too large to be carried by the user. During the prototype development, other antennas that were tried either did not have enough gain, were poorly designed (efficiency and mismatch losses), or were too large. The solution was to design a custom antenna that balanced size and gain that would also work when placed on a hardhat. These antennas were designed by Joseph Diener and had both right-hand circularly polarized (RHCP) versions and left-hand circularly polarized versions (LHCP). Figure 2.9(a) shows the fabricated antennas. All fabricated antennas had gains that were above 5dB, giving it a slightly less read range than the Alien antenna but at about half the size. This allowed for an antenna to be placed on each side of the hardhat, giving around 11-13 feet of read range on either side of the user. Figure 2.9(b) shows the antennas mounted to the hardhat. The final prototype uses the antennas that are right-hand circularly polarized (RHCP) which means that the the electric and magnetic fields rotate in a circular pattern as the wave propagates out of the antenna and can receive signals that are vertical polarized, horizontally polarized, or circularly polarized. The downside to a circularly polarized antenna is that when the RHCP signals are received by a linearly polarized tag, either in the vertical plane or the horizontal plane, half of the power (3dB) is lost. This is also true for the inverse; the tags emit a linearly polarized signal that suffers a 3dB loss when received by a RHCP antenna or LHCP antenna. Future work will be to develop circularly polarized RFID tags which will not accrue the half power signal loss. Even with this half power loss, the read range is acceptable as the width of the mine is around 8 to 10 feet.



(a) Right-hand circularly polarized RFID antenna



(b) Two RFID antennas mounted on helmet

Figure 2.9: RFID antennas and the 3D printed mount for attachment to the hardhat.

2.6 Power Circuit

The developed prototype uses a large 12 Volt battery pack to supply power to all components of the system. The Raspberry Pi and RFID readers require different input voltages in order to function. At first glance, it is tempting to just use a simple voltage divider circuit to provide the correct voltages for each of the devices, but the voltage divider circuit assumes that the battery voltage will always be 12V. As the battery's power drains, the linearly divided voltages will also be lower which will cause all of the loads to fail. This prototype uses a universal battery eliminator circuit (UBEC) and a DC-DC converter to maintain the correct output voltage even as the battery's power decreases. The UBEC and DC-DC converters also maintain the correct voltage even when the load changes. For example, the RFID reader draws much less current when in a sleep state rather than when it is reading tags. Though both of these circuits have different names, they both function as voltage regulators. Figure 2.10 shows the power supply circuit in the prototype and Figure 2.11 shows electrical schematic for the entire system. This schematic does not take into account the RF cable connections from the RFID readers to the antennas.

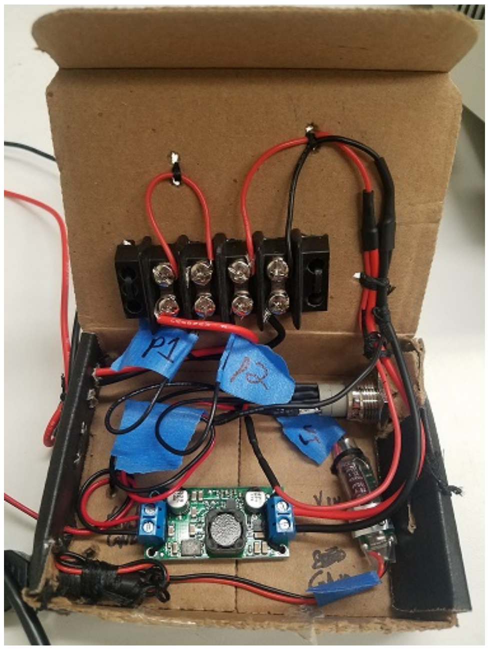


Figure 2.10: Fabricated power supply circuit which contains the UBEC and the DC-to-DC converter as well as the toggle switch for turning on the device.

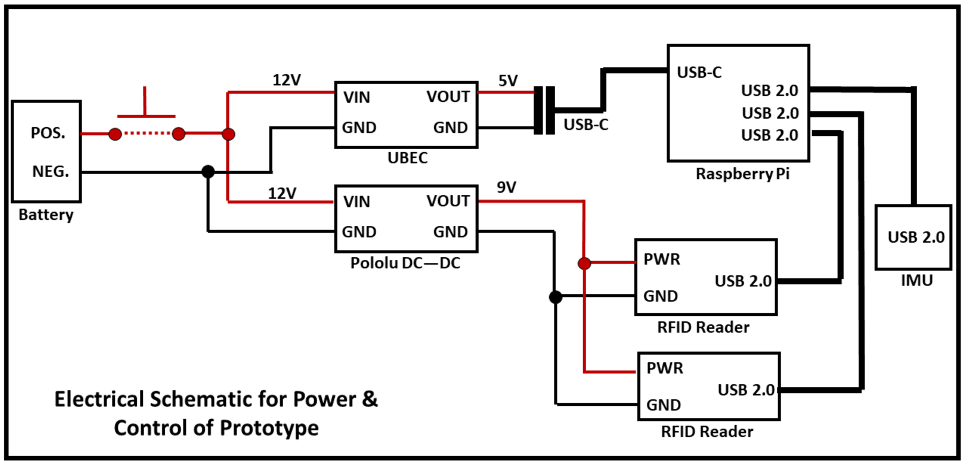


Figure 2.11: Electrical schematic for the power supply to all devices as well as the USB connections required to control all devices.

Table 2.1 lists the power consumed by each of the devices connected to the power circuit. The DC-DC converter supplies a total of 13.4 Watts to the readers during a measurement and with an efficiency of 90%, the DC-DC converter only draws 1.25 Amps from the 12V battery [37]. The UBEC supplies about a total of 4.05 Watts to all of the devices connected to it. The efficiency on the UBEC is 90% when 12V are supplied to it resulting in a consumption of 0.375 Amps from the battery [38]. Between both of these voltage regulators, a total of 1.625 Amps are drawn from the battery during a measurement. The battery used in this prototype has a capacity of 7 Amp-Hours so the entire system can take continuous measurements for about 4 hours before it needs to be recharged.

Table 2.1: Power consumption, voltage, and current of the devices supplied by the power circuit.

Device	Power (W)	Voltage (V)	Current (A)
RFID Reader 1	6.7	9	0.75
RFID Reader 2	6.7	9	0.75
IMU	0.05	5	0.01
Raspberry Pi	3	5	0.6
Touchscreen	1	5	0.2

Once the correct voltages are supplied to the readers and the Raspberry Pi, the IMU and any other peripheral equipment such as keyboards, touchscreens, or mice will have their power supplied via the USB connection to the Raspberry Pi.

2.7 Raspberry Pi and Bash Scripting

A Raspberry Pi 3 B+ is the central component of the localization prototype. The Raspberry Pi is a full computer with a 4-core processor, 1 GB of RAM, an HDMI port, wireless LAN capability, SD card for data storage, and 4 USB ports. The Raspberry Pi is connected to a touchscreen so the user can open code and manage files while inside of the mine. A microcontroller could have been used instead of the Raspberry Pi for controlling the IMU as it has I2C communication capability, however it would have been very hard to interface with the M6e reader using a microcontroller. The RFID readers already had a Python library written for them and were designed to be communicated with over USB which made the Raspberry Pi the most logical choice for a central controller. A benefit of using the Pi is that it contained an on-board integrated development environment (IDE) which was used to develop all of the acquisition code sequentially as more devices were added to the system without the need for another computer to run a compiler. The flexibility of the Raspberry Pi also allows for running different codes on the fly, such as reading one RFID reader to generate the look-up-table or just running the IMU. The WiFi capability of the Raspberry Pi made transferring data to the post-processing algorithm on a host machine very easy. When the Raspberry Pi is inside a wireless network and headless (without a touchscreen, mouse, keyboard, etc.), the host machine can connect to the Raspberry Pi via a virtual private network (VPN) connection to set it up or to extract data from it. However, if no wireless network is present, a USB flash drive could be used instead to transfer data.

The Raspberry Pi's 4-core processor is instrumental for the prototype as it could run the IMU and both readers in parallel in the background of the Raspberry Pi. The control functions for the IMU and RFID are both written in Python 3, however the main data acquisition code is a bash script called from the terminal of the Raspberry Pi that runs all of the Python functions. The Python multiprocessing library was the first attempt at running each of the readers and IMU in parallel but the RFID reader control code was not intended to have two readers running in parallel in the same program so it continuously failed. Running the functions for each device in the background of the Raspberry Pi was the only solution that allowed for parallel operation. The added benefit of running each of the devices in the background is that if one device errors out, it does not affect the other ones.

2.8 Acquisition Code

The data acquisition codes for the IMU and RFID readers are developed using Python 3 as functions that are called by a bash script from the terminal of the Raspberry Pi. This bash script runs all functions for a specified amount of time that is set by the user in the terminal. Listing B.1 contains the bash script code. Since each function is running in the background of Raspberry Pi, it will appear as though nothing happened after executing the script but each of the readers and IMU will be recording data. Once the functions are called, they will never respond back to the bash script and are each responsible for saving their own data files. The data recording will stop when each of the devices hits the time limit as input by the user on execution.

The common time base, to sync all of the devices, is the time since Epoch (January 1st, 1970). At the beginning of execution the bash script takes the time since Epoch and passes it into each of the IMU and reader functions as a constant. Inside each of the functions, they calculate the current time since Epoch and subtract the difference from the inputted constant to get the run time. This ensures that all devices are synced to the Raspberry Pi's system time so that their respective data are on a common time base. Also, each device will have a different initialization time at the start of execution so the common time base ensures that the data time can be compared across all devices without knowing the initialization time offset.

The bash script passes into the reader functions the serial ports the readers are connected to on the Raspberry Pi. If these ports change, then the terminal command: *ls -l /dev/tty** will list the ports that the devices are connected to. Knowing the new ports, the user will then need to update the Python function calls within the bash script the new ports.

2.8.1 RFID Reader Control

The Mercury M6e readers are designed to work with the Mercury API software. In [36] a library is created using Python code to interface with the readers. The RFID reader function uses this Python library to control the two used readers and is provided in Listing B.2. The function takes in four arguments from the bash script: the USB port number the reader is attached to, the reader port that the antenna is connected to, the start time of the bash script, and the duration the reader needs to record data for. The USB port number is the last number of the port name after running *ls -l /dev/tty** which will look like */dev/ttyUSB0* for reader 0 and */dev/ttyUSB1* for reader 1. The antenna port corresponds to the numbered port on the reader that the antenna is connected to. It is important to note that for the two readers configuration, each reader should be connected to the antenna on a different port than the other reader because the control function will query each reader to report back which port the antenna is connected to and save it in the output data. If they are on the same port, then it will appear that only one reader's data are being recorded. The start time is passed in to the function in order to sync the data acquisition with the Raspberry Pi's system time. The measurement run time corresponds to how long the reader will be allowed to read.

The reader control function relies heavily on the Python library from [36] which abstracts away many of the complex tasks that are typical for data acquisition scripts. The reader control function starts by initializing a reader object using the input parameters passed in from the bash script as well as some user defined settings that are unlikely to change between measurements. The initialization sets the output power to 30dBm (3000 centidBm in the code), locks the frequency to 915 MHz (915000 kHz in the code), sets the region to North America 'NA', and sets the read plan to 'GEN2'. The function also initializes an empty string array which will contain the data for the function. After the reader object is initialized, the function defines a callback function which is a function that executes every time its condition occurs, similar to an interrupt service routine on a microcontroller. The callback function is called whenever a new tag is read by the reader which is accessed from the reader's memory via a tag object. The code within the callback function pulls apart the received tag object, saving the tag ID, phase, RSSi, antenna number, and time stamp as a string of characters. The string of characters for each tag read is appended as a new line in the data array. This function does not use any loops, instead all it does to control the reader is to tell it to start reading, wait for the amount of time specified by the bash script run time input, and then stop reading. Over the duration of the wait time, the callback function will trigger everytime there is a new tag read. The start reading command is passed in the callback function defined earlier. After the reader receives the command to stop reading, the function takes the array of data and writes them to a text file.

2.8.2 IMU Control

Once the IMU is programmed with the Arduino library provided in [28] and according to the guide in [34], it can now be controlled over a USB serial connection to the Raspberry Pi. The Arduino code onboard the IMU takes in text characters as an input to either start or stop the data recording, change settings, or to change the IMU output data. The Python function called by the bash script is provided in Listing B.3. This function assumes that the IMU always connects to port: */dev/ttyACM0* which is almost always the case. If the port changes after running *ls -l /dev/tty**, then it will need to be updated within the function. The bash script passes the initial start time and the run time into the function at the start of execution. The output of the function is a text file where every line is an IMU sample.

The IMU control function first starts by creating a serial object, setting the baud rate and packet information. The function then initializes an empty string array, a counter and timing information. To start the IMU data recording, the IMU function sends an ASCII encoded space character over the serial port to the IMU which then detects it and starts the stream of data. The IMU control function will also send another space at the end of the program to stop data recording. This space is a toggle switch to the IMU to turn on or off the data recording. Sometimes, the IMU read function on the Raspberry Pi will end before the data stream can be toggled off (i.e. if the power is shut off mid-program) which will cause future executions of the read function to toggle off the IMU data stream at the start. If the IMU output file after a measurement only has 5 or less entries, this means that the toggle switch is off and the function will keep turning the IMU data stream off at the beginning of the program and on at the end. This can be fixed by sending another space over the serial port using the IDLE IDE terminal window before starting the next measurement. After starting the data stream from the IMU, the IMU control function uses a while loop to save the data stream to the output string array incrementing a counter every loop iteration to advance the row index of the output array. The loop ends when the desired run time is reached. After the stop condition occurs and the function turns off the data stream, it writes the contents of the string array to a text file.

2.9 Conclusion

The M6e readers generated numerous challenges during the entirety of the prototype development. Originally, one reader was used to communicate using both antennas but was highly susceptible to erroring out midway through a measurement. When the reader throws an error, it terminates the Python control function with no error message, just a 'Restart Shell' message that also shows when the program is first run. After numerous measurements, the error appears to be due to the transmit power levels, the received power levels, and the use of the two ports on the same reader. When the reader transmitted at 30dBm, it would consistently error out when 5 feet or less away from an RFID tag. If the reader's transmit power was dropped to 21 dBm instead of 30 dBm, the reader was much more resilient though at the cost of a greatly reduced read range. The current hypothesis on why these errors occur is that the temperature of the reader could have hit a threshold during continuous measurement at maximum power, or that the received power was too high. The implementation in [39] indicates that the device is designed to error out and shutdown when it hits a *FAULT TEMPERATURE EXCEED LIMITS 504h* error or a

FAULT POOR RETURN LOSS 505h error which lends support to the current hypothesis. The *504h* error occurs when the device has overheated, which is consistent with the observation that waiting an hour after the first error would be enough to get the device operating consistently again. This was also consistent with the observation that using only one antenna on one reader would be more resilient than two antennas connected to one reader. Perhaps one reader with two antennas generates additional thermal strain than only the one antenna case. The *505h* error happens when too much power is coming back into the M6e device. Two possible things could have caused this: an impedance mismatch on the output ports of the reader which would reflect back a large portion of the 30dBm back into the device, or being too close to the tags resulted in a received signal that was too strong for the reader to handle. The first scenario is possible as both antennas were connected to right angle connectors and would sometimes become damaged or loose, which would result in a significant impedance mismatch. This was the reason for one of the failed measurements inside of Brown building when one of the antenna's right angle connectors was torqued incorrectly and the center pin broke. This also makes sense that reducing the output power would cause the device to function properly again because the reflected signal would also be of lower power. If mismatch is not coming from the right angle connectors then it could be some sort of proximity loading on the highly directive antennas that would change the input impedance as well. The second reason (being too close to the tags) is consistent with some of the Edgar mine measurements when the user walked within 1 foot of the tags in the narrow mine shaft and the reader immediately stopped working. Both of these reasons are consistent with the observations that reducing power and or adding distance between the tag and the reader reduces the likelihood of getting errors. The error could also be a combination of both temperature and too much received power but it is hard to discern which one as the reader never sends an explicit error message to the Raspberry Pi, only a restart shell trigger in the Python terminal. Since adding more distance between the user and the tags is not possible for narrow corridors and 30 dBm is needed for the large intersections, the solution was to use two readers each transmitting 30 dBm which would operate in parallel. This greatly increased the reliability of the prototype, however the reset error would still occur when in very close proximity to a tag for an extended period of time at 30 dBm. Future work will be to use either a different RFID reader or use software-defined-radios (SDR) to control and record specified RFID data.

This chapter discussed the prototype hardware as well as the data acquisition code used to control all of the devices. The last chapter of this thesis will discuss the hardware improvements planned for the future update of this system.

CHAPTER 3
SOFTWARE

3.1 Overview

The majority of the work done for this thesis lies in the software. This chapter describes in detail all of the major pre-processing steps and functions as well as all of the major post-processing functions. The entire code base is listed in the appendix. This chapter will first describe the pre-processing steps and functions which are: step-length estimation, look-up-table generation, zone information file generation, zone connection file generation, the map approximation of the environment, and data interleaving. The post-processing functions are: RFID categorization, IMU zone assignment, RFID 2D position calculation, IMU pedometer and heading calculations, 2D positioning, map matching, and plotting. Throughout this chapter two data sets will be used interchangeably to visualize the operation of each function; one data set taken in Brown building and the other taken in Edgar experimental mine. See chapter 4 for a description of each of these experiments.

3.2 Pre-processing Measurements and Scripts

Before conducting a localization measurement, the user must setup the environment. For the Edgar mine experiment, these steps took a total of 25 hours which is broken down in Table 3.1. The steps to setup the environment are as follows:

- Install all RFID tags in the environment, recording their IDs, coordinates, and which zone they belong to.
- Create the map of the environment by recording the center coordinates of all intersections or changes to the path heading. These coordinates will mark the starts and stop of the lines that will approximate a zone. These coordinates are based on a user-defined X and Y axes and measured values are in feet, that all points are based

off of. For example, in the Edgar mine the long tunnel running through the center of the mine was used as the Y axis and one of the side branches was used as the X axis.

- Estimate the user's gait length by counting the user's steps along a known distance.
- Measure the RSSi with a single antenna from known distances. Conduct many of these tests in different parts of the environment to build up enough data for the RFID reading look-up-table. It is very important to do this step with the reader set to the same power and frequency that will be used in the localization measurements. This will be discussed in detail in section 3.2.1.

Table 3.1: Time breakdown of the pre-characterization steps for the Edgar mine experiment. The measurement time is also included.

Task	**Time (hours)**
Tag Setup - 78 tags	20
RFID Characterization Measurements	3
Line Map Generation	1
Gait Length Estimation	1
Measurement Time	2

After these steps are accomplished, the user can then run the setup information through the pre-characterization scripts. These scripts take in the user input and output files that the post-processing script can use. Figure 3.1 shows the flow diagram of the various pre-processing scripts, their inputs, and outputs. The scripts that are run once are the look-up-table generator script, the tag/zone info script, the zone connection script, and the map generator script. The look-up-table script takes in the RSSi versus distance measurements of various tags in diverse scenarios and outputs a polynomial that the localization code uses to convert RSSi to distance. The tag/zone info script takes in the user's recorded tag coordinates, IDs, and which zone they are in along with some other information about the zone, and outputs a file that contains information for each zone that the localization code can interpret. The zone connection script takes in user input about which zones connect to each other as well as the average heading the user must enter the zone in on, knowing the zone that the user left. This is described in section 3.2.3. The zone connection script also contains the approximate initial coordinate the user will enter a zone in just in case the RFID 2D localization fails. The RFID 2D localization fails when the user traverses a zone and does not read at least three tags in a specified time window. Finally, the map generator script takes in all of the coordinates of where the path changes in the environment and creates an array of lines that are used in map matching. The only pre-characterization script that needs to be run after every measurement is the RFID data interleaver which takes in the data files generated by both RFID readers and combines them into a single RFID data file. If only one reader is used with two antennas, then this script is not needed.

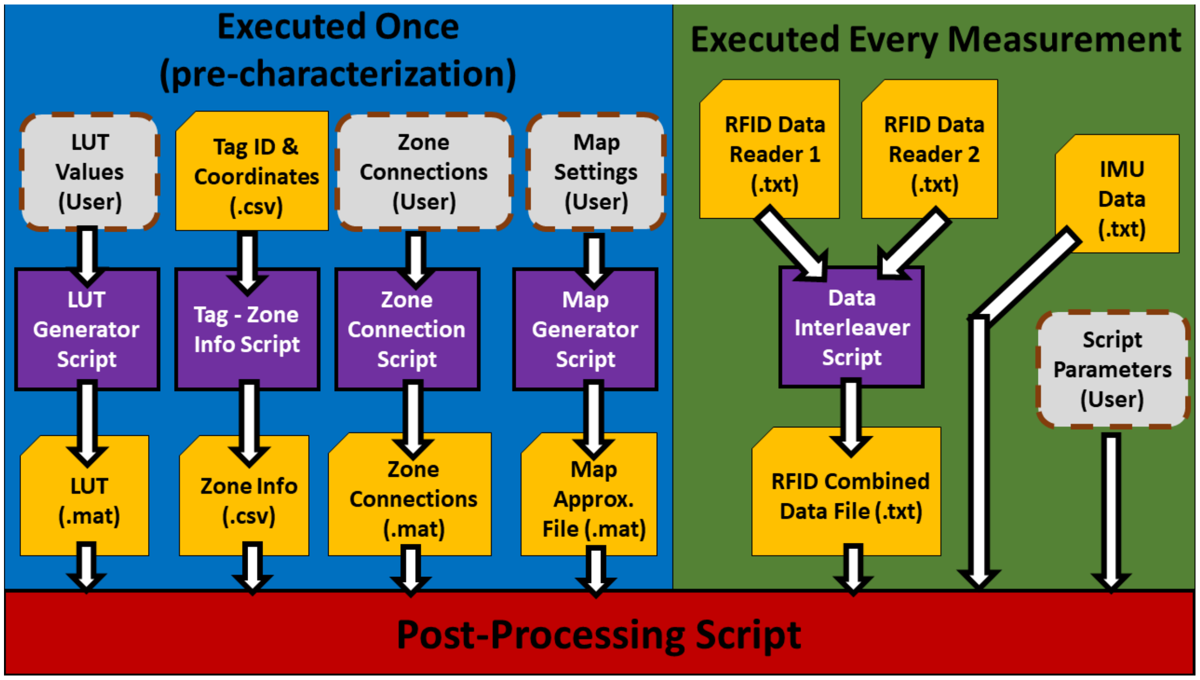


Figure 3.1: Flow diagram of all of the pre-processing scripts and files that feed into the post-processing script.

The main reason why the one-time execution scripts are separated from each other is to let the user precisely change the environment setting without having to redo all of the other settings. For instance, if the user changes the reader power or frequency then only the look-up-table needs to be re-run, not the tag information, zone connections, or environment map approximation. If tags are added to or removed from a zone then the user just needs to re-run the zone info script. The following sections describe each of the pre-processing steps in detail.

3.2.1 Look-Up-Table Generation

A look-up-table (LUT) is a pre-determined set of measurements that are used to convert RSSi values to distance. For this prototype, the LUT is a third order polynomial derived from measured data. Generating a look-up-table is, in general, an easy task, however generating a look-up-table that will work for every scenario in a complex environment that changes as the user progresses is difficult. For instance, in Edgar mine the user will encounter narrow corridors with low ceilings, and large intersections with far away tags and numerous objects in the zone to scatter off of. Sometimes the antenna will only read tags that are off-angle of the antenna's broadside which will have a lower RSSi compared to when it receives them directly in its line of sight. The multipath nature of the mine will greatly skew the received RSSi of signal making it lower after each reflection which translates to a much further distance away from the user than it actually is. One solution is to build up a LUT for each zone in order to most accurately predicate the distances the user will encounter. In a very large environment with many zones this is impractical. Another solution is to take measurements from different scenarios, combine the data into one dataset, and then fit a curve that will work for the average of all scenarios. This is what was done for this system.

The data for the look-up-table is composed of three measurement scenarios inside of Edgar mine. The first scenario is a head-on measurement at a T intersection where the antenna was moved progressively backwards at a 45 degree angle. This scenario simulates the large open-area cavities in the mine where the user will be turning in and will most likely receive the tags when they are in the broadside of the antenna for a brief moment. When they are in the broadside of the measurement, the user will rarely be receiving them at a perfect 90 degree angle to the wall which is why the off-angle of 45 degree is used to generate the LUT. The tag was always kept in the direct broadside of the antenna. The vertically-polarized tag antenna was attached to the wall where it had rock behind it, above it, and on both sides but little rock on the bottom. Figure 3.2 shows a diagram of the measurement for the first scenario.

The second and third scenarios closely approximate the channel that the antennas would experience in the majority of the mine; a narrow corridor with antennas on either side of the user. For the patch antennas used, the center of the main radiation lobe is the direction normal to the face of the patch. In these scenarios, the user would be moving towards the tags and would start recording tag readings in the low gain portions of the antenna's radiation pattern. As the the user progresses the tag readings would come directly in the center of the main radiation lobe of the antenna, which has the highest gain, resulting in a higher RSSi. As the user traveled out of the zone, the tag readings would be on the outside of the center of main radiation lobe and would again have lower gain. The second scenario exemplifies when the tag is on the right side of the user and the user is closer to the opposite wall. The third scenario is the same as the second scenario except that the tag is on the left side of the user and the user is a foot closer to the tag. Scenario two is illustrated in Figure 3.3 and scenario three is illustrated in Figure 3.4. For each of these scenarios, the center of the antenna's main lobe was always facing directly at the wall and the antenna was held at head-level by hand. The antenna was then moved down the mine shaft from 0 feet to 3 feet in 1 foot increments. The tag signal was lost after 3 feet for scenario one but scenario 2 was able to still detect the tag signal at 3.5 feet so this value was also included. At the initial position (0 feet) the tag would be perpendicular to the antenna. The distance that is input into the script, for each RSSi value, is the line of sight distance from the center of the tag to the center of the antenna.

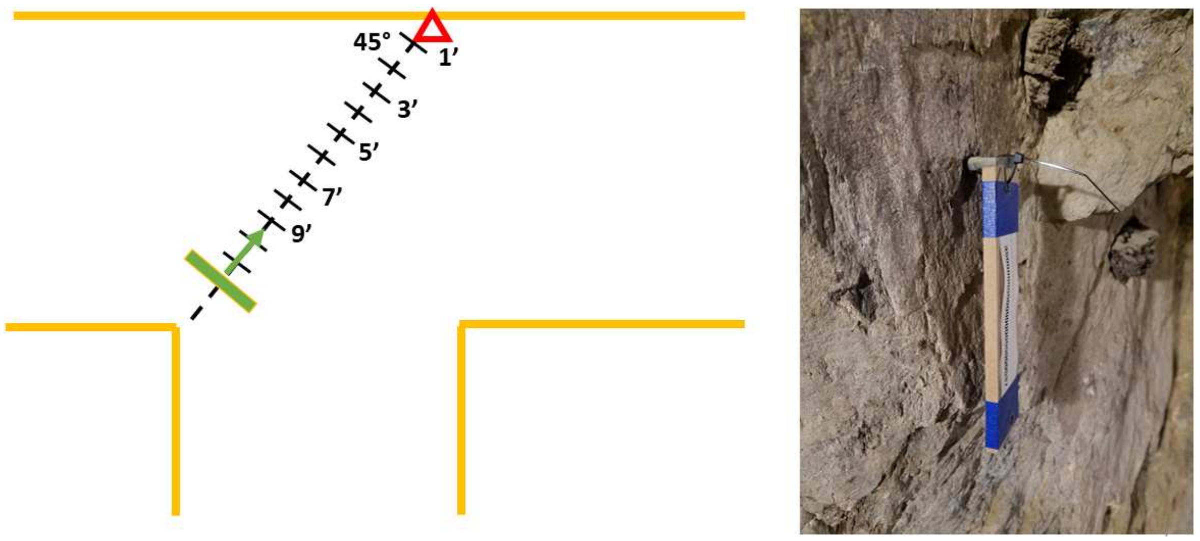


Figure 3.2: First measurement scenario to create data for the look-up-table. The diagram on the left shows the setup of the measurement. The right picture shows the tag attached to the wall for this measurement. The red triangle represent the RFID tag and the green rectangle represents the antenna.

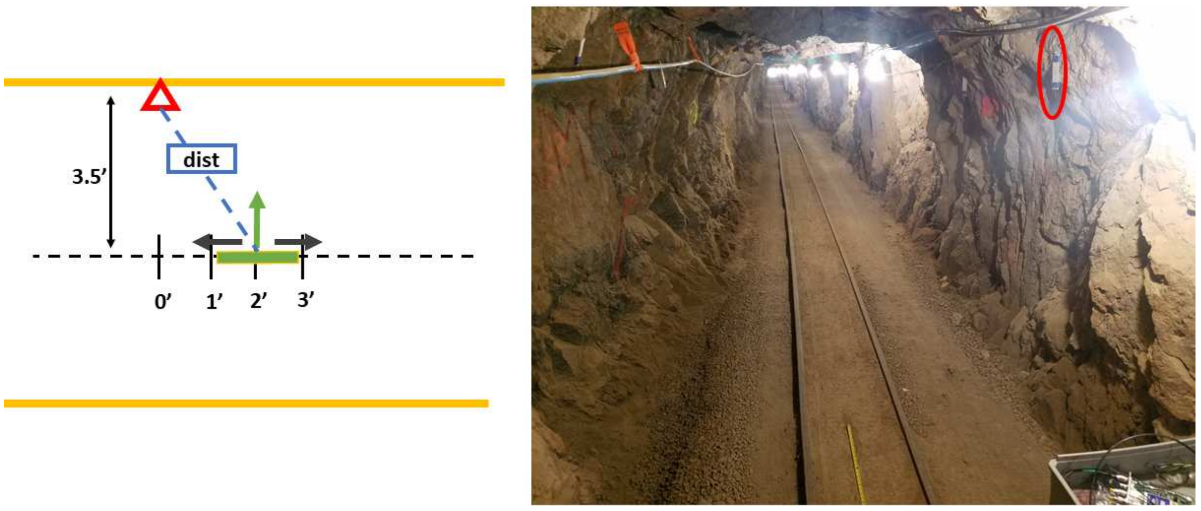
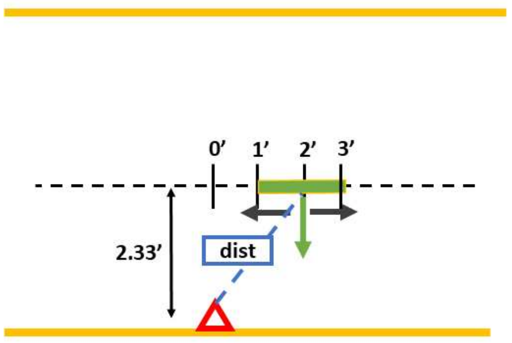


Figure 3.3: Second measurement scenario to create data for the look-up-table. The diagram on the left shows the setup of the measurement.The distance used for the look-up-table is the length from the center of the tag to the center of the antenna. The red triangle represents the RFID tag and the green rectangle represents the antenna. The right picture shows the location of the tag for this measurement. The red oval is the location of the tag under test.



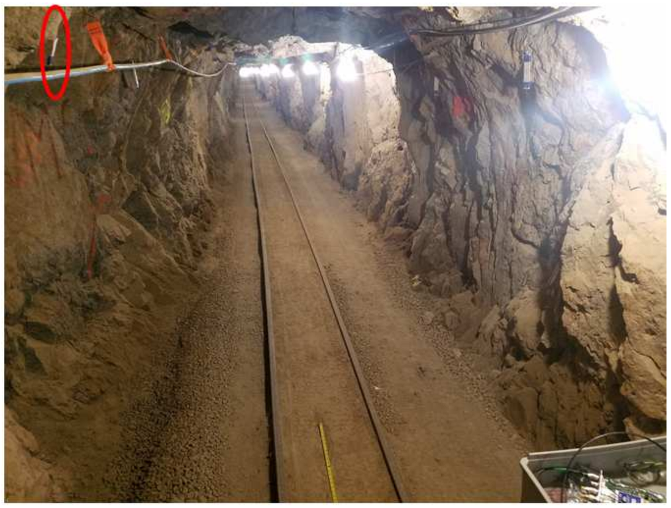


Figure 3.4: Third measurement scenario to create data for the look-up-table. The diagram on the left shows the setup of the measurement. The distance used for the look-up-table is the length from the center of the tag to the center of the antenna. The red triangle represents the RFID tag and the green rectangle represents the antenna. The right picture shows the location of the tag for this measurement. The red oval is the location of the tag under test.

The code used to generate the LUT is provided in Listing C.4. This code should be used as a guideline for the user as different scenarios will require different input values. In its current state, it does not take in any arbitrary data from a file like the other pre-processing scripts do. The input to this script is the distance and RSSi values for every measurement scenario. The codes takes these measurements values and creates a third order polynomial to the scattered data as well as generate the correct file that is needed for the input to the post-processing code. The output file contains the coefficients for the polynomial and their curve fit statistics, stored in a *cfit* object. The *fit* function and the resulting *cfit* object require the MATLAB curve fitting toolbox. Currently, the localization code assumes the LUT is a third order polynomial so any change from this polynomial by future researchers will require changes to the code in Listing C.4 and Listing C.11. Third order is the optimal order of polynomial to approximate the data as it has sharp roll-offs of distance at low RSSi values (< -70 dBm) and high RSSi values (> -55 dBm) but the distance values flatten out in the RSSi range between the high and low RSSi values. The third order polynomial is described by equation (3.1).

$$Distance = A * RSSi^3 + B * RSSi^2 + C * RSSi + D \tag{3.1}$$

where, $A = -0.000978$, $B = -0.168280$, $C = -9.824537$, $D = -191.201163$, are the coefficients obtained from Edgar mine pre-characterization data. $RSSi$ is in decibel-milliwatts (dBm), and $Distance$ is the distance from the user to a tag with units of feet.

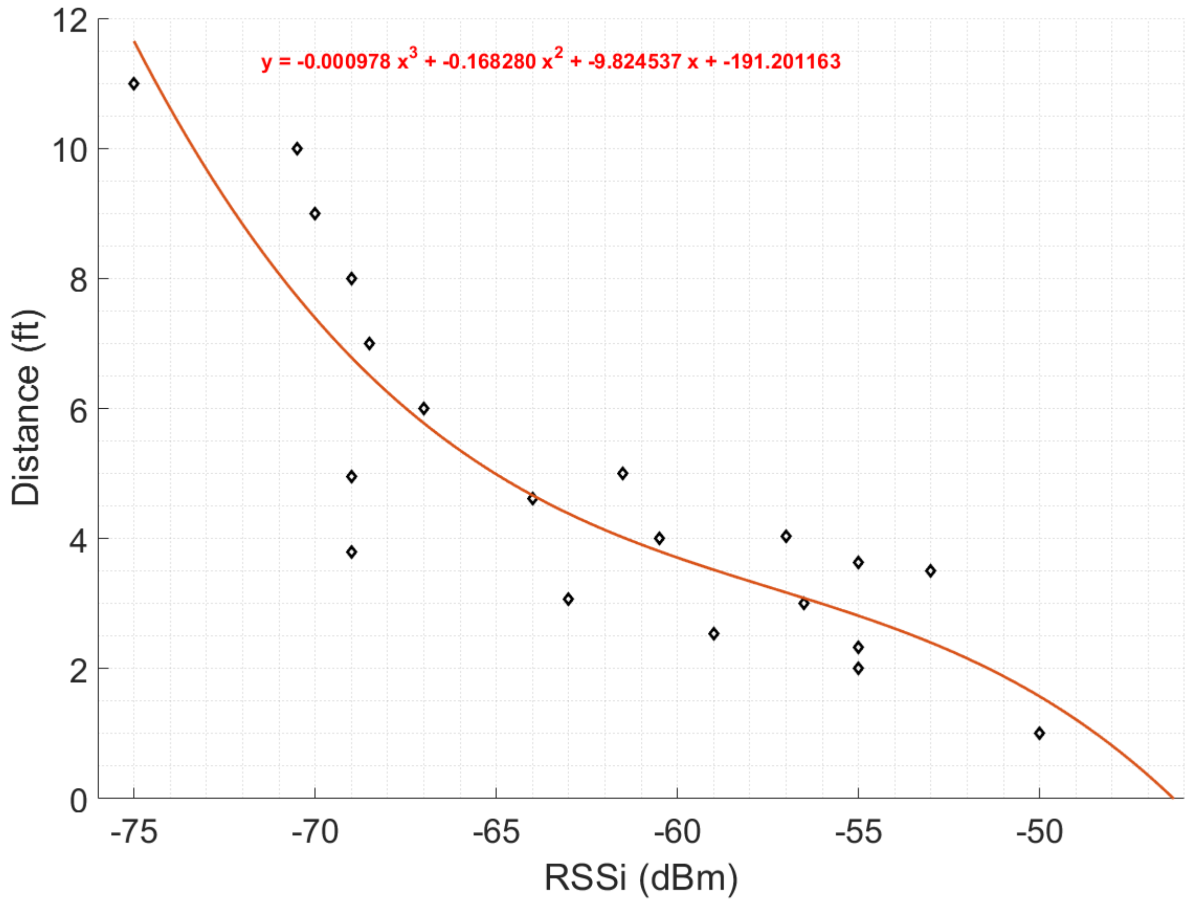


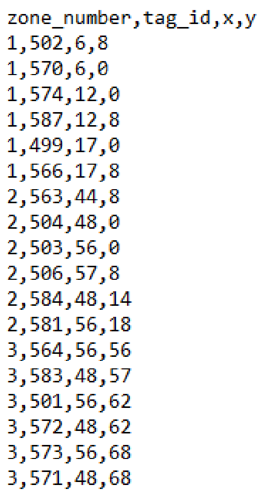
Figure 3.5: Look-up-table generated from the measured data in Edgar mine.

The look-up-table generated inside edgar mine is shown in Figure 3.5. The black diamonds correspond to the measurements taken from two different types of experiments inside of Edgar mine with the RHCP 1 antenna held at head level by hand. The MATLAB function *poly3* created the orange line that fit the data. The corresponding 3rd order polynomial is listed in red at the top of the plot. Typically, the distance determines the RSSi but the equation needed for the look-up-table is the inverse of this, hence why the RSSi is on the independent axis.

3.2.2 Zone Information

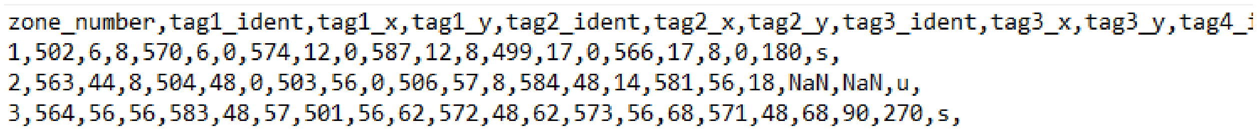
During the tag installation process, the user will record the tag ID, tag coordinates, and which zone the tag belongs to. The user will then need to create a comma separated values (CSV) file that puts each tag entry in the format of: *Zone, Tag ID, X Coord., Y Coord.* An example of this file is shown in Figure 3.6. The reason for using a CSV file is because they are human-readable and very easy to modify.

Once this file is created the user will run the code in Listing C.5 and can add additional information to the file such as what type of the zone each of them are as well as legacy information on the forward zone transition headings and the reverse zone transition headings. The output of this script is another CSV file which is shown in Figure 3.7. Besides the first row which contains the column header, each row corresponds to a zone. Each of these rows contains all of the tag information for the zone as well as the zone type and transition headings. In the Brown building measurements, the forward and reverse zone transition headings were used for the ’s’ type zone (’s’ meaning straight section) to correct the user’s headings. The ’u’ character at the end of the zone 2 entry means that zone 2 is an ’undefined’ type zone and no additional logic is applied to this zone in the localization code. This is because zone 2 is an intersection. If the user decides to apply different logic to each of the zones separately in the localization code, then other characters can also be used. Two simple headings are not enough to characterize all headings into and out of an intersection with more than 2 spokes, hence the zone connection script was created. These headings were kept around for consistency just in case the user decides to use the code to look at earlier Brown building data. This is why there are two ’NaN’ entries before the ’u’ character. The measurements performed in Edgar use the zone connection file to provide the initial headings into any type of zone.



```
zone_number,tag_id,x,y
1,502,6,8
1,570,6,0
1,574,12,0
1,587,12,8
1,499,17,0
1,566,17,8
2,563,44,8
2,504,48,0
2,503,56,0
2,506,57,8
2,584,48,14
2,581,56,18
3,564,56,56
3,583,48,57
3,501,56,62
3,572,48,62
3,573,56,68
3,571,48,68
```

Figure 3.6: Contents of the CSV file that is generated manually by the user. The first column is the zone number, the second column is the last three number of the tag ID, the third column is the X coordinate of the tag in feet, and the fourth column is the Y coordinate in feet. This file was used for the Brown building measurements.

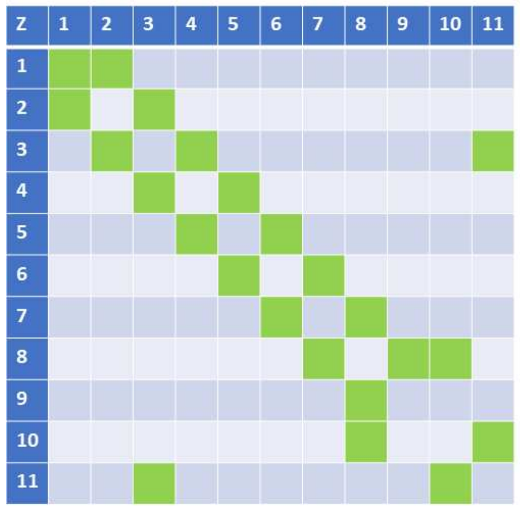


```
zone_number,tag1_ident,tag1_x,tag1_y,tag2_ident,tag2_x,tag2_y,tag3_ident,tag3_x,tag3_y,tag4_
1,502,6,8,570,6,0,574,12,0,587,12,8,499,17,0,566,17,8,0,180,s,
2,563,44,8,504,48,0,503,56,0,506,57,8,584,48,14,581,56,18,NaN,NaN,u,
3,564,56,56,583,48,57,501,56,62,572,48,62,573,56,68,571,48,68,90,270,s,
```

Figure 3.7: Contents of the CSV file that are output by the zone information generator script. The top row is clipped off due to the length going out of bounds of the screen. Each row corresponds to a different zone. This file was used for the Brown building measurements and is the output of the zone information script where Figure 3.6 was the input of. The 'u' at the end of the row starting with 2, says that the zone is of type 'undefined' which will have different logic applied to it in the localization script than the rows with the 's' at the end.

3.2.3 Zone Connection Generator

The heading calculation in the post-processing script requires an initial heading. Since each of the zones are isolated from each other, the end heading from the previous zone is not used as the initial zone heading for the current zone. This is done to eliminate the heading drift error that accumulates very quickly from the gyroscope integration. Instead, Listing C.6 can be used to specify the initial heading into a zone when the previous zone is known. This is implemented in the code using a 2D matrix of structs where zones that are connected together have an entry. If the zones are not connected together they do not have a valid entry and can be used by the localization code to see if the user skipped a zone. See Figure 3.8. Each valid entry contains the heading from zone i to zone j and the initial position in that zone just in case the initial position estimate from the 2D RFID fails. If zone i is connected to j then there will be a valid entry in the matrix for (i, j) and (j, i). Looking again at Figure 3.8, we see that the straight section (only one way in and one way out) zones: 1,2,4,5,6,7,9,10, and 11, only have two entries. Zone 1 is an exception as it is in the user's starting zone and contains a $(1, 1)$ entry. If the user starts in a different zone, the initial condition must be moved to (i, i) that it corresponds to. The intersection zones: 3 and 8, have three entries because they are each connected to three other zones. This zone connection map allows for very complex environments with large numbers of zones to be connected together simply.



Z	1	2	3	4	5	6	7	8	9	10	11
1	🟩	🟩									
2	🟩		🟩								
3		🟩		🟩							🟩
4			🟩		🟩						
5				🟩		🟩					
6					🟩		🟩				
7						🟩		🟩			
8							🟩		🟩	🟩	
9								🟩			
10								🟩			🟩
11			🟩							🟩	

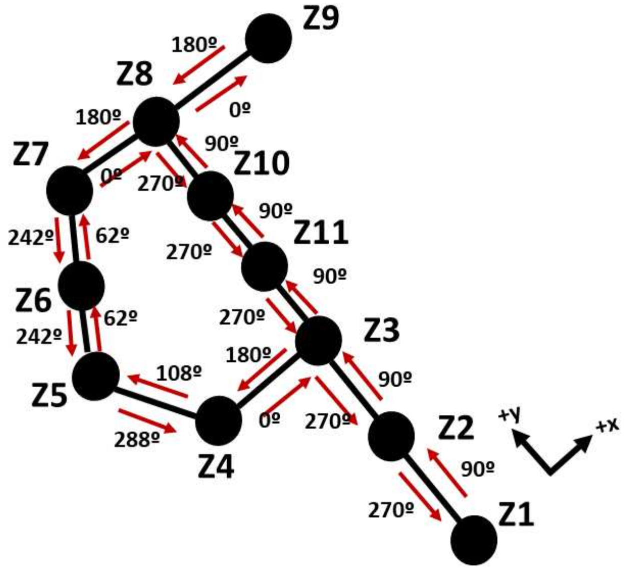


Figure 3.8: Zone connection diagram of Edgar mine. The figure on the left is a visual representation of the struct array where the green boxes signify a valid connection and everywhere else is not valid. On this visual representation, the left and top axis correspond to the zone number. The figure on the right shows the connections between the different zones and what angles the zones connect at. Positive angles are defined as counter-clockwise rotation about the x-axis which is at 0 degrees. The arrows signify the initial heading for the heading calculations in the next zone. The tail of the arrow is the zone the user is leaving and the tip of the arrow is the zone that the user is heading to.

The zone connection script requires the user to manually input these headings and initial positions. To estimate the initial position, take the maximum read range a tag can be detected at and subtract (or add) the coordinates of the tag, in the current zone but the one closest to the previous zone, by this read range. The coordinate is then moved to the center of the path. If the zone is at an angle from the previous zone, the maximum read range will need to be broken apart into X and Y contributions before being subtracted (or added) to the tag closest to the previous zone. The headings are much easier to determine as they are simply the heading the user will be at when entering the zone which is not necessarily the same heading the user would have when leaving the previous zone. This could occur on a corner turn with no zones in between.

3.2.4 Map Approximation

Every indoor environment with narrow corridors can be approximated by a grid of lines where each line runs right in the middle of the corridor, similar to a road map. Each line has a starting coordinate $P1(x_1, y_1)$ and an ending coordinate $P2(x_2, y_2)$. Whether the user is looking at a full map of the environment or in the environment with a tape measure, the user just needs to record the center coordinate of where each corridor changes directions. By using the map approximation script, Listing C.7, the user can easily build up the grid of lines using these corridor change coordinates. The script builds up a size *Nx4* array where *N* corresponds to the total number of lines and the size *4* dimension contains the coordinates of $P1$ and $P2$ put together: (x_1, y_1, x_2, y_2) . This grid of lines is used in the final step of the localization algorithm to map match the user's estimated 2D position, eliminating the non-physical positions that are outside the walls of the mine.

Figure 3.9 shows the actual map of Edgar mine with the approximated line map. Notice that only one line is necessary to run the entire length of the Miami tunnel (part of Edgar mine) since this tunnel stays constant in heading. The tunnels: A-left, B-left, and B-right only have one line to approximate them in this map because the user never advanced further during the testing of the localization system prototype. If future testing progresses farther down A-left and B-left, more lines will be needed. The path from A-left to B-left through the shop requires three lines to approximate the path. In general, corridors with turns will need to be approximated by a series of lines placed end to end.

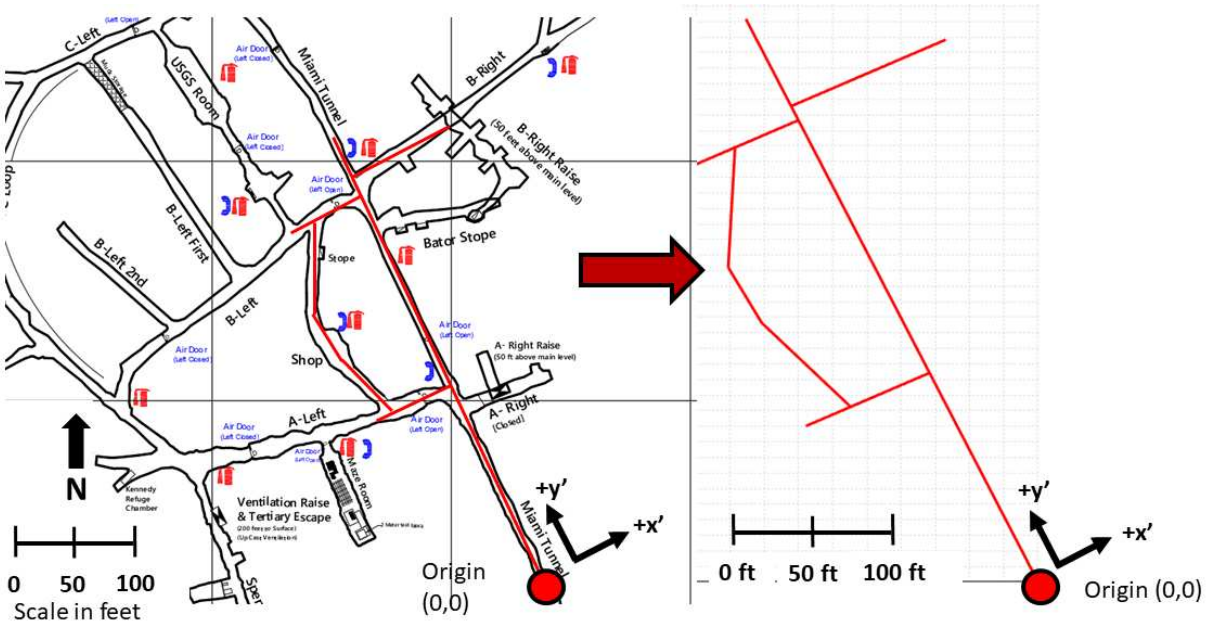


Figure 3.9: Approximation map of Edgar mine with lines. Left picture shows a small section of the full Edgar mine map with the line approximation of it in red. The right picture is the line approximation displayed in MATLAB. x' and y' are the coordinate system that was used for recording the tag coordinates and it is different than the coordinate system that the Edgar mine map uses. The origin dot in the left figure is the origin point for the prime axis not the Edgar map. The origin points corresponds to the halfway point in the full Miami tunnel.

3.2.5 RFID Data Interleaver

The current prototype of the localization system uses two different M6e readers, one connected to the left antenna and the other connected to the right antenna. As mentioned in the hardware chapter, the reason for this was because one reader would consistently throw errors when recording data from two antennas. Due to the bash script running the two readers simultaneously, they now generate separate data files when they are finished. The code in Listing C.8 combines these data files together into one file to appears as if one reader read both antennas. This format is necessary for the post-processing script to interpret it correctly. Future updates of the prototype with more advanced readers can reduce the number of readers to only one with two antennas working simultaneously.

3.3 Post-processing

Once the pre-processing steps are finished the user can now take a measurement using the acquisition code described above. Using the data from the measurement, the post-processing code will determine the user's path through the mine with time stamps that can be synced with other sensors, such as a dust monitor, to create a map of a particular hazard. The main function of the post-processing code is listed in Listing C.17 and it calls all of the other functions described in this section. The post-processing code is called the zone categorization algorithm (ZCA) as it uses the zone structure to split apart the data into bins. These bins correspond to each zone the user transitions into. A zone transition is a change in zone, not just that zone. For instance, the user can go from zone A to zone B to zone C and then back to zone B which would create four bins of zone transitions. For this reason, zones cannot be placed too close to each other or the system will from both zones simultaneously and indicate many zone transitions each with only a couple of IMU and RFID values. Only the processed RFID and IMU data will localize the user's position in that bin ensuring that each zone is calculated independently of other zones. Throughout the rest of this thesis, post-processing and ZCA will be used interchangeably. The benefits of the ZCA are:

- Modular code. User's can swap out different functions for an improved version of either RFID or IMU processing.
- Robust algorithm. At a minimum only one tag needs to be read in a zone which leverages the most resilient and binary feature of RFID: tag identification. At least three tags need to be read within a user specified time window to provide initial position.
- Counters the IMU's tendency to rapidly accumulate error by resetting it every zone.

- Errors do not influence the localization estimates in other zones.
- IMU data categorization allows for minimal tag placement. Does not require tags every a couple of feet on all walls.
- Ability to parallelize the processing for large data sets, due to the zone's independence.
- Once an environment has been pre-characterized, it is very quick for the user to take measurements and post-process the results. Code minimizes the number of user inputs needed to run completely different routes. Already scaled for many users.

The flow diagram of the ZCA is shown in Figure 3.10. Once the zone transitions have been identified, the data are split into a MATLAB struct that has a length that is the same as the number of zone transitions. Once the data are assigned to the correct zone transition index, the ZCA loops across the struct of zone transitions. For each index in the struct, all information about the tags is pinned to the struct as well as all of the categorized data, for that zone. In each index in the loop, the ZCA will calculate the 2D position using just the RFID values using the first coordinate at the user's inital position, determine at what indices in the IMU data the user stepped, determine the user's heading using the gyroscope component of the IMU data, calculate the user's 2D position through the entire zone, and then finally map matching these position to the mine map. Once the ZCA applies these processing steps to all of the zone transitions, it will then plot the results. The inputs into this algorithm are in Table 3.2. The outputs of the algorithm are the plotted 2D coordinates of the user in the mine.

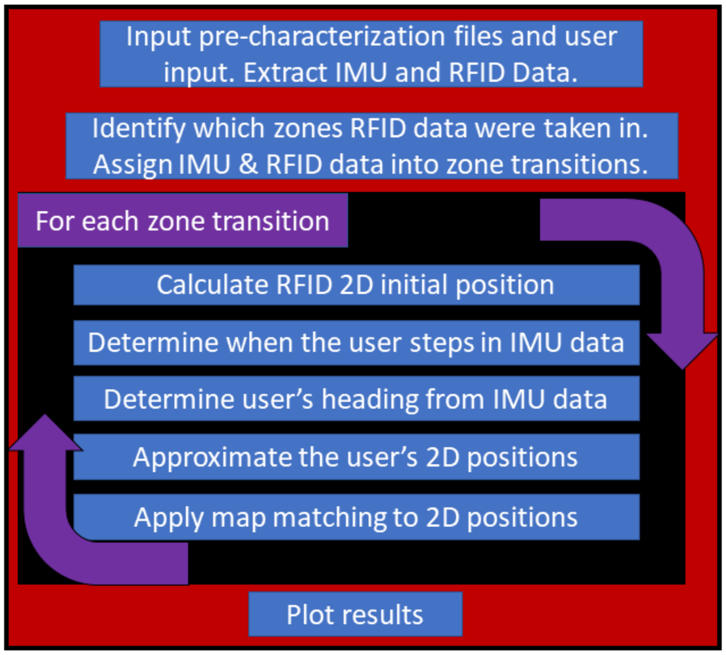

Input pre-characterization files and user input. Extract IMU and RFID Data.
Identify which zones RFID data were taken in. Assign IMU & RFID data into zone transitions.
For each zone transition
Calculate RFID 2D initial position
Determine when the user steps in IMU data
Determine user's heading from IMU data
Approximate the user's 2D positions
Apply map matching to 2D positions
Plot results

Figure 3.10: Flow diagram of the post-processing algorithm.

Table 3.2: User inputs to the algorithm and their default values

Variable	Example value	Description
location	$< C : ...DATA >$	Filepath to Data
filename_imu	$'imu\_data.txt'$	IMU data file name
filename_rfid	$'rfid\_data.txt'$	RFID data file name
filename_zone	$'zone\_info.csv'$	CSV file with tag/zone info
filename_lut	$'edgar\_lut.mat'$	.MAT file containing LUT coefficients
filename_zone_connect	$'zone\_relation.mat'$	.MAT file with connection array
filename_line_map	$'zone\_map.mat'$	.MAT file containing the line map
user_gait	2.3	User gait distance per step (ft/step)
msmnt_ant_num	2	Number of measurement antennas
dt	0.01	Sample rate of IMU (seconds)
gyro_axis	'y'	Gyro axis that corresponds to heading
gyro_gain	1	Gyroscope gain (scale factor)
flip_180	'y'	Whether CW or CCW turns are positive
tws	0.5	Window width in RFID 2D (sec.)
wo	0.25	Window overlap in RFID 2D (sec.)
gni	10	Gauss-Newton Iterations
map_match_yn	'y'	Map Match? Yes,No
result_rotat	25.2826	Rotate 2D results (degrees)

3.3.1 RFID Categorization

After loading all of the pre-processing files, IMU data, and RFID data, the first step of the post-processing algorithm is to determine when the user was in each zone. Using the RFID data and the zone information file, the post-processing script determines which zone each tag reading belongs to just by using the tag ID. Listing C.9 contains the function that accomplishes this. A minimum of only one tag read throughout the zone is all that is needed to determine which zone the user is in. This greatly enhances the robustness of the algorithm as a hazardous environment, like a mine, could provide a chaotic scenario where it is extremely difficult to read tags. A one tag read minimum relieves the pressure to read many tags in order for the rest of the localization code to function.

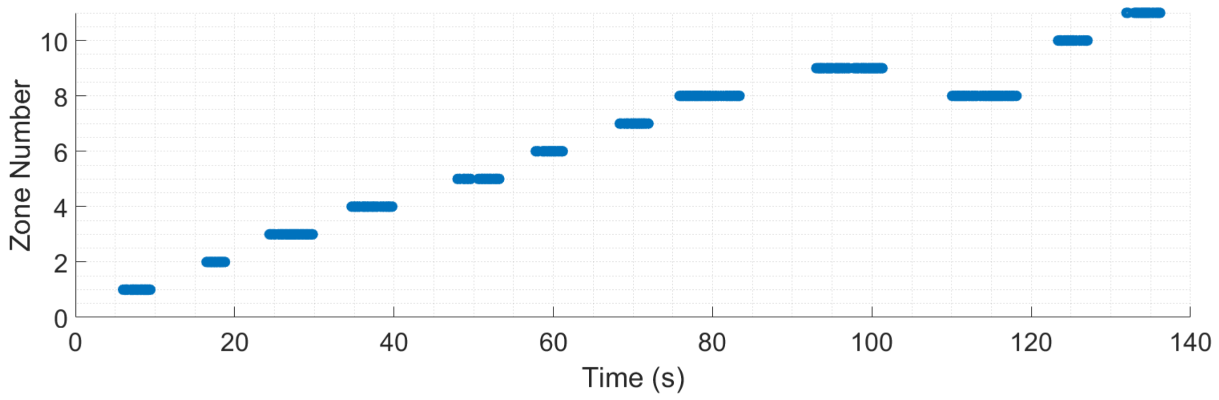


Figure 3.11: RFID tag measurements categorized into zones. The length of the lines corresponds to the length of time that the reader was reading at least one tag in the zone that it is categorized in. The gap between the lines corresponds to the absence of RFID readings in between zones. Data shown are from the Edgar mine measurement.

Listing C.9 outputs an array containing the assigned zone number for each RFID tag read. Figure 3.11 visualizes the output of the function contained in Listing C.9. This plot shows that the zones were optimally spaced so that the reader was not reading tags from more than two zones simultaneously nor are the time gaps between zones greater than 20 seconds. The tighter the zones are, the less error the IMU will accrue in between the zones. Since the localization code now knows when the tags were read and which zones they were apart of, it can now use this information to divide up the IMU data and assign them to the zones they belong to.

3.3.2 IMU Zone Assignment

Using the time stamps of the RFID data and the zones that they are identified in, the localization code divides up the IMU data into which zones they belong to. As soon as a single tag is read from a zone, all IMU data are assigned to that zone until a single tag is read from a different zone. Figure 3.12 illustrates how the RFID data are used to split up the IMU data. This figure shows the user's dead reckoned path which is obtained from the categorized IMU data. The dead reckoning calculations have not occurred at this point but the figure presents the IMU data this way for easy interpretation. Also, the initial position from the RFID 2D localization has not happened at this point either but it is included in this figure as an illustration of how all of the localization functions fit together. The 'initial position' is the first instance where at least three unique tags belonging to that zone are read. As soon as the user reads a single tag from the zone, the IMU data after this point is considered to be part of that zone until a single tag is read from a different zone. Categorizing the IMU data into zones is the primary way the localization algorithm bounds the IMU's dead reckoning drift. Each zone is independent of each other, therefore each zone's localization errors are purely contained in itself and do not influence the user's position in other zones.

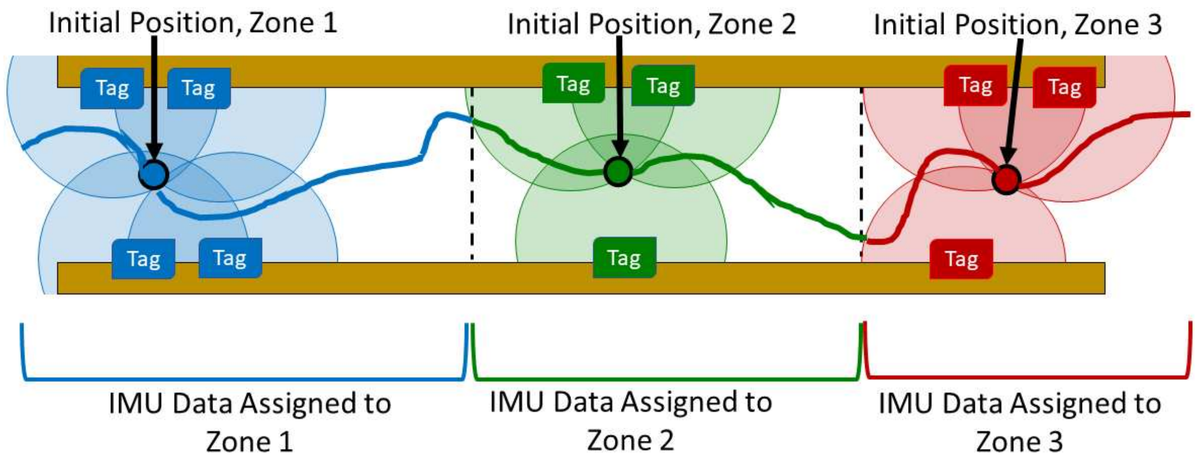

Initial Position, Zone 1
Initial Position, Zone 2
Initial Position, Zone 3
Tag
IMU Data Assigned to Zone 1
IMU Data Assigned to Zone 2
IMU Data Assigned to Zone 3

Figure 3.12: Illustration of the IMU zone categorizations. Starting from the left, the user progresses from zone 1 to zone 2 and finally to zone 3. The large tinted circles are the approximate areas where the tags can be read. Zone transitions are indicated by the dotted black lines. The colored lines indicate which zone the dead reckoned position belong to. The blue line changes to green as soon as a single tag in zone 2 (green) is read and the green line turns into red as soon as a single tag in zone 3 (red) is read.

The IMU assignment function is the heart of the localization algorithm. Listing C.10 contains the code necessary to assign the IMU data. First, the code loops through all of the RFID tag readings and their corresponding zone assignment, determining when the user leaves one zone and enters another. This is called a zone transition. The function then saves the times that these zone transitions occurred. Both the RFID data and the IMU data use the Raspberry Pi as the common time reference so all time stamps in both data sets are already synced. The assignment function takes the times that the user entered a zone (transitioned from a previous one) as well as time the user left the zone, and finds the indices of the IMU that these transition times fall at. The IMU indices in between these two points belong to that zone. The function repeats this process for the remainder of the zone transitions. The output of this function is the arrays of indices in the IMU data for each zone. The main localization script then uses these indices to assign the IMU data to the data struct that corresponds to that zone transition. From here, the IMU data can now be used in the IMU pedometer and IMU heading calculation functions.

3.3.3 RFID 2D Position

The most critical component for accurate dead reckoning is the initial position of the user. A poor initial position will offset the entire dead reckoned path propagating this error to ever coordinate. This is why the error from the initial position is the second most significant error; only second to the user's step length average. In algorithms that are not partitioned into zones, this initial position is determined once at the beginning making it easily justifiable to make it a hardcoded user input but for this algorithm it is calculated in every zone transition which requires a programmatic way of determining it. In other words, the user's position is reset in every zone and no positioning is used from the previous zone. This algorithm uses the RFID data to provide this initial position when the system reads at least three unique tag reads in a user defined time window. In the RFID tag clusters in the zones, the reader will read tags with a 20ms to 500ms time gap between readings. Figure 3.12 illustrates where the initial positions typically land in a zone. A unique tag read is the highest RSSi read of a unique tag ID by any of the antennas in that time window. Sometimes the system will read the same tag with both antennas, each reading a completely different RSSi due to signal reflection. In case the system does not read three tags, the zone connection matrix provides a rough initial position for each zone that the user hardcodes, but the use of this value is avoided at all costs.

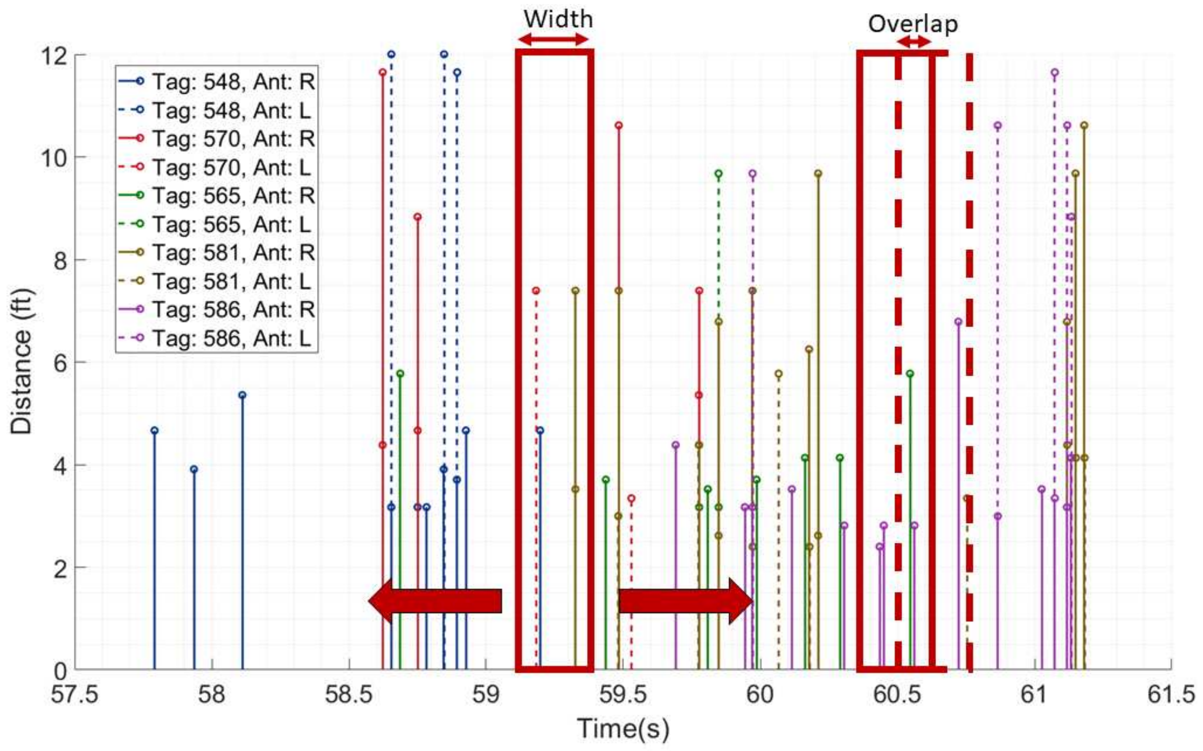


Figure 3.13: Distance from the user to each of the tags versus time. The look-up-table converted the RSSi to the distances shown in the figure. 'L' corresponds to the left antenna and the 'R' corresponds to the right antenna. The tag number corresponds to the last three digits of the tag ID.This figure also illustrates the swept window across the data. The window has a fixed width and is stepped through the code, in time, overlapping the next window by a time set by the user.

The RFID localization function is provided in Listing C.11 which takes the RFID data as an input and outputting the initial coordinate. References [40] and [41] were essential in creating this function. The RFID localization function first uses the look-up-table, such as the LUT shown in Figure 3.5, to convert every RSSi value to distance. The function moves a time window across all of the distance values, determining if at least three unique tag reads happened within it. Figure 3.13 shows the converted distance values across time with the window parameters, width and overlap, visualized. As mentioned above, a unique tag read corresponds to only one tag ID even if both antennas read the same tag. A high RSSi value is the shortest path from user to tag so if the other antenna reads that same tag with a lower RSSi value, it is most likely reading a reflection of the tag's signal. The lower RSSi value on the reflected signal will throw off the positioning as the LUT will convert it to an artificially long distance. The antenna that reads the tag ID with the shortest distance is considered the unique tag reading and the other antenna's RSSi value is ignored, within a window. If the same antenna reads the same tag ID multiple times within a window, the shortest distance value is used. If more than three unique tag reads occur within a window, they are included in the position estimate. Each window is independent of one another, i.e. the left antennas reading of tag ID 1234 will be considered the unique reading for one window but the right antennas reading could be considered the unique reading for the window immediately after. The windows overlap by a user specified time value to eliminate the edge cases where the three unique tag reads are split between the two windows. Without window overlap, neither window in succession would detect the three unique reads. The center position of the window is the time stamp of the estimated coordinate.

For each window that successfully reads at least three unique tags, the function passes these distance values and tag coordinates to a Gauss-Newton iterative method to determine the user's coordinate. The Gauss-Newton method is an iterative process which minimizes the error on a desired goal. The goal for this scenario is to minimize the error between two distance values. The first distance value is the distance from the estimated user's position to the tags and the second is the actual measured distance from the user's position to the tags. This method was chosen over triangulation or trilateration as the measured distance readings are prone to high error. For instance, if the moving window finds four unique tags within the specified time but only three of them have realistic distance readings, the Gauss-Newton algorithm will still be able to get close to a position estimate that satisfies the three realistic tag readings though the estimate will be influenced by the unrealistic distance reading. The function to execute this method is provided in Listing C.11 where the code was adapted from [40]. References [42] and [43] were instrumental in understanding the Gauss-Wewton algorithm when applied to 2D localization.

The tag coordinates array, $\mathbf{T}$, has size $Nx2$ where N corresponds to the number of unique tag reads and the second dimension is x, y.

$$\mathbf{T} = \begin{bmatrix} t_{1,x} & t_{1,y} \\ t_{2,x} & t_{2,y} \\ \vdots & \vdots \\ t_{N,x} & t_{N,y} \end{bmatrix} \tag{3.2}$$

The measured distance data from the user to each of the tags is contained in matrix $\mathbf{M}$, which is of size $Nx1$; one distance measurement for each unique tag.

$$\mathbf{M} = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_N \end{bmatrix} \tag{3.3}$$

Using $\mathbf{T}$, the function determines the initial position guess bounds, $z_{size,x}$ and $z_{size,y}$, by finding the minimum and maximum x and y coordinates across all tags. Equations (3.4) - (3.7) use MATLAB notation to select all values in the appropriate column of $\mathbf{T}$.

$$\begin{aligned} z_{min,x} &= min(\mathbf{T}(:,1)) && (3.4) \\ z_{min,y} &= min(\mathbf{T}(:,2)) && (3.5) \\ z_{max,x} &= max(\mathbf{T}(:,1)) && (3.6) \\ z_{max,y} &= max(\mathbf{T}(:,2)) && (3.7) \\ z_{size,x} &= z_{max,x} - z_{min,x} && (3.8) \\ z_{size,y} &= z_{max,y} - z_{min,y} && (3.9) \end{aligned}$$

After bounding the region for the initial guess, the function calculates the initial guess using 3.10.

$$g_0 = \left[(z_{size,x} * R) + z_{min,x} \quad (z_{size,y} * R) + z_{min,y} \right] \tag{3.10}$$

In equation (3.10), R is a random number between 0 and 1, seeded with the system time. The 0 subscript on g_0 indicates that this is the initial guess of g. After the initial guess, the Gauss-Newton algorithm iterates on this coordinate to converge to the estimated coordinate that minimizes the error between the estimate distance to the tags and the measured distance to the tags [42]. Each iteration of the estimated position is designated by g_i where the iteration number is designated by the subscript i. There is no convergence criteria such as a minimum error because this could easily lead to an infinite loop if the error plateaus at a value that is greater than the convergence error. The users sets the number of iterations at the beginning of the ZCA.

The matrix **G** in equation (3.11) is the current iteration's guess repeated across every row to get to length *N x 2*. The array **Δx** in equation (3.12) is the differences between the guess's x coordinate and each tag's x coordinate, of size *N x 1*. The array **Δy** in equation (3.13) is the differences between the guess's x coordinate and each tag's y coordinate, of size *N x 1*. The array **Δr** in equation (3.14) is the distances from the guess coordinate to each tag coordinate.

$$\mathbf{G}_i = \begin{bmatrix} g_{i,x} & g_{i,y} \\ g_{i,x} & g_{i,y} \\ \vdots & \vdots \\ g_{i,x} & g_{i,y} \end{bmatrix} \tag{3.11}$$

$$\mathbf{\Delta x} = \begin{bmatrix} (g_{i,x} - t_{1,x}) \\ (g_{i,x} - t_{2,x}) \\ \vdots \\ (g_{i,x} - t_{N,x}) \end{bmatrix} \tag{3.12}$$

$$\mathbf{\Delta y} = \begin{bmatrix} (g_{i,y} - t_{1,y}) \\ (g_{i,y} - t_{2,y}) \\ \vdots \\ (g_{i,y} - t_{N,y}) \end{bmatrix} \tag{3.13}$$

$$\mathbf{\Delta r} = \begin{bmatrix} \sqrt{dx_1^2 + dy_1^2} \\ \sqrt{dx_2^2 + dy_2^2} \\ \vdots \\ \sqrt{dx_N^2 + dy_N^2} \end{bmatrix} \tag{3.14}$$

The arrays $\frac{\mathbf{\Delta x}}{\mathbf{\Delta r}}$ and $\frac{\mathbf{\Delta y}}{\mathbf{\Delta r}}$ in equations (3.15) and (3.16) are each of dimension $Nx1$ and are the differences, for each axis, from the guess to the tag divided by the distances from the guess to the tag:

$$\frac{\mathbf{\Delta x}}{\mathbf{\Delta r}} = \begin{bmatrix} dx_1/dr_1 \\ dx_2/dr_2 \\ \vdots \\ dx_N/dr_N \end{bmatrix} \tag{3.15}$$

$$\frac{\mathbf{\Delta y}}{\mathbf{\Delta r}} = \begin{bmatrix} dx_1/dr_1 \\ dx_2/dr_2 \\ \vdots \\ dx_N/dr_N \end{bmatrix} \tag{3.16}$$

The matrix $\mathbf{D}$ is composed of $\frac{\mathbf{\Delta x}}{\mathbf{\Delta r}}$ in the first column and $\frac{\mathbf{\Delta y}}{\mathbf{\Delta r}}$ in the second, making $\mathbf{D}$ a size $Nx2$ matrix. $\mathbf{D}$ is the Jacobian matrix as described in [43].

$$\mathbf{D} = \begin{bmatrix} \frac{\mathbf{\Delta x}}{\mathbf{\Delta r}} & \frac{\mathbf{\Delta y}}{\mathbf{\Delta r}} \end{bmatrix} \tag{3.17}$$

Using $\mathbf{D}$, $\mathbf{M}$, and $\mathbf{\Delta r}$, the function can now calculate the delta that is used to modify the next guess iteration.

$$\Delta = (\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T(\mathbf{\Delta r} - \mathbf{M}) \tag{3.18}$$

The term $(\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T$ is the pseudo-inverse of $\mathbf{D}$ and $(\mathbf{\Delta r} - \mathbf{M})$ is the difference between the guess's distance to each tag and the measured distance to each tag. The next guess is calculated by subtracting the delta from the current guess iteration.

$$g_{i+1} = g_i - \Delta' \tag{3.19}$$

The delta has a prime notation to signify a matrix transpose operation as it is output from equation (3.18) as a $2x1$ array but to subtract it from g_i directly, it needs to be $1x2$. Equations (3.11) - (3.19) are calculated for every Gauss-Newton iteration and the Gauss-Newton estimation method is repeated for every valid window.

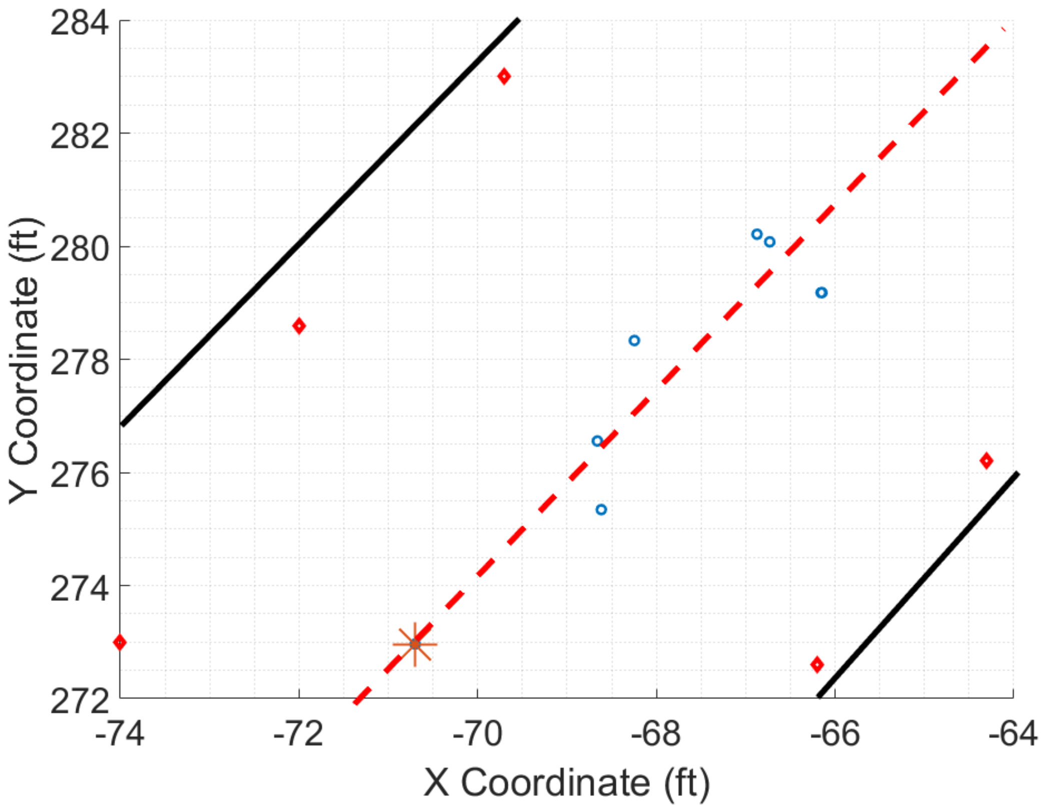


Figure 3.14: Estimated position of the user through a zone. User traveled from the bottom left of the figure to the top right. The red diamonds correspond to the tag locations. The black lines are the walls of the mine. The dotted red line is the actual path of the user through the zone. The blue dots are the RFID localized positions. The orange sun marker at the bottom of the figure is the initial position which is fed into the 2D positioning function. The shown data are from Edgar mine test in zone 6.

By performing the window search and Gauss-Newton estimation on the distance values shown in Figure 3.13, the resulting 2D positions are shown in Figure 3.14. This figure shows that over the RFID data for this chosen zone, the function only found seven valid windows where at least three unique tags were read. The initial position is indicated by the orange sun marker at the bottom left of the figure and it is the position that is output from this function. The function outputs the rest of the positions, signified by the blue circle markers, but they go unused. As seen in Figure 3.14, the estimated positions using just the RFID data show a reasonable estimate in the beginning of the zone but oscillate towards the end of the zone. This figure shows a very plausible initial position in the zone but other zones have an initial position that is unrealistic, which creates an offset for the rest of the dead reckoning. Future work will be to fuse the IMU 2D positions with all of the RFID 2D positions but due to the variation in these positions, there was no reason to incorporate this sensor fusion into the system in its current state. RFID hardware improvements on the prototype may improve the 2D estimate significantly.

3.3.4 IMU Pedometer

The IMU's accelerometer records the user's X axis, Y axis, and Z axis acceleration, measured in g's. These accelerometers are very good at recording the user's sudden impulses of movement, such as each jolt when the the user steps. This developed prototype device and software uses the IMU's accelerometers as a pedometer which counts the user's steps and the times that they occurred at. Once the number of steps are known, the algorithm can then estimate the traveled distance of the user by multiplying the step count by the user's average gait length. The benefit to this method is that there is no drift since there is no integration of the data, just detection of the peaks straight from the acceleration data. The functions used to detect the steps are shown in Listing C.12 and in Listing C.13. Listing C.12 takes the raw acceleration data and the sample period length and outputs the indices of the acceleration data that the steps occurred at. This function first calculates the magnitude of the acceleration data using equation (3.20) where A_x,A_y, and A_z corresponds to the acceleration data along the X axis, Y axis, and Z axis, respectively.

$$A_{mag} = \sqrt{({A_x}^2 + {A_y}^2 + {A_z}^2)} \tag{3.20}$$

Next, the function performs the fast fourier transform (FFT) using MATLAB's *fft* command on A_{mag} to convert it to the frequency domain. The function then filters the data by zeroing out the values that are below $-2.5Hz$ and above $2.5Hz$. After the data have been filtered in the frequency domain, the function performs the inverse fast fourier transform using MATLAB's *ifft* command which makes a clean time domain waveform with a period of roughly $1second$ corresponding to a human's natural stepping rate. If the user is stationary, then the waveform will have a flat value at 0 and will not contribute to the step count, as intended. The peakfinder function, described in Listing C.13 and taken from [44], takes in this filtered time signal and a user specified threshold (0.8 for this prototype) and finds the indices where the peaks occur. These indices are then output from the function and are used later in the localization code to advance the user's travel distance. Figure 3.15 shows the filtered time domain waveform and the detected peaks. Currently, the code only uses a single constant to multiply the step count with which leads to an under approximation or an over approximation of the user's traveled distance. This error is mitigated by shortening the distance between zones so the IMU has less distance to dead reckon over.

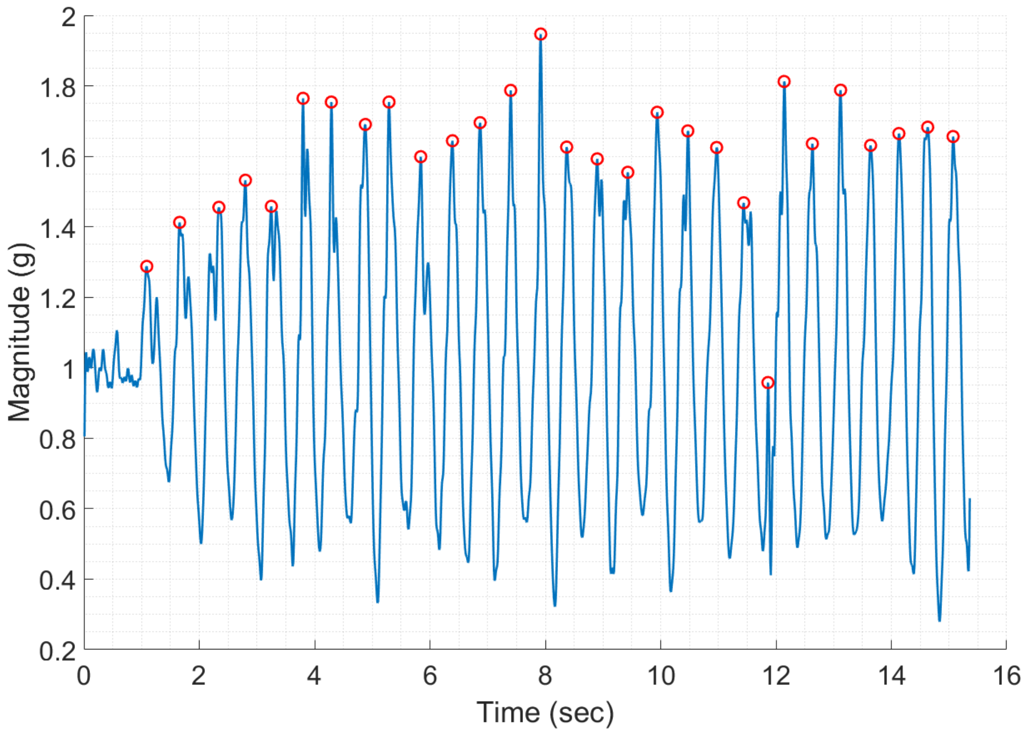


Figure 3.15: Magnitude of the time domain acceleration waveform after being filtered (blue). The red circles correspond to the peaks of the waveform that the peakfinder function [44] detected. Notice how for the first second the user is approximately stationary and no peak is detected. Data were taken from a portion of the IMU measurements during the Edgar mine experiment.

3.3.5 IMU Heading

The IMU's gyroscope records the X axis, Y axis, and Z axis angular velocity about the center of the IMU, measured in degrees per second. Current heading algorithms will fuse this data with the accelerometer and magnetometer data and apply techniques such as Kalman filtering to give a user's heading estimate using the world as the reference frame. For the purposes of this algorithm, only the relative heading difference from an initial orientation is all that is needed to dead reckon the user. This heading is calculated by integrating the gyroscope data along the axis that is normal to the path, assuming that the IMU is fixed in a known orientation and does not pitch or roll. The prototype described above has the IMU locked to the vest so only the yaw axis sees the user's heading change. Due to the swaying and bending of the user through the mine, other axes will have a portion of the heading change resulting in heading error. If future prototypes change the orientation of the IMU, a rotation matrix may be needed to realign the data.

Listing C.14 contains the function used in the localization code to calculate the user's heading. The inputs to this function is: the data struct for the zone which contains only gyroscope data and time data that were assigned to this zone; a variables which specifies the gyroscope axis to calculate across; a gain constant that multiplies the gyroscope data values; and a coefficient which can flip the axis of rotation. For this prototype, the Y axis corresponds to the user's yaw axis which is why it is the only one calculated over to determine the user's heading. Since the user can redefine the axis of rotation for different measurement scenarios, this function uses the axis flip coefficient to multiply all of the gyroscope data by either a 1 or -1 to change the headings to be positive or negative. The input data struct also contains the initial heading which is determined prior to this function. The heading is calculated using MATLAB's *cumtrapz* function which is a cumulative trapezoidal integration method. The system time is used in this function as there is slightly nonuniform spacing between measurement values.

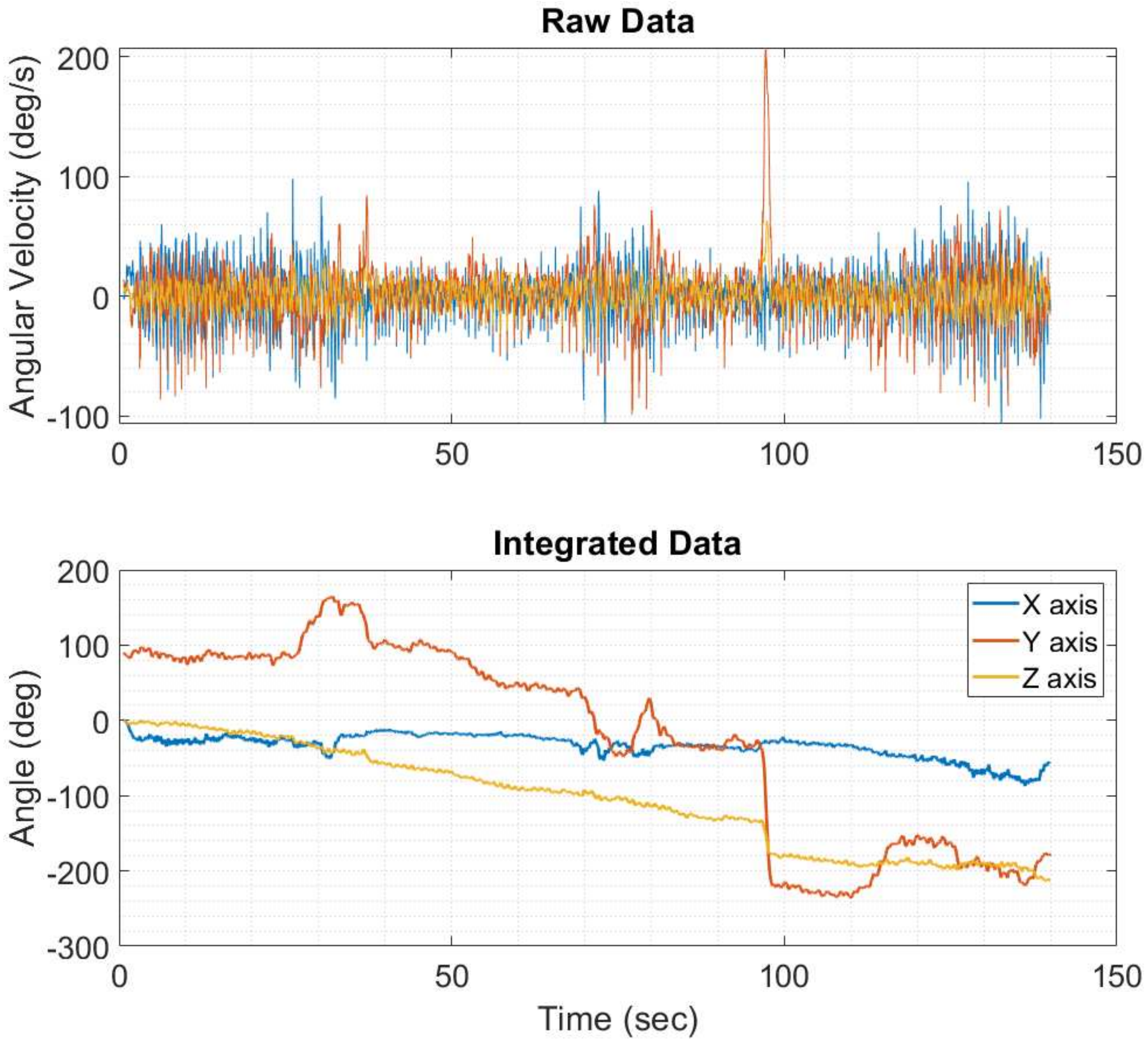


Figure 3.16: Raw data from the gyroscope for each axis (top) and the resulting heading after cumulative trapezoidal integration of the data (bottom). The X axis (blue), Y axis (orange), Z axis (yellow) corresponds to pitch, yaw, and roll, respectively. The user's yaw axis is used to calculate the heading for the IMU dead reckoning. Data shown are from the Edgar mine measurements.

The raw gyroscope data are show in the top plot of Figure 3.16 for an entire measurement but in actuality, the entire measurement set will be split into each of the zones. In this figure, the raw X axis data (blue line) have a strong oscillatory nature about the zero axis but the positive components and negative components are balanced such that the integrated result is close to constant across the whole measurement. It has the large shift in the beginning because the user rocked forward to begin walking. The X axis corresponds to the user's pitch axis which is oscillatory in nature as the user rocks forward and backward as they walk. The raw Y axis data (yellow line) have a similar oscillation as the X axis but it has much larger spikes in the positive axis to turn left and larger spikes in the negative axis to turn right. As mentioned previously, the Y axis corresponds to the user's yaw axis making it the major determinant of the user's heading. The large spike at around 90 seconds corresponds to when the user performed a 180 rotation which is seen in the large negative jump in the bottom plot of Figure 3.16. The raw Z axis data (yellow line) have fairly small in magnitude compared to the X and Y axes making it susceptible to measurement noise which is why it has the worst drift over the measurement when it is integrated. Since the user had a slight roll when conducting the 180 degree turn the z axis contains a small percentage of the user's heading information which is why it also has a significant peak at around 90 seconds. A simple rotation matrix cannot be applied to all of the data to correct this one point because it would cause inaccuracies in all points before and after the 180 turn.

3.3.6 Combined 2D positioning

With the user's step indices, heading, and the RFID initial position along with the time at which it was calculated, the post-processing script can now calculate the user's 2D position. The function used is provided in Listing C.15. The initial position is not the first coordinate the user was at when they entered the zone. Instead, it was the first coordinate where the system read at least three RFID tags within a specified time window. This means that within the same zone, there is IMU data recorded before the initial position and data recorded after. None of this IMU data came from a different zone as at least one tag from the current zone was already read to assign the data. Just not enough tag reads to determine the initial position. If three tag reads are never read in a specified time window throughout the entirety of the zone, the function uses the approximate initial position that is stored in the zone connection file. The 2D positioning function uses the time stamp of the initial position to separate the IMU data into two groups, the pre-initial position data and the post-initial position data. See Figure 3.17 for an illustration of where the initial position falls in relation to the IMU data before and after. The user only advances in the environment using steps so the 2D coordinates are only updated every step. The function loops across the length of the IMU data but it only updates the 2D coordinate if that index in the IMU data is where a step occurred. The step indices were obtained earlier in the post-processing code. Starting, with the post-initial position IMU data, equations (3.21) and (3.22) calculate the user's coordinate. In these equations, x and y correspond to the user's coordinates, i is the user's current step index, *i-1* is the user's previous step index, θ_{i-1} is the user's heading at the previous step index, and *Gait* is the user's average step length in feet.

$$x_i = x_{i-1} + Gait * cos(\theta_{i-1}) \tag{3.21}$$
$$y_i = y_{i-1} + Gait * sin(\theta_{i-1}) \tag{3.22}$$

The initial x and y coordinates are the initial position obtained from the RFID and the initial θ is the user's heading at that point in time. To calculate the user's coordinates before the initial position, move the $Gait * cos(\theta_{i-1})$ term to the left side of equation (3.21) and move $Gait * sin(\theta_{i-1})$ to the left side of equation (3.22). The two main assumptions made by these equations is that the user is walking in the same direction as their indicated heading and that they are stepping at roughly the same gait length for each step. The first assumption is reasonable as the user's trunk will most likely not be moving in a direction that is separate from their step direction unless there is an obstacle in the mine that results in the miner side-stepping to get around it. Since the IMU is on the chest and not on the hardhat, the user can freely look around without the 2D estimate being thrown off. The assumption that the the user is always stepping at their average step length is obviously flawed. There are numerous reasons why the user would walk at a shorter gait distance or a longer gait distance for extended periods of time. This assumption is one of the main sources of error currently in this algorithm. This assumption will need to be improved in the future, possibly by using machine learning to detect user's gait patterns and determine what the most likely gait distance is, such as the method described in [45]. Another solution is to fuze the other RFID 2D position values with the dead reckoned position estimate. In the current prototype, the RFID 2D positions fluctuated too much through a zone to allow for this position fusion. Listing C.15 contains a placeholder where future development can interject their fusion algorithm. After this function calculates the user's 2D position estimate, they are input into the map matching function.

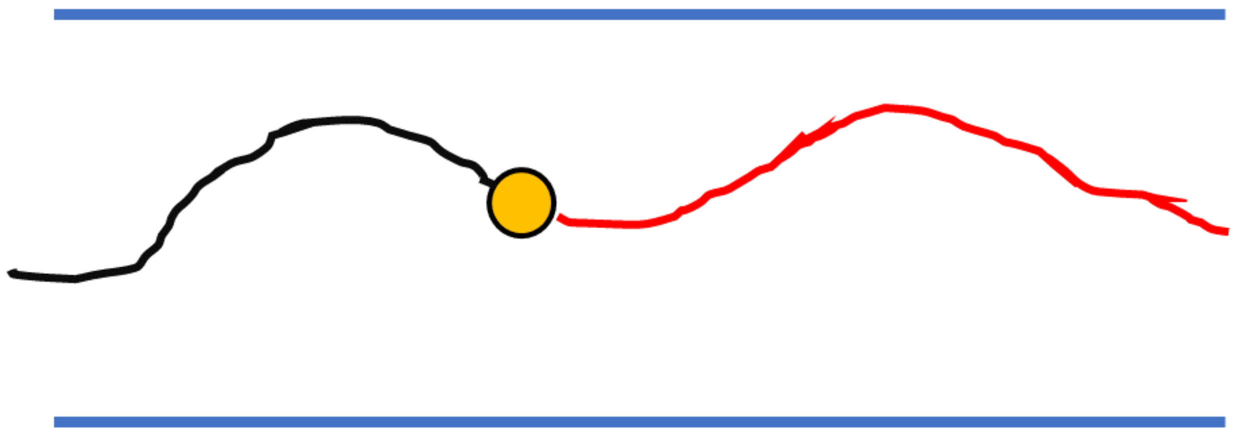


Figure 3.17: Simple diagram showing the initial position for the 2D function. The black line is the dead reckoned user position, from the IMU data, before the initial position. The red line is the dead reckoned user position after the initial position. All data still belongs to one zone.

3.3.7 Geometric Point-to-Curve Map Matching

The narrow corridors of underground mines and indoor spaces are very useful to localization algorithms due to their constraints of a user's lateral displacement from the center path of the corridor. This information allows for the sacrifice of the users lateral distance from a center path in favor of a position that can only be on the path, see Figure 3.18. Geometric-point-to-curve map matching, as described in [46], is one of the simplest and most effective ways of accomplishing this task. This map matching technique takes the approximated coordinate of a user, finds the closest line that point is located to, and then projects that point onto that line. Map matching is the final step of the developed 2D localization algorithm and removes a majority of the drift errors that arise from the IMU and RFID localization. Since the drift errors can quickly become nonphysical and destructive to the overall localization, this is a reasonable trade off. The code used to perform the map matching is shown in Listing C.16 and is the MATLAB adaptation of [47].

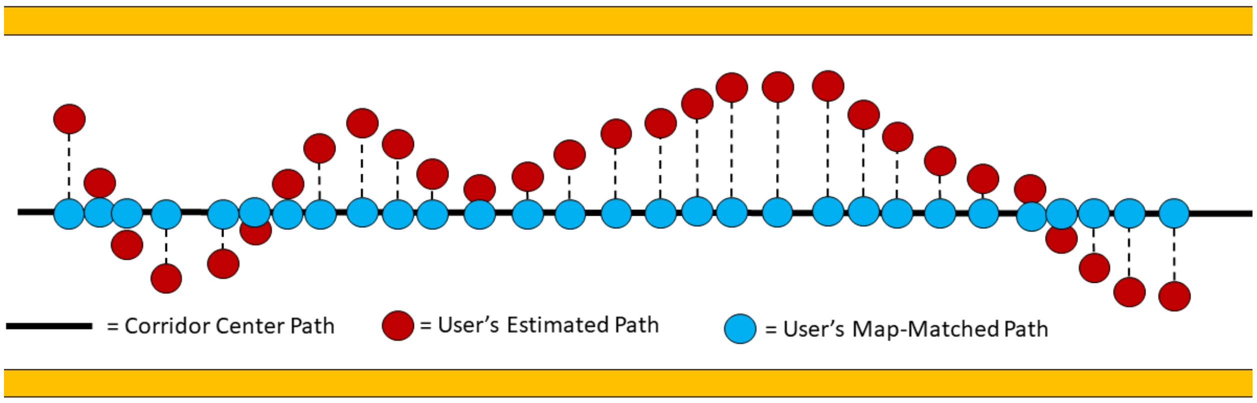


Figure 3.18: Diagram of point-to-curve map matching. All lateral position information is lost as the user's estimated position is projected onto the center path.

The inputs to the map matching code are the user's estimated position obtained from the previous localization functions in the algorithm as well as an approximation of the environment map using lines along the center paths. Each of these lines are a line segment which have a coordinate to map the beginning of the line and a coordinate to map the end of the line segment. For each of the user's estimated position coordinates, the map matching algorithm finds the closest line segment, identifies where the point is in relation to the line segment, and then finally projects where the point falls onto the line segment. Figure 3.19 graphically shows the geometry used to project the point to the line as well as the different scenarios that the point could be in relation to a given line segment.

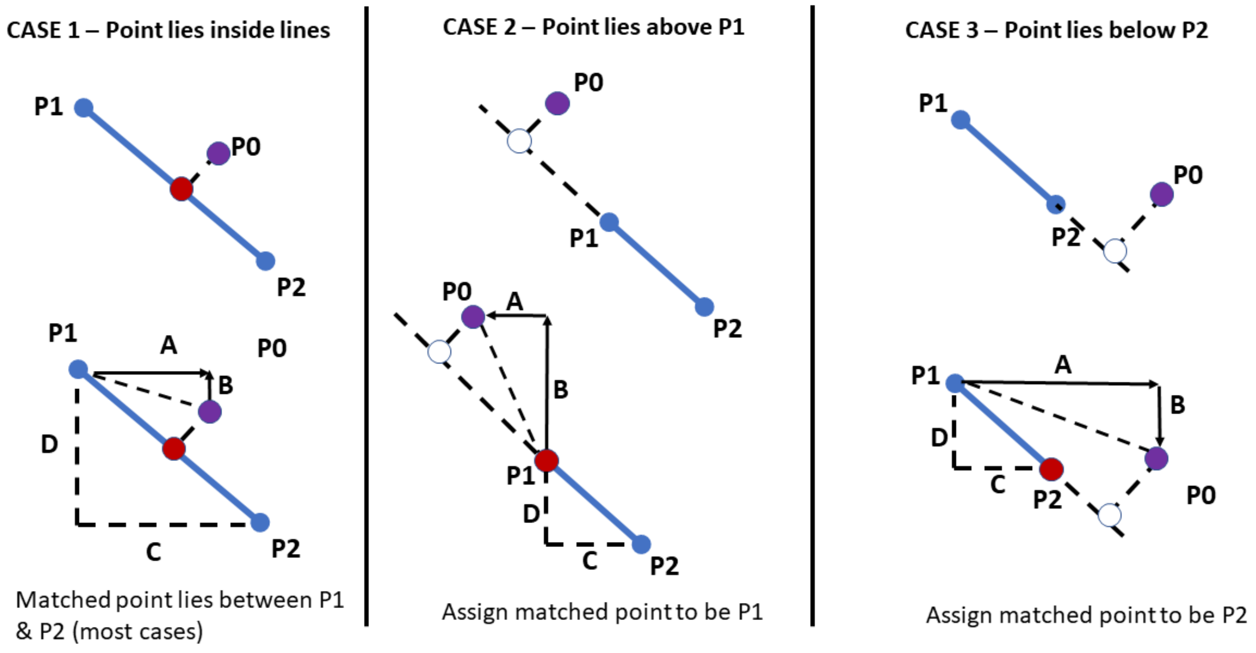

CASE 1 – Point lies inside lines
P1
P0
P2
P1
A
P0
B
D
C
P2
Matched point lies between P1 & P2 (most cases)
CASE 2 – Point lies above P1
P0
P1
P2
P0
A
B
P1
D
C
P2
Assign matched point to be P1
CASE 3 – Point lies below P2
P1
P0
P2
P1
A
B
D
C
P2
P0
Assign matched point to be P2

Figure 3.19: The three possible scenarios when matching a point to curve; figure modified from the one shown in [47]. P0 is the point that is being matched. P1 is the starting coordinate of a line segment. P2 is the end coordinate of a line segment. A,B,C,D are geometric quantities used in the map matching code. The red dot is the map matched point. The white dot shows where the point would have projected to if the line extended to infinity. See Listing C.16

Given a line segment that begins with coordinate $P1(x_1, y_1)$ and ends in $P2(x_2, y_2)$, and a user's estimated position coordinate $P0(x_0, y_0)$, we seek the user's map matched point $PM(x_m, y_m)$ [47]. We first calculate the vectors A, B, C, D using equations (3.23), (3.24), (3.25), and (3.26). These vectors are: the beginning of the line segment to the user's estimated coordinate, the vector from the beginning of the line segment to the user's estimated coordinate, the X component of the line segment, and the Y component of the line segment, respectively. The line segment, $\vec{L}$, is denoted by equation (3.27) and the vector from $P1$ to $P0$ is denoted by $\vec{T}$, obtained from equation (3.28).

$$\begin{aligned}
\vec{A} &= (x_0 - x_1)\hat{i} && (3.23)\\
\vec{B} &= (y_0 - y_1)\hat{j} && (3.24)\\
\vec{C} &= (x_2 - x_1)\hat{i} && (3.25)\\
\vec{D} &= (y_2 - y_1)\hat{j} && (3.26)\\
\vec{L} &= \vec{C} + \vec{D} && (3.27)\\
\vec{T} &= \vec{A} + \vec{B} && (3.28)
\end{aligned}$$

After the calculations of these four vectors, the dot product of the point onto the line segment,$\vec{L}$, is given by:

$$\vec{T} \cdot \vec{L} = (\vec{A} * \vec{C}) + (\vec{B} * \vec{D}) \tag{3.29}$$

and the squared magnitude of $\vec{L}$ is give by:

$$\left\|\vec{L}\right\|^2 = (\vec{C} * \vec{C}) + (\vec{D} * \vec{D}) \tag{3.30}$$

Dividing the dot product projection by the squared magnitude of the line segment, gives the equation (3.31) which provides the magnitude of the projected vector, $\left\|\vec{M}\right\|$.

$$\left\|\vec{M}\right\| = \frac{\vec{T} \cdot \vec{L}}{\left\|\vec{L}\right\|^2} \tag{3.31}$$

The magnitude of $\vec{M}$ determines where the map matched point is relative to the line segment. If $\left\|\vec{M}\right\|$ is greater than one or less than zero than it lies outside of the line segment and must be matched to either $P1$ or $P2$, respectively. Refer to Figure 3.19. If $\left\|\vec{M}\right\|$ is between zero and one then it falls inside the line segment and equations (3.32) and (3.33) can be used to find the matched coordinate.

$$x_m = x_1 + \left\|\vec{M}\right\| * \vec{C} \tag{3.32}$$

$$y_m = y_1 + \left\|\vec{M}\right\| * \vec{D} \tag{3.33}$$

Finally, the following equations find the distance from the line segment to the user's estimated position, which is the same as the distance from P_M to P_0:

$$
\begin{aligned}
D_x &= x_0 - x_m & (3.34)\\
D_y &= y_0 - y_m & (3.35)\\
Distance &= \sqrt(D_x^2 + D_y^2) & (3.36)
\end{aligned}
$$

The code contained in Listing C.16 calculates the matched coordinates and distance for every line segment in the entire map for each user's estimated position. Only the matched coordinate that corresponds to the shortest distance to a line segment will be assigned and the user's position will be matched to that line segment.

The pre-map matched results and post-map matched results from the experiments conducted in Brown building are shown in Figure 3.20. The left figures shows how gyroscope's heading readings accumulate error very quickly as the user progresses through the hallway, making the user to appear that they are walking at an angle away from the center of the hallway which would eventually take them through a wall (nonphysical). On top of this, the initial position provided by the RFID 2D estimate shifts both the blue and red lines away from the center of the path. The right figure shows the map matched result where it is obvious where the user went. The right figure also shows one of the weaknesses of the map matching, in the case of the red zone, where map matching cannot correct for the shift of the entire line which leads to red curve not connecting to the green curve (on top of the errors from non-uniform gait).

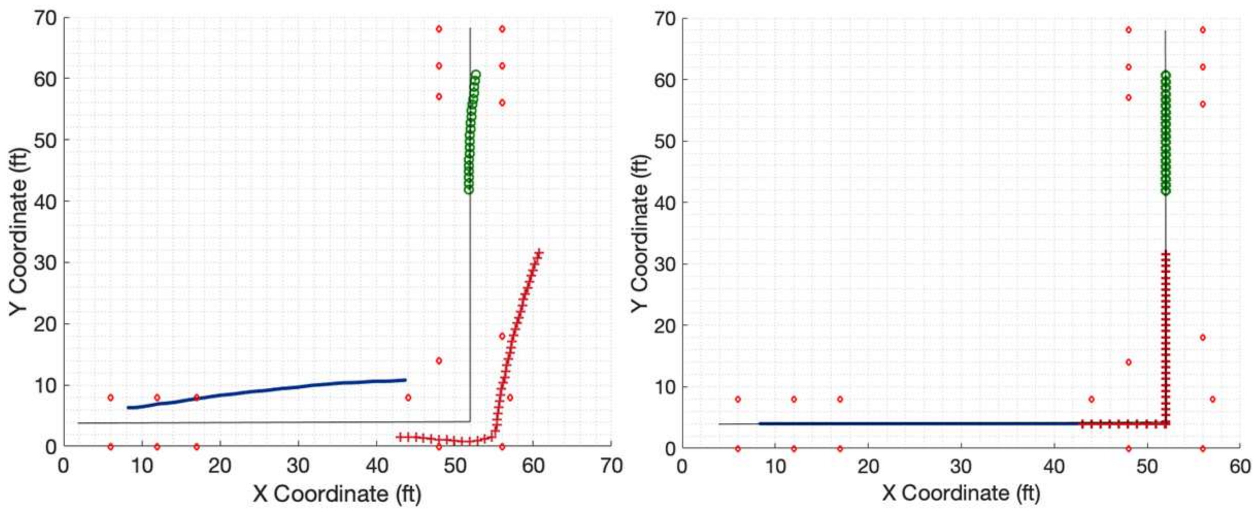


Figure 3.20: Pre-map matching user's estimated positions (left). Post-map matching user's estimated position (right). Data taken in the hallway of Brown building. The solid blue line denotes the positions associated with zone 1. The red line with crosses denotes the positions associated with zone 2. The green line made of circles denotes the positions associated with zone 3. The solid black line is the center path of each of the hallways as well as the path the user walked. The red diamonds indicate where the RFID tags were placed.

The point-to-curve map matching technique allows for a cleaner estimate of a user's position on a map with the addition of a couple of errors. The loss of the lateral positional information adds error proportional to the width of the corridor, which is why this technique should only be applied to narrow corridors. Another error that this technique accrues is when the approximated positions of the user are at a significant different heading than the heading of the center path. Only the component that is parallel to the center path will be recorded so a large heading deviation will result in an under approximation of the user's distance along a path. If the heading is exactly 90 degree different, then it will appear that the user is stationary on the path and no forward movement was achieved. Another error that arises from this technique is when the user's positions bends around a 90 degree path, the coordinates will be matched around the turning point but it will appear that the user 'jumped' from one line to the next. This happens when the path estimate is on the interior of a path that has a turn. The last error that occurs with map matching is when a map has numerous paths that are tightly clustered together, such as a

4-way intersection, and the user's map matched path bounces back and forth between the different curves. It is up to the previous steps of the localization to get the user estimated close enough for map matching to be effective. Some of these errors are discussed in detail in [46].

3.3.8 Plotting

As the ZCA finishes with the map matching function, it now moves on to plot the user's 2D coordinates through the entire mine. This step may seem trivial but due to the zone structure of the algorithm, the plotting function needed to be completely automated and scalable to an inordinate number of zones. The solution was to assign each of the data structs in each zone a unique color and linestyle. Even though each data struct in the central loop of the ZCA corresponds to a zone transition it also contains the zone number that the user is operating in. This means that if a user moves from zone 1 to zone 2 to zone 3 and then back to zone 2, the plotted results from zone 2 will have the same color and linestyle on both zone transitions. The code written in [41] was essential in developing this capability. The plotting function in the ZCA also reads in all of the tag coordinates and the line map approximation and plots them on the figure with no extra work needed by the user. At the beginning of the ZCA the user can enter a degree value which the plotting function will use to rotate all 2D results, line map, and tag coordinates about the origin to better align with a different coordinate axis. All of this plotting functionality makes generating the final result figures very easy and scalable to massive environments. Examples of the final result figures are provided in the next chapter and will show the results of the Brown building and the Edgar mine measurement experiments.

CHAPTER 4
SYSTEM MEASUREMENTS

4.1 Experimental Results

The testing of the localization prototype and ZCA were carried out in two environments: Colorado School of Mines's Brown building, and Edgar experimental mine. Brown building was used for the majority of the testing due to convenience but also because it was a simple environment with few of the challenges present in underground mines. Edgar mine was the final testing ground for the system as it was an actual underground mine.

4.1.1 Brown Building Experiment

The 3rd floor of Brown building provided an ideal testing environment as the walking surface was flat with no obstacles and the walls were perfectly smooth. Figure 4.1 shows a picture of zone 1 and Figure 4.2 shows a picture of zone 3 in Brown building. Zone 2 is in the corner or the two hallways. The floor tiles in Brown building were exactly 1 foot by 1 foot so it was easy to mark the locations of the tags and the user's path. The look-up-table method described in chapter 3 uses multiple antenna orientations in different parts of the environment to build up the data. For the Brown building tests, only one antenna configuration in the intersection was used to build up the the look-up-table. This resulted in a moderate RFID 2D positioning error for these experiments but influenced the improved look-up-table generation procedure that was used in Edgar mine.

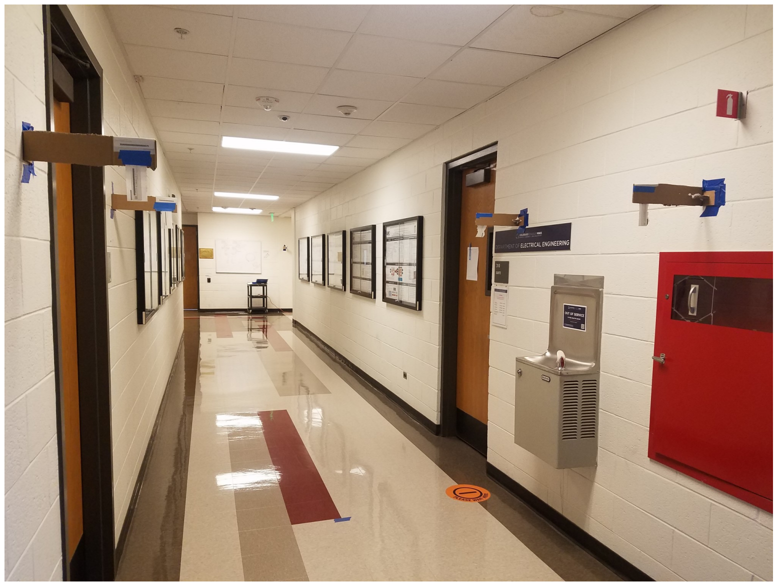


Figure 4.1: Picture of zone 1 in Brown building experiment.

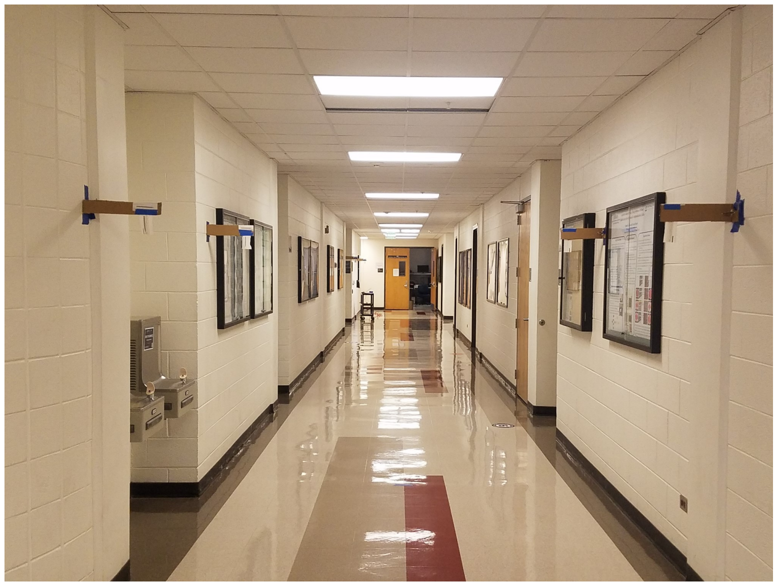


Figure 4.2: Picture of zone 3 in Brown building experiment.

Figure 3.20 shows the user's estimated position, before and after map matching. The user starts on the bottom left of the plot at (0',0'), walks along the X axis to coordinate (52',4'), and then turns 90 degrees and walks along the Y axis up to the coordinate (52',60'). At this point, the user stays stationary till the data acquisition system finishes running. Because the user does not step, the algorithm successfully keeps the user at this point. There are three zones for this measurement each with 6 RFID tags. The colors of the localized positions in zones 1, 2 and 3, are blue, red, and green, respectively. The first observation is that the gyroscope drifts very quickly in the 1st and 2nd zones, with very little drift in the third zone. The RFID 2D position appears to have shifted the blue zone to the right by a foot or two as the initial position should have been closer to the origin. If the RFID 2D position was accurate, the line would start before the zone, ever so slightly. The blue curve is also shifted in the Y direction by about 5 feet at the beginning which is due to the RFID initial position being off. The error in the RFID initial position is very clear in zone 2 where the line is shifted below and to the right of the path. If this shift was not here, the red line before the turn would be right on the path and the points after the turn would have brought it closer to connecting with zone 3. Zone 3 shows no obvious signs of error from either the RFID initial position or gyroscopic drift.

After map matching, the lines corresponding to the positions in each zone get snapped to the line map approximation and immediately we see that zone 1 connects to zone 2 very well but zone 2 does not connect to zone 3 very well. This is due to the underestimation of the user's gait length and the shift from the RFID initial position error. If the blue line had a correct initial position, the entire line would be shifted to the left and it too would not connect to zone 2. The pre-characterization experiments established the user's gait length at around 1.3 feet per step but this number would need to be increased to around 2 feet per step for the lines to connect. For the sake of not deviating from the pre-characterization, the post-processing code uses 1.3 feet, even though the artificially increased 2 feet would have connected better.

The experiment in Brown building showed good results with only 9 feet of disconnect between zones 2 and 3 a third of which was accounted for by RFID 2D positioning error. The dead reckoning drift increased rapidly even for this small scale experiment but the map matching was able to successfully correct it. This experiment showed that the look-up-table is crucial in reducing the RFID 2D positioning error and this lesson influenced the creation of the look-up-table generation method described in chapter 3 which was enacted in the Edgar mine experiments.

4.1.2 Edgar Mine Experiment

Edgar mine provided no such luxuries as Brown building. The mine was full of obstacles such as low hanging ceiling, railroad tracks, and mining vehicles. When the user avoided these obstacles, their gait length and heading would change, adding error to the dead reckoning position. On top of this, the walls of Edgar mine were jagged; oscillating between jutting rocks and 3 foot recesses which changed the scattering of the RFID tags. These challenges are commonplace inside of mines and relatively minor compared to the other hazards that miners must avoid.

The experiment route in Edgar mine is shown in Figure 4.3. The user starts at the 'START' marker, walks to the first intersection and takes a left into A-Left. Immediately, the user enters another intersection where they make a right onto the path that connects A-Left to B-Left by way of the Shop. The user leaves this path by making a right onto B-left and then traveling immediately into another large intersection. This intersection was the most spacious in the mine, requiring tags to be placed at the entrances of all possible routes out. In this zone, the user made a left turn into the Miami tunnel and then an immediate right into B-Right. The user advances along B-Right and makes a U-turn right before the B-Right Raise crossing. The user walks back along B-Right and makes a left onto the Miami tunnel. The measurement ends when the user walks past the Air Door in between the Bator Stope and A-Left. The measurement did not continue past this point as there was a large rail car blocking the route back to the starting point. This measurement was repeated twice with the results being exactly the same, showing that the system gives repeatable results over the large mine environment. Pictures of zones 1, 3, 6, and 8 are shown in Figure 4.4, Figure 4.5, Figure 4.6, Figure 4.7, and Figure 4.8. These zones exemplify different aspects of the challenging underground mine environment from a over-the-air communications perspective.

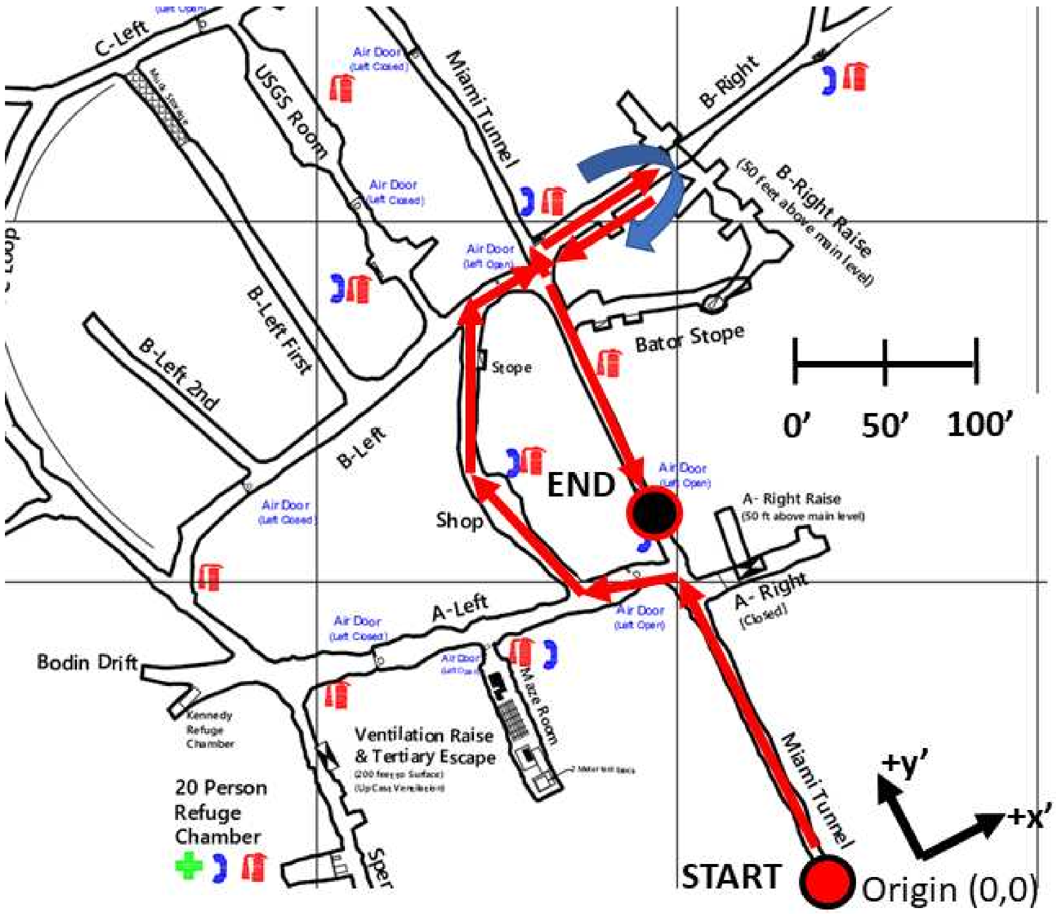


Figure 4.3: User's path in the Edgar mine experiment. User starts at the 'START' marker and stops at the 'END' marker, which is right before the intersection. The blue turn around arrow is where the user turned 180 degrees.

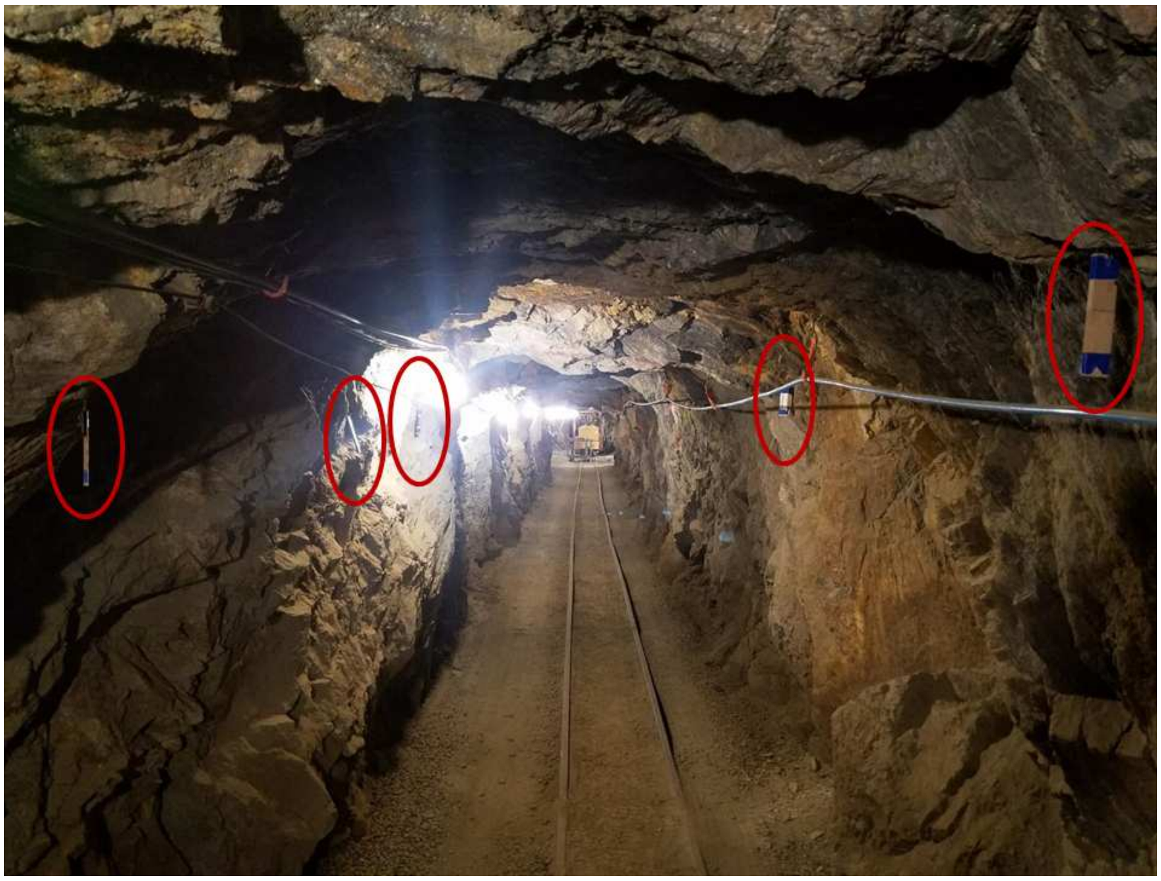


Figure 4.4: Picture of zone 1; an exemplary straight section. The red ovals highlight the tags in view.

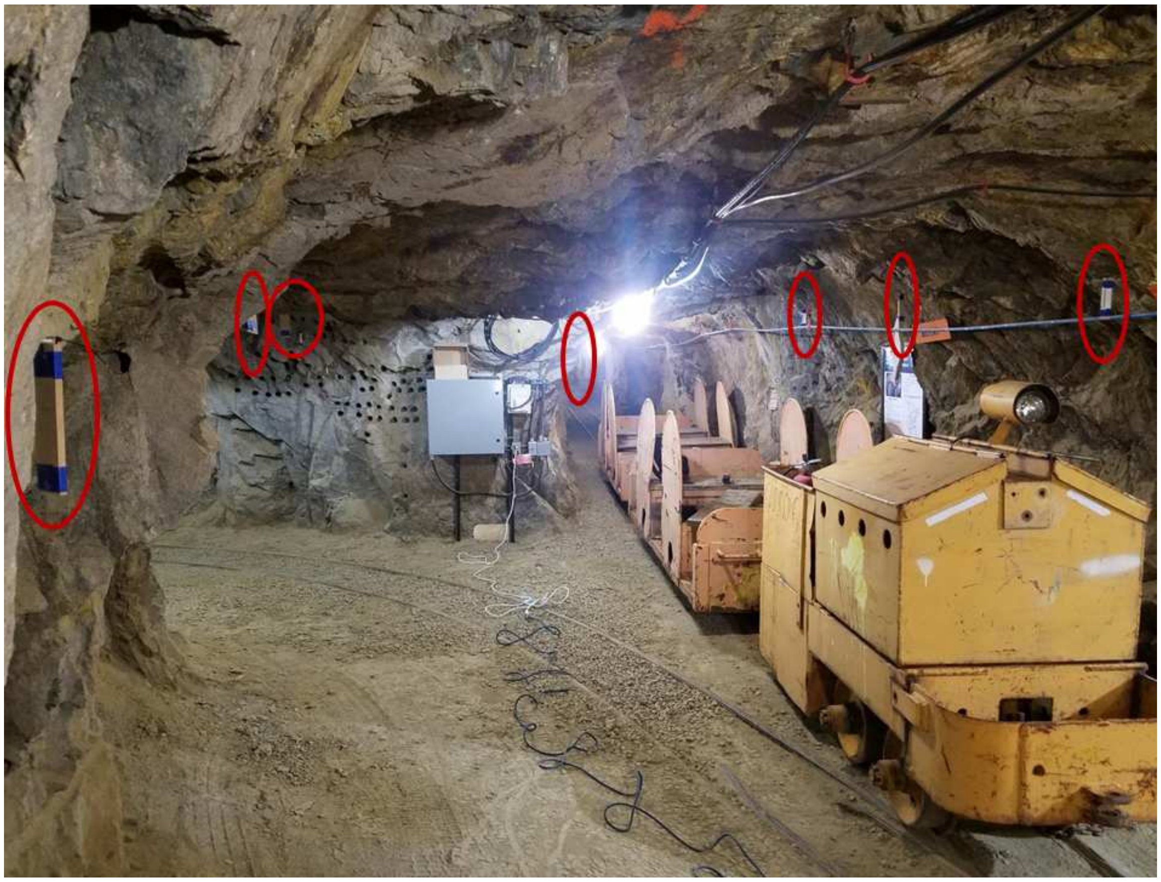


Figure 4.5: Picture of zone 3; an exemplary narrow intersection with the trolley acting as a large scatterer. The red ovals highlight the tags in view.

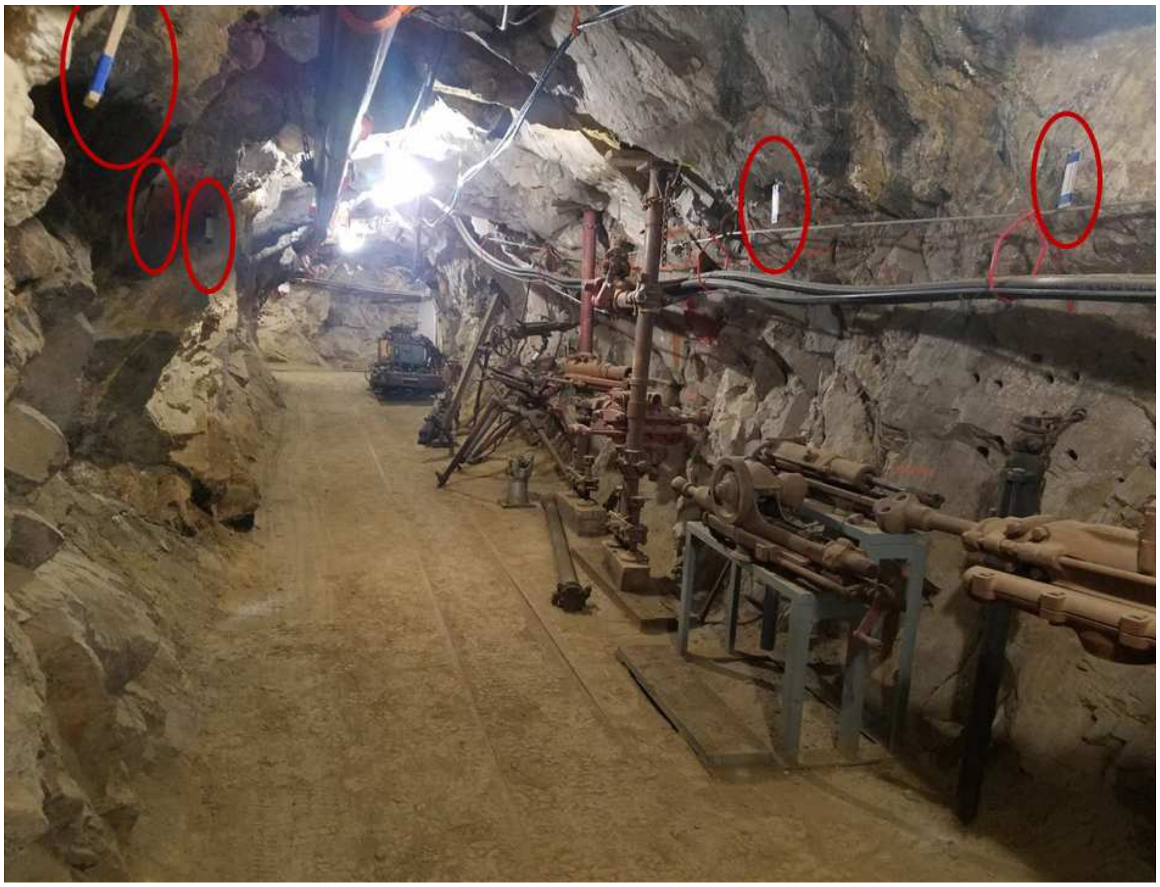


Figure 4.6: Picture of zone 6; a straight section full of mining equipment making it a complex scattering environment. The red ovals highlight the tags in view.

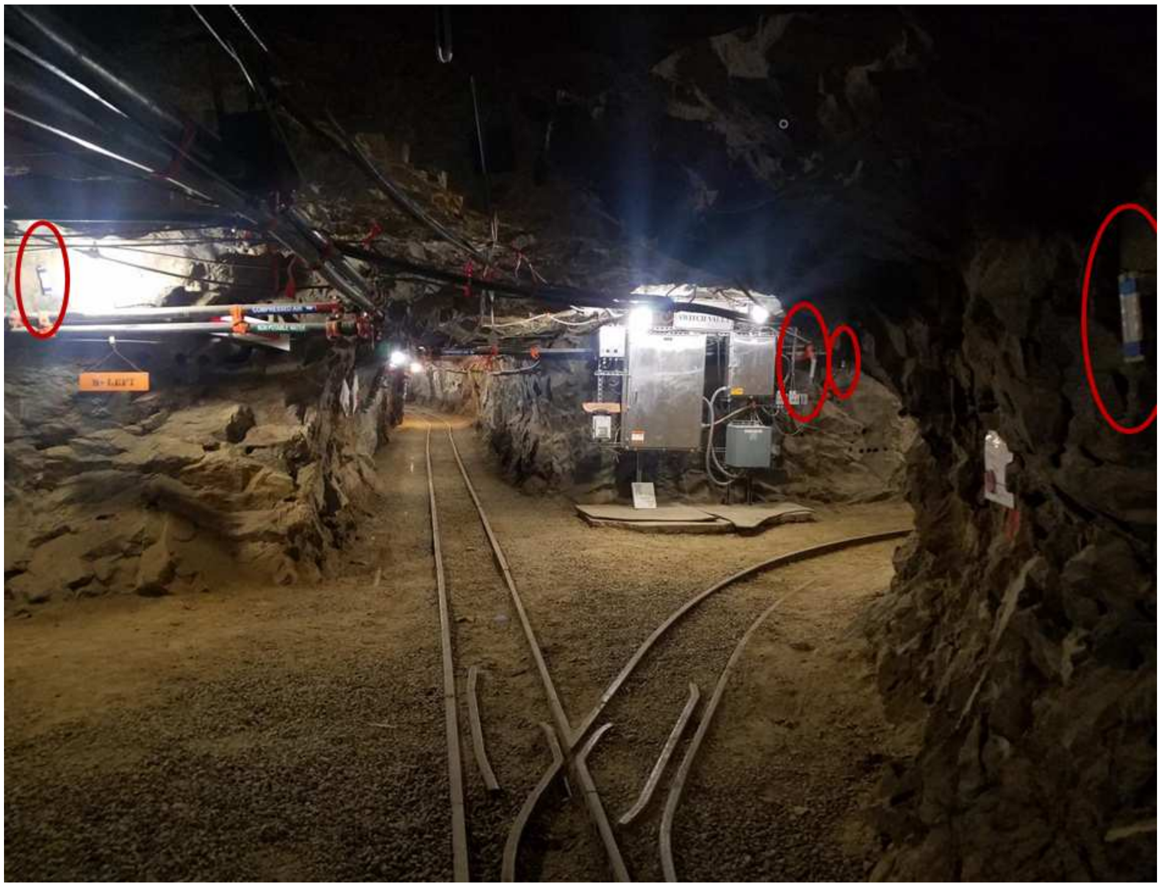


Figure 4.7: Picture of zone 8; an exemplary spacious intersection. The red ovals highlight the tags in view.

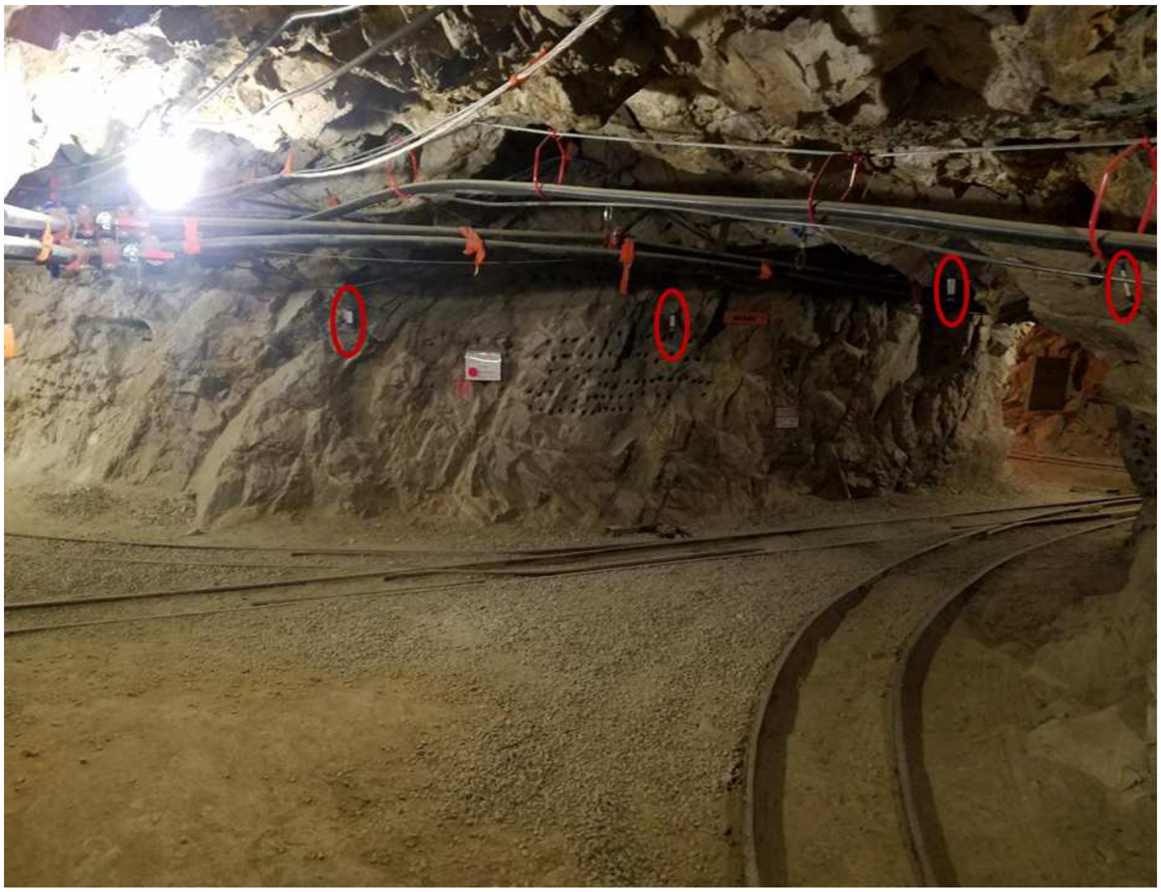


Figure 4.8: Another view of zone 8. The red ovals highlight the tags in view.

The results from this experiment are shown in Figure 4.9 and Figure 4.10. Figure 4.9 show the output plots of the ZCA without map matching. The predicted heading of the user's turns were either overshot or undershot. All of the intersection zones: 3, 4, and 8 with the exception of zone 7 were affected by the gyroscopic drift. Interestingly, the gyroscope seemed to perfectly record the 180 degree turn in zone 9 but then started to drift afterwards. None of the heading drifts were large enough to create bad results post-map matching. After map matching, the heading drifts disappear and the final results in Figure 4.10 look very good with issues only with the lines not connecting to each other. The lines do not connect to each other due to the fluctuations in the user's gait length as well as errors in the RFID initial position. Future improvements to the system will need to use an adaptive gait estimator to mitigate these errors. Zones 1, 4, 5, 6, and 10 were significantly affected by the gait length undershoot. For the Edgar mine pre-characterization experiments, the user's gait length averaged out to be 2.3 feet per step which is a foot greater than the average step distance obtained in Brown building (same user). The step length would have to be 2.7 feet for all lines to connect but would result in zones 7, 8, and 9 overlapping each other. This drastic increase over the Brown building gait length makes sense as the user was stepping over obstacles, such as rail road tracks, and ducking to avoid the low hanging ceilings. This reason is also why zones 10 and 11 have significant gyroscopic deviation for zones that are in the straight sections of the mine. Towards the end of the data acquisition the reader would consistently fail as the user walked past the tags in zones 10 and 11, so for the later experiments in the mine, the user turned their head as they walked through these zones so the main beams of the antennas were never directly in front of the tags at very close range. By doing this, the reader stopped erroring out in these two zones. Because the IMU was mounted at chest level, the heading deviation from turning the head while walking was very low and did not affect the final results. This is a clear reason for why the IMU cannot be placed on the hardhat in future prototype iterations.

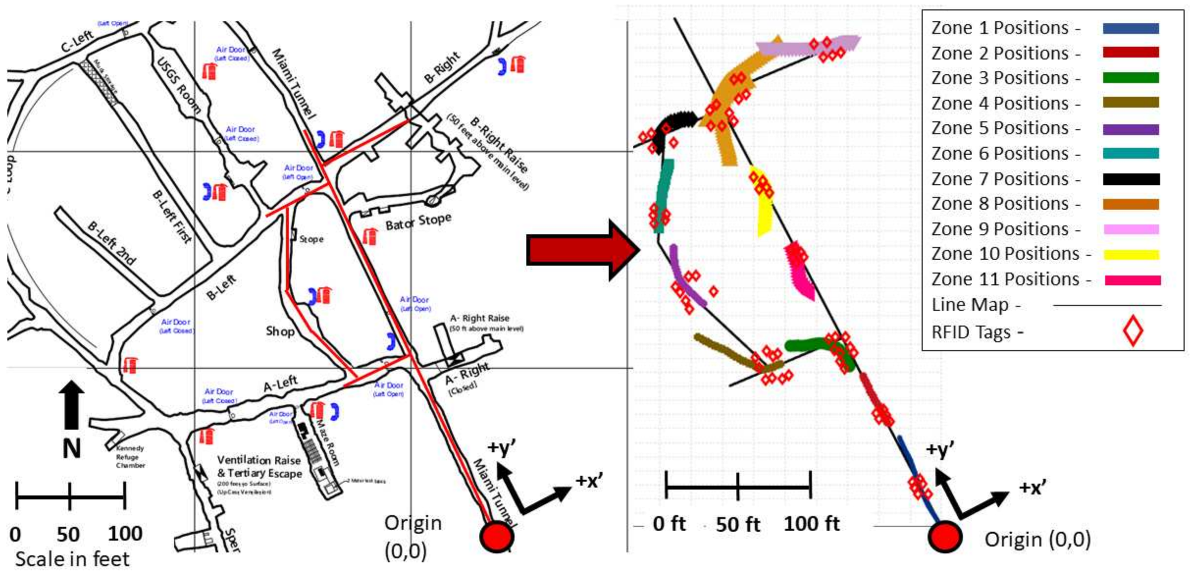


Figure 4.9: Final results of the Edgar mine tests pre-map matching.

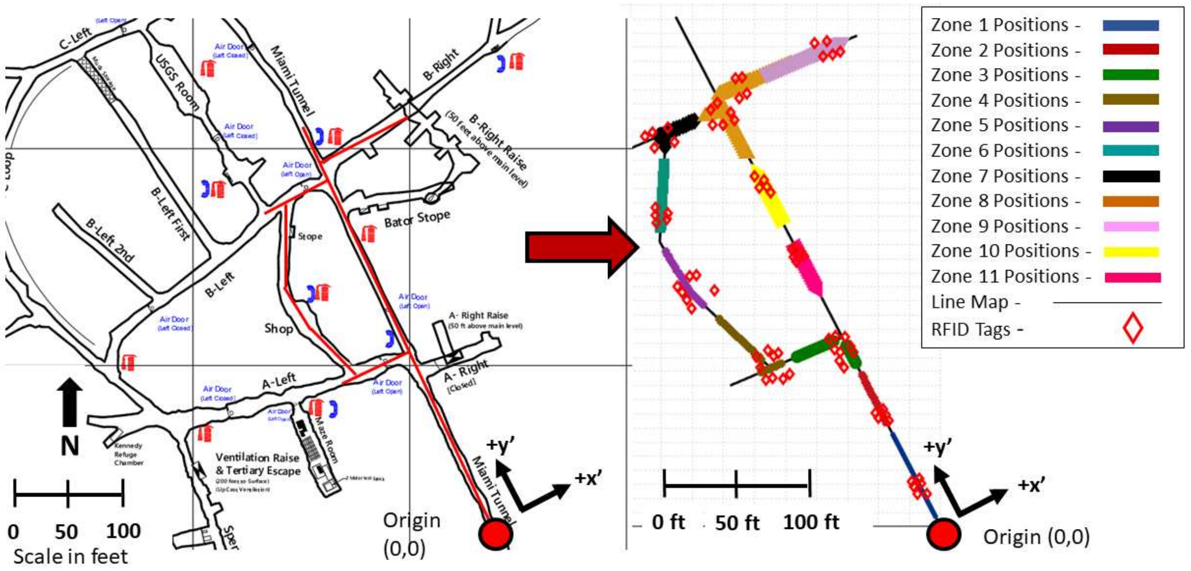


Figure 4.10: Final results of the Edgar mine tests post-map matching.

CHAPTER 5
DISCUSSION, FUTURE WORK, AND CONCLUSION

5.1 Discussion

Due to the mine being a GPS-denied environment, there was no feasible way of using a different system to track the user and truly bound the error on the system. However, data collected with the system revealed four main errors, listed in order of impact: gait length, RFID initial position, gyroscopic drift, and map matching projection. Even with these errors, the localization system functioned successfully showing that both the look-up-table generation process is robust and that tag placement is forgiving.

The average gait length of the user is very easy to obtain and implement in the dead reckoning algorithm at the expense of gait length errors accruing proportional to distance that the zones are apart from each other. As the user progresses through the mine, various scenarios will cause the user to walk at a faster pace, slower pace, wider pace, or narrower pace. For the Edgar mine results, the lines between zones had around 10 feet of gap that was primarily due to the gait length underestimation because the user had an accelerated gait. In developing the localization system, the hypothesis was that the user's gait length would fluctuate around the average gait length where the errors in the shorter steps would be negated by the larger steps. It is clear now that a more advanced gait estimation algorithm, is needed to fix this issue.

Gyroscopic drift is only obvious before the map matching as it will often show the user walking through a wall. After map matching, this error translates into a shorter line as only the component of the predicted path that is parallel to the line map, is kept. For instance, if the gyroscope drifted to a 90 degree heading difference, and the result was map matched, the user would appear stationary and all points beyond would be error. This is why it is the second most impactful error. Fortunately, the Edgar mine results only had minor heading drifts in some of the intersection zones which the map matching algorithm was able to correct for.

The RFID initial position is the anchor point for the dead reckoning estimates in the zones. The dead reckoning calculations that use the IMU data before the initial position reference this point and the dead reckoning calculations that use the IMU data after the initial position also reference this point. If the initial position is significantly off, then every point on the line will have a 2D offset. Again, this error is really only visible before the map matching but it is obvious in the map matched results whenever the user appeared to have jumped a corner in an intersection or if the user appears to have gone a foot or two down a route that was different from the one they actually went down. The Edgar mine results show this error occurring in Zones 3 and 4 where the user jumped onto the path from the path they turned from but this does not contribute to the error significantly. However, in the results for Brown building, Figure 3.20, this error is substantial. In this figure, the entire curve for zones 2 is 2-4 feet below and to the right of the curve which adds to the discontinuity between zones 2 and 3 when map matching is applied. On this figure, zone 1 is also shifted above the path by 4 to 6 feet but this shift is corrected by the map matching. Post-map matching, the lateral shift on the path is the only error the initial position contributes on zone 1. This error is minor on straight sections and significant on intersections.

As the old saying in the simulation community goes, 'garbage in, garbage out', the map matching algorithm relies heavily on the inputs being as accurate as they can be. Projecting onto a dimension by definition removes information, which translates into error. To minimize the error, minimize the projection distance. With significant angular difference between the predicted path and the map's path, the projection will have distance loss proportional to the angular difference between paths. Intersections are particularly tough on map matching as there are numerous lines coming together at a point and any error from the RFID initial position offset will cause the algorithm to place a point on a line that is completely different from the user's traveled one. Overshoots of the user's gait estimate led to overshoots in the predicted path which when map matched, could be placed on the wrong line in the map. Since the predicted positions for both the Brown building and Edgar mine were decent, the map matching algorithm did not need to correct much but it did improve the user's visual confidence in interpreting the final output.

Throughout the mine, the look-up-table held up to the changes in the environment. This proves that combining multiple RSSi vs. distance measurements taken in diverse environments with various antenna orientations and then fitting a look-up-table to all of them together was a robust way to create the look-up-table. The shop is a complex environment from an RF perspective as it is a large opening with rows of large metal machines in the middle of it, which can scatter RFID signals. The intersections were expansive, deviating from the waveguide effect that was a characteristic of the narrow tunnels and still the RFID initial position worked. Besides the issues in Edgar mine's zones 10 and 11 where the antenna got too close to the tags, the RFID 2D algorithm correctly placed the user's 2D position in all of the narrow straight section zones. Even in zones 10 and 11 when user turned their head so the antennas read the tags off-broadside, the system could still lock onto an initial position inside each zone.

The tags in the mine were placed haphazardly throughout each zone, in relatively low numbers. There was no need for precise spacing between tags nor was there a need for optimal positioning of the tags within a zone to make it easier for the RFID 2D algorithm, just the criteria that at least 3 tags could be read within a specified time window at some point in the zone. Some tags were placed in the troughs of the mine walls while some were placed on the pieces that jutted out. Some tags had significant loading from the surrounding rock and some tags were practically hanging in free space. Even with this casual tag placement, the algorithm never needed to use the back up initial condition that was established in the pre-characterization zone connection array. Every zone had a point where at least three unique tags could be read in rapid succession. This speaks to the ease at which any user can setup the environment as well as how resilient the system is to the environmental influences on the wireless channel.

The system's zone based localization successfully bounded the errors from both the IMU and RFID positioning. Using this device, the miner will not only know what zone they were in but will also have a fine grained estimate of their position inside of a zone. The errors discussed in this section are significant, but there are many hardware and software improvements that will improve mitigate them. Some of these improvements will be discussed in the following sections.

5.2 RFID System Improvements

This system relies heavily on the RFID to determine the zone the presence of a user as well as determine the initial position inside a zone. Currently, the RFID hardware is prone to erroring out and the RFID 2D localization algorithm will sometime give an initial position with high error. If the RFID hardware can be made more resilient, and the associated RFID software can be more reliable then future iterations of the localization system can fuse the IMU and RFID estimates whenever there are three unique tags in view, not just for the initial position. The main improvements are: more accurate and precise RFID readings; incorporate more information into the tag reads such as the angle-of-arrival (AoA), and time-of-arrival (ToA); use the tag's changing phase as part of the algorithm; get smaller readers and smaller antennas so the entire RF system can be mounted on the helmet; and finally, improve the read range. This section will suggest some hardware improvements to the system that will enable the improvements mentioned above.

The current antennas on the helmet are circularly polarized, directive, and have high gain but are also large, and relatively heavy. It will be tough to beat the gain performance of the antennas as they can only be made so small for 915 MHz, however, antennas that conform to the shape of the helmet would be much more desirable even at the cost of 1-2 dB of gain loss. Some examples of flexible and conformal antennas are discussed in [48], [49], and [50]. Hardhat (helmet) specific antennas are discussed in [51], [52]. Beyond making the hardhat antennas smaller and sleeker, if a circular or semi-circular antenna phased array is placed onto the helmet it can provide angle-of-arrival information. Making a small, conformal, directive, phased array at 915 MHz to go on a helmet is no trivial task and this could be an area of high impact to the antenna community and the first-responder community. An example of a body-wearable with a circularly configured phased array is described in [53]. Obtaining angle-of-arrival information could add a significant increase to the accuracy of the localization code. Since the tag 2D coordinate is known, the angle from the AoA and the distance from the RSSi would be all that is needed to determine the user's 2D coordinate. [24], [54], and [55] are examples of algorithms that determine AoA and use it for object tracking.

The current linear RFID tags work really well in the mine even with the loading from the rock being in close proximity to them. Because the tags are linear and the reader antennas are RHCP, there is a half-power loss (3dB) when the tag reads the inquiry signal from the reader and a half-power loss when the reader receives the tag's radiated information. If the tag was circularly polarized then there would be no loss due to polarization mismatch. Some circularly polarized tags are described in [56], [57], and [58]. The dipole-like radiation pattern of the linear tags will have light up a significant portion of the mine's rock walls in the azimuth plane of the tag. Due to the random tag placement, the radiation patterns of the tags will have varying levels of constructive and destructive interference caused by the close proximity of the rock, distorting the radiation pattern in unpredictable ways. The RFID tags can be further improved by having a ground plane behind them that is resistant to the loading effects of the rock, like the ones described in [56] and [58]. This will result in the tags having a much more predictable radiation pattern, less loss from the rock in the near-field, and higher directivity. The tags could also be designed to be chipless, which gives the designer much more flexibilty to the frequency, modulation scheme, and the information that is encoded in the tag. With chipless RFID, the modulation of the signal could lead to better read range and lower bit error rate inside of the mine, as described in [59]. The downside to chipless RFID is that the RFID readers will need to be replaced by software defined radios which adds considerable more complexity to the system and increased development time. [60], References [61], and [62] are all examples of chipless RFID systems.

The system in its current state must be contained on the vest due to the size of all of the components. The RFID readers are by far the largest components as they use the large M6e 4 chip. There is a small arduino hat from [63] that shrinks the board down significantly, as the expense of having a maximum output power of 27 dBm instead of 30dBm. If the CP tags were implemented, then the 3 dB of gain from the matched polarization would make up for the 3 dB of loss by switching readers. If the arduino is unnecessary then small dedicated boards with just two M6e nano chips could be made as a hat for the Raspberry Pi. With this improvement, it is possible for the entire system to fit on a helmet or on a belt.

5.3 IMU Improvements

The gait length estimate caused significant error in the final 2D position but there are many solutions in the literature that can mitigate this error. Dead reckoning with the IMU is an entire area of research by itself with many researchers using Kalman filters, statistical models, autocorrelation, pose estimation, and many other techniques to improve the estimate. The first place to start when improving the gait length estimate is with the acceleration data of the IMU. [64] and [65] both performs rigorous analyses on the accelerometer waveforms to accurately determine a user's step distance based on the trunk movement. [66] uses wavelet decomposition, and Kalman filters to determine the step length for each leg so if the user has a disability or injury results in a non-symmetric stride this algorithm will still be able to obtain the correct gait value. [67] uses unbiased autocorrelation analysis of the accelerometer data to estimate the user's step and stride regularity as well as symmetry of the steps but would be restricted to being purely a post-processing solution. [68] suggests training a hidden Markov model and a Gaussian mixture model to classify when the user is in various states such as walking, running, walking up an incline, etc. which will allow for a more precise gait length estimate for various tasks. All of these algorithms can be used in conjunction with one another to improve the dead reckoning estimate of the system.

5.4 Towards a Real-Time Algorithm

This localization system was developed to give positioning estimates in conjunction with a dust monitor. The user only obtains the data from the dust monitor when the data acquisition period is over and post-processes the results. Since dust monitoring was not real-time, the localization system did not need to be real-time to solve the problem. By modifying the system to be real-time, it could be used to inform the miner of their location in the mine which will allow for other applications. Once the real-time algorithm is operational it could be integrated into the existing wireless communication infrastructure in the mine such as the leaky feeder line, allowing for the system to relay the miner's position back to the surface.

The presented localization algorithm can be made real-time by converting the MATLAB functions into ones that can be integrated onto a programmable interface controller (PIC) or written in Python to run on the Raspberry Pi. Figure 5.1 shows the block diagram of how the real-time version of this algorithm may work, assuming that no hardware or software improvements were implemented from the previous two sections. This high level diagram does not assume any hardware nor does it suggest optimal integration. The IMU data and RFID acquisition must sample as fast as they can with no interruptions, therefore, they need to run in parallel with a central function that uses their data to perform the dead reckoning and map matching. Both the RFID and IMU data acquisition functions will take care of very basic operations within themselves. The IMU function will take care of determining when steps occur and keeping a cumulative heading reading. The RFID function will take care of determining when three unique tags within a specified time window are read, finding the zone and tag information that will be used by the rest of the program, and resetting the initial heading of the IMU. The central program will update the user's position every time it gets a step value from the IMU; this involves using the step length for distance, using the cumulative heading for direction, and the mine map to immediately place the coordinate onto the path. This central function will also look for the RFID function to send the initial 2D position. Once it has the initial position, it will update the position estimate and all future steps in the same zone will be referenced to this initial position. At the end of each step calculation, the system will display where the user is on the map.

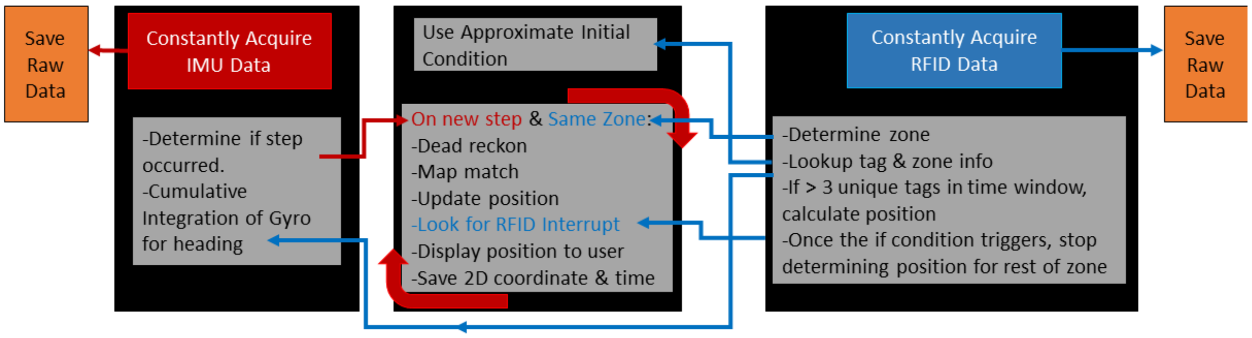

Save Raw Data
Constantly Acquire IMU Data
-Determine if step occurred.
-Cumulative Integration of Gyro for heading
Use Approximate Initial Condition
On new step & Same Zone:
-Dead reckon
-Map match
-Update position
-Look for RFID Interrupt
-Display position to user
-Save 2D coordinate & time
Constantly Acquire RFID Data
-Determine zone
-Lookup tag & zone info
-If > 3 unique tags in time window, calculate position
-Once the if condition triggers, stop determining position for rest of zone
Save Raw Data

Figure 5.1: Block diagram for the proposed real-time algorithm.

5.5 Conclusion

The presented localization system successfully localizes the user inside an underground mine with a minimal amount of added infrastructure. Even with sparse tag placement just in the zones, the system had enough information to provide the initial position of the user without the need for an approximate position defined by the user. The IMU provided fine detail of the user's position in and between zones, once its errors were bounded by the zone RFID position estimate. The zone structure kept the estimated positions with large errors from influencing the position estimates of future zones. The code provided in this thesis allows for the system to work in an arbitrary sized mine with a massive number of zones each with many tags. Even though there are many improvements to be made on the hardware and software levels, this modular system provides future researchers a way to insert their IMU or RFID algorithms without changing the entirety of the code. Hardware improvements will reduce the size of the system, improve the read range, and increase the reliability. The conversion of the algorithm from a post-processing one to a real-time one will allow for a plethora of other applications inside of the mine. This research advances underground positioning by demonstrating a simple, robust, modular system that uses the minimum amount of added infrastructure to provide the position of a miner to any environmental sensor thereby allowing for surgical hazard mitigation and the increased safety for all in the mine.

REFERENCES CITED

[1] P. Maier, P. Hartlieb, and J.F. Brune. Laboratory Scaled Coal Dust Explosions and Physical Test Results for CFD Explosion Models. BHM Berg- und Hüttenmännische Monatshefte, 165(6):265–269, 2020. ISSN 1613-7531. URL https://doi.org/10.1007/s00501-020-00985-0.

[2] J.F. Brune, K.L. Cashdollar, and R.K. Zipf. Explosion Prevention in United States Coal Mines. In Proceedings of the 32nd international conference of safety in mines research institutes, pages 1–7, 2007.

[3] J.F. Brune. Mine Ventilation Networks Optimized for Safety and Productivity. In Advances in Productive, Safe, and Responsible Coal Mining, pages 83–99. Woodhead Publishing, 2019. ISBN 978-0-08-101288-8. doi: https://doi.org/10.1016/B978-0-08-101288-8.00005-5. URL https://www.sciencedirect.com/science/article/pii/B9780081012888000055.

[4] Thermo Fisher Scientific. PDM3700 Instruction Manual, 2016. URL https://assets.thermofisher.com/TFS-Assets/LSG/manuals/EPM-manual-PDM3700.pdf.

[5] N. Tahir, M.Md. Karim, K. Sharif, F. Li, and N. Ahmed. Quadrant-Based Weighted Centroid Algorithm for Localization in Underground Mines. In Proceedings of the 2013 13th International Conference on Wireless Algorithms, Systems, and Applications, pages 462–472. Springer International Publishing, 2018. ISBN 978-3-319-94268-1.

[6] H. Xu, F. Li, and Y. Ma. A Zigbee-Based Miner Localization System. In Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pages 919–924, 2012. doi: 10.1109/CSCWD.2012.6221931.

[7] P. Lin, Q. Li, Q. Fan, X. Gao, and S. Hu. A Real-Time Location-Based Services System Using WiFi Fingerprinting Algorithm for Safety Risk Assessment of Workers in Tunnels. Mathematical Problems in Engineering, 2014(371456), 2014. URL https://doi.org/10.1155/2014/371456.

[8] F. Pereira. Positioning Systems for Underground Tunnel Environments. PhD thesis, University of Porto, Porto, Portugal, 2016.

[9] A. Dehghan Firoozabadi, C. Azurdia-Meza, I. Soto, F. Seguel, N. Krommenacker, D. Iturralde, P. Charpentier, and D. Zabala-Blanco. A Novel Frequency Domain Visible Light Communication (VLC) Three-Dimensional Trilateration System for Localization in Underground Mining. Applied Sciences, 9(7), 2019. ISSN 2076-3417. doi: 10.3390/app9071488. URL `https://www.mdpi.com/2076-3417/9/7/1488`.

[10] A. RayChowdhury, A. Pramanik, and G. Roy. New Approach for Localization and Smart Data Transmission Inside Underground Mine Environment. SN Applied Sciences, 3, 06 2021. doi: 10.1007/s42452-021-04589-2.

[11] P. Palacios Játiva, C.A. Azurdia-Meza, M. Román Cañizares, D. Zabala-Blanco, and C. Saavedra. Propagation Features of Visible Light Communication in Underground Mining Environments. In International Conference on Applied Technologies, pages 82–93, Cham, 2020. Springer International Publishing. ISBN 978-3-030-42531-9.

[12] Q. Niu, X. Yang, and Y. Yin. IPL: Image-Assisted Person Localization for Underground Coal Mines. Sensors, 18(11), 2018. ISSN 1424-8220. doi: 10.3390/s18113679. URL `https://www.mdpi.com/1424-8220/18/11/3679`.

[13] N. Xiao, X. Li, G. Qin, S. Ma, and L. Zhang. Localization System Based on RFID and GIS for Underground Moving Targets. In 2008 IEEE International Conference on Mechatronics and Automation, pages 808–813, 2008. doi: 10.1109/ICMA.2008.4798861.

[14] K. Hlophe. An Embedded Underground Navigation System, 2011.

[15] A. Poulose, O. Eyobu, and D. Han. An Indoor Position-Estimation Algorithm Using Smartphone IMU Sensor Data. IEEE Access, 7:11165–11177, 2019.

[16] B. Li, K. Zhao, S. Saydam, C. Rizos, Q. Wang, and J. Wang. Positioning Technologies for Underground Mines. Far East Journal of Electronics and Communications, 18(6): 871 – 893, 2018. URL `http://dx.doi.org/10.17654/ec018060871`.

[17] N.J. Lavigne and J.A. Marshall. A Landmark-Bounded Method for Large-Scale Underground Mine Mapping. Journal of Field Robotics, 29(6):861–879, 2012. doi: https://doi.org/10.1002/rob.21415. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21415`.

[18] S. Rusu. Real-Time Localization In Large-Scale Underground Environments Using RFID-Based Node Maps. M.S. thesis, Department of Mechanical and Aerospace Engineering, Carleton University, Ottawa, Canada, 2011. https://www.collectionscanada.gc.ca/obj/thesescanada/vol2/002/MR83067.PDF.

[19] J. Larsson, M. Broxvall, and A. Saffiotti. A Navigation System for Automated Loaders in Underground Mines. Springer Tracts in Advanced Robotics, 25:129–140, 01 2005. doi: 10.1007/978-3-540-33453-8_12.

[20] P.K. Mishra, R.F. Stewart, M. Bolic, and M. Yagoub. RFID in Underground-Mining Service Applications. IEEE Pervasive Computing, 13(1):72–79, 2014. doi: 10.1109/MPRV.2014.14.

[21] L. Ni, D. Zhang, and M. Souryal. RFID-Based Localization and Tracking Technologies. IEEE Wireless Communications, 18(2):45–51, 2011. doi: 10.1109/MWC.2011.5751295.

[22] W. Zhu, J. Cao, Y. Xu, L. Yang, and J. Kong. Fault-Tolerant RFID Reader Localization Based on Passive RFID tags. IEEE Transactions on Parallel and Distributed Systems, 25(8):2065–2076, 2014. doi: 10.1109/TPDS.2013.217.

[23] A. Parr, R. Miesen, F. Kirsch, and M. Vossiek. A Novel Method for UHF RFID Tag Tracking Based on Acceleration Data. In 2012 IEEE International Conference on RFID, pages 110–115, 2012. doi: 10.1109/RFID.2012.6193037.

[24] A. Zarrini, A.Z. Elsherbeni, and J.F. Brune. Direction of Arrival Tag Response for RFID Localization. Applied Computational Electromagnetics Society (ACES), 34(2): 323–325, 2019.

[25] NIOSH. Reverse Implementation of Radio Frequency Identification (RFID) Technology for Personnel Tracking in Underground Mines. *Milestones in Mining Safety and Health Technology*, 543(2011-209), 2011.

[26] C. Sunderman and J. Waynert. An Overview of Underground Coal Miner Electronic Tracking System Technologies. In 2012 IEEE Industry Applications Society Annual Meeting, pages 1–5, 2012. doi: 10.1109/IAS.2012.6374081.

[27] R. Haupt. Wireless Communications Systems. John Wiley & Sons, Hoboken, NJ, 2020.

[28] J. Lindblom, B. Hagman, and L. Crotser. SparkFun MPU-9250 Digital Motion Processor (DMP) Arduino Library. GitHub, 2019. URL `https://github.com/sparkfun/SparkFun_MPU-9250-DMP_Arduino_Library`.

[29] Analog Devices. Digital Accelerometer, ADXL345, 5 2009. Rev. 0.

[30] InvenSense Inc. ITG-3200, 3 2010. Rev. 1.4.

[31] Honeywell. 3-Axis Digital Compass IC HMC5883L, 5 2009. Rev. 0.

[32] J. Wang, Y. Guo, L. Guo, B. Zhang, and B. Wu. Performance Test of MPMD Matching Algorithm for Geomagnetic and RFID Combined Underground Positioning. IEEE Access, 7:129789–129801, 2019. doi: 10.1109/ACCESS.2019.2926098.

[33] Q. Huang, X. Zhang, and J. Ma. Underground Magnetic Localization Method and Optimization Based on Simulated Annealing Algorithm. In 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), pages 168–173, 2015. doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.47.

[34] J. Lindblom. 9DoF Razor IMU M0 Hookup Guide. SparkFun Electronics, 2016. URL `https://learn.sparkfun.com/tutorials/9dof-razor-imu-m0-hookup-guide?_ga=2.112718212.1838163393.1580079595-1050252564.1484965463#resources--going-further`.

[35] A.R. Jimenez, F. Seco, C. Prieto, and J. Guevara. A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu, 2009.

[36] P. Gotthard. Python Wrapper for the ThingMagic Mercury API. GitHub, 2016. URL `https://github.com/gotthardp/python-mercuryapi`.

[37] Pololu. Pololu 9V Step-Up/Step-Down Voltage Regulator S18V20F9, 2021.

[38] MPS. MP2307, 5 2008. Rev. 1.9.

[39] ThingMagic. M6e Family Hardware Guide, 3 2016. 09 Revision A.

[40] H. Ramezani. Single/Multiple target Localization. MATLAB Central File Exchange, 2021. URL `https://www.mathworks.com/matlabcentral/fileexchange/33792-single-multiple-target-localization?s_tid=srchtitle`.

[41] A. Weiss. *format_plot.m*. GitHub, 2021. URL `https://github.com/Sheekaboom/WeissTools/tree/master/WeissTools/MATLAB`.

[42] B. Béjar, P. Belanovic, and S. Zazo. Distributed Gauss-Newton Method for Localization in Ad-Hoc Networks. In 43rd Asilomar Conference on Signals, Systems, and Computers, 7-10 Nov 2010; Pacific Grove, pages 1452–1454, 11 2010. doi: 10.1109/ACSSC.2010.5757776.

[43] R. Wang. Gauss-Newton Algorithm for Nonlinear Models. `http://fourier.eng.hmc.edu/e176/lectures/NM/node36.html`, 2015.

[44] N. Yoder. $peakfinder(x0, sel, thresh, extrema, includeEndpoints, interpolate)$. MATLAB Central File Exchange, 2016. URL `https://www.mathworks.com/matlabcentral/fileexchange/25500-peakfinder-x0-sel-thresh-extrema-includeendpoints-interpolate`.

[45] T.T. Pham and Y.S. Suh. Histogram Feature-Based Approach for Walking Distance Estimation Using a Waist-Mounted IMU. IEEE Sensors Journal, 20(20):12354–12363, 2020. doi: 10.1109/JSEN.2020.2999930.

[46] D. Bernstein and A. Kornhauser. An Introduction to Map Matching for Personal Navigation Assistants. Technical report, New Jersey TIDE Center, Newark, NJ, 1996.

[47] J. Perina. Shortest Distance Between a Point and a Line Segment. Stack Overflow, 2011. URL `https://stackoverflow.com/questions/849211/shortest-distance-between-a-point-and-a-line-segment`.

[48] H. Subbaraman. Printable Silicon Nanomembranes for Solar-Powered, Bi-Directional Phased-Array-Antenna Communication System on Flexible Substrates. Technical Report AFRL-OSR-VA-TR-2013-0081, Air Force Research Laboratory, Arlington, Virginia, 2013.

[49] E. Crespo-Bardera, A. Garrido Martin, A. Fernandez-Duran, and M. Sanchez-Fernandez. Design and Analysis of Conformal Antenna for Future Public Safety Communications: Enabling Future Public Safety Communication Infrastructure. IEEE Antennas and Propagation Magazine, 62(4):94–102, 2020. doi: 10.1109/MAP.2020.3000711.

[50] E. Çelenk and N.T. Tokan. Frequency Scanning Conformal Sensor Based on SIW Metamaterial Antenna. IEEE Sensors Journal, pages 1–1, 2021. doi: 10.1109/JSEN.2021.3075556.

[51] J.J.H. Wang, J.K. Tillery, K.E. Bohannan, and G.T. Thompson. Helmet-Mounted Smart Array Antenna. In IEEE Antennas and Propagation Society International Symposium 1997. Digest, volume 1, pages 410–413 vol.1, 1997. doi: 10.1109/APS.1997.630181.

[52] S. Shabina. Smart Helmet Using RF and WSN Technology for Underground Mines Safety. In 2014 International Conference on Intelligent Computing Applications, pages 305–309, 2014. doi: 10.1109/ICICA.2014.105.

[53] Ł. Januszkiewicz, P. Di Barba, and S. Hausman. Optimal Design of Switchable Wearable Antenna Array for Wireless Sensor Networks. Sensors, 20(10), 2020. ISSN 1424-8220. doi: 10.3390/s20. URL `https://www.mdpi.com/1424-8220/20/10/2795`.

[54] K. Honda, D. Iwamoto, and K. Ogawa. Angle of Arrival Estimation Embedded in a Circular Phased Array 4 × 4 MIMO Antenna. In 2017 IEEE Asia Pacific Microwave Conference (APMC), pages 93–96, 2017. doi: 10.1109/.

[55] C.R. Karanam, B. Korany, and Y. Mostofi. Magnitude-Based Angle-of-Arrival Estimation, Localization, and Target Tracking. In 2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), pages 254–265, 2018. doi: 10.1109/IPSN.2018.00053.

[56] H. Chen, S. Kuo, C. Sim, and C. Tsai. Coupling-Feed Circularly Polarized RFID Tag Antenna Mountable on Metallic Surface. IEEE Transactions on Antennas and Propagation, 60(5):2166–2174, 2012. doi: 10.1109/TAP.2012.2189702.

[57] H.H. Tran, S.X. Ta, and I. Park. A Compact Circularly Polarized Crossed-Dipole Antenna for an RFID Tag. IEEE Antennas and Wireless Propagation Letters, 14: 674–677, 2015. doi: 10.1109/LAWP.2014.2376945.

[58] Y. Chen and A.Z. Elsherbeni. Circularly Polarized RFID Tag Antenna Design for Underground Localization System. In 2021 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM), pages 205–206, 2021. doi: 10.23919/USNC-URSINRSM51531.2021.9336514.

[59] A.E. Forooshani, S. Bashir, D.G. Michelson, and S. Noghanian. A Survey of Wireless Communications and Propagation Modeling in Underground Mines. IEEE Communications Surveys Tutorials, 15(4):1524–1545, 2013. doi: 10.1109/SURV.2013.031413.00130.

[60] R. Anee and N.C. Karmakar. Chipless RFID Tag Localization. IEEE Transactions on Microwave Theory and Techniques, 61(11):4008–4017, 2013. doi: 10.1109/TMTT.2013.2282280.

[61] R. Rezaiesarlak and M. Manteghi. A Space-Frequency Technique for Chipless RFID Tag Localization. IEEE Transactions on Antennas and Propagation, 62(11): 5790–5797, 2014. doi: 10.1109/TAP.2014.2350523.

[62] N. Zhang, M. Hu, L. Shao, and J. Yang. Localization of Printed Chipless Rfid in 3-d Space. IEEE Microwave and Wireless Components Letters, 26(5):373–375, 2016. doi: 10.1109/LMWC.2016.2549264.

[63] N. Seidle. Simultaneous RFID Tag Reader Hookup Guide. SparkFun Electronics, 2017. URL `https://learn.sparkfun.com/tutorials/simultaneous-rfid-tag-reader-hookup-guide/all`.

[64] Z. Wiebren and H. At L. Assessment of Spatio-Temporal Gait Parameters from Trunk Accelerations During Human Walking. Gait and Posture, 18(2):1–10, 2003. ISSN 0966-6362. doi: https://doi.org/10.1016/S0966-6362(02)00190-X. URL `https://www.sciencedirect.com/science/article/pii/S096663620200190X`.

[65] S.M. Rispens, M. Pijnappels, K.S. van Schooten, P.J Beek, A. Daffertshofer, and J.H. van Dieën. Consistency of Gait Characteristics as Determined from Acceleration Data Collected at Different Trunk Locations. Gait and Posture, 40(1):187–192, 2014. ISSN 0966-6362. doi: https://doi.org/10.1016/j.gaitpost.2014.03.182. URL `https://www.sciencedirect.com/science/article/pii/S096663621400277X`.

[66] A. Köse, A. Cereatti, and U. Della Croce. Bilateral Step Length Estimation using a Single Inertial Measurement Unit Attached to the Pelvis. Journal of NeuroEngineering and Rehabilitation, 9, 2012. doi: 10.1186/1743-0003-9-9.

[67] R. Moe-Nilssen and J.L. Helbostad. Estimation of Gait Cycle Characteristics by Trunk Accelerometry. Journal of Biomechanics, 37(1):121–126, 2004. ISSN 0021-9290. doi: https://doi.org/10.1016/S0021-9290(03)00233-1. URL `https://www.sciencedirect.com/science/article/pii/S0021929003002331`.

[68] G. Panahandeh, N. Mohammadiha, A. Leijon, and P. Händel. Chest-Mounted Inertial Measurement Unit for Pedestrian Motion Classification using Continuous Hidden Markov Model. In 2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings, pages 991–995, 2012. doi: 10.1109/I2MTC.2012.6229380.

APPENDIX A
COST BREAKDOWN OF THE PROTOTYPE

Table A.1: List of devices used in the localization prototype and their respective costs.

Device	**Unit Cost**	**Quantity**	**Subtotal**
RFID Development Kit	\$901	2	\$1802
Raspberry Pi Touchscreen	\$61	1	\$61
Razor 9 DoF IMU	\$50	1	\$50
3D Print Filament Spool	\$45	1	\$45
MOLLE Vest	\$44	1	\$44
Raspberry Pi 3 B+	\$40	1	\$40
RHCP Custom Antennas	\$40	2	\$80
RP-TNC Male to SMA Male RF Cable 6ft	\$39	2	\$78
Battery Charger	\$35	1	\$35
Mining Hard Hat	\$32	1	\$32
1/4” Tie Wire Wedge Anchors	\$30	1	\$30
Raspberry Pi Case	\$28	1	\$28
MOLLE Pouches	\$28	2	\$56
12V 7.0 AH Lead Acid Battery	\$26	2	\$52
9V Step-Up/Step-Down Voltage Regulator	\$23	1	\$23
Right Angle SMA Connector	\$15	2	\$30
128 GB SD Card for Raspberry Pi	\$15	1	\$15
Reflective Vest	\$15	1	\$15
UBEC DC/DC Step Converter - 5V 3A	\$10	1	\$10
1/4” MDF Board 2ft x 4ft	\$9	2	\$18
RP-TNC Right Angle Connector	\$9	2	\$18
24 AWG Silicon Wire - 1 foot	\$2	4	\$8
22 AWG Silicon Wire - 1 foot	\$2	4	\$8
5.5/2.1mm Barrel Connector	\$1	2	\$2
AD-229r6 RFID tags	\$0.25	80	\$20
-	-	**Total**	\$2572

APPENDIX B

DATA ACQUISITION CODE

Listing B.1: Bash script code to control the IMU and RFID Python functions.

```
##GET INPUT FROM USER##
echo Enter the runtime:
read runtime

##GRAB THE SYSTEM TIME (since EPOCH) that will be passed into all
    subsequent functions##
start_time=$(date +'%s')

##Print that code is starting##
echo Starting execution of IMU and RFID readers in background for
    $runtime seconds


##RUN THE IMU IN THE BACKGROUND##
#First argument corresponds to the system time
#Second argument corresponds to the user's desired run time
python3 ./imu_mk3_ARGS.py "$start_time" $runtime &


##RUN BOTH READERS IN THE BACKGROUND ##
#First argument corresponds to the last number of the USB port
#Second argument corresponds to the antenna that the reader is
    connected to
#Third argument corresponds to the system time
#Fourth argument corresponds to the user's desired run time
python3 ./rfid_mk5_ARGS.py 0 1 "$start_time" $runtime &
python3 ./rfid_mk5_ARGS.py 1 2 "$start_time" $runtime &
```

Listing B.2: Python acquisition code function to control the M6e readers.

```
#From https://github.com/gotthardp/python-mercuryapi
#Note: to access ports like /dev/ttyUSB0 as a non-root user you may
    need to add this user
#to the dialout group:
#sudo usermod -a -G dialout $USER
```

```
#To find what port the reader is on:
#1. Have reader unplugged
#2. Open terminal and enter: ls -l /dev/tty*
#3. Plug reader in
#4. re-run step before and find the additional entry, it should
#be somethign like: /dev/ttyUSB1 or /dev/ttyUSB0
#5. Change the line: mercury.Reader("tmr:///dev/ttyUSB1") with new path

#ARGUMENTS: com_port  ant_index

#/----------INITIALIZATION CODE----------/
import codecs
import mercury
import time
import sys
reader_serial_string = "tmr:///dev/ttyUSB"+str(sys.argv[1]) #Arg must
    be the # of the USB
reader = mercury.Reader(reader_serial_string)
output_filename = r"/home/pi/Documents/combined_m6e_imu/program_output/
    reader" + str(sys.argv[1]) + "_out.txt"

#/-------------------SETUP READER-------------------------/
tx_power = 3000 #measured in centidBm, max is 3000
ant_index = int(sys.argv[2]) #Which port is the antenna connected to?
reader.set_region("NA")
reader.set_read_plan([ant_index], "GEN2")
reader.set_hop_table([915000]) #Lock Frequency
reader.set_gen2_tari(2)
reader.set_gen2_q(1, 2)
reader.set_read_powers([(ant_index,tx_power)])
#print(reader.get_read_powers())

RUN_TIME = float(sys.argv[4]) #Number of seconds the reader will read
    for

#/-------------------DECLARE TIME START-----------------/
#stopwatch_start = time.time() #Start timer, all time values are
    relative to this.
stopwatch_start = float(sys.argv[3])
save_string_array = []

#/--------------CALLBACK FUNCTION FOR THE START READING SECTION------/
#CSV Format: tag_name,tag_phase,tag_system_time
def mycallback(tag):
    save_string_array.append(str(tag.epc) + "," + str(tag.phase) + ","
      + str(tag.rssi) + "," + str(tag.antenna) + "," + str(time.time()
      -stopwatch_start))
```

```
#/----------------START READING-------------------/
#print("READ START")
reader.start_reading(mycallback)
time.sleep(RUN_TIME) #sets the  time the reader will read for
reader.stop_reading()
#print("READ STOPPED")

#/------------------SAVE FILE----------------/
output_file = open(output_filename,"w+")
for line in save_string_array:
    output_file.write(line)
    output_file.write("\n")

output_file.close()
```

Listing B.3: Python acquisition code function to control the Razor 9DoF IMU.

```
#---------IF THE IMU IS ALREADY IN OUTPUT MODE THEN THE CODE WILL NOT
    WORK--------
#you will know if the imu is off a space when the file only has around
    5 entries in it
import time
import serial
import sys
#-----------DECLARATIONS & INITIALIZATION----------
output_filename = "/home/pi/Documents/combined_m6e_imu/program_output/
    imu_out"

ser = serial.Serial(
        port='/dev/ttyACM0', #Subject to change, Find this with the
            arduino IDE
        baudrate = 115200,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)

counter = 0
stop_time = float(sys.argv[2]) #[s] time to stop reading

space = "␣"

save_string_array = []
```

```
#——————START THE DATA OUTPUT FROM IMU————
stopwatch_start = float(sys.argv[1]) #Start timer, all time values are
    relative to this


ser.write(space.encode("ascii"))

#Read for stop_time duration from IMU
while (time.time()-stopwatch_start) < stop_time:
        x=ser.readline()
        save_string_array.append(str(x.decode().rstrip()) + ", "  + str
            (time.time()-stopwatch_start))
        counter += 1

#————STOP THE DATA OUTPUT FROM IMU & SAVE DATA——————
ser.write(space.encode("ascii"))

output_file = open('%s.txt' % output_filename,"w+")
for line in save_string_array:
    output_file.write(line)
    output_file.write("\n")

output_file.close()
```

APPENDIX C
MATLAB CODE

Listing C.4: Pre-processing script where the user manually enters in the measured data from the various RSSi vs distance experiments and the script then generates a third order polynomial to curve fit the data.

```
%Code combined from:
%https://www.mathworks.com/help/matlab/ref/polyfit.html#d123e1032145
%https://www.mathworks.com/help/matlab/ref/scatter.html
%https://www.mathworks.com/matlabcentral/answers/401297-hello-how-do-i-
    display-the-equation-for-a-polyfit-line-on-this-plot

%% NARROW SECTION
% Defined distances and RSSI Vals
left_tag_lateral_dist = 2.33;%ft
right_tag_lateral_dist = 3.5;%ft

left_tag_parallel_dist = [0,1,2,3];%ft (parallel to the normal vector
    of the tag surface -> distance between lines)
right_tag_parallel_dist = [0,1,2,3,3.5];%ft (parallel to the normal
    vector of the tag surface -> distance between lines)

left_tag_rssi = [-55,-59,-63,-69]; %RSSi (dBm)
right_tag_rssi = [-53,-55,-57,-64,-69]; %RSSi (dBm)

% Distance equation
%Returns C, the straight line distance of a right triangle
calc_dist = @(a,b)(sqrt((a.^2) + (b.^2)));

%Calculate direct distance to each tag
left_tag_dist = calc_dist(left_tag_lateral_dist ,left_tag_parallel_dist)
    ;
right_tag_dist = calc_dist(right_tag_lateral_dist ,
    right_tag_parallel_dist);


%% HEAD ON SECTION
head_on_dist = [1,2,3,4,5,6,7,8,9,10,11]; %Distance (ft)
head_on_rssi = [-50,-55,-56.5,-60.5,-61.5,-67,-68.5,-69,-70,-70.5,-75];
     %RSSi (dBm)
```

```
%Try poly fit
rssi_coords = [head_on_rssi,left_tag_rssi,right_tag_rssi];
dist_coords = [head_on_dist,left_tag_dist,right_tag_dist];
p = polyfit(rssi_coords,dist_coords,3);
x1 = linspace(-75,-45);
y1 = polyval(p,x1);
curve_fit_object = fit(rssi_coords',dist_coords','poly3');

%% Plotting
figure
hold on
grid minor
title('Distance Independent')
plot(left_tag_dist,left_tag_rssi)
plot(right_tag_dist,right_tag_rssi)
plot(head_on_dist,head_on_rssi)
hold off

figure
hold on
grid minor
title('RSSi Independent')
plot(left_tag_rssi,left_tag_dist)
plot(right_tag_rssi,right_tag_dist)
plot(head_on_rssi,head_on_dist)
hold off

figure
hold on
grid minor
scatter(rssi_coords,dist_coords,'dk','LineWidth',2)
plot(x1,y1,'LineWidth',2)
xlabel('RSSi (dBm)')
ylabel('Distance (ft)')
set(gca,'FontSize',24)
xl = [-76,-46];
yl = [0,12];
xlim(xl);
ylim(yl);
xt = 0.15 * (xl(2)-xl(1)) + xl(1);
yt = 0.95 * (yl(2)-yl(1)) + yl(1);
caption = sprintf('y = %f x^3 + %f x^2 + %f x + %f', p(1), p(2), p(3),
   p(4));
text(xt, yt, caption, 'FontSize', 15, 'Color', 'r', 'FontWeight', 'bold
   ');
```

```
hold off

%% Save Look up Table
save('edgar_lut_rhcp1_30dbm.mat','curve_fit_object') %Saves the
    polynomial
```

Listing C.5: Script that takes in a the csv file that contains the tag IDs, the zone they belong to, and their coordinates, and outputs a file that the post-processing code can interpret. This output file has each zone listed with all of the tags associated in it along with the type of zone it is and legacy information on what the heading angles are into and out of the zone. The heading angles are filled with NaNs as they are taken care of with the zone connection file.

```
%This Program Takes in a 'tag_info.csv' file and generates the
%'zone_info.csv' file

%NOTE: 'tag_info.csv' must have the tags in groups with each other, do
    not
%scatter around the list!! (i.e. 1;1;1;2;2;2;2;2;2;      not
    1;2;2;1;1;2;2;2;1;2;2)
%ALSO KEEP THEM IN ZONE NUMERICAL ORDER!!
%(i.e) 1;1;2;2;2;3;3;4       NOT      2;1;3;4;3;2;2;1

%% /////////////INPUTS//////////////
location = '/Users/rj/Documents/RFID_project/edgar_mine_results/code/';
     %WINDOWS
filename_tags = 'tag_info.csv';
output_filename = 'zone_info.csv';

%In numerical zone (i.e. zone # 1,2,3)order, enter the zone types in
    matrix format
%'s' for straight section, 'u' for undefined section
zone_types = ['u';'u';'u';'u';'u';'u';'u';'u';'u';'u';'u'];
%The fzt (forward zone transitions) headings and rzt (reverse zone
    transitions) headings are legacy implementation to the code and are
    used in older data sets where the zone connection diagram is
    unavailable. So these arrays are kept but are filled with NaNs as
    they are not needed.
zone_fzt_headings = [NaN;NaN;NaN;NaN;NaN;NaN;NaN;NaN;NaN;NaN;NaN]; %
    Array of forward zone transition headings if straight section! If
    not, then put NaN
zone_rzt_headings = [NaN;NaN;NaN;NaN;NaN;NaN;NaN;NaN;NaN;NaN;NaN]; %
    Array of reverse zone transition headings if straight section! If
    not, then put NaN
%Note: if other logic is added to treat various zones differently, add
    it
%above with its own special character



%% //////////////EXECUTION CODE///////
%ASSUMING THE ORTHOPEDIC RFID TAGS
%In 2D matrix format, list the last three digits of the tag identifier
%Rows are tag IDs; columns are the zones in numerical order
%Blank entries must have 'NaN' put in them
opts = detectImportOptions([location filename_tags]);
tag_info = readtable([location filename_tags],opts);

%Convert table to array
tag_info_array = table2array(tag_info);
%Get the tag info array into multiple arrays corresponding to the right
%zones
unique_zones = 0; %Init
for index = 1:1:size(tag_info_array,1) %Loop through the number of tags
    if tag_info_array(index,1) == unique_zones %If the tag belongs to a
         zone already seen
         %Add tag info to the list of the zone it belongs to
         zone_cell_array{unique_zones} = cat(1,zone_cell_array{
            unique_zones},tag_info_array(index,:));
    elseif tag_info_array(index,1) ~= unique_zones %If tag does not
        belong to a zone already seen
         %Create a new zone in the struct when encountered in tag list
         unique_zones = unique_zones + 1; %Increae the unique zone
            counter
         zone_cell_array{unique_zones} = tag_info_array(index,:);
    end
end %End of for loop

%Number of zones:
num_zones = size(zone_cell_array,2);
%Make sure that all of the user inputs matches the unique zone number.
%If it doesn't, quit program with error
if (num_zones ~= size(zone_types,1))||(num_zones ~= size(
    zone_fzt_headings,1))||(num_zones ~= size(zone_rzt_headings,1))
    disp('Zone number does not align with the specified input matrix
        sizes!!')
    return
end
```

```
%Maximum number of tags that could be in any zone (i.e undefined zones
   may
%have 4-9tags)
max_num_tags = 1; %Initialize variable
for index = 1:1:num_zones %Loop through each zone
    %Find the number of tags for the zone
    num_tags = size(zone_cell_array{1,index},1);
    if(num_tags > max_num_tags ) %If the zone has a greater number of
        tags than the highest number seen so far
      max_num_tags = num_tags; %Set the highest recorded tag count as
          the max_num_tags
    end
end %End of for loop

%First line of CSV file
first_line_string = 'zone_number,';
for index = 1:1:max_num_tags
    temp_string = ['tag',num2str(index),'_ident,tag',num2str(index),'_x
       ,tag',num2str(index),'_y,'];
    first_line_string = [first_line_string,temp_string];
end %End of for loop
first_line_string = [first_line_string,'fz_heading,rz_heading,type,'];

%Compose the body of the string array
center_string_array = {}; %Initialize array
for zone_index = 1:1:num_zones %loop across number of zones
    center_string_array{zone_index}(1,:) = [num2str(zone_index),','];
    for tag_index = 1:1:max_num_tags %loop across max number of tags
        if(tag_index <= size(zone_cell_array{1,zone_index},1)) %If the
            number of tags in the zone are greater than the tag index
            %Fill with info that we know
            temp_ident = num2str(zone_cell_array{zone_index}(tag_index
                ,2));
            temp_x = num2str(zone_cell_array{zone_index}(tag_index,3));
            temp_y = num2str(zone_cell_array{zone_index}(tag_index,4));
            temp_string = [temp_ident,',',temp_x,',',temp_y,','];
        else %Extra spaces (if max_num_tags is greater than the tags
            that are listed for that zone) are filled with NaN's
            temp_string = ['NaN,','NaN,','NaN,']; %Fill zones with only
                a few tags with NaNs till they match the length of the
                larger zone tag number
        end
        %Concatenate the string to the array
        center_string_array{zone_index} = cat(2,center_string_array{
            zone_index}(1,:),temp_string);
    end %End of tag loop
    %Add ending characters
    temp_string = [num2str(zone_fzt_headings(zone_index)),',',num2str(
        zone_rzt_headings(zone_index)),',',zone_types(zone_index),','];
    center_string_array{zone_index} = cat(2,center_string_array{
        zone_index}(1,:),temp_string);
end %End of zone loop

%Write to CSV file (Use csvwrite command if issues arise)
fid = fopen([location,output_filename],'wt'); %Open file
fprintf(fid,[first_line_string,'\n']); %Write first line.
for index = 1:1:num_zones %Fill body
    %fprintf(fid,); %Write new line, '\' is the new line character
    fprintf(fid,[center_string_array{1,index}(1,:),'\n']);
end
fclose(fid);

%% /////////////FUNCTIONS/////////////
function [full_array] = fill_out_array(tag_array,maximum_number_tags)
%   @brief  Fills out the rest of the array with the full size of tag
%     numbers with
%   @param[in]  tag_array - actual tag ID's
%   @param[in]  maximum_number_tags - Largest number of tags
   encountered in
%     any zone
%   @param[out] full_array - 1xN array where N is the maximum number of
%     tags
%   @todo
full_array = zeros(maximum_number_tags,1);
tag_array_size = length(tag_array);
    for index = 1:1:maximum_number_tags %Loop through the number of
        tags
        if(index <= tag_array_size)
            full_array(index,1) = tag_array(index);
        elseif(index > tag_array_size) %Fill the remainder of the array
             with NaN
            full_array(index,1) = NaN;
        end
    end
end %end of function
```

Listing C.6: Code to assist the user in creating the zone connection matrix that is used by the post-processing code.

```
%% Struct generation
%Need to output an NxN matrix of structs where i,j shows the relation
%between the current zone, i, to j, the zone the user is traveling from
```

```
%fzt/fz: forward zone transition
%rzt/rz: Reverse zone transition
%[zone_currently_in, zone_coming_from]

%SHOULDNT NEED THE DIAGONAL (i = j)
%Initial Condition
zone_relation_struct(1,1).fzt_heading = 90;
zone_relation_struct(1,1).xy_fz = [0,21];

%Zone 1 to 2
zone_relation_struct(1,2).fzt_heading = 90;
zone_relation_struct(1,2).xy_fz = [0,78];

%Zone 2 to 1
zone_relation_struct(2,1).fzt_heading = 270;
zone_relation_struct(2,1).xy_fz = [0,41];

%Zone 2 to 3
zone_relation_struct(2,3).fzt_heading = 90;
zone_relation_struct(2,3).xy_fz = [0,124.5];

%Zone 3 to 2
zone_relation_struct(3,2).fzt_heading = 270;
zone_relation_struct(3,2).xy_fz = [0,95.5];

%Zone 3 to 4
zone_relation_struct(3,4).fzt_heading = 180;
zone_relation_struct(3,4).xy_fz = [-12.5,148.5];

%Zone 4 to 3
zone_relation_struct(4,3).fzt_heading = 0;
zone_relation_struct(4,3).xy_fz = [-38,148.5];

%Zone 4 to 5
zone_relation_struct(4,5).fzt_heading = 108.84;
zone_relation_struct(4,5).xy_fz = [-72,206];

%Zone 5 to 4
zone_relation_struct(5,4).fzt_heading = 288.84;
zone_relation_struct(5,4).xy_fz = [-55,164];

%Zone 5 to 6
zone_relation_struct(5,6).fzt_heading = 61.73;
zone_relation_struct(5,6).xy_fz = [-84,267];

%Zone 6 to 5
zone_relation_struct(6,5).fzt_heading = 241.73;
zone_relation_struct(6,5).xy_fz = [-76,232];

%Zone 6 to 7
zone_relation_struct(6,7).fzt_heading = 61.73;
zone_relation_struct(6,7).xy_fz = [-39,318];

%Zone 7 to 6
zone_relation_struct(7,6).fzt_heading = 241.73;
zone_relation_struct(7,6).xy_fz = [-82,283];

%Zone 7 to 8
zone_relation_struct(7,8).fzt_heading = 0;
zone_relation_struct(7,8).xy_fz = [-11.5,328];

%Zone 8 to 7
zone_relation_struct(8,7).fzt_heading = 180;
zone_relation_struct(8,7).xy_fz = [-34.5,328];

%Zone 8 to 9
zone_relation_struct(8,9).fzt_heading = 0;
zone_relation_struct(8,9).xy_fz = [69,338];

%Zone 9 to 8
zone_relation_struct(9,8).fzt_heading = 180;
zone_relation_struct(9,8).xy_fz = [24.5,338];

%Zone 8 to 10
zone_relation_struct(8,10).fzt_heading = 270;
zone_relation_struct(8,10).xy_fz = [0,280];

%Zone 10 to 8
zone_relation_struct(10,8).fzt_heading = 90;
zone_relation_struct(10,8).xy_fz = [0,313.5];

%Zone 10 to 11
zone_relation_struct(10,11).fzt_heading = 270;
zone_relation_struct(10,11).xy_fz = [0,205.5];

%Zone 11 to 10
zone_relation_struct(11,10).fzt_heading = 90;
zone_relation_struct(11,10).xy_fz = [0,257.5];

%Zone 11 to 3
zone_relation_struct(11,3).fzt_heading = 270;
zone_relation_struct(11,3).xy_fz = [0,155.5];

%Zone 3 to 11
```

```
zone_relation_struct(3,11).fzt_heading = 90;
zone_relation_struct(3,11).xy_fz = [0,205.5];


%Loop through all possible zone combinations and determine if they are
%valid connections or not.
for i = 1:1:size(zone_relation_struct,1)
    for j = 1:1:size(zone_relation_struct,2)
        if isempty(zone_relation_struct(i,j).fzt_heading)
            zone_relation_struct(i,j).valid = 'n';
        else
            zone_relation_struct(i,j).valid = 'y';
        end
    end
end


%% File Output
save('zone_relation.mat','zone_relation_struct')
```

Listing C.7: Code to assist the user in generating the line map approximation.

```
%% Line generation
% Each point will be be either a starting position or end position of a
% line segment
% The array a is Nx4 where N is the number of line segments and 4 are
   the
% x1,y1,x2,y2 coordinates, respectively. The start of the line segment
   is
% coordinate: (x1,y1)  and the end of the line segment is coordinate (
   x2,y2)


%Line 1
line_map_array(1,:) = [0,0,0,400];

%Line 2
line_map_array(2,:) = [0,148.5,-80,148.5];

%Line 3
line_map_array(3,:) = [-51,148.5,-75,219.5];

%Line 4
line_map_array(4,:) = [-75,219.5,-77.5,260];

%Line 5
```

```
line_map_array(5,:) = [-77.5,260,-41,328];

%Line 6
line_map_array(6,:) = [-80,328,0,328];

%Line 7
line_map_array(7,:) = [0,338,100,338];


%% Plot the lines
figure
hold on
grid minor
set(gca,'FontSize',20)
for index = 1:1:size(line_map_array,1)
    plot(line_map_array(index,[1,3]),line_map_array(index,[2,4]),'k','
        LineWidth',2)
end
xlabel('X coordinate (ft)')
ylabel('Y coordinate (ft)')
hold off


%% File Output
save('zone_map.mat','line_map_array')
```

Listing C.8: Code to interleave the RFID data files output from the two readers. Put both the reader files in the same folder level as this code and just run this script.

```
format long
%Code that takes the two RFID data sets and interleaves them based on
    time.
%Uses a while loop to loop across the combined the file size number of
%indices. Each loop index, it determines if file 1 has an earlier time
    than
%file 2, or later, or the same. Then it assigns the earlier time and
%increments the file index that was the earlier one.
%% User Inputs
base_filepath = '';
rfid_data_1_name = 'reader0_out.txt';
rfid_data_2_name = 'reader1_out.txt';
output_rfid_data_name = 'rfid_combined_data.txt';

%% File Input
[rfid_data1] = extract_rfid_data_factory(base_filepath,rfid_data_1_name
    );
```

```
[rfid_data2] = extract_rfid_data_factory(base_filepath,rfid_data_2_name
    );


%% Interleaving
%Create indices
final_indices = length(rfid_data1.reader_time) + length(rfid_data2.
    reader_time);
data_index_1 = 1; %Index through file 1
data_index_2 = 1; %Index through file 2
index = 1;
while index <= final_indices
    temp1_time = rfid_data1.reader_time(data_index_1);
    temp2_time = rfid_data2.reader_time(data_index_2);

    if(temp2_time > temp1_time) %Time 2 is greater than time 1
        %Build up string
        temp_string = ['b',char(39),'350000000000000000000',...
                       num2str(rfid_data1.tag_ident(data_index_1)),char
                           (39),',',...
                       num2str(rfid_data1.phase(data_index_1)),',',...
                       num2str(rfid_data1.rssi(data_index_1)),',',...
                       num2str(rfid_data1.antenna(data_index_1)),','
                           ,...
                       num2str(rfid_data1.reader_time(data_index_1),'
                           %.16f')];
        output_string_line{index} = temp_string; %Pass string to output
        data_index_1 = data_index_1 + 1; %Increase data file index

        %Check to make sure hat we do not exceed the matrix size
        if(data_index_1 > length(rfid_data1.reader_time) )
            data_index_1 = length(rfid_data1.reader_time); %Fix the
                index to max
        end
        index = index + 1; %Increase while loop index

    elseif(temp2_time < temp1_time) %Time 1 is greater than time 2
        %Build up string
        temp_string = ['b',char(39),'350000000000000000000',...
                       num2str(rfid_data2.tag_ident(data_index_2)),char
                           (39),',',...
                       num2str(rfid_data2.phase(data_index_2)),',',...
                       num2str(rfid_data2.rssi(data_index_2)),',',...
                       num2str(rfid_data2.antenna(data_index_2)),','
                           ,...
                       num2str(rfid_data2.reader_time(data_index_2),'
                           %.16f')];
        output_string_line{index} = temp_string; %Pass string to output
        data_index_2 = data_index_2 + 1; %Increase data file index

        %Check to make sure that we do not exceed the matrix size
        if(data_index_2 > length(rfid_data2.reader_time))
            data_index_2 = length(rfid_data2.reader_time); %Fix the
                index to max
        end
        index = index + 1; %Increase while loop index

    else %CONDITION FOR IF BOTH TIMES ARE THE EXACT SAME
        %Enter line for the 1st Reader
        %Build up string
        temp_string = ['b',char(39),'3500000000000000000',...
                       num2str(rfid_data1.tag_ident(data_index_1)),char
                          (39),',',...
                       num2str(rfid_data1.phase(data_index_1)),',',...
                       num2str(rfid_data1.rssi(data_index_1)),',',...
                       num2str(rfid_data1.antenna(data_index_1)),','
                          ,...
                       num2str(rfid_data1.reader_time(data_index_1),'
                          %.16f')];
        output_string_line{index} = temp_string; %Pass string to output
        data_index_1 = data_index_1 + 1; %Increase data file index
        index = index + 1; %Increase while loop index

        %Enter line for the 2nd Reader
        %Build up string
        temp_string = ['b',char(39),'3500000000000000000',...
                       num2str(rfid_data2.tag_ident(data_index_2)),char
                          (39),',',...
                       num2str(rfid_data2.phase(data_index_2)),',',...
                       num2str(rfid_data2.rssi(data_index_2)),',',...
                       num2str(rfid_data2.antenna(data_index_2)),','
                          ,...
                       num2str(rfid_data2.reader_time(data_index_2),'
                          %.16f')];
        output_string_line{index} = temp_string; %Pass string to output
        data_index_2 = data_index_2 + 1; %Increase data file index
        index = index + 1; %Increase while loop index

    end %End of if else statement
    clear temp1_time temp2_time
end %End of while loop

%% File Save
fid = fopen(output_rfid_data_name,'w');
```

```
for index = 1:1:size(output_string_line,2)
    fprintf(fid,[output_string_line{index},'\n']);
end
fclose(fid);


%% Functions

function [rfid_data] = extract_rfid_data_factory(location,filename_rfid
    )
     % @brief Function extracts the data from the RFID .txt files
     % @param[in] location - Location for where the rfid data is stored
     % @param[in] filename_rfid - Filename of the rfid data
     % @param[out] rfid_data - Structure that contains all of the RFID
         data
     % @todo Create a version of this function that works with the
         custom tags

     rfid_dataset = readtable([location filename_rfid]);

     tag_ident_raw = table2array(rfid_dataset(:,1));

     %Remove the excess characters - HARDCODED BASED ON THE TAGS USED
        FOR THIS
     %EXPERIMENT (the end-5 and end-1 component)
     for index = 1:1:size(tag_ident_raw,1)
          temp = (regexprep(tag_ident_raw{index,1},'b',''));
          tag_ident(index,1) = str2double(temp(end-5:end-1));
     end

        rfid_data.tag_ident = tag_ident;
        rfid_data.reader_time = table2array(rfid_dataset(:,5)); %seconds
        rfid_data.antenna = table2array(rfid_dataset(:,4));
        rfid_data.rssi = table2array(rfid_dataset(:,3)); %dB
        rfid_data.phase = table2array(rfid_dataset(:,2)); %Degrees
end %End of function
```

Listing C.9: Function that takes in the zone information file, which was generated in the pre-processing step, and the tag ID data and outputs the indices of the RFID data that belong to each zone. This code also outputs the column header names from the zone information file so later functions can directly call from the table without needing logic on how many tags are in each zone.

```
function [rfid_zone_array,max_num_tags,var_name_x,var_name_y,
    var_name_ident] = categorize_rfid_data_ntags(zone_info_table,
    tag_identifier_data)
    % @brief Function that determines which of the RFID values were in
        which
    % zone. Can operate anywhere from 1 to n tags.
    % @param[in] zone_info_table - Tabular data from the zone
        information csv file
    % @param[in] tag_identifier_data - Array of tag identifiers
        obtained from the
    % RFID data
    % @param[out] rfid_zone_array - Array output that is the same
        length as the RFID
    % data points that determines which RFID values were in which zone
    % @param[out] max_num_tags - Single number that states what the
        highest
    %  number of tags that were attributed to a single zone were
    % @param[out] var_name_x - A list of which of the variable names
    %  corresponded directly to the tag '_x'.
    % @param[out] var_name_y - A list of which of the variable names
    %  corresponded directly to the tag '_y'.
    % @param[out] var_name_ident - A list of which of the variable
        names
    %  corresponded directly to the tag '_ident'.
    % @todo

    %Pre-allocate array size
    rfid_zone_array = zeros(length(tag_identifier_data),1);


    %Grab all of the variable names from the zone table
    table_var_names = zone_info_table.Properties.VariableNames;

    %In the zone table, search for which columns ones end in ident
    i = 1; %Init sub-index
    for index = 1:1:size(table_var_names,2)
        temp_str_search = strfind(table_var_names{1,index},'_ident');

        if(isempty(temp_str_search))
            %Do Nothing
        else
            %Found a column (or variable name) that ends in ident
            index_record(i) = index; %Record index
            var_name_record{i} = table_var_names{1,index};
            i = i + 1; %Advance through the columns
        end
    end
```

```
%Output ident var names
var_name_ident = var_name_record;


%In the zone table, search for which ones end in _x
i = 1; %Reset iterator
for index = 1:1:size(table_var_names,2)
    temp_str_search = strfind(table_var_names{1,index},'_x');

    if(isempty(temp_str_search))
        %Do Nothing
    else
        %Found a column (or variable name) that ends in ident
        var_name_x{i} = table_var_names{1,index};
        i = i + 1;
    end
end

%In the zone table, search for which ones end in _y
i = 1; %Reset iterator
for index = 1:1:size(table_var_names,2)
    temp_str_search = strfind(table_var_names{1,index},'_y');

    if(isempty(temp_str_search))
        %Do Nothing
    else
        %Found a column (or variable name) that ends in ident
        var_name_y{i} = table_var_names{1,index};
        i = i + 1;
    end
end


%loop through RFID values
for index = 1:1:length(tag_identifier_data)
    temp = tag_identifier_data(index);

    %loop through zones -if rfid zone identifier matches any of the
    %three conditions then set the rfid zone array to the zone
       number
    for zone_index = 1:1:size(zone_info_table,1)

        %Check all of the variables names, found in the search
           string
        %for if the data matches up with ident
```

```
            possible_tag_array = table2array(zone_info_table(zone_index
                ,var_name_record));

            %Subtract temp tag number from the possible tag array
            search_array = possible_tag_array - temp;
            %Search for a zero (an array that contains a zero means
                exact
            %match)
            cond_array = (search_array == 0);

            %Conditional so that if any values in the array are 1 (aka
            %match) then the if condition executes successfully.
            if(any(cond_array))
                rfid_zone_array(index,1) = zone_info_table.zone_number(
                    zone_index);
            end
        end
    end
    max_num_tags = size(possible_tag_array,2); %This only works if NaNs
        are
                                                %used to fill the array
                                                    with
                                                %zones with fewer number
                                                    of
                                                %tags. Use the
                                                %file_generator script!!
end %End of function
```

Listing C.10: Function that takes in the IMU data, the RFID time data, and the RFID zone ID array and outputs the indices in the IMU data that demarcate a transition to a different zone. This function also outputs the times at which this occurs as well as the indices in the RFID data that a transition occurred.

```
function [imu_zone_indices, imu_zone_times, zone_progression,
    rfid_zone_tr_index] = categorize_imu_data(imu_t, rfid_t,
    rfid_zone_array)
    % @brief Core component of the ZCA. Takes in the RFID zone data and
        the
    %  IMU data and organizes the IMU data into zones
    % @param[in] imu_t - Time array of the IMU data
    % @param[in] rfid_t - Time array of the RFID reader data
    % @param[in] rfid_zone_array - Array of which zone the RFID data
        corresponds to
    % @param[out] imu_zone_indices - Zone transitions in IMU indices
    % @param[out] imu_zone_times - Zone transitions in IMU time
    % @param[out] rfid_zone_tr_index - RFID Zone transition index array
    % @todo


    %Find times where the zone transitions
    last_zone = 1; %Initialize what the last zone was
    i = 1; %Initialize zone transition counter
    zone_transition_indices = []; %Initialize empty zone transition
        index matrix
    zone_progression(1) = rfid_zone_array(1); %STARTS AT WHATEVER THE 1
        st RECORDED ZONE WAS
    for index = 1:1:size(rfid_zone_array,1)
        if (last_zone ~= rfid_zone_array(index)) %If you encountered a
            new zone
            zone_transition_indices(i) = index;  %Record index this
                occurred at
            i = i + 1; %Increment zone transition index
            last_zone = rfid_zone_array(index); %Update the last zone
            zone_progression(i) = last_zone; %Record the zone that we
                are at now
        end
    end
    zone_transition_times = rfid_t(zone_transition_indices); %Find the
        times that we transitioned to a different zone

    %Output for when the RFID Data Transitions (indices)
    rfid_zone_tr_index = [1,zone_transition_indices,(size(
        rfid_zone_array,1)+1)];
    %Added 1 to the maximum as the code in the main body will subtract
        it
    %out. If an error occurs then the main body logic must have changed
        .

    %Find where these times correspond to indices in the IMU data
    imu_zone_indices = [];

    for index = 1:1:size(zone_transition_times,1)
        %Only need to do minimum because you are using time and time
            shouldn't
        %have local minimum that are not the global minimum
        [~,idx] = min(abs(imu_t-zone_transition_times(index)));
        imu_zone_indices(index) = idx;
    end
    %Find the indices in the IMU that demarcate
    imu_zone_times = imu_t(imu_zone_indices);
end %End of function
```

Listing C.11: Function and helper functions used to calculate the RFID 2D position. Uses the gauss-newton method to approximate the user's 2D position using code from [40]. To plot the tag readings, code was used from [41]

```
function [output_2d_array] = rfid_2d_pos_prototype_fxn_mk3(input_struct
    ,zone_struct,time_window_size,window_overlap,gauss_iters,lut_vals,
    plot_intermed_results_yn)
% @brief Uses the RFID data, look up table, tag information to
    calculate the user's 2D
%   position. Moves a time window across the data, detects if at least
%   three tags are read, and then invokes the gauss-newton method to
%   estimate the most likely 2D position for each window. Save the time
%   stamp for the center of the window along with the 2D position.
% @param[in] input_struct - size 1x1 struct of all of the data and
%   information for the zone that is being processed.
% @param[in] zone_struct - size 1x1 struct containing all of the
%   information about RFID tags and zone information
% @param[in] time_window_size - Window width in seconds that is stepped
%   through the data. Only values that fall within this time window
    will be
%   used to calculate distance (provided there are at least three tags)
% @param[in] window_overlap - Time in seconds that the windows overlap
    each
%   other. CANNOT BE GREATER THAN OR EQUAL TO time_window_size
% @param[in] gauss_iters - Number of iterations for the gauss-newton
    method
% @param[in] lut_vals - Look up table values. Input as a curve fit
    object
%   (cfit) which contains the 3rd order polynomial coefficients and
%   statistical information about how well they fit the look up table
%   measurements.
% @param[in] plot_intermed_results_yn - Switch . Put y to plot the RSSi
    vs
%   time and distance vs time plots,
% @param[out] output_2d_array - %3 dimensional data [N x 3]
%   (x_coord, y_coord, time) where N is the number of 2D positions


%% USER Specified Values & LUTs
%time_window_size and window_overlap cannot be equal nor can window
    overlap
%be greater

%Gaussian Newton iterations
iters = gauss_iters;
```

```
%Check window sizing
if time_window_size <= window_overlap
    error('window_overlap_is_greater_than_or_equal_to_time_window_size'
       )
end

%Only need to process
temp_data_struct = input_struct;


%% CALCULATIONS
%Pull out look up table information - Needs to have RSSi as the
   dependent
%value
coeff_array = coeffvalues(lut_vals);
%Equation for poly2 curve fit. Added manual shift.
lut_eqn = @(x) (coeff_array(1)*(x.^3))+(coeff_array(2)*(x.^2)) + (
   coeff_array(3)*x) + (coeff_array(4));


%Convert all RSSI values to distance using LUT for each antenna, each
   tag
for tag_index = 1:1:size(temp_data_struct.sorted_rssi,1) %loop through
   tags
    for ant_index = 1:1:size(temp_data_struct.sorted_rssi,2) %loop
       through antennas
        temp_data = temp_data_struct.sorted_rssi{tag_index,ant_index};
        temp_data_struct.converted_dist{tag_index,ant_index} = lut_eqn(
           temp_data);

        %DISTANCE RAILS - CAN'T GO NEGATIVE DISTANCE AND CANT GO ABOVE
           12
        %ft
        neg_indices = lut_eqn(temp_data) < 0;
        temp_data_struct.converted_dist{tag_index,ant_index}(
           neg_indices) = 1; %Set negative distances to 1 ft
        large_indices = lut_eqn(temp_data) > 12; %Set distances that go
            beyond the look up table to 12 ft (the max of the look up
           table)
        temp_data_struct.converted_dist{tag_index,ant_index}(
           large_indices) = 12;

        clear temp_data
    end
end
```

```
%If desired, plot the distance versus time
if (plot_intermed_results_yn == 'y') || (plot_intermed_results_yn == 'Y
    ')
     plot_rfid_dist_results(temp_data_struct,2);
end

%Determine the number of time windows in the RFID readings
complete_time_val_array = temp_data_struct.complete_rfid_time_data;
num_windows = floor((complete_time_val_array(end) -
    complete_time_val_array(1))/(time_window_size-window_overlap));
rfid_start_time = complete_time_val_array(1) + (time_window_size/2);

for index = 1:1:num_windows
     %Window Parameters
     window_center_time(index) = rfid_start_time + ((index-1) *
        time_window_size) - ((index-1) * window_overlap);
     window_start_time(index) = (window_center_time(index) -
        time_window_size/2);
     window_stop_time(index) = (window_center_time(index) +
        time_window_size/2);

     %Search through all antennas and tags - build array of rfid
        readings
     %that fall in this window
     for ant_index = 1:1:size(temp_data_struct.sorted_rssi,2)
          index_in_window = 1; %Need to build a dimension of the struct
             array for each antenna (could have separate LUTs for each
             antenna)
          for tag_index = 1:1:size(temp_data_struct.sorted_rssi,1)
               temp_time = temp_data_struct.sorted_time{tag_index,
                  ant_index};

               %Find the indices of the time values that fall within the
                  time
               %window
               index_array = (temp_time >= window_start_time(index)) & (
                  temp_time <= window_stop_time(index));
               %Time array for the recorded time values inside of the
                  array
               w_time_array = temp_time(index_array);
               %zone_num = temp_data_struct.zone;
               if (~isempty(w_time_array))
                   %Average the time stamps in window
                   temp_window_struct(index,ant_index).time(index_in_window
                       ) = mean(w_time_array);
                   %All idents should be the same (because we are looping
                       through them)
```

```
                %so, just grab the first val
                temp_window_struct(index,ant_index).id(index_in_window)
                   = temp_data_struct.sorted_ident{tag_index,ant_index
                   }(1);
                %Grab the minimum distances in window - eliminates large
                     distances from multipath
                temp_window_struct(index,ant_index).dist(index_in_window
                   ) = min(temp_data_struct.converted_dist{tag_index,
                   ant_index}(index_array));
                %Put the x,y coordinate of the tag in the array (x,y)
                temp_window_struct(index,ant_index).tag_xy(
                   index_in_window,1) = table2array(zone_struct.xy(
                   tag_index,1));
                temp_window_struct(index,ant_index).tag_xy(
                   index_in_window,2) = table2array(zone_struct.xy(
                   tag_index,2));

                %Increment the window index
                index_in_window = index_in_window + 1;
            end

            clear w_time_array index_array temp_time
        end %End of tag indexing loop
    end %End of antenna indexing loop
end %End of Window indexing loop



%Calculate the 2D RFID position
%If there are greater than 3 unique tags read between two
%antennas then proceed to calculate the 2d estimate\
index = 1;
for window_index = 1:1:size(temp_window_struct,1)

     temp_prep_array = prep_estimate_calc_mk2(temp_window_struct(
        window_index,:));

     %Check to see if there are at least three unique tags
     if size(temp_prep_array,1) >=3
        %Use the median of windowand tack on the window time to the end
            of
        %the array
        temp_prep_array(:,5) =  window_center_time(1,window_index);

        prepped_rfid_array_for_estimate{index} = temp_prep_array;

        %Find combinations
        num_uniq_tags   = size(temp_prep_array,1);
        base_combo_idx_array = 1:1:num_uniq_tags;
        num_combins = nchoosek(num_uniq_tags,3); %Find all combinations
        combo_idx_array = nchoosek(base_combo_idx_array,3);

        %For each combination calculate 2D position - DOES NOT
        %CONTRIBUTE TOFINAL OUTPUT, LEFT HERE FOR FUTURE TESTING
        for combo_idx = 1:1:num_combins %(also could be size(
            combo_idx_array,1))
             rfid_2d_estimate_combo(combo_idx,:) =
                 rfid_localization_fxn_spec(temp_prep_array(
                 combo_idx_array(combo_idx,:) ,:),iters);
        end

        %Calculate 2D position using all tags - THIS IS THE ONE THAT IS
        %OUTPUT
        rfid_2d_estimate_all = rfid_localization_fxn_spec(
            temp_prep_array,iters);

        %Create struct that has all of the 2D results
        rfid_struct_for_final_estimate(index).window_idx = window_index
            ;
        rfid_struct_for_final_estimate(index).rfid_combo =
            rfid_2d_estimate_combo;
        rfid_struct_for_final_estimate(index).rfid_all =
            rfid_2d_estimate_all;
        rfid_struct_for_final_estimate(index).time_stamp =
            temp_prep_array(1,end);
        rfid_struct_for_final_estimate(index).avg_combo = mean(
            rfid_2d_estimate_combo,1,'omitnan');

        index = index + 1;
    end %End of if statement
    clear temp_prep_array_array combo_idx_array rfid_2d_estimate_all
        rfid_2d_estimate_combo
end

%OUTPUT 2D POSITION
i = 1;
for index = 1:1:length(rfid_struct_for_final_estimate)
    if ~isnan(rfid_struct_for_final_estimate(index).rfid_all) %filter
        out nans
         %[x_coord,y_coord,time]
        output_2d_array(i,:) = [rfid_struct_for_final_estimate(index).
            rfid_all,rfid_struct_for_final_estimate(index).time_stamp];
        i = i + 1;
    end
```

```
end

%Plot the result of the 2D position
figure
hold on
set(gca,'FontSize',24)
title(['RFID␣2D␣Zone:␣', num2str(temp_data_struct.zone)])
for i = 1:1:length(rfid_struct_for_final_estimate)
    temp_x(i) = rfid_struct_for_final_estimate(i).rfid_all(1,1);
    temp_y(i) = rfid_struct_for_final_estimate(i).rfid_all(1,2);
    temp_time(i) = rfid_struct_for_final_estimate(i).time_stamp;
end
plot(temp_x,temp_y,'-o','LineWidth',2)
plot(temp_x(1,1),temp_y(1,1),'*','MarkerSize',18)

tag_xy_array = table2array(zone_struct.xy);
plot(tag_xy_array(:,1),tag_xy_array(:,2),'dr','LineWidth',3)
grid minor
hold off

% FUNCTION OUTPUT
%3 dimensional data [N x 3] (x_coord, y_coord, time)
output_2d_array = [temp_x',temp_y',temp_time'];


%% FUNCTIONS
    function plot_rfid_dist_results(input_struct,num_antennas)
    % @brief - A void function that plots the rssi values of the rfid
        data.
    % @param[in] input_struct - Fully developed data struct (all zone
    %   transitions)
    % @param[in] num_antennas - Number of antennas used by the
        acquistion
    %   system
    % @todo

    %Each antenna will have its own linestyle
    %Each tag will have its own color
    for index = 1:1:num_antennas
        [~,temp_linestyle,~]= set_zone_unique_color(index);
        ant_linestyle_array(index) = string(temp_linestyle);
        clear temp_linestyle
    end

    %Loop across zones
    for index = 1:1:size(input_struct,2)
        %Give each tag a unique color
        for color_index = 1:1:size(input_struct(index).sorted_rssi,1)
            [temp_color,~,~]= set_zone_unique_color(color_index);
            tag_color_array(color_index,:) = temp_color;
            clear temp_color
        end

        %Generate new figure for each zone transition
        figure
        hold on
        set(gca,'FontSize',24)
        grid minor
        xlabel('Time(s)')
        ylabel('distance (ft)')
        title(['Zone Transition: ',num2str(index),', Zone #: ',num2str(
           input_struct(index).zone)])
        legend_entry_array = []; %Clear out the legend entry array
        for tag_index =  1:1:size(input_struct(index).sorted_rssi,1) %
           Loop across tags
            for ant_index = 1:1:size(input_struct(index).sorted_rssi,2)
                %Loop across antennas
                temp_x = input_struct(index).sorted_time{tag_index,
                   ant_index};
                temp_y = input_struct(index).converted_dist{tag_index,
                   ant_index};
    %             temp_color = input_struct(index).color;
                temp_linestyle = ant_linestyle_array(ant_index);
    %             temp_marker = struct(index).marker;
                if (~isempty(temp_x) && ~isempty(temp_y))
                    stem(temp_x,temp_y,'Color',tag_color_array(
                       tag_index,:),'LineStyle',temp_linestyle,'
                       LineWidth',2)
                    temp_tag_ident = input_struct(index).sorted_ident{
                       tag_index,ant_index}(1,1);
                    temp_legend_entry = string(['Tag: ',num2str(
                       temp_tag_ident),', Ant: ',num2str(ant_index)]);
                    legend_entry_array = cat(2,legend_entry_array,
                       temp_legend_entry);
                    clear  temp_x temp_y temp_tag_ident
                       temp_legend_entry
                end%End of if else
            end
        end
        legend(legend_entry_array)
        hold off
    end %End of zone for loop
    end %end of function
```

```
function [color,linestyle,marker] = set_zone_unique_color(seed)
  % @brief - Function that returns an array of unique plotting
      settings
  % @param[in] seed - the random seed
  % @param[out] color - Plot line color
  % @param[out] linestyle - Plot linestyle for zone
  % @param[out] marker - Plot line marker
  % @todo

  %Code taken from Alec Weiss's (Sheekaboom) code:
  %https://github.com/Sheekaboom/WeissTools/tree/master/WeissTools/
     MATLAB

  %Color list
   colors = ([ 0   ,50  ,135; 200,20  ,30 ; 0   ,120,0   ; 120,90  ,0
      ;...
             150,50  ,165; 10  ,145,120; 0   ,0   ,0   ; 220,150,20  ;...
             200,150,200; 255,255,0   ; 255,0   ,128; 150,150,0   ;...
             0   ,255,255; 0   ,255,0   ; 255,80   ,255]/255);

  %LineStyle
  linestyles = '-|--|:|-.';
  markers    = '.|+|o|*|x|s|d|^|v|>|<|p|h';
  ls_cell = repmat(strsplit(linestyles,'|'),1,4);
  mk_cell = strsplit(markers,'|');

  %Output
  color = colors(mod(seed-1,length(colors))+1,:);
  linestyle = ls_cell{mod(seed-1,length(ls_cell))+1};
  marker = mk_cell{mod(seed-1,length(mk_cell))+1};
end %End of function


function [temp_array] = prep_estimate_calc_mk2(input_struct)
% @brief - Array that takes in the RFID data corresponding to a
    specific
%  window and preps the output array that will be used by the 2d
    rfid
%   positioning function
% @param[in] input_struct - struct that contains all of the
    variables for
%        every window. Struct should be a 1xN array where N is the
    number of
%        antnnas
% @param[out] output_array -  a K x 4 array where is the number of
    unique
%   tags and the 4 corresponds to (tag ID, radial distance, tag x,
    tag y)
     temp_array = [0,0,0,0];
     k = 1;
     for i = 1:1:size(input_struct,2) %Loop across antenna number
          for j = 1:1:size(input_struct(1,i).id,2) %Loop across tag
              ids
               check_unique_tag_array = (temp_array(:,1) ==
                   input_struct(1,i).id(1,j));
               if isempty(find(check_unique_tag_array == 1)) %
                   Condition for no duplicates
                    %Create entry in the output array
                    temp_array(k,1) = input_struct(1,i).id(1,j); %Tag
                        ID
                    temp_array(k,2) = input_struct(1,i).dist(1,j); %
                        Radial Distance
                    temp_array(k,3) = input_struct(1,i).tag_xy(j,1); %X
                    temp_array(k,4) = input_struct(1,i).tag_xy(j,2); %Y
                    k = k + 1;
               elseif sum(check_unique_tag_array) >= 1 %If duplicates
                   occur, get the minimum distance
                    idx_of_dupl_id = find(check_unique_tag_array == 1);
                    %Grab the minimum values
                    temp_array(idx_of_dupl_id,2) = min([temp_array(
                        idx_of_dupl_id,2) , input_struct(1,i).dist(1,j)
                        ]);

               end %End of if statement
               clear check_unique_tag_array
          end %end of tag id for loop
     end %end of antenna number for loop


%       %Get rid of duplicates between antennas.
%       for k = 1:1:size(temp_array,1)
%
%       end
     output_array = temp_array;
end %End of function



function [rfid_2d_output] = rfid_localization_fxn_spec(input_array,
    iterations)
```

```
%   @brief  The core function of the RFID localization that takes
    in the
%       tag coordinates and the distances to each tag, and uses the
%       Gaussian Newton method to find the x y coordinate. The spec
%       addition means that it is specific to this program.
%   @param[in] tag_coord - 2D array of tag coordinates, size: Nx2
%   @param[in] dist_array - 2D array of distances to the tag: NxM
%   @param[in] iterations - Number of iterations the Gaussian
    newton will
%       loop over
%   @param[out] rfid_2d_output - 2d array of positions, obtained
    from RFID
%       tags. Mx2 (M should usually be 1)
%   @todo
%   @reference - Code taken from:
%https://www.mathworks.com/matlabcentral/fileexchange/33792-single-
    multiple-target-localization

%Assignments: Parsed into from the prepped tag information array
%input_array : (tag_id, dist_to_tag, tag_x_coord, tag_y_coord, time
    )
 tag_coord = input_array(:,(3:4));
 dist_array = input_array(:,2);

N = size(dist_array,1); %Number of unique tags
M = size(dist_array,2); %Number of distance values (usually 1)


%Find the min and max size of the zone
 zone_min_size_x = min(tag_coord(:,1));
 zone_min_size_y = min(tag_coord(:,2));
 zone_max_size_x = max(tag_coord(:,1));
 zone_max_size_y = max(tag_coord(:,2));

 zone_size_x = zone_max_size_x - zone_min_size_x;
 zone_size_y = zone_max_size_y - zone_min_size_y;

%Form the initial guess array
 rfid_2d_output(:,1) = (zone_size_x*rand(M,1))+zone_min_size_x;
 rfid_2d_output(:,2) = (zone_size_y*rand(M,1))+zone_min_size_y;


 for m = 1 : M %Loop through number of distances that need to be
    solved for
     for i = 1 : iterations %Loop through
         % computing the esimated distances
```

```
            distanceEst    = sqrt(sum( (tag_coord - repmat(
                rfid_2d_output(m,:) ,N,1)).^2 , 2));
            % computing the derivatives
                % d0 = sqrt( (x-x0)^2 + (y-y0)^2 )
                % derivatives -> d(d0)/dx = (x-x0)/d0
                % derivatives -> d(d0)/dy = (y-y0)/d0
            distanceDrv    = [(rfid_2d_output(m,1)-tag_coord(:,1))./
                distanceEst ... % x-coordinate
                              (rfid_2d_output(m,2)-tag_coord(:,2))./
                                  distanceEst];   % y-coordinate
            % delta
            %delta = - (distanceDrv.'*distanceDrv)^-1*distanceDrv.' * (
                distanceEst - distanceNoisy(:,m));
            delta = - ((distanceDrv.'*distanceDrv)^-1)*distanceDrv.' *
                (distanceEst - dist_array(:,m));
            % Updating the estimation
            rfid_2d_output(m,:) = rfid_2d_output(m,:) + delta.';
        end
    end

    end %End of function

%% END OF FUNCTION
end %END OF FUNCTION
```

Listing C.12: Code that takes in the accelerometer data from the IMU and finds the indices in the data where the user stepped. Code was mostly written by Joseph Diener.

```
function [peak_index] = calculate_step_count_struct(input_struct,dt)
        % @brief Determines which indices of the imu data the steps
            occurred
        %  and what the magnitudes of the peaks were at these times
        % @param[in] input_struct - size 1x1 struct of all of the
        % @param[in] dt - Sample time in seconds
        % @param[out] peak_index - Index of the IMU data where the user
            stepped
        % @recognition - Code was written by Joseph Diener

        %Input data from struct
        ax = input_struct.accel_x;
        ay = input_struct.accel_y;
        az = input_struct.accel_z;
        timearb = 1 : length(ax);

        %Calculate the magnitude of acceleration
        mag_accel = sqrt((ax.^2)+(ay.^2)+(az.^2));
        %Accelerometer FFT
        accel_mag_freq = fftshift(fft(mag_accel)); %Magnitude of
           Acceleration in Frequency Domain
        freq_values = (timearb - length(timearb)/2).*dt;
        freq_absolutewindower = 2.5; %%%We window to +/- this in Hz

        %Frequency domain windowing
        [~, window_minindex] = min(abs(freq_values +
           freq_absolutewindower));
        [~, window_maxindex] = min(abs(freq_values -
           freq_absolutewindower));

        accel_mag_freq(1:window_minindex - 1) = 0;
        accel_mag_freq(window_maxindex+1 : end) = 0;

        accel_mag_freqsmoothed = ifft(ifftshift(accel_mag_freq), '
           symmetric');

        %Thresh-hold value?
        peak_acceldata = max(accel_mag_freqsmoothed);
        thresh = 0.8; %80% G of peak.

        %Peak finding
        [peak_index, peak_mag] = peakfinder(abs(accel_mag_freqsmoothed)
           , [], thresh);

end %End of function
```

Listing C.13: Code taken from [44] that finds the peaks of a signal.

```
function varargout = peakfinder(x0, sel, thresh, extrema,
   includeEndpoints, interpolate)

%%%SOURCE -
%%% https://www.mathworks.com/matlabcentral/fileexchange/25500-
   peakfinder-x0-sel-thresh-extrema-includeendpoints-interpolate
%%%

%PEAKFINDER Noise tolerant fast peak finding algorithm
%   INPUTS:
%       x0 - A real vector from the maxima will be found (required)
%       sel - The amount above surrounding data for a peak to be,
%           identified (default = (max(x0)-min(x0))/4). Larger values
   mean
%           the algorithm is more selective in finding peaks.
%       thresh - A threshold value which peaks must be larger than to
   be
%           maxima or smaller than to be minima.
%       extrema - 1 if maxima are desired, -1 if minima are desired
%           (default = maxima, 1)
%       includeEndpoints - If true the endpoints will be included as
%           possible extrema otherwise they will not be included
%           (default = true)
%       interpolate - If true quadratic interpolation will be performed
%           around each extrema to estimate the magnitude and the
%           position of the peak in terms of fractional indicies. Note
   that
%           unlike the rest of this function interpolation assumes the
%           input is equally spaced. To recover the x_values of the
   input
%           rather than the fractional indicies you can do:
%           peakX = x0 + (peakLoc - 1) * dx
%           where x0 is the first x value and dx is the spacing of the
%           vector. Output peakMag to recover interpolated magnitudes.
%           See example 2 for more information.
%           (default = false)
%
%   OUTPUTS:
%       peakLoc - The indicies of the identified peaks in x0
%       peakMag - The magnitude of the identified peaks
%
%   [peakLoc] = peakfinder(x0) returns the indicies of local maxima
   that
%       are at least 1/4 the range of the data above surrounding data.
%
%   [peakLoc] = peakfinder(x0,sel) returns the indicies of local maxima
%       that are at least sel above surrounding data.
%
%   [peakLoc] = peakfinder(x0,sel,thresh) returns the indicies of local
%       maxima that are at least sel above surrounding data and larger
%       (smaller) than thresh if you are finding maxima (minima).
%
%   [peakLoc] = peakfinder(x0,sel,thresh,extrema) returns the maxima of
    the
%       data if extrema > 0 and the minima of the data if extrema < 0
%
%   [peakLoc] = peakfinder(x0,sel,thresh,extrema, includeEndpoints)
%       returns the endpoints as possible extrema if includeEndpoints
   is
%       considered true in a boolean sense
%
%   [peakLoc, peakMag] = peakfinder(x0,sel,thresh,extrema,interpolate)
%        returns the results of results of quadratic interpolate around
    each
%        extrema if interpolate is considered to be true in a boolean
    sense
%
%   [peakLoc, peakMag] = peakfinder(x0,...) returns the indicies of the
%        local maxima as well as the magnitudes of those maxima
%
%   If called with no output the identified maxima will be plotted
    along
%        with the input data.
%
%   Note: If repeated values are found the first is identified as the
    peak
%
% Example 1:
% t = 0:.0001:10;
% x = 12*sin(10*2*pi*t)-3*sin(.1*2*pi*t)+randn(1,numel(t));
% x(1250:1255) = max(x);
% peakfinder(x)
%
% Example 2:
% ds = 100;  % Downsample factor
% dt = .001; % Time step
% ds_dt = ds*dt; % Time delta after downsampling
% t0 = 1;
% t = t0:dt:5 + t0;
% x = 0.2-sin(0.01*2*pi*t)+3*cos(7/13*2*pi*t+.1)-2*cos((1+pi/10)*2*pi*t
    +0.2)-0.2*t;
% x(end) = min(x);
% x_ds = x(1:ds:end); % Downsample to test interpolation
% [minLoc, minMag] = peakfinder(x_ds, .8, 0, -1, false, true);
% minT = t0 + (minLoc - 1) * ds_dt; % Take into account 1 based
    indexing
% p = plot(t,x,'-',t(1:ds:end),x_ds,'o',minT,minMag,'rv');
% set(p(2:end), 'linewidth', 2); % Show the markers more clearly
% legend('Actual Data', 'Input Data', 'Estimated Peaks');
% Copyright Nathanael C. Yoder 2015 (nyoder@gmail.com)
% Perform error checking and set defaults if not passed in
narginchk(1, 6);
nargoutchk(0, 2);
s = size(x0);
flipData =  s(1) < s(2);
len0 = numel(x0);
if len0 ~= s(1) && len0 ~= s(2)
    error('PEAKFINDER:Input','The input data must be a vector')
elseif isempty(x0)
```

```
    varargout = {[],[]};
    return;
end
if ~isreal(x0)
    warning('PEAKFINDER:NotReal','Absolute value of data will be used')
    x0 = abs(x0);
end
if nargin < 2 || isempty(sel)
    sel = (max(x0)-min(x0))/4;
elseif ~isnumeric(sel) || ~isreal(sel)
    sel = (max(x0)-min(x0))/4;
    warning('PEAKFINDER:InvalidSel',...
        'The selectivity must be a real scalar.  A selectivity of %.4g 
           will be used',sel)
elseif numel(sel) > 1
    warning('PEAKFINDER:InvalidSel',...
        'The selectivity must be a scalar.  The first selectivity value
           in the vector will be used.')
    sel = sel(1);
end
if nargin < 3 || isempty(thresh)
    thresh = [];
elseif ~isnumeric(thresh) || ~isreal(thresh)
    thresh = [];
    warning('PEAKFINDER:InvalidThreshold',...
        'The threshold must be a real scalar. No threshold will be used
           .')
elseif numel(thresh) > 1
    thresh = thresh(1);
    warning('PEAKFINDER:InvalidThreshold',...
        'The threshold must be a scalar.  The first threshold value in 
           the vector will be used.')
end
if nargin < 4 || isempty(extrema)
    extrema = 1;
else
    extrema = sign(extrema(1)); % Should only be 1 or -1 but make sure
    if extrema == 0
        error('PEAKFINDER:ZeroMaxima','Either 1 (for maxima) or -1 (for
           minima) must be input for extrema');
    end
end
if nargin < 5 || isempty(includeEndpoints)
    includeEndpoints = true;
end
if nargin < 6 || isempty(interpolate)
    interpolate = false;
```

```
end
x0 = extrema*x0(:); % Make it so we are finding maxima regardless
thresh = thresh*extrema; % Adjust threshold according to extrema.
dx0 = diff(x0); % Find derivative
dx0(dx0 == 0) = -eps; % This is so we find the first of repeated values
ind = find(dx0(1:end-1).*dx0(2:end) < 0)+1; % Find where the derivative
    changes sign
% Include endpoints in potential peaks and valleys as desired
if includeEndpoints
    x = [x0(1);x0(ind);x0(end)];
    ind = [1;ind;len0];
    minMag = min(x);
    leftMin = minMag;
else
    x = x0(ind);
    minMag = min(x);
    leftMin = min(x(1), x0(1));
end
% x only has the peaks, valleys, and possibly endpoints
len = numel(x);
if len > 2 % Function with peaks and valleys
    % Set initial parameters for loop
    tempMag = minMag;
    foundPeak = false;
    if includeEndpoints
        % Deal with first point a little differently since tacked it on
        % Calculate the sign of the derivative since we tacked the
            first
        %  point on it does not neccessarily alternate like the rest.
        signDx = sign(diff(x(1:3)));
        if signDx(1) <= 0 % The first point is larger or equal to the
            second
            if signDx(1) == signDx(2) % Want alternating signs
                x(2) = [];
                ind(2) = [];
                len = len-1;
            end
        else % First point is smaller than the second
            if signDx(1) == signDx(2) % Want alternating signs
                x(1) = [];
                ind(1) = [];
                len = len-1;
            end
        end
    end
    % Skip the first point if it is smaller so we always start on a
    %   maxima
```

```
if x(1) >= x(2)
    ii = 0;
else
    ii = 1;
end
% Preallocate max number of maxima
maxPeaks = ceil(len/2);
peakLoc = zeros(maxPeaks,1);
peakMag = zeros(maxPeaks,1);
cInd = 1;
% Loop through extrema which should be peaks and then valleys
while ii < len
    ii = ii+1; % This is a peak
    % Reset peak finding if we had a peak and the next peak is
        bigger
    %   than the last or the left min was small enough to reset.
    if foundPeak
        tempMag = minMag;
        foundPeak = false;
    end
    % Found new peak that was lager than temp mag and selectivity
        larger
    %   than the minimum to its left.
    if x(ii) > tempMag && x(ii) > leftMin + sel
        tempLoc = ii;
        tempMag = x(ii);
    end
    % Make sure we don't iterate past the length of our vector
    if ii == len
        break; % We assign the last point differently out of the
            loop
    end
    ii = ii+1; % Move onto the valley
    % Come down at least sel from peak
    if ~foundPeak && tempMag > sel + x(ii)
        foundPeak = true; % We have found a peak
        leftMin = x(ii);
        peakLoc(cInd) = tempLoc; % Add peak to index
        peakMag(cInd) = tempMag;
        cInd = cInd+1;
    elseif x(ii) < leftMin % New left minima
        leftMin = x(ii);
    end
end
% Check end point
if includeEndpoints
    if x(end) > tempMag && x(end) > leftMin + sel
```

```
            peakLoc(cInd) = len;
            peakMag(cInd) = x(end);
            cInd = cInd + 1;
        elseif ~foundPeak && tempMag > minMag % Check if we still need
           to add the last point
            peakLoc(cInd) = tempLoc;
            peakMag(cInd) = tempMag;
            cInd = cInd + 1;
        end
    elseif ~foundPeak
        if x(end) > tempMag && x(end) > leftMin + sel
            peakLoc(cInd) = len;
            peakMag(cInd) = x(end);
            cInd = cInd + 1;
        elseif tempMag > min(x0(end), x(end)) + sel
            peakLoc(cInd) = tempLoc;
            peakMag(cInd) = tempMag;
            cInd = cInd + 1;
        end
    end
    % Create output
    if cInd > 1
        peakInds = ind(peakLoc(1:cInd-1));
        peakMags = peakMag(1:cInd-1);
    else
        peakInds = [];
        peakMags = [];
    end
else % This is a monotone function where an endpoint is the only peak
    [peakMags,xInd] = max(x);
    if includeEndpoints && peakMags > minMag + sel
        peakInds = ind(xInd);
    else
        peakMags = [];
        peakInds = [];
    end
end
% Apply threshold value.  Since always finding maxima it will always be
%   larger than the thresh.
if ~isempty(thresh)
    m = peakMags>thresh;
    peakInds = peakInds(m);
    peakMags = peakMags(m);
end
if interpolate && ~isempty(peakMags)
    middleMask = (peakInds > 1) & (peakInds < len0);
    noEnds = peakInds(middleMask);
    magDiff = x0(noEnds + 1) - x0(noEnds - 1);
    magSum = x0(noEnds - 1) + x0(noEnds + 1)  - 2 * x0(noEnds);
    magRatio = magDiff ./ magSum;
    peakInds(middleMask) = peakInds(middleMask) - magRatio/2;
    peakMags(middleMask) = peakMags(middleMask) - magRatio .* magDiff
        /8;
end
% Rotate data if needed
if flipData
    peakMags = peakMags.';
    peakInds = peakInds.';
end
% Change sign of data if was finding minima
if extrema < 0
    peakMags = -peakMags;
    x0 = -x0;
end
% Plot if no output desired
if nargout == 0
    if isempty(peakInds)
        disp('No significant peaks found')
    else
        figure;
        plot(1:len0,x0,'.-',peakInds,peakMags,'ro','linewidth',2);
    end
else
    varargout = {peakInds,peakMags};
end
```

Listing C.14: Code used to calculate heading from the IMU's gyroscope data.

```
function [output] = calculate_gyro_angle_struct_cut_down(input_struct,
  direc,gyro_gain_mag,flip_yn)
    % @brief Function that integrates the gyroscope data to get the
        heading
    % @param[in] imu_data - Nx1 angular velocity values that will be
        integrated
    % @param[in] direc - Sets which axis is the central axis of
        rotation
    % @param[in] gyro_gain_mag - Coefficient to multiply the gyro data
        by
    % @param[in] flip_yn - Allows for rotating the heading 180 deg if
    %  IMU was flipped
    % @param[out] output - Nx1 data set of angular headings in degrees
    %Populate imu_data - Nx1 angular velocity values that will be
        integrated
    if(direc == 'x')
        gyro_data = gyro_gain_mag*input_struct.gyro_x;
    elseif(direc == 'y')
        gyro_data = gyro_gain_mag*input_struct.gyro_y;
    elseif(direc == 'z')
        gyro_data = gyro_gain_mag*input_struct.gyro_z;
    end
    time_data = input_struct.imu_time;

    %If IMU was physically flipped, allow for the option of rotating
        the
    %data 180
    if (flip_yn == 'n')
        flip_coeff = 1;
    elseif(flip_yn == 'y')
        flip_coeff = -1;
    else
        disp('PLEASE PUT y OR n FOR flip_yn!!!!')
    end

    %Populate the initial heading
    initial_heading = input_struct.initial_heading; %Degrees

    %Assign output
    output = (flip_coeff*cumtrapz(time_data,gyro_data)) +
        initial_heading;

end %End of function
```

Listing C.15: Code that takes in the user's step indices in the IMU data, the cumulative heading of the user, the RFID 2D initial position and its time of occurrence and then calculates the user's 2D coordinates. The user's position is only advanced when there is a step. A placeholder is left towards the end of the code if someone wants to fuse all RFID positions with all dead reckoned positions.

```
function [position_2d] = calculate_2d_position_struct_mk2_fxn(
    input_struct,gait_coeff,invoke_rfid_2d_positioning)
    % @brief - Function that caluclates the 2d position using imu and
    %    rfid data
    % @param[in] input_struct - Zone struct that contains all of the
        data
    %    required
    % @param[in] gait_coeff - Constant value for the user's ft/step
        travel distance
    % @param[in] invoke_rfid_2d_positioning - Whether to just use IMU
        for position or
    %  RFID. 'y' for yes/ 'n' for no (just use IMU). ZCA will always
        still
    %  use RFID to categorize regardless of whether 'y' or 'n' is put.
    % @param[out] position_2d - Estimated 2D position from IMU data (
        Nx2)->X:(:,1)
    % Y:(:,2)
    % @todo

    % Sources
    % https://blog.endaq.com/quaternions-for-orientation
    % https://github.com/jerabaul29/IntegrateGyroData

    %Time array of the IMU data
    imu_t = input_struct.imu_time;


    %Determine where the initial time reading falls in with the IMU
        data
    initial_time = input_struct.initial_enter_time;
    [~,insertion_idx] = min(abs(imu_t - initial_time));

    %pk_idx - Indices of the user's steps for the IMU data array
    pk_idx = input_struct.step_indices;
    %x,y initial position
    initial_position = input_struct.initial_xy;
    %gyro_angle - Array of angles calculated from the gyroscope data
    gyro_angle = input_struct.angle_gyro;
    %Initial heading
    angle(1) = input_struct.initial_heading; %Degree

    %Create array that has a 1 for when the user steps. The user's
        position
    %is advanced only when this array has a step value. The user is not
    %advanced in the absence of steps
    peak_binary_array = zeros(length(imu_t),1);
    peak_binary_array(pk_idx) = 1;


    %declare indexing variables and initial position
    xy_pair = initial_position; %ft?
    post_insertion_imu_position_2d(1,:) = xy_pair;
    i = 1;
```

```
for(index = insertion_idx:1:length(imu_t))
%Always update angle
    angle(index) = gyro_angle(index);

    %Peak find - increment displacement, then update xy_coord
    if(peak_binary_array(index) == 1)
        xy_pair(1) =  xy_pair(1) + (gait_coeff * cosd(angle(index))
           );
        xy_pair(2) =  xy_pair(2) + (gait_coeff * sind(angle(index))
           );
        i = i + 1;
        post_insertion_imu_position_2d(i,:) = xy_pair;
    end
end


%Determine the pre_insertion 2d position
xy_pair = post_insertion_imu_position_2d(1,:);%Initial condition
pre_insertion_imu_position_2d(1,:) = xy_pair;
i = 1;
for(index = insertion_idx:(-1):1) %Countdown from instertion idx
%Always update angle
    angle(index) = gyro_angle(index);

    %Peak find - increment displacement, then update xy_coord
    if(peak_binary_array(index) == 1)
        xy_pair(1) =  xy_pair(1) - (gait_coeff * cosd(angle(index))
           );
        xy_pair(2) =  xy_pair(2) - (gait_coeff * sind(angle(index))
           );
        i = i + 1;
        pre_insertion_imu_position_2d(i,:) = xy_pair;
    end
end
%Flip matrix as we built it up from the end
pre_insertion_imu_position_2d = flip(pre_insertion_imu_position_2d)
   ;

%Get rid of last row as it is redundant
pre_insertion_imu_position_2d = pre_insertion_imu_position_2d(1:(
   end-1),:);

%Concatenate the pre and post insertion value
imu_position_2d = cat(1,pre_insertion_imu_position_2d,
   post_insertion_imu_position_2d);

%RFID + IMU FUSION?
```

```
    if invoke_rfid_2d_positioning == 'n'
        position_2d = imu_position_2d;

    elseif invoke_rfid_2d_positioning == 'y'
        %//FIXME Need to fuse IMU and RFID data here
    end
end %End of function
```

Listing C.16: Function used to perform map matching. Based on the code described in [47].

```
function [map_matched_2d_coords] = map_match_fxn(input_2d_coords,
   line_map_array)
    % @brief For each input coordinate, calculate the distance between
       it and
    %   every line segment. The minimum distance is the closest line
       segment.
    % @param[in] input_2d_coords - The array of XY coordinates that
       need to
    %   be matched. Data format: [N x 2] where N is the number of XY
    %   coordinates. 2 corresponds to the [X,Y]
    % @param[in] line_map_array - The array of coordinates that mark
       the
    %   start and stop coordinates of a line. Data fromat: [M x 4]
       where M is
    %   the number of line segments in the map, and 4 corresponds to
    %   [X_begin, Y_begin, X_end, Y_end].
    % @param[out] map_matched_2d_coords - The array of map matched
    %   coordinates. Data format: [N x 2] where N is the number of XY
    %   coordinates. 2 corresponds to the [X,Y]

    %Code taken from:
    %https://stackoverflow.com/questions/849211/shortest-distance-
       between-a-point-and-a-line-segment
    %Contributing User: Joshua. Accessed: 3/17/21

    for idx = 1:1:size(input_2d_coords,1)
        %Temp variables
        tpx = input_2d_coords(idx,1); %x coordinate of point
        tpy = input_2d_coords(idx,2); %y cooridnate of point
        tx1 = line_map_array(:,1); %X coordinate of line segment
           beginning
        ty1 = line_map_array(:,2); %Y coordinate of line segment
           beginning
        tx2 = line_map_array(:,3); %X coordinate of line segment end
        ty2 = line_map_array(:,4); %Y coordinate of line segment end
```

```
A = tpx - tx1; %Vector from beginning of line segment to point
B = tpy - ty1; %Vector from end of line segment to point
C = tx2 - tx1; %X vector component of the line segment
D = ty2 - ty1; %Y vector component of the line segment

%Calculations
dot = (A.*C) + (B.*D); %Dot product
len_sq = (C.*C) + (D.*D); %Length calculation

for line_idx = 1:1:size(line_map_array,1) %Need to do the logic
     check for each line
     param = -1;
     if (len_sq(line_idx) ~= 0) %Check to make sure it is not a
         zero length vector
         param = dot(line_idx)./len_sq(line_idx);
     end

     %Check the three cases
     % Case 1: Matched point lies within segment
     % Case 2: Closest point in line segment is the beginning
     % Case 3: Closest point in line segment is the end
     if (param < 0) %Check case 2
          xx(line_idx) = tx1(line_idx);
          yy(line_idx) = ty1(line_idx);
          %Matches point to the beginning of the line if outside
              line
     elseif (param > 1) %Check case 3
          xx(line_idx)  = tx2(line_idx);
          yy(line_idx)  = ty2(line_idx);
          %Matches point to the end of the line if outside line
     else %Case 1
          xx(line_idx)  = tx1(line_idx) + param.*C(line_idx);
          yy(line_idx)  = ty1(line_idx) + param.*D(line_idx);
     end %End of if-else
end %End of for loop

%xx: x coordinate of the matched point!
%yy: y coordinate of the matched point!

%Calculate distance in x and y direction from point to matched
%point
dx = tpx - xx; %X Distance from matched point to
dy = tpy - yy; %Y Distance from matched point to
d_out = sqrt((dx.*dx) + (dy.*dy)); %Calculate distance
    magnitude
```

```
        [~,min_idx(idx)] = min(d_out);
        map_matched_2d_coords(idx,:) = [xx(min_idx(idx)),yy(min_idx(idx
           ))];
        clear tpx tpy tx1 ty1 tx2 ty2 temp_numer temp_denom
           temp_dist_to_segment

    end %End of for loop

end %End of function
```

Listing C.17: The main localization function. This script calls all of the previous functions. Takes in all of the pre-characterization files, data files, and user input to run. This script outputs the localized positions graphically.

```
clear all
close all
tic %Start recording program run time
%Heading Convention
%------------------90----------------
%--------------180-{}-0--------------
%------------------270---------------

%% USER_DEFINED GLOBAL VARIABLES
location = 'G:\CSM_GRAD\rfid_project\edgar_mine_results\data\140sec\
   run2\'; %WINDOWS
% location = '/Users/rj/Documents/RFID_project/edgar_mine_results/data
   /140sec/run2/'; %MAC

filename_imu = 'imu_out.txt';
filename_rfid = 'rfid_combined_data.txt';
filename_zone = 'zone_info.csv';
filename_lut = 'edgar_lut_rhcp1_30dbm.mat';
filename_zone_connect = 'zone_relation.mat'; %Give us variable:
   zone_relation struct
filename_line_map = 'zone_map.mat'; %This is what grabs the '
   line_map_array' variable

%Gait Constants
step_rob = 2.3;  %ORIGINAL was 1.3207 in brown. 2.3 in Edgar
user_gait = step_rob;


%Tag Parameters - last three number of tag identifier (HARDCODED)
number_of_msmnt_antennas = 2; %How many measurement antennas are used
   on the localization harness
```

```
%IMU_SAMPLE RATE
%If we sample at 100 Hz (assumed), then every sample is 10 ms. So ;
dt = 10/1000;

%Heading Tolerance
%Used in the heading correction code. If mean angle is greater than +/-
    the tolerance
%then apply correction. If not do nothing
heading_tol = 5; %degrees

%Gyro Heading Direction (Gyroscope direction that will be integrated to
%find the heading
gyro_heading_axis = 'y'; % Choice are 'x', 'y', 'z' ('z' is nominal)
gyro_gain = 1.0; % %called the 'Scale Factor' in the literature - Used
   if you need to scale
                %the time domain values of the gyroscope readings. Set
                   to 1 for no scaling
flip_180 = 'y'; %CCW turns should be postive and CW turns should be
   negative.
                %If the IMU is flipped physically then you have the
                   ability
                %to flip the heading 180 by the introduction of a minus
                %sign in the gyroscope heading function.
                %Put a 'y' to do this. 'n' to not.


%RFID 2D Positioning Inputs
tws = 0.5; %Time Window Size
wo = 0.25; %Window overlap
gni = 10; %Gauss newton iterations

%2D POSITION ANALYSIS TYPE
use_2d_rfid_pos = 'n'; %'n' for no rfid 2d position just imu, 'y' to
   fuse IMU and RFID 2d positioning estimates

%PLOTTING SWITCHING
plot_inter_results = 'n'; %'y' for yes, 'n' for no
plot_imu_only_and_rfid_only = 'y'; %'y' for yes, anything else for no.

%Apply map matching
map_match_yes_no = 'y';

%Rotate final results
result_rotat_angle = 25.2826; %Enter the angle one wishes to rotate all
    results.
```

```
                              %If you do not want to rotate results, put a
                                 zero
                              %CCW is positive

%Initial Zone (the zone the user starts in)
initial_zone = 1; %If this changes, make sure the zone connection map
   is
                        %altered so that
                        %zone_relation_struct(initial_zone,initial_zone)
                        %is a valid entry


%% LOAD IN DATA & SPLIT INTO COMPONENTS
%------------LOAD ZONE INFORMATION------------
opts = detectImportOptions([location filename_zone]);
zone_info = readtable([location filename_zone],opts);
number_of_zones = size(zone_info,1); %Zone info - tag coords, zone #,
   and zone type

%------------LOAD LUT------------
load(filename_lut)
lut_eqn = curve_fit_object;

%------------LOAD ZONE RELATION------
load(filename_zone_connect)
%Gives us the variable: zone_relation_struct

%------------LOAD BASIC MAP FOR MAP MATCHING---------
load(filename_line_map) %This is what grabs the 'line_map_array'
   variable

%------------IMU SPLIT------------
[imu_data] = extract_imu_data_ag(location,filename_imu);
accel_x = imu_data.ax; accel_y = imu_data.ay; accel_z = imu_data.az;
gyro_x = imu_data.gx; gyro_y = imu_data.gy; gyro_z = imu_data.gz;
imu_time = imu_data.time;


%------------RFID SPLIT------------
[rfid_data] = extract_rfid_data_factory(location,filename_rfid);
tag_ident_tail = rfid_data.tag_ident;
reader_time = rfid_data.reader_time;


[rfid_zone,max_poss_num_tags,tag_x_var_names,tag_y_var_names,
   tag_ident_var_names] = categorize_rfid_data_ntags(zone_info,
   tag_ident_tail);
```

```
%% Suppress Warning
%Suppress warnings
warning('off')
%Needed because the RFID 2D calculation generate
%many, many warnings about poorly scaled matrices
%Does not suppress errors.

%% Take IMU Data and Categorize it into zones
[imu_zone_indices, imu_zone_times, zone_progression, rfid_zone_indices]
    = categorize_imu_data(imu_time, reader_time, rfid_zone);


%% CREATE STRUCTS
% Assign a unique color, linestyle, and marker scheme to each zone.
for index = 1:1:number_of_zones
  %Plotting Settings
  [temp_color,temp_linestyle,temp_marker]= set_zone_unique_color(index)
     ;
  zone_struct(index).color = temp_color;
  zone_struct(index).linestyle = temp_linestyle;
  zone_struct(index).marker = temp_marker;
  clear temp_color temp_linestyle temp_marker

  %Assign zone type to the struct
  zone_struct(index).type = zone_info.type(index);

  %XY Coordinates
  for tag_index = 1:1:max_poss_num_tags
      %Ignore the label given to the tags when they are placed into the
          .xy
      %property of the struct; they are not all tag1_x, tag1_y, but
         instead
      %are tag1_x,tag2_x,...,tagN_x and tag1_y,tag2_y,...tagN_y, etc.
      zone_struct(index).xy(tag_index,:) = [zone_info(index,
         tag_x_var_names(1,tag_index)),zone_info(index,tag_y_var_names
         (1,tag_index))];
  end

end


%ASSUME TIME STARTS AT ZERO
%Initialization
%ztr: zone transition
imu_ztr_indices = [initial_zone imu_zone_indices size(imu_time,2)];
```

```
imu_ztr_times = [0 imu_zone_times imu_time(end)];
zone_count = size(zone_progression,2);
for index = 1:1:zone_count
    %///Data Assignment////
    %Zone Information
    data_struct(index).zone = zone_progression(index);
    temp_zone_ident = data_struct(index).zone;

    %SET THE INITIAL HEADING AND ESTIMATED POSITION (JUST IN CASE IT IS
    %NEEDED)
    if(index ~= 1)
        prev_zone_ident = data_struct(index-1).zone;
        data_struct(index).initial_heading = zone_relation_struct(
           prev_zone_ident,temp_zone_ident).fzt_heading; %degrees
        default_xy_pos = zone_relation_struct(prev_zone_ident,
           temp_zone_ident).xy_fz ;
    else %If index = 1, then just set the zone
        data_struct(index).initial_heading = zone_relation_struct(
           temp_zone_ident,temp_zone_ident).fzt_heading; %degrees
        default_xy_pos = zone_relation_struct(temp_zone_ident,
           temp_zone_ident).xy_fz ;
    end
    temp_initial_heading = data_struct(index).initial_heading;

    %RFID Information
    data_struct(index).possible_tag_tails = tag_name_pull_func(
       tag_ident_var_names,data_struct(index).zone,zone_info);
    data_struct(index).rfid_idx_begin = rfid_zone_indices(index);
    data_struct(index).rfid_idx_end = rfid_zone_indices(index+1)-1; %
       Subtract 2nd array value by 1
    [temp_rssi,temp_phase,temp_ident,temp_time] =
       split_rfid_data_by_antenna_and_tag_id(data_struct(index),
       rfid_data,number_of_msmnt_antennas);
    data_struct(index).sorted_rssi = temp_rssi;
    data_struct(index).sorted_phase = temp_phase;
    data_struct(index).sorted_ident = temp_ident;
    data_struct(index).sorted_time = temp_time;
    data_struct(index).complete_rfid_time_data = rfid_data.reader_time(
       data_struct(index).rfid_idx_begin : data_struct(index).
       rfid_idx_end);
    clear temp_rssi temp_phase temp_ident temp_time

    %Type of zone: 's' for straight section, 't' for T junction, 'b'
       for 90
    %degree bend
    data_struct(index).zone_type = zone_struct(temp_zone_ident).type;
```

```
%Zone plotting information
data_struct(index).color = zone_struct(data_struct(index).zone).
   color;
data_struct(index).linestyle = zone_struct(data_struct(index).zone)
   .linestyle;
data_struct(index).marker = zone_struct(data_struct(index).zone).
   marker;

%IMU Information
if index < zone_count
    imu_data_index_range = imu_ztr_indices(index):(imu_ztr_indices(
       index+1)-1);
elseif index == zone_count %Do not chop off last time sample
    imu_data_index_range = imu_ztr_indices(index):imu_ztr_indices(
       index+1);
end %End of if-else conditional
data_struct(index).imu_time = imu_time(imu_data_index_range);
data_struct(index).accel_x = accel_x(imu_data_index_range);
data_struct(index).accel_y = accel_y(imu_data_index_range);
data_struct(index).accel_z = accel_z(imu_data_index_range);
data_struct(index).gyro_x = gyro_x(imu_data_index_range);
data_struct(index).gyro_y = gyro_y(imu_data_index_range);
data_struct(index).gyro_z = gyro_z(imu_data_index_range);


%///Calculations///
%Calculate heading
data_struct(index).angle_gyro =
   calculate_gyro_angle_struct_cut_down(data_struct(index),
   gyro_heading_axis,gyro_gain,flip_180);

%Apply mean heading correction (Light Map Matching)
if string(data_struct(index).zone_type) == 's' %Straight section
   conditional
    %If mean angle is greater than +/- the tolerance
    %then apply correction. If not do nothing
    %Note:Might need to apply this at the 2d position level
    temp_mean_heading = mean(data_struct(index).angle_gyro);

    %Correct for heading values over 360?

    %Calculate difference between the initial heading (which should
        be
    %the average heading through the zone) and the calculated mean
    %heading through the zone
    diff = temp_mean_heading - temp_initial_heading; %Calculate
        difference
```

```
            if abs(diff) >= heading_tol
                data_struct(index).angle_gyro = data_struct(index).
                    angle_gyro - diff;
            end
        end

        %Calculate RFID 2D Position
        data_struct(index).rfid_2d_position = rfid_2d_pos_prototype_fxn_mk3
            (data_struct(index),zone_struct(temp_zone_ident),tws,wo,gni,
            lut_eqn,plot_inter_results);



        %CALCULATE INITIAL POSITION
        [temp_initial_xy,temp_initial_time] = initial_pos_calc_fxn(
            data_struct(index),default_xy_pos);
        data_struct(index).initial_xy = temp_initial_xy;
        data_struct(index).initial_enter_time = temp_initial_time; %
            Determine via RFID
        clear temp_initial_heading temp_initial_xy

        %Calculate number of steps
        data_struct(index).step_indices = calculate_step_count_struct(
            data_struct(index),dt);

        %Calculate IMU (+RFID) 2D position
        data_struct(index).xy_coord = calculate_2d_position_struct_mk2_fxn(
            data_struct(index),user_gait,use_2d_rfid_pos);

        %MAP MATCHING HERE
        if (map_match_yes_no == 'y' || map_match_yes_no == 'Y')
            data_struct(index).xy_coord = map_match_fxn(data_struct(index).
                xy_coord,line_map_array);
        end

        %Clear temp variables for this loop iteration
        clear temp_initial_heading temp_zone_ident temp_mean_heading
            default_xy_pos
    end %End of for loop

    %% PLOTTING
    if (plot_inter_results == 'y') || (plot_inter_results == 'Y')
        plot_rfid_results(data_struct,number_of_msmnt_antennas)
    end
    plot_2d_position_mk2(data_struct,zone_struct,line_map_array,
        result_rotat_angle)
    toc %Print execution time
```

```
%save data:
save('imu_processed_data.mat','data_struct','zone_struct')

%% FUNCTIONS
function [imu_data] = extract_imu_data_ag(location,filename_imu)
    % @brief Takes in the IMU .txt file and extracts the accleration
       and
    %  gyroscope data
    % @param[in] location - Where the.txt file is located
    % @param[in] filename - Name of the .txt file
    % @param[out] imu_data - Output struct that contains all of the
       output data
    % @todo
    imu_dataset = dlmread([location filename_imu]);

    %Grab accelerometer data
    accel = imu_dataset(:, [2 3 4]);
    imu_data.ax = accel(:,1);
    imu_data.ay = accel(:,2);
    imu_data.az = accel(:,3);

    %Grab Gyroscope data:
    gyro = imu_dataset(:, [5 6 7]);
    imu_data.gx = gyro(:,1);
    imu_data.gy = gyro(:,2);
    imu_data.gz = gyro(:,3);

    %Grab Magnetometer data:
    magnet = imu_dataset(:, [8 9 10]);
    imu_data.mx = magnet(:,1);
    imu_data.my = magnet(:,2);
    imu_data.mz = magnet(:,3);

    %Grab time domain data
    imu_data.time = imu_dataset(:, 11)';

end %End of function


function [rfid_data] = extract_rfid_data_factory(location,filename_rfid
   )
    % @brief Function extracts the data from the RFID .txt files
    % @param[in] location - Location for where the rfid data is stored
    % @param[in] filename_rfid - Filename of the rfid data
    % @param[out] rfid_data - Structure that contains all of the RFID
       data
    % @todo Create a version of this function that works with the
        custom tags

    rfid_dataset = readtable([location filename_rfid]);

    tag_ident_raw = table2array(rfid_dataset(:,1));

    %Remove the excess characters - HARDCODED BASED ON THE TAGS USED
        FOR THIS
    %EXPERIMENT (the end-5 and end-1 component)
    for index = 1:1:size(tag_ident_raw,1)
         temp = (regexprep(tag_ident_raw{index,1},'b',''));
         tag_ident(index,1) = str2double(temp(end-5:end-1));
    end

       rfid_data.tag_ident = tag_ident;
       rfid_data.reader_time = table2array(rfid_dataset(:,5)); %seconds
       rfid_data.antenna = table2array(rfid_dataset(:,4));
       rfid_data.rssi = table2array(rfid_dataset(:,3)); %dB
       rfid_data.phase = table2array(rfid_dataset(:,2)); %Degrees
end %End of function


function [color,linestyle,marker] = set_zone_unique_color(seed)
  % @brief -
  % @param[in] seed - the random seed
  % @param[out] color - Plot line color
  % @param[out] linestyle - Plot linestyle for zone
  % @param[out] marker - Plot line marker
  % @todo

  %Ripped from Alec Weiss's (Sheekaboom) code:
  %https://github.com/Sheekaboom/WeissTools/tree/master/WeissTools/
     MATLAB

  %Color list
   colors = ([ 0   ,50  ,135; 200,20  ,30  ; 0   ,120,0   ; 120,90  ,0   ;...
              150,50  ,165; 10  ,145,120; 0   ,0   ,0   ; 220,150,20  ;...
              200,150,200; 255,255,0   ; 255,0   ,128; 150,150,0   ;...
              0   ,255,255; 0   ,255,0   ; 255,80   ,255]/255);

  %LineStyle
  linestyles = '-|--|:|-.';
  markers     = '.|+|o|*|x|s|d|^|v|>|<|p|h';
  ls_cell = repmat(strsplit(linestyles,'|'),1,4);
  mk_cell = strsplit(markers,'|');
  %Output
  color = colors(mod(seed-1,length(colors))+1,:);
  linestyle = ls_cell{mod(seed-1,length(ls_cell))+1};
  marker = mk_cell{mod(seed-1,length(mk_cell))+1};

end %End of function


function plot_2d_position_mk2(complete_input_struct,input_zone_struct,
    map_approx,rotate_results_angle)
    % @brief -
    % @param[in] complete_input_struct -
    % @param[in] input_zone_struct -
    % @param[in] map_approx - array of coordinates to make the line
    %   approximation of the map
    % @param[in] rotate_results_angle - Angle to rotate all results by
    % @todo - Finish comments, make beacons the same color as lines,
        put
    %   black circle around beacons to signify they are not apart of
        the data

    %Rotation matrix
    map_approx_rot(:,[1,2]) = rotate_2d_coordinates(map_approx(:,1),
       map_approx(:,2),rotate_results_angle);
    map_approx_rot(:,[3,4]) = rotate_2d_coordinates(map_approx(:,3),
       map_approx(:,4),rotate_results_angle);

    figure;
    hold on
    grid minor
    xlabel('X␣Coordinate␣(ft)')
    ylabel('Y␣Coordinate␣(ft)')
    set(gca,'FontSize',24)
    %Plot Map Approximation
    for index = 1:1:size(map_approx,1)
        plot(map_approx_rot(index,[1,3]),map_approx_rot(index,[2,4]),'k
            ','LineWidth',2)
    end

    %Plot lines
    for index = 1:1:size(complete_input_struct,2)
        temp_x = complete_input_struct(index).xy_coord(:,1);
        temp_y = complete_input_struct(index).xy_coord(:,2);
        temp_rot = rotate_2d_coordinates(temp_x,temp_y,
           rotate_results_angle);
        temp_color = complete_input_struct(index).color;
        temp_linestyle = complete_input_struct(index).linestyle;
        temp_marker = complete_input_struct(index).marker;
        %plot(temp_x,temp_y,'LineStyle',temp_linestyle,'Color',
            temp_color,'Marker',temp_marker,'LineWidth',4,'MarkerSize
            ',15);
        plot(temp_rot(:,1),temp_rot(:,2),'LineStyle',temp_linestyle,'
            Color',temp_color,'Marker',temp_marker,'LineWidth',4);
        clear temp_x temp_y temp_color temp_linestyle temp_marker
    end

    %Plot beacon locations
    for index = 1:1:size(input_zone_struct,2)
        beacon_x_temp = table2array(input_zone_struct(index).xy(:,1));
        beacon_y_temp = table2array(input_zone_struct(index).xy(:,2));
        beacon_rot = rotate_2d_coordinates(beacon_x_temp,beacon_y_temp,
            rotate_results_angle);
        plot(beacon_rot(:,1),beacon_rot(:,2),'rd','LineWidth',2)
    end

    hold off
end

function [tag_tail_array] = tag_name_pull_func(tag_ident_var_names,
   zone_index,zone_table)
% @brief Takes in the zone info and the zone index to output what are
   the
%  tails (the last couple of digits)of the RFID tag identifiers for
   that
%  zone
% @param[in] tag_ident_var_names - 1xM cell, where M is maximum
   possible
%  number of tags for any given zone
% @param[in] zone_index - double value that determine which zone we are
%  operating in
% @param[in] zone_table - Complete tabular zone data of size NxP.
%  Where N is the total number of zones and P is the number of
   variables
% @param[out] tag_tail_array - An array of doubles corresponding to the
%  possible tails that will be seen in this zone
% @todo

tag_tails  = table2array(zone_table(zone_index,[tag_ident_var_names]));

%Cutdown NaN entries
tag_tail_array = tag_tails(~isnan(tag_tails));

end%end of function
```

```
function [output_rssi,output_phase,output_ident,output_time] =
    split_rfid_data_by_antenna_and_tag_id(input_struct,input_rfid_data,
    num_ant)
% @brief Function that takes in the complete RFID data and partitions
    it
%   via the tags in the zone and the number of msmnt antennas
% @param[in] input_struct - Complete input data struct
% @param[in] input_rfid_data -  struct that contains all of the rfid
    data
% @param[in] num_ant - number of antennas on acquisition system
% @param[out] output_rssi - Rssi values in cell array of size QxR,
    where Q is the number
%   of tags associated with the zone and R is the number of msmnt
    antennas
% @param[out] output_phase - phase values in cell array of size QxR
% @param[out] output_ident - Tag ID values in cell array of size QxR
% @param[out] output_time - Time values in cell array of size QxR
% @todo
     temp_idx_begin = input_struct.rfid_idx_begin; %Shorter var name
     temp_idx_end = input_struct.rfid_idx_end; %Shorter var name

     %Separate rss, phase, and ident, into separate arrays based on
        antenna
     %Note: This will not work with the previous acquistion code that
        did not list
     %the antenna!!
     temp_rssi_data = input_rfid_data.rssi(temp_idx_begin:temp_idx_end
        ,1);
     temp_phase_data = input_rfid_data.phase(temp_idx_begin:temp_idx_end
        ,1);
     temp_tag_ident_data = input_rfid_data.tag_ident(temp_idx_begin:
        temp_idx_end,1);
     temp_ant_data = input_rfid_data.antenna(temp_idx_begin:temp_idx_end
        ,1);
     temp_time_data = input_rfid_data.reader_time(temp_idx_begin:
        temp_idx_end,1);
     for ant_index = 1:1:num_ant
         %Find indices of the rfid data for this zone, that corresponded
              to
         %which antenna
         ant_indices = find(temp_ant_data == ant_index);
         temp_ant_split_rssi{1,ant_index} = temp_rssi_data(ant_indices
            ,1);
         temp_ant_split_phase{1,ant_index} = temp_phase_data(ant_indices
            ,1);
         temp_ant_split_ident{1,ant_index} = temp_tag_ident_data(
            ant_indices,1);
         temp_ant_split_time{1,ant_index} = temp_time_data(ant_indices
            ,1);
         clear ant_indices

         %Separate the data further based on tag id
         for tag_index = 1:1:size(input_struct.possible_tag_tails,2)
             %Find indices that correspond to the specific tag id
              this_iter_tag_tail = input_struct.possible_tag_tails(
                 tag_index);
              id_indices = find(temp_ant_split_ident{1,ant_index} ==
                 this_iter_tag_tail);

              if ~isempty(id_indices) %id array is not empty
                  temp_ant_tag_split_rssi{tag_index,ant_index} =
                     temp_ant_split_rssi{1,ant_index}(id_indices);
                  temp_ant_tag_split_phase{tag_index,ant_index} =
                     temp_ant_split_phase{1,ant_index}(id_indices);
                  temp_ant_tag_split_ident{tag_index,ant_index} =
                     temp_ant_split_ident{1,ant_index}(id_indices);
                  temp_ant_tag_split_time{tag_index,ant_index} =
                     temp_ant_split_time{1,ant_index}(id_indices);
              else %id array is empty
                  temp_ant_tag_split_rssi{tag_index,ant_index} = [];
                  temp_ant_tag_split_phase{tag_index,ant_index} = [];
                  temp_ant_tag_split_ident{tag_index,ant_index} = [];
                  temp_ant_tag_split_time{tag_index,ant_index} = [];
              end %End of if else

              clear id_indices
         end %end of tag id for loop
     end %end of antenna index for loop

%Finalize output
output_rssi = temp_ant_tag_split_rssi;
output_phase = temp_ant_tag_split_phase;
output_ident = temp_ant_tag_split_ident;
output_time = temp_ant_tag_split_time;
end%End of function

function plot_rfid_results(input_struct,num_antennas)
% @brief - A void function that plots the rssi values of the rfid data.
% @param[in] input_struct - Fully developed data struct (all zone
%   transitions)
% @param[in] num_antennas - Number of antennas used by the acquistion
%   system
% @todo
```

```
%Each antenna will have its own linestyle
%Each tag will have its own color
for index = 1:1:num_antennas
    [~,temp_linestyle,~]= set_zone_unique_color(index);
    ant_linestyle_array(index) = string(temp_linestyle);
    clear temp_linestyle
end

%Loop across zones
for index = 1:1:size(input_struct,2)
    %Give each tag a unique color
    for color_index = 1:1:size(input_struct(index).sorted_rssi,1)
        [temp_color,~,~]= set_zone_unique_color(color_index);
        tag_color_array(color_index,:) = temp_color;
        clear temp_color
    end

    %Generate new figure for each zone transition
    figure
    hold on
    set(gca,'FontSize',24)
    grid minor
    xlabel('Time(s)')
    ylabel('RSSi(dBm)')
    title(['Zone␣Transition:␣',num2str(index),',␣Zone␣#:␣',num2str(
       input_struct(index).zone)])
    legend_entry_array = []; %Clear out the legend entry array
    for tag_index =  1:1:size(input_struct(index).sorted_rssi,1) %Loop
       across tags
        for ant_index = 1:1:size(input_struct(index).sorted_rssi,2) %
           Loop across antennas
            temp_x = input_struct(index).sorted_time{tag_index,
               ant_index};
            temp_y = input_struct(index).sorted_rssi{tag_index,
               ant_index};
%             temp_color = input_struct(index).color;
            temp_linestyle = ant_linestyle_array(ant_index);
%             temp_marker = struct(index).marker;
            if (~isempty(temp_x) && ~isempty(temp_y))
                stem(temp_x,temp_y,'Color',tag_color_array(tag_index,:)
                   ,'LineStyle',temp_linestyle,'LineWidth',2)
                temp_tag_ident = input_struct(index).sorted_ident{
                   tag_index,ant_index}(1,1);
                temp_legend_entry = string(['Tag:␣',num2str(
                   temp_tag_ident),',␣Ant:␣',num2str(ant_index)]);
                legend_entry_array = cat(2,legend_entry_array,
                   temp_legend_entry);
                clear  temp_x temp_y temp_tag_ident  temp_legend_entry
            end%End of if else
        end
    end
    legend(legend_entry_array)
    hold off
end %End of zone for loop
end %end of function



function [out_2d_pos,out_time_step] = initial_pos_calc_fxn(input_struct
    ,baseline_xy_estimate)
 %IF RFID DATA AVAILABLE, USE RFID DATA
    if(size(input_struct.rfid_2d_position,1) > 0) %Make sure there is
        an RFID position
        out_2d_pos = input_struct.rfid_2d_position(1,1:2);
        out_time_step = input_struct.rfid_2d_position(1,3);
    %IF NOT, ESTIMATE IT FROM THE TAGS AND THE PREVIOUS ZONE TRANSITION
    else
        out_2d_pos = baseline_xy_estimate; %Hardcoded appriximations
        out_time_step =  input_struct.complete_rfid_time_data(1,1); %
            Assign this as the time of first tag read
    end %End of if-else
end %End of function



function [output_matrix] = rotate_2d_coordinates(input_array_x,
    input_array_y,rot_angle)
% @brief Function that takes arrays of x,y coordinates and rotates them
% @param[in] input_array_x - x coordinate matrix of either length 1xN
    or
%   Nx1
% @param[in] input_array_y - y coordinate matrix of either length 1xN
    or
%   Nx1
% @param[in] rot_angle - rotation angle in degrees
% @param[out] output_matrix - Nx2 matrix where N is the number of
%   coordinates and the 1st column are the x coordinates and the 2nd
    column
%   are the y coordinates
    output_matrix = zeros([length(input_array_x),2]); %Preallocate
    for index = 1:1:length(input_array_x)
        output_matrix(index,1) = (input_array_x(index)*cosd(rot_angle))
            - (input_array_y(index)*sind(rot_angle));
        output_matrix(index,2) = (input_array_x(index)*sind(rot_angle))
            + (input_array_y(index)*cosd(rot_angle));
    end
end %End of function
```