2014

Mobile Indoor Positioning for Augmented Reality Systems

Robert B. Glass
Virginia Commonwealth University, glassrb@vcu.edu

Mobile Indoor Positioning for Augmented Reality Systems

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

By

Robert B Glass
Bachelor of Science, Virginia Commonwealth University, United States, 2012

Director: Dr. Vojislav Kecman
Professor, Department of Computer Science

Committee Members
Dr. Vojislav Kecman
Dr. Tom Arodz
Professor Pamela Turner

Virginia Commonwealth University
Richmond, Virginia
December, 2014

Acknowledgement

Foremost, I would like to express my sincere gratitude to my advisor Prof. Vojislav Kecman for the continuous support of my M.S. study and research. His guidance helped me immensely in research and writing this thesis. I would also like to thank the rest of my thesis committee: Prof. Tom Arodz, and Prof. Pam Turner. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the thesis work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues.

Table of Contents

List of Tables

List of Figures

Abbreviations

AR – Augment Reality

RW – Real World

IPS – Indoor Positioning System

IMU – Inertial Measurement Unit

MEMS – Micro electromechanical Systems

TOA – Time of Arrival

TDOA – Time Difference of Arrival

AOA – Angle of Arrival

RSS – Received Signal Strength

RSSI – Received Signal Strength Indication

ADB – Android Debug Bridge

APK – Android Package Installer File

IW – Linux Wireless

ORM – Object Relational Mapping

QR Code – Quick Response Code

IPS – Indoor Positioning System

INS – Inertia Navigation System

HDMI – High-Definition Multimedia Interface

USB – Universal Serial Controller

OS – Operating System

FSPL – Free Space Path Loss

Abstract

MOBILE INDOOR POSISTIONING FOR AUGMENTED REALITY SYSTEMS

By Robert B. Glass, BS.

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

Virginia Commonwealth University, 2014.

Major Director: Dr. Vojislav Kecman,
Professor, Department of Computer Science

This thesis explores the creation and setup of a prototype that allows users of the device to interact within an indoor real world environment and a virtual environment simultaneously using high-tech common technology. The prototype is comprised of a small mobile device such as a cellular mobile phone, Raspberry Pi computer, a battery powered handheld Pico projector, and software developed for the Android OS. The software can easily be ported to other mobile and non-mobile operating systems. The mobile device must contain accelerometer, magnetometer, and gyroscope embedded sensors as well as 802.11 wireless network chip. The prototype software implements an indoor positioning system to track the current location and orientation of the prototype device in real time. It also displays a virtual world projection upon the surfaces of the real world in relation to the prototype's physical location and orientation.

Three different orientation estimation methods were tested and compared in this thesis. Accelerometer and magnetometer based method, gyroscope based method, and a combined method using a technique called sensor fusion were implemented. A

multilateration approach was used for location estimation. Location estimates were calculated from the measured received signal strength of multiple 802.11 wireless network access points. The location of all wireless access points were known and fixed. Received signal strength data was converted to meters using a log distance propagation model, and tests were conducted to compare actual distance with converted distance. Tests were also conducted to compare multilateration estimates from unfiltered or raw *RSS* and filtered *RSS* data using a Kalman filter.

1 An Introduction to Augmented Reality and Indoor Navigation

1.1 Introduction

Augmented reality is becoming more significant and common as computing devices become smaller and faster. Smartphones and other compact mobile devices are containing more sensors that can interpret real world data such as light, proximity, acceleration, magnetic field, temperature, humidity, and many other physical or semi-physical elements. Mobile hardware can track the world around us as well as the characteristics of a user interacting with the hardware. Mobile technology as such can be used to further enhance augmented reality experiences and help augmented reality

based applications exists in areas that in the past have not been feasible.

The experiments in this thesis focus on the creation of prototype that incorporates a mobile device and is capable of tracking its own physical orientation and location. The prototype device is held in the hand of a user, and projects a virtual environment from a forward facing projector upon the objects and surfaces in an indoor environment. Orientation and location information from the prototype device is used to determine the virtual orientation and location of a user within a virtual environment. The user is able to explore and interact with the real world (*RW*) augmented or overlaid with a virtual world. The combining these two elements creates augmented reality experience for the user.

1.2 Motivation and Goals

The motivation of this work is an immersive multi-user environment comprised of real and virtual objects. The want is to create an augmented reality experience allowing users to explore an indoor real world environment while simultaneously exploring virtually created environment. The virtual environment is presented using a projector embedded in a handheld device. The device acts as a virtual reality flashlight and allows users to uncover a virtual world or peel away the real world. This also enables users to physically move around in the real world environment which is not as easily possible when using a display such as a computer monitor, VR headset, or screen of a mobile device for an augmented reality application.

The goal of this work is to create a prototype device that allows users to interact with both the real world and virtual world using high-tech common mobile or portable devices that anyone can acquire at a relatively affordable cost. The prototype must be able to estimate the physical orientation and location of the prototype within an indoor environment relative to a global reference. The experiments in this thesis use an HTC Sensation 4G smartphone, Raspberry Pi computer, and a Pico handheld battery powered projector. The choice of this hardware was due to the compact nature of each device and the sensors embedded in the smartphone. All three devices were used in the construction of a compact prototype device that should be easily and comfortable held in a user's hand.

The goal of this work is not to create a completed augmented reality game for multiple users, but to create the technology and prototype for such an augmented reality game. This is the first step to be able to create a system that allows multiple users to interact within the real world and virtual world. Another goal of these experiments was to test the functionality and feasibility of such a system.

The ideal maximal variance between the prototype's physical or real location and the estimated location is one meter. The ideal maximum variance between the prototype’s physical or real orientation and the estimated orientation comprised of azimuth, pitch, and roll is one degree for each axis. Figure 1 explains the definition of azimuth, pitch, and roll.

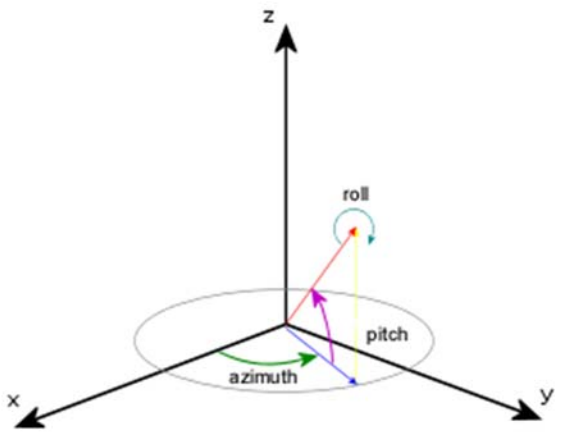


Figure 1.1: Definition of azimuth, pitch, and roll

1.3 Augment Reality

Augmented Reality (*AR*) implementations have been defined by many researchers. One definition of augmented reality is a technology which allows computer generate virtual imagery to exactly overlay physical objects in real time [1]. Another definition states augmented reality consists of a combination of three traits: (1) combines real and virtual worlds, (2) is interactive in real time, and (3) registers in 3-D [2]. Augmented reality has also been defined as the emergence of the real world with superimposed virtual images, combining the advantage of both real and virtual environments [3]. Augmented Reality also refers to a wide spectrum of technologies that project computer generated materials, such as text, images, and video, onto users' perceptions of the real world [4].

The history of augmented reality started in the late 1950s when a simulator called

“Sensorama” was developed by Morton Heilig [4], but it wasn't until the 90s that inertia became significant, and the numbers of researchers and developers in the *AR* field increased. Today there are a growing number of companies that are researching and developing applications and devices for a multitude of applications involving augmented reality. This is primarily due to new technology such as small mobile computers and devices which come in many forms. Many fields and technologies have incorporated augmented reality to aid in various tasks such as training of personnel, to enhance visualizations, to help in repairs of equipment, to display vital information for navigation, and many other uses.

1.4 Indoor Navigation

An Indoor positioning system (*IPS*) or indoor navigation system can be defined as any system that provides a precise position inside of a closed structure, such as a shopping mall, hospitals, airport, a subway, and university campuses [5]. Many different types of indoor positioning systems exist and most indoor positioning systems operate by estimating distances to known fixed points using radio based signals although there are some indoor positioning systems that do not. Refer to section 1.4.2 for an example of a positioning system that does not use radio signals.

Different types of location information also exist and are required for different types of applications. Location types range from physical location, symbolic location, absolute

location, and relative location [6]. The experiments in this thesis focus on physical location and is expressed in the form of coordinates, which identify a point on a 2-D/3-D map [7]. Below is a quick overview of different technologies that exist today and are used in various indoor positioning applications.

1.4.1 RFID

RFID stands for radio frequency identification. RFID tags are small devices made up of an antenna and chip. Many manufacturing processes incorporate RFID tags to enable product tracking along assembly lines and have also been used in warehouse storage to easily locate and verify inventory. Two types of RFID tags exists and are called passive and active tags. Active RFID tags use an internal power source to actively power the device while passive RFID tags rely on RFID readers for power [8]. Active RFID tags have a much higher range than passive tags, and are also more expensive. The experiments in this thesis do not focus on using RFID technology for indoor positioning because most modern day mobile devices are not equipped with the proper hardware to transmit or receive RFID signals.

1.4.2 Inertial Navigation

An inertial navigation is a self-contained navigation technique in which measurements provided by accelerometers and gyroscopes are used to track the position and orientation of an object relative to a known starting point, orientation and velocity [9]. Inertial navigation is commonly used in aircraft. Inertial measurement units (*IMUs*) measure angular velocity and linear acceleration to track position and orientation. Microelectromechanical Systems (*MEMS*) technology, such as sensors contained in mobile devices, allow many modern day devices to be capable of being used as inertial navigation systems. Inertial navigation systems work best when adjusted independently for characteristics for each user. For example, two individuals will most likely have different walking stride lengths. The experiments in this thesis do not focus on using inertial navigation.

1.4.3 WIFI (802.11) and Bluetooth

Indoor positioning systems using wireless 802.11 and Bluetooth networks function by estimating the distances between the current location of a user and wireless network access points. Currently, there are many methods of estimation such as Time of Arrival (*TOA*), Time Difference of Arrival (*TDOA*), Angle of Arrival (*AOA*), and Received Signal Strength (*RSS*) [10]. Time based techniques require recorded time stamps at either the time of sending or receiving packets and control packets such as RTS, CTS, and ACK

[11]. Not all 802.11 wireless router manufacturers produce hardware meeting such requirements. RSS based techniques estimate the distance between multiple 802.11 wireless access points using the difference between the transmitted and received signal strengths. Trilateration is then used to estimate the location of a user. The experiments in this thesis focus on RSS based location estimation.

2 Prototype Setup

2.1 Introduction

The hardware and software used to build the prototype are explained in the following sections. Prototype construction is also explained. Section 2.2 details prototype hardware setup. Hardware specifications for the equipment used in the experiments are listed in section 2.3. Software components, Android device setup and modification, and explanation of libraries and SDKs used in the prototype software are in sections 2.4, 2.5, and 2.6, respectively.

2.2 Hardware Setup

The prototype was created from commonly available technology and therefore can be acquired easily and reconstructed at a fairly low cost. The main components of the hardware consisted of an HTC Sensation 4g smartphone with Android 4.4.4 operating system, a Pico hand-held battery powered projector, and a Raspberry Pi running Debian Wheezy. Firmware on all the routers was flashed with the latest version of DD-WRT.

The HTC Sensation 4g smartphone was connected to the Raspberry Pi through the micro USB port on the smartphone to a USB port of the Raspberry Pi. The Raspberry Pi was connected to the Pico projector using the HDMI ports on both devices. Figure 2.1 shows how the smartphone, Raspberry Pi, and projector were connected. The wireless access points were placed in separate corners in an indoor space.

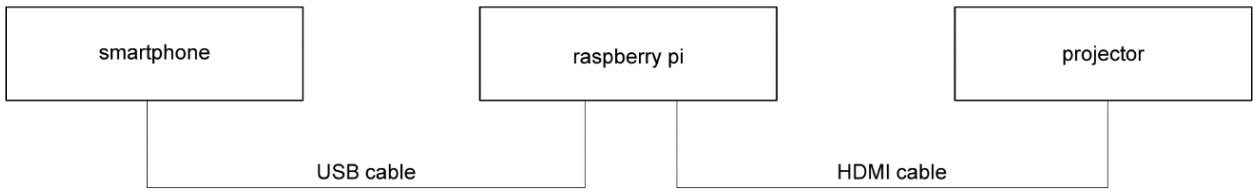


Figure 2.1: Overview of system component setup

2.3 Hardware Specifications

Below is the main hardware component specifications used in the prototype. Table 2.1 lists specifications for the projector used in the prototype. The smartphone device specifications is listed in 2.2. The client machine and wireless access point specifications are listing in tables 2.3 and 2.4.

Pico Projector	
Description:	Pico Projector
Native Resolution:	800 x 600
Aspect Ratio:	4:3
Rated Brightness:	32 ANSI lumens
Supported Video Formats:	480i

Table 2.1: Hardware specifications for projector

Smartphone	
Description:	HTC Sensation 4G with Beats Audio
CPU:	1.8 GHz ARMv7 Processor revision 2
Operating System:	Android 4.4.4 (KitKat)
Sensors:	accelerometer, magnetometer, gyroscope, proximity, light

Table 2.2: Hardware specifications for smartphone

Client Machine	
Description:	Raspberry Pi version B
CPU:	700 MHz ARM Processor
RAM:	512 MB
Operating System:	Raspian Wheezy

Table 2.3: Hardware specifications for client machine

Wireless Access Points	
Description:	Linksys WAP54G
Standards:	802.11g, 802.11b, 802.3, 802.3u
Transmit Power:	22mw
Firmware:	DD-WRT

Table 2.4: Hardware specifications for wireless access points

2.4 Software Setup

The system for the prototype is comprised of two main components. The first is an Android application that is deployed on a smartphone to estimate the location and orientation. Location is calculated using the wireless network interfaces integrated on the smartphone and orientation is calculated using the sensors on the smartphone. The application also starts a server for Android Debug Bridge (*ADB*) connections.

The second software component is a program written in Java and runs on a Raspberry Pi. The two main function of the Java program is open a client connection the Android application to receive location and orientation from the Android application and send visual information to visual output display device. After the connection is completed the Android application transmits the calculated orientation and location data to the client machine. This program also uses OpenGL to display a virtual world through the projector that is controlled by the location and orientation of the smartphone.

2.5 Android Device Setup

The following section details the steps required to prepare a device running the Android operating system. A device will be capable of running the prototype software after setup. It is assumed that the android device has an SD card installed, the device is root access capable, and is running a compatible Android operating system version (Ice Cream Sandwich, Jellybean, Kitkat, and possible future releases). There are various pre-rooted custom ROMs available for many Android powered devices available for download. Many resources are also available documenting how to setup custom ROMs on mobile devices, but is beyond the scope of this thesis.

2.5.1 Steps to Prepare an Android Device

1. Enable Android debugging or USB debugging in Android settings. If Android neither option exists then go to *Settings >> About phone >> Build Number*. Tap on build number 7 times to unlock developer options. After this Android debugging or USB debugging will appear in Android settings.

2. Compile the Linux Wireless package tool *iw* for Android available from http://wireless.kernel.org/en/users/Documentation/iw. This thesis does not detail how to compile Linux Wireless for Android. Various resources can be found on the internet or a pre-built version can be used if available. A pre-built version of *iw* has been distributed with this thesis for ease of use. It has been

tested and confirmed working on Ice Cream Sandwich, Jellybean, and Kitkat Android operating system versions.

3. Use a micro to type-A USB cable to connect the Android device to a computer.

The following steps outlined below modifies files on the internal storage of the Android operating system. Proceed with caution.

4. Download and install Android Debug Bridge (*adb*), a command line tool used to communicate with Android-powered devices [1]. More information can be found at http://developer.android.com/tools/help/adb.html.

5. On some versions of Android, root access for *adb* must be enabled. On the Android device go to *Settings >> Developer Options >> Root Access* and change to *Apps and ADB.*

6. Change to the directory where the *adb* executable resides. The following commands must be executed in the same directory as *adb*. Individual commands are labeled with lowercase letters.

7. Transfer *iw* to SD card in Android device.

   a. `adb push ./iw /mnt/sdcard/iw`

8. Start a shell instance on the mobile device using the Android Debug Bridge.

    b. `adb shell`

9. Request root access on the Android device.

    c. Su

10. Find the system partition on the Android device.

    d. `mount | grep system`

This command will produce output similar to the following:

`/dev/block/platform/msm_sdcc.1/by-name/system /system ext4 ro, seclabel, relatime, user_xattr, barrier=1, data=ordered, noauto_da_alloc 0 0`

The important part of the output is `/dev/block/platform/msm_sdcc.1/by-name/system`. The output will vary slightly from device to device, and is used in the next step (the underlined text).

11. Remount the system partition with read and write access.

    e. `mount –o rw,remount`
    `/dev/block/platform/msm_sdcc.1/by-name/system`
    `/system`

12. Transfer *iw* from the SD card to the system partition.

    f. `cp /mnt/sdcard/iw /system/xbin/iw`

13. Remount the system partition in read only mode.

    g. `mount –o ro,remount`
    `/dev/block/platform/msm_sdcc.1/by-name/system`
    `/system`

The Android powered device is now capable of running the software described in this thesis. An Android operating system installation file (APK) is included with this thesis.

2.6 Libraries

Various Android and Java libraries were used to construct the prototype. Each library used was chosen to meet requirements needed for a successful prototype design and functionality of the prototype is enhanced by the libraries included. An explanation as to how and why each library was used is provided in the following subsections contained within this chapter.

2.6.1 Android SDK

The Android software development kit provides API libraries and development tools to create applications on Android powered devices. The Android operating system was chosen because many mobile devices run the Android operating system and Android applications are written in Java which allows the prototype software to be easily ported to other devices that support Java. More information and downloads for the Android SDK can be found at http://developer.android.com/sdk/index.html.

2.6.2 Linux Wireless (iw)

Linux Wireless or iw is a new nl80211 based CLI configuration utility for wireless devices [2]. The software interacts with wireless network adapters and provides functionality to perform common wireless network operations such as scanning for networks, connecting to networks, and configuring device attached network adapters. This software was used to implement a custom WifiManager service to be cable of performing selective wireless network scans on subsets of frequencies and SSIDs. iw is a much faster wireless scanning utility compared to the native Android WifiManager service. Refer to section 5.2 for further explanation explaining the difference between successive wireless network scan intervals when scanning with the native Android WifiManager versus a custom WifiManager service. More information about Linux Wireless can be found at http://wireless.kernel.org/.

2.6.3 Google Gson

Google Gson is a java library for converting Java objects into JSON representations and convert JSON strings into equivalent Java objects. The library also has the ability to work with arbitrary Java objects including pre-existing objects even in instances where source code is not available [3]. The library also supports Java Generics. The library was used to allow network map data to be shared in a common format structured as JSON arrays, and to allow the possibility of sharing network map data via RESTful methods. More information and downloads can be found at https://code.google.com/p/google-gson/.

2.6.4 Android QR Code Decoder and Encoder

Android QR Code Decoder and Encoder is an Android library to encode and decode barcodes and quick response codes. The library is a port of the ZXing (version 2.1) project but [is] reduced in size and scope and can be used as a direct call from any Android project instead of using the ZXing Intents mechanisms [4]. The library helps make working with QR codes easier in Android operating systems. This library was used to share network maps in a common graphical format support by mobile devices. JSON strings containing network map data can be encoded as barcodes or quick response codes and shared more easily than long JSON strings. More information and downloads can be found at https://code.google.com/p/android-quick-response-code/.

2.6.5 Libsuperuser

Libsuperuser is an Android persistent root environment access library written by the author of SuperSU and Superuser. Both SuperSU and Superuser allow Android applications root access in a similar way ‘sudo’ allows specified users root access in some Linux operating systems. The [library allows] applications requiring root access to overcome common problems when using root with as little code as possible [5]. This library was used to access a persistent root shell from the Android operating system reduce the execution time for multiple root shell commands. One root shell access request is granted instead of one time per each command requiring root access. More information and downloads can be found at http://su.chainfire.eu/.

2.6.6 AChartEngine

AChartEngine is an Android charting library supporting many common chart types. Features for charts include multiple series per chart, real-time chart updating, optimized data structures to accommodate large datasets per chart, and combined charts using multiple rendering styles. Charts can be built as a view that can be added to a view group or as an intent [6]. This makes the library easy to integrate with the Android user interface. The library was used to help in the creation of charting experimental data. More information and downloads can be found at https://code.google.com/p/achartengine/.

2.6.7 Efficient Java Matrix Library

Efficient Java Matrix Library is a Java library for linear algebra used for manipulating large and small dense matrices. The library is designed to be as computationally and memory efficient as possible. Algorithms used for computations can be chosen at runtime to enhance performance and reduce memory usage. The library is currently the fastest single threaded pure Java library [for matrices] and contains additional optimizations for small matrices [7].This library was used for the data storage capabilities of the library, and to perform common matrix operations. More information and downloads can be found at https://code.google.com/p/efficient-java-matrix-library/.

2.6.8 OrmLite

OrmLite (Object Relational Mapping Lite) is an Android and Java library for persisting Java Objects to SQL databases. The library supports multiple SQL databases and also includes native calls to Android operating system database APIs. The goal of the library is to avoid the complexity and overhead of more standard ORM packages [8]. The Android operating system uses an embedded SQLite database for storing application data. This library was used to interact with a SQLite database for persistent data storage and retrieval. More information and downloads can be found at http://ormlite.com/sqlite_java_android_orm.shtml/.

2.7 Network Map Format File

The following is the contents of an example wireless network map file formatted using a common JSON string. The below example consists of four wireless network access points. The wireless access points are represented within the JSON string as a JSON formatted array. The maximum storage offered by QR Codes is 1,264 characters of ordinary / ASCII text character [9]. The size of the JSON string formatted wireless network map cannot exceed the maximum QR Code storage capacity if stored within a single QR Code. This means that approximately only twelve wireless access points can be stored within one QR code because each access point takes roughly 100 characters of storage.

Due to this limitation only relatively small wireless network maps can be stored using a single QR Code. Larger wireless network maps would require sufficiently more storage capacity and more QR Codes and could be represented with many QR Codes, but this would be burdensome to import many QR Codes. An alternative is to encode the JSON string formatted wireless network map using some encoding process and compress the number of characters per access point. This would allow for more access points on a single QR Code and larger wireless network maps. The original JSON string would then be decoded after the QR Code is captured. Encoding (compressing) and decoding (decompressing) QR Code data is not within the scope of this thesis.

```
[  "name":"lab",
   "description":"3 routers",
   [ { "bssid":"00:21:91:da:27:ab",
       "sssid":"router1",
       "frequency":2442,
       "x":0,
       "y":22,
       "z":0 },
     { "bssid":"00:90:4c:91:00:03",
       "ssid":"router2",
       "frequency":2412,
       "x":19,
       "y":22,
       "z":0 },
     { "bssid:"00:11:22:33:44:58",
       "ssid":"router3",
       "frequency":2412,
       "x":0,
       "y":0,
       "z":0 },

     { "bssid:"00:11:22:33:44:58",
       "ssid":"router4",
       "frequency":2412,
       "x":0,
       "y":0,
       "z":0 }
   ]
]
```

Listing 2.1: Sample Network map file with four wireless access points

3 Overview of the Experiment

3.1 Introduction

The goal of these experiments was to use common high-tech devices to create a prototype device capable of estimating its own orientation and location within an indoor environment. This prototype should aid in creating an augmented reality experience for a user by projecting a virtually created three-dimensional environment onto the surfaces of the physical real world environment. Orientation and location calculations needed to be computed simultaneously and in real-time. The software for the prototype needed to compute individual location and orientation algorithms on independent threads.

The estimated orientation of the prototype needed to correlate to a global

coordinate system. The global referenced coordinate system is defined by magnetic North and the gravitational direction of Earth. The orientation of the prototype also needed to be computed using hardware available on most mobile devices. The sensor hardware used for calculating orientation were the magnetometer, accelerometer, and gyroscope. Accuracy for orientation estimates needed to be within one degree.

Location prediction needed to be relative to an indoor environment that was defined by a wireless network map. Location needed to be estimated using the received signal strength (*RSSI)* from wireless access points. The prototype needed to implement an indoor position system (*IPS*) capable of synchronizing the true position of the prototype in the real world to a position in the virtual world and vice versa. Assumptions made about location prediction calculations were as follows:

1. The location of all wireless network access points are predetermined and known by the prototype. Wireless network locations have been stored in a wireless network map file.

2. The measured signal strength all wireless access points reflect signal strength values that have only been transmitted through air and did not pass through walls or any other solid object.

3. A minimum of three wireless access points are available when estimating the location of the prototype.

3.2 Experimental Overview – Process

The overall experimental process was broken into two distinct parts. The first was to implement an algorithm to estimate the orientation of the prototype. The second was to create an estimation method to track the location of the prototype. The process and testing of both experimental parts were conducted independently. Software was created to estimate the orientation of the prototype and separate software was created to estimate the location of the prototype. After both individual components of the prototype were tested and verified to produce acceptable results, then both were combined into a grouped software package running on independent threads that communicated. Testing was performed on the combined solution to verify both processes were operating correctly in the integrated software.

3.3 Content of Experimental Chapters

Chapters 3 through 6 detail the calculations and results for orientation and location tracking of the prototype. Prototype device hardware and software setup is detailed in Chapter 3. Libraries and development kits used in the prototype software is explained in section 3.4. Wireless network map file example is detailed in section 3.5. Chapter 4 explains the orientation calculations and details how to calculate orientation using the

accelerometer and magnetometer, gyroscope, and by using all three sensors using a technique called sensor fusion. Refer to section 2.4 for explanations of different orientation representations. Chapter 5 details the how location of the prototype is calculated using wireless access points at predetermined positions in a two-dimensional coordinate space. Chapter 6 lists test results from orientation and location tests.

3.4 Representations of Orientation

The orientation of the prototype can be represented in different forms that are each equivalent of another form. Each representation of orientation can be transformed into any other representation using mathematical conversions. Euler angles, rotations matrices, and quaternions are used in the calculations in device orientation and are detailed below.

3.4.1 Euler Angle Representation

Euler angles are the easiest three dimensional orientation vector to understand and consists of three values defining azimuth, pitch, and roll of an object. Euler angles depend upon two known fixed points of origin. The first point is the direction of magnetic North, and the second is the direction of gravity. Euler angle rotations are not

commutative, meaning $a \otimes b \neq b \otimes a$. In other words performing three successive rotations does not yield the same orientation if the order of rotations is changed [10]. A major drawback to Euler angles is gimbal lock. Gimbal lock is the loss of one degree of freedom in a three dimensional coordinate space. This occurs when orientation cannot be uniquely represented using Euler angles and is dependent upon the order of rotation [11]. An example of when gimbal lock could happen is when pitch equals $\pm 90°$. The form of a vector comprised of Euler angles is represented in 3.1.

$$\vec{e} = [azimuth \quad pitch \quad roll] \tag{3.1}$$

3.4.2 Rotation Matrix Representation

A rotation matrix is a rotation represented by an orthogonal matrix, and is a sequence of rotations, with each rotation about a principle axis is a coordinate system [12]. Three by three rotation matrices are explained since orientation is computed in three dimensional space. After all rotation computations have been completed a final rotation is computed. Rotation matrix multiplication is not commutative, meaning $a \otimes b \neq b \otimes a$. This is because the final rotation can be computed in different orders for each of the principle axis. Rotation matrix orientation is not prone to gimbal lock, but is much more computensive to use in a three coordinate system. The form of a rotation matrix for a three axis rotation is represented in 3.2.

$$r = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} \tag{3.2}$$

3.4.3 Quaternion Representation

A quaternion is a four-dimensional complex number that can be used to represent the orientation of a ridged body or coordinate frame in three-dimensional space [13]. A quaternion is comprised of four components, one real and three imaginary [14]. The components are *x*, *y*, *z*, (imagery) and *w* (real) where *x*, *y*, and *z* are coordinates and *w* is rotation. Two quaternions can be multiplied using the Hamilton rule and the product of two quaternions is not commutative, meaning $a \otimes b \neq b \otimes a$. Quaternions are also not prone to gimbal lock and are less computensive than rotation matrices for three dimensional coordinate systems. The form a quaternion is represented as a vector and is shown in 3.3.

$$\vec{q} = [x \quad y \quad z \quad w] \tag{3.3}$$

4 Prototype Orientation Tracking

4.1 Introduction

Development and testing for prototype orientation accuracy and response was broken into two parts. The first part was to determine the level of accuracy and amount of noise recorded by the magnetometer, accelerometer, and gyroscope sensors. The prototype remained stationary during testing and was not moved, touched, or interacted with once sensor test readings started being captured.

The second part of development and testing for determining accurate and responsive prototype orientation was to record and minimize the time between the real world movement of the physical prototype and the calculated orientation within the prototype software calculations. This was tested by interacting with the prototype.

Raw sensor data tests using the accelerometer, magnetometer, and gyroscope are

discussed in section 4.2. Accelerometer and magnetometer calculated orientation is discussed in section 4.3. Gyroscope calculated orientation is discussed in section 4.4. Orientation calculated by fusing sensor data from the accelerometer, magnetometer, and gyroscope is discussed in section 4.5. Raw sensor data with bias is discussed in section 4.6.

4.2 Raw Sensor Data

The first tests involved recording and analyzing the sensor data from the magnetometer, accelerometer, and gyroscope. This was done to determine accuracy of each sensor and discover the amount of noise that was recorded in the senor data. High levels of noise and low accuracy results would require more post processing and filtering after the sensor data is captured. Sensor data recording for raw sensor data was performed with the mobile device lying flat and stationary on a surface parallel to the ground with the screen side (front) of the device was facing up towards the sky. The orientation of the device relative to North was not taken into consideration since it was irrelevant when gathering this data.

The magnetometer sensor contained the largest amount of noise, followed by the accelerometer sensor. These two sensors would produce a considerable amount of variance when calculating the orientation of the prototype. The gyroscope sensor had almost no error in the readings, and produced much more stable and accurate results.

Filtering the sensor data from the accelerometer and magnetometer to produce more accurate results was necessary.

4.3 Accelerometer and Magnetometer Calculated Orientation

The orientation of the prototype relative to a global coordinate system is possible by combining the data from the accelerometer and magnetometer. The global coordinate system is representative of the real world coordinate system which is determined by the gravitational and magnetic forces of Earth. Tilt (*pitch* and *roll*) angles can be computed using the accelerometer and heading (*azimuth*) angle can be computed using the magnetometer.

The accelerometer provided a three-dimensional vector containing acceleration values for each axis measured in meters per second squared (m/s^2). The form of the acceleration vector was $\vec{a} = [x \quad y \quad z]$. The force of gravitational acceleration upon the accelerometer must be taken into account when calculating the orientation of the prototype. For example, the accelerometer would return a vector close to $\vec{a} = [0 \quad 0 \quad 9.8]\, m/s^2$ when the prototype is laying on a horizontal surface at rest. If the prototype is in free fall then then accelerometer would return a vector close to $\vec{a} = [0 \quad 0 \quad 0]\, m/s^2$. The effect of gravity is why pitch and roll angles of the prototype can be determined using the accelerometer.

The magnetometer provided a three-dimensional vector containing the ambient

magnetic field strengths for each axis measured in micro-Tesla (μT). The form of the magnetic field strength vector was $\vec{m} = [x \quad y \quad z]$. This sensor responds to the magnetic field of Earth in the same manner as a compass. The heading of magnetic North in relation to the prototype can be determined using the magnetometer and also the azimuth angle of the prototype.

Calculating the orientation (*azimuth, pitch,* and *roll*) of the prototype using data provided by the accelerometer and magnetometer is detailed as follows. Record the vector data from the accelerometer and magnetometer and store in $\vec{m}$ and $\vec{a}$.

$$\vec{m} = [m_x \quad m_y \quad m_z] \tag{4.1}$$

$$\vec{a} = [a_x \quad a_y \quad a_z] \tag{4.2}$$

Equation 4.3 calculates the cross product of $\vec{m}$ and $\vec{a}$, which is the sensor data from the magnetometer and accelerometer. The normal of $\vec{h}$ and $\vec{a}$ is calculated in equations 4.4 and 4.5.

$$\vec{h} = [h_x \quad h_y \quad h_z] = [m_y a_z - m_z a_y \quad m_z a_x - m_x a_z \quad m_x a_y - m_y a_x] \tag{4.3}$$

$$\|\vec{h}\| = \frac{1}{\sqrt{{h_x}^2 + {h_y}^2 + {h_z}^2}} \tag{4.4}$$

$$\|\vec{a}\| = \frac{1}{\sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2}} \tag{4.5}$$

Check if $\|\vec{h}\| < .01$. If $\vec{h}$ is not greater than .01 the prototype is in free fall or too close to the North Pole. No further calculations can be performed and orientation must be

determined after the next data sample is recorded from the accelerometer and magnetometer. If $\vec{h}$ is greater than .01, continue and calculate the inverse of $\vec{h}$ and $\vec{a}$ shown in equations 4.6 and 4.7. The constant .01 was obtained from the Android SDK.

$$\vec{h}^{-1} = \begin{bmatrix} {h^{-1}}_x & {h^{-1}}_y & {h^{-1}}_z \end{bmatrix} = \begin{bmatrix} \frac{h_x}{\|\vec{h}\|} & \frac{h_y}{\|\vec{h}\|} & \frac{h_z}{\|\vec{h}\|} \end{bmatrix} \tag{4.6}$$

$$\vec{a}^{-1} = \begin{bmatrix} {a^{-1}}_x & {a^{-1}}_y & {a^{-1}}_z \end{bmatrix} = \begin{bmatrix} \frac{a_x}{\|\vec{a}\|} & \frac{a_y}{\|\vec{a}\|} & \frac{a_z}{\|\vec{a}\|} \end{bmatrix} \tag{4.7}$$

Equation 4.8 calculates the cross product of $\vec{a}^{-1}$and $\vec{h}^{-1}$ and store in $\vec{j}$.

$$\vec{j} = [j_x \quad j_y \quad j_z] = \vec{a}^{-1} \times \vec{h}^{-1} \tag{4.8}$$

Create a rotation matrix based on the vectors $\vec{h}^{-1}$, $\vec{j}$, and $\vec{a}^{-1}$.

$$R_{MA} = \begin{bmatrix} r_{ma_0} & r_{ma_1} & r_{ma_2} \\ r_{ma_3} & r_{ma_4} & r_{ma_5} \\ r_{ma_4} & r_{ma_7} & r_{ma_8} \end{bmatrix} = \begin{bmatrix} {h^{-1}}_x & {h^{-1}}_y & {h^{-1}}_z \\ j_x & j_y & j_z \\ {a^{-1}}_x & {a^{-1}}_y & {a^{-1}}_z \end{bmatrix} = \begin{bmatrix} \vec{h}^{-1} \\ \vec{j} \\ \vec{a}^{-1} \end{bmatrix} \tag{4.9}$$

Convert the rotation matrix to Euler angles where e_{ma_a} is azimuth (4.11), e_{ma_p} is pitch (4.12), and e_{ma_r} is roll (4.13).

$$\overrightarrow{e_{ma}} = [e_{ma_a} \quad e_{ma_p} \quad e_{ma_r}] \tag{4.10}$$

$$e_{ma_a} = arctan\left(\frac{{h^{-1}}_y}{j_y}\right) \tag{4.11}$$

$$e_{ma_p} = arctan\left(a^{-1}{}_y\right) \tag{4.12}$$

$$e_{ma_r} = arctan\left(\frac{a^{-1}{}_x}{a^{-1}{}_z}\right) \tag{4.13}$$

Combining data from the accelerometer and magnetometer to calculate the orientation of the prototype produced unacceptable results. Both sensors, especially the magnetometer, were inaccurate and introduced noise to the measurements [15]. The calculated orientation of the prototype jumped sporadically on each sample. The reaction time of the magnetometer was much slower than the accelerometer. The lag in the magnetometer resulted in the azimuth angle of the prototype updating at a visibly slow rate especially when the device was turned horizontally. The raw sensor test data explained in the previous section also confirm a large amount of noise in both the accelerometer and magnetometer.

4.4 Gyroscope Calculated Orientation

The gyroscope sensor provided a three-dimensional vector containing the angular rotational speed for each axis measured in radians per second (*rad/sec*). The form of the gyroscope vector was $\vec{g} = [x \quad y \quad z]$. Calculating the orientation of the prototype using the gyroscope was done by integrating the gyroscope sensor data with respect to time measured in nanoseconds (*ns*). The idea is to record the current angular speed of the prototype using the gyroscope. Then using a previously known orientation of the

prototype, apply the current angular speed to the previously known orientation to calculate the new orientation of the prototype.

The limitation of this method is that the initial orientation of the prototype must be known otherwise the orientation cannot be computed relative to a global coordinate system. Calculating the orientation (*azimuth, pitch,* and *roll*) of the prototype using data provided by the gyroscope is detailed as follows. The previously calculated orientation of the prototype is $\overrightarrow{q_g}$ (4.14), the current gyroscope data is $\vec{g}\,(4.15)$, and dt is the time between the previously computed orientation of the prototype and the current orientation calculation (4.16). The first $\overrightarrow{q_g}$ is the initial frame of the prototype.

$$\overrightarrow{q_g} = [q_{g_x} \quad q_{g_y} \quad q_{g_z} \quad q_{g_w}] \tag{4.14}$$

$$\vec{g} = [g_x \quad g_y \quad g_z] \tag{4.15}$$

$$dt = time_{current} - time_{last\ update} \tag{4.16}$$

Calculate the angular speed of the data sample.

$$s = \sqrt{{g_x}^2 + {g_y}^2 + {g_z}^2} \tag{4.17}$$

If the angular speed of the data sample is greater than ε then normalize the gyroscope sensor data shown in equation (4.18). For this application $\varepsilon = 1x10e - 9$ and is obtained from the Android SDK.

$$\overrightarrow{g'} = \left[{g'}_x \quad {g'}_y \quad {g'}_z\right] = \left[\frac{g_x}{s} \quad \frac{g_y}{s} \quad \frac{g_z}{s}\right] \tag{4.18}$$

Convert the axis angle $\vec{g}'$ to a rotation vector in the form of a quaternion (4.20).

$$\overrightarrow{q_{dt}} = [q_{dt_x} \quad q_{dt_y} \quad q_{dt_z} \quad q_{dt_w}] \tag{4.19}$$

$$\overrightarrow{q_{dt}} = \left[sin\left(\frac{s*dt}{2}\right)g'_x \quad sin\left(\frac{s*dt}{2}\right)g'_y \quad sin\left(\frac{s*dt}{2}\right)g'_z \quad cos\left(\frac{s*dt}{2}\right)\right] \tag{4.20}$$

Multiply original orientation $\overrightarrow{q_g}$ by $\overrightarrow{q_{dt}}$ shown in equations (4.12) through (4.24).

$$q_{g_x} = \left(q_{g_x} * q_{dt_x}\right) - \left(q_{g_y} * q_{dt_y}\right) - \left(q_{g_z} * q_{dt_z}\right) - \left(q_{g_w} * q_{dt_w}\right) \tag{4.21}$$

$$q_{g_y} = \left(q_{g_y} * q_{dt_x}\right) + \left(q_{g_x} * q_{dt_y}\right) - \left(q_{g_w} * q_{dt_z}\right) + \left(q_{g_z} * q_{dt_w}\right) \tag{4.22}$$

$$q_{g_z} = \left(q_{g_z} * q_{dt_x}\right) + \left(q_{g_w} * q_{dt_y}\right) + \left(q_{g_x} * q_{dt_z}\right) - \left(q_{g_y} * q_{dt_w}\right) \tag{4.23}$$

$$q_{g_w} = \left(q_{g_w} * q_{dt_x}\right) - \left(q_{g_z} * q_{dt_y}\right) + \left(q_{g_y} * q_{dt_z}\right) + \left(q_{g_x} * q_{dt_w}\right) \tag{4.24}$$

Convert $\overrightarrow{e_g}$ to Euler angles where e_{g_a} is azimuth (4.26), e_{g_p} is pitch (4.27), and e_{g_r} is roll (4.28).

$$\overrightarrow{e_g} = [e_{g_a} \quad e_{g_p} \quad e_{g_r}] \tag{4.25}$$

$$e_{g_a} = arctan\left(\frac{2\left(q_{g_x} * q_{g_w} + q_{g_y} * q_{g_z}\right)}{1-2\left({q_{g_z}}^2 + {q_{g_w}}^2\right)}\right) \tag{4.26}$$

$$e_{g_p} = arcsin\left(2\left(q_{g_x} * q_{g_y} + q_{g_z} * q_{g_w}\right)\right) \tag{4.27}$$

$$e_{g_r} = arctan\left(\frac{2\left(q_{g_x} * q_{g_y} + q_{g_z} * q_{g_w}\right)}{1-2\left({q_{g_y}}^2 + {q_{g_z}}^2\right)}\right) \tag{4.28}$$

Calculating the orientation of the prototype with only the gyroscope also produced unacceptable results. The results were much more stable than using the accelerometer and magnetometer. The biggest problem with calculating the orientation of the prototype with only the gyroscope was drift. Gyroscope drift occurs from the small amount of error in the gyroscope sensor accumulating over time and occurs through temperature and motion changes [13]. At first the drift amount is small, but grows quite quickly due to the way gyroscope orientation is computed through integration. Over time, the calculated orientation of the prototype deviates greater and greater from the true orientation of the prototype referenced from the initial frame.

4.5 Sensor Fused Calculated Orientation

The calculated orientation of the prototype from the accelerometer and magnetometer contained too much variance over successive calculations while the variance in the orientation calculated from gyroscope sensor data was much lower. Accelerometer and magnetometer calculated orientation was relative to a global coordinate system, but the gyroscope orientation was not. Both methods needed to be combined using sensor fusion to calculate the orientation of the prototype containing less variance, relative to a global coordinate system, and not prone to drift. Sensor fusion is combining data from multiple sensors to achieve improved accuracies and more specific inferences than could not be achieved by the use of a single sensor alone [16]. In other

words, the calculated orientation of the prototype accuracy would improve by minimizing the weaknesses and complementing the strengths of the sensors.

Prototype orientation needed to be primarily calculated from the gyroscope sensor data and updated intermediately from the accelerometer and magnetometer sensor data. This would yield a smooth orientation that would be corrected for gyroscope drift periodically and allow the orientation of the prototype to be referenced using a global coordinate system. The accelerometer and magnetometer sensor data was processed using a low pass filter and the gyroscope sensor data was processed using a high pass filter. This combined filtering is called a complimentary filter [17] [18]. The following is a filter implementation proposed by Shane Colton.

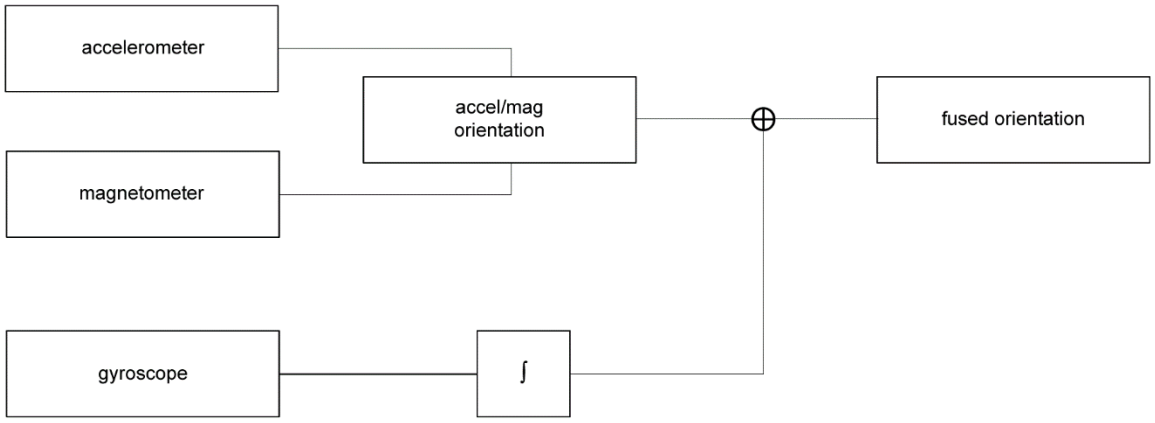


Figure 4.1: Overall Complimentary Filter Implementation

The low-pass filter removes short term variance in a signal allowing only long term signal changes to pass through the filter. A high-pass filter does the exact opposite of a low-pass filter. It allows only short term signal changes to pass through the

filter while filtering out signals that are steady over time and can be used to cancel out drift [18]. Applying a complementary filter to the accelerometer and magnetometer calculated orientation and gyroscope calculated orientation is shown in equation (4.29).

$$orientation = FC * gyroOrientation + (1 - FC) * accMagOrientation \quad (4.29)$$

Two filter coefficient constants (*FC* and *1 - FC*) are introduced in equation (4.29). Optimal results have been computed using $FC = .98$. The low-pass filter part of the equation is $(1 - FC) * accMagOrientation$ and the high-pass filter part is $FC * gyroOrientation$. The low-pass portion averaged the accelerometer and magnetometer orientation calculations by applying changes to the global orientation slowly. The high-pass portion minimized the drift in the gyroscope calculated orientation.

The generalized complimentary filter implementation pseudo code is shown in listing 4.5.1. The complementary filter portion has been bolded in the listing. This section of code runs on a separate thread on the prototype hardware to maximize performance. The orientation calculations based upon the sensor data from the accelerometer and magnetometer was computed the same way as explained in section 4.3. The same is true for the orientation calculations for the gyroscope sensor data explained in section 4.4.

```
INTERVAL = 30 //milliseconds
FC = .98 //filter coefficients FC and 1 – FC
accelMagOrientation() { /* refer to section 4.3 */ }
gyroOrientation() { /* refer to section 4.4 */ }
fusedOrientation() {
     return FC * orientation + (1 – FC) * accelMagOrientation()
}
while(true) {
     accelMagOrientation()
     gyroOrientation()
     if(currentTime>=INTERVAL)
          orientation = fusedOrientation()
}
```

Listing 4.5.1: Pseudo code implementation of fused orientation

The fused orientation was computed incrementally in a timed loop with a time constant of 30 milliseconds. The time constant is the relative duration of signal the filter will act on. For a low-pass filter, signals much longer than the time constant pass through unaltered while signals shorter than the time constant are filtered out and the opposite is true for a high-pass filter [18].

The following details the computations of the `fusedOrientation` function in Listing 4.5.1 and uses $\overrightarrow{e_{ma}}$ and $\overrightarrow{e_g}$ to calculate the fused orientation $\overrightarrow{e_f}$, where e_{f_a} is azimuth (4.31), e_{f_p} is pitch (4.32), and e_{f_r} is roll (4.33).

$$\overrightarrow{e_f} = [e_{f_a} \quad e_{f_p} \quad e_{f_r}] \tag{4.30}$$

$$e_{f_a} = FC * e_{g_a} + (1 - FC) * e_{ma_a} \quad (4.31)$$

$$e_{f_p} = FC * e_{g_p} + (1 - FC) * e_{ma_p} \quad (4.32)$$

$$e_{f_r} = FC * e_{g_r} + (1 - FC) * e_{ma_r} \quad (4.33)$$

The orientation computed from fusing the accelerometer and magnetometer orientation and gyroscope orientation yielded much better results. Variance over successive calculations was highly reduces and a global coordinate system was able to be referenced. Gyroscope drift was also eliminated.

4.6 Sensor Bias

Further reducing the amount of variance in the fused orientation was possible by eliminating some of the sensor noise in the raw sensor data. This was done by introducing a bias to the sensors to negate inaccuracies and filter out noise. This was computed prior to calculating orientation. For the scope of this work bias is defined as the average recorded error from a sensor accumulated over a predefined number of sensor readings and is shown in equations (4.34) and (4.35). 1000 sensor readings were recorded for each sensor and the bias value calculated for each sensor was done so independently from all other sensors. In other words, the bias value calculated for the accelerometer was not influenced by the bias calculated for the gyroscope and is also conversely true. The bias can also be considered a very simplistic filter. Bias was calculated using the

following procedure. Bias was calculated by the following where b_A is for the accelerometer, and b_G is for the gyroscope.

$$b_a = \begin{bmatrix} b_{a_x} & b_{a_y} & b_{a_z} \end{bmatrix} = \frac{\sum_{n=0}^{1000} [a_x \quad a_x \quad a_x]}{1000} \tag{4.34}$$

$$b_g = \begin{bmatrix} b_{g_x} & b_{g_y} & b_{g_z} \end{bmatrix} = \frac{\sum_{n=0}^{1000} [g_x \quad g_x \quad g_x]}{1000} \tag{4.35}$$

Removing the bias from the raw sensor data reduced the variance of the calculated fused orientation. Orientation calculated with the accelerometer and magnetometer as well as the gyroscope also could benefit from bias removal. Fused orientation results without bias removal were much better than both the accelerometer and magnetometer and the gyroscope results. Because of this, tests results with bias removal for the accelerometer and magnetometer calculated orientation and the gyroscope calculated orientation are not included in this thesis.

$$\vec{a} = \begin{bmatrix} a_x - b_{a_x} & a_y - b_{a_y} & a_z - b_{a_z} \end{bmatrix} \tag{4.36}$$

$$\vec{g} = \begin{bmatrix} g_x - b_{g_x} & g_y - b_{g_y} & g_z - b_{g_z} \end{bmatrix} \tag{4.37}$$

Bias was removed from the accelerometer and gyroscope raw sensor data by subtracting the bias vector from both respective vectors and shown in equation (4.36) and (4.37).

5 Prototype Location Tracking

5.1 Location Tracking

The location of the prototype is estimated by performing successive wireless scans measuring the received signal strength (*RSS*) for specific wireless network access points. Wireless network access points can be comprised of multiple types such as 802.11 networks, Bluetooth devices, RFID tags, and any other wireless network where *RSS* can be determined. At the time of writing this thesis, only 802.11 network scanning has been implemented in the prototype. The method described assumes the location of each access point has been predetermined and recorded using Cartesian coordinates. This method also assumes that the measured *RSS* of all wireless access points only pass through free space or atmosphere. Refer to section 2.5 for how to up a wireless network

map. A minimum of three wireless access points were required for the method described to work.

A custom wireless network scanner was created to maximize the number of scans per second. This is discussed in section 5.2. Wireless network scan data is stored in a two-dimensional matrix to preserve scan history. A scan result is a tuple comprised of the identification marker of an access point and the distance in centimeters from the prototype to the access point as detailed in (5.1). The identification marker for each access point is unique, such as the basic service set identifier (*BSSID*) for 802.11 networks. During scanning, the newest scan result is stored in the n^{th} row of the scan result matrix and the oldest scan result is removed from the scan result matrix when a specified max scan history is reached (5.2). Each row in the scan result matrix represents individual wireless network scan results and each column represents individual access points (5.3).

$$scanResult = (point\ of\ interest\ id, distance\ to\ poi) \tag{5.1}$$

$$n = max\ scan\ history \tag{5.2}$$

$$scanResults = \begin{bmatrix} (poi_1, cm)_1 & (poi_2, cm)_1 & \dots & (poi_n, cm)_1 \\ (poi_1, cm)_2 & (poi_b, cm)_2 & \dots & (poi_n, cm)_2 \\ \vdots & \vdots & & \vdots \\ (poi_1, cm)_n & (poi_2, cm)_n & \dots & (poi_n, cm)_n \end{bmatrix} \tag{5.3}$$

The distance to an access point is estimated using the free space loss equation described in section 5.3. The *RSS* for a wireless access point greatly fluctuated and a filter had to be implemented to reduce the variances in *RSS* over successive wireless network scans. This is explained in section 5.4. After all distances have been estimated for a single wireless network scan, the location of the prototype is estimated using a least-

squares model for trilateration described in section 5.5. The weighted center of the estimated locations was determined when a specified number of estimated locations were calculated. This is described in section 5.6. A threshold value is then used to determine if a final location update should occur and is described in section 5.7. The pseudo-code for the overall location tracking implementation is shown in listing 5.1.

```
locationPred //2d list of predicted locations
locationWeighted //2d list of weighted locations
location //vector final calculated location
scanresults //stream of scan results
centroidK = 5 //refer to section 5.6
threshold = 1 //refer to section 5.7
dBmToCM(scanresults) { /* refer to section 5.4 */ }
leastSumErrors(scanresults) { /* refer to section 5.5 */ }
weightedCentroid(location) { /* refer to section 5.6 */ }
distanceTo(temp, location) { /* refer to section 5.7 */ }
while(true) {
     scanresultsCm = dBmToCM(scanresults.next())
     locationPred.add(leastSumErrors(scanresultsCm))
     if(size(locationPred) >= centroidK) {
          temp = weightedCentroid(locationPred)
          if(threshold <= distanceTo(temp, location))
               location = temp
     }
}
```

Listing 5.1: Pseudo code for implementation of location tracking

5.2 Custom WifiManager VS Android WifiManager

A custom 802.11 wireless network scanner was created to minimize the time between successive wireless network scans. The custom WifiManager was created using the Linux wireless (iw) software discussed in section 2.4.2. The WifiManager service built into the Android operating system is extremely slow and time between successive wireless network scan is approximately 1 second. This is because there are 14 channels on independent frequencies that need to be scanned for wireless 802.11 access points [19]. Figure 5.1 displays the 802.11 channels and frequencies where the top row represents channels and the bottom row represents frequencies.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2.412	2.417	2.422	2.427	2.432	2.437	2.442	2.447	2.452	2.457	2.462	2.467	2.472	2.484

Figure 5.1: 802.11 wireless network channels and corresponding frequencies

Creating a custom WifiManager based from Linux wireless allowed scanning for wireless network access points on a subset of 802.11 frequencies. The built-in WifiScanner did not have functionality to selectively choose frequencies, channels, or network name. Scanning on a subset of frequencies highly reduced the amount of time between successive scans. The greatest increase in scans per second was seen by minimizing the number of frequencies per scan to a single frequency.

Increasing the number of wireless networks scans per second was important because there existed significant error in the received signal strength from the wireless access points as discussed in section 5.3. Maximizing the number of scans per second

reduced the level of error because location estimation calculation could utilize more data for calculations. In other words, there exists an inverse correlation between the number of wireless network scans per second and accuracy of the estimated location.

5.3 RSS Distance Conversion

Received Signal Strength (*RSS*) is the amount of measured power in a received radio transmission or signal and is often referred to as Received Signal Strength Indicator (*RSSI*). The idea behind received signal strength is that the configured transmitted power at the transmitter device directly affects the received power at the receiving device [19]. According to Frii's free space transmission equation (5.4), the strength of a signal degrades quadratically in relation to the distance of the sender and is used to convert *RSS* to distance.

$$P_{r(d)} = \frac{P_t G_t G_r \lambda^2}{(4\pi d)^2} \tag{5.4}$$

$P_{r(d)}$ is the received power at the receiver is, P_t is the transmission power of the sender, G_t is the gain of the transmitter, G_r is the gain of the receiver. In embedded devices, such as mobile devices, $G_t = G_r = 1$ [19]. The wavelength of the signal is λ and d is the distance between the sender and receiver normally. The wavelength (λ) can be substituted by equation (5.5) where c is the speed of light and f is the frequency of the

signal in hertz.

$$\lambda = \frac{c}{f}, \text{ where } c = 299{,}792{,}458 \tag{5.5}$$

Substituting λ yields equation (5.6). Equations (5.7) through (5.13) derive the final free space path loss equation to calculate distance from receiver to sender based upon the frequency and received signal strength of the signal.

$$\frac{P_{r(d)}}{P_t} = G_t G_r \left(\frac{c}{4\pi df}\right)^2 \tag{5.6}$$

$$\frac{P_t}{P_{r(d)}} = \left(\frac{4\pi df}{c}\right)^2 \tag{5.7}$$

$$\frac{P_t}{P_{r(d)}} = \left(\frac{4\pi df * 10^9}{299{,}972{,}458}\right)^2 \tag{5.8}$$

Applying $log_{10}(log)$ to both sides of equation (5.8) obtains the decibel (*dB*) version of the free space path loss equation (5.9) since RSS is measured in decibels-milliwatts (*dBm*) and is the logarithmic measurement of signal strength [20]. $FSPL$ is also substituted for $\frac{P_t}{P_{r(d)}}$.

$$FSPL = 10\ log_{10}\left(\frac{4\pi df * 10^9}{299{,}972{,}458}\right)^2 \tag{5.9}$$

$$FSPL = 20\ log_{10}\left(\frac{4\pi * 10}{2.99972458}\right) + 20\ log_{10}\ (f) + 20\ log_{10}\ (d) \tag{5.10}$$

$$20\ log_{10}\ (d) = FSPL - 20\ log_{10}\ (f) - 32.4423 \tag{5.11}$$

$$d = 10^{\left(\frac{FSPL - 20\log_{10}(f) - 32.4}{20}\right)} \quad (5.12)$$

Equation (5.12) calculates d in kilometers and must be converted to centimeters as shown in equation (5.13).

$$d_{cm} = d * 10^5 \quad (5.13)$$

Tests to determine the accuracy of the calculated distance based on *RSS* compared to a measured distance were performed. These tests were conducted by placing four 802.11 wireless access points at the same location. The prototype was then located at *50cm, 1m, 2m, 5m,* and *10m* away from access points. The *RSS* for each access point was recorded at each location. The *RSS* values were then converted to distance in centimeters using Frii's free path loss equation. Results revealed a high level of variance in *RSS* for successive scan and filtering the received signal was necessary. Figure 5.2 displays RSS conversions to distances in meters.

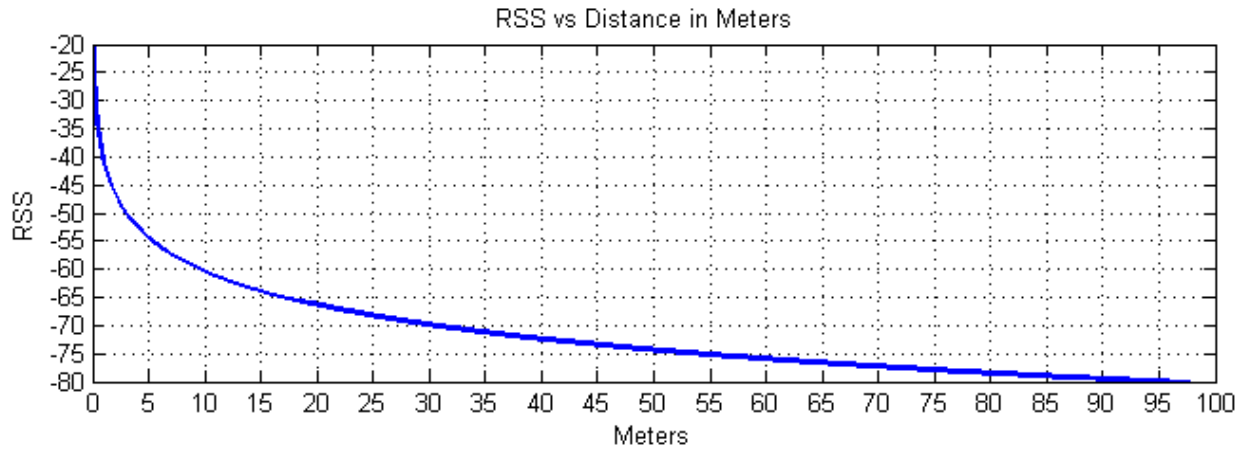


Figure 5.2: RSS to distance conversion in meters

5.4 Kalman Filter for RSS

Ideally, the *RSS* for a wireless network is steady and does not fluctuate. This is not the case in practical situations. Fluctuations in received signal can be caused by different factors: e.g. physical distance, reflections of objects, environmental parameters, movement of objects or change in the environment, antenna position and polarization etc. [21]. *RSS* fluctuations can be seen when recorded from a stationary location. Signal strength can increase or decrease greatly depending on the orientation of the access point in relation to the device measuring *RSS* values. For example, when the prototype is held in the hand of a user and is pointed to face a wireless access point the *RSS* could be -55dBm and have a lower value when the prototype is facing the opposite direction. This is because the human body contains more than 70% water and the resonance frequency of water is 2.4 gigahertz which most wireless networks operate on [22]

A Kalman filter was implemented to reduce the amount of variance in the *RSS*. This filters is an optimal estimator – i.e. infers parameters of interest from indirect, inaccurate and uncertain observations and is recursive so that new measurements can be processed as they arrive [23]. It is assumed that the process noise, which is the noise in the transmission power itself is negligible when compared to the measurement noise [24]. Equations (5.14) through (5.19) detail the simplified Kalman filter where x_n is the filtered output at the n^{th} iteration, p_n is the estimation error covariance, k_n is the gain, Q is the process noise covariance, R is the measurement noise covariance, and n is the current iteration of the filter (5.14). Values are chosen for $Q = .125$ and $R = 32$.

$$n = \{1, 2, 3, \dots, \infty\} \tag{5.14}$$

The first two equations, (5.15) and (5.16) are the prediction phase. The value x_n for the current filter iteration is updated with the filtered output value from the previous iteration (5.15). The estimation error covariance is updated using the process noise covariance value.

$$x_n = x_{n-1} \tag{5.15}$$

$$p_n = p_{n-1} + Q \tag{5.16}$$

The estimation error covariance and measurement noise covariance are used to update the gain (5.17). Raw *RSS* values from an access point is stored in $dBmRSS$ on every filter iteration and the filtered result is stored in x_n (5.18). The estimation error covariance is updated with the gain (5.19).

$$k_n = \frac{p_n}{p_n + R} \tag{5.17}$$

$$x_n = x_n + k_n * (dBmRSS - x_n) \tag{5.18}$$

$$p_n = (1 - k_n) * p_n \tag{5.19}$$

5.5 Least Squares Multilateration

Multilateration is a process which estimates the target position based on measurements of distances from at least three known fixed points. This information was based on verbal communication [37]. Equation (5.21) is based upon Pythagorean Theorem and is broken into two distinct parts where d_i is the estimated distance from the i[th] wireless access point, and ap_{x_i}, ap_{y_i} are the Cartesian coordinates for the location of the wireless access point.

$$i = \{1, 2, 3, \dots, n\}, where\ n = number\ of\ POIs \tag{5.20}$$

$$d_i{}^2 = ap_{x_i}{}^2 + ap_{y_i}{}^2 \tag{5.21}$$

The values x and y in equation (5.22) are unknown and represent the Cartesian coordinates for the location of the prototype. Equations (5.22) through (5.25) derive equation (5.22) for further calculations in estimating the location of the prototype.

$$d_i{}^2 = \left(x - ap_{x_i}\right)^2 + \left(y - ap_{y_i}\right)^2 \tag{5.22}$$

$$d_i{}^2 = x^2 - 2ap_{x_i}x + ap_{x_i}{}^2 + y^2 - 2ap_{y_i}y + ap_{y_i}{}^2 \tag{5.23}$$

$$d_i{}^2 - ap_{x_i}{}^2 - ap_{y_i}{}^2 = x^2 - 2ap_{x_i}x + y^2 - 2ap_{y_i}y \tag{5.24}$$

$$d_i{}^2 - ap_{x_i}{}^2 - ap_y{}^2 = -2ap_{x_i}x + -2ap_{y_i}y + x^2 + y^2 \tag{5.25}$$

Equation (5.25) is substituted for equation (5.26) and represents one scan result

of n total scan results. The left side of equation (5.25) represents the left side of equation (5.26) and the same for the right side of both equations (5.25) and (5.26). The equation in (5.27) represents all n total scan results and is a system on equations with two unknowns.

$$b_i = a_i * w \tag{5.26}$$

$$\vec{b} = A * \vec{w} \tag{5.27}$$

The total scan results must be equal or greater than three, $n \geq 3$. Enforcing that n is greater than three creates and over-determined system of equations where there is a single specific minimal length solution which can be obtained using the pseudoinverse [34]. The vector $\vec{b}$ and matrix A are detailed in (5.28) and (5.29). Matrix A is augmented with a column of ones in order to calculate the pseudoinverse.

$$\vec{b} = \begin{bmatrix} {distance_1}^2 - {ap_{x_1}}^2 - {ap_{y_1}}^2 \\ {distance_2}^2 - {ap_{x_2}}^2 - {ap_{y_2}}^2 \\ \vdots \\ {distance_n}^2 - {ap_{x_n}}^2 - {ap_{y_n}}^2 \end{bmatrix} \tag{5.28}$$

$$A = \begin{bmatrix} -2 * ap_{x_1} & -2 * ap_{y_1} & 1 \\ -2 * ap_{x_2} & -2 * ap_{y_2} & 1 \\ \vdots & \vdots & \vdots \\ -2 * ap_{x_n} & -2 * ap_{y_n} & 1 \end{bmatrix} \tag{5.29}$$

Equation (5.30) is created by rearranging (5.29). $PINV$ represents the pseudoinverse portion of the equation. The equation (5.31) has the pseudoinverse portion

expanded from (5.30).

$$\vec{w} = PINV(A) * \vec{b} \tag{5.30}$$

$$\vec{w} = ((A'A)^{-1} * A') * \vec{b} \tag{5.31}$$

After solving the system of equations the vector $\vec{w}$ contains three values, but only the first two values are needed. The first value (w_x) is the estimated x coordinate and second value (w_y) is the estimated y coordinate as shown in equation (5.32). Estimated coordinated pairs are stored in the nth row of matrix l_p and the oldest is removed after every estimation when a specified capacity is reached same as detailed in equation (5.33).

$$\vec{w} = [w_x \quad w_y \quad \blacksquare] \tag{5.32}$$

$$l_p = \begin{bmatrix} w_{x_1} & w_{y_1} \\ w_{x_2} & w_{y_2} \\ \vdots & \vdots \\ w_{x_n} & w_{y_n} \end{bmatrix} \tag{5.33}$$

5.6 Weighted Centroid

The weighed center of all locations stored in l_p is calculated and stored in vector $\overrightarrow{l_w}$ (5.34) after a predetermined number of estimated coordinate locations have been calculated and based from section 5.5. The weighted position is derived from the centroid determination which calculates the position of [the prototype] by averaging the coordinates of know reference points [25]. The known reference points are calculated from the last section and stored in matrix l_p (5.33).

$$\overrightarrow{l_w} = [l_{w_x} \quad l_{w_z}] \tag{5.34}$$

The weighted centroid is calculated by summing both columns of matrix l_p shown in equations (5.35) and (5.36) and the multiplying each respective summation by the variable $weight$ shown in equation (5.37).

$$l_{w_x} = \sum_{i=0}^{n} l_{p_{x_i}} * weight \tag{5.35}$$

$$l_{w_y} = \sum_{i=0}^{n} l_{p_{y_i}} * weight \tag{5.36}$$

$$weight = \frac{1}{length(l_p)} \tag{5.37}$$

5.7 Distance Threshold

The calculated position of the prototype would change slightly when the prototype remained stationary. This was due the noise that originated in the *RSS* values. A threshold value was introduced to minimize the location sporadically changing. The threshold value is a constant value. The final calculated location of the prototype is stored in vector $\vec{l}$ (5.38) and is only updated if the distance between the current value of $\vec{l}$ and $\overrightarrow{l_w}$ is greater than the threshold value as shown in equations (5.39) and (5.40).

$$\vec{l} = [l_x \quad l_y] \tag{5.38}$$

$$\vec{l} = \overrightarrow{l_w} \; \boldsymbol{iff} \; threshold \leq distanceTo \tag{5.39}$$

$$distanceTo = \sqrt{\left|{l_x}^2 - {l_{w_x}}^2\right| + \left|{l_y}^2 - {l_{w_y}}^2\right|} \tag{5.40}$$

The vector $\vec{l}$ stores the final location of the prototype relative to a referenced wireless network map.

6 Experimental Results and Discussion

6.1 Understanding Test Results

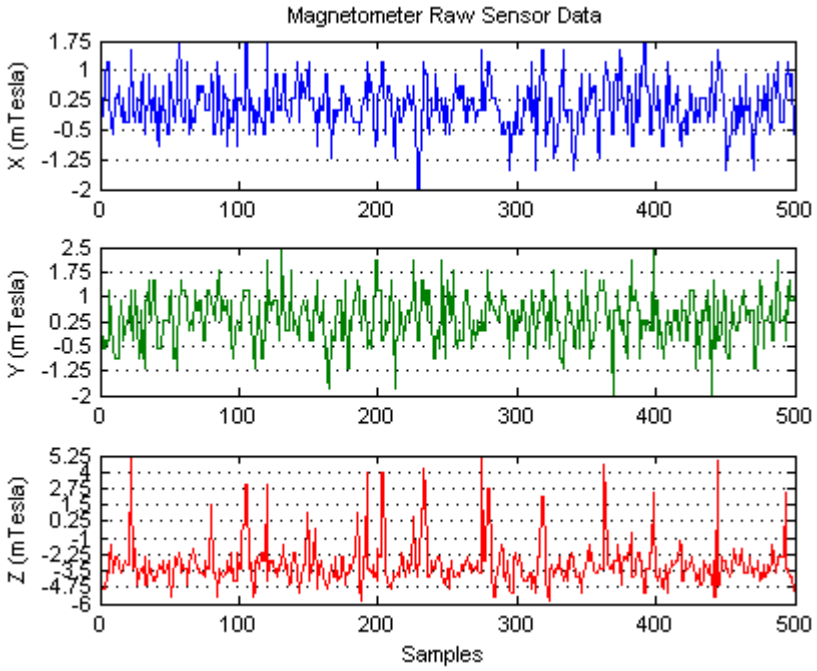
Chapter 6 contains experimental tests results. Raw sensor data results are shown in section 6.2. Orientation test results are shown in section 6.3 and are divided into accelerometer and magnetometer calculated orientation, gyroscope calculated orientation, and fused orientation calculated orientation. Results with and without bias removal are provided for each of the orientations tests. Chapter 4 details calculations for orientation estimation.

Sections 6.4 through 6.6 provide tests results for location estimation. *RSS* results for raw and filtered wireless scan data is provided in section 6.4. Distance determination from received signal strength tests are detailed in section 6.5. Section 6.6 contains test results for location prediction. Test results are provided with and without *RSS* filtering for

sections 6.5 and 6.6. Chapter 5 details location prediction calculations.

6.2 Raw Sensor Data Results

Experimental tests were conducted to measure the amount of noise in raw sensor data recorded from the magnetometer, accelerometer, and gyroscope. Results are shown in figures 6.1 through 6.3 and contain 500 samples of data. Sensor data was independently recorded for the *X*, *Y*, and *Z* axes. Each of the raw sensor data tests were conducted with the mobile device lying flat and stationary on a surface parallel to the ground with the screen side (front) of the device facing towards the sky.



SD	0.6242
SE	0.0279
CI(90%)	±0.0460

SD	0.7493
SE	0.0335
CI(90%)	±0.0552

SD	1.6620
SE	0.0743
CI(90%)	±0.1225

Figure 6.1: Raw Magnetometer Sensor Data

Sensor data for the magnetometer in figure 6.1 are recorded in microTesla. The magnetometer sensor had the highest amount of overall noise for raw sensor data tests. The standard deviation of the sensor readings for the magnetometer ranged from .062 (*X* axis) to 1.66 (*Z* axis) and show the inaccuracies of the magnetometer sensor hardware.

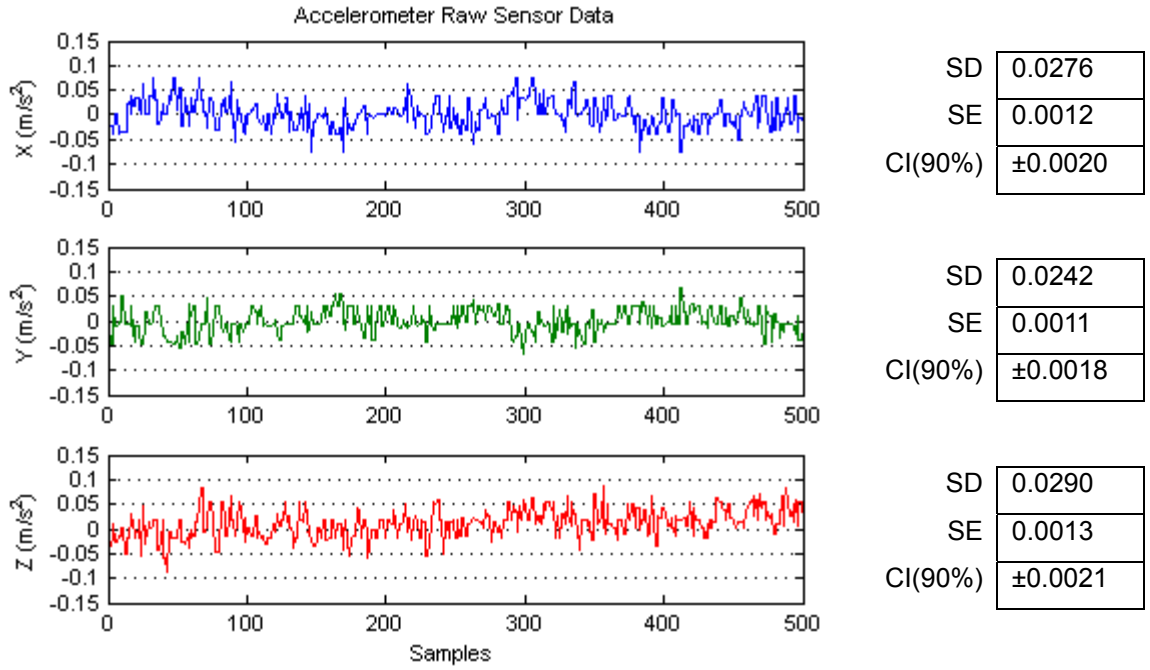
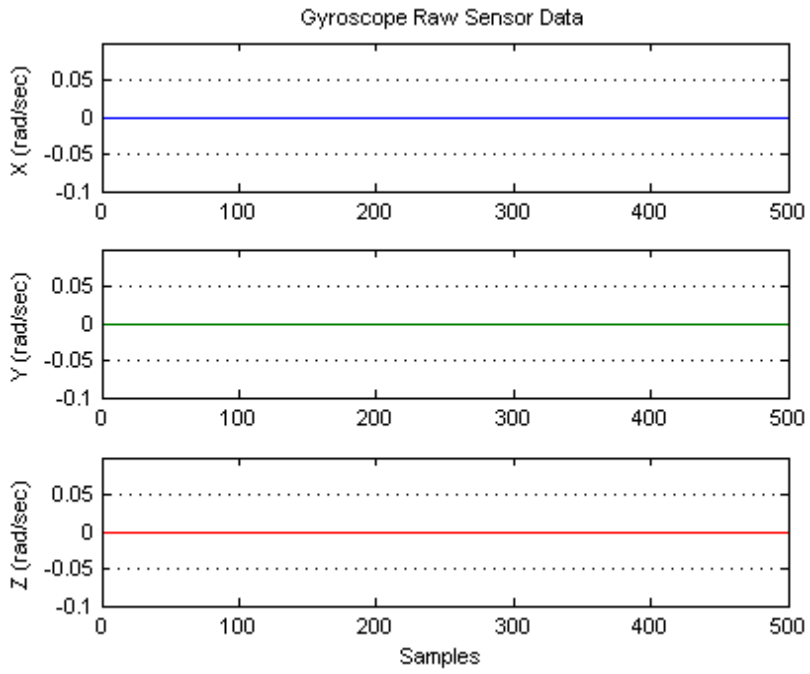


Figure 6.2: Raw Accelerometer Sensor Data

Figure 6.2 contains the raw sensor data recorded from the accelerometer and is measured in units of meters per second squared $\left(\frac{m}{sec^2}\right)$. The raw accelerometer sensor data also contained noise, but much less than the magnetometer. The standard deviation for raw accelerometer sensor data ranged from .024 (Y axis) to .029 (Z axis).

The raw gyroscope sensor data shown in figure 6.3 is measured in radians per second $\left(\frac{rads}{sec}\right)$. The gyroscope sensor contained almost no noise and has the least

amount of noise when compared to the magnetometer and accelerometer. The standard deviation was so small that it could not be computed accurately.



SD	2.44E-15
SE	1.09E-16
CI(90%)	±1.8E-16

SD	2.44E-15
SE	1.09E-16
CI(90%)	±1.8E-16

SD	2.44E-15
SE	1.09E-16
CI(90%)	±1.8E-16

Figure 6.3: Raw Gyroscope Sensor Data

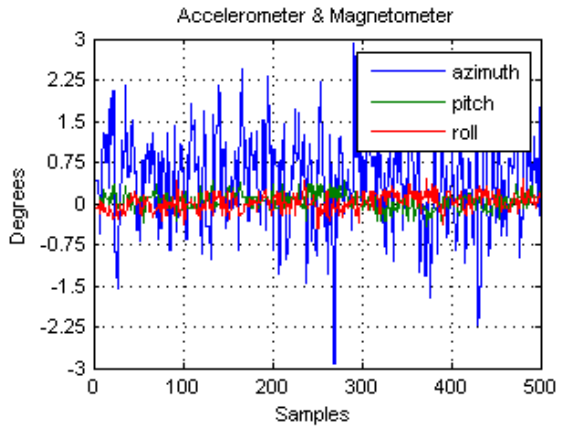
The results of the raw sensor data tests show a high amount of noise in the magnetometer sensor (figure 6.1) and accelerometer sensor (figure 6.2). The gyroscope sensor (figure 6.3) produced very steady test results.

6.3 Orientation Tracking Results

Experimental tests were conducted to determine the amount of variance in the estimated orientation over 500 successive samples. Orientation was calculated using three different methods and variance was measured in degrees. Figures 6.4 through 6.9 show tests results for different orientation methods. Test results have been recorded with and without bias removal for each estimation method. Bias is described in section 4.6. Tests were conducted with the mobile device lying flat and stationary on a surface parallel to the ground with the screen side (front) of the device was facing towards the sky.

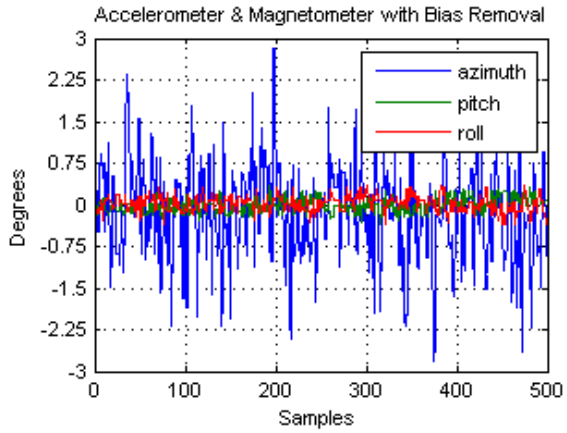
6.3.1 Accelerometer and Magnetometer Orientation Results

Figure 6.4 and 6.5 show test results from orientation estimation calculations using the accelerometer and magnetometer sensors. Figure 6.4 is without bias removal and figure 6.5 is with bias removal. Both tests show that the azimuth has the highest standard deviation when compared to pitch and roll. This is because azimuth is calculated from the magnetometer sensor data. Pitch and roll are computed from the accelerometer sensor data. These results correlate to the raw sensor data test results shown in figures 6.1 and 6.2. Applying bias removal, shown in figure 6.5, did not significantly improve the estimated orientation results using this method. This method could not be used to estimate prototype orientation because the standard deviation of the azimuth was too great.



	azimuth	pitch	roll
SD	0.8525	0.1430	0.1569
SE	0.0381	0.0064	0.0070
CI(90%)	±0.0628	±0.0104	±0.0116

Figure 6.4: Accelerometer/magnetometer calculated orientation



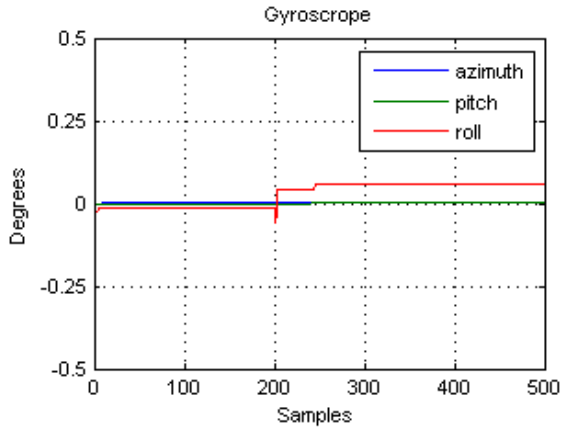
	azimuth	pitch	roll
SD	0.9183	0.1310	0.1478
SE	0.0411	0.0059	0.0066
CI(90%)	±0.0677	±0.0097	±0.0109

Figure 6.5: Accelerometer and magnetometer with bias removal calculated orientation

6.3.2 Gyroscope Orientation Results

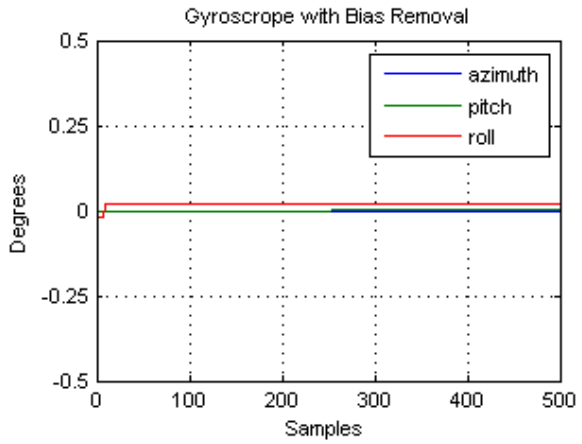
Figure 6.6 and 6.7 show test results from orientation estimation calculations using the gyroscope sensor. Figure 6.6 is without bias removal and figure 6.7 is with bias removal. The standard deviation of the gyroscope estimated orientation was much smaller than experimental test results for the accelerometer and magnetometer test

results shown in figures 6.4 and 6.5, and was so small it could not be accurately computed. These results correlate to the tests results shown in figure 6.3. Applying bias removal, shown in figure 6.7, did not significantly improve test results. The results computed from this method also contained gyroscopic drift. Over time the estimated orientation would deviate from the true orientation of the prototype. This method could not be used to estimate prototype orientation because of the gyroscopic drift.



	azimuth	pitch	roll
SD	1.1E-13	0.0003	0.0049
SE	5.1E-15	1.29E-5	0.0002
CI(90%)	±8E-15	±2.1E-5	±0.0004

Figure 6.6: Gyroscope calculated orientation

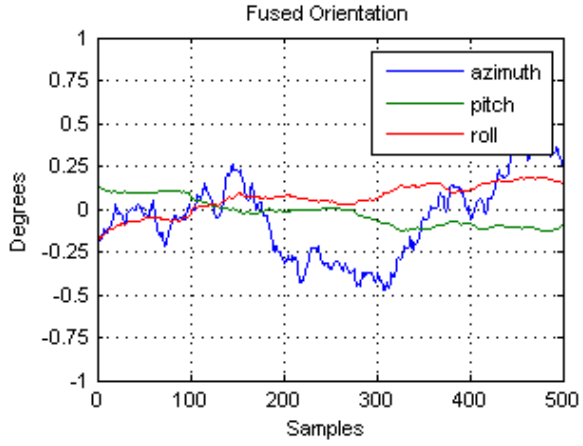


	azimuth	pitch	roll
SD	2.51E-6	0.0003	0.0346
SE	1.12E-7	1.49E-5	0.0015
CI(90%)	±1.8E-7	±2.5E-5	±0.0026

Figure 6.7: Gyroscope with bias removal calculated orientation

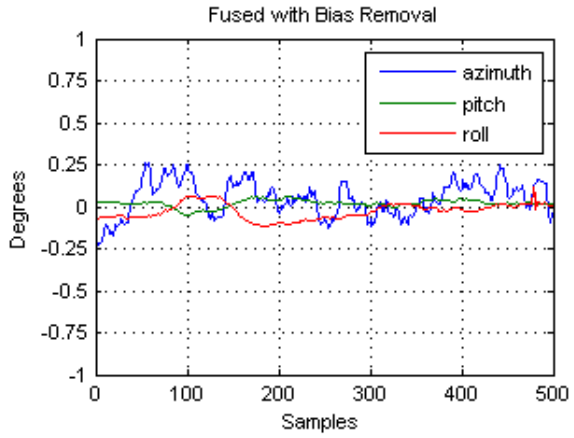
6.3.3 Sensor Fused Orientation Results

Figure 6.8 and 6.9 show test results from orientation estimation calculations using sensor fusion. This method was implemented using a complimentary filter comprised of data from the accelerometer, magnetometer, and gyroscope sensors. Figure 6.8 is without bias removal and figure 6.9 is with bias removal. The standard deviation of this method for both biased and non-biased test results were within acceptable limits. Applying bias removal, shown in figure 6.9, significantly improved the estimated orientation results for this method and the standard deviation was much less than tests without bias removal. This method has a much smaller standard deviation than the test results of the accelerometer and magnetometer estimated orientation test results shown in figures 6.4 and 6.5. When compared to the gyroscope orientation test results shown in figures 6.6 and 6.7, the standard deviation for this method is greater, but the estimated orientation did not deviate from the true orientation of the prototype. This method with bias removal is used to estimate the orientation of the prototype.



	azimuth	pitch	roll
SD	0.2308	0.0788	0.0856
SE	0.0103	0.0035	0.0038
CI(90%)	±0.0170	±0.0058	±0.0063

Figure 6.8: Sensor fused calculated orientation



	azimuth	pitch	roll
SD	0.1049	0.0224	0.0477
SE	0.0047	0.0010	0.0021
CI(90%)	±0.0077	±0.0016	±0.0035

Figure 6.9: Sensor fused with bias removal calculated orientation

6.3.4 Orientation Calculation Comparison

The following figures 6.10 through 6.12 compare the test results from the different orientation estimation methods shown in figures 6.4 through 6.9. The same test data used for figures 6.4 through 6.9 is used in figures 6.10 though 6.12. Figure 6.10 compares the three different orientation methods for azimuth. Figure 6.11 compares the three different orientation methods for pitch. Figure 6.12 compares the three different orientation methods for roll. The bolded cyan plot line in figures 6.10 through 6.12 is the sensor fused orientation method with bias removal. This method is the best when compared with the accelerometer and magnetometer method and gyroscope method.

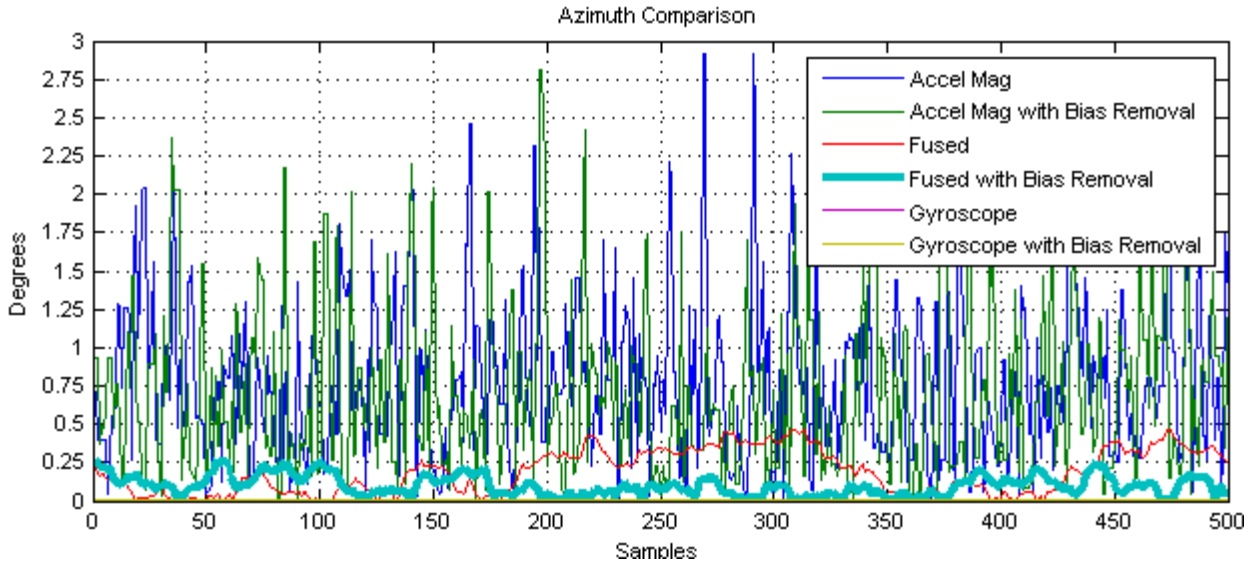


Figure 6.10: Azimuth comparsion

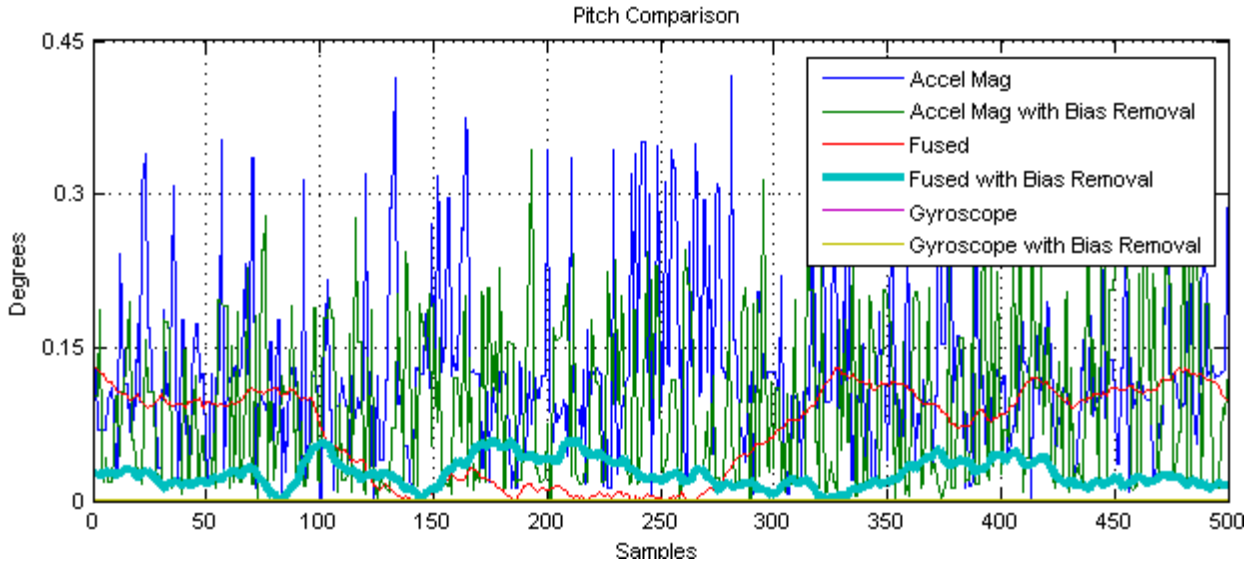


Figure 6.11: Pitch comparison

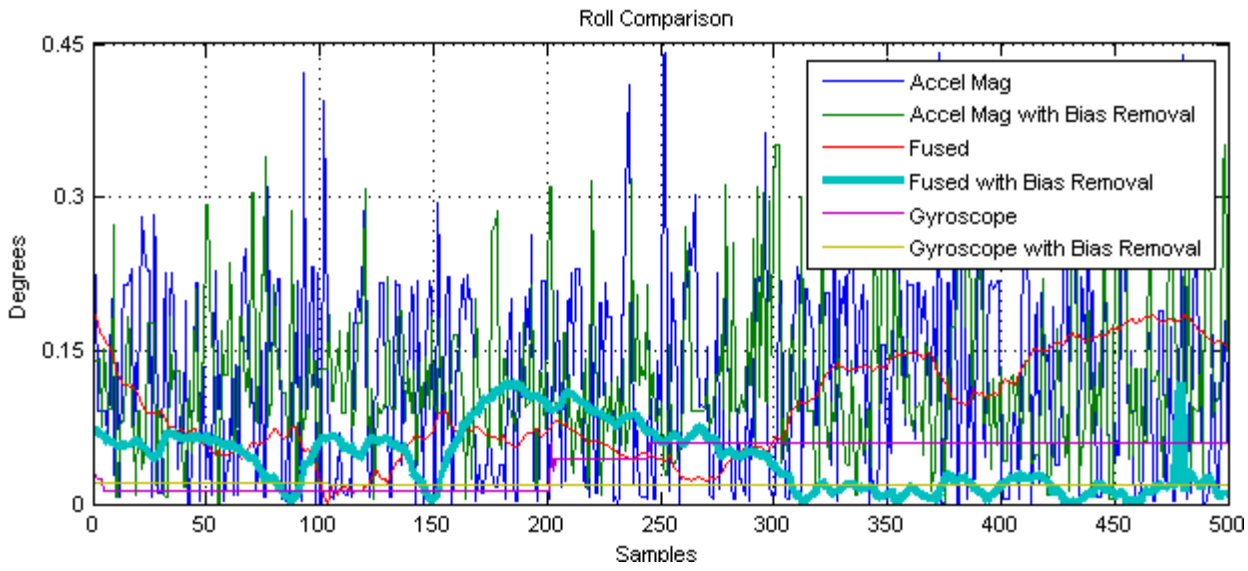


Figure 6.12: Roll comparison

6.4 Received Signal Strength Results

The following two subsections show test results using the prototype to measure *RSS* values at varying distances from four 802.11 wireless access points to compare unfiltered or raw signal strength values with filtered signal strength values. Data was recorded at 50 centimeters, 1 meter, 2 meters, 5 meters, and 10 meters. Each test consists of 100 samples of recorded data. Subsection 6.4.1 contains test results for unfiltered or raw data and subsection 6.4.2 contains test results for filtered data. The same data recorded data is used for both raw and filtered tests.

6.4.1 Raw RSS

Experimental test results for unfiltered or raw data is shown in figures 6.13 through 6.17. Each test is comprised of four wireless access points with 100 samples of data. Figure 6.13 shows test results at 50 centimeters with *RSS* values varying from -40dBm to -52dBm. Figure 6.14 shows test results at 1 meter with *RSS* values varying from -40dBm to -55dBm. Figure 6.15 shows test results at 2 meters with *RSS* values varying from -53dBm to -63dBm. Figure 6.16 shows test results at 5 meters with *RSS* values varying from -52dBm to -69dBm. Figure 6.17 shows test results at 10 meters with *RSS* values varying from -53dBm to -70dBm.

The tests results from measuring the *RSS* at different distances from the wireless access points display the high amount of fluctuation in the signal strengths. This high amount of fluctuation caused erroneous values when converting the *RSS* to distance and had to be filtered. It can also be seen from these tests that as distance from each wireless access point increases, the overall numerical range in received signal strength also increases.

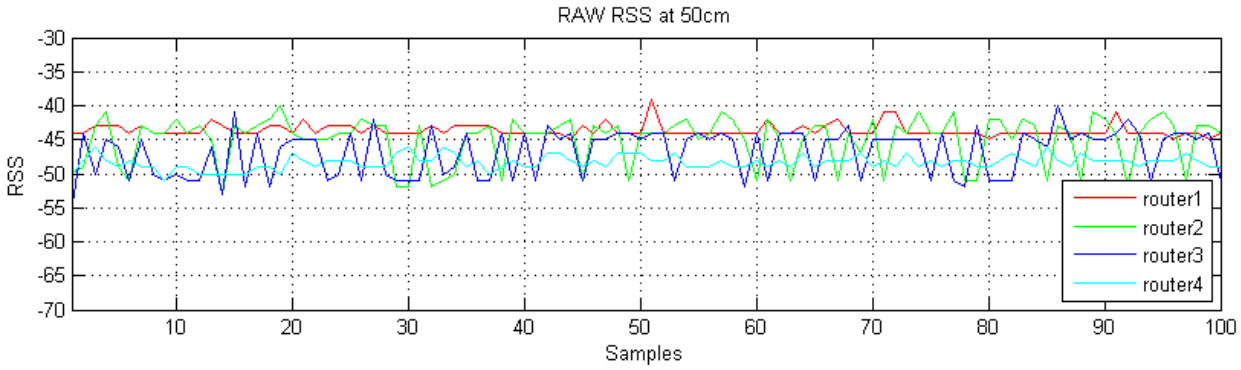


Figure 6.13: Raw RSSI, 50 centimeters distance from all access points

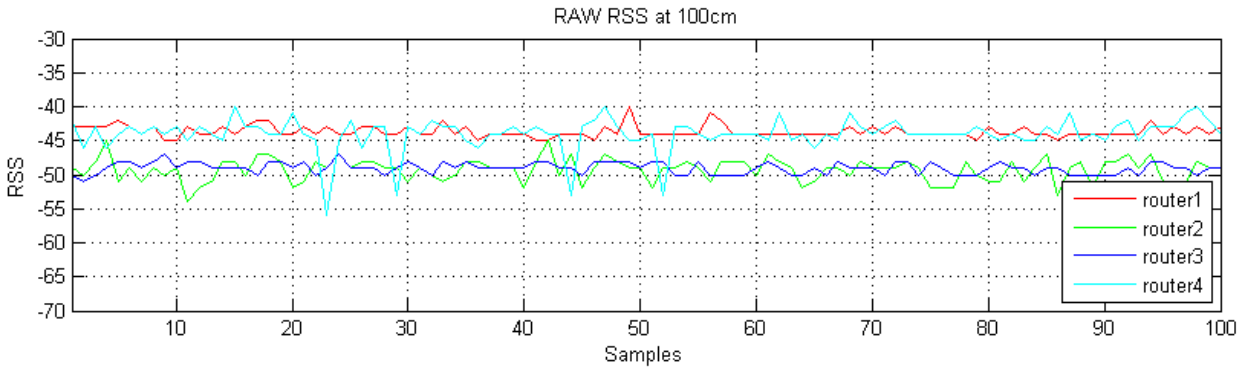


Figure 6.14: Raw RSSI, 1 meter distance from all access points

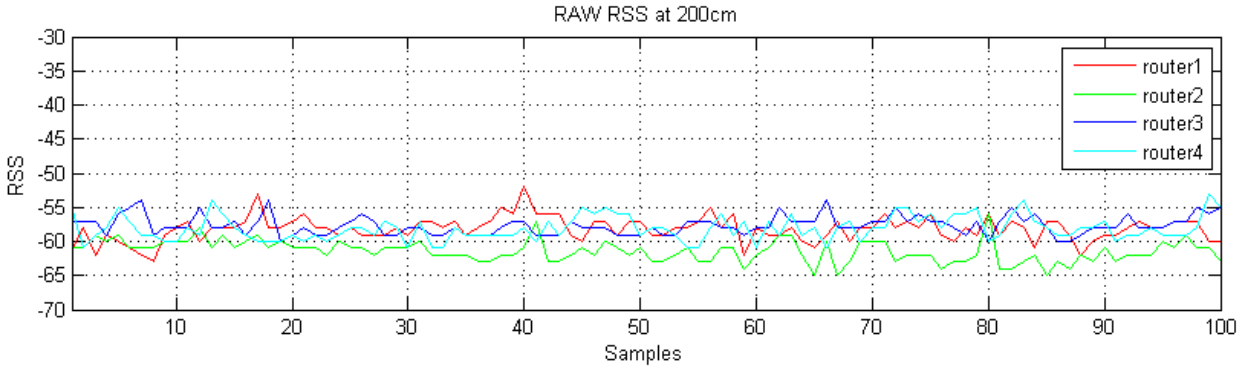


Figure 6.15: Raw RSSI, 2 meters distance from all access points

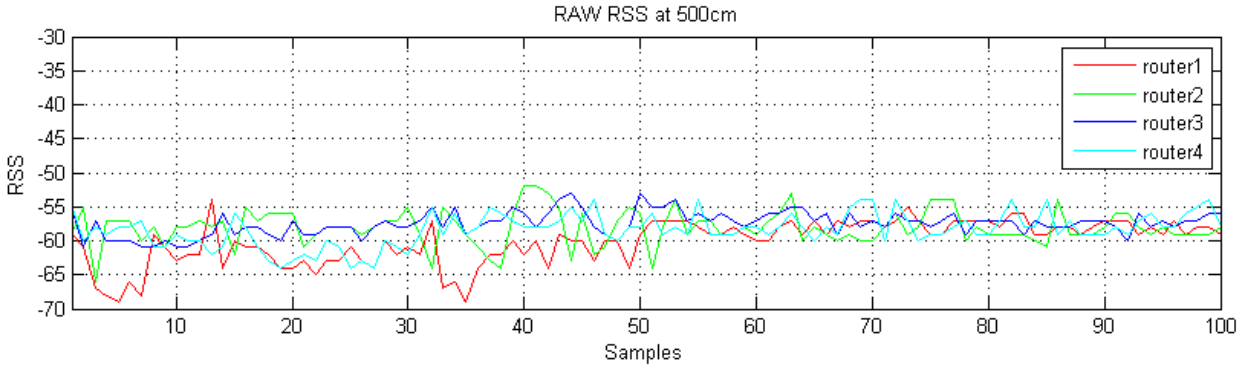


Figure 6.16: Raw RSSI, 5 meters distance from all access points

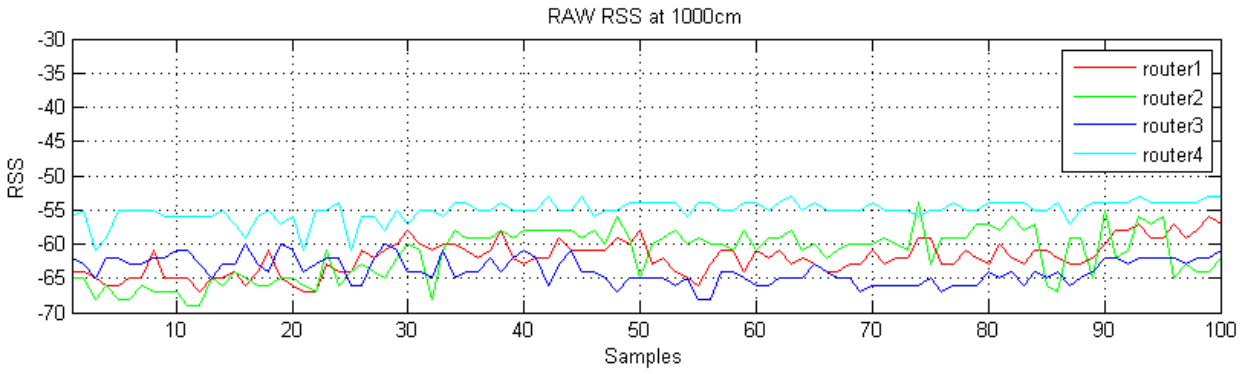


Figure 6.17: Raw RSS, 10 meters distance from all access points

6.4.2 Filtered RSS

Experimental test results using Kalman filtered data is shown in figures 6.18 through 6.22 using the same data from figures 6.13 through 6.17, respectively. The Kalman filter is explained in section 5.4. Kalman filter values for all experimental tests were conducted with $Q = .125$ and $R = 32$.

Figure 6.18 shows test results at 50 centimeters with *RSS* values varying from -44dBm to -49dBm. Figure 6.19 shows test results at 1 meter with *RSS* values varying from -44dBm to -51dBm. Figure 6.20 shows test results at 2 meters with *RSS* values varying from -57dBm to -62dBm. Figure 6.21 shows test results at 5 meters with *RSS* values varying from -57dBm to -63dBm. Figure 6.22 shows test results at 10 meters with *RSS* values varying from -55dBm to -65dBm.

In all test results, filtering the *RSS* values from the wireless access points decreased the numerical range, increased the minimum *RSS* value, decreased the

minimum *RSS* value, and produced overall steadier signal data. Again, it can been seen as in test results in figures 6.13 through 6.17 that as distance from the wireless access points increase, the overall numerical range in received signal strength also increases.

RSS distance conversions using Kalman filtered data improved distance conversions when compared to computing estimated distances using unfiltered or raw *RSS* values. This is due to the much steadier non-sporadic received signal strength data produced from the filter.

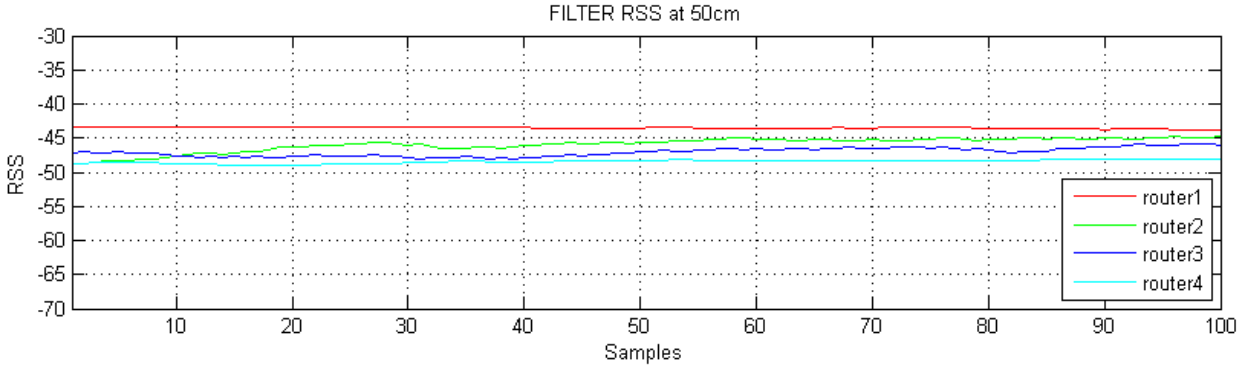


Figure 6.18: Filtered RSS, 50 centimeters distance from all access points

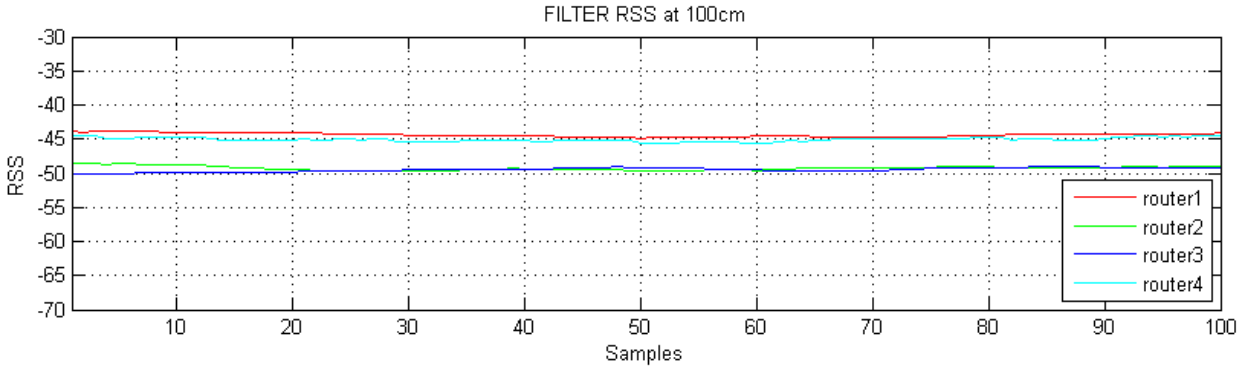


Figure 6.19: Filtered RSS, 1 meter distance from all access points

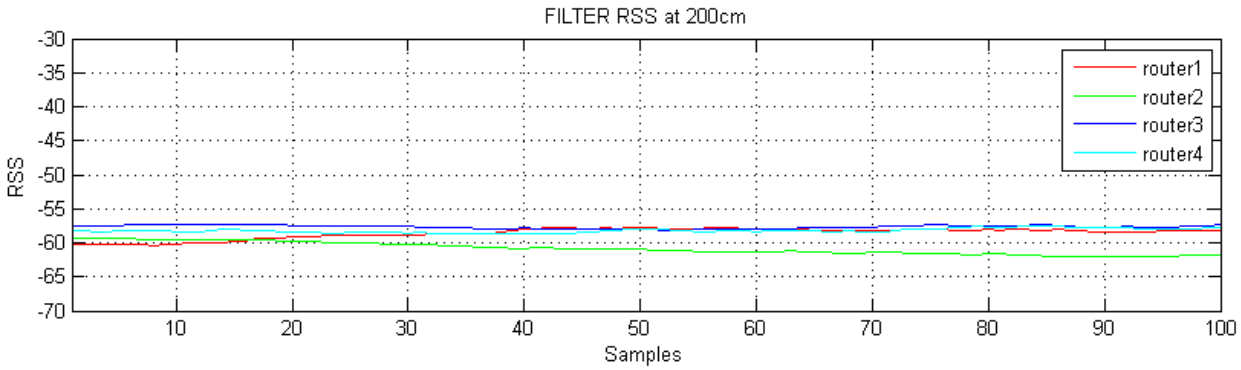


Figure 6.20: Filtered RSS, 2 meters distance from all access points

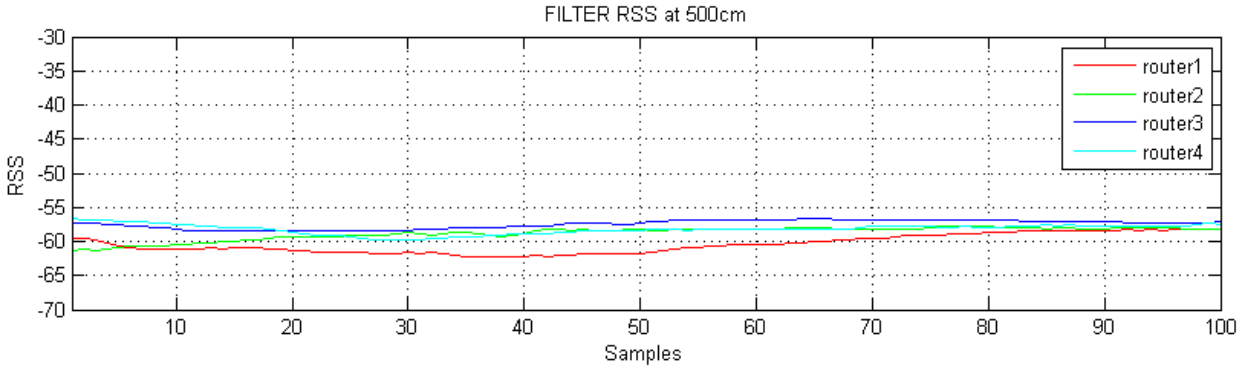


Figure 6.21: Filtered RSS, 5 meters distance from all access points

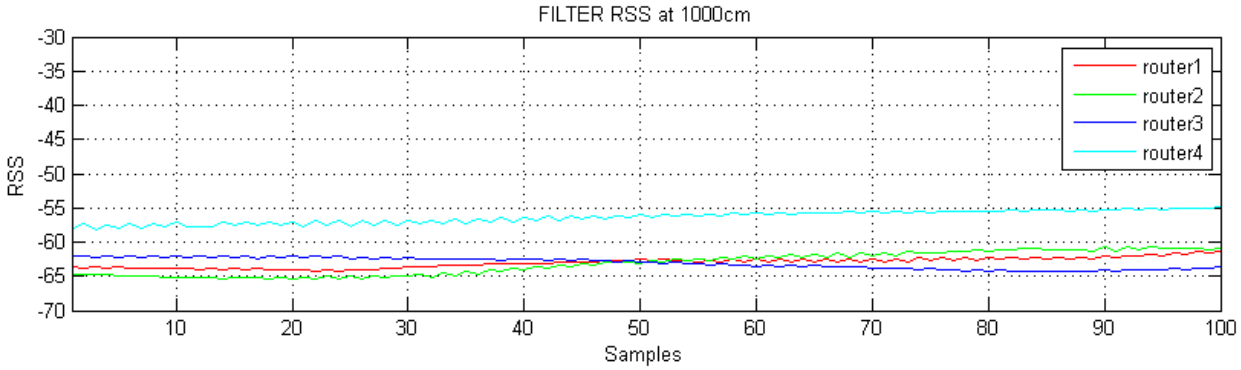


Figure 6.22: Filtered RSS, 10 meters distance from all access points

6.5 RSS Estimated Distance VS Actual Distance

The estimated distance calculated from the *RSS* values using Frii's free space path loss equation is shown in figure 6.23. Table 6.1 contains percent errors of the estimated distances from actual measured distances. The data used in figure 6.23 and table 6.1 is the same test data used in figures 6.18 through 6.22 in the previous subsection.

It can also be seen in figure 6.23 that as distance from the wireless access points increase, the fluctuation in *RSS* values also increase. This correlates directly to the tests results in figures 6.18 through 6.22, and also correlates to test results in figures 6.13 through 6.17 for unfiltered or raw *RSS* values. Percent error of the estimated distances to the real distances appear to decrease as shown in table 6.1 except when *RSS* was measured at 2 meters, but overall deviation of the estimated distances to the real distances increase as the wireless access points move further away from the prototype.

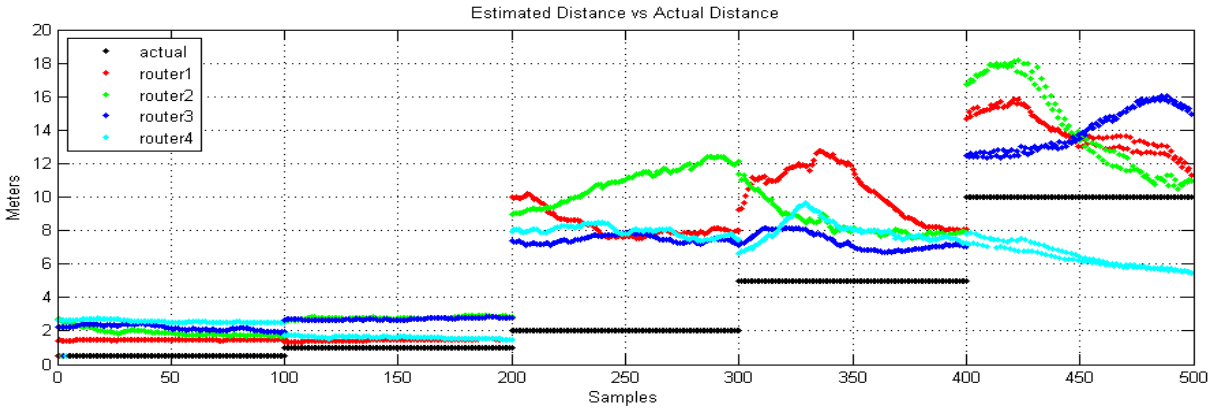


Figure 6:23: Estimated Distance VS Actual Distance measured in meters

	50 cm	1 m	2 m	5 m	10 m
router1	192.8163	45.2607	316.3574	106.3421	38.4594
router2	287.8415	179.24	442.5555	70.3171	42.1488
router3	340.5913	171.7048	273.3680	46.5706	39.7410
router4	416.3006	61.9333	297.5566	59.1920	-35.5219
average	309.39	99.051	221.82	70.605	21.207

Table 6.1: Percent error of estimated distance from actual distance

6.6 Location Estimation Results

The following two subsections contain test results for location estimation comparing raw and filtered *RSS* data. Tests were conducted in an indoor environment that measured 14.0462 meters wide and 17.4498 meters deep. An 802.11 wireless access point was placed in each corner of the indoor environment. Figure 6.24 gives the layout of the test environment and position of the access points and four test locations.

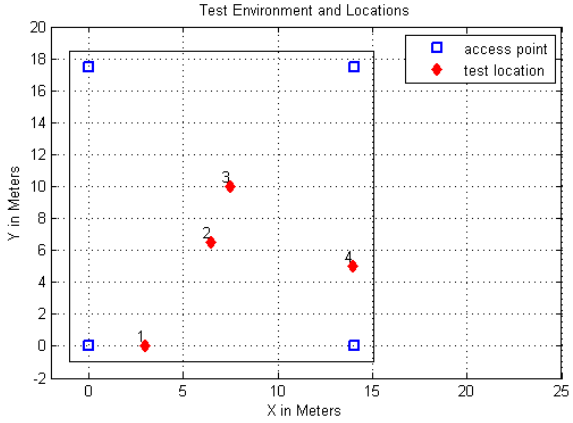


Figure 6.24: Indoor test environment and test locations

All experimental tests were conducted with a threshold value of 1 meter and conducted over 100 received *RSS* samples from all four wireless access points. Weighted locations are calculated from 10 predicted locations. Four tests were conducted at stationary locations at coordinates (3m, 0m), (6.5m, 6.5m), (7.5m, 10m) and (14m, 5m) shown in figure 6.24. The same datasets are used in tests results for each position comparing unfiltered and filtered *RSS* data.

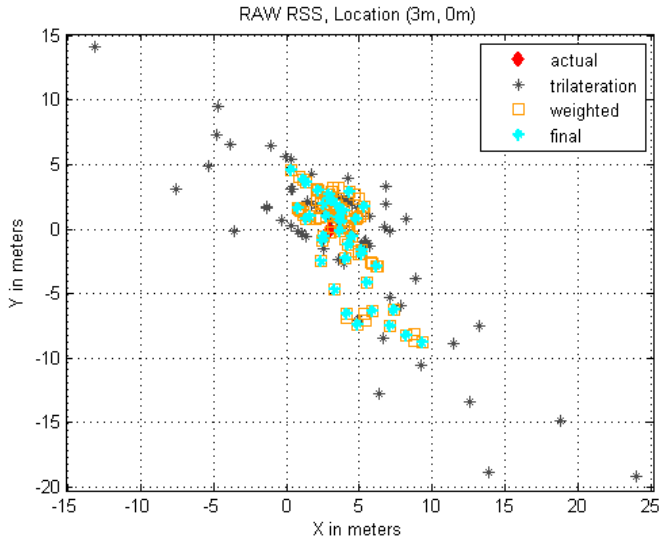
For each test location, the total number of final predicted location updates were counted, the average distance from the actual locations to the estimated final locations were determined, and the standard deviation over all final predicted locations were computed. The percentage of final locations estimated within the threshold limit of 1 meter as well as 2 meters were calculated for each test.

Figures 6.25 through 6.27 contain four symbol types. The red diamond is the location of the prototype. Black circles are multilaterated location estimates. Orange squares are weighted multilaterated location estimates. Cyan diamonds are weighted multilaterated location estimates or final estimated locations bound to a threshold.

6.6.1 Location 1

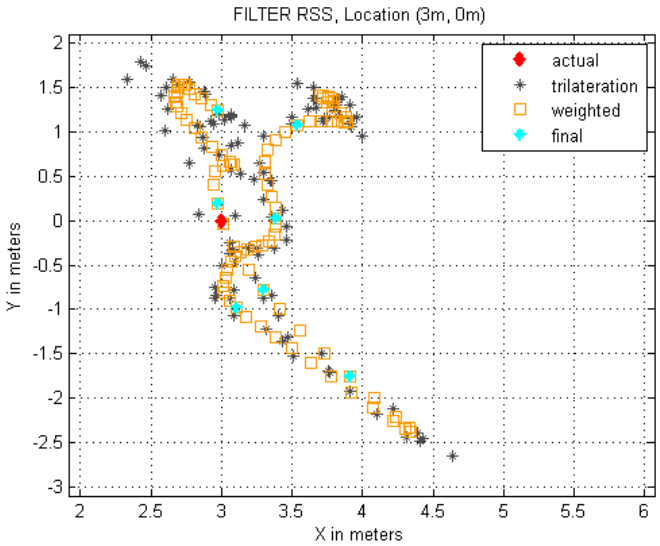
One hundred *RSS* readings were recorded at coordinates (3m, 0m). Figure 6.25 contains unfiltered or raw data and figure 6.26 contains Kalman filtered data. It can be seen from the tests results at location 1 the filtered *RSS* data improved location estimation. The total number of final location updates were significantly smaller for filtered

data. The average distance from the estimated final locations to the real location and standard deviation improved for filtered data. The percentage of estimated final locations within 1 meter and 2 meters improved with filtered *RSS* test data.



	Final Location
Updated #	39
Average Meters	3.5164
SD	2.5994
SE	0.4162
CI(90%)	±0.702
% <= 1 meter	5.1282
% <= 2 meters	38.46

Figure 6.25: Estimated position for location 1, raw data

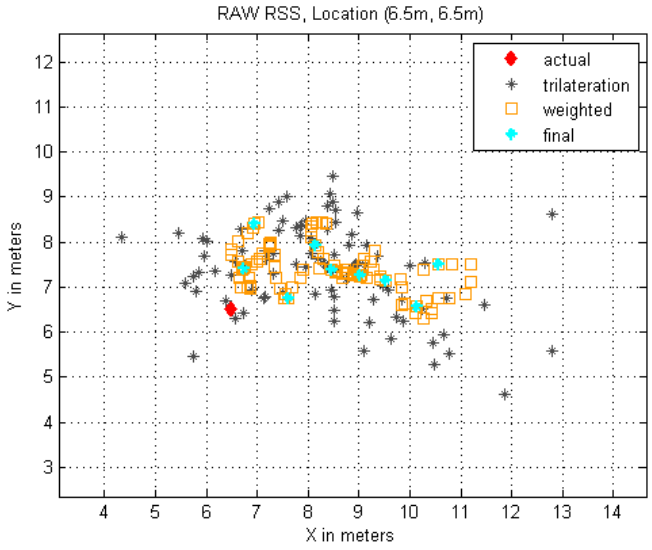


	Final Location
Updated #	7
Average Meters	0.9763
SD	0.5909
SE	0.2233
CI(90%)	±0.434
% <= 1 meter	57.14
% <= 2 meters	100

Figure 6.26: Estimated position for location 1, filter data

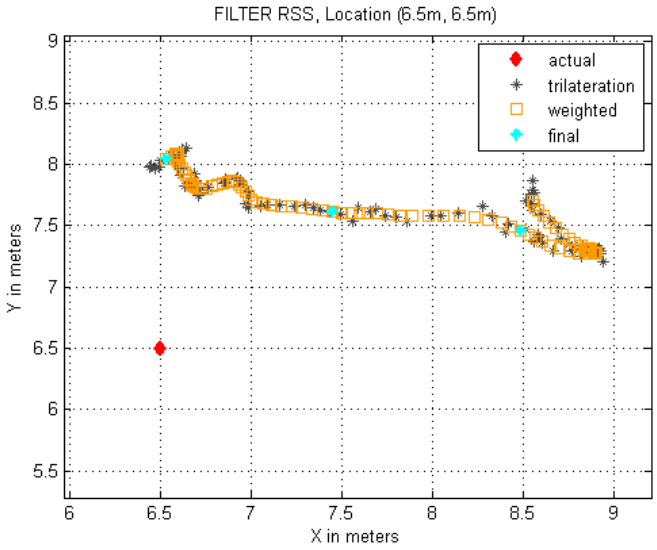
6.6.2 Location 2

One hundred *RSS* readings were recorded at coordinates (6.5m, 6.5m). Figure 6.27 contains unfiltered or raw data and figure 6.28 contains Kalman filtered data. It can be seen from the tests results at location 2 the filtered *RSS* data improved location estimation. The total number of final location updates were significantly smaller for filtered data. The average distance from the estimated final locations to the real location and standard deviation improved for filtered data. The percentage of estimated final locations within 1 meter did not improve, but at 2 meters improvement occurred with filtered *RSS* test data.



	Final Location
Updated #	9
Average Meters	2.4302
SD	1.0725
SE	0.3575
CI(90%)	±0.6648
% <= 1 meter	0
% <= 2 meters	33.33

Figure 6.27: Estimated position for location 2, raw data

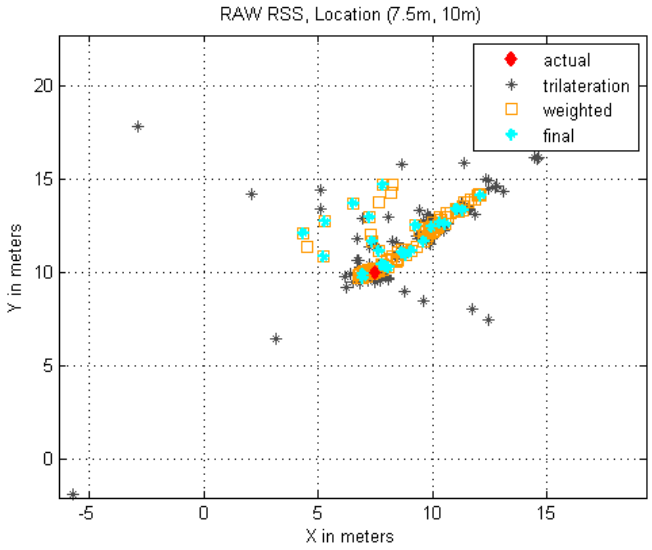


Final Location	
Updated #	3
Average Meters	1.7397
SD	0.4068
SE	0.2349
CI(90%)	±0.686
% <= 1 meter	0
% <= 2 meters	66.66

Figure 6.28: Estimated position for location 2, filter data

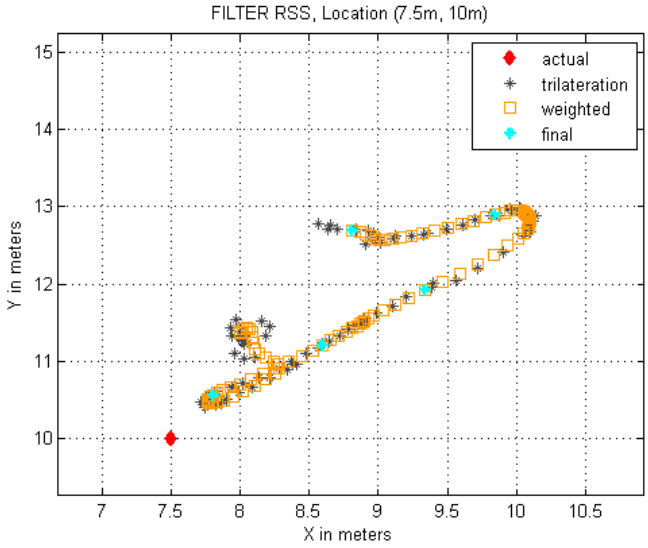
6.6.3 Location 3

One hundred *RSS* readings were recorded at coordinates (7.5m, 10m). Figure 6.29 contains unfiltered or raw data and figure 6.27 contains Kalman filtered data. It can be seen from the tests results at location 3 the filtered *RSS* data improved location estimation. The total number of final location updates were significantly smaller for filtered data. The average distance from the estimated final locations to the real location and standard deviation improved for filtered data. The percentage of estimated final locations within 1 meter and 2 meters slightly decreased with filtered *RSS* test data and raw data performed better.



	Final Location
Updated #	26
Average Meters	2.7520
SD	1.6985
SE	0.3331
CI(90%)	±0.57
% <= 1 meter	23.0769
% <= 2 meters	42.3077

Figure 6.29: Estimated position for location 3, raw data

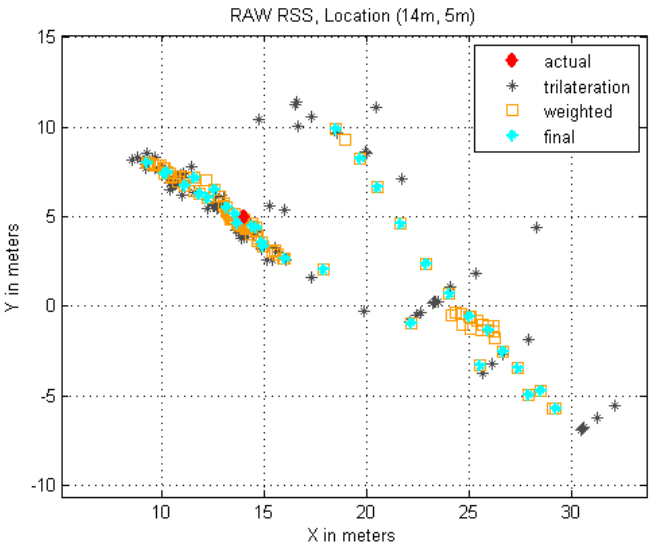


	Final Location
Updated #	5
Average Meters	2.3333
SD	1.2082
SE	0.5403
CI(90%)	±1.1519
% <= 1 meter	20
% <= 2 meters	40

Figure 6.30: Estimated position for location 3, filter data

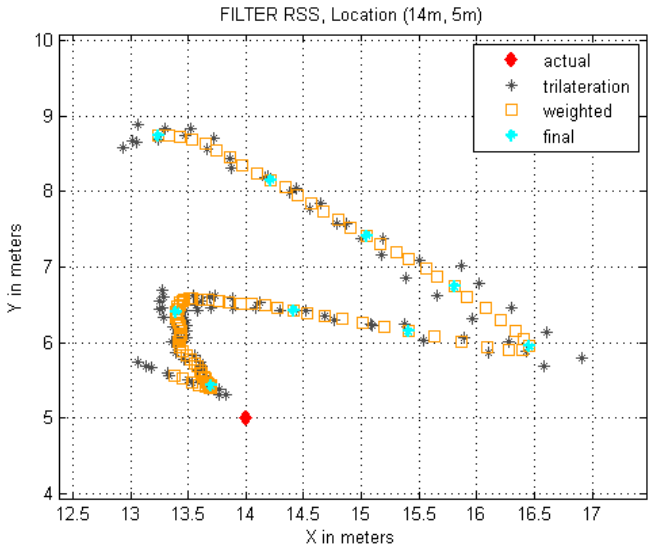
6.6.4 Location 4

One hundred *RSS* readings were recorded at coordinates (14m, 5m). Figure 6.31 contains unfiltered or raw data and figure 6.32 contains Kalman filtered data. It can be seen from the tests results at location 4 the filtered *RSS* data improved location estimation. The total number of final location updates were significantly smaller for filtered data. The average distance from the estimated final locations to the real location and standard deviation improved for filtered data. The percentage of estimated final locations within 1 meter and 2 meters improved with filtered *RSS* test data.



	Final Location
Updated #	32
Average Meters	7.0104
SD	5.7221
SE	1.0115
CI(90%)	±1.7151
% <= 1 meter	5.556
% <= 2 meters	16.667

Figure 6.31: Estimated position for location 4, raw data



	Final Location
Updated #	9
Average Meters	2.2361
SD	0.9922
SE	0.3307
CI(90%)	±0.615
% <= 1 meter	11.11
% <= 2 meters	44.44

Figure 6.32: Estimated position for location 4, filter data

7 Conclusions

7.1 Conclusion

Sensor fusion with bias removal produced the best results for orientation estimates. The estimated azimuth angle contained the highest amount of variance over successive sensor readings for all test incorporating the magnetometer sensor. This was because the magnetometer had the highest variance in the raw sensor data. Tests involving the gyroscope contained the least amount of variance over successive sensor readings, but also contained gyroscopic drift misaligning the orientations of the real and virtual worlds over time.

Sensor fused orientation estimates with bias removal performance was the best of out all three methods tests. Maximum variance of the true orientation of the prototype versus the estimated orientation was less than 1 degree over successive orientation

estimates and orientation changes are smooth. The prototype orientation remained aligned to a global reference. The sensor fused orientation tracking method is acceptable and can be used for further development of the prototype.

Estimating the location of the prototype when stationary can be computed somewhat accurately. The goal of less than one meter of the estimated location to the actual location cannot be met using 802.11 wireless network access points. When the prototype is moving location estimation is very limited and unacceptable. This is due to the high amount of noise in the *RSS* for each of the 802.11 wireless access points. When testing was conducted, variances could be seen in the *RSS* when the prototype remained stationary and was held in the hand of a user in different ways. Stabilizing the *RSS* using a Kalman filter introduced a small amount of lag for location estimation updates, but the number of overall final location estimates over time were greatly reduced when compared to using unfiltered or raw *RSS* data for location estimates. The physical space needed for this method to work optimally is about 10 to 20 meters and many indoor environments do not accommodate this physical size. Different technology must be explored to achieve more reliable and steady *RSS* readings.

7.2 Future Works

There are many possible directions to extend the research described within this thesis. First, it would be interesting to compare the received signal strength variance and

accuracy using different technologies such as Bluetooth or other wireless based network hardware. Secondly, incorporating prototype orientation into the multilateration method described should be investigated since *RSS* values highly decrease when the prototype is not facing the wireless access point. Finally, the research completed here can be used to continue to develop a prototype device to allow a user to experience a virtual reality experience using common high-tech devices.

References

[1] F. Zhou, H.-L. Duh and M. Billinghurst, "Trends in augmented reality tracking, interaction and display: A review of ten years in ISMAR," *Mixed and Augmented Reality,* pp. 193-202, 2008.

[2] R. Azuma, "A Survey of Augmented Reality," *Presence: Teleoperators and Virtual Environments,* vol. 6, no. 4, pp. 355-385, 1997.

[3] H. Benko, E. Ishak and S. Feiner, "Collaborative Mixed Reality Visualization of an Archaeological Excavation," *Proceedings of the International Symposium on Mixed and Augmented Reality,* vol. 3, pp. 132-140, 2004.

[4] S. Yuen, G. Yaoynuneyong and E. Johnson, "Augmented Reality: An Overview and Five Directions for AR in Education," *Journal of Educational Technology Development and Exchange,* vol. 4, no. 1, pp. 119-140, 2001.

[5] D. Zhang, F. Xia, Z. Yang, L. Yao and W. Zhao, "Localization technologies for indoor human tracking," *Proceedings of the 5th International Conference on Future Information Technology,* vol. abs/1003.1833, 2010.

[6] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *Computer,* vol. 34, no. 8, 2001.

[7] H. Liu, H. Darabi, P. Banerjee and J. Liu, "Survey of Wireless Indoor Positioning Techniques and Systems," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions,* vol. 37, no. 6, pp. 1067-1080, 2007.

[8] Atlas RFID, "Active RFID vs. Passive RFID," 2014. [Online]. Available: http://atlasrfid.com/auto-id-education/active-vs-passive-rfid/. [Accessed 10 October 2014].

[9] O. J. Woodman, "An introduction to inertial navigation," University of Cambridge, Cambridge, 2007.

[10] A. Bahillo, S. Mazuelas, R. M. Lorenzo, P. Fernandez, J. Preito and E. J. Abril, "Indoor Location Based on IEEE 802.11 Round-Trip Time Measurements with Two-Step NLOS Mitigation," *Progress In Electromagnetics Reasearch,* vol. 15, no. B, pp. 285-306, 2009.

[11] C. Hoene and J. Willmann, "Four-way TOA and Software-Based Trilateration of IEEE 802.11 Devices," [Online]. Available: http://www.net.in.tum.de/fileadmin/bibtex/publications/papers/hoene_pimrc2008.pdf. [Accessed 5 3 2014].

[12] Google, Inc., "Developer Android," [Online]. Available: http://developer.android.com/tools/help/adb.html.

[13] "Linux Wireless," [Online]. Available: http://wireless.kernel.org/en/users/Documentation/iw#About_iw.

[14] j. l. inder123, "google-gson - A Java library to convert JSON to Java objects and vice-versa," [Online]. Available: https://code.google.com/p/google-gson/.

[15] J. Wetherell, "android-quick-response-code Android QR Code Decoder and Encoder," [Online]. Available: https://code.google.com/p/android-quick-response-code/.

[16] J. ". Jongma, "How-To SU Guidelines for problem-free su usage," [Online]. Available: http://su.chainfire.eu/.

[17] D. Dromereschi, "achartengine Charting library for Android," [Online]. Available: https://code.google.com/p/achartengine/.

[18] peter.ab...@gmail.com, "efficient-java-matrix-library A fast and easy to use dense matrix linear algebra library written in Java.," [Online]. Available: https://code.google.com/p/efficient-java-matrix-library/.

[19] G. Watson, "OrmList - Lightweight Java ORM Spupports Andoird and SQLite," [Online]. Available: http://ormlite.com/sqlite_java_android_orm.shtml.

[20] S. Dey, "SD-EQR: A New Technique To Use QR Codes™ in Cryptography".

[21] F. C. Moon, Applied Dynamics, Wiley-VCH, 2008.

[22] CHRobotics LLC., "Understanding Euler Angles," [Online]. Available: http://www.chrobotics.com/library/understanding-euler-angles.

[23] G. G. Slabaugh, *Computing Euler angles from a rotation matrix.*

[24] S. O. Madgwick, A. J. Harrison and R. Vaidyanathan, *An efficient orientation filter for inertial and inertial/magnetic sensor arrays,* 2010.

[25] J. Miller, *An Introduction to Quaternions and their Applications to Rotations in Computer Graphics,* 2006.

[26] D. Ren, L. Wu, M. Cui, Z. You and M. Hu, "Design and Analyses of a MEMS Based Resonant Magnetometer," *Sensors,* 2009.

[27] D. L. Hall, "An introduction to multisensor data fusion," *Proceddings of the IEEE,* pp. 6-23, 1997.

[28] A.-J. Baerveldt and R. Klang, "A Low-cost and Low-weight Attitude Estimation System for an Autonomous Helicopter," *Proceedings of IEEE International Conference on Intelligent Engineering Systems,* pp. 391-395, 1997.

[29] S. Colton, *The balance filter - a simple solution for integrating accelerometer and gyroscope measurements for a balanceing platform,* 2007.

[30] "List of WLAN channels," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/List_of_WLAN_channels. [Accessed 13 August 2014].

[31] O. O.S., O. V.N., A. Abe and O. B.O., "Outdoor Localization System Using RSSI Measurement of Wireless Sensor Network," *International Journal of Innovative Technology and Exploring Engineering,* vol. 2, no. 2, pp. 1-6, January 2013.

[32] J. Bardwell, The Truth About 802.11 Signal and Noise Metrics, Connect802 Corporation, 2004.

[33] S. H. Ahmed, S. H. Bouk, N. Javaid and I. Sasase, *Combined Human, Antenna Orientation in Elevation Direction and Ground Effect on RSSI in Wireless Sensor Networks.*

[34] G. Deak, K. Curran and J. Condell, "Filters for RSSI-based measurements in a Device-free Passive Localisation Scenario," *International Journal of Image Processing and Communications,* pp. 23-34, 2011.

[35] L. Kleeman, "Understands and Applying Kalman Filtering," [Online]. Available: http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf.

[36] N. K. Sharma, "A Weighted Center of Mass based Trilateration Approach for Locating Wireless Devices in Indoor Environment," [Online]. Available: http://www.cl.cam.ac.uk/research/srg/netos/sla/mobileman/d15/papers06/238_2_Trilateration.pdf.

[37] V. Kecman, Personal Communication, 2014.

[38] V. Kecman, Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models, Cambridge: MIT Press, 2001.

[39] J. Blumenthal, R. Grossmann, F. Golatowski and D. Timmermann, "Weighted Centroid Localization in Zigbee-based Sensor Networks," *Intelligent Signal Processing,* pp. 1-6, 2007.

[40] F. Hamza-Lup, J. Rolland and C. Hughes, "A Distributed Augmented Reality System for Medical Training and Simulation," *Energy, Simulation-Training, Ocean Engineering and Instrumentation: Research Papers of the Link Foundation Fellows,* vol. 4, pp. 213-235, 2204.