2017

Correcting Pedestrian Dead Reckoning with Monte Carlo Localization Boxed for Indoor Navigation

Akira T. Murphy
Colby College

COLBY COLLEGE

BACHELOR'S THESIS

Correcting Pedestrian Dead Reckoning with Monte Carlo Localization Boxed for Indoor Navigation

Author:
Akira MURPHY

Supervisor:
Dr. Ying LI

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Arts

in the

Computer Science

May 20, 2017

Declaration of Authorship

I, Akira Murphy, declare that this thesis titled, “Correcting Pedestrian Dead Reckoning with Monte Carlo Localization Boxed for Indoor Navigation” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this College.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this College or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________

Date: ______________________

Colby College

Abstract

Ying Li
Computer Science

Bachelor of Arts

Correcting Pedestrian Dead Reckoning with Monte Carlo Localization Boxed for Indoor Navigation

by Akira MURPHY

Localization of phones is a ubiquitous part of the modern mobile electronics landscape. However, there are many situations where the current method of networked localization fails. A Pedestrian Dead Reckoning System where the location of the user is calculated by counting the steps and direction of the user was implemented as an iOS app with python for data analysis. A novel algorithm for wireless sensor localization using Ad-Hoc Bluetooth networks was proposed. A small experiment was performed proving that the system is nearly equal to state of the art algorithms.

Acknowledgements

I would like to thank my research adviser Ying Li for her great patience and for inspiring me to work on networks.

Contents

List of Figures

List of Tables

List of Abbreviations

BLE	**B**luetooth **L**ow **E**nergy
GCS	**G**lobal **C**oordinate **S**ystem
GPS	**G**lobal **P**ositioning **S**ystem
LCS	**L**ocal **C**oordinate **S**ystem
MANET	**M**obile **A**d-hoc **NET**works
MCBL	**M**onte **C**arlo **B**oxed **L**ocalization
PDR	**P**edestrian **D**ead **R**eckoning
WSN	**W**ireless **S**ensor **N**etworks

Physical Constants

Peak Minimum	$a_{peak} = 1.00ms^{-2}$
Peak to Peak Threshold	$a_{pp} = 0.005ms^{-2}$
Step Length Algorithm Selection Threshold	$a_{\tau}^{step} = 3.230ms^{-2}$
Step Length Constant Scale	$\beta = 0.38454 \sqrt[\frac{3}{4}]{m}\sqrt{s}^{-1}$
Step Length Constant Offset	$\gamma = -0.32734m$
Moving Average Filter Window Size	$W = 5$

List of Symbols

a_z	Filtered Acceleration in the global coordinate system (GCS) along the z-axis	ms^{-2}
a_{LCS}	Acceleration in the local coordinate system (LCS)	ms^{-2}
a_{GCS}	Acceleration in the global coordinate system (GCS)	ms^{-2}
l	Step Lengths	m
ψ	Yaw or Heading	rad
$\mathbf{q}$	Heading quaternion	
σ	Constant factor the step finding constants β and γ were scaled by	
t^{peak}	Step times detected by looking for peak value over a threshold	
t^{pp}	Step times detected by looking a_z values above others in window W	
t^{slope}	Step times detected by looking at slopes	
$t^{heading}$	Headings at step valleys	rad

For Christina

Chapter 1

Introduction

Mobile phones are a ubiquitous part of the modern landscape. Most people carry them at all times, and rely on them to complete any number of basic tasks. Of these, one of the most common is to guide the user from place to place and provide geographic context. Localization is critical to providing a user with directions, or direct services to the user as in apps like Uber and Lyft. More recently interest has grown in using localization data to give users information relating to their current location like nearby shops. To do so, a natural first step is to know where the user is, called the problem of *localization*. Since the explosion of cellular phones over the last two decades, many different strategies have been proposed. Currently, the most widely used are the **G**lobal **P**ositioning **S**ystem or GPS, and using know cell tower locations (provided by cell companies) or WiFi points (collected by traveling while recording open WiFi services, and commonly called *wardriving*) to triangulate the user's location. Smart phone platforms like Android or iOS use a combination of these strategies in their core location services to return a phone's last location [1] [2].

However, all three strategies are reliant on access to at least one in place network infrastructure. GPS works by connecting to a system of satellites around the earth, cellular towers rely on the user having a cellular network connection, and WiFi connections require a rebuilt Internet framework for the user. However, there are many situations in which none of these systems can be accessed. For instance, in remote wilderness locations like Acadia National Park in Maine there may be no cellular service or Internet access, and GPS may become unreliable as it requires clear line of sight to the satellites in the sky. Another example is a building, where both GPS services and cellular data may not be able to penetrate the walls, and there may not be enough WiFi access points to reliably locate the user. Finally, in a rescue situation after some sort of disaster we may see all infrastructure fail, leaving emergency responders with no way to accurately locate themselves or use services reliant on location data. All of these present significant challenges for computer science.

In order to solve these challenges, interest has grown around ways to locate users without a network infrastructure. The most common method has been to use some kind of *dead reckoning*, the act of repeatedly updating a known location with estimates of speed, and then use another source of information to correct errors from dead reckoning [3].

1.1 Ad-Hoc Networks

Simultaneous with the growth of interest in the problem of localization of phones is the growth of interest in the problem of mobile network localization. Take a group of wireless sensors, all linked in a network. In particular, this thesis is interested in the idea of *Ad-Hoc Networks*, networks where there is no fixed routing infrastructure between nodes, but each node acts to pass along packages itself. So, we can imagine a group of phones which pass messages between them using Bluetooth or WiFi without sending messages through a router or some other network component as a simple Ad-Hoc Network. Many WSNs are also *Mobile Ad-Hoc Networks* abbreviated MANETs, where each node passes messages between each other while also moving. So, going back to our animal example, each wireless sensor node may be attached to a moving animal, and form connections when it is in range of another device, and break connections as the device moves away from nodes. Similarly, we can think of phones in the pockets of different people as forming a MANET if all are connected by some Ad-Hoc network. As people go a bout their day, connections are formed, and then disconnected as different users pass each other by.

Combining these two approaches leads to some interesting observations. Since phones can be both wireless sensor networks and gather significantly more sensor data than many WSN nodes, they have properties that can use ideas from both areas. This thesis involves using two ideas, PDR and network positioning, to try and improve on localization algorithms.

1.2 MANET Localization

I am particularly interested in one kind of network. As an example, consider a network that is used to gather data on wildlife. Each animal has a small device, abstractly a *node* that measures some data on the animal. Each node is wireless, and sends data through other nodes. These sorts of systems are referred to as a *Wireless Sensor Network* (WSN). Many of these devices may be too small or have too little battery to have a GPS or cellular receiver built in, and since they may move into areas with an Internet infrastructure, will not be connected to a fixed WiFi system. However, many uses for these systems may require localization information. Therefore, many different algorithms have been suggested for localization of individual nodes within the WSN.

In this study I use PDR coupled with a WSN algorithm using ad-hoc Bluetooth connections between iOS devices acting as a mobile WSN. with messages between phones, a better idea of location can be generated by using the various different mobile phones as what are refereed to as *landmarks*. In particular, in this case we use mobile landmarks, meaning that the landmarks themselves move, not just the nodes, as in this set up the nodes are the landmarks and vice versa. Because the maximum range is known already, we can infer that the user must be within the range of any connected nodes, and thereby use their position to help locate the user.

1.3 Pedestrian Dead Reckoning

The first step uses ideas from Pedestrian Dead Reckoning, a specific type of dead reckoning mentioned above. Since we can measure the acceleration and angular velocity of the phone, we can get an idea of the motion of the user. Then, using this data and assuming that the user is a pedestrian who is walking we can determine when the user takes a step. Compare this with *Inertial Navigation* where there is no assumption about the motion of the user and the acceleration and angular velocity are used to find the velocity and attitude, and then the position with double integration given a known change in time. This method is generally not usable with a phone, as the commercial sensors in phones lack the necessary accuracy [4]. Then, as each step is detected using acceleration data, the heading is found using gyroscope and magnetosensor data. With the last known position, the heading, and stride length, location can then be updated by adding the displacement of the step in the direction of the heading to the last known location.

1.4 Thesis Outline

In chapter 2, there is a literature review of PDR systems and Monte Carlo Localization for WSNs. Following in chapter 3 is an explanation of the method broken down by the architecture of the project. There is a brief description of the overall architecture, the iOS client, the server, and the post data collection processing. Chapter 4 contains the results of experiments using the platform. Chapter 5 is a conclusion and a plan for future work.

Chapter 2

Related Work

Many different approaches to phone localization and pedestrian dead reckoning (PDR) have been used [5–9]. All of these methods use the same fundamental approach: using accelerometers, gyroscopes, and magnetosensors to calculate the number of steps and direction for PDR, and then trying to error correct using some other source of information. Several have used bluetooth as error correction[10, 11]. However these approaches so far suffer from a similar set of problems: 1) Requiring fixed network landmarks set up before localization, 2) using Recieved Signal Strength Indication (RSSI) to get distance from the source even though RSSI is not standardized for bluetooth devices and is inaccurate, 3) using a complicated network relying on many different heterogeneous devices, and 4) requiring the user to hold the phone face up, an awkward position for daily use. Some have tried correcting a few of these problems, but not all.

There has also been significant work on localization of wireless sensor networks (WSN) with mobile landmarks [12, 13]. This has assumed that the WSN nodes have weak bounds on their movement, however thanks to the advances in internal sensors of smartphones we can get much stronger limits on the movement of the phone. In particular, PDR has become feasible within the past decade even without specialized inertial measurement units, which have a high cost and are not common among non-specialists. These systems are also usually tested in simulation and not implemented on a real device.

Constandache et. all describe one of the first systems of GPS-less phone localization, reliant on having a map availible [5]. Using the map to find segments that are walkable or not, they match up segments with user sensor readings, choosing the best path. Initial location is given by a GPS reading. This system relies on having certain segments marked as “walkable” or not. Another system is Wang et. all who assume a dense landscape of static landmarks, identifiable from WiFi, and magnetic or physical signals [14]. Their system UnLoc system uses k-clustering to discover locations of landmarks after several passes, using several predefined types of landmarks commonly found in indoor environments. They combine these landmarks with PDR data to correct for inaccuracies to correct inaccurate heading data. Both of these examples use static access to networks, including WiFi and GPS signals. Kang and Han present a PDR system for modern phones which has both a step detection system and dynamic step length estimation model, using the fact that modern consumer phones typically contain 3-axis accelerometer, 3-axis gryoscope, and a magneto sensor [9]. They were able to use the system without having the phone facing directly up by combining readings from the magnetosensor and the 3-axis gyroscope, a major improvement.

Bluetooth aided localization has been studied by many authors. Early work by Feldmann et. all used Bluetooth signal strength through RSSI and triangulation with known static landmarks to locate a PDA [10]. They used least squares estimation to triangulate given distances from a polynomial fit to RSSI. A more recent approach was taken by Mirowski et. all in their SignalSLAM system [11]. SignalSLAM uses the GraphSLAM algorithm to locate and map the user, combining the signal readings and PDR measurements of a group of networked phones. A variety of different static landmarks were used, including WiFi hotspots, bluetooth RSSI, LTE, GPS, and NFC data. Using this they reconstructed a map of an indoor space. The actual localization took place off the phone on a seperate back end due to the computational complexity of GraphSLAM. A more recent approach by Liao et. all uses bounding boxes from known static bluetooth landmarks [8]. They used bounding boxes and triangulation to get initial estimates of position, which they then update with PDR, but did not try and combine the PDR values and bluetooth values beyond that. They did not use RSSI values except as a threshold of being within range, avoiding the problem of inaccurate RSSI values.

In a seperate area, WSNs are a popular field. Several approaches have been suggested for systems with mobile landmarks and mobile nodes. Hu and Evans worked on a system that uses a Monte Carlo simulation filtered by connection data [13]. They use both “positive” data of a succesful connection and “negative” data of a lost connection to filter - assuming that a node within range of another node is within that area, and that a node not in the range of a recently lost connection must have moved out of range of the previous node. They also use 2nd order connections. Nodes are assumed to have some maximum velocity from their previous location, but that is the only bound on the movement of the node. Baggio and Langendoen working from Hu and Evans, used an algorithm that pre-filtered instead of post filtered in the Monte Carlo simulation, and disregarded negative data [12]. Because radio signals are easily blocked by walls and other obstructions even when in range, Baggio and Langendoen found that by excluding areas from disconnections the data became more inaccurate, as it’s possible the node had not gone out of range, but had simply gone behind some solid structure. Both approaches were tested on simulations and not in a real implementation.

All the reviewed phone localization systems assumed fixed landmarks, whether they were pre located or discovered. Many algorithms including [6, 11, 14] rely on making several different passes, either with the same device or as part of a network. Phone localization assumes some fixed infrastructure like WiFi systems. WSNs algorithms on the other hand are built with very little information on the movement of the nodes besides network data for localization. Therefore we propose a system assuming mobile landmarks, but also using the PDR data to provide more accurate bounds on the movement of the node. Liao came closest to fixing several of these problems, but they used bluetooth only for initialization, instead of trying to combine bouth sources of information when they conflict.

Chapter 3

Method

3.1 Architecture

In my study the system was built in two parts, a client side on the iPhone, and a server side run on a PC. The system was split in order to make running the analysis on data easier, this way I could turn on and off features at different times. In particular, I could analyze the effect of using Bluetooth correction by comparing the result on the same data using the Bluetooth algorithm, and then not using Bluetooth reliant solely on the PDR algorithm. This does present the problem that the algorithm was not designed for the constraints of running on a mobile device, and when transported to the phone (as planned), may be too computationally or energy intensive. However, due to the overall simplicity of the algorithm, I believe it will be doable as part of possible future work.

The client collects the necessary sensor data, and sends it back to the server which records the data in log files. Afterwards, data analysis was performed. I will describe each step in sequence. For a full diagram of the system, see Figure 3.1

3.2 Client

When a user opened the app on the iPhone (see Figure 3.2), they entered some identification (name, experiment number), and a fixed landmark given to them by the experimenter to serve as a known starting location. Then each client opened a TCP connection to the server, and sent accelerometer, gyroscope, magnetosensor, and Bluetooth data to the server. All iphones have an accelerometer, for example the iPhone 6 comes with both a three-axis Bosch BMA280 accelerometer and a MPU-6700 six-axis accelerometer from InvenSense, and all iPhones past the 3G also have a magnetosensor and a gyroscope (note that the gyroscope is often built into the accelerometer as in the example iPhone 6). All iPhones of models 4s and later have Bluetooth Low Energy (BLE) built in as well. Therefore, 4s was the minimum the system could be run on. The app read accelerometer, gyroscope, and magnetosensor data at 60Hz, and registered BLE connections and disconnections. The app sent a message to the server indicating that a connection was made. Similarly, the user could input that they are at a known fixed location, to allow the experimenter to analyze the data after the experiment.

When done, the user simply closed the app or starts a new session.

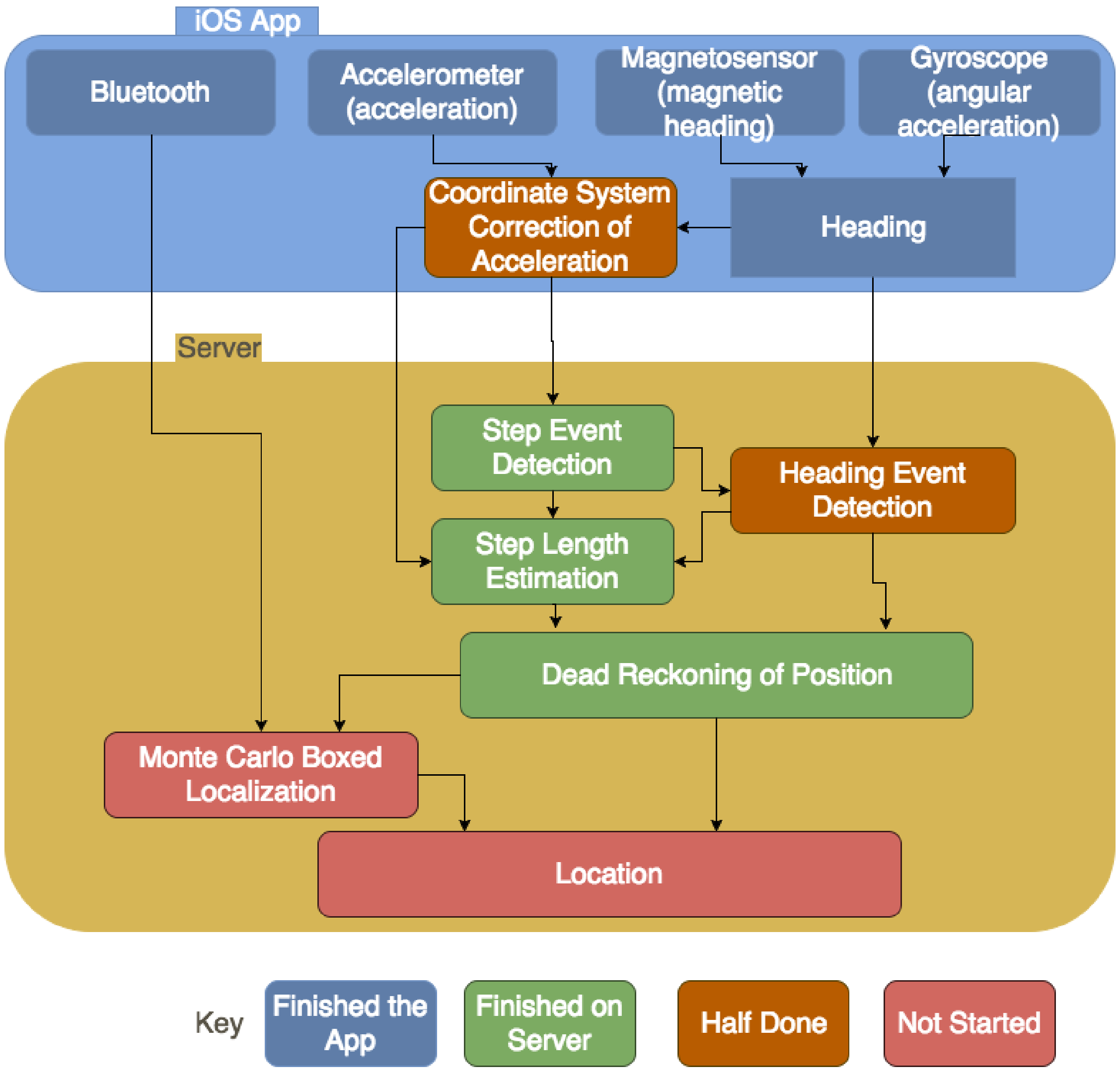
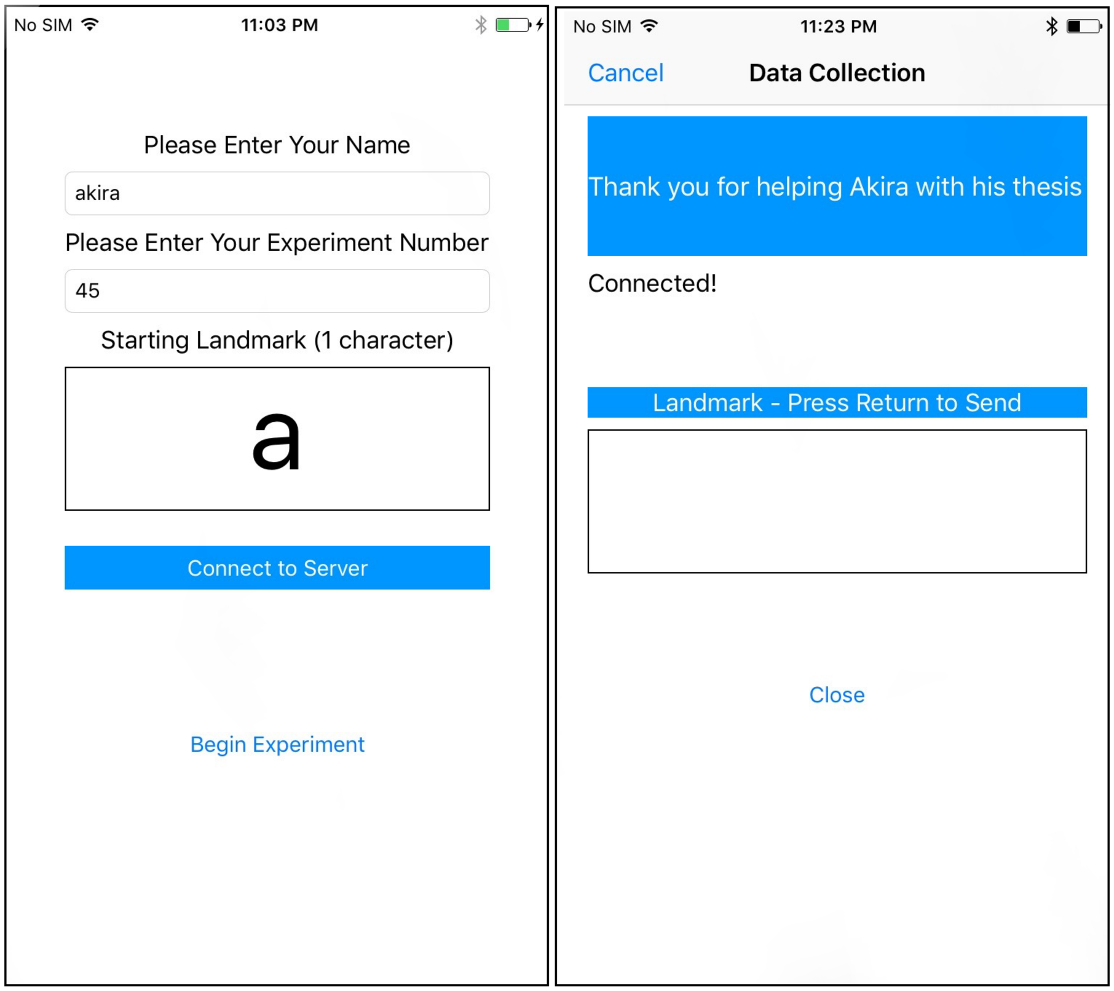

iOS App
Bluetooth
Accelerometer (acceleration)
Magnetosensor (magnetic heading)
Gyroscope (angular acceleration)
Coordinate System Correction of Acceleration
Heading
Server
Step Event Detection
Heading Event Detection
Step Length Estimation
Dead Reckoning of Position
Monte Carlo Boxed Localization
Location
Key
Finished the App
Finished on Server
Half Done
Not Started

FIGURE 3.1: A chart of the overall system, and how information flows through the system. Certain sections are incomplete as of this time and are labelled so.



Please Enter Your Name: akira

Please Enter Your Experiment Number: 45

Starting Landmark (1 character): a

Thank you for helping Akira with his thesis

Connected!

FIGURE 3.2: The left screen is the intro screen, where the user inputs some identifying information. The right screen is the screen once the experiment has begun, where the user can put down landmarks, identified by a single letter, to help the experimenter calculate error

3.3 Server

The server is a simple system in Python that accepted TCP connections from each client, and then logged each raw transmission in the order of reception. It was not built to handle more than 20 or so connections, since it was only for experimental purposes.

3.4 Data Processing

The following approach to Data Processing for PDR was primarily drawn from Kang and Han [9], with a different heading system, and the Bluetooth work was from the algorithm presented by Baggio and Langendoen [12]. The overall approach to data processing was as follows: convert the acceleration readings into the global coordinate system using the attitude, use acceleration to detect steps, use the attitude to find the user's heading, detect step length, predict location from PDR using the detected steps and heading, and then correct using Bluetooth data.

3.4.1 Coordinate System Transformation

In order for the data to be usable, there must be a coordinate system transformation from the local coordinate system (LCS) of the iPhone, to the global coordinate system (GCS) used by all the devices. This is done in two steps: correcting for the heading and gravity, and correcting for the initial heading.

Since I needed a fixed starting point, I needed both a known location in the coordinate plane, and a known 3-axis orientation or attitude. There are many ways to represent the attitude of an object. Two common ways are with *Euler Angles* and *quaternions*. The Euler Angles of an object are the angles around each of the 3 axes: the *pitch* around the x-axis, the *roll* around the y-axis, and the *yaw* around the z-axis. I assumed that the phone was held flat facing up when the user starts with the phone in front of the user, so that the z-axis was pointing up away from the ground. See figure 3.3 for the location of the axes on an iPhone for all iPhone sensors. This was important as it means that the yaw of the phone was the user's heading, assuming that the user walks forwards, where the heading was the user's direction of movement on the 2-d coordinate plane. The quaternions are an extension of the complex numbers in the form $a + b\hat{\imath} + c\hat{\jmath} + d\hat{\mathbf{k}}$, where $\hat{\imath}, \hat{\jmath}, \hat{\mathbf{k}}$ are the fundamental quaternion units. Quaternions can be used to represent attitudes in 3-dimensional space. Particularly for our case, given a vector **a** (which I cant treat as a quaternion with a real part equal to 0) , and a quaternion **q** that represents a rotation, then the rotation of **a** by **q** was given by

$$\mathbf{qaq}^{-1} \tag{3.1}$$

where I treat multiplication as the Hamiltonian product, and the inverse as the Hamiltonian inverse [4].

Apple's Core Motion package, which allows the user to access accelerometer, gyroscope, and magnetosensor data, provides for a quaternion representation of the attitude of the phone relative to the starting location. Therefore, the client before sending the data rotated the raw acceleration vector from the accelerometer a_{LCS} by the attitude **q**, and sent $a_{GCS} = \mathbf{qaq}^{-1}$ to the server. Quaternions were chosen as Euler angles are

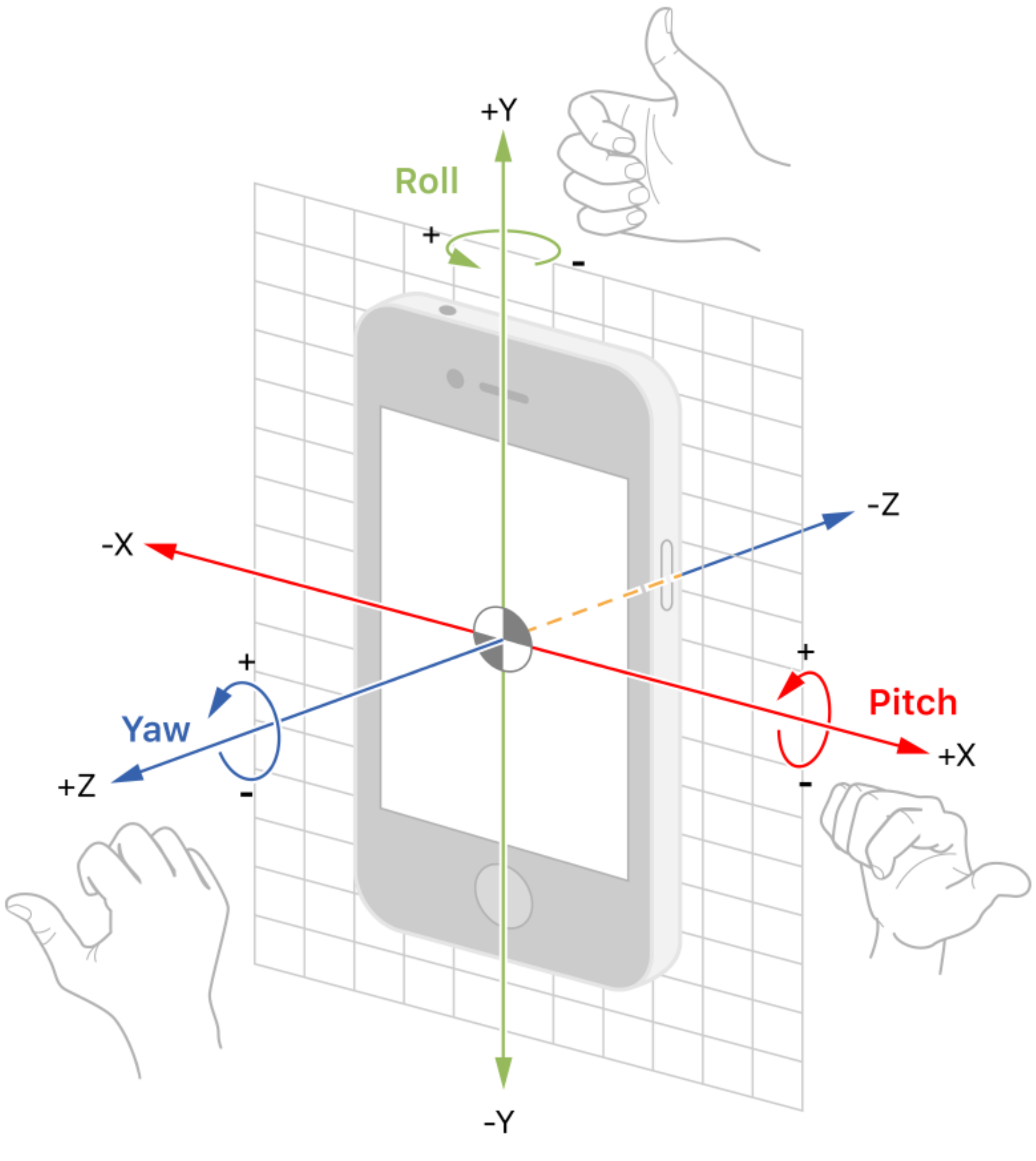


FIGURE 3.3: The axes of an iPhone. [15]

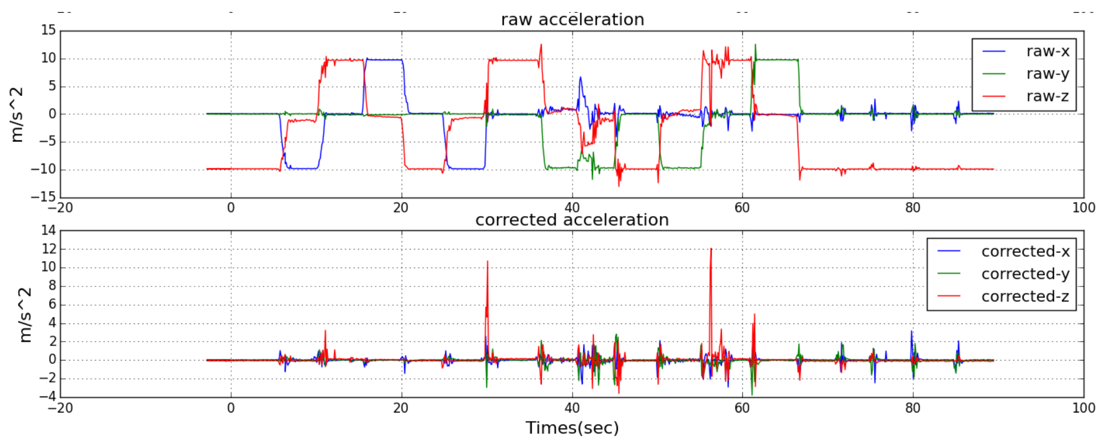


FIGURE 3.4: Raw versus corrected acceleration

susceptible to the problem of gymbal lock, but quaternions are not. I also subtracted a vector representing gravity g $= -9.8k$, to remove the effects of gravity.

For an example of data that has been corrected from the local coordinate system to the global coordinate system see Figure 3.4.

One more step was needed however, to rotate in reference to the initial heading. Because I assume the phone was held flat to begin with I could find the initial heading by using the phone's magnetosensor to get the phone's magnetic heading. Given the

Unfortunately, it was not possible to get the magnetic heading system working properly. The read magnetic heading value was not accurate, and therefore a manual correction was applied afterwards to rotate the coordinates to the GCS. It's not clear why this happened, I suspect the magnetosensor in iPhones is not accurate enough to use just one reading.

Finally, the z-axis acceleration data was filtered using a moving average filter, with the window size W set to 5 on the z-axis. This acted as a low pass filter, as I was seeking signal of frequencies in the low Hertz, and human steps have a period of a few seconds. A moving average filter was chosen as it is the optimal filter for removing random white noise while keeping the optimal step response [16]. I needed good step response because the step response was used in the step detection 3.4.2, to determine when a step was taken, and I wanted to remove random noise from accelerometer error. See Figure 3.5 for an example on a set of steps.

The mathematics are as follows, where $a_z(t)$ is the filtered z-acceleration in the GCS as a function of time, and $a_z^{COR}(t)$ is the unfiltered z-acceleration in the GCS as a function of time. Then

$$a_z(i) = \frac{1}{W} \sum_{j=-\frac{W}{2}}^{\frac{W}{2}} a_z^{COR}(i+j) \tag{3.2}$$

3.4.2 Step Detection

The goal of step detection was to find the points in time where the user took a step. Following the work of Kang and Han [9], I divided this into three steps: finding steps

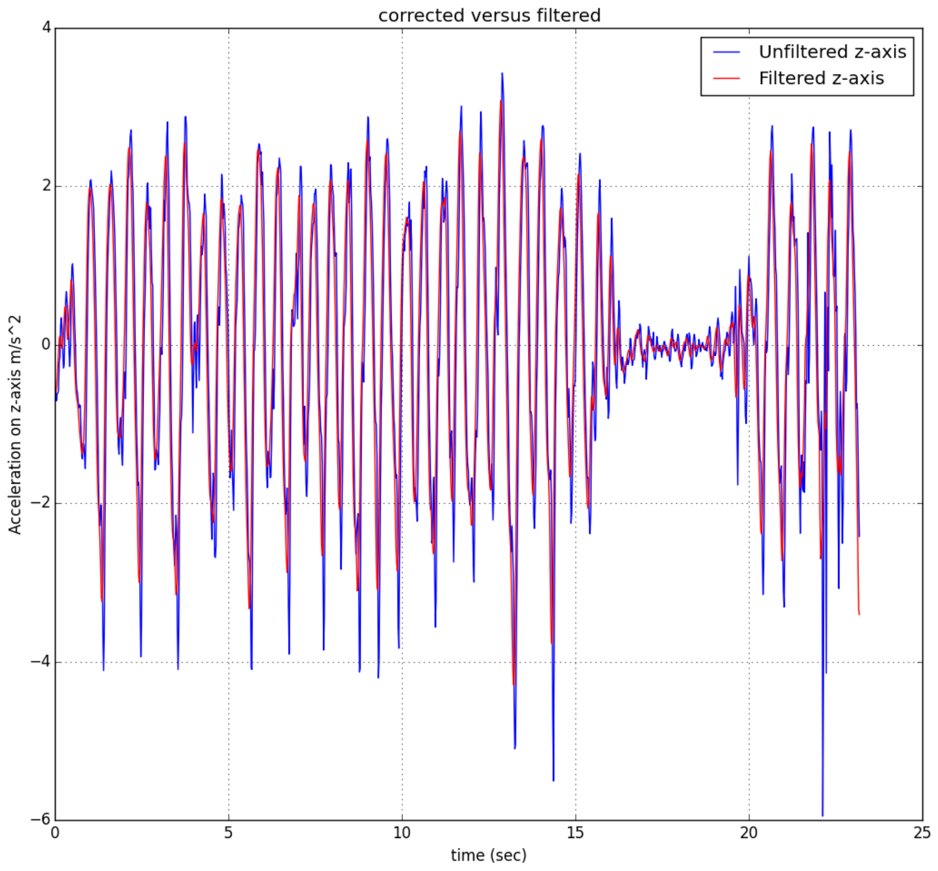


FIGURE 3.5: The raw acceleration data versus the filtered acceleration data

by looking for peaks, finding the step by looking at the peak to peak difference, and finding the step by looking at slopes, and then taking the intersection as the set of steps. In all cases, the idea is similar: a step will appear on the filtered $z - acceleration$ in GCS as a peak and a valley as the foot lifts off (I chose the peak as the step), and I want to find the peaks that represents steps and not just noise. I take the intersection, because each of the methods generates false positives, and fewer false negatives, and therefore the intersection contains fewer false positives. Figure 3.6 shows all three methods and the intersection for a sample walk.

Peak Detection

In peak detection I looked for all peaks, points higher then all other points within the window, and above a threshold $a_{peak} = 1.00ms^{-2}$. That is I looked for the set of points t^{peak} in time, such that

$$t^{peak} = \left\{ t \middle| \, a_z(t) > a_z(t+i), |i| < \frac{W}{2}, i \neq 0 \right\} \tag{3.3}$$

This was mostly correct but sees several false positives when there are smaller peaks next to one peak due to random noise.

Peak to Peak Detection

Here, I looked for all peaks such that the difference between them and all points in the window W is greater than some threshold $a_{pp} = 0.005ms^{-2}$. That is, letting such points be called t^{pp}

$$t^{pp} = \left\{ t \middle| \, |a_z(t) - a_z(t+i)| > a_{pp}, |i| < \frac{W}{2}, i \neq 0 \right\} \tag{3.4}$$

This generated many false positives when the user is not moving, as it sees small peaks from random noise, not drowned out by the movement of the user.

Slope Detection

Another way to think about steps is to look for points where all points leading up to that points are increasing (positive slope) and all points leading away from that point are decreasing (negative) slope within the window. That is I are looking for points t^{slope} where

$$t^{slope} = \left\{ t \middle| \, a_z(t) - a_z(t+k) > 0, a(z_t) - a_z(t+j) < 0, 0 > k > -\frac{W}{2} - 1, 0 < j < \frac{W}{2} + 1 \right\} \tag{3.5}$$

Finally I combine all three methods by taking the intersections. If the final set of times where the algorithm detects a step is t^{step} I set

$$t^{step} = t^{peak} \cap t^{pp} \cap t^{slope} \tag{3.6}$$

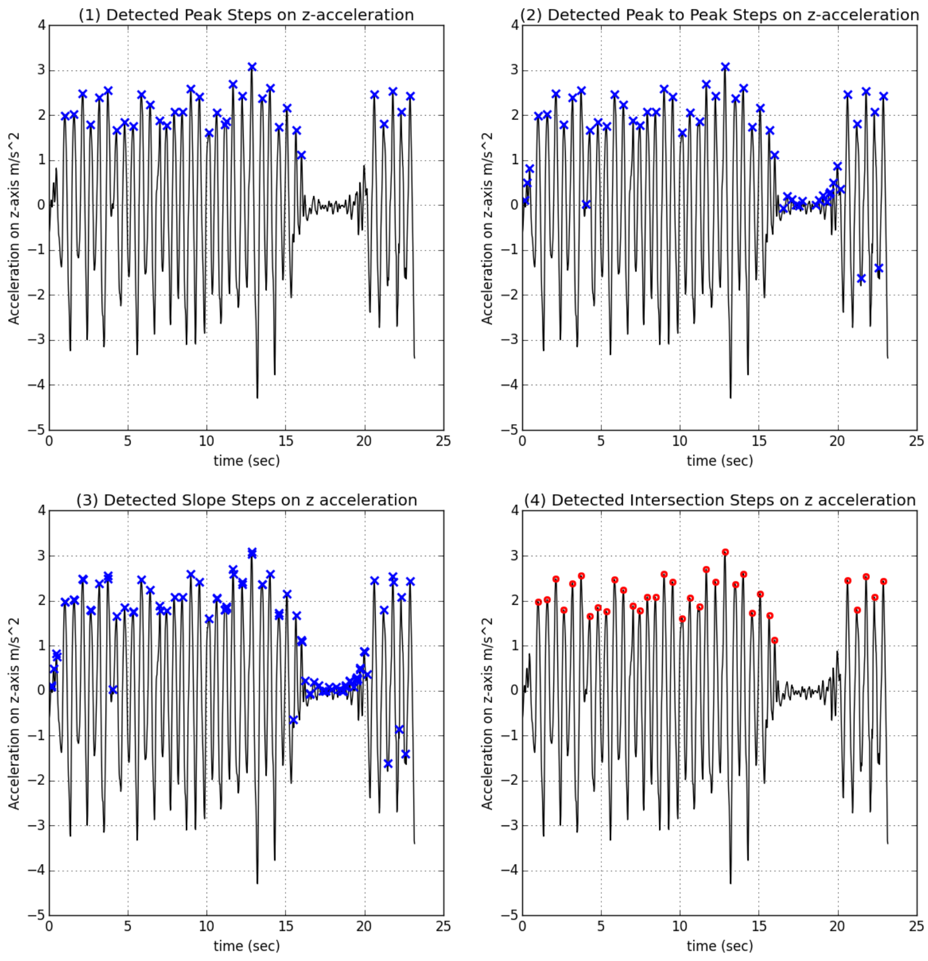


FIGURE 3.6: Step detection in a sample with 34 steps. In all sections, the line is filtered z-acceleration over time, and dots represent detected steps. Blue Xs are from one method, red from the intersection. In method (1) I looked for peaks above a threshold, but notice that this catches an extra double peak at 11 seconds. Then in (2) I looked for steps by seeking points that are above all of their neighbors, and got many false positives in the region from 15 to 20 seconds which are random noise (the example user was standing still at that time). In (3), I look for peaks where the reading has a positive slopes for all points left of it in the window, and a negative slope to all points rightwards in the window, again getting many false positives in the still region from 15 to 20s. In (4) I show the intersection, which correctly identifies the 34 steps in red.

3.4.3 Heading Detection

For heading detection, I used the Core Motion Package's attitude package. To do this, a Core Motion Manager is started, and the Device Motion Updates are started. The starting initial reference frame is `CMAttitudeReferenceFrameXArbitraryZVertical`, which means that the phone is assumed when starting to be facing with the z-axis pointing directly up, as mentioned earlier. Then, at each update, the quaternion form of the phone's attitude is sent to the server. See Section 3.4.1 for details on how this was used to correct for the user's acceleration data.

Then, since the user's initial reference frame had the z-axis pointing up, the user's heading on the 2-d x-y plane was simply the user's Yaw ψ. I found this by transforming the quaternion form to the Euler angle form, and then taking the yaw. The formula for this is as follows. Given a quaternion of the form

$$\mathbf{q} = [q_w\ q_x\ q_y\ q_z]^T = q_w + q_x\hat{\imath} + q_y\hat{\jmath} + q_z\hat{\mathbf{k}} \tag{3.7}$$

, then we can use the `atan2` function, the inverse tangent, to get

$$\psi = atan2(2(q_wq_z + q_xq_y), 1 - 2(q_y^2 + q_z^2)) \tag{3.8}$$

This provides a continuous 60Hz sample of headings. However, I only wanted the heading at each step. Following Kang and Han [9], I used the acceleration valley as the start of each step, and the acceleration peak as the time of each step. To put it another way, I assumed that the user's heading when taking a step is fixed upon the beginning of the step, which occurs at the valley of low acceleration before a step occurs, when the foot pushes off, and then that the step is done at the peak of the step, which corresponds (theoretically) to the middle of the step, as the whole step cycle would see valley-peak-valley. To determine this, we take the t^{step} found in Subsection 3.4.2, and then find the argmin of $a_z(t)$ between this step and the last step (or the beginning of data for the first step) which is taken to be the valley. Mathematically

$$t^{headings}(i) = \text{argmin}_{t^{steps}(i-1)<t<t^{steps}(i)} a_z(t) \tag{3.9}$$

For an example, please see Figure 3.7.

3.4.4 Step Length Detection

The goal of step length detection was to determine at each step, how long each of the user's steps was. Following the Improvements of Kang and Han [9] on Weinberg [17], I did so in the following way.

I start with the assumption that distance between valley acceleration found in Subsection 3.4.3 and the peaks found Subsection 3.4.2for each step, the distance between each peak and valley for a step should correlate with the total length of the step. Weinberg, using a kinesiological argument where the leg is a series of bars on pivots, find that for a given step, $l(i)$, then with an acceleration difference of $a_z^{step}(i)$ defined as $a_z^{peak}(i) - a_z^{valley}(i)$, with constants β and γ, then

$$l(i) = \beta\sqrt[4]{a_z^{step}(i)} + \gamma \tag{3.10}$$

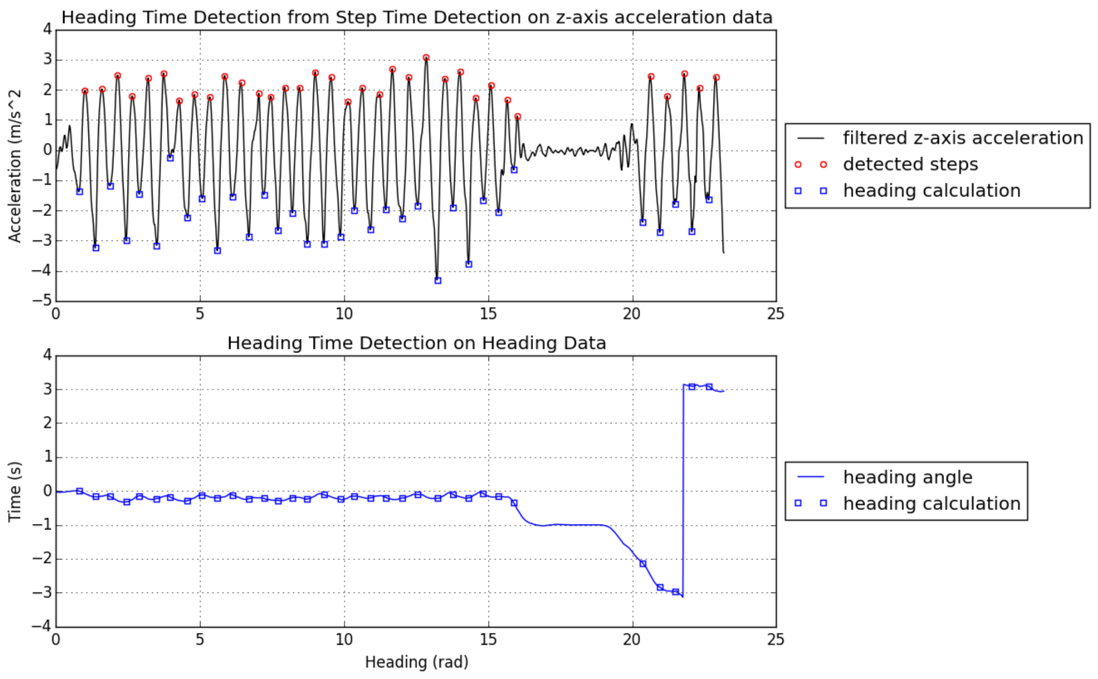


FIGURE 3.7: Heading Detection. The blue squares represent the heading times, determined as the valley below the steps in red circles on z acceleration data in the top graph. Below we show the found heading with heading times overlaid. The sample is for a test where the user walked straight at a heading of $0rad$, and then turned around going the opposite direction at πrad.

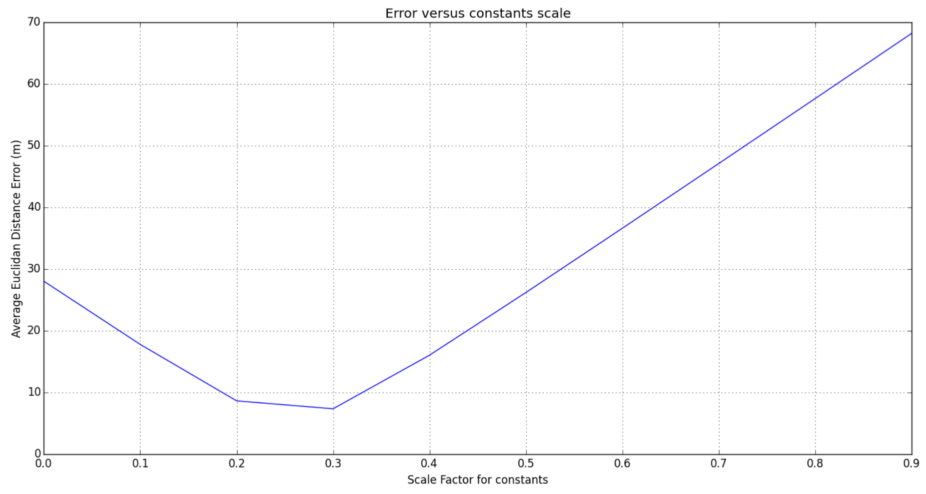


FIGURE 3.8: To minimize error, I found that tweaking the constants for step length discovery was needed.

Kang and Han [9] also used the formula $\beta \log a_z^{step}(i) + \gamma$ and found it was more effective above a threshold $a_\tau^{step} = 3.230ms^{-2}$.

So then

$$l(i) = \begin{cases} \beta \sqrt[4]{a_z^{step}(i)} + \gamma, & \text{if } a_z(i) < a_\tau^{step} \\ \beta \log a_z^{step}(i) + \gamma, & \text{otherwise} \end{cases} \tag{3.11}$$

The values for β, γ were found by using the ratio given in [9], and then minimizing over the average error in several runs. Kang and Han provided a set of constant values, however when using them I got wildly inaccurate results. In playing with it, I found that the ratio $\frac{\beta}{\gamma}$ was important, and varying away from the values provided in [9] resulted in a high error. I denoted the constant factor the ratio $\frac{\beta}{\gamma}$ was multiplied by as the 'scale' σ. The values that minimized average error was found to be $\beta = 0.38454 \sqrt[\frac{3}{4}]{m}\sqrt{s}^{-1}$ and $\gamma = -0.32734m$ when run over a set of sample results. An example error curve is presented in Figure 3.8. In the tests performed, I found $\sigma = 0.4$ Created thebest results.

3.4.5 Finding Location

The algorithm for finding the user's location is as follows: At each detected step i, the user's position $\binom{x}{y}$ is updated by finding the heading $\psi(i)$ (see 3.4.3) and the step length $l(i)$ (see 3.4.4), we combine this projecting the step length vector onto the heading, and adding this to the previous x and y value.

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} \cos\psi(i) \cdot l(i) \\ \sin\psi(i) \cdot l(i) \end{bmatrix} + \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \tag{3.12}$$

3.4.6 Bluetooth Correction

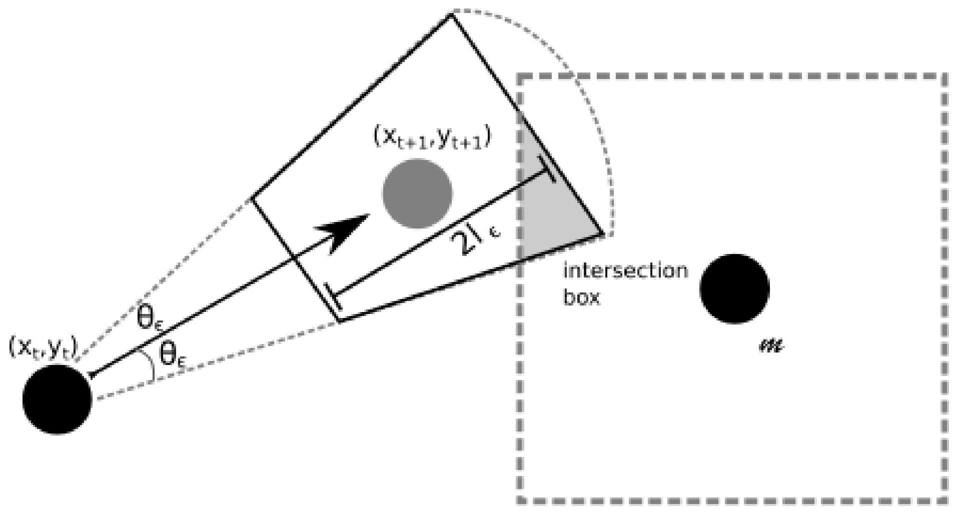


FIGURE 3.9: This figure illustrates how Monte Carlo Boxed Localization (MCBL) would be performed in the app. At time $t + 1$, from an original location (x_t, y_t) the app calculates the next location (x_{t+1}, y_{t+1}) . Then a bounding box is drawn around the new location defined by the heading error θ_ϵ and a step error l_ϵ. Note how in the diagram this bounding box is truncated to a trapezoid for easier calculating. This is then intersected with the range of a nearby other device m that is connected via Bluetooth to the device. Again, the range is presented as a box for easier calculation. Points are then drawn from the intersection and the weighted average taken as the location. In practice, there would be many more connections.

The idea of Monte Carlo Boxed Localization as presented in Baggio and Langendoen [12] is simple in concept.

In each time interval, generate a set of samples around the user's predicted location. Filter those samples by removing all of those out of the range of the other Bluetooth connected devices. Then take the weighted average of the points as the new location.

I propose to use this, since Bluetooth has a maximum range, and the app connects to other users of the app via Bluetooth.

For a figure explaining this algorithm see 3.9. What makes this 'boxed' is that the filtering is done before hand, and points are drawn from the intersection to start instead of filtering. In Baggio and Langendoen the initial box is drawn on the previous point (x_t, y_t) from some calculated $v_{max}\delta t$ which is the maximum velocity of a pedestrian times the change in time since the last calculation.

This expands on Baggio and Langendoen's work because in the proposed algorithm, we know significantly more about the movement of the user. Here we can draw the initial box (really a trapezoid) around the predicted location from the app described in the previous sections. Then assuming that there is some maximum error to the heading $2\theta_\epsilon$ and some maximum step length error l_ϵ, our box can be defined as the trapezoidal approximation of the conic section defined as

$$\begin{bmatrix} \cos(\psi \pm \theta_\epsilon) \cdot (l \pm l_\epsilon) \\ \sin(\psi \pm \theta_\epsilon)\,(l \pm l_\epsilon) \end{bmatrix} + \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \tag{3.13}$$

As seen in figure 3.9. We intersect this with the box of the other connected devices, in the figure just one m is shown, and draw samples p_i from the intersection $(x_{t+1}, y_{t+1}) = \sum_i p_i$

Unfortunately, enough data was not collected to test this algorithm, and so it was not implemented.

Chapter 4

Results

4.1 Experimental Design

To test the app, the following experiment was conducted. ‘Landmarks’ consisting of a piece of paper printed with a lowercase letter from ‘a’ to ‘v’ were placed at 1.5m on the walls of the second floor of the Davis science building at Colby College. The Davis science building second floor is a loop of approximately 40m by 20m. The placement of the landmarks is in Figure 4.1. Three volunteers were recruited and provided an iPhone to which the app was installed. They then performed three trials where they walked on the second floor of Davis for 180sec, marking on the app each time they passed a landmark by sending the associated letter.

The data was then sent to a computer running the server over a WiFi network for data analysis.

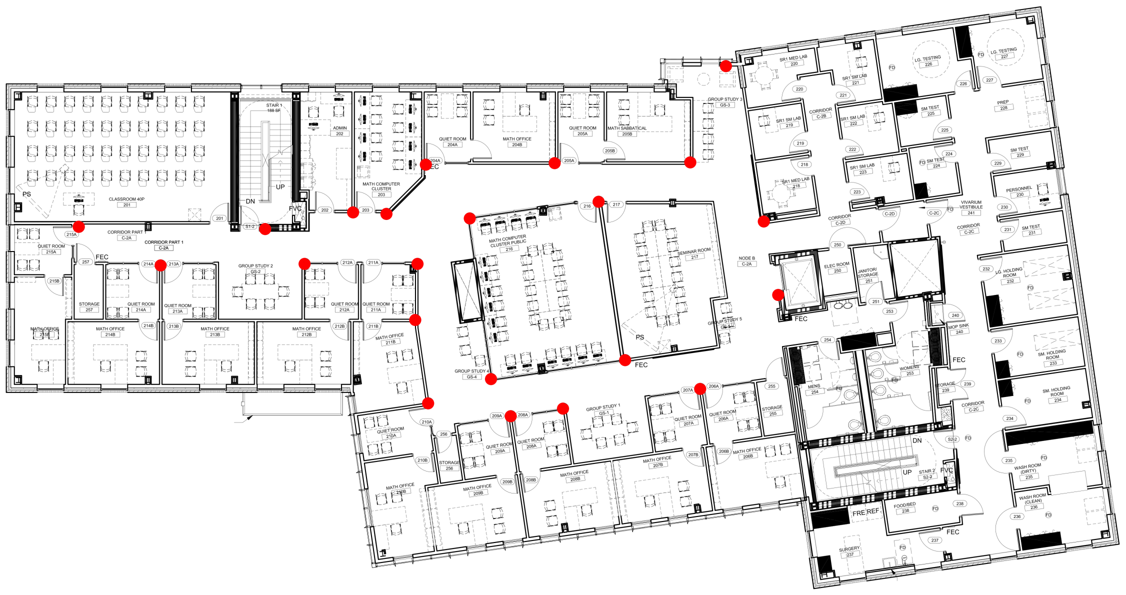
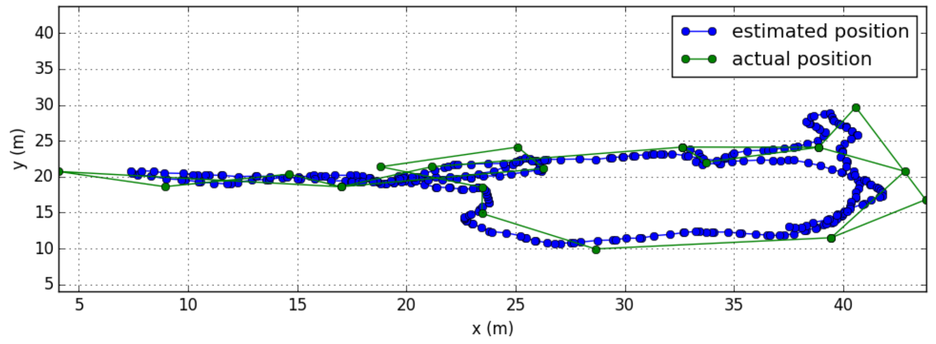


FIGURE 4.1: Placement of landmarks on the second floor of the Davis Science Building in colby College. Each Red dots represent one landmark. Each landmark was a small piece of paper with a printed letter on it. Upon reaching a landmark users sent their fixed landmark location as a check on the app’s accuracy.

The three figures were of varying height, with leg lengths of $90cm, 97cm, 120cm$ each. They did not start in the same position, but at any landmark they chose. However, they had to indicate before the app started which landmark they started, and for the experiment to work they had to beholding their phone flat, as explained in Subsection 3.4.3. As we will see, this proved problematic There was no fixed direction of travel, they were allowed to walk as they liked as long as they stayed on the second floor of the Davis science building.

The landmarks were then used to compare the app's position calculations to the ground truth. Since at the point when the user sent in the landmark, I knew their real location at that time. The analysis of the collected data is in Section 4.2.

4.2 Collected Data



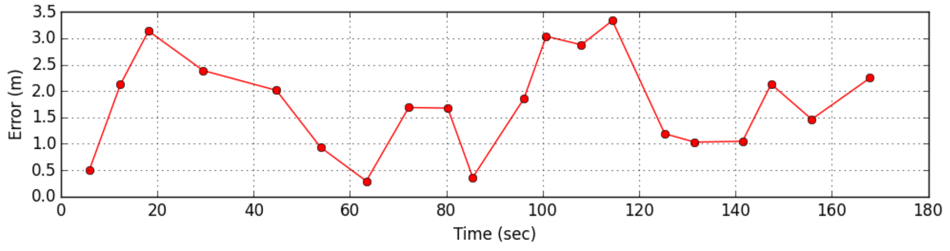


FIGURE 4.2: One example run, where the subject walked around the floor of Colby's Davis building twice. The top section is the user's calculated location (blue) compared to the known location of the user from the landmarks (green). The bottom is the euclidean distance error at each landmark reading in red. This never goes beyond 3m, and has no obvious pattern over the 3m. The user's starting landmark was the landmark at 26m on the x axis. Note that the x and y axes in this plot are completely arbitrary, chosen just to make the picture easiest to see, and in the creation of the graph a rotation was performed to get these results.

As explained in Chapter 3, the position of the user was calculated by the app. We can then compare this to the actual location of the volunteer using the landmarks. See Figure 4.2 for an example of one such run. An animated version of another trial ca be seen at

The 'scale' σ used in the step finding constants was 0.4 as found in the section denoted 3.8 in Chapter 3.

Error was determined by finding the euclidean distance $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ between the calculated location and the known landmark location.

Notice that there is no obvious pattern to the error as over the course of the 3 minutes. Usually in inertial navigation systems error accumulates over time as any error in the measured acceleration propagates to the velocity and then the location. Therefore the error over time increases often very drastically. Interestingly in pedestrian navigation systems that is often not the case, including in this example.

Of the collected data, only three runs were usable all from the same volunteer. This is because the instructions were not well explained to the other volunteers, and so they did not hold their phone flat when they began . This rendered their data useless as it meant that all of the data was rotated incorrectly to an incorrect global coordinate system.

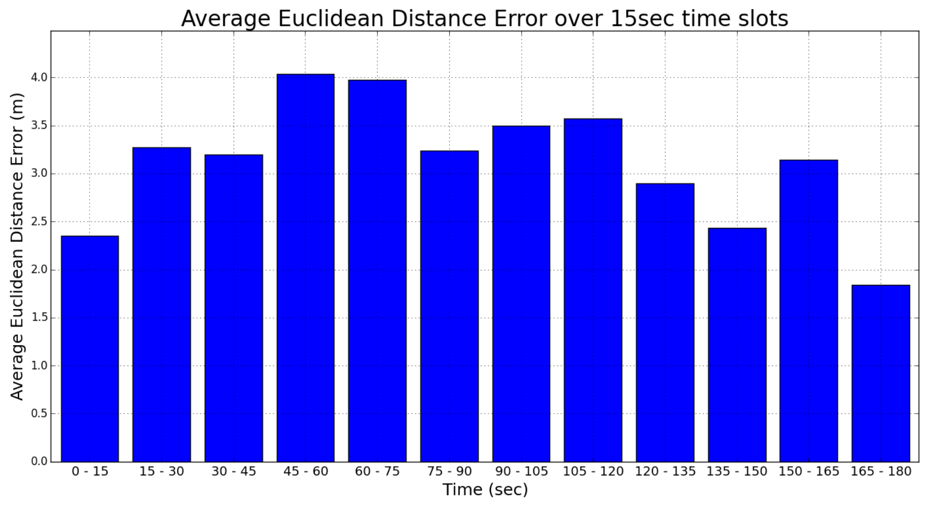


FIGURE 4.3: Taken from one subject walking three times, for a total of 142 data points. Each data point consisted of a time and an error from a known position at that time. Data points were binned in 15sec long chunks. Each run lasted 3 minutes in length. Error showd no consistent patter and varied between 0m and 6m over all three runs.

The collected data of the three usable runs was then used to form Figre 4.3 as a summary results. All of the known time and euclidean error pairs from all three experiments were binned in 15sec increments from 0sec to 180sec. The average of each point was then plotted in Figure 4.3. The average error does not go higher then 4.0m, and again there is no obvious pattern, even decreasing over time.

The maximum error experienced in any run was $7.1m$, the minimum was $0.29m$. The average was $3.19m$, with standard error $1.65m$.

The main barrier to performing more trials turned out to be the fact that Apple Free Developer's accounts have a maximum of 5 devices that can have an app loaded onto them without using the app store. This can be lifted to 100 devices for an annual subscription of $\$100$ which I did not have the funds to pay. So I was limited on the number of devices I could use, which prevented testing the MCBL algorithm.

4.2.1 Bluetooth Results

Although the proposed algorithm was not implemented, since only one of the volunteers data was usable, all three devices did connect and disconnect correctly and record this. This suggests that in a real test with the algorithm and more participants, the app is usable. Due to the small scale of the Davis Science Building most users were connected to the other two devices at most times, but did disconnect when behind walls or other obstacles.

Chapter 5

Conclusion

In Kang and Han [9], average localization error is kept below 2m with trials of 120sec. This is slightly better in performance than the implementation of their algorithm contained in this paper, where the average error was below 4.0m. They do not provide maximums, so we cannot compare all aspects of the project. The maximum average value in my study also occurred before 120sec, so it is not due to the increased length of the trials in this study. The patterns used by the volunteers were also more complex than in Kang and Han, who had all subjects walk in a fixed loop.

The only major difference between Kang and Han's algorithm and the one implemented here is that they use a more complex algorithm for heading detection. Kanga and Hang directly use the magnetosensor and gyroscope to determined heading by diving the readings into four different cases and calculating each case separately. In this app, Apple's provided API was used. Kang and Han demonstrate that other methods of heading detection significantly increase the error, and so this is the likely source. However, Apple's built in platform still performed relatively well and better than just using the gyroscope or magnetosensor data from Kang and Han [9] if you compare the heading error data they present using just magnetosensor or gyroscope to the results found in this paper.

Since the phones are different in the two studies (Kang and Han usa Samsung Galaxy Notes, generation I and II) it's possible the difference comes from the hardware. However the Samsung Galaxy Note uses the STMicroelectronics LSM330DLC for it's 6 axis accelerometer and gyroscope [18], which has $0.03dps/\sqrt{Hz}$ for rate noise density, a measurement of the precision of a gyroscope [19]. In comparison, the iPhone 6's MPU-6700 six-axis accelerometer has a rate noise density of $0.01dps/\sqrt{Hz}$ [20], so it is more accurate. This makes sense as it is a few years younger. So it is unlikely to be hardware error that cased the worse performance.

Broadly, the accurracy we see in Kang and Han is typical of methods in the last several years [7, 8, 11]. Thus the platform built in this project is near the best performance, and improvements in heading detection could likely make the algorithm competitive.

The proposed wireless algorithm has not been tried before in a PDR system, see Chapter 2. It also contains improvements over the algorithm proposed by Baggio and Langendoen [12]. Using the new information provided by the PDR system beyond a hypothetical maximum velocity. Since Bluetooth is already a ubiquitous element of smart phones it could be widely used.

In total, this thesis implemented a PDR system as a base for further experimentation, proposed a novel wireless localization algorithm using PRD information, and a small experiment was performed to check the accuracy of the system. Through step detection, heading detection, and step length detection it is possible to construct a relatively good localization system on commercial phones, with accuracy within single digit meters.

5.1 Future Work

The work that remains is in two parts: to implement the proposed Bluetooth algorithm, and to perform larger tests.

The proposed wireless localization algorithm is certainly computationally feasible and would present an advance over current systems. using the PDR information we can get a much better ideas as to the likely location on the next step, thereby making our initial sample space more accurate. The main block to implementing this feature was the lack of data to use, which brings up the next problem.

The fact that two of the three volunteers data was unusable severely limited the ability to test the algorithm. This was purely the fault of experimental design and a lack of clear communication and could easily be removed, although it does suggest the brittleness of the app when it comes to initial conditions. Similarly, Apple's limitation to 5 devices could be alleviated with a little more funding. The original experiment design called for five volunteers for the initial tests shown in Chapter 4. With 5 participants, we would see more connections and disconnections, and would hope to see that as the number of participants increased, there would be better accuracy.

A larger experiment may also need a larger space. There are a few challenges in this instance, the largest one being that if the space is too open then Bluetooth devices connect too easily to one another, making the test not as illuminating. All devices connected to one another is not a very realistic scenario. Therefore a larger indoor space with corridors and walls is preferable. An experiment with 10 - 20 participants would probably be sufficient. This may also require a better system for landmarks, as landmarking a larger space could prove difficult.

Finally, another avenue of possible work would be to take the current algorithm on post-processing side and move it to the app. The use of the server was purely to make data collection easier. A purely app based algorithm would be more realistic and move this project closer to being usable in the real world, in order to help localization in situations where access to GPS, WiFi, and Cellular networks are limited.

Bibliography

[1] *Android Location and Maps API*. https://developer.android.com/guide/topics/location/index.html. Accessed: 2017-04-10.

[2] *CoreLocation API*. https://developer.apple.com/reference/corelocation. Accessed: 2017-03-24.

[3] Robert Harle. "A survey of indoor inertial positioning systems for pedestrians". In: *IEEE Communications Surveys & Tutorials* 15.3 (2013), pp. 1281–1293.

[4] Oliver J Woodman. "An introduction to inertial navigation". In: *University of Cambridge, Computer Laboratory, Tech. Rep. UCAMCL-TR-696* 14 (2007), p. 15.

[5] I. Constandache, R. R. Choudhury, and I. Rhee. "Towards Mobile Phone Localization without War-Driving". In: *INFOCOM, 2010 Proceedings IEEE*. 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5462058.

[6] Anshul Rai et al. "Zee: zero-effort crowdsourcing for indoor localization". In: *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM. 2012, pp. 293–304.

[7] Zhice Yang, Xiaojun Feng, and Qian Zhang. "Adometer: Push the limit of pedestrian indoor localization through cooperation". In: *IEEE Transactions on Mobile Computing* 13.11 (2014), pp. 2473–2483.

[8] Jhen-Kai Liao, Kai-Wei Chiang, and Zhi-Ming Zhou. "The Performance Analysis of Smartphone-Based Pedestrian Dead Reckoning and Wireless Locating Technology for Indoor Navigation Application". In: *Inventions* 1.4 (2016), p. 25.

[9] Wonho Kang and Youngnam Han. "SmartPDR: Smartphone-based pedestrian dead reckoning for indoor localization". In: *IEEE Sensors journal* 15.5 (2015), pp. 2906–2916.

[10] Silke Feldmann et al. "An Indoor Bluetooth-Based Positioning System: Concept, Implementation and Experimental Evaluation." In: *International Conference on Wireless Networks*. 2003, pp. 109–113.

[11] Piotr Mirowski et al. "SignalSLAM: Simultaneous localization and mapping with mixed WiFi, Bluetooth, LTE and magnetic signals". In: *Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on*. IEEE. 2013, pp. 1–10.

[12] Aline Baggio and Koen Langendoen. "Monte Carlo localization for mobile wireless sensor networks". In: *Ad hoc networks* 6.5 (2008), pp. 718–733.

[13] Lingxuan Hu and David Evans. "Localization for mobile sensor networks". In: *Proceedings of the 10th annual international conference on Mobile computing and networking*. ACM. 2004, pp. 45–57.

[14] He Wang et al. "No need to war-drive: unsupervised indoor localization". In: *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM. 2012, pp. 197–210.

[15] *UIAcceleration API*. https://developer.apple.com/library/content/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/GettingRotationalInformationfromtheGyroscopes.html. Accessed: 2017-03-24.

[16] Steven W Smith et al. "The scientist and engineer's guide to digital signal processing". In: (1997).

[17] Harvey Weinberg. "Using the ADXL202 in pedometer and personal navigation applications". In: *Analog Devices AN-602 application note* 2.2 (2002), pp. 1–6.

[18] *Samsung Galaxy Note Accelerometer*. https://www.ifixit.com/Answers/View/133525/Accelerometer-gyroscope+-+easy+to+fix. Accessed: 2017-05-12.

[19] *LSM330DLC iNEMO inertial module:3D accelerometer and 3D gyroscope*. http://www.st.com/content/ccc/resource/technical/document/datasheet/bd/61/af/53/b5/f5/4d/7b/DM00037200.pdf/files/DM00037200.pdf/jcr:content/translations/en.DM00037200.pdf. Accessed: 2017-05-12.

[20] *InvenSense MPU-6500*. https://www.invensense.com/products/motion-tracking/6-axis/mpu-6500/. Accessed: 2017-05-12.