DEAD RECKONING ALGORITHMS FOR INDOOR LOCALIZATION

Submitted by

ZHANG YU

Department of Electrical & Computer Engineering

In partial fulfillment of the requirements for the

Degree of Master of Engineering

National University of Singapore

Jan 2012

Abstract

The recent development of smart phones and network has given rise to an increasing demand of location based services which can be easily catered by Global Positioning System (GPS) in outdoor environments, especially in rural areas. However, in urban zones with tall buildings, and particularly in indoor environments, the GPS signals are not reliable or even not available. To provide a reliable and robust localization method for such environments, various technologies have been investigated by researchers and engineers, such as dead reckoning method by steps counting, WLAN fingerprinting, ultra-wide band method, Cricket, etc. In this project, three dead reckoning algorithms (DRAs), DRA-I, DRA-II, DRA-III, have been proposed and investigated.

Accelerometer and gyroscope are applied in Phase1 of DRA-I and is based on a strap down technique to provide displacement direction. Given the displacement direction from Phase1 and stride lengths estimated by step counting algorithm, Phase2 estimates the pedestrian's position and velocity, which will then be provided to Phase1 as measurements. Given an initial condition, it can accurately estimate the pedestrian's position for the first 10-15 meters. It then becomes dysfunctional because DRA-I lacks a robust attitude estimation capability.

An attitude algorithm based on gyroscope, magnetometer and accelerometer has been investigated. Although it provides acceptable estimation while the device is stationary, estimation errors of the device's roll and pitch angles have large errors if the pedestrian is walking. It is adapted in DRA-II, which contains two steps. The calibration step is to calculate the device's initial state. The step tracking contains

two phases. Phase1 is the same as that of DRA-I. Phase2 compares the device's status in the tracking mode and the initial state and estimate the device's position and velocity with the help of the step counting algorithm. As the device's roll angle and pitch angle are not accurately estimated, outputs of Phase1 are unsatisfactory, while position estimations from Phase2 are accurate.

Attitude estimation can be further improved by applying the novel magnetometer calibration proposed by [32], which is adapted in DRA-III. DRA-III indeed provides better attitude and position estimation and is adequate to be implemented for real time tracking purpose.

In summary, due to insufficient accurate attitude estimation, strap down techniques are not adequate to use in pedestrians' tracking problem. Instead, DRA-II and DRA-III, which estimate stride lengths by step counting algorithm, provide precise and robust position estimations. They are not complete solutions for the indoor localization problem, as correction technologies are not included. However, these algorithms can be easily integrated into other technologies and therefore the effort has built solid foundations for further extensions.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor Professor Lawrence Wong, for his guidance and continuous support throughout the project. I was deeply influenced by his enthusiasm, motivation and immense knowledge, which have greatly inspired me.

Besides my supervisor, my sincere gratitude shall also be extended to Dr. Jin Yunye, Phd Candidates Meng Xiaoli, Bao haitao, Wang Ke and Wang Lei for their valuable advices and insightful comments.

I would also take this opportunity to express my appreciation to my lab mates, Song Xianlin, Li Kang, and Guo Jie for their generous help, continuous support and encouragement.

I wish to thank National University of Singapore for granting me the opportunity to benefit from many erudite and warm-hearted professors.

In particular, I would thank my families who have always been supportive, patient and encouraging.

List of Figures

List of Tables

LIST OF SYMBOLS AND ABBREVIATIONS

LBS	Location based services
DRA	Dead Reckoning algorithm
DRA-I	Proposed Dead Reckoning Algorithm I
DRA-II	Proposed Dead Reckoning Algorithm II
DRA-III	Proposed Dead Reckoning Algorithm III
GPS	Global Positioning System
TOA	Time Of Arrival
TDOA	Time Difference Of Arrival
RSS	Received Signal Strength
CP	Calibration Point
AP	Access Point
PDF	Probability Density Functions
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
SSID	Service Set ID
API	Application Programming Interface
SDK	Software Development Kit
GAE	Google App Engine
$\hat{x}_{k\|k}$	A posteriori state estimate at time *k* given observations up to and including at time k
$P_{k\|k}$	A posteriori error covariance matrix
A_k	The state transition model which is applied to the previous state x_{k-1}
B_k	The control-input model which is applied to the control vector u_k
w_k	The process noise which is assumed to be drawn from a zero

	mean multivariate normal distribution with covariance Q_k
Q_k	The covariance of w_k
z_k	The observation vector
C_k	The observation model which maps the true state space into the observed space
v_k	The observation noise
R_k	The covariance of v_k
X	State vector
$\begin{bmatrix} x & y & z \end{bmatrix}^T$	Samsung tab's position vector in the global coordinate
$\begin{bmatrix} u & v & w \end{bmatrix}^T$	The velocity vector in the global coordinate
$\begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$	The acceleration vector in the body coordinate
$\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$	The quaternion vector in the global coordinate
$\begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$	The angular velocity vector in the body coordinate
$\begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T$	The magnetometer measurement vector in the body coordinate
$\begin{bmatrix} m_{x0} & m_{y0} & m_{z0} \end{bmatrix}^T$	Magnetic field strength reference in the global coordinate
$\begin{bmatrix} g_{x0} & g_{y0} & g_{z0} \end{bmatrix}^T$	Gravitational acceleration reference in the global coordinate
$\begin{pmatrix} \gamma_x & \gamma_y & \gamma_z \end{pmatrix}$	Semi-axis lengths of an ellipsoid
$\begin{pmatrix} b_x & b_y & b_z \end{pmatrix}$	Center position of an ellipsoid
R_{rot}	The rotation matrix from the body coordinate to the global coordinate
g	Gravity acceleration

h	Geomagnetic field strength
A	State transition matrix
W	Model noise matrix
A_{ref}	The latest previous peak or vale value of the acceleration magnitude
A_{direc}	A Boolean number to note down whether A_{ref} is peak or vale
τ	Constant, τ=1 in this project
P1	Phase 1 of dead reckoning
P1P2	Combination of Phase 1 and Phase 2 of dead reckoning

CHAPTER 1 INTRODUCTION

1.1 Introduction

The recent fast development of smart phones and deployment of 3G network and hotspots in urban areas have boosted the demand for location based services. Location based services (LBS) are information services accessible with mobile devices through the mobile network and utilizing the ability to make use of the location of the mobile device [2]. LBS are useful in many areas, travel, social networking, location tracking, etc. On the one hand, LBS grants users the ability to easily access more information, for example, the GPS based Google map, with which people can easily find nearby restaurants, gas stations, places of interest, etc. On the other hand, LBS also helps the service providers to improve their services. For example, a cycle Hire application helps its users to find bicycles and rental locations in London by GPS equipped smart phones [1].

Currently, Google Maps, and other similar GPS navigation systems are playing an important role in global positioning; however, their performance is far below customer's expectation for localization in urban areas and indoor environment under which circumstances GPS signal is not reliable or even available. Therefore, to provide robust and reliable localization in indoor environment, more advanced technologies must be applied. In the past years, various localization technologies suitable for indoor environments have been investigated by the researchers and engineers.

1.2 Literature Review

Various technologies have been investigated to estimate the location in indoor environments and the majority of them use solely the wireless signals GSM [11], Wi-Fi [4] [5] [6] [7] [8], Bluetooth [3], etc.

As a result of the rapid increase of Wi-Fi access points and devices embedded with Wi-Fi-detection capabilities, Wi-Fi-based indoor localization are widely studied on location fingerprinting; triangulation-based Time of Arrival (TOA), Time Difference of Arrival (TDOA), RSS, and etc.[4]. The principal of TOA or TDOA is to calculate the distance from the object or people to at least three base stations, and apply a triangulation method to estimate the location which is not practical in terms of precise estimation because multipath rich-scattering in non-line-of sight indoor environments makes it difficult to identify the direct path [5]. Besides, the RF signals travel at the speed of light; therefore, even 1 nanosecond's error in time will lead to a 3 meters' error in estimation. Received signal strength based triangulation (RSS) is a powerful algorithm but it encounters strong propagation loss due to shadowing and signal scattering [5]. Moreover, strong interferences from non-relevant access points or cordless phones, multipath mitigation, etc. also limit the use of these techniques [5].

Wi-Fi fingerprinting based on Received signal strength is another technique developed for indoor localization [4] [5] [6] [7] [8]. A fingerprinting technique usually involves two phases, namely, off-line phase (training phase) and online phase. During the offline phase, RSS values from different Wi-Fi access points are measured

at pre-selected positions which are usually termed calibration points (CPs). The measured RSS values, with the CPs will eventually form a database which is called radio map. During the on-line phase, or estimation phase, new measured RSS values are compared with the information stored in the radio map for location estimation. The estimation methods can be categorized in pattern recognition and probabilistic algorithms [9]. In the training phase, the pattern recognition algorithm assigns a pattern vector containing the average RSS values of all the channels of access points (APs) in the defined area to the correspondent CP. In the estimation phase, usually, K-nearest neighbor algorithm is applied [7]. Given the new measured RSS values vector, its K nearest neighbors, whose pattern vectors achieve the least vector distance are classified as neighborhoods. The estimated position is just the average of all the K CPs' coordinates [9]. By applying probabilistic algorithm, RSS values from each AP are summarized to probability density functions and each CP in the radio map has the probability density functions (PDFs) of all the measured RSS of each AP. In the estimation phase, Bayes' rule with maximum likelihood criteria can be applied to estimate the position. Helena et al [9]. has done a comparison between probabilistic and pattern recognition algorithm and concludes that pattern recognition achieves an accuracy of 10 meters in hall-like open areas, while the probabilistic algorithm's accuracy is 6-7 meters.

To overcome the drawbacks of Wi-Fi fingerprinting, such as massive calculation and time consumption during the training phase [9], Hyuk Lim et al. [10] have proposed a zero-configuration system in which signals emitted by routers are measured by other

routers. As the relative positions of these routers are known, a state matrix is created and updated in real time and thus an accuracy of 3m can be achieved.

Apart from Wi-Fi signals, other wireless signals have also been investigated [3] [11].

Veljo et al. [11] have examined the feasibility of using GSM signals by means of using wide signal-strength fingerprints. The advantages of GSM signals are wide coverage, "wide acceptance of cellular phones", and dispersed cellular towers, which will ensure usual performance of indoor localization even if a building's electricity had failed. The fingerprinting results of GSM signals showed strong spatial variability and signal strength's time consistence. What's more the GSM signals are more stable than 802.11 signals that use the unlicensed overcrowded 2 GHz band and consequently are subjected to strong interference from nearby hotspots or cordless phones, according to Veljo [11].

Apart from the wireless signals, the inertial navigation system (INS) is also widely studied for indoor localization. Dead reckoning method using accelerometer and gyroscope are widely investigated as well. As an inertial measurement unit (IMU) only provides relative information, and the errors of dead reckoning method can accumulate fast with time, they are always combined with other positioning systems, such as Wi-Fi fingerprinting, to correct the accumulated errors [12][13][14][15][16].

Wang Hui et al. [14] proposed a fusion framework by combining WLAN fingerprinting, IMU and Map information, based on particle filters. An IMU accelerometer is used for step counting to calculate the walking distances that together

with Map information, and Wi-Fi fingerprinting, are filtered to estimate the actual position.

To further improve the calibration qualities, Frank et al. [16] investigated a shoe-mounted navigation system [16]. They mounted a sensor unit of accelerometer and gyroscope on shoes and use extended Kalman filter to estimate the position and attitude. The advantage of the shoe-mounted system is to use the rest phases [15] during the walking process to trigger zero-velocity (virtual) measurements to update the filter. The magnetometer is applied to correct the heading drift errors.

For hand-held devices, [13] makes an assumption that the device's yaw direction aligns with pedestrian's heading direction by restricting the way to hold a HTC G1device. Thus, the estimation method is to take the average of yaw readings given by a digital compass equipped on the device, from the starting point to the ending point of each step. Together with stride length estimation by the step-counting algorithm, the pedestrian's location can be estimated.

For the aforementioned dead reckoning algorithms, stride lengths can always be accurately estimated by step counting algorithms; the heading direction is more difficult to deal with. [12][13][15][16] estimate the heading direction by setting restrictions on the device, whether by mounting the device on shoes or by restricting the way to hold the device. In this project, one main objective is to release these restrictions and therefore, other existing techniques [24][25][26][27][28][29] for the device attitude estimation have been reviewed.

Kim and Golnaraghi [24] proposed an algorithm to estimate a device, which can perform three degrees-of-freedom rotations, using only an accelerometer and a gyroscope. The state vector in this algorithm includes the quaternion vector, angular velocity and gyroscope drift error. The measurements are signals from the accelerometer and gyroscope. As no translational movement is assumed possible, the accelerometer's measurement is just gravity force with white Gaussian noise. On the other hand, the gyroscope's measurement is the combination of the device's angular velocity with gyroscope's drift error. The algorithm applied an extended Kalman filter as the fusion filter. Although their experiments showed that the device quaternion was stable for 60s. However, taking into account that a high performance gyroscope (zero drift over a one-hour test, noise rating $0.05\deg/s\sqrt{hz}$) is used and the device has no translational movement, it may not be stable any more in the pedestrian localization situation.

[26] proposes an algorithm for Low- Earth-Orbit (LEO) gyro-less satellite's attitude estimation by magnetic vector. However, other sensors like gravity sensor or sun sensors are necessary for accurate attitude estimation.

[25] investigates a more stable algorithm by adding magnetometer measurements. Thus, the state vector has seven states, four from the quaternion vector and three from angular velocity. The measurement vector has acceleration, angular velocity and magnetic strength in each direction, which adds up to 9 states in total. However, as the state vector and measurement vector are not linearly related, unscented Kalman filter

applied to the fusion algorithm. The accelerometer's measurement is assumed as gravitation force corrupted by white Gaussian noise.

QUaternion ESTimator(QUEST) [29] is a popular method that has been widely used. It uses a triad of accelerometers and magnetometers to measure gravity and local magnetic field, and the measured vectors are compared to reference vectors in order to determine the orientation. The reference vectors are usually assumed to be constant. In other words, the acceleration is only due to gravity force and the magnetic field is constant. However, in indoor environments, where there are ferrous objects and electrical interference, the magnetic field varies. On the other hand, while the pedestrian walks, the measured acceleration is a combination of downward gravity and horizontal acceleration.

To reduce the effect of varying indoor magnetic field on attitude determination and computational expense, Yun et al. [30] proposed the Factored Quaternion algorithm (FQA). A triad of accelerometers and magnetometers are orthogonally mounted and their outputs are decoupled. The output of the magnetic field is only used to determine the azimuth direction and thus minimizes the influences from the local magnetic field in yaw and roll estimation. Computational complexity is also effectively reduced. The experiments conclude that FAQ can provide nearly identical accuracy as the QUEST algorithm.

1.3 Project Objectives

The objective of this project is to investigate the robustness of several dead reckoning algorithms which can be corrected by other localization methods, such as Wi-Fi fingerprinting and error compensating or correction measures for accurate pedestrian localization in indoor environments on a hand-held device like smart phones and tablet PC.

CHAPTER 2 SYSTEM DESCRIPTON AND EXPERIMENTAL METHODOLOGY

In section 2.1, the implementation platform of this project, including a brief introduction to the Samsung Galaxy Tab and Android system is presented. Two important APIs and Google App Engine are also briefly introduced. In section 2.2, sensor signals that are available on a Samsung Tab are examined. In section 2.3, global coordinate is defined and geomagnetic field strengths with respect to the global coordinate' three axes are presented.

2.1 Introduction to Platform

2.1.1 Platform Requirements

To implement proposed algorithms, the basic requirement is a hand-held device with a programmable operating system, embedded with accelerometer, gyroscope, and magnetometer for the dead-reckoning purpose, and Wi-Fi detection unit for Wi-Fi fingerprinting. Wi-Fi fingerprinting requires the information of the access points' Service Set ID (SSID) and their signal strength levels. Besides, as the proposed algorithms involve intensive computation, a high-performance and energy efficient processor is required.

According to the aforementioned requirements, existing smart phones or tablet PCs with proper Application Programming Interface (API) for the implementation of the proposed algorithm were targeted. Apple's IPhone 4 has an excellent hardware foundation for the implementation. However, Apple's public Wi-Fi APIs are only restricted to a few names and the API is kept as private without documentation.

Therefore, the open source Android system is the only choice for this project and the Samsung Tab that is then the only Android device embedded with a gyroscope is chosen.

2.1.2 Brief on Samsung Galaxy Tab

The Samsung Tab is an Android-based compact tablet PC, running Android 2.2 Froyo. The open source Android operating system gives us the freedom to access its hardware, such as gyroscope, accelerometer and Wi-Fi detector.

The Samsung Tab has a 7-inch touch screen, 1 GHz ARM Cortex-A8 processor, Wi-Fi and 3G compatibility, and multiple sensors, including tri-axis gyroscope, accelerometer, proximity sensor, temperature sensor, magnetic field and light sensors. The Wi-Fi compatibility, accelerometer and gyro sensors satisfy our device requirements. The 3G compatibility makes it possible to set up a Google App Engine server to share filtering results of positions or even leave the Kalman filtering computation to the server for battery saving purpose.

What's more, the GPS capability makes it possible to extend the localization algorithm to outdoor environments. The specification of the Samsung Tab used for the experiment is summarized in Appendix 1.

2.1.3 Android Platform

Android is software stack for mobile devices that includes an operating system, middleware, and key applications owned by Google Inc. [17] Android operating system is based on a modified version of Linux kernel. Android's development and release is collaboration between Google and other members of the Open Handset

Alliance, including major smart phones manufacturers, such as Motorola, Samsung, HTC, etc. making the Android platform one of the world's most popular best-selling Smartphone operating platform. This allows us to change our platform easily if new devices are available. Compared to other operating systems, such as Apple iOS, Microsoft Windows Phone 7 or Windows mobile, the open source Android system gives developers more freedom. To develop an algorithm on Android platform, its software development kit (SDK) is used. Android SDK includes a comprehensive set of development tools, which includes a debugger, libraries, a handset emulator, documentation, sample code and tutorials [17]. Currently supported development platforms of smart phones include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.4.9 or later, Windows XP or later. The officially supported integrated development environment (IDE) is Eclipse (currently 3.5 or 3.6) using the Android Tools (ADT) Plug-in. The Android SDK uses the Java programming language.

Top Smartphone Platforms
3 Month Avg. Ending Jan. 2011 vs. 3 Month Avg. Ending Oct. 2010
Total U.S. Smartphone Subscribers Ages 13+
Source: comScore MobiLens

	Share (%) of Smartphone Subscribers		
	Oct-10	Jan-11	Point Change
Total Smartphone Subscribers	*100.0%*	*100.0%*	*N/A*
Google	23.5%	31.2%	7.7
RIM	35.8%	30.4%	-5.4
Apple	24.6%	24.7%	0.1
Microsoft	9.7%	8.0%	-1.7
Palm	3.9%	3.2%	-0.7

Figure 1. Top Smartphone Platforms Statistics

2.1.4 SensorManager API and WifiManager API

The Android system provides a set of APIs for retrieving information from sensors. The three main APIs used in this project is the WifiManager API, SensorListerner and SensorManager API.

SensorManager is a class that permits to access to the sensors available within the Android devices. To activate one sensor and retrieve its value, we need to register the sensor to listen its activities, and if we do not need one sensor anymore, it's better to deactivate it by unregistering it from the SensorManager to save power.

While the SensorManager decides whether a sensor is activated or not, the SensorListerner API is used to "listen" if the values of one or more activated sensors have changed. It includes two required methods as summarized in **Table 1**.

Table 1. Functions Used in SensorListener [18]

Function	Features
onSensorChanged(int sensor, float values[])	This method is invoked whenever a sensor value has changed. The method is invoked only for sensors being monitored by this application (more on this below). The arguments to the method include an integer that identifies the sensor that changed, along with an array of float values representing the sensor data itself. Some sensors provide only a single data value, while others provide three float values. The orientation and accelerometer sensors each provide three data values
onAccuracyChanged(int sensor, int accuracy)	This method is invoked when the accuracy of a sensor has been changed. The arguments are two integers: One represents the sensor, and the other represents the new accuracy value for that sensor.

The WifiManager API is a primary class to manage the Wi-Fi functionality such as Wi-Fi network connection configuration and management, Access-Point detection and Wi-Fi connectivity monitoring. [19]. Two functions which are of prime importance to the current project are summarized in **Table 2**.

Table 2. Functions Used in WifiManger

Function	Features
startScan();	This method is invoked to request a scan for available access points nearby. This function will not return the resultant list of access points
getScanResults();	This method is invoked to get the results of the latest access points scan

2.1.5 Introduction to Google App Engine

Google App Engine (GAE) which is used extensively in the project is a platform for developing and hosting web applications in Google's infrastructure [20]. Current supported programming languages for GAE are Python, Java and other Java Virtual Machine languages. The advantage of GAE lies in Google's high performance infrastructure which permits intensive computation, making the applications more portable and energy efficient. For this project a GAE application has been successfully set up to receive the signals from the Samsung Tab. In future, the application can be further developed to perform filtering computation which is processed previously in the Samsung Tab for power saving purpose.

2.2 Global Coordinate Definition and Geomagnetic Field in the Test Laboratory Environment

2.2.1 Global Coordinate Definition

To facilitate the experiment, a global coordinate system is defined. Z-axis of the global coordinate is defined as the downward direction. X-axis and Y-axis are defined as shown in **Figure 2**.

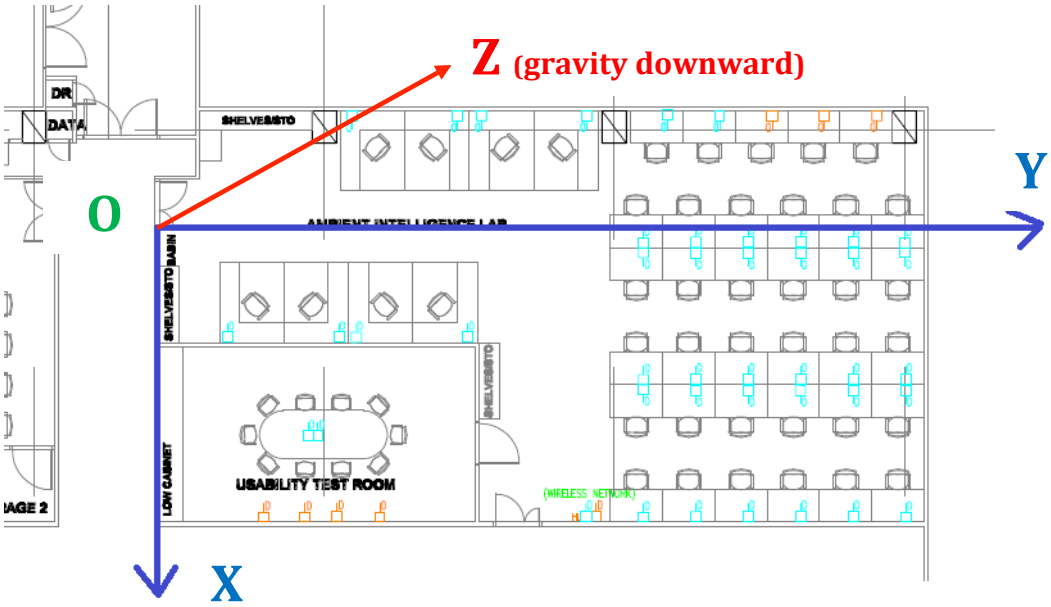

Figure 2. Global Coordinate Definition

2.2.2 Geomagnetic Field in AMI lab

According to IGRF [33], the geomagnetic filed strength in the laboratory, which is the Ambient Intelligence (AMI) Laboratory is shown in **Table 3**.

Table 3. Geomagnetic Field according to IGRF

13'	- 15° 11'	40.6513 μT	-11,036.1 μT

Declination is a deviation of the magnetic field from true north and is defined as positive if it is eastward. [34] Inclination is the angle made by a compass needle with horizontal on the Earth's surface, and it is defined as positive if the magnetic field is pointing down. [35]. Unit for magnetic strength is micro Tesla. Therefore, the magnetic field strength in true north direction is 40.65610 μT and in the east direction, it is 0.1544 μT .

To compute the geomagnetic field strength in the global coordinate defined in section 2.2.1, we need to know the relation between the global coordinate and true north, which is shown in the following figure.

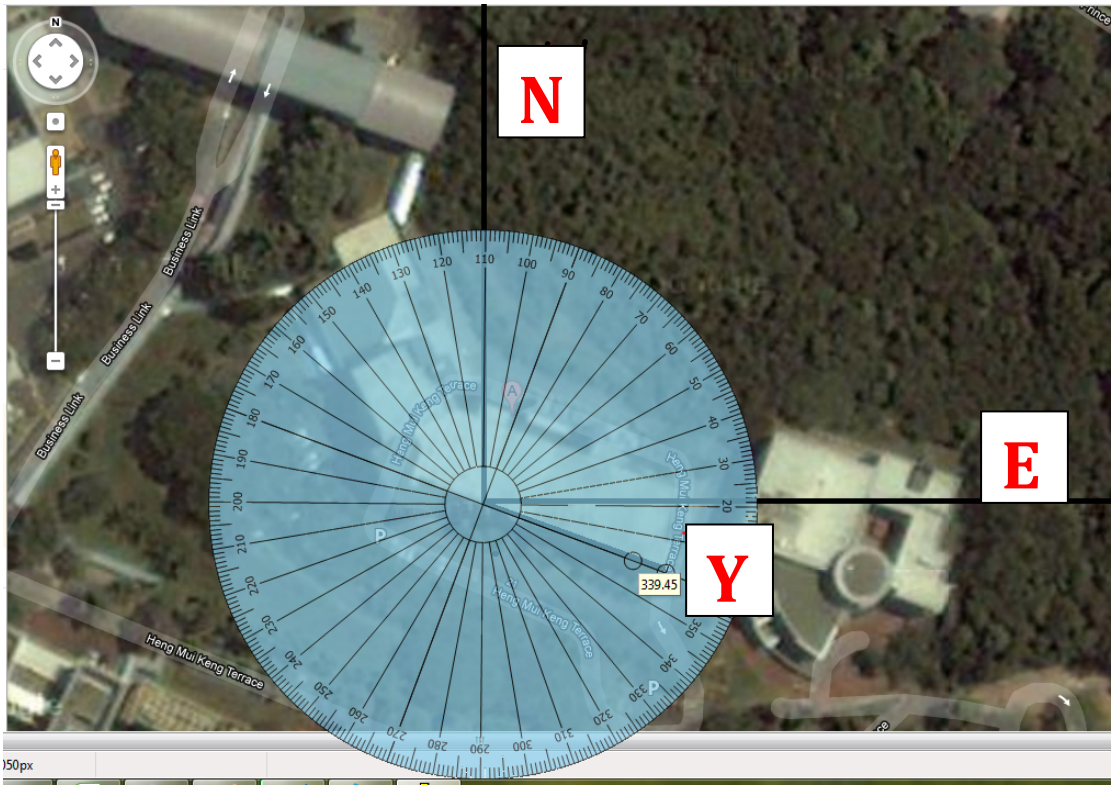

Figure 3. AMI Lab Geographical Location

Figure 3 is a snapshot from Google Map. Y means Y-axis of the global coordinate. N means truth north and E stands for east direction. The angle shown in **Figure 3** is 339.45^{o} and the magnetic field strengths in the global coordinate are calculated as:

$$\begin{bmatrix} m_{x0} \\ m_{y0} \\ m_{zo} \end{bmatrix} = \begin{bmatrix} 38.1994 \\ 13.9035 \\ 11.036 \end{bmatrix} \quad \textbf{(1)}$$

Unit in Equation (1) is μT

As presented in section 2.3.3, due to the buildings' attenuation effect, magnetometer measurements are much smaller than the theoretical values. Through magnetic field analysis in the AMI lab, it is found that attenuation effect is different in different locations. Therefore, the reference vector is normalized as:

$$\begin{bmatrix} g_{x0} \\ g_{y0} \\ g_{zo} \end{bmatrix} = \begin{bmatrix} 0.9069 \\ 0.3301 \\ 0.2620 \end{bmatrix} \quad \textbf{(2)}$$

2.3 Sensors Signal Analysis

This section includes the signal analysis of the device. As the main purpose of the Samsung Tab's embedded sensors is for entertainment, relatively low-cost sensors have been used and this will to some extent affect the experimental results.

2.3.1 Accelerometer

The Samsung Tab's accelerometer model is BMA150, a digital tri-axial sensor from BOSCH whose acceleration refresh rate is 2700Hz according to the datasheet of

BMA150. However, in the Android system, the maximum update rate is limited to 50 Hz which is set mainly for power saving purposes. Thus, the Android system wakes up the BMA, gets the value and then puts it back to standby. Note that the wake up time for BMA150 1.5ms. The zero-g offset for BMA150 from the datasheet is $\pm$ 0.5886 m/s^2. The offset value is due to stress effects during the soldering process.

As the Android system does not have an API to retrieve the sensor signals at a fixed rate, the sensors refresh rate is set as fastest delay, and the sensors' value is updated to a variable. A 20 Hz timer is set to fetch the values contained in the variable every 50ms. The result is not that satisfactory as described by the datasheet. The device is put on a flat, static table; the signals are recorded over 100 seconds. The plot of the signals is as follows:

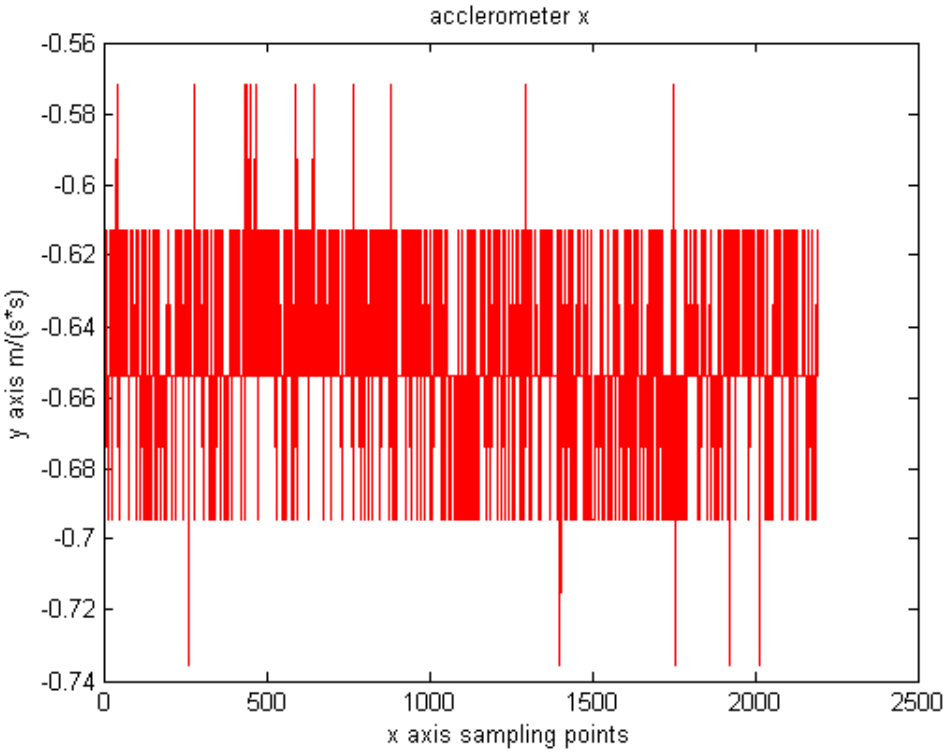

Figure 4. Static Test of Accelerometer's X-axis

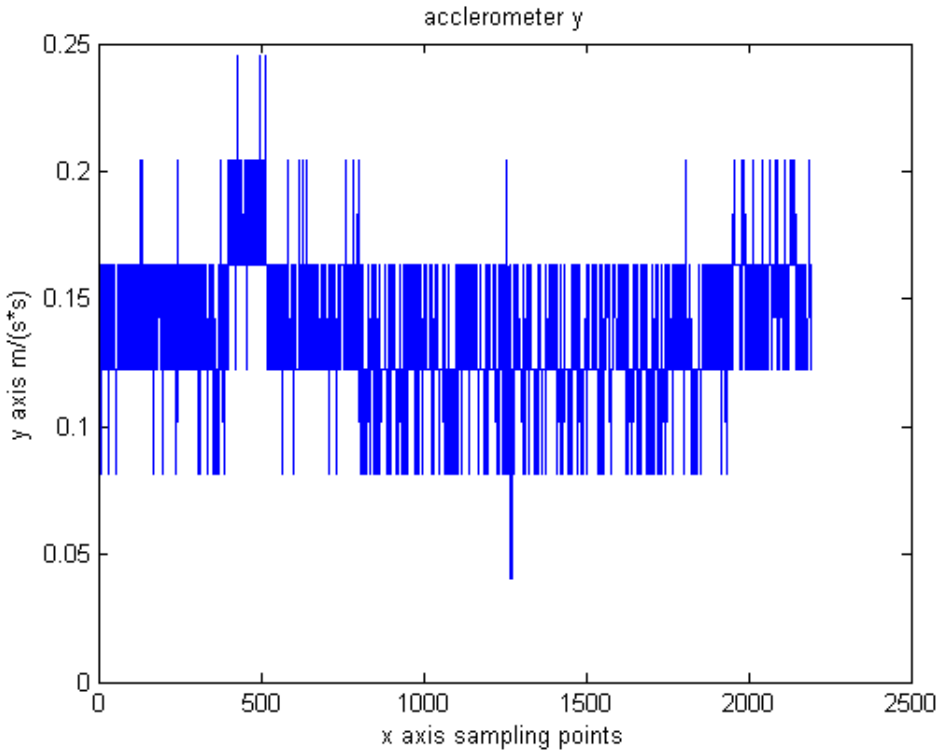

Figure 5. Static Test of Accelerometer's Y-axis

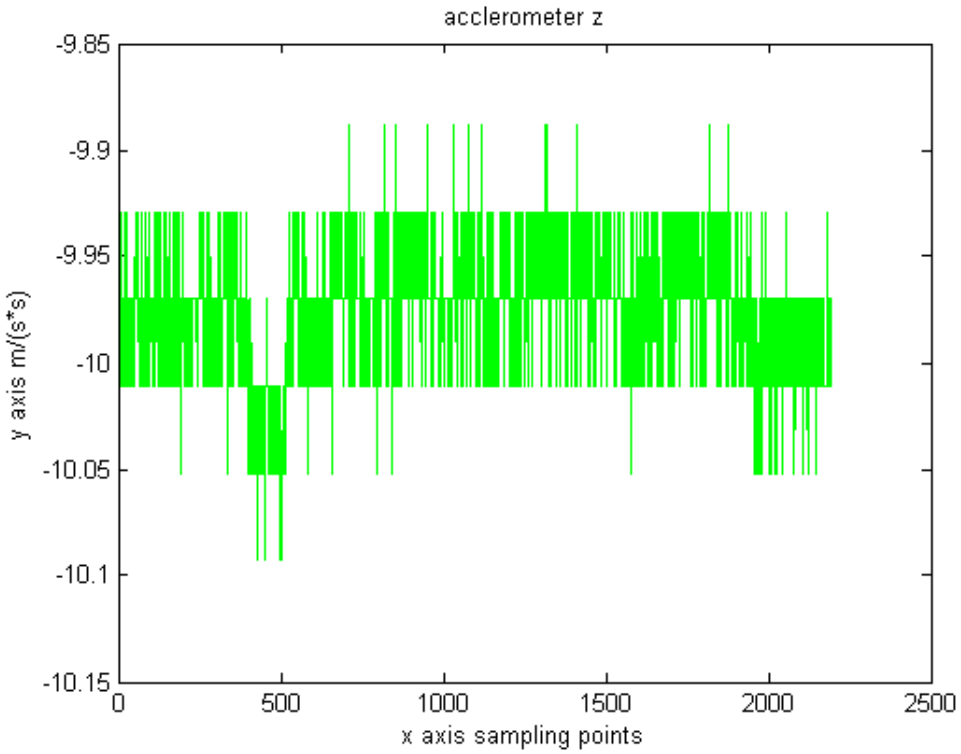

Figure 6. Static Test of Accelerometer's Z-axis

The resolution of accelerometer is $0.05m^2/s$. The signal variations are due to measurement noise and they are modeled as white Gaussian noise. Other than the noise, offsets of each axis also cause measurement errors. In DRA-I and DRA-II, the accelerometer's measurement errors are classified as white Gaussian noise and offsets. In DRA-III, the sensitivity difference of the accelerometer axis is taken in account and a more sophisticated algorithm for calibration is implemented. The zero-g offsets and variances for the three axes are summarized in **Table 4**.

Table 4. Zero-g Offsets and Standard Deviations for Accelerometer

Term	Axis	Accelerometer (m/s^2)
Mean	X	-0.25515
	Y	0.119715
	Z	0.756851
Standard Deviation	X	0.026861
	Y	0.024305
	Z	0.033794

Although the standard deviation is acceptable, the drift in the Z-axis is too large and a severe fluctuation is triggered for different orientation of the device, which results in inaccuracy in the estimation. **Table 5** shows the offsets and standard deviation when the device is reoriented.

Table 5. Offsets and Standard Deviation of Reoriented Accelerometer

Term	Axis	Accelerometer (m/s^2)
Mean	X	-0.6496
	Y	0.1374
	Z	-0.1653
Standard Deviation	X	0.024635
	Y	0.027415
	Z	0.031178

Different offset errors might be due to the soldering and assembly operations during the manufacturing process. The soldering of BMA150 to the motherboard of Samsung Tab creates extra pressure on the digital sensor, and therefore a larger zero-g offset value. During the assembly process, if the motherboard is not fixed well, say 5 degrees drift from the horizontal level; it gives a 0.85 m^2/s deviation in the X-axis. After the assembly, the engineers use Android's calibration utility to set the offset in all axes to 0. But in the reverse direction, the offset value is increased, as indicated in **Table 5**.

2.3.2 Gyroscope

The gyroscope's signals are shown in the following three figures.

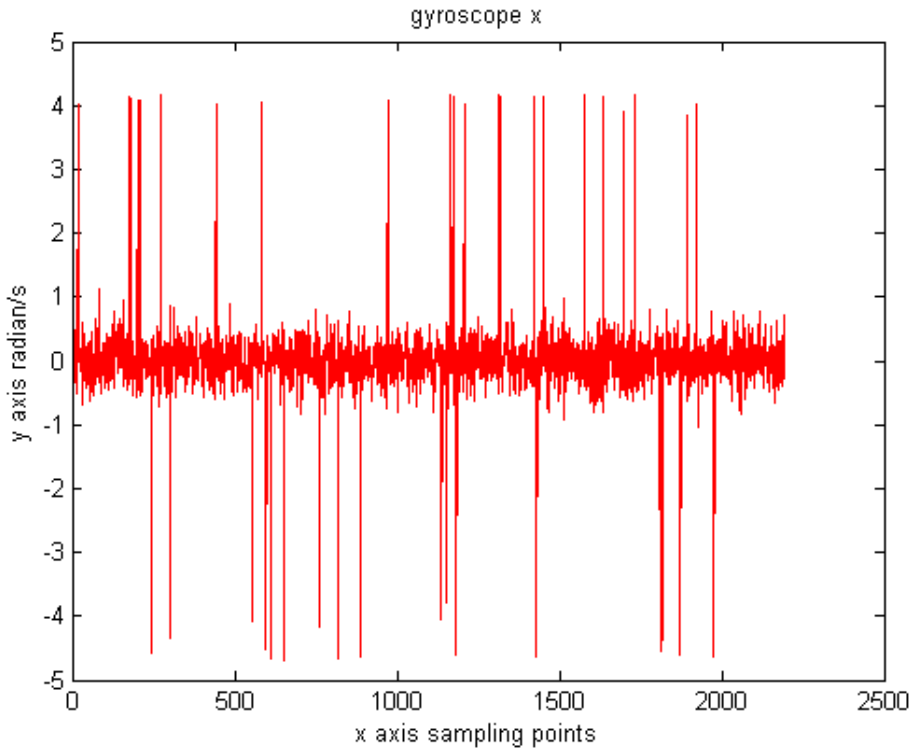

Figure 7. Static Test of Gyroscope's X-axis

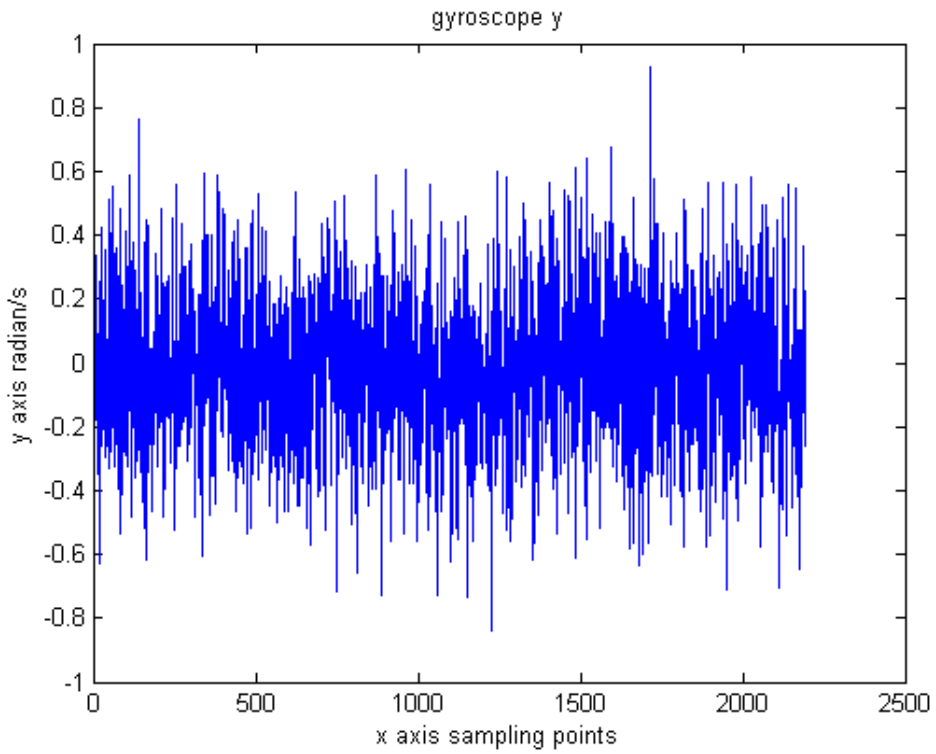

Figure 8. Static Test of Gyroscope's Y-axis

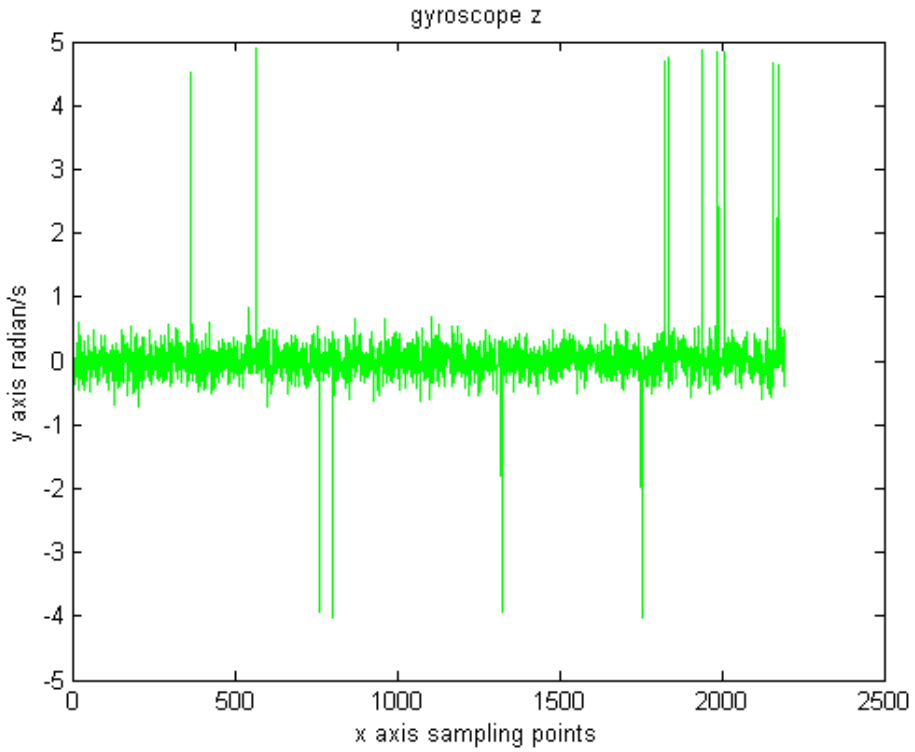

Figure 9: Static Test gyroscope's z axis

The gyroscope resolution is much better than that of the accelerometer and its zero-offset and standard deviation are also satisfactory. The mean and standard deviation of each axis are shown in **Table 6**.

Table 6. Offsets and Standard Deviations for Gyroscope

Term	Axis	Gyroscope (deg/s)
Mean	X	-0.2550
	Y	0.7025
	Z	0.7590
Standard Deviation	X	0.4189
	Y	0.2848
	Z	0.2712

It is observed that, even while the device is static, there are spikes coming out. To investigate the reason for the spikes, various kinds of tests have been conducted. It has

been verified that the spikes have no relationship with environmental changes, such as switching on/off of the laptop, light vibration of the test table, electromagnetic interference from other devices or temperature effect. The potential explanation lays either in the quality of the digital sensor itself, L3G4000 from STMicroelectronics, or the soldering defects of the sensor to the motherboard.

2.3.3 Magnetometer

The magnetometer's signals are shown in the following figures.

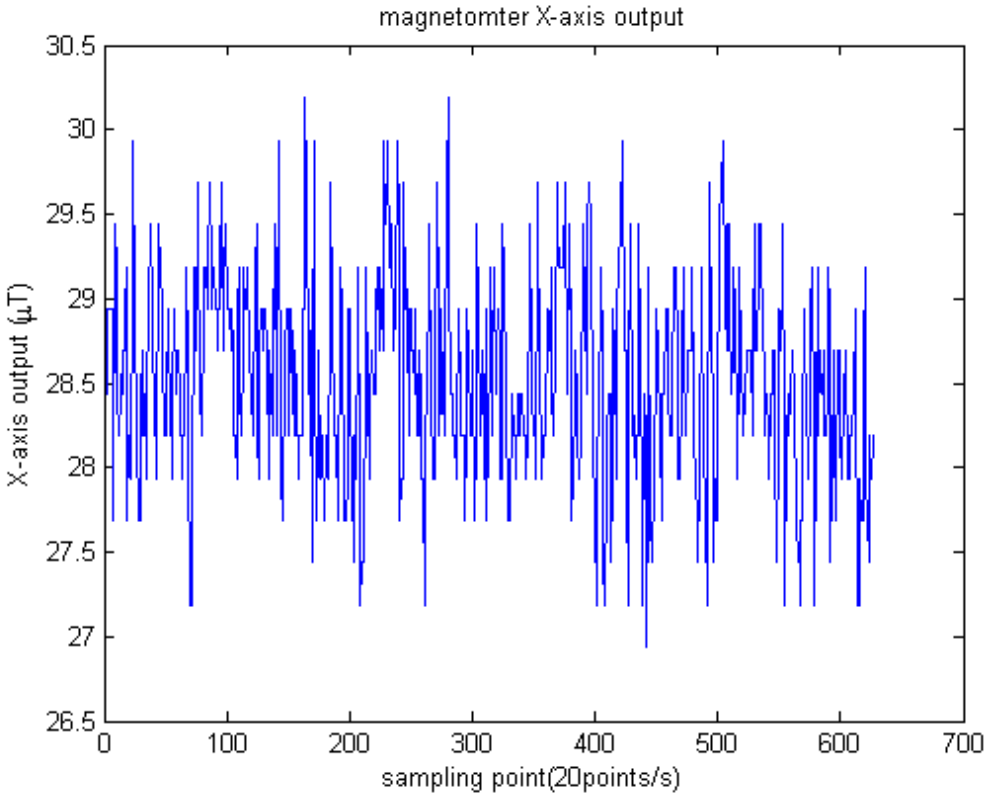

Figure 9. Magnetometer X-axis Signals

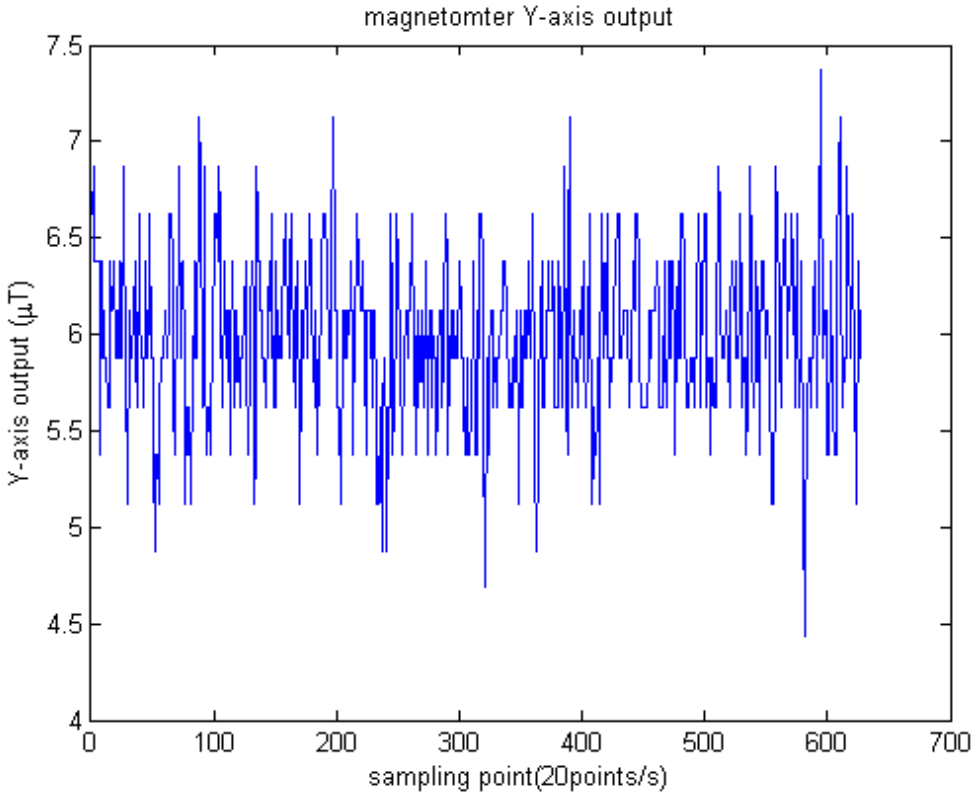

Figure 10. Magnetometer Y-axis Signals

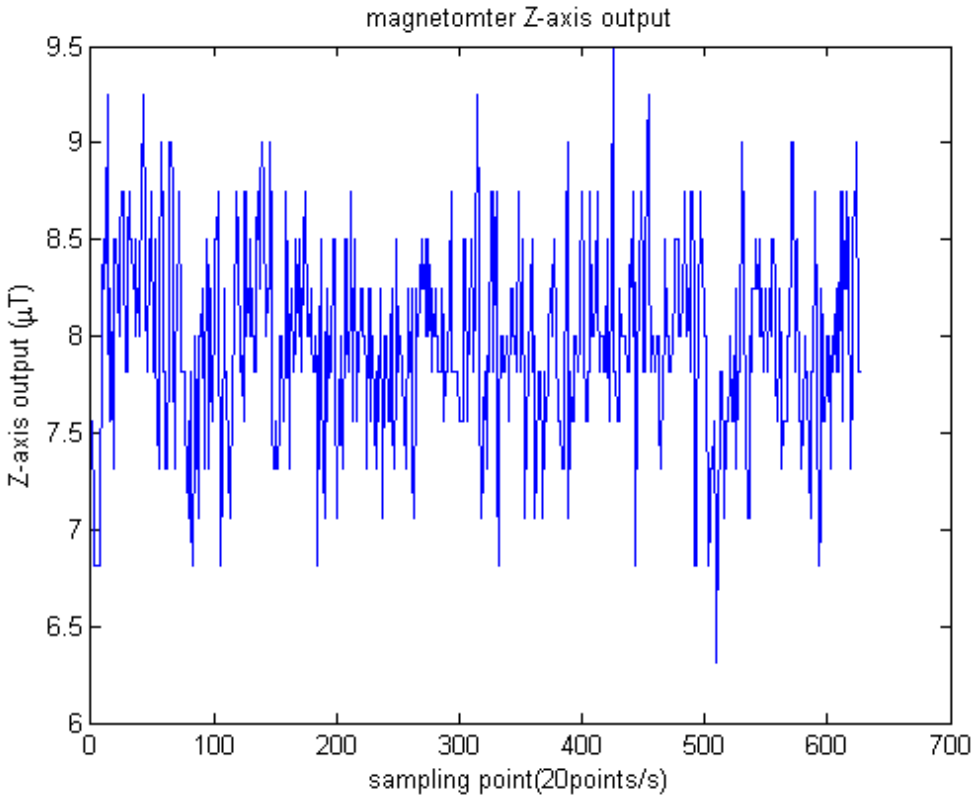

Figure 11. Magnetometer X-axis Signals

Unlike the accelerometer and gyroscope, other than offsets and measurements noise, the magnetometer measurement errors are subjected to building attenuation effect, wide band noise, and external interferences, which can be classified as soft and hard irons [32]. In DRA-I and DRA-II, the error factors are simplified as offsets, white Gaussian noise and attenuation factor. **Table 7** shows the offsets of the accelerometer's axes and standard deviation of the measurement noises. The magnetometer measurement average amplitude is 30.2313 μT. In DRA-III, a more complicated calibration algorithm is adapted and will be presented later.

Table 7. Offsets and Standard Deviations for Magnetometer

Terms	Axis	Magnetometer(μ Tesla)
Mean	X	-4.6451
	Y	2.1553
	Z	3.0132
Standard Deviation	X	0.5241
	Y	0.4510
	Z	0.5272

CHAPTER 3 PROPOSED DEAD RECKONING ALGORITHM - I

Three dead reckoning algorithms (DRAs) have been proposed in this project and they are presented separately in Chapters 3, 4 and 5. Chapter 3 introduces the first proposed algorithm, DRA-I and it is divided into five sections. Section 3.1 presents the step counting algorithm used in all the DRAs, which provides accurate stride length estimations and is designed to work in real time. Section 3.2 gives a brief introduction to the Kalman filter, which is commonly used as a fusion algorithm. In this project, due to the non-linear relationships either in the system dynamic model or between the state vector and measurement vector, a variation of the Kalman filter named as extended Kalman filter is applied instead in all the three DRAs. Section 3.3 introduces DRA-I and Wi-Fi fingerprinting in details. Experiment results of the algorithm are summarized in section 3.4 and section 3.5 provides corresponding discussion.

3.1 Step Counting Algorithm

Step counting has been widely used in indoor localization, due to its accurate estimate of stride lengths and simplicity in implementation. Steps are usually counted by peak and valley detection of the accelerometer's outputs and the selection of accelerometer's signals could be different depending on the techniques being used. [13][30][31]. [13][30] compute acceleration amplitudes, while [31] only uses acceleration signals in vertical axis for step counting.

For stride length estimation, [13] assumes a fixed step-length which can be determined by dividing a fixed walking distance to the number of steps taken. [30] estimates the stride length l by taking the following formula.

$$l = \sqrt[4]{(A_{\max} - A_{\min})} * K \quad \textbf{(3)}$$

$A_{\max}$, $A_{\min}$ refer to the maximum and minimum values of the vertical acceleration in a stride and K is a constant.

In this project, the acceleration amplitude is used for step counting purpose. A modified Formula (3) was used for strides length estimation.

3.1.1 Principles of Step Counting Algorithm

The proposed step counting algorithm is designed to work in real time. However, before going into how to detect one step in real time, we will first set up rules on how to define one step from the accelerometer signals. **Figure 12** illustrates an accelerometer signal with 5 separate steps.

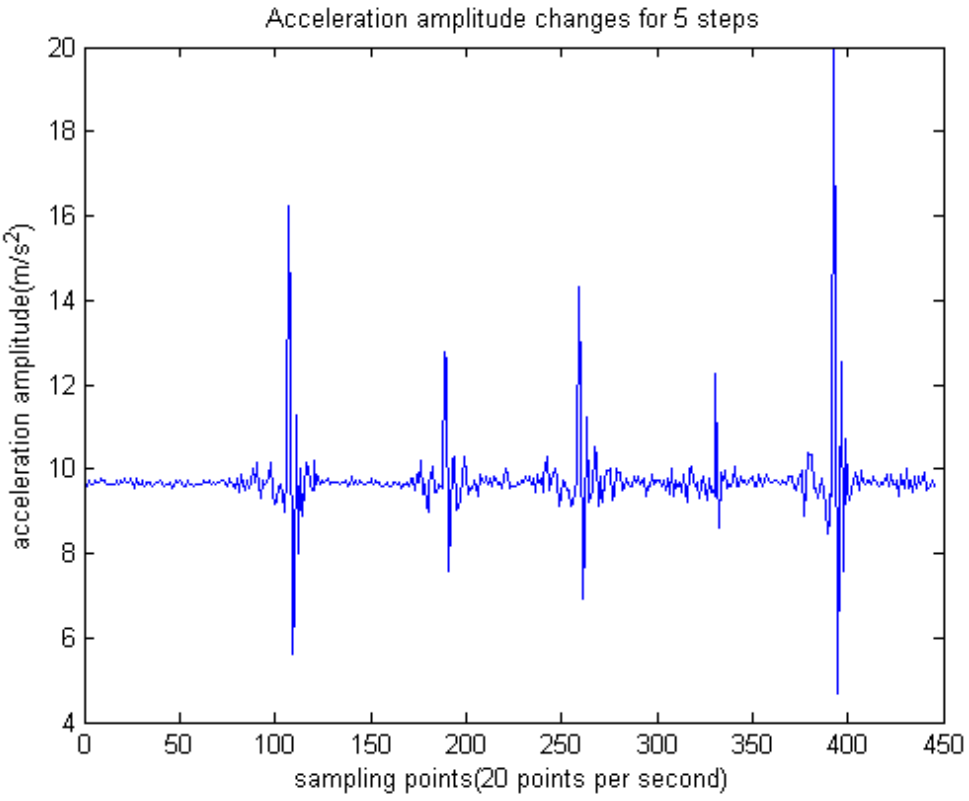

Figure 12: Accelerometer Output for 5 Steps with Pause in-between

A measurement with pauses between every walking step is plotted in **Figure 12** for better illustration of the relationship between number of steps and the corresponding accelerometer signals. From **Figure 12**, the five pulses of the acceleration's amplitude correspond to five steps walked. Therefore, if one such pulse is detected, we can conclude that one step has been taken. This relationship is defined in the First rule, which is described as follows:

Rule 1: Each step includes a peak (local maximum) and the next valley (local minimum) of the acceleration amplitude. In other words, if we detect one peak and one valley, one step shall be counted.

From the signals shown in **Figure 12**, many small peaks and valleys can be observed, which are due to noise or possible shakings of the device holder. These variations are named as non-conformant pairs of peak and valley. A pair of peak and valley which can be used to count one step is called an effective pair of peak and valley. Non conformant pairs of peak and valley shall be filtered out and the second criterion is set for this purpose:

Rule 2: The difference between an effective pair of peak valley shall exceed a predefined threshold value, $\Delta_{h.}$ The threshold value can be determined by offline training. For example, a pedestrian is asked to walk for a fixed number of steps several times to determine the threshold value that can correctly filter out noises. In this project, ten tests of 50 steps have been conducted to determine the threshold value, which is found to be 3.5 m/s^2.

An additional criterion for defining an effective pair of peak and valley is spelt out in Rule 3.

Rule 3: For an effective pair of peak and valley, the time difference between the peak and the valley shall exceed 100ms.

100ms time difference implies that the pedestrian will not have a step rate greater than 5 steps per second. This is a reasonable assumption as even the Olympic champion in 100 meters contest, Usain Bolt can only achieve a step rate 4.23 steps per second. [36]

To further improve the step counting efficiency in real time, half-steps are counted rather than full steps. A half step is detected if Rule 1a is triggered, which is a variation of Rule 1.

Rule 1a: A half-step includes a peak and a valley. A half-step is counted if a valley is detected next to a peak or if a peak is detected next to a valley.

3.1.2 Step Counting Algorithm in Real Time

If the acceleration amplitudes of a test are known, it is easy to implement the previous three rules and count the step accurately. In real time as future acceleration amplitudes are not known, the implementation is not intuitive but rather complicated. The following example shown in **Figure 13** is used for better illustration.

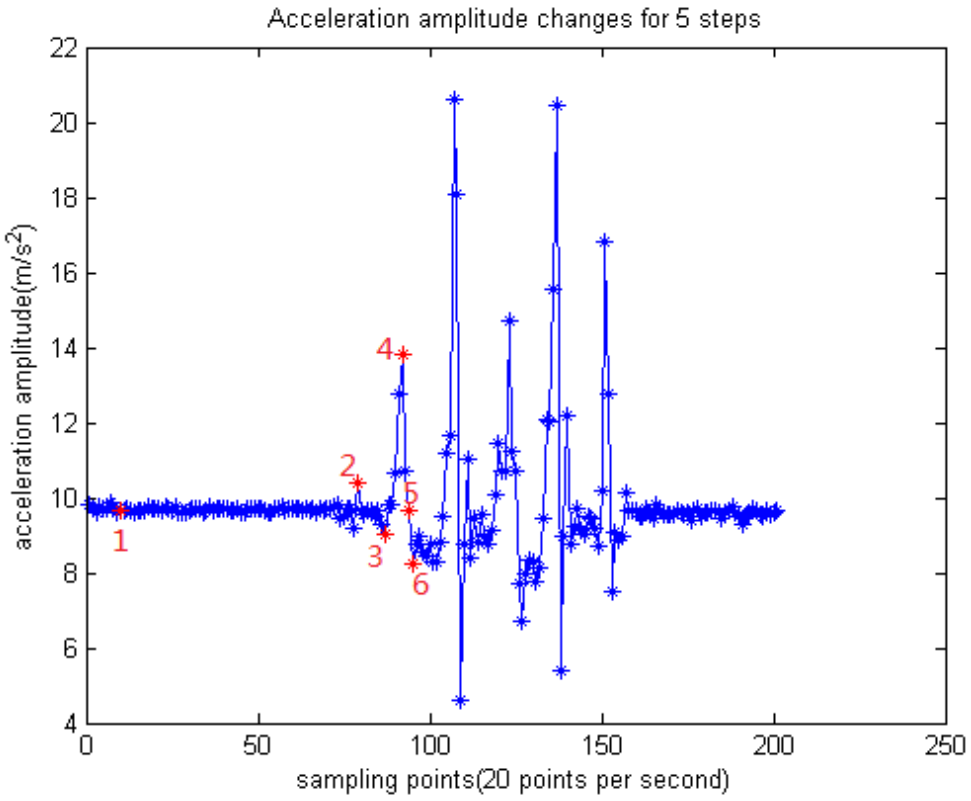

Figure 13: Accelerometer Output for 5 Steps without Pause in-between

First, we need to define two parameters: A_{ref} and A_{dir}. A_{ref} is the last extremum's (peak/valley) value and A_{dir} indicates whether it is a peak or a valley. If A_{dir} is 1, A_{ref} is the value of a valley and if A_{dir} is 0, then A_{ref} is the value of a peak. A_{dir} is essential to determine if a half-step shall be counted or not. If A_{dir} is 0, the previous extremum is a peak and we need the next extremum to be a valley to count a half-step and vice versa. The next challenge is to determine whether a new measurement is a peak or valley without knowing the future measurements, so that a half-step is counted. In fact, it is not necessary and a half-step can be counted even before next peak or valley is measured, but as long as they are expected to be near ahead. The next paragraph shows how it works. Besides, the following paragraphs also show how to make sure A_{ref} is indeed the last extremum's value when a half-step is counted.

Accelerations are measured 20 times per second, as shown by blue stars in **Figure 13**. To simplify the interpretation, six typical points are chosen to elaborate the algorithm, noted as points 1, 2, 3, 4, 5, and 6 shown in **Figure 13**. Their values are noted as A_1, A_2, A_3, A_4, A_5 and A_6, respectively. First, A_{ref} and A_{dir} are initialized. As shown in **Figure 12**, if the pedestrian starts from a stationary position, the first extremum of acceleration is always a peak. However, Rule 2 requires the difference between a peak and next valley but not between the baseline and a peak. Therefore, from the starting point, if a valley is found, two half-steps are counted and A_{dir} is set as 0 and A_{ref} is set as the acceleration amplitude of the first measurement, point 1, and $A_{ref}=A_1=9.6479$ m/s^2. Our algorithm guarantees A_{ref} is gradually updated as A_4, the first peak's value, before the first two half-steps are counted.

Point 1: As discussed in previous paragraph. $A_{dir} = 1 \quad A_{ref} = A_1 = 9.6479 m/s^2$

Point 2: $A_2 = 10.3917 m/s^2$, $A_{ref} = 0$ and $A_2 > A_{ref}$, hence $A_{ref} = 10.3917$.

Point 3: $A_{dir} = 0$ and $A_3 = 9.0271 m/s^2 < A_{ref}$ but the difference is smaller than $3.5 m/s^2$, no action is taken.

Point 4: $A_{dir} = 0$ and $A_4 = 13.8706 m/s^2 > A_{ref}$ hence $A_{ref} = A_4 = 13.8706$. Note that A_{ref} is now the value of the first peak.

Point 5: $A_{dir} = 0$, $A_5 = 9.6497 m/s^2 < A_{ref}$ and the difference is larger than $3.5 m/s^2$.

Therefore, the next valley is either A_5 or near ahead, as in this case of A_6 being the valley. Therefore, Rule 2 is triggered and a half-step from the first peak to the first valley is counted. As the half-step from baseline to the first peak has not been counted, two half-steps are registered here. The following actions are taken. $A_{ref} = A_5 = 9.6497 m/s^2$, $A_{dir} = 1$

Point 6: $A_{dir} = 1$, $A_6 = 8.2455 m/s^2 < A_{ref}$ hence $A_{ref} = A_6 = 8.2455 m/s^2$. Note that A_{ref} is again an extremum value.

The process repeats until the last measurement from the accelerometer. The stride length is estimated by **Equation (3)** with its parameters $A_{\max}$, $A_{\min}$ defined as peak and valley of acceleration amplitudes respectively. In this example, $A_{\max}$ is the value of A_4 and $A_{\min}$ is the value of A_6. The final half-step is from the last valley to the baseline. As their difference values are generally less than the threshold value, an additional half-

step is counted if no new measurement comes from accelerometer. Matlab codes for step counting are appended in **Appendix 2**.

3.1.3 Experiment Results and Discussion

Figure 14 shows a 50-step counting result. 49.5 steps are detected by the proposed algorithm. **Figure 15** shows a result for a 50-step running and 49 steps are triggered by the proposed algorithm. **Figure 16** shows a result of walking down along staircases, the steps counted are totally unrealistic, and hence it can be concluded that the algorithm is only suitable for planar movement.

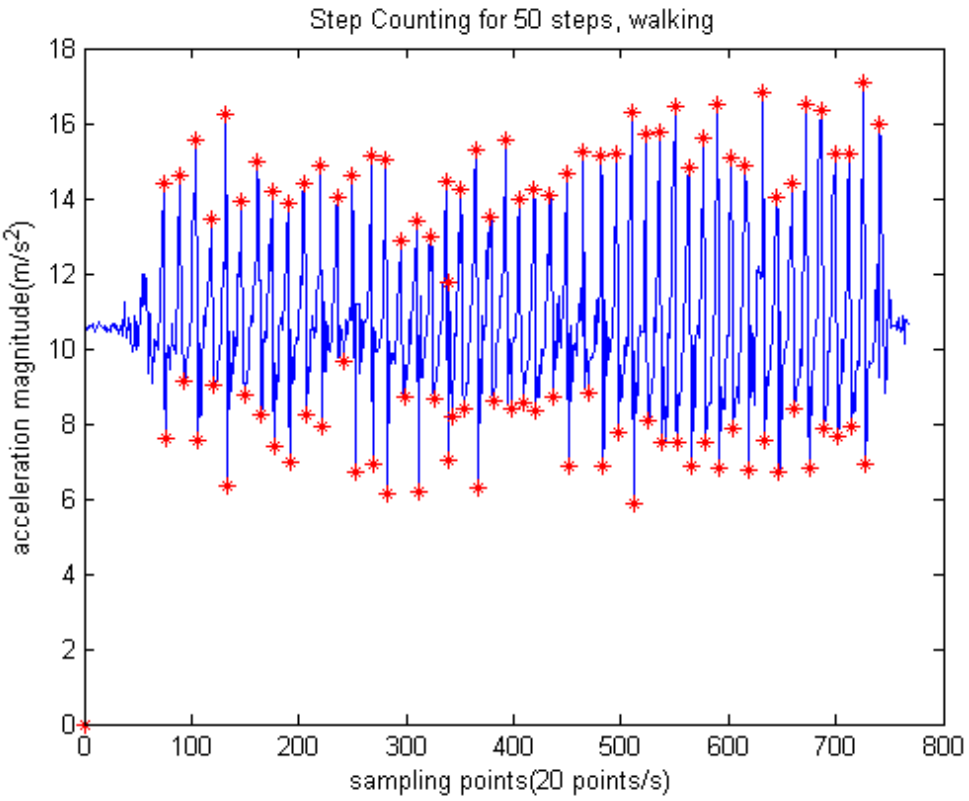

Figure 14: Step Result for 50 Steps' Walking

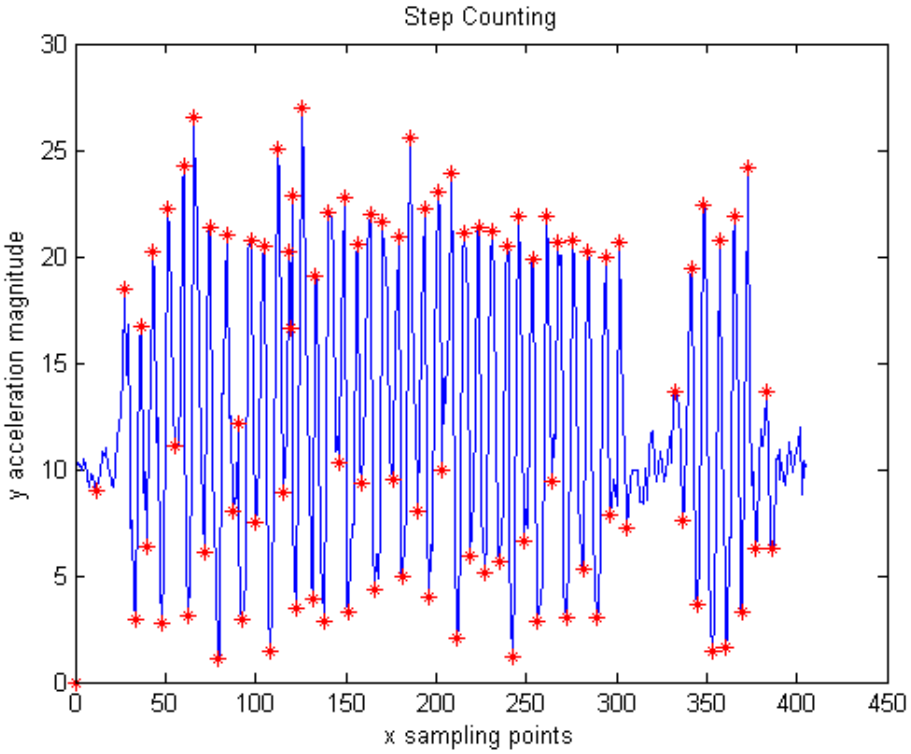

Figure 15: Step Counting Result for 50 Steps' Runing

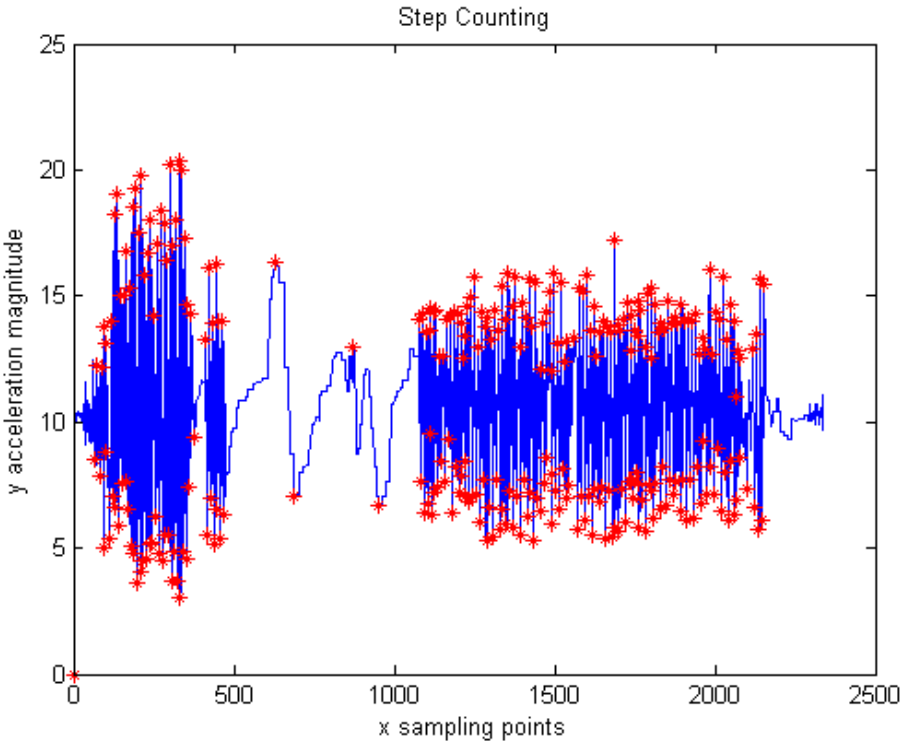

Figure 16: Step Counting Result for Descending From Staircases

3.2 Overview of the Kalman Filter

The Kalman filter is a mathematical method named after Rudolf E.Kalman. Its purpose is to use measurements observed over time, containing noise and other inaccuracies, and produce values that tend to be closer to the true values of the measurements [21]. The Kalman filter uses a system's dynamic model and measurements to form an estimation of the system's varying quantities (its state), which is better than the estimation obtained by using any one measurement alone.

The Kalman filter produces estimates of the true values of measurements and their associated calculated values by predicting a value, estimating the uncertainty of the predicted value, and computing a weighted average of the predicted value and the measured value. The most weight is given to the value with the least uncertainty. The estimates produced by the method tend to be closer to the true values than the original measurements because the weighted average has a better estimated uncertainty than either of the values that went into the weighted average [21].

The Kalman filter is a recursive filter with two phases, predict phase and update phase and its detailed formulations are summarized as follows.

Predict

Predicted (a priori) state estimate

$$\hat{x}_{k|k-1} = A_k \hat{x}_{k-1|k-1} + B_k u_k \quad \textbf{(4)}$$

Predicted (a priori) estimate covariance

$$\mathrm{P}_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_k \quad \textbf{(5)}$$

Update

Innovation or measurement residual	$\tilde{y}_k = z_k - C_k \hat{x}_{k\|k-1}$	**(6)**
Innovation (or residual) covariance	$S_k = C_k P_{k\|k-1} C_k^T + R_k$	**(7)**
Optimal Kalman gain	$K_k = P_{k\|k-1} C_k^T S_k^{-1}$	**(8)**
Updated (a posteriori) state estimate	$\hat{x}_{k\|k} = \hat{x}_{k\|k-1} + K_k \tilde{y}_k$	**(9)**
Updated (a posteriori) estimate covariance	$P_{k\|k} = (I - K_k C_k) P_{k\|k-1}$	**(10)**

Where

- $\hat{x}_{k|k}$, the a posteriori state estimate at time k given observations up to and including at time k;
- $P_{k|k}$, the a posteriori error covariance matrix (a measure of the estimated accuracy of the state estimate).

- A_k is the state transition model which is applied to the previous state x_{k-1};
- B_k is the control-input model which is applied to the control vector u_k;
- w_k is the process noise which is assumed to be drawn from a zero mean multivariate normal distribution with covariance Q_k

$$w_k \approx N(0, Q_k) \quad \textbf{(11)}$$

At time k an observation (or measurement) z_k of the true state x_k is made according to

$$z_k = H_k x_k + v_k \quad \textbf{(12)}$$

H_k is the observation model which maps the true state space into the observed space and v_k is the observation noise which is assumed to be zero mean Gaussian white noise with covariance R_k.

$$v_k \approx N(0, R_k) \quad \textbf{(13)}$$

The basic Kalman filter is limited to a linear assumption. For non-linear systems, extended Kalman filter (EKF) shall be used.

The system model of EFK is:

$$\dot{X} = f(X(t), t) + w(t) \quad (14)$$

The function f can be used to compute the predicted state from the previous estimate and similarly the function h can be used to compute the predicted measurement from the predicted state. However, f and h cannot be applied to the covariance directly. Instead a matrix of partial derivatives (the Jacobian matrix) is computed. At each time step the Jacobian matrix is evaluated with current predicted states. These matrices can be used in the Kalman filter equations. This process essentially linearizes the non-linear function around the current estimate.

3.3 Proposed DRA-I

DRA-I algorithm includes two phases. Phase 1 is based on integration of the accelerometer and gyroscope signals, and Phase 2 is based on step counting. Phase 1 and phase 2 operate in different frequencies. Phase 1 frequency is 20 Hz, phase 2

frequency is 1 Hz. Phase 2 sends position and velocity vector to Phase 1 and Phase 1 sends the object's normalized displacement vector to Phase 2. The step counting algorithm is added to count the steps in each second and subsequently gives the stride length within each second from the corresponding steps. The multiplication of stride length estimation by the normalized displacement vector will return a measured velocity vector for phase 2. The relationship between Phase 1 and Phase 2 is summarized in **Figure17**. In both phases, the Kalman filter will be applied for better estimation.

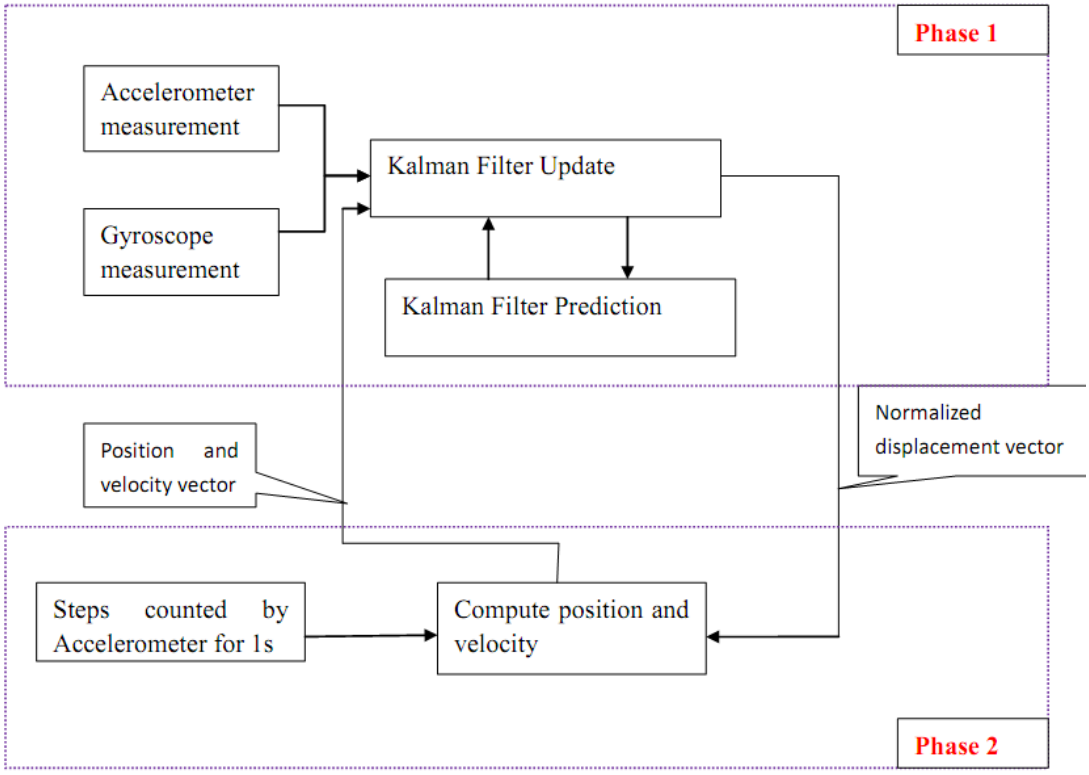

Figure 17: Relationship between Phase 1 and Phase 2 of DRA-I

3.3.1 Phase 1 Algorithm

Phase 1 has the feature of a strap down navigation system [22], which processes the position and velocities provided by phase 2 together with the angular velocity and

acceleration vector, provided by the inertial sensors, to compute the position, velocity, and attitude through a Kalman filter algorithm. Due to the non-linearity of the system dynamic model, the extended Kalman filter (EKF) is employed for the filtering. As the signals from the INS sensors are relative to the device, they were specified by the body coordinate. On the contrary, the position and velocity of the device is in global coordinate. The system vector of the system dynamic model is

$$X = \begin{pmatrix} x & y & z & u & v & w & a_x & a_y & a_z & q_0 & q_1 & q_2 & q_3 & w_x & w_y & w_z \end{pmatrix}^T \quad \textbf{(15)}$$

Vector $\begin{bmatrix} x & y & z \end{bmatrix}^T$ is the Samsung tab's position vector in global coordinate.

Vector $\begin{bmatrix} u & v & w \end{bmatrix}^T$ is the velocity vector in global coordinate.

Vector $\begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$ is the acceleration vector in body coordinate.

Vector $\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$ is the quaternion vector in global coordinate.

Vector $\begin{bmatrix} w_x & w_y & w_z \end{bmatrix}^T$ is the angular velocity vector in body coordinate.

Quaternion is used to describe the orientation of the device, to avoid the Gimbal lock in the dead reckoning process if using Euler angles.

The differentiation of velocity vector and the acceleration vector is as follows:

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = R_{rot} \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \tag{16}$$

R_{rot} is the rotation matrix from the body coordinate to the global coordinate. The form of the rotation matrix is:

$$\begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_2) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \tag{17}$$

R_{rot} is an orthonormal matrix and the inversion of R_{rot} is its tranpose.

While the device is static, the value of the accelerometer is 9.8 m/s^2 in $Z-axis$ of the global coordinate and this will cause tremendous error in future numerical integration, and therefore, it is necessary to add a "$-g$" to this direction. Hence, the relationship between the measured acceleration matrix and real acceleration matrix is:

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = R_{rot}^T \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} + \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix} \tag{18}$$

The quaternion matrix dynamic evolution [24] is as follows:

$$\begin{cases} \dot{q}_0 = -\frac{1}{2} q_1 w_x - \frac{1}{2} q_2 w_y - \frac{1}{2} q_3 w_z \\ \dot{q}_1 = +\frac{1}{2} q_0 w_x - \frac{1}{2} q_3 w_y + \frac{1}{2} q_2 w_z \\ \dot{q}_2 = +\frac{1}{2} q_3 w_x + \frac{1}{2} q_0 w_y - \frac{1}{2} q_1 w_z \\ \dot{q}_3 = -\frac{1}{2} q_2 w_x + \frac{1}{2} q_1 w_y + \frac{1}{2} q_0 w_z \end{cases} \tag{19}$$

Therefore, the whole dynamic system of the device can be described by:

$$
\begin{cases}
\dot{x} = u \\
\dot{y} = v \\
\dot{z} = w \\
\dot{u} = \left(q_0^2 + q_1^2 - q_2^2 - q_3^2\right)a_x + 2\left(q_1 q_2 - q_0 q_3\right)a_y + 2\left(q_1 q_3 + q_0 q_2\right)a_z \\
\dot{v} = 2\left(q_1 q_2 + q_0 q_3\right)a_x + \left(q_0^2 - q_1^2 + q_2^2 - q_3^2\right)a_y + 2\left(q_2 q_3 - q_0 q_1\right)a_z \\
\dot{w} = 2\left(q_1 q_3 - q_0 q_2\right)a_x + 2\left(q_2 q_3 + q_0 q_1\right)a_y + \left(q_0^2 - q_1^2 - q_2^2 + q_3^2\right)a_z \\
\dot{a}_x = -\dfrac{1}{\tau}a_x \\
\dot{a}_y = -\dfrac{1}{\tau}a_y \\
\dot{a}_z = -\dfrac{1}{\tau}a_z \\
\dot{q}_0 = -\dfrac{1}{2}q_1\omega_x - \dfrac{1}{2}q_2\omega_y - \dfrac{1}{2}q_3\omega_z \\
\dot{q}_1 = +\dfrac{1}{2}q_0\omega_x - \dfrac{1}{2}q_3\omega_y + \dfrac{1}{2}q_2\omega_z \\
\dot{q}_2 = +\dfrac{1}{2}q_3\omega_x + \dfrac{1}{2}q_0\omega_y - \dfrac{1}{2}q_1\omega_z \\
\dot{q}_3 = -\dfrac{1}{2}q_2\omega_x + \dfrac{1}{2}q_1\omega_y + \dfrac{1}{2}q_0\omega_z \\
\dot{\omega}_x = -\dfrac{1}{\tau}\omega_x \\
\dot{\omega}_y = -\dfrac{1}{\tau}\omega_y \\
\dot{\omega}_z = -\dfrac{1}{\tau}\omega_z
\end{cases}
\qquad \textbf{(20)}
$$

We can rewrite the model as:

$$\dot{X} = f(X(t), t) + w(t) \qquad \textbf{(21)}$$

X is the state vector. f is the state transition function and $w(t)$ is the model noise.

The measurement vector is

$Z = \begin{bmatrix} x & y & z & u & v & w & a_x & a_y & a_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ *and*

$\begin{bmatrix} x & y & z & u & v & w \end{bmatrix}^T$ is provided by phase 2.

$$Z = CX(t) + v(t) \tag{22}$$

Z is the measurement vector. C is the observation matrix and v is the observation noise matrix. The relation between body coordinate vectors and global coordinate vectors are non-linear. Therefore, an extended Kalman filter is applied instead of the basic Kalman filter. The linearized model is described by:

$$\dot{X} = AX + W \tag{23}$$

Where

$$A = \frac{\partial f}{\partial X} \tag{24}$$

The measurement vector in our case is a discrete sampled signal from the devices. Therefore, we need to transform our model to a discrete time model. For any continuous time dynamic model [23] as follows

$$\begin{cases} \dot{X} = AX(t) + w(t) \\ \dot{Z} = CX(t) + v(t) \end{cases} \tag{25}$$

Its discrete time model is:

$$\begin{cases} X_{k+1} = A_d X_k + W_d(t) \\ Z_k = h_k X_k + v_k \end{cases} \tag{26}$$

where

$$A_d = e^{Adt} \tag{27}$$

and

$$W_d(k) = \int_0^{dt} e^{Av} w((k+1)dt - v)dx \quad \textbf{(28)}$$

By Taylor expansion,

$$A_d = I + Adt \quad \textbf{(29)}$$

$$W_d = Wdt + (AW + (AW)^T)\frac{dt^2}{2} + AWA^T\frac{dt^3}{3} \quad \textbf{(30)}$$

The third factor of W_d is to ensure the noise matrix is positive definite.

3.3.2 Phase 2 Algorithm

Phase 2 uses the normalized displacement vector provided by Phase 1 and stride lengths, estimated by the step counting algorithm to compute the displacement in last second and thus provides the position and velocity vector for Phase 1.

Phase 2 contains two vectors:

$\hat{\vec{x}} = \begin{bmatrix} x & y & z \end{bmatrix}$

$\hat{\vec{v}} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}$

$\hat{\vec{x}}$ and $\hat{\vec{v}}$ represent position and velocity vector respectively.

The normalized displacement vector provided by Phase 1 is:

$\hat{\vec{s}} = \begin{bmatrix} s_x & s_y & s_z \end{bmatrix}$ and we have $\|s\| = 1$ The estimated stride length is denoted as l.

After each second, given $\hat{\vec{s}}$ and l provided by Phase 1 and step counting algorithm, the position vector is updated as:

$$\hat{\vec{x}}_{(+)} = l \times \hat{\vec{s}} + \hat{\vec{x}}_{(-)} \quad \textbf{(31)}$$

$\hat{\vec{x}}_{(-)}$ is the position vector before the update.

$\hat{\vec{x}}_{(+)}$ is the position vector after the update.

As Phase 1 provides the displacement vector once per second, the velocity vector is:

$$\hat{\vec{v}} = l \times \hat{\vec{s}} \quad \textbf{(32)}$$

Then the position vector and velocity vector are provided to Phase 1 as measurements for the next second.

Matlab code for DRA-I is attached as **Appendix 3.**

3.3.3 Wi-Fi fingerprinting

Wi-Fi fingerprinting data is collected by another student, Zhou Wenyi, and K-nearest neighborhood algorithm with the database is implemented on Google App Engine. During a test, the device scans available Wi-Fi information and exchanges the information with the GAE server for location estimation. However, the estimation accuracy is unsatisfactory; therefore, it is not integrated into our algorithm. In next chapter, Wi-Fi fingerprinting estimation is used as one comparison.

3.4 Experiment Results

This section presents the experiment results of the dead reckoning algorithm. In the following figures, the red line is the estimated path and the blue line is the actual path. The first test is 50 m straight walking. The result is shown in **Figure 18**. If we take the estimated points less than 1 m from the actual path as the stable points, it is clear that the algorithm can estimate the actual path well within 10 -15 m, after which, the

accumulated errors will enormously affect the attitude of the object. Without correct attitude estimation, the acceleration and angular velocities in the body coordinate cannot correctly update the prediction in the Kalman filter. Hence, the displacement vector from Phase 1 is no more correct. Even the stride length is well counted, Phase 2 cannot restore the correct direction and therefore the errors accumulate further and eventually lead to the dysfunction of the estimation.

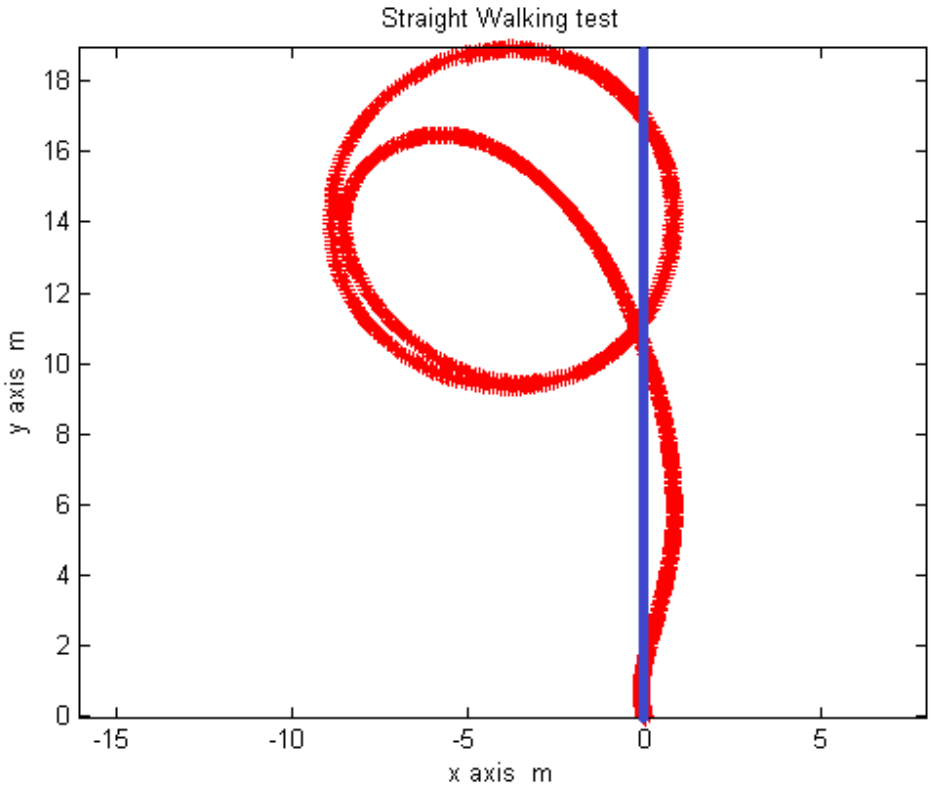

Figure 18: Straight Walking Test

To verify if the dead reckoning can trace turning movements, I walked with the device for 4 m straight, turn left and walked another 4 m straight. The result is shown in **Figure 19**. It can be observed that the path is well recovered by the algorithm, which in a way proves the feasibility and robustness of the proposed algorithm. However, it is worth of mentioning that the circumference is within 15 m.

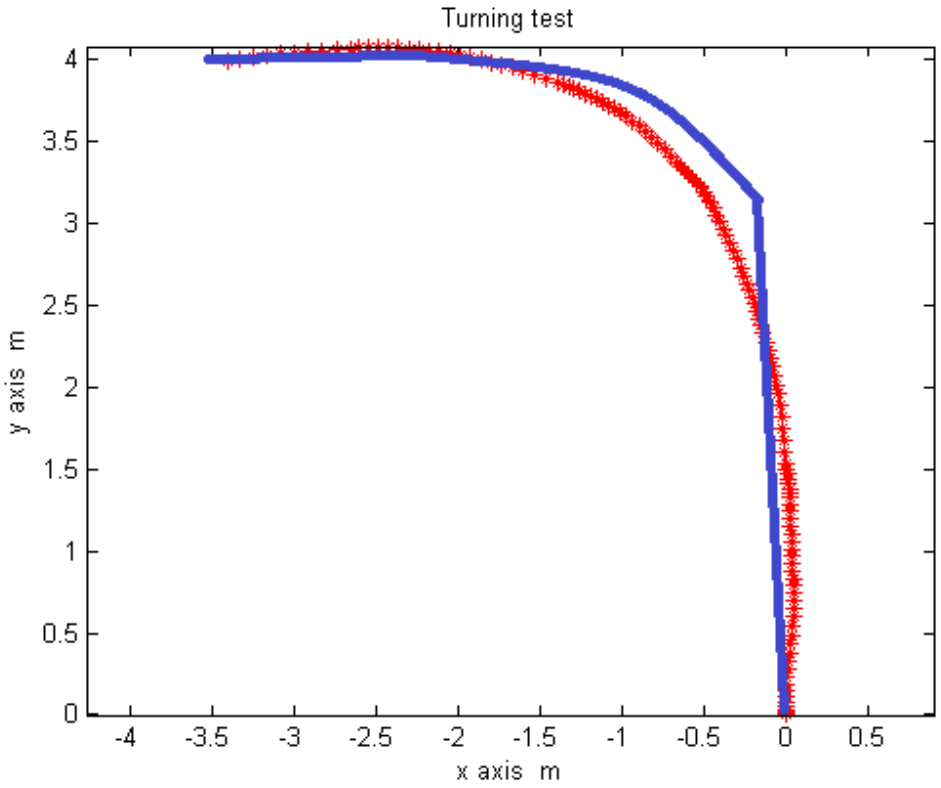

Figure 19: Turning Test

Another important test is the static rotation test, in which the device is rotated by approximately 900 degrees, with very slight translational movement. The calculated result is shown in **Figure 20**.

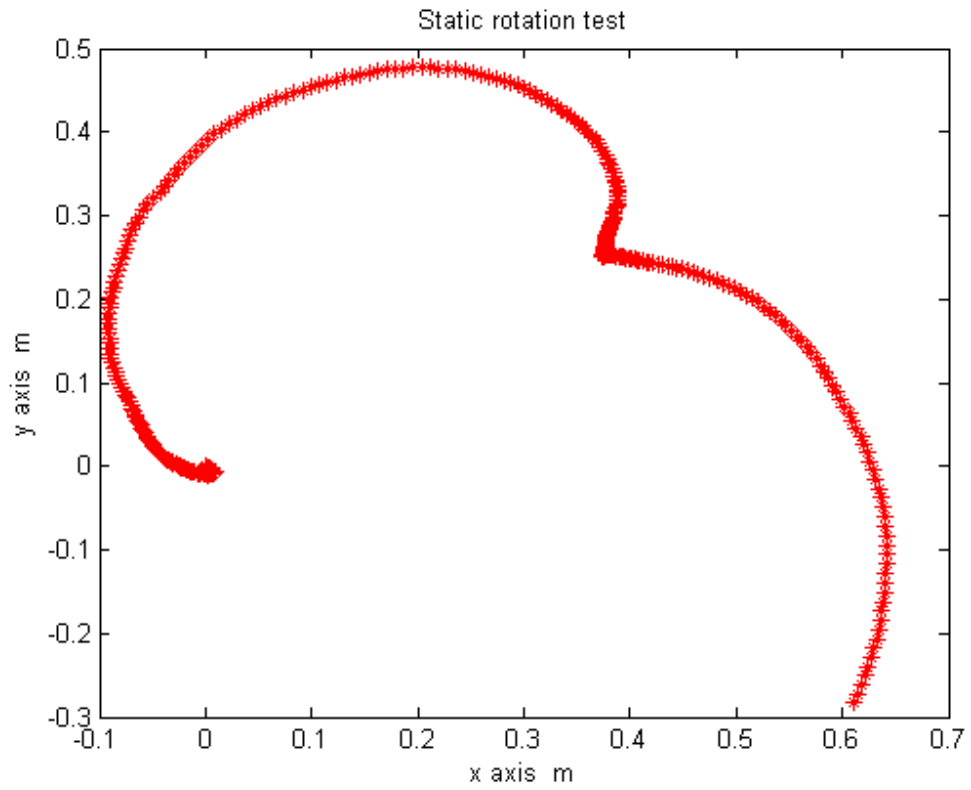

Figure 20: Static Rotation Test

The in-plane rotation is done on a flat table by hand, and it is quite hard to keep the translation to be zero or to note down the actual path. In particular, the device's translation created fluctuations of the accelerometer's output and thus the step counting algorithm is triggered and estimates the resultant stride lengths, which explains the variation of the device's position from the actual path. Therefore, it shall be noted that an obvious drawback for all dead reckoning algorithms based on step counting is that relative movements between a tracking device and a pedestrian might generate inaccuracy. For example, if a pedestrian is standing still and swinging the device back and forth, step-counting algorithms will count steps due to the fluctuation of the accelerometer's output and drift the pedestrian's position vector, as if the

pedestrian is moving. This phenomenon needs to be excluded by an assumption, which will be presented later.

3.5 Discussion

In all of the tests, DRA-I can accurately describe the object's motion within 10-15 m with the combination of its dual phase. Phase 1 can provide a filtered displacement and orientation changes of the last second. Phase 2 uses step-counting method to calculate the stride length and combines the displacement vector from Phase 1 to reset the estimated velocity vector. Thus, Phase 2 can limit the accumulated error in Phase 1.

Although, the algorithm has not been integrated with a correction phase, the successful improvements in dead reckoning, namely extension in stable duration, and reduction in estimation error, gives us a solid foundation for the final fusion algorithm. However, as DRA-I is lacking in attitude recalibration capability, the device's attitude estimation errors keep accumulating and finally make it dysfunctional. In Chapter 4, attitude estimation algorithms will be analyzed and the DRA-II will be presented.

CHAPTER 4 PROPOSED DEAD RECKONING ALGORITHM - II

As aforementioned, accumulated errors of attitude estimation solely by a gyroscope will gradually deteriorate the estimation algorithm and leads to its dysfunction. Therefore, a more robust attitude estimation algorithm is required. There are some existing solutions proposed by other researchers. [12] and [15] proposed solutions to mount the sensors on shoes to force the heading direction to be aligned with a fixed direction of the sensors, say the y-axis of the magnetometer. The heading direction can be thus estimated from the magnetometer's output. [13] applies a similar principle, by restricting the way that the device is held, so that the heading direction of pedestrian is aligned with yaw direction of the device. Unlike [12][13][15], [27][28][29] have no such restrictions, but rather they fusion outputs from accelerometer, gyroscope and magnetometer to have an accurate attitude estimation. As the objective of this project is to permit the device to be freely held or put into the pocket or bag, restrictions on how to hold the device shall be released. Thus, a method similar to [27][28][29] is applied and the details are explained in the following section.

4.1 Attitude Estimation Algorithm

4.1.1 Algorithm Description

In the attitude determination algorithm, the signals of the magnetometer, gyroscope, and accelerometer are combined and filtered by an extended Kalman filter. The state vector of the system is

$$X = \begin{pmatrix} q_0 & q_1 & q_2 & q_3 & \omega_x & \omega_y & \omega_z \end{pmatrix}^T$$

Vector $\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$ is the quaternion vector in the global coordinate.

Vector $\begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ is the angular velocity vector in the body coordinate.

The quaternion matrix dynamic evolution [23] is as follows:

$$\begin{cases} \dot{q}_0 = -\frac{1}{2} q_1 \omega_x - \frac{1}{2} q_2 \omega_y - \frac{1}{2} q_3 \omega_z \\ \dot{q}_1 = +\frac{1}{2} q_0 \omega_x - \frac{1}{2} q_3 \omega_y + \frac{1}{2} q_2 \omega_z \\ \dot{q}_2 = +\frac{1}{2} q_3 \omega_x + \frac{1}{2} q_0 \omega_y - \frac{1}{2} q_1 \omega_z \\ \dot{q}_3 = -\frac{1}{2} q_2 \omega_x + \frac{1}{2} q_1 \omega_y + \frac{1}{2} q_0 \omega_z \end{cases} \quad \textbf{(33)}$$

Vector $\begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ on the left hand side are taken as constant and the relation between $\begin{bmatrix} \dot{q}_0 & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 \end{bmatrix}^T$ and $\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$ are thus linear

The angular velocity dynamic evolution is:

$$\begin{cases} \dot{\omega}_x = -\frac{1}{\tau} \omega_x \\ \dot{\omega}_y = -\frac{1}{\tau} \omega_y \\ \dot{\omega}_z = -\frac{1}{\tau} \omega_z \end{cases} \quad \textbf{(34)}$$

It means if there is no new measurement, the angular velocity will gradually decrease to zero. The system dynamic model is linear. Therefore, we can rewrite the model as:

$$\dot{X} = AX(t) + W(t) \quad \textbf{(35)}$$

Matrix A is described as:

$$A = \begin{bmatrix} 0 & -\frac{1}{2}\omega_x & -\frac{1}{2}\omega_y & -\frac{1}{2}\omega_z & 0 & 0 & 0 \\ \frac{1}{2}\omega_x & 0 & \frac{1}{2}\omega_z & -\frac{1}{2}\omega_y & 0 & 0 & 0 \\ \frac{1}{2}\omega_y & -\frac{1}{2}\omega_z & 0 & \frac{1}{2}\omega_x & 0 & 0 & 0 \\ \frac{1}{2}\omega_z & \frac{1}{2}\omega_y & -\frac{1}{2}\omega_x & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{\tau} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau} \end{bmatrix} \quad \textbf{(36)}$$

$W(t)$ is as assumed to be zero mean white Gaussian random noise. The measurement vector is:

$$Z = \begin{pmatrix} a_x & a_y & a_z & \omega_x & \omega_y & \omega_z & m_x & m_y & m_z \end{pmatrix}^T$$

Vector $\begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$ is the acceleration measurement vector measured by the accelerometer.

Vector $\begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ is the angular velocity measurement vector measured by the gyroscope.

Vector $\begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T$ is the magnetic field measurement vector measured by the magnetometer.

As for magnetic measurement and magnetic field in global coordinate, we have:

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = R_{rot} \begin{bmatrix} m_{x0} \\ m_{y0} \\ m_{z0} \end{bmatrix} + W_{mV} \tag{37}$$

$\begin{bmatrix} m_{x0} & m_{y0} & m_{z0} \end{bmatrix}^T$ is the magnetic field strength in the global coordinate. In this project, we have:

$$\begin{bmatrix} m_0 \\ m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} 38.1994 \\ -13.9035 \\ -11.036 \end{bmatrix} \tag{38}$$

The unit of this vector is μT. R_{rot} is the rotation matrix from the body coordinate to the global coordinate, specified similarly as:

$$\begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_2) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \tag{39}$$

For the accelerometer, we have:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = R_{rot} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + W_{aV} \tag{40}$$

For the gyroscope:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + W_{\omega V} \tag{41}$$

W_{mV}, W_{aV} and $W_{\omega V}$ are measurement noises and they are modeled as zero mean white Gaussian noises.

The relationship between measurement vector and state vector are obviously non-linear. Therefore, linearization is needed. The linearized model is:

$$Z = CX(t) + V(t) \tag{42}$$

where $V(t)$ is the measurement noise and is presumed to be zero mean white Gaussian noise. C is the Jacobian matrix from Z to X:

$$C = \frac{\partial Z}{\partial X} \tag{43}$$

and C is as follows:

$$C = \begin{bmatrix} -2q_2g & 2q_3g & -2q_0g & 2q_1g & 0 & 0 & 0 \\ 2q_1g & 2q_0g & 2q_3g & 2q_2g & 0 & 0 & 0 \\ 2q_0g & -2q_1g & -2q_2g & 2q_3g & 0 & 0 & 0 \\ 2q_0b_{x0} - 2q_2b_{z0} + 2q_3b_{y0} & 2q_1b_{x0} + 2q_2b_{y0} + 2q_3b_{z0} & 2q_1b_{y0} - 2q_0b_{z0} - 2q_2b_{x0} & 2q_0b_{y0} + 2q_1b_{z0} - 2q_3b_{x0} & 0 & 0 & 0 \\ 2q_0b_{y0} + 2q_1b_{z0} - 2q_3b_{x0} & 2q_0b_{z0} - 2q_1b_{y0} + 2q_2b_{x0} & 2q_1b_{y0} + 2q_2b_{x0} + 2q_3b_{z0} & 2q_2b_{z0} - 2q_0b_{x0} - 2q_3b_{y0} & 0 & 0 & 0 \\ 2q_0b_{z0} - 2q_1b_{y0} + 2q_2b_{x0} & 2q_3b_{x0} - 2q_1b_{z0} - 2q_0b_{y0} & 2q_0b_{x0} - 2q_2b_{z0} + 2q_3b_{y0} & 2q_1b_{x0} + 2q_2b_{y0} + 2q_3b_{z0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The measurements in our case are discrete signals, so we need to transform our model to a discrete time model. For any continuous time dynamic model [23] given by

$$\begin{cases} \dot{X} = AX(t) + w(t) \\ \dot{Z} = CX(t) + v(t) \end{cases} \tag{44}$$

its discrete time model is:

$$\begin{cases} X_{k+1} = A_d X_k + W_d(t) \\ Z_k = h_k X_k + v_k \end{cases} \tag{45}$$

where $A_d = e^{Adt}$ and $W_d(k) = \int_0^{dt} e^{Av} w((k+1)dt - v)dx$

By Taylor expansion,

$$A_d = I + Adt \tag{46}$$

$$W_d = Wdt + (AW + (AW)^T)\frac{dt^2}{2} + AWA^T\frac{dt^3}{3} \tag{47}$$

The third factor of W_d is to ensure that the noise matrix is positive definite. Matlab code for the proposed attitude algorithm is attached as **Appendix 4**.

4.1.2 Experimental Results

Testing on the attitude algorithm is conducted on the Samsung Tab. As discussed in section 2.2, the measurement errors are modeled as offsets of each of the axes, and zero mean white Gaussian noise. Experiments for static cases and walking cases have been conducted in the AMI lab. Although quaternion is used to represent the device's attitude, it is not intuitive to represent the error between the attitude estimation and ground truth. Therefore, quaternion is converted into Tai-Bryan angles representation, say yaw, roll and pitch angles and the errors are the differences between the angles estimation with the ground truth. Static results are shown in **Figures 21**, **22** and **23**, for the error estimation for yaw, pitch and roll respectively.

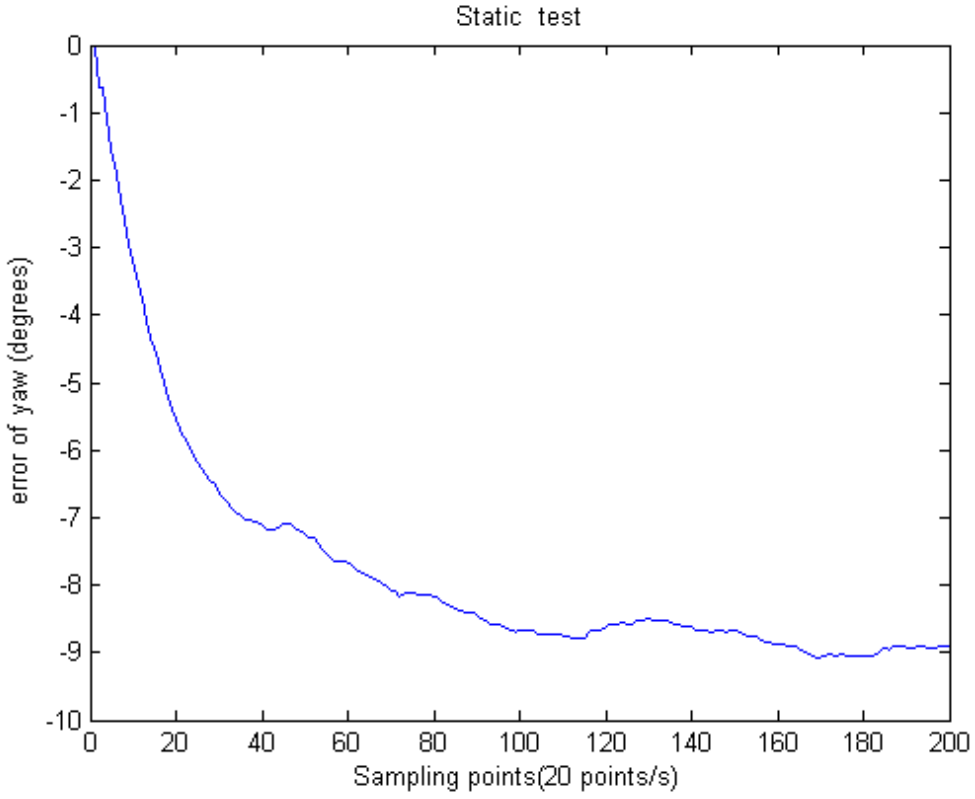

Figure 21: Static Case Test Yaw Angle Error

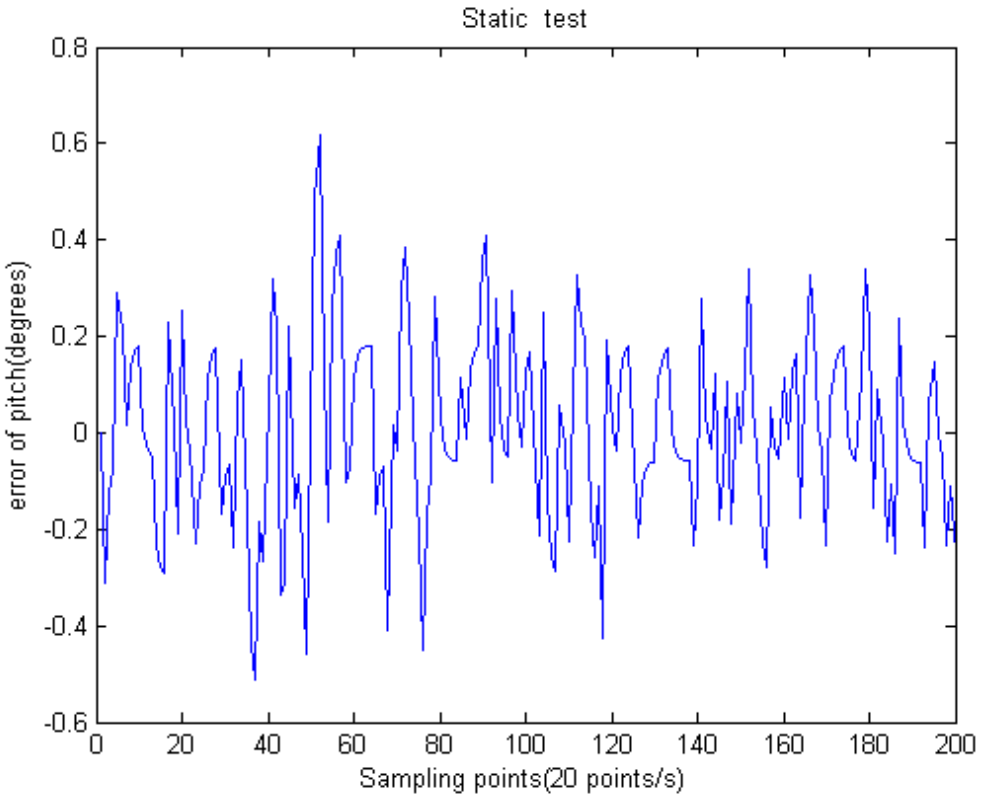

Figure 22: Static Case Test Pitch Angle Error

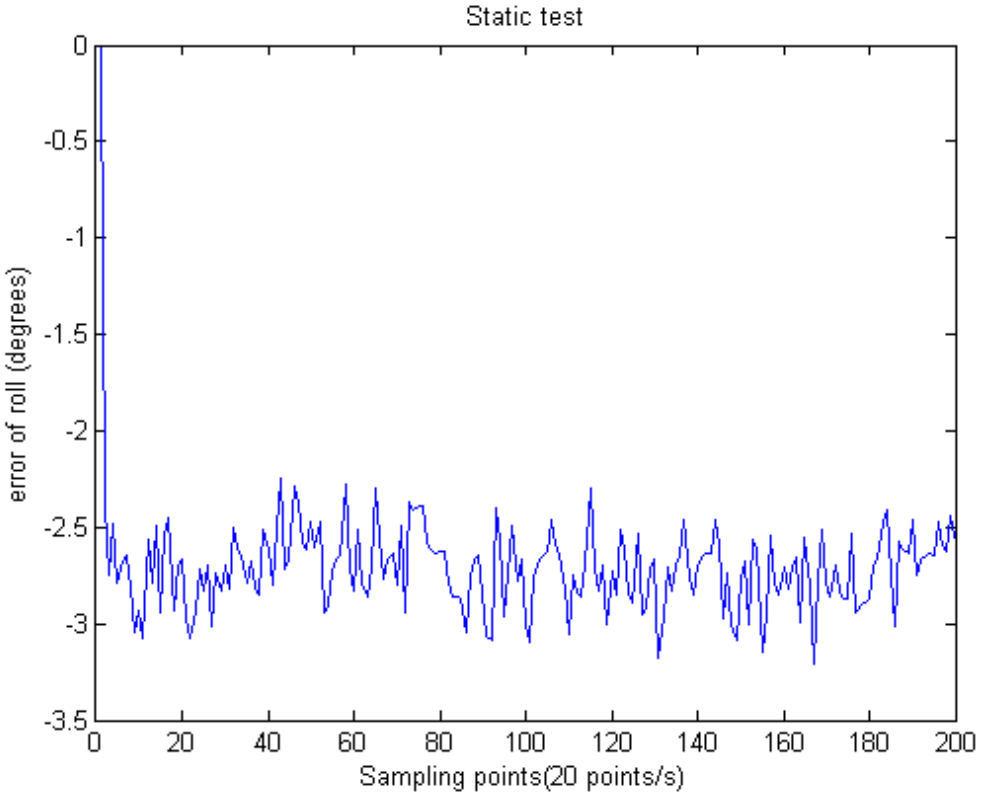

Figure 23: Static Case Test Roll Angle Error

From Figure 22, the pitch angle errors are within 0.6 degrees and can thus be considered as negligible. Roll angles and yaw angles are much larger. As shown in section 2.3, the accelerometer and magnetometer measurement errors are much more complicated than what our model has described by a bias with zero mean Gaussian variable for each axis; especially for the magnetometer which is seriously affected by external interferences. Attitude estimation errors are due to the error modeling simplification. Estimation error variations are due to measurement noise of the sensors, which are modeled as white Gaussian noise. Although errors are not fully eliminated, errors variations are small, which proves the accuracy and robustness of the attitude estimation algorithm. With a better calibration algorithm and more complicated error modeling, the results will be improved, as shown in the DRA-III.

The walking case is then conducted with a pedestrian holding the device by hand. In this test, attitude estimation errors are expected to be much larger than those in static case, as the accelerometer's measurement is not only due to gravitational force but also the pedestrian. This phenomenon is named as corruption effect, i.e. the accelerometer's measurement is corrupted.

To benchmark the device's attitude during the test, the device is firstly held with its body coordinate aligned with the global coordinate. Its Y-axis is the heading direction and Z-axis points downward. During the test, roll and pitch angles are zero and the yaw angle depends on the pedestrian's heading direction. The pedestrian walks along a path similar to a U-turn, first along the +Y-axis of the global, and then turning right by 90 degrees along the +X-axis before turning right again by 90 degrees and along –Y-axis. Theoretically, the device's pitch and roll are zero. However, as the device is hand-held, small rotations around the X-axis and Y-axis of its body coordinates due to pedestrian movement is unavoidable, so roll and pitch angles are expected to fluctuate around zero. The same logic also applies to the yaw angle. While we assume the ground truth changes from 0 to 90, and then to -180 degrees, the device's true yaw angles might have small variations from these values. Besides, for comparison purpose, as the device is held with Y-axis of its body coordinate as the heading direction, we can also compute its heading direction by the magnetometer alone. The algorithm has no restriction on the way of holding the device, and it is only for attitude benchmark purpose. The experimental results for roll angle, pitch angle and yaw angle are shown in **Figures 24**, **25**, **26** respectively.

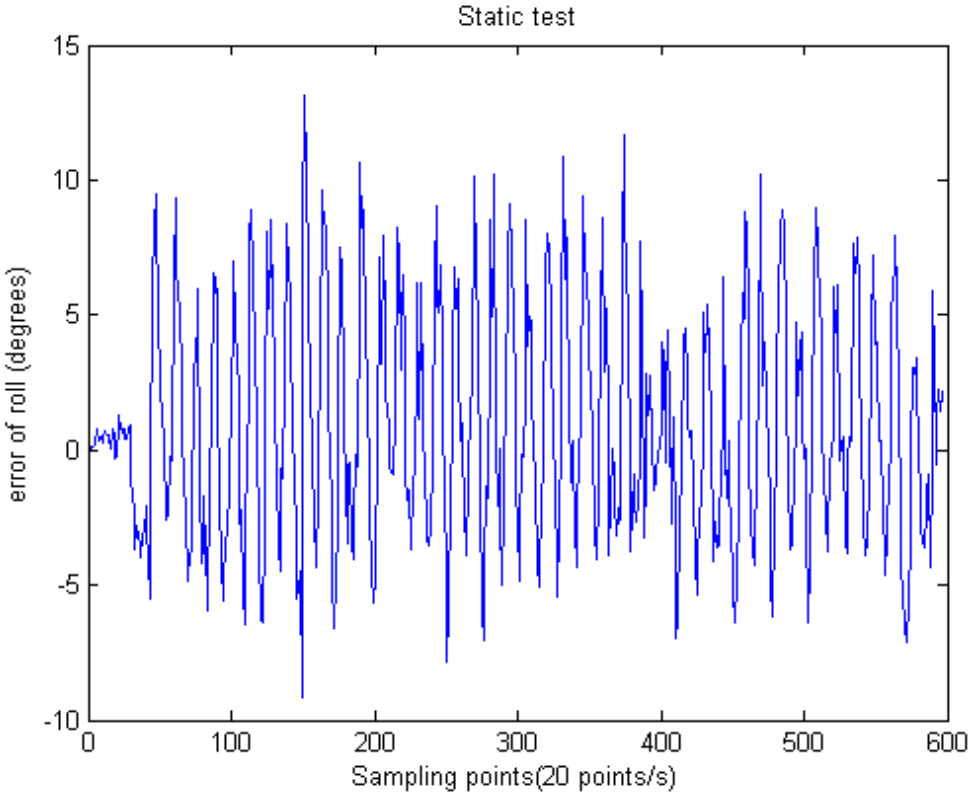

Figure 24:Walking Test Roll Angle Error

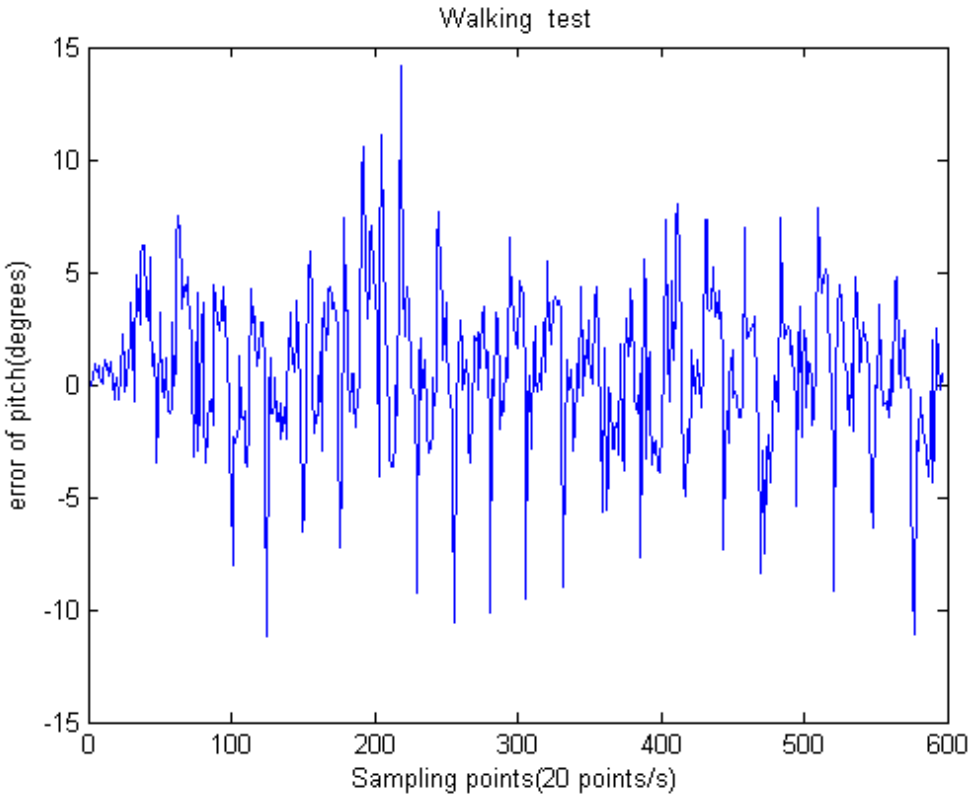

Figure 25: Walking Test Pitch Angle Error

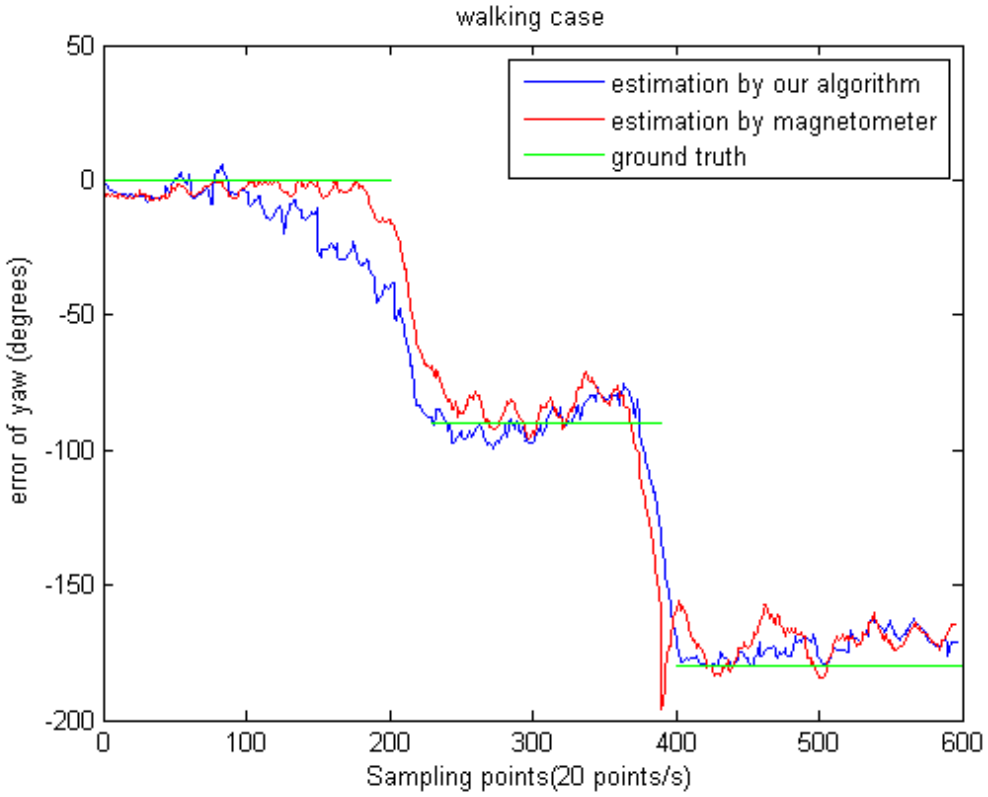

Figure 26: Yaw Angle in Walking Test

From **Figures 24** and **25**, the variations of the devices' yaw and roll angles from zero values are much larger than those of the static cases. One explanation is that variations are due to the corruption effect, as stated in the previous paragraph. As the accelerometer's outputs are corrupted by the pedestrian's acceleration, the accelerometer's measurement is no longer pointing downward as assumed in our attitude estimation algorithm, so the algorithm's accuracy is decreased and the variations indicate that the algorithm performance has deteriorated. Besides, during the test, the pedestrian will unintentionally rotate the device around its X and Y-axis, which will induce variations in the roll and pitch angles. These rotations can be observed from the gyroscope's X and Y-axis outputs, shown in **Figure 27** and **Figure 28** correspondingly. While roll and pitch variations due to the corruption effect

suggests deterioration of the algorithm's performance, the variations due to small rotations of the device proves the robustness of our algorithm. A supplementary test is conducted and will be presented later for parametric sensitivity study.

Figure 26 relates to the yaw angle. The green line shows the ground truth of the heading direction. The red line is the heading direction computed by the magnetometer output. Our algorithm has comparable performance with respect to magnetometer-only estimation and it is more robust to sharp magnetic field interferences as shown by the 400pts of X-axis in **Figure 27**. Besides, the magnetometer-only algorithm restricts the way of holding the device and limits its practical usage, which is successfully avoided in the proposed algorithm.

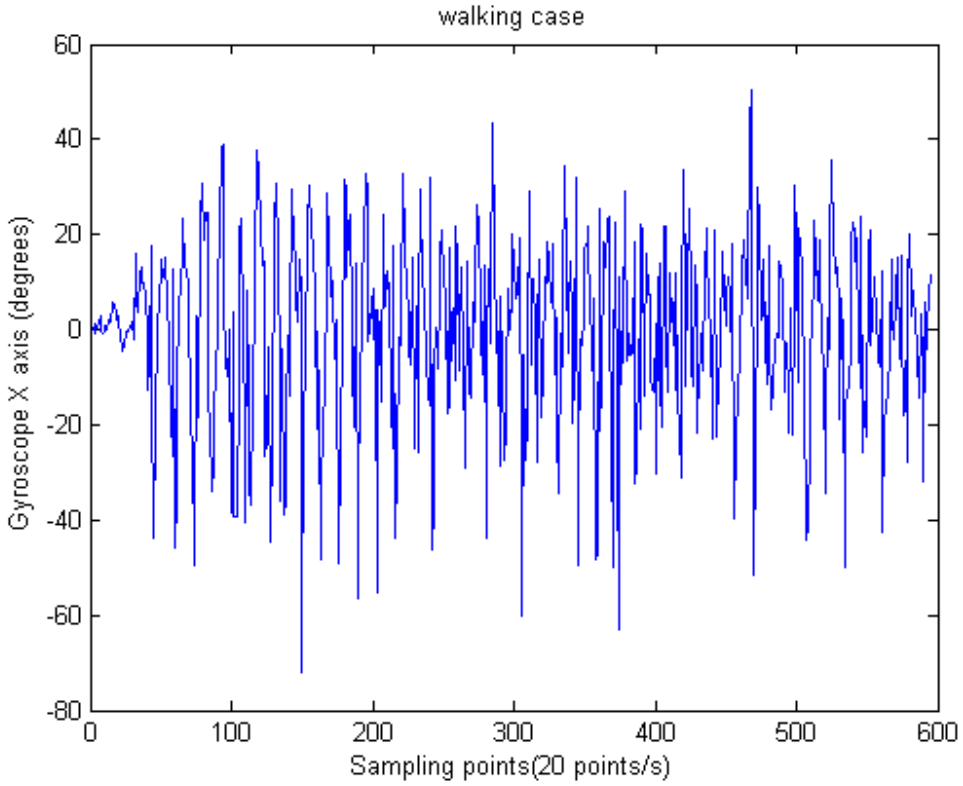

Figure 27: Walking Case Gyroscope X-axis Output

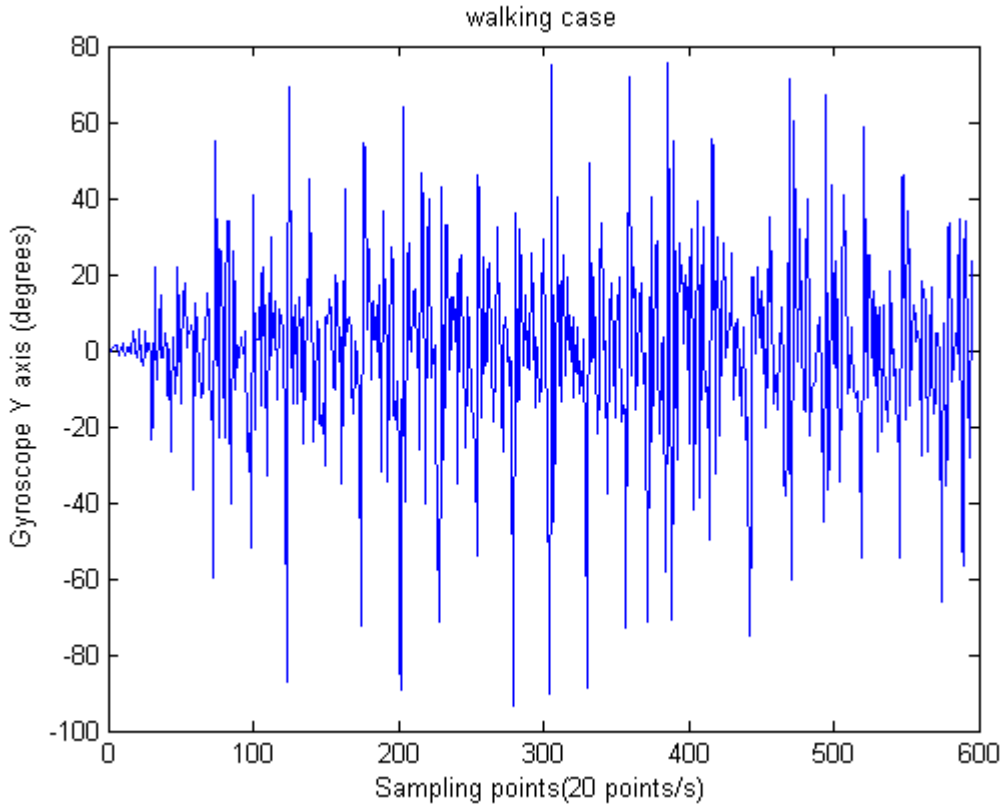

Figure 28: Walking Case Gyroscope Y-axis Output

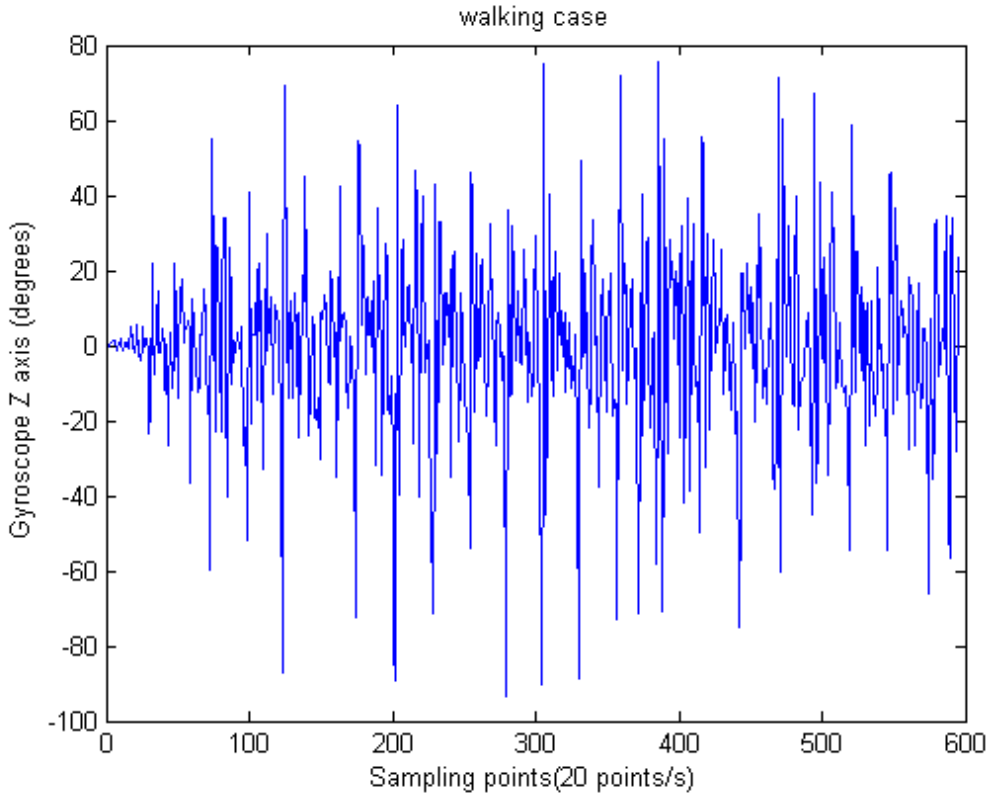

Figure 29: Walking Case Gyroscope Z-axis Output

To explore the attitude estimation errors due to the corruption effect, a supplementary test is set up. In this test, the device is put on top of a table with its body coordinate aligned with the global coordinate. As such the accelerometer's X-axis and Y-axis outputs are zero and the Z-axis output is the gravitational acceleration. To simulate the pedestrian's acceleration to corrupt the other axes' output, move the device along its Y-axis of the global coordinate. The device's accelerometer X-axis,Y-axis and Z-axis outputs are shown in **Figure 30**, **31** and **32** respectively. It is clear that the outputs have been corrupted by $\pm 3m/s^2$ on X-axis and Y-axis, similar to the walking case.

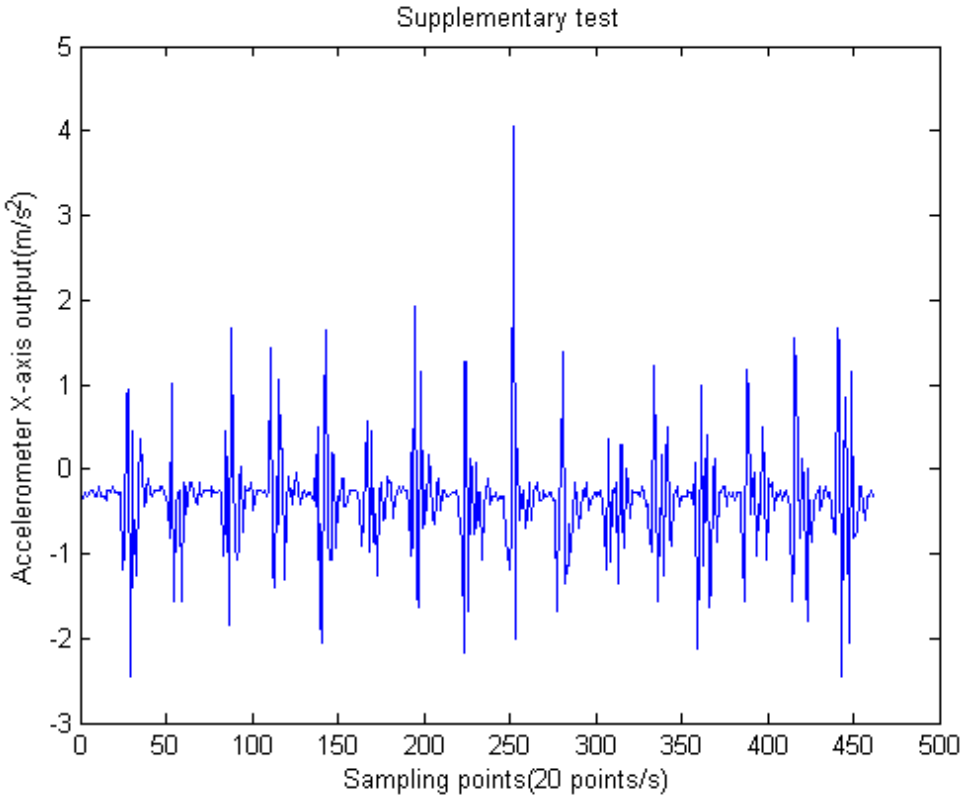

Figure 30: Supplementary Test Accelerometer X-axis Output

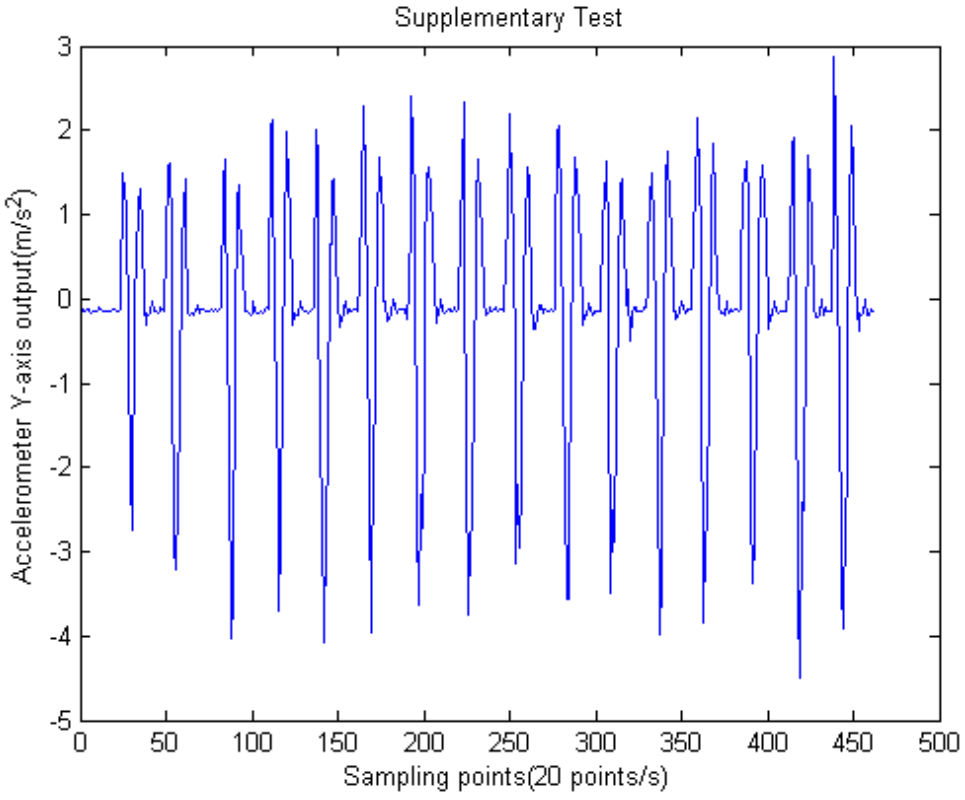

Figure 31: Supplementary Test Accelerometer Y-axis Output

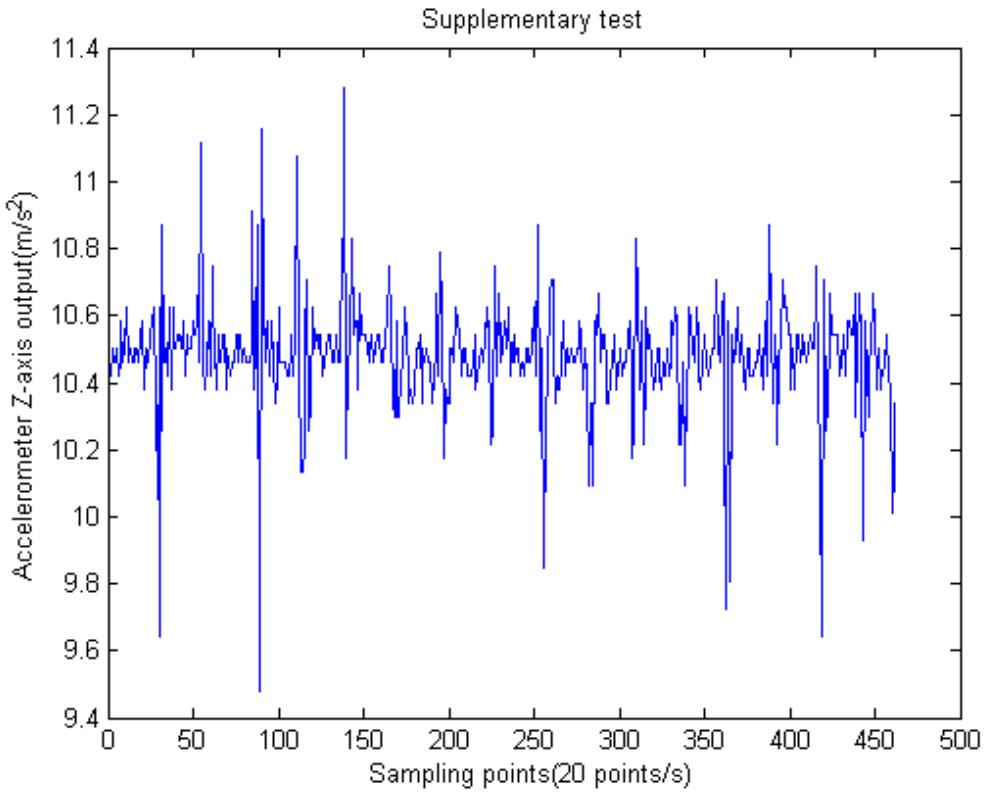

Figure 32: Supplementary Test Accelerometer Z-axis Output

The attitude errors of roll and pitch angles of the supplementary test are shown in **Figures 33** and **34**.

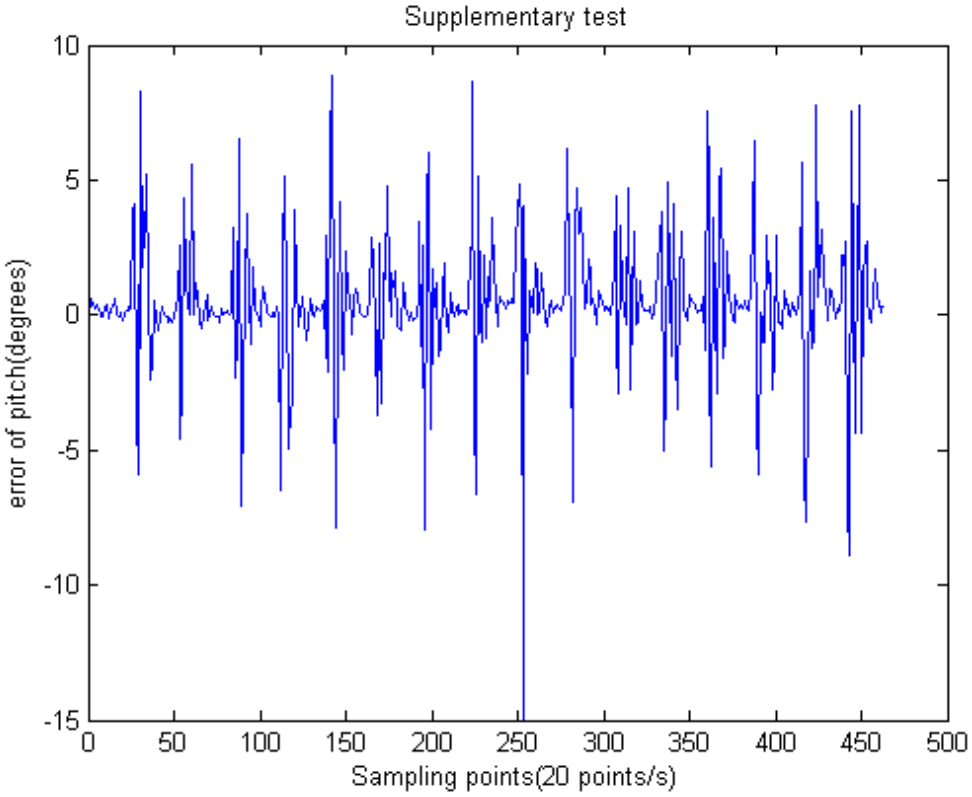

Figure 33: Supplementary Test Pitch Angle Errors

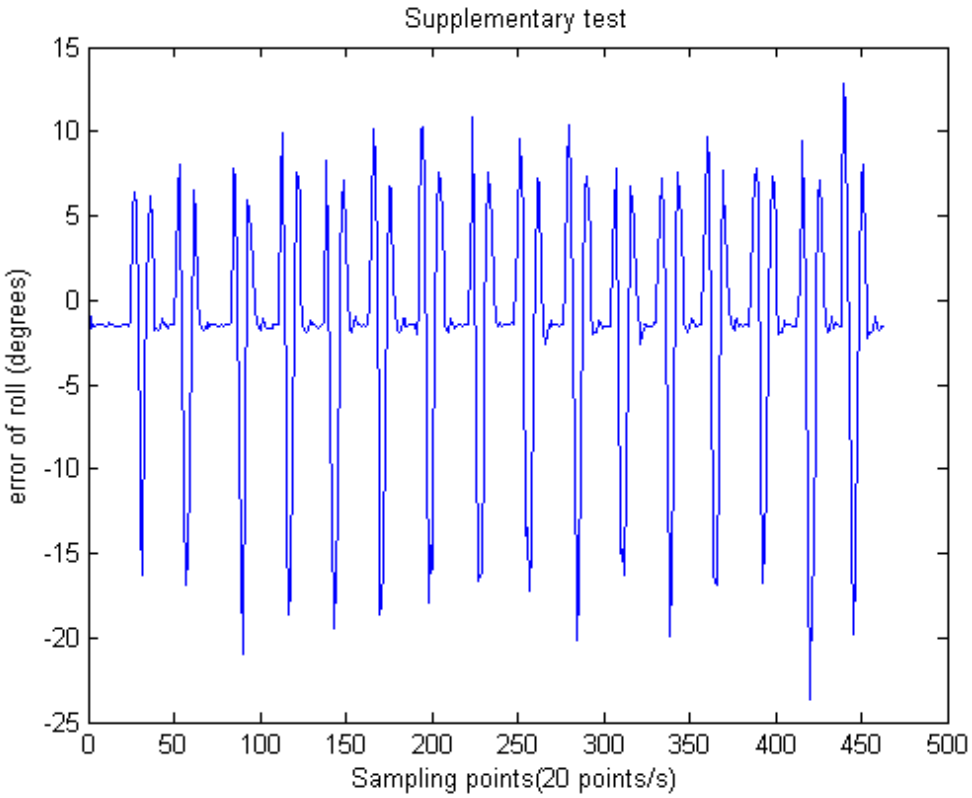

Figure 34: Supplementary Test Roll Angle Errors

The errors shown in **Figure 33** and **Figure 34** are only due to the corruption effect and they are similar to roll and pitch errors in previous walking case. Therefore, we can conclude that the roll and pitch errors due to the corruption effect in the walking test are significant. These errors will not affect heading direction determination. However, they will affect how to project the gravitational force onto the X-axis and Y-axis. If velocity and location are determined by integrating accelerometer's outputs, the projected gravitational force will significantly affect the results and make them unreliable.

4.2 Integrated Dead Reckoning Algorithm with Attitude Estimation

The attitude estimation algorithm is then integrated into the DRA-I. This section gives a brief introduction on the new algorithm and then presents its experiment results.

4.2.1 Algorithm Description

The state vector is the same as in the DRA-I:

$$X = \begin{pmatrix} x & y & z & u & v & w & a_x & a_y & a_z & q_0 & q_1 & q_2 & q_3 & w_x & w_y & w_z \end{pmatrix}^T \quad \textbf{(48)}$$

The system dynamic equations are the same as the equation **(15)**. The state transition matrix is the same as **(29)**.

The measurement vector is:

$$Z = \begin{bmatrix} x & y & z & u & v & w & a_x & a_y & a_z & na_x & na_y & na_z & m_x & m_y & m_z & \omega_x & \omega_z & \omega_z \end{bmatrix}^T$$

The vector $\begin{bmatrix} na_x & na_y & na_z \end{bmatrix}^T$ is the normalized acceleration measurement. Other vectors are as stated before.

Unlike in DRA-I, the relation between the observation and the state vector is non-linear. The observation matrix C is the Jacobian matrix from Z to X, as described by

$$C = \frac{\partial Z}{\partial X} \tag{49}$$

C is as follows:

$$C = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -2q_2g & 2q_3g & -2q_0g & 2q_1g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2q_1g & 2q_0g & 2q_3g & 2q_2g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2q_0g & -2q_1g & -2q_2g & -2q_3g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \begin{matrix}2q_2m_{x0} - 2q_2m_{z0} \\ +2q_3m_{y0}\end{matrix} & \begin{matrix}2q_1m_{x0} + 2q_2m_{y0} \\ +2q_3m_{z0}\end{matrix} & \begin{matrix}2q_1m_{y0} - 2q_0m_{z0} \\ -2q_2m_{x0}\end{matrix} & \begin{matrix}2q_0m_{y0} + 2q_1m_{z0} \\ -2q_3m_{x0}\end{matrix} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \begin{matrix}2q_0m_{y0} + 2q_1m_{z0} \\ -2q_3m_{x0}\end{matrix} & \begin{matrix}2q_0m_{z0} - 2q_1m_{y0} \\ +2q_2m_{x0}\end{matrix} & \begin{matrix}2q_1m_{y0} + 2q_2m_{x0} \\ +2q_3m_{z0}\end{matrix} & \begin{matrix}2q_2m_{z0} - 2q_0m_{x0} \\ -2q_3m_{y0}\end{matrix} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \begin{matrix}2q_0m_{z0} - 2q_1m_{y0} \\ +2q_2m_{x0}\end{matrix} & \begin{matrix}2q_3m_{x0} - 2q_1m_{z0} \\ -2q_0m_{y0}\end{matrix} & \begin{matrix}2q_0m_{x0} - 2q_2m_{z0} \\ +2q_3m_{y0}\end{matrix} & \begin{matrix}2q_1m_{x0} + 2q_2m_{y0} \\ +2q_3m_{z0}\end{matrix} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}$$

The extended Kalman filter is applied as the fusion algorithm, due to the non-linearity of system dynamic model and the relation between the state vector and the measurement vector. Similar to the DRA-I, two phases are implemented. Phase1 provides displacement direction and Phase 2 combines stride length estimated by step counting and the normalized displacement vector and provides position and velocity estimation to Phase1.

Matlab code for this algorithm is attached as **Appendix 5**.

4.2.2 Experimental Results and Discussion

In the following figures, the green lines are the ground truth of the pedestrian's path. The red stars are locations estimated by Wi-Fi fingerprinting. The red lines are the pedestrian's paths estimated by Wi-Fi fingerprinting. The proposed algorithm output is shown in Phase1 (blue points) and Phase 2 (black points) output. The pedestrian's paths for all the three figures are the same. The first two figures are tests conducted in the same condition and the third figure shows one of the tests with strong magnetic interference at the starting point.

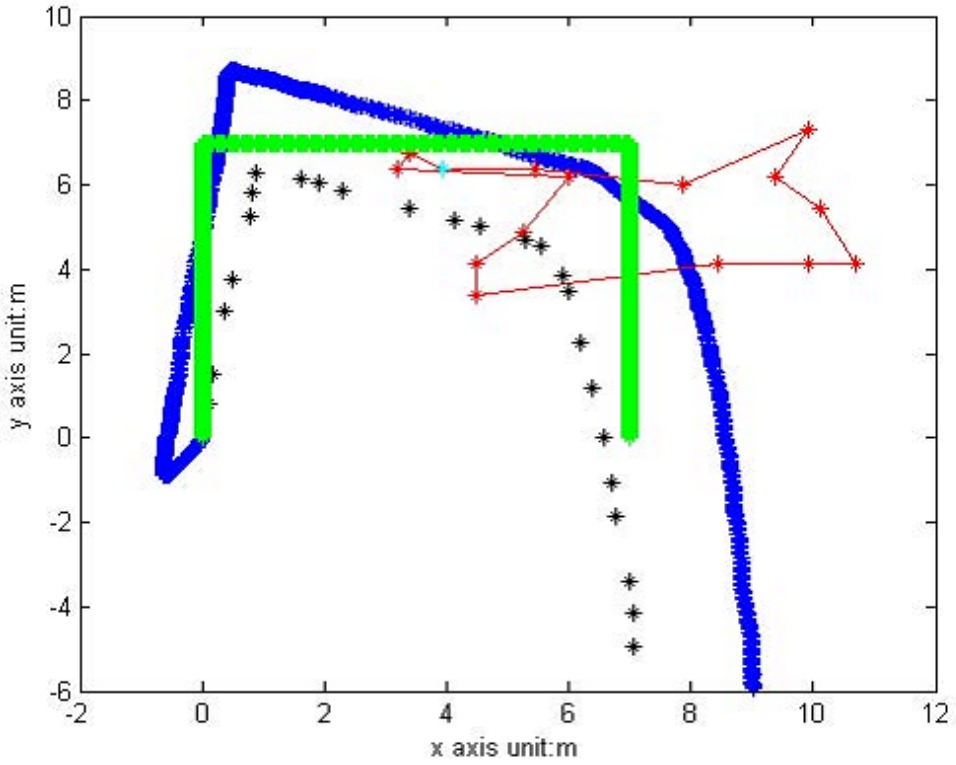

Figure 35: Waking Test of the Integrated Algorithm

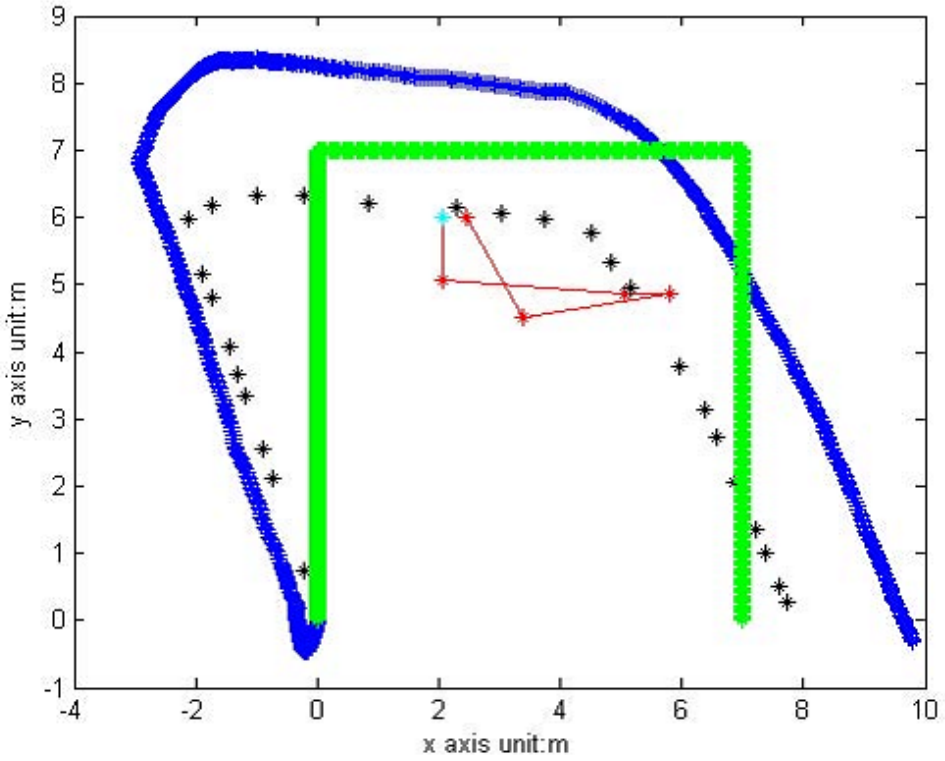

Figure 36: Waking Test of the Integrated Algorithm

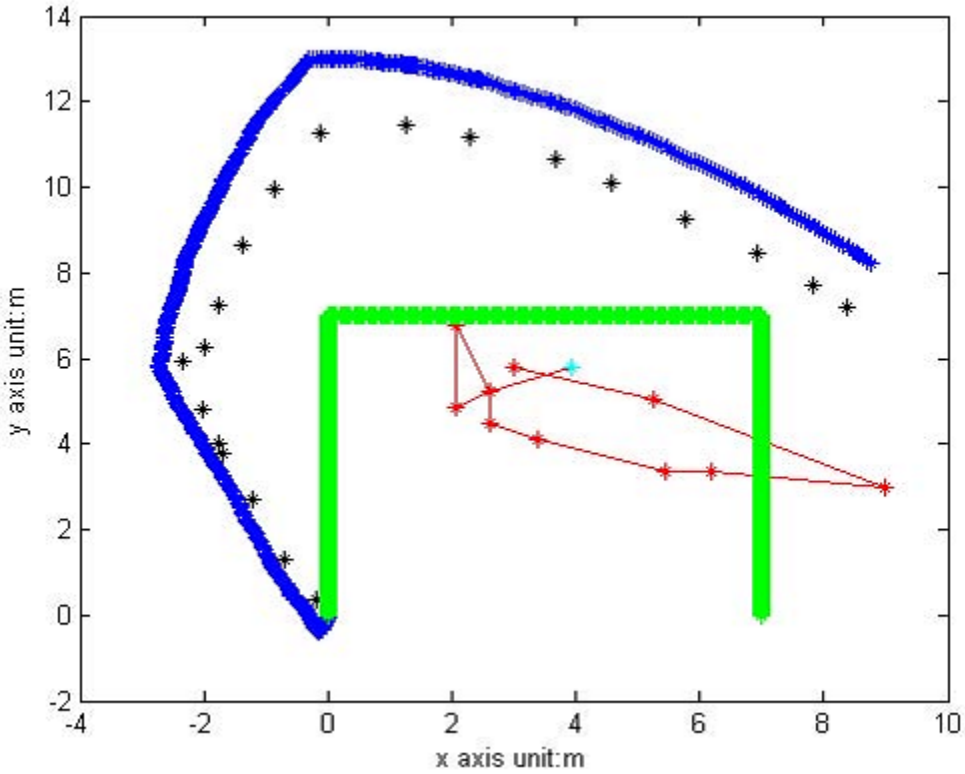

Figure 37: Waking Test with Strong Initial Magnetic Interference

From the above figures, it can be concluded that the algorithm is very sensitive to magnetic interference due to the correlation between Phase1 and Phase 2. Phase 2 computes the device's velocity vector by the normalized displacement vector from Phase 1 and then provides the velocity vector as measurement to Phase1. However, Phase 1's displacement is correlated with the velocity measurement; therefore, if magnetic interference affects the heading direction estimation during the walking test, it takes quite a long time to recalibrate the heading direction.

One possible solution to eliminate the correlation is to apply Principle Component Analysis (PCA) on the accelerometer's signals for heading direction estimation in Phase 2. This requires accurate attitude estimation, such that the gravitational acceleration component in accelerometer's measurements can be eliminated and the direction with largest acceleration (after gravitational acceleration being eliminated) variation is the heading direction. However, our attitude algorithm cannot provide sufficient attitude estimation to perform PCA due to the corruption effect during the walking case; thus the device's attitude is not sufficiently accurate to totally eliminate gravitational acceleration and to perform PCA. Besides, the BMA150 accelerometer's low resolution, and high measurement errors also make the implementation of PCA difficult on this device. Therefore, another practical modification on Phase 2 is proposed and introduced in section 4.3.

4.3 DRA-II

4.3.1 Algorithm Description

DRA-II is designed to resist magnetic interference with a basic assumption that there is no relative movement between the device and the pedestrian. Two examples are given to elaborate this assumption.

Example 1: If the device is put into the pocket in the calibration step and the pedestrian wants to take out and hold the device in hand, then the calibration step shall be redone, as it violates our assumption

Example 2: If the device is hand-held with its Y-axis pointing in the heading direction, but the pedestrian rotates the device and lets its X-axis be the heading direction, the calibration step shall be retriggered.

In this algorithm, no restriction on the way of holding the device has been imposed and the device can be freely mounted to the pedestrian through hand-holding and pocket or bag carrying. The only requirement is that there should be no more relative movement after the calibration stage. Although the assumption seems restrictive, it is reasonable. As discussed in Section 3.1, if relative movements exist, for example the pedestrian just swing the device without moving, the step counting estimation of stride lengths will deteriorate. Another reason supporting this assumption will be presented in Chapter 5.

The calibration step algorithm is the same as the attitude estimation algorithm presented in Section 4.1. During the calibration, the pedestrian is required to face a

certain direction and stand still. The direction can be chosen freely, whether the North direction or a fixed direction in the global coordinate. In this project, the direction is chosen to be Y-axis of the global coordinate. The calibration step algorithm needs as short as 2s to estimate the device's attitude, which has been demonstrated in the experiments. However, as the magnetometer scans the result 10 times per second, 5 s is chosen to have at least 50 magnetic field measurements for calibration purpose. At the end of the calibration step, the device's attitude is recorded as the reference quaternion, $\begin{bmatrix} q_{r0} & q_{r1} & q_{r2} & q_{r3} \end{bmatrix}^T$.

The device calibration is followed by tracking step which contains two phases, denoted as Phase 1 and Phase 2. Phase 1 is the same as the integrated algorithm. Phase 2 is different in terms of the determination of the heading direction. Rather than determining the heading direction by the given normalized displacement vector from Phase 1, the heading direction is computed by comparing the quaternion given by Phase 1 and the reference quaternion. Phase 2 algorithm is elaborated as follows.

Phase 2 has two vectors:

$\begin{bmatrix} x & y & z \end{bmatrix}^T$ is the position vector, which is then provided to Phase 1 as position measurement.

$\begin{bmatrix} u & v & w \end{bmatrix}^T$ is the velocity vector, which is then provided to Phase 2 as velocity measurement.

From the output of Phase 1, the device's quaternion is calculated, noted as $\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$. We can convert it to Tait-Bryan angle representation.

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} a\tan 2(2(q_o q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \arcsin(2(q_o q_2 - q_3 q_1)) \\ a\tan 2(2(q_o q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix} \quad \textbf{(50)}$$

ψ is the yaw angle, the rotation about the x axis. $\text{atan}2$ is a variation of the $\arctan$ function and it is defined as :

$$a\tan 2(x,y) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \textit{undefined} & y = 0, x = 0 \end{cases} \quad \textbf{(51)}$$

From the device's reference quaternion $\begin{bmatrix} q_{r0} & q_{r1} & q_{r2} & q_{r3} \end{bmatrix}^T$, we can convert it to be Tait-Bryan representation as well, noted as $\begin{bmatrix} \phi_r & \theta_r & \psi_r \end{bmatrix}^T$.

The difference of the yaw angles is the difference between the current heading direction and the reference heading direction. The reference heading direction is known as Y-axis of the global coordinate in this project. Suppose the yaw angle difference is $\Delta\psi$ and the stride lengths estimation by step counting algorithm is l, the new position and velocity vector is:

$$\begin{bmatrix} u(+) & v(+) & w(+) \end{bmatrix}^T = \begin{bmatrix} l*\sin\Delta\psi & l*\cos\Delta\psi & 0 \end{bmatrix}^T \quad \textbf{(52)}$$

$$\begin{bmatrix} x(+) & y(+) & z(+) \end{bmatrix}^T = \begin{bmatrix} x(-)+l*\sin\Delta\psi & y(-)+l*\cos\Delta\psi & z(-) \end{bmatrix}^T \quad \textbf{(53)}$$

The two vectors will then be provided to Phase 1 as measurement. Matlab code for DRA-II is attached as **Appendix 6**.

4.3.2 Experimental Results

Various tests for DRA-II have been conducted as shown in the following figures. Green lines stand from ground truths, while Phase1's outputs and Phase2's outputs are shown as blue lines and black stars respectively. **Figure 38** shows the experiment results for a test in which the device's body coordinate is aligned with the global coordinate and the pedestrian walks for 50m. **Figure 39** and **40** show experiment results for a U-turn path. In **Figure 39**, the device is held with its body coordinate aligned with the global coordinate. In **Figure 40**, to prove that our algorithm does not require a fixed way to hold the device, we choose the Z-axis of the body coordinate as the heading direction. These figures show stable estimations even for a path as long as 50m. Phase1 outputs have much larger deviations from the actual path, while Phase2 estimation is quite accurate. Since Phase1depends on integration of accelerometer outputs, it requires high accuracy in the attitude estimation to eliminate gravitational force effect on acceleration calculation of the device. However, due to the corruption effect, attitude estimation accuracy deteriorates and becomes unreliable in Phase1. Phase2's estimates are acceptable, as it only depends on the heading direction and stride lengths, both of which can be provided accurately by DRA-II.

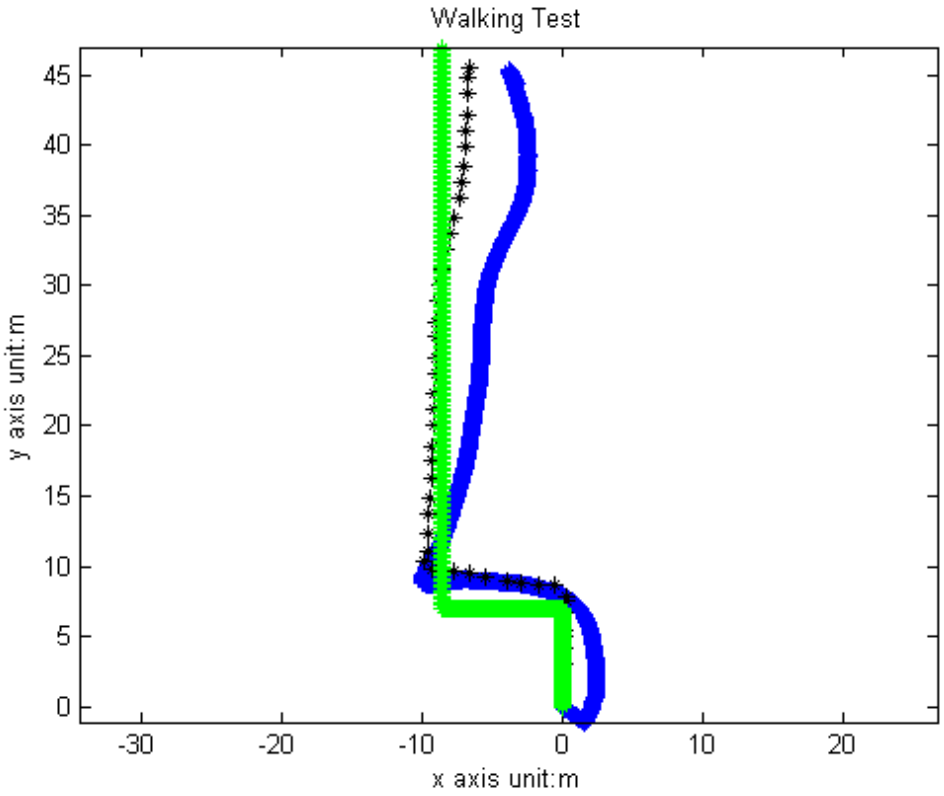

Figure 38: Walking Test of DRA-II

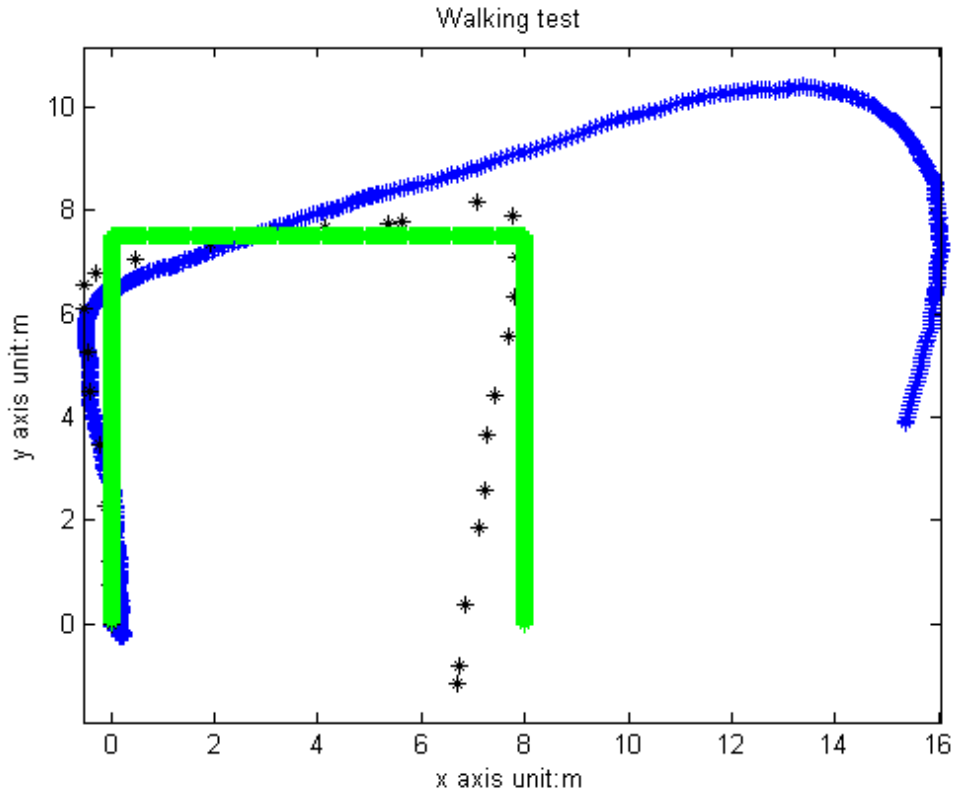

Figure 39: Walking Test for the DRA-II

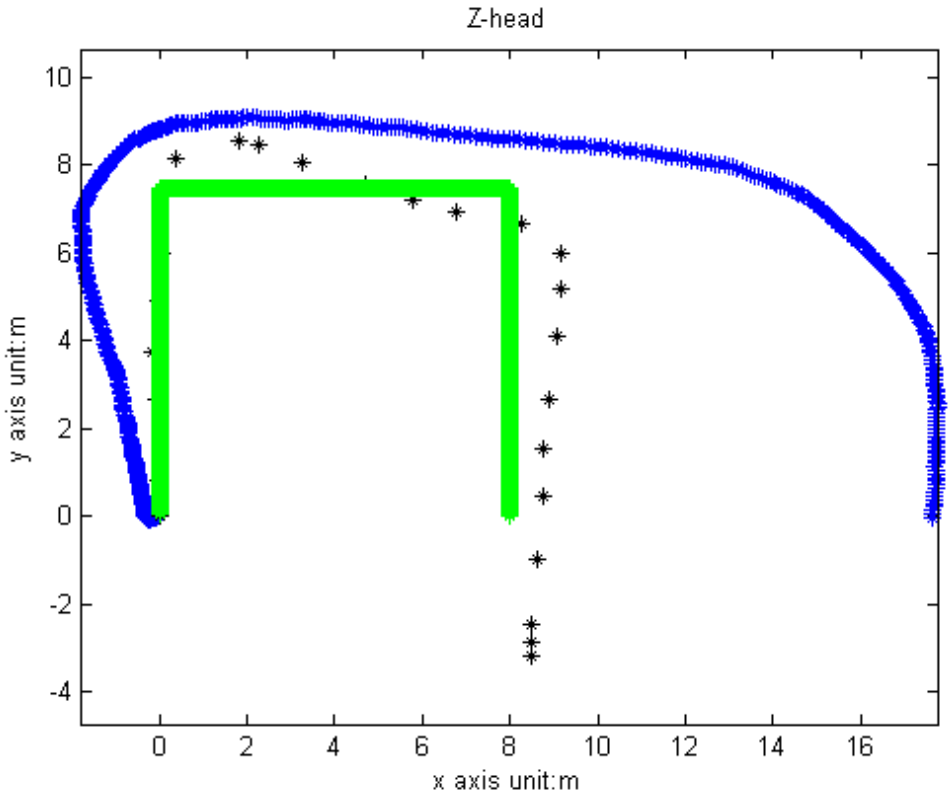

Figure 40: Walking Test for DRA-II, Z-axis as Heading Direction

4.4 Discussion

Attitude estimation algorithm has been explored in this chapter. In the static case, the device's attitude can be estimated in an accurate and stable manner. In the walking case, yaw angle estimation is acceptable, which has comparable performance with magnetometer-only heading estimation without exerting restrictions on how to hold the device. The proposed algorithm can also reject sharp magnetic field changes due to external interference. By now, integrating attitude estimation into the DRA-I has solved instability issues in indoor localization. More advanced algorithms have been proposed to achieve better accuracy in path estimation which will be covered in Chapter 5.

CHAPTER 5 PROPOSED DEAD RECKONING ALGORITHM - III

5.1 Magnetometer Calibration

In DRA-I and DRA-II, the sensors' error models are simplified as a combination of offsets and zero mean white Gaussian noises. From signal analysis in section 2.3, it is obvious that the errors' models are much more complicated and therefore, to improve the performance of our algorithm, it is important to adopt a more sophisticated calibration method. For the magnetometer calibration, we adopt the algorithm proposed in [32]. The algorithm is presented in details in **Appendix 7**. **Table 8** shows the parameters of the calibration result.

Table 8: Magnetometer Calibration Parameters

Parameters	X	Y	Z	γ_x	γ_y	γ_z
Values	0.4805	-0.2166	-0.7032	0.7362	0.7723	0.7403

Magnetometer's measurements are calibrated as

$$\begin{bmatrix} m_{cx} \\ m_{cy} \\ m_{cz} \end{bmatrix} = \begin{bmatrix} \frac{m_x - m_{x0}}{\gamma_{mx}} \\ \frac{m_y - m_{y0}}{\gamma_{my}} \\ \frac{m_z - m_{z0}}{\gamma_{mz}} \end{bmatrix} \quad \textbf{(54)}$$

The calibrated results lie on the surface of a sphere whose radius is the geomagnetic field strength according to IGRF.

In **Figure 41**, blue points are magnetometer measurements divided by an attenuation factor and red points are calibration results. The sphere is centered at origin with

radius R=42.1227, the geomagnetic field strength. It is clear that calibrated results are closer to the sphere than blue points and it validates this calibration algorithm.

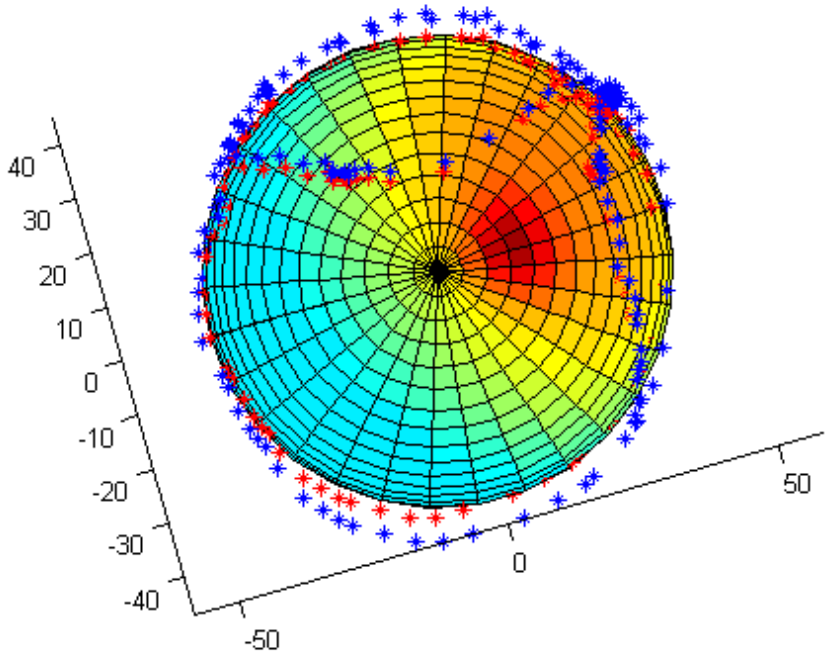

Figure 41: Magnetometer calibration result

5.2 Accelerometer Calibration

As shown in previous chapters, the accelerometer's measurement errors are much more complicated than the modeling in section 2.3 and it is a significant error source for attitude estimation. Accelerometers' measurement errors can be classified as measurement bias, scaling factors due to different sensitivity of each axis and wide band noises. They are very similar to magnetometer measurement errors; therefore, the calibration algorithm proposed in [32] can also be applied to accelerometer calibration and the details are shown in **Appendix 8**.

The estimated parameters for the device are listed in **Table 9**:

Table 9: Accelerometer Calibration Method

	g_{x0}	g_{y0}	g_{z0}	γ_x	γ_y	γ_z
Values	-0.3708	-0.2256	0.2424	1.0352	1.0891	1.0442

If the device is stationary, calibrated results shall lie on the surface of an ellipsoid with center at origin and radius as 9.81. In **Figure 42**, blue points are the accelerometer's measurements and red points are the calibrated results. The sphere is centered at origin with a radius as 9.81. Red points are closer to the sphere and thus the calibration algorithm is validated

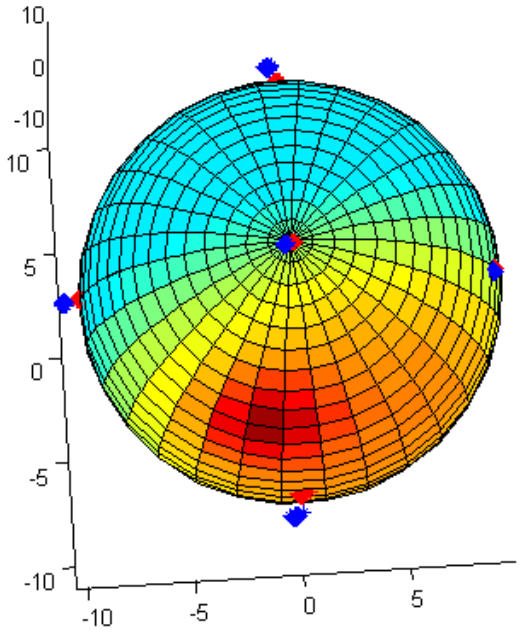

Figure 42: Accelerometer Calibration Result

5.3 DRA-III

DRA-III is presented in this section.

5.3.1 Algorithm Description

The calibration step and Phase1 in the tracking step is the same as the attitude estimation algorithm in section4.1. Phase2 of the tracking step in DRA-III is the same as the one in DRA-II, which estimates stride lengths of each second and given quaternion estimated by Phase 1 and reference quaternion given by the calibration step, the position of the device is computed.

Matlab code for DRA-III is attached as **Appendix** 9.

5.3.2 Experiment Results

Attitude estimation accuracy is examined and experiment results are shown in this section. The same data of static test and walking test in section 4.1.2 are processed by the calibration step of DRA-III to investigate if novel calibration algorithms improve attitude estimation. **Figures 43**, **44**, **45** shows respectively yaw, roll and pitch angles of the static case. Roll angle errors are within 3.5 degrees and pitch angle errors are within 1 degree. The results are better than those in section 4.1.2, as the accelerometer's measurement errors are better modeled and thus eliminated. Yaw angle errors have also been significantly reduced, from 9 degrees to 1 degree owing to the novel calibration algorithm.

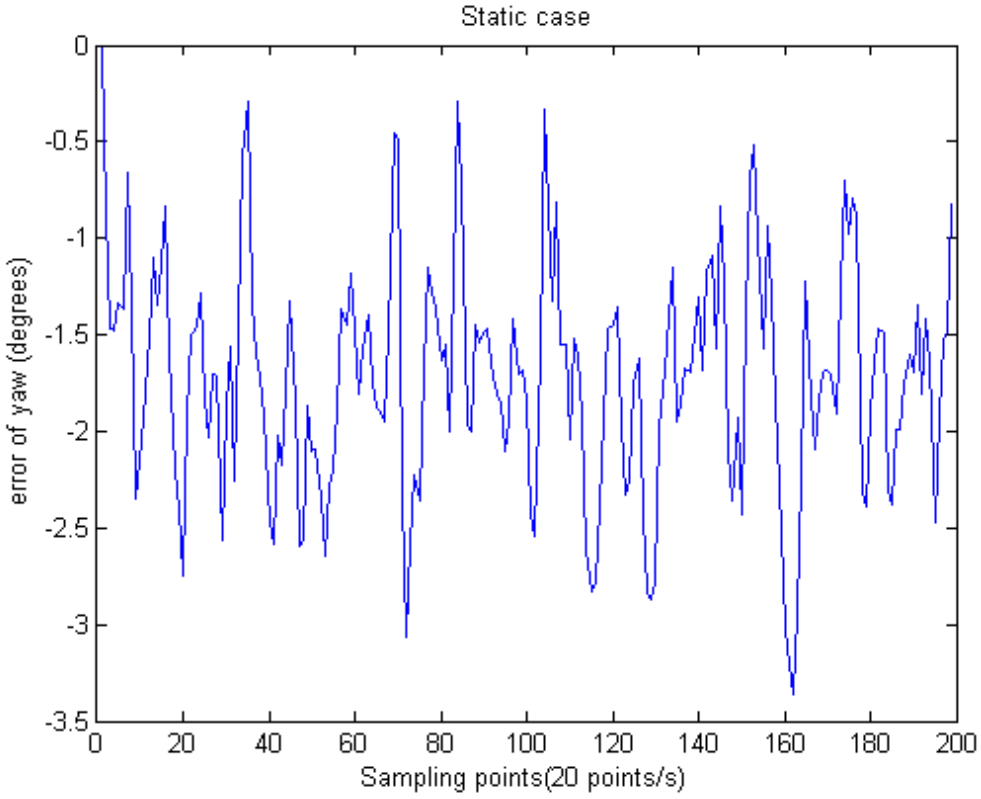

Figure 43: Yaw Angle Estimation Errors of Static Case

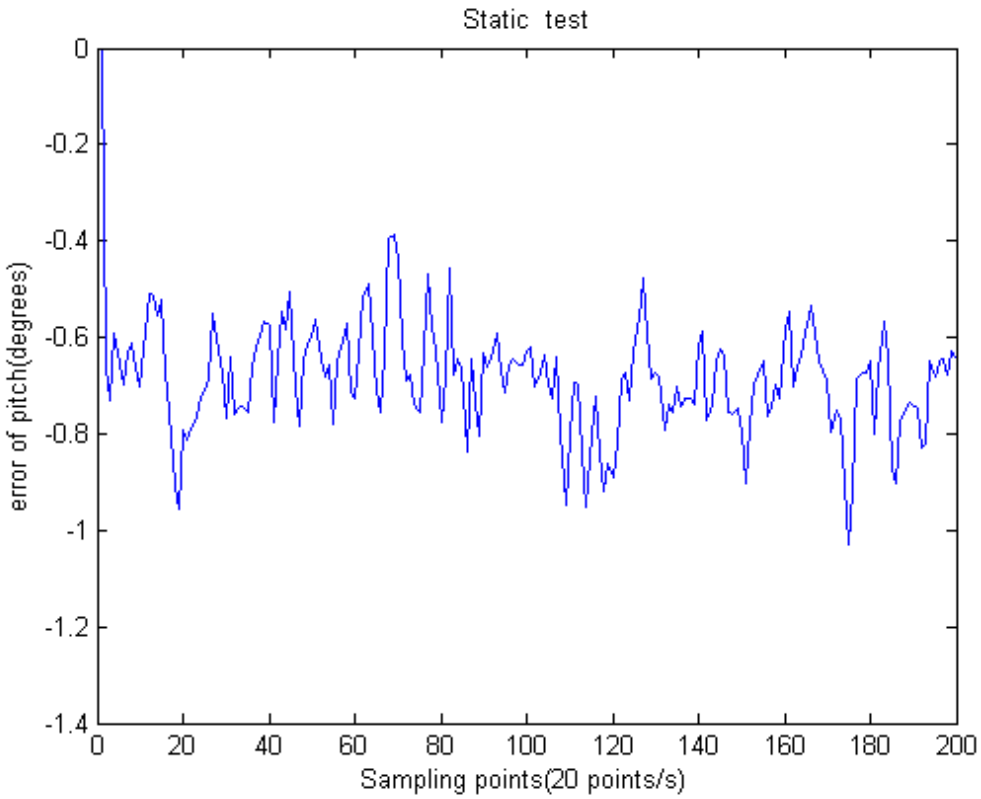

Figure 44: Pitch Angle Estimation Errors of Static Case

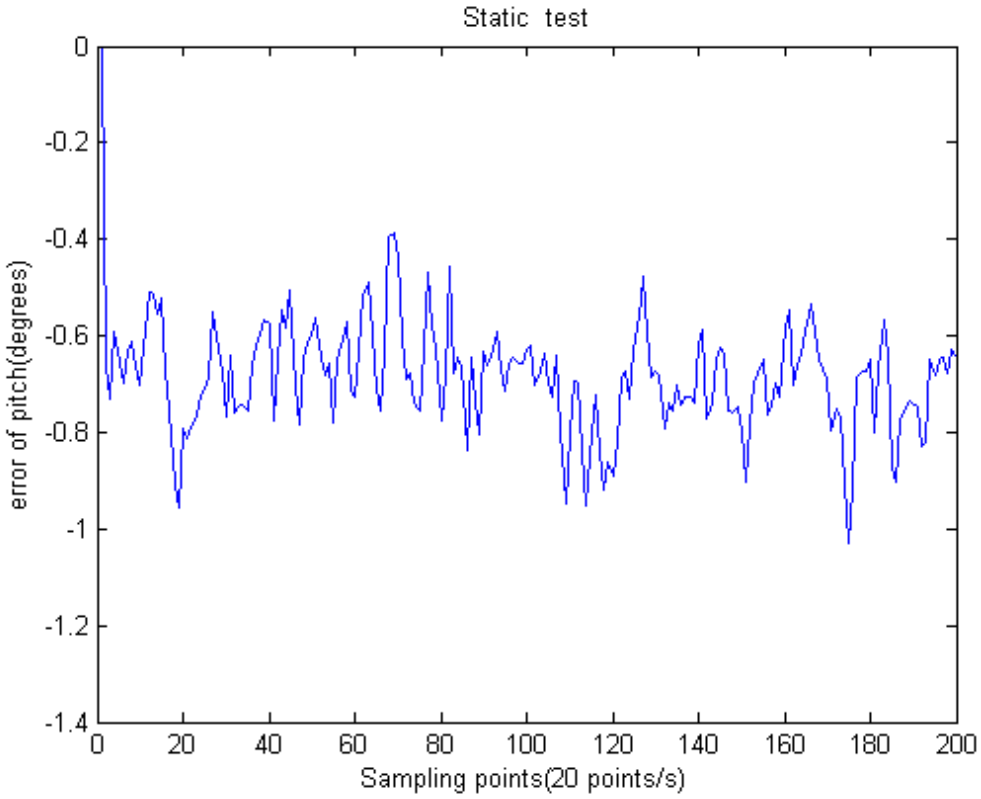

Figure 45: Roll Angle Estimation Errors of Static Case

Figures 46, **47** and **48** show roll, pitch and yaw angles estimation in the walking case. Roll and pitch angles are similar to the results shown in section 4.1.2. **Figure 48** shows estimations of the device's yaw angles. Comparing with **Figure 26**, the calibration step in DRA-III also rejects sharp changes of the magnetic field and its outputs are closer to the ground truth, owing to adaption of the novel calibration algorithm.

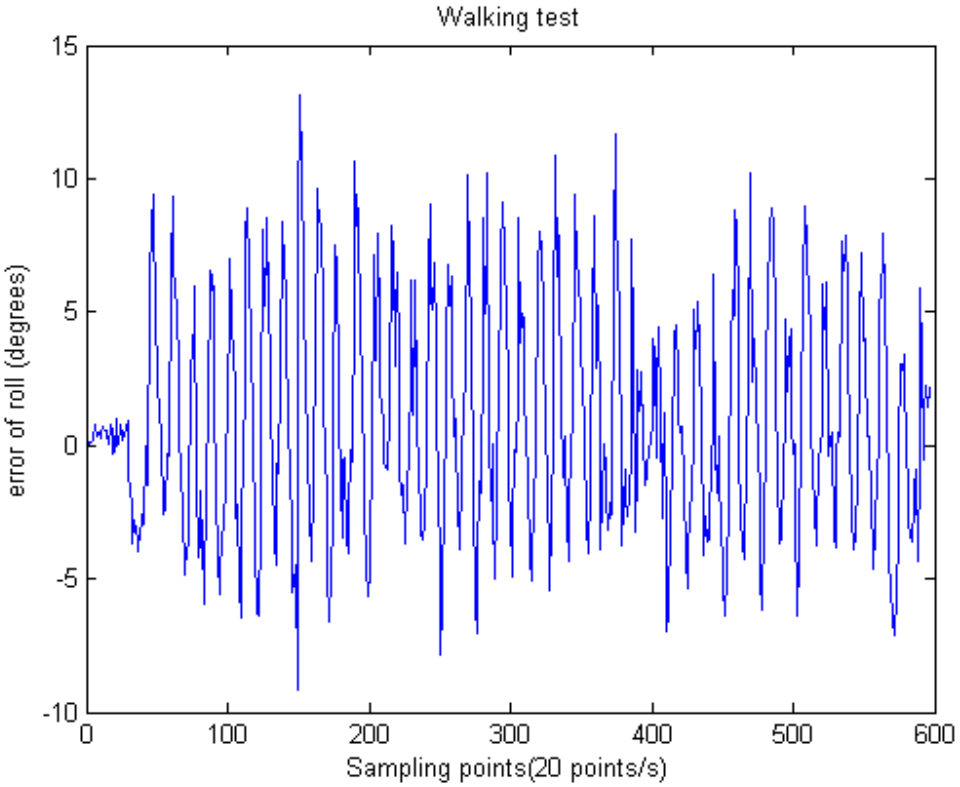

Figure 46: Roll Angle Estimation Errors of Walking Test

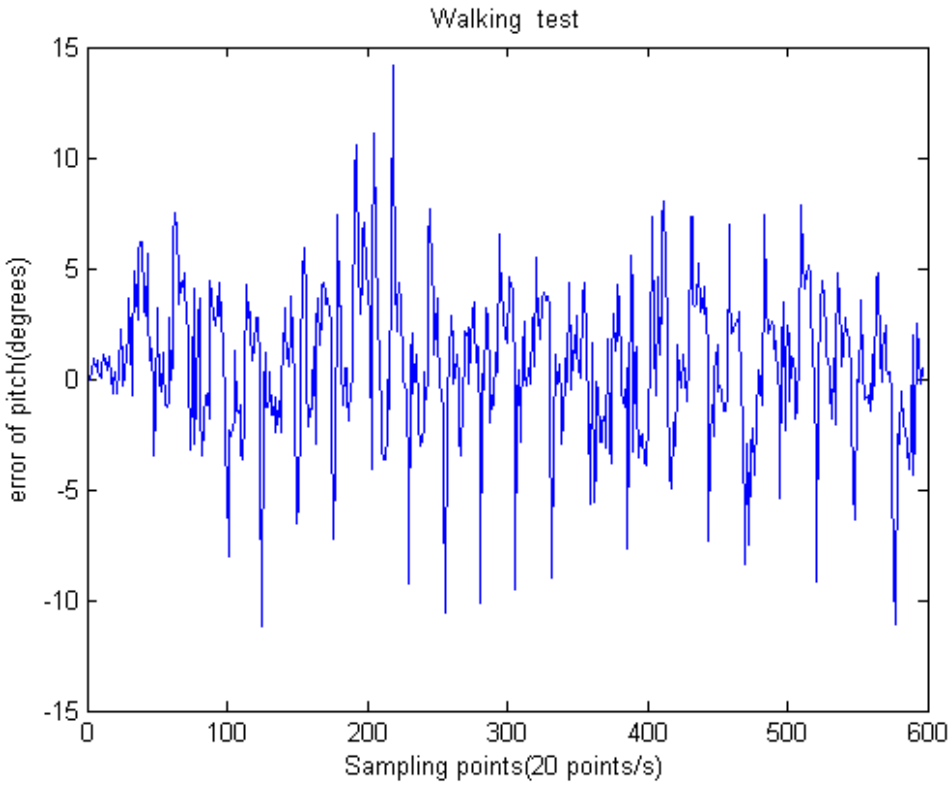

Figure 47: Pitch Angle Estimation Errors of Walking Test

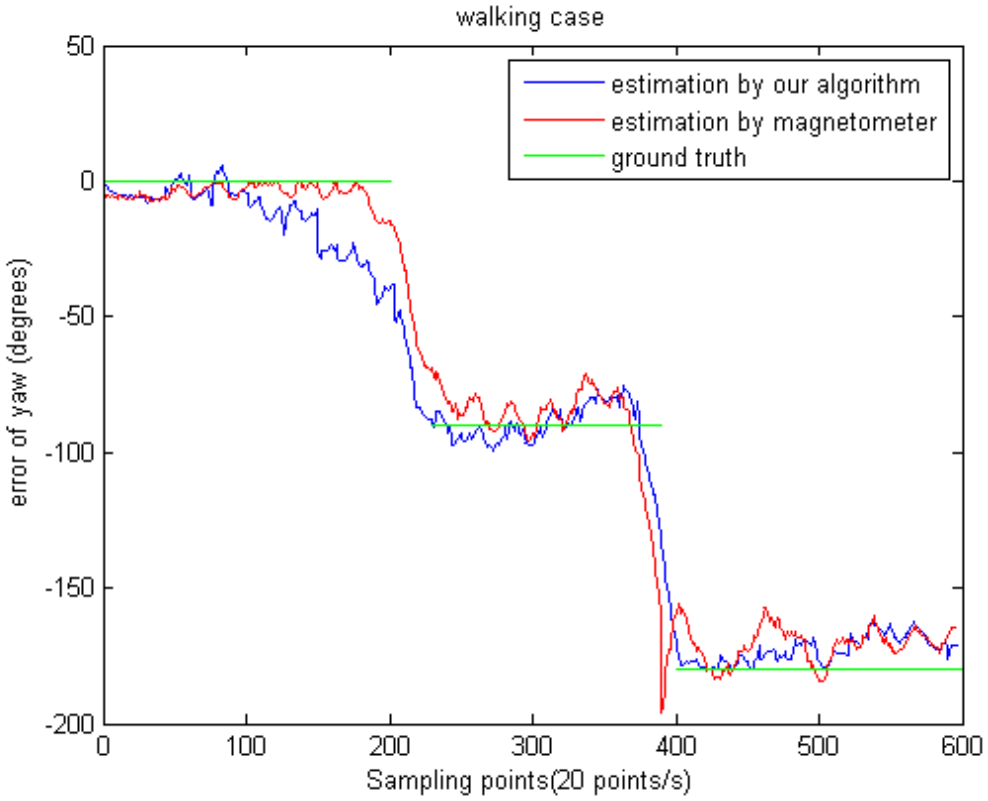

Figure 48: Yaw Angle Estimation Errors of Walking Test

To test the tracking step of DRA-III, the same data of U-turn path used in section 4.1.2 are processed here. Owing to improvement in attitude estimations, position estimation errors in both tests have been significantly reduced in comparison with **Figures 39** and **40**.

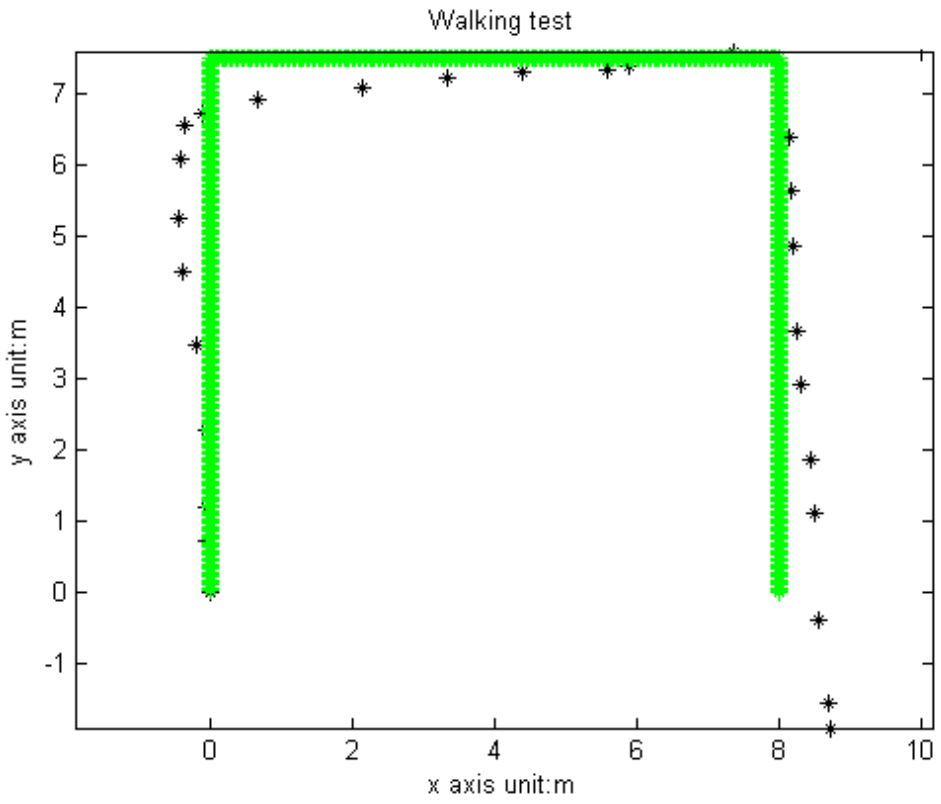

Figure 49: Position Estimation of DRA-III

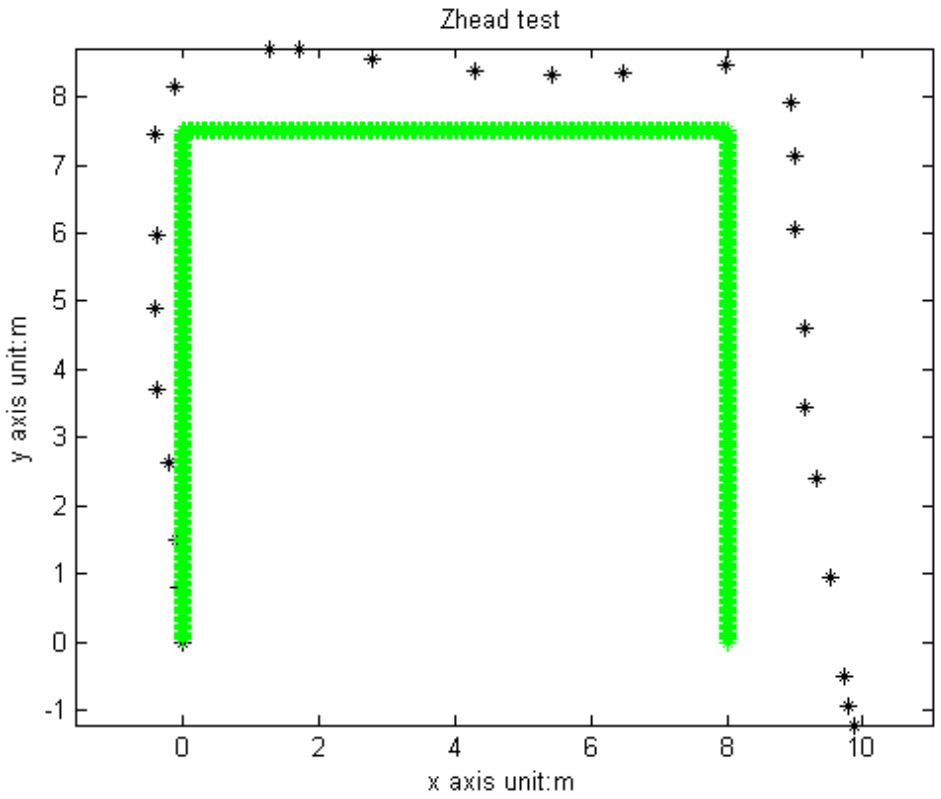

Figure 50: Position Estimation of DRA-III

5.4 Discussion

The proposed DRA-III includes two steps as DRA-II. Calibration step is the same as the attitude estimation algorithm proposed in section 4.1. Tracking step includes two phases. Phase1 is the same as its calibration step and Phase2 is the same as Phase2 of DRA-II's tracking step.

In response to oversimplified modeling of sensors' measurements which deteriorates attitude estimation, novel calibration algorithms are adapted in DRA-III. The algorithm for magnetometer proposed by [32] classifies its measurement errors as wide band noise, scaling factors, misalignment and magnetic interferences and uses a two-step calibration algorithm to estimate these parameters. The accelerometer measurement errors can be classified as bias, scaling factors and wide band noise. Owing to the novel proposed calibration algorithms, attitude estimation accuracies have been improved and position estimation henceforth.

CHAPTER 6 CONCLUSION AND FUTURE WORK

6.1 Remark on Aforementioned Three DRAs

In this project, three DRAs for indoor localization purpose have been proposed and investigated. DRA-I is based on integration of sensors' outputs with constraints on positions and velocities exerted by step counting algorithm. Given the initial condition, it can accurately estimate the pedestrian's position for the first 10-15 meters. It then becomes dysfunctional because DRA-I is lack of robust attitude estimation capability. An attitude estimation algorithm is proposed and validated. It gives precise estimation if the device is stationary. If the device moves with the pedestrian, its heading direction can be precisely estimated, but its roll angle and pitch angle estimation is erroneous due to the corruption effect. Erroneous roll and pitch angles estimation make it impossible to eliminate gravitational acceleration from accelerometer's measurements and this will cause tremendous errors in the integration of the sensors' output. Therefore, this algorithm is not a solution to DRA-I drawbacks, and integration of this algorithm with DRA-I will not give satisfactory position estimation, as discussed in section 4.2. In section 4.3, DRA-II is proposed and tested, which applies the proposed attitude estimation algorithm. DRA-II has two steps, the calibration step and the tracking step. The calibration step is used to calculate the device's attitude while it is stationary and assumes that the pedestrian's heading direction is the Y-axis of the global coordinate. DRA-II assumes after the calibration step, there is no more relative movement between the device and the pedestrian; in other words, the pedestrian's heading direction is solely determined by changes of the

device's attitude. The tracking step contains two phases as described in section 4.3. It tracks the pedestrian's position and experimental results show the pedestrian's position can be correctly tracked for a walk as long as 50 meters with errors within 4 meters.

DRA-III is a simplified version of DRA-II with adaption of novel calibration algorithms for the magnetometer [32] and accelerometer. In consequence, attitude estimation errors are significantly reduced for both the static case and the walking case. Hence, position estimation errors are also reduced to be within 1m for a 25-meters walk. Besides, DRA-III is much simplified with respect to DRA-I and DRA-II and needs less computational power to work in real time. It is feasible for a Samsung Tab whose specifications are listed in **Appendix 1**. As step counting algorithm applied in this project is also designed to use in real time, DRA-III is a version ready to be implemented to work in real time.

6.2 Recommendation for Future Work

As the main purpose of this project is to investigate dead reckoning algorithm for indoor localization purpose, initial conditions are assumed known. Besides, although position estimations for DRA-II and DRA-III are acceptable for a 50m walk, errors of these dead reckoning algorithms are still accumulating. Therefore, integrating the proposed dead reckoning algorithms with a correction phase is a possible extension of this project. The correction phase needs to provide position estimations. Then the

Kalman filter or particle filter can be applied to fuse the results from the proposed dead reckoning algorithms and the correction phase.

Several existing technologies are potential choices for the correction phase, such as Wi-Fi fingerprinting, Sparse tracking technologies [13], etc. Estimation errors of Wi-Fi fingerprinting are generally within 3-7 meters. As AP numbers are increasing rapidly in urban area, Wi-Fi fingerprinting is a promising choice as the correction phase. However, Wi-Fi fingerprinting offline training is labor and time consuming and its online tracking results in a heavy computational load. The Sparse track method has comparable performance to Wi-Fi fingerprinting and it does not need offline training. However, extra equipment is required to implement the sparse track which causes extra costs for large scale implementation. Another possible solution is using the Wi-Fi signal strength directly. In the Android platform, Wi-Fi signal strengths are retrieved at discrete levels and represented as integers from -99 to -1, and a bigger value means stronger signal strength. It has been found if the signal level is larger than -35, the device's position is within 2-4 meters from the corresponding AP. If AP's position is carefully chosen, the device's position can be thus estimated. If no Wi-Fi signal's level is more than -35, the dead reckoning algorithm can be applied. In urban areas rich with APs, it is definitely a feasible solution.

REFERENCE

[1] Rapid cloud development using App Engine for the Cycle Hire Widget Android application. Retrieved April 06, 2011, from http://googleappengine.blogspot.com/2010/08/rapid-cloud-development-using-app.html

[2] S. Steiniger, M. Neun, and A. Edwardes, " Foundations of Location Based Services" ,Retrieved April 06, 2011, from http://nchc.dl.sourceforge.net/project/jump-pilot/w_other_freegis_documents/articles/lbs_lecturenotes_steinigeretal2006.pdf

[3] L. Aalto, N. Gothlin, J. Korhonen, & T. Ojala. (2004) "Bluetooth and WAP push based location-aware mobile advertising system", *inProceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 49–58. ACM Press

[4] Badawy, O.M. Hasan, M.A.B. (2007). "Decision Tree Approach to Estimate User Location in WLAN Based on Location Fingerprinting", *inRadio Science Conference, 2007. NRSC 2007. National* , 1-10.

[5] Callaghan et al. (2010) "Correlation-based radio localization in an indoor environment". Retrieved April 06, 2011, from http://math.stanford.edu/~papanico/pubftp/corrbasedlocalization.pdf

[6] Ma et al. (2010). "Accuracy Enhancement for Fingerprint-based WLAN Indoor Probability Positioning Algorithm", *in Pervasive Computing Signal Processing and Applications (PCSPA), 2010 First International Conference on,* 167-170

[7] Ma et al. (2008). "Cluster Filtered KNN: A WLAN-Based Indoor Positioning Scheme", *inWorld of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a,* 1-8

[8] Xu et al. (2010), "Optimal KNN Positioning Algorithm via Theoretical Accuracy

Criterion in WLAN Indoor Environment", *in GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference*

[9] H. Leppakoski, S. Tikkinen, A. Perttula, J.Takala, "Comparison of indoor positioning algorithms using wlan fingerprints" Retrieved April 08, 2011, from http://www.enc-gnss09.it/proceedingsPDF/B4/3_B4.PDF

[10] H. Lim, L. Kung, J. Hou, H. Luo . (2006), "Zeroconfiguration, robust indoor localization theory and experimentation" , *in INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings,*1-12

[11] V. Otsason, A. Varshavsky, A. LaMarca, and E. de Lara. (2005), "Accurate GSM indoor localization," *UbiComp 2005,* Lecture Notes Computer Science,Springer-Varlag, vol. 3660, pp. 141–158

[12] R. Stirling, J. Collin, K. Fyfe, and G. Lachapelle (2003), "An innovative shoemounted pedestrian navigation system*," in Proceedings of European Navigation Conference GNSS 2003*

[13] Jin et al (2010), "SparseTrack: Enhancing Indoor Pedestrian Tracking with Sparse Infrastructure Support", *in INFOCOM, 2010 Proceedings IEEE*

[14] H. Wang, H. Lenz, A. Szabo, J. Bamberger, U.D. Hanebeck. (2007), "WLAN-BasedPedestrian Tracking Using Particle Filters and Low-Cost MEMS Sensors", *in Positioning, Navigation and Communication, 2007. WPNC '07. 4th workshop*

[15] Eric Foxlin. (2005), "Pedestrian Tracking with Shoe-Mounted Inertial Sensors," *IEEE Computer Graphics and Applications*, vol. 25, no. 6, pp. 38-46

[16] K. Frank, B. Krach, N. Catterall, P Robertson. (2009) "Development and Evaluation of a Combined WLAN & Inertial Indoor Pedestrian Positioning System", Retrieved April 08, 2011, from http://futurecomm.tssg.org/public/publications/2009_IONGNSS_frank_et_al.pdf

[17] Android (operating system) Retrieved April 08, 2011, from http://en.wikipedia.org/wiki/Android_(operating_system)

[18] F. Ableson. (2009) "Tapping into Android's sensors", Retrieved April 08, 2011, from http://www.ibm.com/developerworks/opensource/library/os-android-sensor/index.html

[19] "WifiManager", Retrieved April 08, 2011, from http://developer.android.com/reference/android/net/wifi/WifiManager.html

[20] Wikipedia , "Google App Engine", Retrieved April 08, 2011, from http://en.wikipedia.org/wiki/Google_App_Engine

[21] Wikipedia, "Kalman filter" ， Retrieved April 08, 2011, from http://en.wikipedia.org/wiki/Kalman_filter

[22] D. Tiltterton& J. Weston. (2005). *Strapdown Inertial Navigation Technology (2nd ed.)*. AIAA.

[23] D.Alazard, (2005) Introduction au filtre de Kalman

[24] A. Kim & M.F. Golnaraghi. (2004), "A quaternion-based orientation estimation Algorithm Using an inertial Measurement Unit", *in Position Location and Navigation Symposium, 2004. PLANS 2004*

[25] E.Kraft. (2003), "A quaternion-based unscented Kalman Filter for orientation Tracking", *in Proceedings of the Sixth International Conference of Information Fusion - FUSION 2003*

[26] G.A.Natanson, M.S.Challa, J.Deutschmann, D.F.Baker.(1994), "Magnetometer-only attitude and rate determination for a Gyro-less spacecraft", *in its Third International Symposium on Space Mission Operations and Ground Data Systems, Part 2 p 791-798 (SEE N95-17531 04-17)*

[27] I.Y.Bar-Itzhack, Y.Oshman.(1985), "Attitude determination from Vector observations: Quaternion estimation", *in IEEE Transactions on Aerospace and Electronic Systems Jan. 1985*

[28] M. Shuster and S. Oh.(1981), "Three-axis attitude determination from vector observations", *in J. Guid. Control, vol. 4, no. 1, pp. 70–77, Jan./Feb. 1981.*

[29] F.L.Markley. (1988), "Attitude determination using vector observations and the singular value decomposition", *in Journal of the Astronautical Sciences, 1988*

[30] A simplified quaternion-based algorithm for orientation estimation from earth gravity and magnetic field measurements

[31] L. Fang, P. J. Antsaklis, L. A. Montestruque et al (2005), "Design of a wireless assisted pedestrian dead reckoning system –The NavMote experience," *in IEEE Trans. Instrumentation and Measurement,vol. 54, no. 6, pp. 2342-2358, Dec. 2005.*

[31] H.-J. Jang, J. W. Kim, D.-H. Hwang. (2007), "Robust step detection method for pedestrian navigation systems", *in IEEE Electronics Letters, vol. 43, iss. 14, Jul. 2007.*

[32] D. Gebre-Egziabher, G. H. Elkaim, J. D. Powell, and B. W. Parkinson.(2006), "Calibration of strapdown magnetometers in magnetic field domain", *in ASCE Journal of Aerospace Engineering, vol. 19, no. 2, pp. 1–16, April 2006*

[33] National Geographical Data Center, "Geomagnetism", Retrieved November 08, 2011, from http://www.ngdc.noaa.gov/geomagmodels/IGRFWMM.jsp

[34] Wikipedia, "Earth's Magnetic Field", Retrieved December 08, 2011, from http://en.wikipedia.org/wiki/Earth's_magnetic_field

[35] Wikipedia, "Magnetic Dip"， Retrieved December 08, 2011, from http://en.wikipedia.org/wiki/Magnetic_dip

[36] Educated Runner, "The Fundamentals of Usain's Insane 100-Meter Bolt", Retrieved September 11, 2011, from http://www.educatedrunner.com/Blog/tabid/633/articleType/ArticleView/articleId/157/The-Fundamentals-of-Usains-Insane-100-Meter-Bolt.aspx

APPENDIX

APPENDIX 1 SAMSUNG TAB SPECIFICATION

GENERAL	**2G Network**	**GSM 850 / 900 / 1800 / 1900**
	3G Network	HSDPA 900 / 1900 / 2100
	Announced	2010, September
	Status	Available. Released 2010, October
SIZE	Dimensions	190.1 x 120.5 x 12 mm
	Weight	380 g
DISPLAY	Type	TFT capacitive touchscreen, 16M colors
	Size	600 x 1024 pixels, 7.0 inches
		- Gorilla Glass display
		- TouchWiz UI
		- Multi-touch input method
		- Accelerometer sensor for UI auto-rotate
		- Three-axis gyro sensor
		- Touch-sensitive controls
		- Proximity sensor for auto turn-off
		- Swype text input
SOUND	Alert types	Vibration; MP3, WAV ringtones
	Loudspeaker	Yes, with stereo speakers
	3.5mm jack	Yes, check quality
MEMORY	Phonebook	Practically unlimited entries and fields, Photocall
	Call records	Practically unlimited
	Internal	16/32 GB storage, 512 MB RAM
	Card slot	microSD, up to 32GB, buy memory
DATA	GPRS	Yes
	EDGE	Yes
	3G	HSDPA, 7.2 Mbps; HSUPA, 5.76 Mbps
	WLAN	Wi-Fi 802.11 b/g/n, Wi-Fi hotspot
	Bluetooth	Yes, v3.0 with A2DP
	Infrared port	No
	USB	Yes, v2.0 (proprietary)
CAMERA	Primary	3.15 MP, 2048x1536 pixels, autofocus, LED flash,check quality
	Features	Geo-tagging
	Video	Yes, 720x480@30fps
	Secondary	Yes, 1.3 MP
FEATURES	OS	Android OS, v2.2 (Froyo)

	CPU	ARM Cortex A8 processor, 1 GHz processor; PowerVR SGX540 graphics
	Messaging	SMS(threaded view), MMS, Email, Push Mail, IM, RSS
	Browser	HTML
	Radio	No
	Games	Yes
	Colors	Black and Grey
	GPS	Yes, with A-GPS support
	Java	Yes, MIDP 2.1
		- Social networking integration
		- Digital compass
		- Full HD video playback
		- Up to 7h movie playback
		- TV-out
		- MP4/DivX/WMV/H.264/H.263 player
		- MP3/WAV/eAAC+/AC3/FLAC player
		- Organizer
		- Image/video editor
		- Thinkfree Office (Word, Excel, PowerPoint, PDF)
		- Google Search, Maps, Gmail,
		YouTube, Calendar, Google Talk, Picasa integration
		- Readers/Media/Music Hub
		- Adobe Flash 10.1 support
		- Voice memo/dial/commands
		- Predictive text input
BATTERY		Standard battery, Li-Po 4000 mAh
	Stand-by	
	Talk time	Up to 28 h (2G) / Up to 25 h 30 min (3G)
MISC	SAR US	1.00 W/kg (head)
	SAR EU	1.07 W/kg (head)

APPENDIX 2: MATLAB CODE FOR STEP COUNTING ALGORITHM

```
%bad points counting and vector norm drawing

clc
clear
close all

A=dlmread('50Step\gyroData.txt');
l=length(A)
Fs=20;
t=1/Fs;
axOffset=-0.1635;
ayOffset=1.08589;
azOffset=0.03;

noStep=0;
flagUp=0;
D=sqrt((A(1,3)-axOffset)^2+(A(1,4)-ayOffset)^2+(A(1,5)-azOffset)^2);
stepLength=[0,0,0];
meanA=[A(1,3),A(1,4),A(1,5)];
stepPt=[0,0];
ind=0;
for i=1:l

  accNorm=sqrt((A(i,3)-axOffset)^2+(A(i,4)-ayOffset)^2+(A(i,5)-azOffset)^2);
  if(flagUp==1)
    flag=(accNorm-D)>3.5;
  else
    flag=(D-accNorm)>3.5;
  end
  if(flag==1)
    if(D>accNorm)
      flagUp=1;
    else flagUp=0;
    end
    stepPt=[stepPt;ind,D];
    D=accNorm;
    ind=i;
    if noStep==0
      noStep=noStep+1;
    end
```

```
      noStep=noStep+1;
      stepLength=[stepLength;meanA*t];
      meanA=[0,0,0];

    else
      if(((flagUp==1)&D>accNorm)|((flagUp==0)&D<accNorm))
        D=accNorm;
        ind=i;
      end
      meanA=meanA+[A(i,3),(A(i,4)-1.01),A(i,5)];

    end
end

noOfStep=(noStep+1)/2
totalLength=sum(stepLength(1:noStep,2))
diff=noStep-length(stepLength)


%radio=(bNo+cNo)/l
E=sqrt((A(1,3)-axOffset)^2+(A(1,4)-ayOffset)^2+(A(1,5)-azOffset)^2);
for i=2:l

   E=[E;sqrt((A(i,3)-axOffset)^2+(A(i,4)-ayOffset)^2+(A(i,5)-azOffset)^2)];
end


meanAx=mean(A(1:l,3))
meanAy=mean(A(1:l,4))
meanAz=mean(A(1:l,5))
meanGx=mean(A(1:l,6))
meanGy=mean(A(1:l,7))
meanGz=mean(A(1:l,8))
meanA=mean(D)
meanG=mean(E)

index3=find(abs(E)>3);
totalBadGyroPoints=length(index3)
radio=totalBadGyroPoints/l%subplot(2,2,1)
%plot(A(1:l,6),'r')
%title('Gyroscope X axis');
%subplot(2,2,2)
%plot(A(1:l,7),'g')
%title('Gyroscope Y axis');
```

```
%subplot(2,2,3)
%plot(D)
%axis tight
%title('Acclerometer norm');
%figure(3)
plot(E)
xlabel('sampling points(20 points/s)');
ylabel('acceleration magnitude(m/s^2)');
title('Step Counting for 50 steps, walking');
hold on;
lt=length(stepPt);
plot(stepPt(1:lt,1),stepPt(1:lt,2),'r*')
hold on;
```

APPENDIX 3: MATLAB CODE FOR DRA-I

```
%implementation of the first phase. First using the byZ Y head example
clc
clear
close all

syms q0 q1 q2 q3 x y z ax ay az u v w wx wy wz f0 f1 f2 b0 b1 b2;
g=9.81;
tau=1;
taup=1;
bvn=[-0.2314  0.9664 0.1116]';
bx=-0.2314;
by=0.9664;
bz=0.1116;



xdot=u;
ydot=v;
zdot=w;

udot=(q0^2+q1^2-q2^2-q3^2)*ax+(2*q1*q2-2*q0*q3)*ay+(2*q1*q3+2*q0*q2)*az;
vdot=(2*q1*q2+2*q0*q3)*ax+(q0^2-q1^2+q2^2-q3^2)*ay+(2*q2*q3-2*q0*q1)*az;
wdot=(2*q1*q3-2*q0*q2)*ax+(2*q2*q3+2*q0*q1)*ay+(q0^2-q1^2-q2^2+q3^2)*az;

axdot=(-1/tau)*ax;
aydot=(-1/tau)*ay;
azdot=(-1/tau)*az;
%axdot=(-1/tau)*ax+(2*q1*q3-2*q0*q2)*g;
%aydot=(-1/tau)*ay+(2*q2*q3+2*q0*q1)*g;
%azdot=(-1/tau)*az+(q0^2-q1^2-q2^2+q3^2)*g;

Qcrt=1-sqrt(q0^2+q1^2+q2^2+q3^2);%quaternion correction

q0dot=(-1/2)*q1*wx-(1/2)*q2*wy-(1/2)*q3*wz%+Qcrt*q0;
q1dot=(1/2)*q0*wx-(1/2)*q3*wy+(1/2)*q2*wz%+Qcrt*q1;
q2dot=(1/2)*q3*wx+(1/2)*q0*wy-(1/2)*q1*wz%+Qcrt*q2;
q3dot=(-1/2)*q2*wx+(1/2)*q1*wy+(1/2)*q0*wz%+Qcrt*q3;

wxdot=(-1/taup)*wx;
wydot=(-1/taup)*wy;
wzdot=(-1/taup)*wz;

% measurement
```

```
f0=(2*q1*q3-2*q0*q2);
f1=(2*q0*q1+2*q2*q3);
f2=(q0^2-q2^2-q1^2+q3^2);

b0=(q0^2+q1^2-q2^2-q3^2)*bx+(2*q1*q2+2*q0*q3)*by+(2*q1*q3-2*q0*q2)*bz;
b1=(2*q1*q2-2*q0*q3)*bx+(q0^2-q1^2+q2^2-q3^2)*by+(2*q2*q3+2*q0*q1)*bz;
b2=(2*q1*q3+2*q0*q2)*bx+(2*q2*q3-2*q0*q1)*by+(q0^2-q1^2-q2^2+q3^2)*bz;

JdVA=jacobian([udot;vdot;wdot],[ax;ay;az]);
%JdVQ=jacobian([udot;vdot;wdot],[q0;q1;q2;q3]);
%JdAQ=jacobian([axdot;aydot;azdot],[q0;q1;q2;q3]);
JdQQ=jacobian([q0dot;q1dot;q2dot;q3dot],[q0;q1;q2;q3]);
JdfQ=jacobian([f0;f1;f2],[q0;q1;q2;q3]);
JdbQ=jacobian([b0;b1;b2],[q0;q1;q2;q3]);

%JdQomega=jacobian([q0dot;q1dot;q2dot;q3dot],[wx;wy;wz]);% This matrix shall not
be used

%offset matrix
%ofs=[-0.255 0.120 0.759 -0.00024 0.0002 0.00002 ];
ofs=[-0.255 0.7025 0.759 -0.00024 0.0002 0.00002 -3.8 -6 3];
% phase 1 parameters
X=[x;y;z;u;v;w;ax;ay;az;q0;q1;q2;q3;wx;wy;wz];
%Observation Matrix
C=[1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0];

%Model noises
```

```
W=[0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004];

%observation Noises phase 1
V=[ 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.028 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.025 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.033 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.007 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.005 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.005 0.000 0.000
0.000 0.000 0.000 0.000 0.000
```

```
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.010 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.010
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.010 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.500 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.500 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.500];

%Initial Prediction
Pkplus=[0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05];

%parameters of phase 2
%
A2=[0 0 0 1 0 0
   0 0 0 0 1 0
   0 0 0 0 0 1
   0 0 0 -1 0 0
   0 0 0 0 -1 0
   0 0 0 0 0 -1];
%model noise
W2=[0.004 0.000 0.000 0.000 0.000 0.000
   0.000 0.004 0.000 0.000 0.000 0.000
   0.000 0.000 0.004 0.000 0.000 0.000
   0.000 0.000 0.000 0.004 0.000 0.000
```

```
   0.000 0.000 0.000 0.000 0.004 0.000
   0.000 0.000 0.000 0.000 0.000 0.004];

% Observation Noises phase 2
V2=[1.00 0.0 0.0 0.0 0.0 0.0
  0.0 1.00 0.0 0.0 0.0 0.0
  0.0 0.0 1.00 0.0 0.0 0.0
  0.0 0.0 0.0 0.01 0.0 0.0
  0.0 0.0 0.0 0.0 0.01 0.0
  0.0 0.0 0.0 0.0 0.0 0.01];
%observation matrix
C2=[1 0 0 0 0 0
   0 1 0 0 0 0
   0 0 1 0 0 0
   0 0 0 1 0 0
   0 0 0 0 1 0
   0 0 0 0 0 1];
Pkplus2=[0.05 0.0 0.0 0.0 0.0 0.0
      0.00 0.05 0.0 0.0 0.0 0.0
      0.00 0.0 0.05 0.0 0.0 0.0
      0.00 0.0 0.0 0.05 0.0 0.0
      0.00 0.0 0.0 0.0 0.05 0.0
      0.00 0.0 0.0 0.0 0.0 0.05 ];
%phase2 matrix
A2=zeros(6);

A2(1:3,4:6)=eye(3);
A2(4:6,4:6)=-eye(3);


%phase2 parameters
oldPosition=[0 0 0];
newPosition=[0 0 0];
oldVelo=[0 0 0];
newVelo=[0 0 0];
um=0;
vm=0;
wm=0;
xm=0;
ym=0;
zm=0;
dspVec=[0 0 0];%displacement vector
norDspVec=[sqrt(3)/3 sqrt(3)/3 sqrt(3)/3];%normalised displacement vector
data=dlmread('stats3/test6/gyroData.txt');
```

```
deg2rad=pi/180;
%Initial condition

   x=0;
   y=0;
   z=0;
   u=0;
   v=0;
   w=0;
   ax=0;
   ay=0;
   az=0;
   q0=1;
   q1=0;
   q2=0;
   q3=0;
   wx=data(1,6)*deg2rad;
   wy=data(1,7)*deg2rad;
   wz=data(1,8)*deg2rad;

Xe=eval(X);
Xe2=[Xe(1:6)];
Fs=20;
dt=1/Fs;
Fs2=1;
dt2=1/Fs2;
stride=0;
l=length(data)
%position=[0 0 0];
stateMemory=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
G=[0 0 0 0 0 0 0 0 g 0 0 0 0 0 0 0]';
%quaternion=[0,0,0,0]

%step counting parameters
accRef=sqrt(data(1,1)^2+data(1,2)^2+data(1,3)^2);
flagUp=0;
stepPt=[0,0];
stepNo=0;%will not be used
accPeak1=[1,accRef];%[index, acc]
accPeak2=[1,accRef];%[index, acc]
walkingLength=0;
stridesMem=0;

%
```

```
for i=0:l/20
   %phase 1
   for j=1:20
      index=20*i+j;
      if(index>l)
         break;
      end;
      A=zeros(16);
      A(1:3,4:6)=eye(3);
      A(4:6,7:9)=eval(JdVA);
      A(7:9,7:9)=-eye(3);
      A(10:13,10:13)=eval(JdQQ);
      A(14:16,14:16)=-eye(3);

      %Prediction
      Xkplusunmoins=Xe+A*Xe*dt;
      Ad=eye(16)+A*dt;
      Wd=W*dt+(A*W+(A*W)')*(dt^2)/2+A*W*A*(dt^3)/3;
      Pkplusunmoins=Ad*Pkplus*Ad'+Wd;

      % to compensate the gravity

      gravityM=[2*(q1*q3-q0*q2)  2*q2*q3+2*q0*q1  q0^2-q1^2-q2^2+q3^2];

      acc=[data(index,1:3)-ofs(1:3)]-gravityM*g

      amp=sqrt((data(index,1)-ofs(1))^2+(data(index,2)-ofs(2))^2+(data(index,3)-
ofs(3))^2);
      mfAmp=sqrt((data(index,11)-ofs(7))^2+(data(index,12)-
ofs(8))^2+(data(index,13)-ofs(9))^2);
      Ym=([xm,ym,zm,um,vm,wm,acc,data(index,6:8)*deg2rad,(data(index,1:3)-
ofs(1:3))/amp,(data(index,11)-ofs(7))/mfAmp,(data(index,12)-
ofs(8))/mfAmp,(data(index,13)-ofs(9))/mfAmp]-[0 0 0 0 0 0 0 0 0 ofs(4) ofs(5) ofs(6) 0
0 0 0 0 0])';
    %explanation take g=amplitude of measurement
         %counting step

      accNorm=sqrt(data(index,1)^2+data(index,2)^2+data(index,3)^2);
      if(flagUp==0)
         flag=(accNorm-accRef)>3.5;

      else
         flag=(accRef-accNorm)>3.5;
```

```
    end
    if(flag==1)       %means one step is found
      if(accRef>accNorm)
        flagUp=0;
      else flagUp=1;
      end
      accPeak1=accPeak2;
      accPeak2=[index,accRef];
     %
meanAcc=[mean(data(accPeak1(1,1):accPeak2(1,1),1)),mean(data(accPeak1(1,1):accPe
ak2(1,1),2)),mean(data(accPeak1(1,1):accPeak2(1,1),3))];
     stride=[stride;sqrt(sqrt(abs(accPeak1(1,1)-
accPeak2(1,1))))/4.1277];%1/2*((accPeak2(1,1)-
accPeak1(1,1))*dt)^2*abs(accPeak2(1,2)-accPeak1(1,2))];
      %stride=[stride;1/2*((accPeak2(1,1)-
accPeak1(1,1))*dt)^2*abs(sqrt(meanAcc(1,1)^2+meanAcc(1,2)^2+meanAcc(1,3)^2))];

      stepPt=[stepPt;index,accRef];
    else
      if(((flagUp==1)&accRef<accNorm)|((flagUp==0)&accRef>accNorm))
        if(index-accPeak1(1,1)>20)
          accPeak1=accPeak2;
          accPeak2=[index,accRef];
        end

        accRef=accNorm;

      end

    end

    %Ym=[xm,ym,zm,um,vm,wm,acc,(data(index,6:8)-ofs(4:6))*deg2rad]';

    %Update phase
     q0=Xkplusunmoins(10);
     q1=Xkplusunmoins(11);
     q2=Xkplusunmoins(12);
     q3=Xkplusunmoins(13);


    %update observation matrix
    C(13:15,10:13)=eval(JdfQ);
    C(16:18,10:13)=eval(JdbQ);
    K=(Pkplusunmoins*C')/(C*Pkplusunmoins*C'+V);
```

```
    tmp=C*Xkplusunmoins;
    yk=[tmp(1:12,1)', eval(f0),eval(f1),eval(f2),eval(b0),eval(b1),eval(b2)]';
    Xkplusunplus=Xkplusunmoins+K*(Ym-yk);
    Xe=Xkplusunplus;

    Pkplusunplus=(eye(size(K*C,1))-K*C)*Pkplusunmoins*(eye(size(K*C,1))-
K*C)'+K*V*K';
    Pkplus=Pkplusunplus;
     x=Xe(1,1);
     y=Xe(2,1);
     z=Xe(3,1);
     u=Xe(4,1);
     v=Xe(5,1);
     w=Xe(6,1);
     ax=Xe(7,1);
     ay=Xe(8,1);
     az=Xe(9,1);
     q0=Xe(10,1);
     q1=Xe(11,1);
     q2=Xe(12,1);
     q3=Xe(13,1);
     wx=Xe(14,1);
     wy=Xe(15,1);
     wz=Xe(16,1);
     i
     stateMemory=[stateMemory;Xe(1:16,1)',data(index,15) ];

  end

  %phase 2

  %Prediction
  %Xkplusunmoins2=Xe2+A2*Xe2*dt2;
 % Ad2=eye(6)+A2*dt2;
  %Wd2=W2*dt2+(A2*W2+(A2*W2)')*(dt2^2)/2+A2*W2*A2*(dt2^3)/3;
 % Pkplusunmoins2=Ad2*Pkplus2*Ad2'+Wd2;
  %qNormVec=[Xe(10,1),Xe(11,1),Xe(12,1),Xe(13,1)];
  % qNorm=norm(qNormVec);
  % Xe(10,1)=Xe(10,1)/qNorm;
  % Xe(11,1)=Xe(11,1)/qNorm;
  % Xe(12,1)=Xe(12,1)/qNorm;
  % Xe(13,1)=Xe(13,1)/qNorm;
  % q0=Xe(10,1);
```

```
    % q1=Xe(11,1);
    % q2=Xe(12,1);
    % q3=Xe(13,1);
    %mesurement from phase 1

    %newVelo=[u v w];

    %dspVelo=newVelo-oldVelo[];

    newPosition=[u,v,w];
    dspVec=newPosition-oldPosition
    norDspVec=(dspVec)/sqrt(dspVec(1,1)^2+dspVec(1,2)^2+dspVec(1,3)^2);
    strides=sum(stride);
    walkingLength=walkingLength+sum(stride);
    um=norDspVec(1,1)*strides;
    vm=norDspVec(1,2)*strides;
    wm=0;%norDspVec(1,3)*strides;
    xm=xm+um;
    ym=ym+vm;
    zm=zm+wm;
   % ampVm=sqrt(umm^2+vmm^2+wmm^2);
   % ampV=sqrt(u^2+v^2+w^2);
   % um=u/ampV*ampVm;
   % vm=v/ampV*ampVm;
   % wm=w/ampV*ampVm;
   stridesMem=stridesMem+strides;
    oldPosition=newPosition;
    stride=0;
end
dlmwrite('stateVec.txt',stateMemory);
```

APPENDIX 4: MATLAB CODE FOR ATTITUDE ALGORITHM

```
clc
clear
close all

syms q0 q1 q2 q3 wx wy wz d0 d1 d2 f0 f1 f2 b0 b1 b2 q0t q1t q2t q3t f0t f1t f2t b0t b1t
b2t
wtau=1;
dtau=1;
g=9.8;
bv=[   -6.8626   28.6608 3.3083]';%statistics of magnetic field
bvn=[-0.2314  0.9664 0.1116]';%normalised magnetic field
bl=32.13% magnetic field length

bx=-0.2314;
by=0.9664;
bz=0.1116;

q0dot=(-1/2)*q1*wx-(1/2)*q2*wy-(1/2)*q3*wz;
q1dot=(1/2)*q0*wx-(1/2)*q3*wy+(1/2)*q2*wz;
q2dot=(1/2)*q3*wx+(1/2)*q0*wy-(1/2)*q1*wz;
q3dot=(-1/2)*q2*wx+(1/2)*q1*wy+(1/2)*q0*wz;

wxdot=(-1/wtau)*wx;
wydot=(-1/wtau)*wy;
wzdot=(-1/wtau)*wz;

%f0=((-2)*q1*q3+2*q0*q2);
%f1=(-2*q0*q1-2*q2*q3);
%f2=(-q0^2+q2^2+q1^2-q3^2);

f0=(2*q1*q3-2*q0*q2)*g;
f1=(2*q0*q1+2*q2*q3)*g;
f2=(q0^2-q2^2-q1^2+q3^2)*g;

b0=(q0^2+q1^2-q2^2-q3^2)*bx+(2*q1*q2+2*q0*q3)*by+(2*q1*q3-2*q0*q2)*bz;
b1=(2*q1*q2-2*q0*q3)*bx+(q0^2-q1^2+q2^2-q3^2)*by+(2*q2*q3+2*q0*q1)*bz;
b2=(2*q1*q3+2*q0*q2)*bx+(2*q2*q3-2*q0*q1)*by+(q0^2-q1^2-q2^2+q3^2)*bz;

%b0t=(q0t^2+q1t^2-q2t^2-q3t^2)*bx+(2*q1t*q2t+2*q0t*q3t)*by+(2*q1t*q3t-
2*q0t*q2t)*bz;
```

```
%b1t=(2*q1t*q2t-2*q0t*q3t)*bx+(q0t^2-q1t^2+q2t^2-
q3t^2)*by+(2*q2t*q3t+2*q0t*q1t)*bz;
%b2t=(2*q1t*q3t+2*q0t*q2t)*bx+(2*q2t*q3t-2*q0t*q1t)*by+(q0t^2-q1t^2-
q2t^2+q3t^2)*bz;


JdQQ=jacobian([q0dot;q1dot;q2dot;q3dot],[q0;q1;q2;q3]);
JdfQ=jacobian([f0;f1;f2],[q0;q1;q2;q3]);
JdbQ=jacobian([b0;b1;b2],[q0;q1;q2;q3]);

%model noise

W=[0.001 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.001 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.001 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.001 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.001 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.001 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.001];

%observation matrix

C=[0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 1.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 1.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 1.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0];

%observation nosie

V=[0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.01 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.01 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.01 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.01 0.0 0.0 0.0
  0.0 0.0 0.0 0.00 0.0 0.0 0.5 0.0 0.0
  0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.5 0.0
  0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.5];
%initial prediction
```

```
Pkplus=[0.01 0.0 0.0 0.0 0.0 0.0 0.0
      0.0 0.01 0.0 0.0 0.0 0.0 0.0
      0.0 0.0 0.01 0.0 0.0 0.0 0.0
      0.0 0.0 0.0 0.01 0.0 0.0 0.0
      0.0 0.0 0.0 0.0 0.01 0.0 0.0
      0.0 0.0 0.0 0.0 0.0 0.01 0.0
      0.0 0.0 0.0 0.0 0.0 0.0 0.01];
%initial condition
q0=1;
q1=0;
q2=0;
q3=0;
wx=0;
wy=0;
wz=0;
%f0=g;
%f1=0;
%f2=0;
data=dlmread('gyroData.txt');
stateMemory=[0 0 0 0 0 0 0];
l=length(data);
Xe=[q0,q1, q2, q3, wx, wy, wz]';
ofs=[-0.255 0.120 0.759 -0.00024 0.0002 0.00002 -3.8 -6 3];
deg2rad=pi/180;
dt=0.05;

delta=[0 0 0 0 0 0 0 0 0];
for i=1:l
   A=zeros(7);
   A(1:4,1:4)=eval(JdQQ);
   A(5:7,5:7)=eye(3)*1/wtau;
    %prediction
   Xkplusunmoins=Xe+A*Xe*dt;
    Ad=eye(7)+A*dt;
    Wd=W*dt+(A*W+(A*W)')*(dt^2)/2+A*W*A*(dt^3)/3;
    Pkplusunmoins=Ad*Pkplus*Ad'+Wd;


    %measurement
    amp=sqrt((data(i,1)-ofs(1))^2+(data(i,2)-ofs(2))^2+(data(i,3)-ofs(3))^2);
    mfAmp=sqrt((data(i,11)-ofs(7))^2+(data(i,12)-ofs(8))^2+(data(i,13)-ofs(9))^2);
    Ym=([(data(i,1:3)-ofs(1:3))/amp*g,data(i,6:8)*deg2rad,(data(i,11)-
ofs(7))/mfAmp,(data(i,12)-ofs(8))/mfAmp,(data(i,13)-ofs(9))/mfAmp]-[0 0 0 ofs(4)
ofs(5) ofs(6) 0 0 0])';
```

```
    %explanation take g=amplitude of measurement
    q0=Xkplusunmoins(1);
    q1=Xkplusunmoins(2);
    q2=Xkplusunmoins(3);
    q3=Xkplusunmoins(4);

    C(1:3,1:4)=eval(JdfQ);
    C(7:9,1:4)=eval(JdbQ);

    %update

    K=(Pkplusunmoins*C')/(C*Pkplusunmoins*C'+V);




yk=[eval(f0),eval(f1),eval(f2),Xkplusunmoins(5),Xkplusunmoins(6),Xkplusunmoins(7),e
val(b0),eval(b1),eval(b2)]'

   delta=[delta;(Ym-yk)'];
    Xkplusunplus=Xkplusunmoins+K*(Ym-yk);
    Xe=Xkplusunplus;
    Pkplusunplus=(eye(size(K*C,1))-K*C)*Pkplusunmoins;
    Pkplus=Pkplusunplus;

    q0=Xe(1);
   q1=Xe(2);
   q2=Xe(3);
   q3=Xe(4);
   wx=Xe(5);
   wy=Xe(6);
   wz=Xe(7);

   stateMemory=[stateMemory;Xe'];


end
dlmwrite('stateVec.txt',stateMemory);
```

APPENDIX 5: MATLAB CODE FOR INTEGRATED ALGORITHM

```
%implementation of the first phase. First using the byZ Y head example
clc
clear
%close all

syms q0 q1 q2 q3  x y z ax ay az u v w wx wy wz f0 f1 f2 b0 b1 b2;
g=-9.81;
tau=1;
taup=1;
bvn=[-0.2314  0.9664 0.1116]';
bx=0.9664;
by=-0.2314;
bz=-0.1116;



xdot=u;
ydot=v;
zdot=w;

udot=(q0^2+q1^2-q2^2-q3^2)*ax+(2*q1*q2-2*q0*q3)*ay+(2*q1*q3+2*q0*q2)*az;
vdot=(2*q1*q2+2*q0*q3)*ax+(q0^2-q1^2+q2^2-q3^2)*ay+(2*q2*q3-2*q0*q1)*az;
wdot=(2*q1*q3-2*q0*q2)*ax+(2*q2*q3+2*q0*q1)*ay+(q0^2-q1^2-q2^2+q3^2)*az;

axdot=(-1/tau)*ax;
aydot=(-1/tau)*ay;
azdot=(-1/tau)*az;
%axdot=(-1/tau)*ax+(2*q1*q3-2*q0*q2)*g;
%aydot=(-1/tau)*ay+(2*q2*q3+2*q0*q1)*g;
%azdot=(-1/tau)*az+(q0^2-q1^2-q2^2+q3^2)*g;

Qcrt=1-sqrt(q0^2+q1^2+q2^2+q3^2);%quaternion correction

q0dot=(-1/2)*q1*wx-(1/2)*q2*wy-(1/2)*q3*wz%+Qcrt*q0;
q1dot=(1/2)*q0*wx-(1/2)*q3*wy+(1/2)*q2*wz%+Qcrt*q1;
q2dot=(1/2)*q3*wx+(1/2)*q0*wy-(1/2)*q1*wz%+Qcrt*q2;
q3dot=(-1/2)*q2*wx+(1/2)*q1*wy+(1/2)*q0*wz%+Qcrt*q3;

wxdot=(-1/taup)*wx;
wydot=(-1/taup)*wy;
wzdot=(-1/taup)*wz;
```

```
% measurement
f0=-(2*q1*q3-2*q0*q2);
f1=-(2*q0*q1+2*q2*q3);
f2=-(q0^2-q2^2-q1^2+q3^2);

b0=(q0^2+q1^2-q2^2-q3^2)*bx+(2*q1*q2+2*q0*q3)*by+(2*q1*q3-2*q0*q2)*bz;
b1=(2*q1*q2-2*q0*q3)*bx+(q0^2-q1^2+q2^2-q3^2)*by+(2*q2*q3+2*q0*q1)*bz;
b2=(2*q1*q3+2*q0*q2)*bx+(2*q2*q3-2*q0*q1)*by+(q0^2-q1^2-q2^2+q3^2)*bz;

JdVA=jacobian([udot;vdot;wdot],[ax;ay;az]);
%JdVQ=jacobian([udot;vdot;wdot],[q0;q1;q2;q3]);
%JdAQ=jacobian([axdot;aydot;azdot],[q0;q1;q2;q3]);
JdQQ=jacobian([q0dot;q1dot;q2dot;q3dot],[q0;q1;q2;q3]);
JdfQ=jacobian([f0;f1;f2],[q0;q1;q2;q3]);
JdbQ=jacobian([b0;b1;b2],[q0;q1;q2;q3]);

ofs=[-0.255 0.7025 0.759 -0.00024 0.0002 0.00002 -3.8 -6 3];
% phase 1 parameters
X=[x;y;z;u;v;w;ax;ay;az;q0;q1;q2;q3;wx;wy;wz];
%Observation Matrix
C=[1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0];

%Model noises

W=[0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
  0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004];

%observation Noises phase 1
V=[ 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.01 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.01 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.01 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.0010
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.0010 0.000 0.000 0.000 0.000
```

```
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.0010 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.500 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.500 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.500];

%Initial Prediction
Pkplus=[0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05];




%phase2 parameters
oldPosition=[0 0 0];
newPosition=[0 0 0];
um=0;
vm=0;
wm=0;
xm=0;
ym=0;
zm=0;
dspVec=[0 0 0];%displacement vector
norDspVec=[sqrt(3)/3 sqrt(3)/3 sqrt(3)/3];%normalised displacement vector
adr='WifiMf1\xwalking\test4\'
data=dlmread( strcat(adr,'gyroData.txt'));
deg2rad=pi/180;
```

```
%Initial condition

   x=0;
   y=0;
   z=0;
   u=0;
   v=0;
   w=0;
   ax=0;
   ay=0;
   az=0;
   q0=0;
   q1=0;
   q2=1;
   q3=0;
   wx=data(1,6)*deg2rad;
   wy=data(1,7)*deg2rad;
   wz=data(1,8)*deg2rad;

Xe=eval(X);
Xe2=[Xe(1:6)];
Fs=20;
dt=1/Fs;
Fs2=1;
dt2=1/Fs2;
stride=0;
l=length(data)
%position=[0 0 0];
stateMemory=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
G=[0 0 0 0 0 0 0 0 g 0 0 0 0 0 0 0]';
%quaternion=[0,0,0,0]

%step counting parameters
accRef=sqrt(data(1,3)^2+data(1,4)^2+data(1,5)^2);
flagUp=0;
stepPt=[0,0];
stepNo=0;%will not be used
accPeak1=[1,accRef];%[index, acc]
accPeak2=[1,accRef];%[index, acc]

walkingLength=0;
posiMem=[0 0 0];

stridesMem=0;
```

```
flag1s=0;
q01=q0;
q11 =q1;
q21 =q2;
q31=q3;
%

for i=0:l/20
   %phase 1
   for j=1:20



      index=20*i+j;
      if(index>l)
         break;
      end;
      A=zeros(16);
      A(1:3,4:6)=eye(3);
      A(4:6,7:9)=eval(JdVA);
      A(7:9,7:9)=-eye(3);
      A(10:13,10:13)=eval(JdQQ);
      A(14:16,14:16)=-eye(3);

      %Prediction
      Xkplusunmoins=Xe+A*Xe*dt;
      Ad=eye(16)+A*dt;
      Wd=W*dt+(A*W+(A*W)')*(dt^2)/2+A*W*A*(dt^3)/3;
      Pkplusunmoins=Ad*Pkplus*Ad'+Wd;


      % to compensate the gravity

      gravityM=[2*(q1*q3-q0*q2)  2*q2*q3+2*q0*q1  q0^2-q1^2-q2^2+q3^2];

      acc=[data(index,3:5)-ofs(1:3)]-gravityM*g;
      [eval(udot),eval(vdot),eval(wdot)]

      amp=sqrt((data(index,3)-ofs(1))^2+(data(index,4)-ofs(2))^2+(data(index,5)-
ofs(3))^2);
      mfAmp=sqrt((data(index,9)-ofs(7))^2+(data(index,10)-ofs(8))^2+(data(index,11)-
ofs(9))^2);
```

```
    Ym=([xm,ym,0,um,vm,0,acc,data(index,6:8)*deg2rad,(data(index,3:5)-
ofs(1:3))/amp,(data(index,9)-ofs(7))/mfAmp,(data(index,10)-
ofs(8))/mfAmp,(data(index,11)-ofs(9))/mfAmp]-[0 0 0 0 0 0 0 0 0 ofs(4) ofs(5) ofs(6) 0
0 0 0 0 0])';
   %explanation take g=amplitude of measurement


    %counting step
    accNorm=sqrt(data(index,3)^2+data(index,4)^2+data(index,5)^2);
    if(flagUp==0)
      flag=(accNorm-accRef)>3.5;

    else
      flag=(accRef-accNorm)>3.5;
    end
    if(flag==1)       %means one step is found
      if(accRef>accNorm)
        flagUp=0;
      else flagUp=1;
      end
      accPeak1=accPeak2;
      accPeak2=[index,accRef];
      %
meanAcc=[mean(data(accPeak1(1,1):accPeak2(1,1),1)),mean(data(accPeak1(1,1):accPe
ak2(1,1),2)),mean(data(accPeak1(1,1):accPeak2(1,1),3))];
      stride=[stride;sqrt(sqrt(abs(accPeak1(1,1)-
accPeak2(1,1))))/4.1277];%1/2*((accPeak2(1,1)-
accPeak1(1,1))*dt)^2*abs(accPeak2(1,2)-accPeak1(1,2))];
       %stride=[stride;1/2*((accPeak2(1,1)-
accPeak1(1,1))*dt)^2*abs(sqrt(meanAcc(1,1)^2+meanAcc(1,2)^2+meanAcc(1,3)^2))];

      stepPt=[stepPt;index,accRef];

    else
      if(((flagUp==1)&accRef<accNorm)|((flagUp==0)&accRef>accNorm))
        if(index-accPeak1(1,1)>20)
          accPeak1=accPeak2;
          accPeak2=[index,accRef];
        end

        accRef=accNorm;

      end
```

```
end

%Ym=[xm,ym,zm,um,vm,wm,acc,(data(index,6:8)-ofs(4:6))*deg2rad]';

%Update phase
 q0=Xkplusunmoins(10);
 q1=Xkplusunmoins(11);
 q2=Xkplusunmoins(12);
 q3=Xkplusunmoins(13);


%update observation matrix
C(13:15,10:13)=eval(JdfQ);
C(16:18,10:13)=eval(JdbQ);

K=(Pkplusunmoins*C')/(C*Pkplusunmoins*C'+V);

tmp=C*Xkplusunmoins;
yk=[tmp(1:12,1)', eval(f0),eval(f1),eval(f2),eval(b0),eval(b1),eval(b2)]';
Xkplusunplus=Xkplusunmoins+K*(Ym-yk);
Xe=Xkplusunplus;

Pkplusunplus=(eye(size(K*C,1))-K*C)*Pkplusunmoins*(eye(size(K*C,1))-
K*C)'+K*V*K';
Pkplus=Pkplusunplus;


 x=Xe(1,1);
 y=Xe(2,1);
 z=Xe(3,1);
 u=Xe(4,1);
 v=Xe(5,1);
 w=Xe(6,1);
 ax=Xe(7,1);
 ay=Xe(8,1);
 az=Xe(9,1);
 q0=Xe(10,1);
 q1=Xe(11,1);
 q2=Xe(12,1);
 q3=Xe(13,1);
 wx=Xe(14,1);
 wy=Xe(15,1);
 wz=Xe(16,1);
```

```
      stateMemory=[stateMemory;Xe(1:16,1)',data(index,12) ];

   end
    %phase 2


   newPosition=[x,y,z];
   dspVec=newPosition-oldPosition;
   norDspVec=(dspVec)/sqrt(dspVec(1,1)^2+dspVec(1,2)^2+dspVec(1,3)^2);
   strides=sum(stride);
   walkingLength=walkingLength+sum(stride);
   um=norDspVec(1,1)*strides*2;
   vm=norDspVec(1,2)*strides*2;
   wm=0;

    if(sum(stride)==0)
      xm=Xe(1,1);
      ym=Xe(2,1);


      Xe(4,1)=0;
      Xe(5,1)=0;
      Xe(6,1)=0;


   end

   xm=xm+um;
   ym=ym+vm;
   zm=zm+wm;


  Xe(3,1)=0;

   stridesMem=stridesMem+strides;
   oldPosition=newPosition;
   stride=0;
   %oldPosition-[xm,ym,0]
   posiMem=[posiMem;xm,ym,zm];

   i
end
```

```
walkingLength

dlmwrite(strcat(adr,'stateVec2.txt'),stateMemory);
dlmwrite(strcat(adr,'posiMem2.txt'),posiMem);
```

APPENDIX 6: MATLAB CODE FOR DRA-II

```
%implementation of the first phase. First using the byZ Y head example
clc
clear
%close all

syms q0 q1 q2 q3  x y z ax ay az u v w wx wy wz f0 f1 f2 b0 b1 b2;
g=-9.81;
tau=1;
taup=1;
bvn=[-0.2314  0.9664 0.1116]';
bx=0.8664;
by=-0.2114;
bz=-0.1116;

bamp=34.5197



xdot=u;
ydot=v;
zdot=w;

udot=(q0^2+q1^2-q2^2-q3^2)*ax+(2*q1*q2-2*q0*q3)*ay+(2*q1*q3+2*q0*q2)*az;
vdot=(2*q1*q2+2*q0*q3)*ax+(q0^2-q1^2+q2^2-q3^2)*ay+(2*q2*q3-2*q0*q1)*az;
wdot=(2*q1*q3-2*q0*q2)*ax+(2*q2*q3+2*q0*q1)*ay+(q0^2-q1^2-q2^2+q3^2)*az;

axdot=(-1/tau)*ax;
aydot=(-1/tau)*ay;
azdot=(-1/tau)*az;
%axdot=(-1/tau)*ax+(2*q1*q3-2*q0*q2)*g;
%aydot=(-1/tau)*ay+(2*q2*q3+2*q0*q1)*g;
%azdot=(-1/tau)*az+(q0^2-q1^2-q2^2+q3^2)*g;

Qcrt=1-sqrt(q0^2+q1^2+q2^2+q3^2);%quaternion correction

q0dot=(-1/2)*q1*wx-(1/2)*q2*wy-(1/2)*q3*wz%+Qcrt*q0;
q1dot=(1/2)*q0*wx-(1/2)*q3*wy+(1/2)*q2*wz%+Qcrt*q1;
q2dot=(1/2)*q3*wx+(1/2)*q0*wy-(1/2)*q1*wz%+Qcrt*q2;
q3dot=(-1/2)*q2*wx+(1/2)*q1*wy+(1/2)*q0*wz%+Qcrt*q3;

wxdot=(-1/taup)*wx;
wydot=(-1/taup)*wy;
wzdot=(-1/taup)*wz;
```

```
% measurement
f0=-(2*q1*q3-2*q0*q2);
f1=-(2*q0*q1+2*q2*q3);
f2=-(q0^2-q2^2-q1^2+q3^2);

b0=(q0^2+q1^2-q2^2-q3^2)*bx+(2*q1*q2+2*q0*q3)*by+(2*q1*q3-2*q0*q2)*bz;
b1=(2*q1*q2-2*q0*q3)*bx+(q0^2-q1^2+q2^2-q3^2)*by+(2*q2*q3+2*q0*q1)*bz;
b2=(2*q1*q3+2*q0*q2)*bx+(2*q2*q3-2*q0*q1)*by+(q0^2-q1^2-q2^2+q3^2)*bz;

JdVA=jacobian([udot;vdot;wdot],[ax;ay;az]);
%JdVQ=jacobian([udot;vdot;wdot],[q0;q1;q2;q3]);
%JdAQ=jacobian([axdot;aydot;azdot],[q0;q1;q2;q3]);
JdQQ=jacobian([q0dot;q1dot;q2dot;q3dot],[q0;q1;q2;q3]);
JdfQ=jacobian([f0;f1;f2],[q0;q1;q2;q3]);
JdbQ=jacobian([b0;b1;b2],[q0;q1;q2;q3]);

%JdQomega=jacobian([q0dot;q1dot;q2dot;q3dot],[wx;wy;wz]);% This matrix shall not
be used

%offset matrix
%ofs=[-0.255 0.120 0.759 -0.00024 0.0002 0.00002 ];
ofs=[-0.255 0.7025 0.759 -0.00024 0.0002 0.00002 -4.6451 2.1553 3];
% phase 1 parameters
X=[x;y;z;u;v;w;ax;ay;az;q0;q1;q2;q3;wx;wy;wz];
%Observation Matrix
C=[1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0];
```

```
%Model noises

W=[0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0004];

%observation Noises phase 1
V=[ 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.01 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.01 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.01 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
```

```
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.010 0.000
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.010
0.000 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.010 0.000 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.010 0.000 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.010 0.000
   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.010];

%Initial Prediction
Pkplus=[0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.05];




%phase2 parameters
oldPosition=[0 0 0];
newPosition=[0 0 0];
um=0;
vm=0;
wm=0;
xm=0;
ym=0;
```

```
zm=0;
dspVec=[0 0 0];%displacement vector
norDspVec=[sqrt(3)/3 sqrt(3)/3 sqrt(3)/3];%normalised displacement vector
adr='WifiMf1\newMFPosi2\test2\';
data=dlmread( strcat(adr,'calibrateData.txt'));
deg2rad=pi/180;
%Initial condition

   x=0;
   y=0;
   z=0;
   u=0;
   v=0;
   w=0;
   ax=0;
   ay=0;
   az=0;
   q0=0;
   q1=0;
   q2=1;
   q3=0;
   wx=data(1,6)*deg2rad;
   wy=data(1,7)*deg2rad;
   wz=data(1,8)*deg2rad;

Xe=eval(X);
Xe2=[Xe(1:6)];
Fs=20;
dt=1/Fs;
Fs2=1;
dt2=1/Fs2;
stride=0;
l=length(data)
%position=[0 0 0];
stateMemory=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
G=[0 0 0 0 0 0 0 0 g 0 0 0 0 0 0 0]';
%quaternion=[0,0,0,0]

%step counting parameters
accRef=sqrt(data(1,3)^2+data(1,4)^2+data(1,5)^2);
flagUp=0;
stepPt=[0,0];
stepNo=0;%will not be used
accPeak1=[1,accRef];%[index, acc]
```

```
accPeak2=[1,accRef];%[index, acc]

walkingLength=0;
posiMem=[0 0 0];

stridesMem=0;

flag1s=0;
q01=0;
q11 =0;
q21 =1;
q31=0;
%
mNotOk=0
gNotOk=0;
for i=0:l/20
   %phase 1
   for j=1:20



      index=20*i+j;
      if(index>l)
         break;
      end;
      A=zeros(16);
      A(1:3,4:6)=eye(3);
      A(4:6,7:9)=eval(JdVA);
      A(7:9,7:9)=-eye(3);
      A(10:13,10:13)=eval(JdQQ);
      A(14:16,14:16)=-eye(3);

      %Prediction
      Xkplusunmoins=Xe+A*Xe*dt;
      Ad=eye(16)+A*dt;
      Wd=W*dt+(A*W+(A*W)')*(dt^2)/2+A*W*A*(dt^3)/3;
      Pkplusunmoins=Ad*Pkplus*Ad'+Wd;
      %angular

    if(abs(data(index,8))>80)
       flag1s=1;
  %     um=0;
   %     vm=0;
    %    Xkplusunmoins(1,1)=xm;
```

```
%   Xkplusunmoins(2,1)=ym;
 %  Xkplusunmoins(4,1)=0;
  % Xkplusunmoins(5,1)=0;
   %flag1s=1;
end



   % to compensate the gravity

   gravityM=[2*(q1*q3-q0*q2)  2*q2*q3+2*q0*q1  q0^2-q1^2-q2^2+q3^2];
   acc=[data(index,3:5)-ofs(1:3)]-gravityM*g;

   amp=sqrt((data(index,3)-ofs(1))^2+(data(index,4)-ofs(2))^2+(data(index,5)-
ofs(3))^2);
   mfAmp=sqrt((data(index,9)-ofs(7))^2+(data(index,10)-ofs(8))^2+(data(index,11)-
ofs(9))^2);
   Ym=([xm,ym,0,um,vm,0,acc,data(index,6:8)*deg2rad,(data(index,3:5)-
ofs(1:3))/amp,(data(index,9)-ofs(7))/mfAmp,(data(index,10)-
ofs(8))/mfAmp,(data(index,11)-ofs(9))/mfAmp]-[0 0 0 0 0 0 0 0 0 ofs(4) ofs(5) ofs(6) 0
0 0 0 0 0])';
   if abs(mfAmp-bamp)>5
     V(16,16)= 100;
     V(17,17)= 100;
     V(18,18)= 100;
     mNotOk=mNotOk+1
   else
     V(16,16)= 0.01;
     V(17,17)= 0.01;
     V(18,18)= 0.01;
   end
   if abs(amp-9.8)>2
      V(7,7)=100;
      V(8,8)=100;
      V(9,9)=100;
      gNotOk=gNotOk+1
   else
      V(7,7)=0.01;
      V(8,8)=0.01;
      V(9,9)=0.01;
   end
   %explanation take g=amplitude of measurement
```

```
%counting step
accNorm=sqrt(data(index,3)^2+data(index,4)^2+data(index,5)^2);
if(flagUp==0)
  flag=(accNorm-accRef)>3.5;

else
  flag=(accRef-accNorm)>3.5;
end
if(flag==1)       %means one step is found
  if(accRef>accNorm)
    flagUp=0;
  else flagUp=1;
  end
  accPeak1=accPeak2;
  accPeak2=[index,accRef];
 %
meanAcc=[mean(data(accPeak1(1,1):accPeak2(1,1),1)),mean(data(accPeak1(1,1):accPe
ak2(1,1),2)),mean(data(accPeak1(1,1):accPeak2(1,1),3))];
   stride=[stride;sqrt(sqrt(abs(accPeak1(1,1)-
accPeak2(1,1))))/4.1277];%1/2*((accPeak2(1,1)-
accPeak1(1,1))*dt)^2*abs(accPeak2(1,2)-accPeak1(1,2))];
    %stride=[stride;1/2*((accPeak2(1,1)-
accPeak1(1,1))*dt)^2*abs(sqrt(meanAcc(1,1)^2+meanAcc(1,2)^2+meanAcc(1,3)^2))];

    stepPt=[stepPt;index,accRef];

else
  if(((flagUp==1)&accRef<accNorm)|((flagUp==0)&accRef>accNorm))
    if(index-accPeak1(1,1)>20)
      accPeak1=accPeak2;
      accPeak2=[index,accRef];
    end

    accRef=accNorm;

  end


end

%Ym=[xm,ym,zm,um,vm,wm,acc,(data(index,6:8)-ofs(4:6))*deg2rad]';

%Update phase
```

```
      q0=Xkplusunmoins(10);
      q1=Xkplusunmoins(11);
      q2=Xkplusunmoins(12);
      q3=Xkplusunmoins(13);


      %update observation matrix
      C(13:15,10:13)=eval(JdfQ);
      C(16:18,10:13)=eval(JdbQ);

      K=(Pkplusunmoins*C')/(C*Pkplusunmoins*C'+V);

      tmp=C*Xkplusunmoins;
      yk=[tmp(1:12,1)', eval(f0),eval(f1),eval(f2),eval(b0),eval(b1),eval(b2)]';
      Xkplusunplus=Xkplusunmoins+K*(Ym-yk);
      Xe=Xkplusunplus;

      Pkplusunplus=(eye(size(K*C,1))-K*C)*Pkplusunmoins*(eye(size(K*C,1))-
K*C)'+K*V*K';
      Pkplus=Pkplusunplus;


      x=Xe(1,1);
      y=Xe(2,1);
      z=Xe(3,1);
      u=Xe(4,1);
      v=Xe(5,1);
      w=Xe(6,1);
      ax=Xe(7,1);
      ay=Xe(8,1);
      az=Xe(9,1);
      q0=Xe(10,1);
      q1=Xe(11,1);
      q2=Xe(12,1);
      q3=Xe(13,1);
      wx=Xe(14,1);
      wy=Xe(15,1);
      wz=Xe(16,1);

      stateMemory=[stateMemory;Xe(1:16,1)',data(index,12) ];

   end
    %phase 2
```

```
   strides=sum(stride);
   walkingLength=walkingLength+sum(stride);

   qtheta=atan2(2*(q0*q3+q1*q2),1-2*(q2^2+q3^2));
   qtheta1=atan2(2*(q01*q31+q11*q21),1-2*(q21^2+q31^2));
         delta=qtheta-qtheta1 ;

   um=strides*sin(delta);
   vm=strides*cos(delta);
   wm=0;
   xm=xm+um;
   ym=ym+vm;
   zm=zm+wm;
   stridesMem=stridesMem+strides;

   stride=0;
   posiMem=[posiMem;xm,ym,zm];

   i
end

walkingLength

dlmwrite(strcat(adr,'stateVec9.txt'),stateMemory);
dlmwrite(strcat(adr,'posiMem9.txt'),posiMem);
```

APPENDIX 7: NOVEL CALIBRATION ALGORITHM FOR MAGNETOMETER

The novel calibration algorithm for magnetometer is proposed in [32] and is summarized in this section.

Magnetometer measurement errors can be classified as wide band random noises, scaling factors, misalignment, and magnetic field interferences.

Wide Band Radom noises

Wide band random noises are due to electrical circuits, or environment changes, such as voltages fluctuations, temperatures changes, etc. These parameters might change sensors' sensitivity or create random magnetic field interferences, and thus corrupted the magnetometer's output. Usually the wide band random noises are modeled as a zero mean white Gaussian random noises. Noise deviation of the magnetometer used in this project is 0.5 μT .

The wide band noise is represented as:

$$\vec{\sigma} = \begin{bmatrix} \sigma_x & \sigma_y & \sigma_z \end{bmatrix}^T \tag{55}$$

Scaling Factors

Scaling factors are due to different sensitivities of each sensing elements of a triad magnetometer; in other words, the proportionalities of output from three axes to the same input are different. In particular, in the indoor environment, the shielding effect of buildings that reduces the geomagnetic strength, introduces attenuation effect to

magnetometer measurements. For example, in AMI lab, the magnetic field amplitude is always less than 33 μT, while the international Geomagnetic Field Reference indicates the magnetic field strength is about 42.122 μT.

The scaling factor is represented as the following matrix:

$$C_{sf} = \begin{bmatrix} 1+sf_x & 0 & 0 \\ 0 & 1+sf_y & 0 \\ 0 & 0 & 1+sf_z \end{bmatrix} \quad (56)$$

Misalignment

In an ideal installation, the magnetometer's axes are perfectly aligned with the device's axes. In practice, there are always some misalignments and affect magnetometer's measurements. [32]models them by the misalignment matrix C_m which is described as follows:

$$C_m = \begin{bmatrix} 1 & -\varepsilon_z & \varepsilon_y \\ \varepsilon_z & 1 & -\varepsilon_x \\ -\varepsilon_y & \varepsilon_x & 1 \end{bmatrix}$$

C_m is like a small rotation matrix that brings the magnetometer axes to be perfectly aligned with the body axes. In this project, the magnetometer is mounted to the mother board of Samsung Tab, and misalignments are assumed as negligible. Therefore, C_m is an identity matrix.

Magnetic field interference

External magnetic interference can be divided into two categories, soft iron and hard iron interferences. Hard iron consists of constant inteferences or slow-varying

interference generated by ferromagnetic materials near the magnetometer. The soft iron comes from materials that "generate their own magnetic field in response to an external magnetic field". [32]

Then we can represent the hard iron as:

$$\vec{b} = \begin{bmatrix} b_{x0} & b_{y0} & b_{z0} \end{bmatrix}^T$$

The soft iron can be represented as:

$$C_{si} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}$$

From (6), it is justified that $\alpha_{ij} = 0$ if $i \neq j$.

Therefore, the measurement vector of magnetic field can be represented as:

$$\hat{\vec{h}}^b = R_{rot}\left[C_m C_{sf} C_{si}(\vec{h}^G + \vec{b} + \vec{\sigma})\right] \tag{57}$$

The matrix R_{rot} is the rotation matrix from body coordinate to global coordinate. $\vec{h}^G$ is geomagnetic field vector in the global coordinate, or say geomagnetic field reference vector. In this project, as shown in section 2.2.1, it is defined as:

$$\vec{h}^G = \begin{bmatrix} m_{x0} \\ m_{y0} \\ m_{z0} \end{bmatrix} = \begin{bmatrix} 38.1994 \\ 13.9035 \\ 11.036 \end{bmatrix} \tag{58}$$

The vector $\hat{\vec{h}}^b = \begin{bmatrix} \hat{h}_x^b & \hat{h}_y^b & \hat{h}_z^b \end{bmatrix}^T$ is magnetometer measurement.

By making the assumption that the misalignment matrix is an identity matrix and $\alpha_{ij} = 0 \text{ if } i \neq j \text{ in} C_{si}$, we have the following equation:

$$h^2 = \left(\frac{\hat{h}_x^b - b_x}{\gamma_x} \right)^2 + \left(\frac{\hat{h}_y^b - b_y}{\gamma_y} \right)^2 + \left(\frac{\hat{h}_z^b - b_z}{\gamma_z} \right)^2 \tag{59}$$

h is geomagnetic field strength according to IGFR[33]. Equation (68) defines an ellipsoid with $\begin{pmatrix} b_x & b_y & b_z \end{pmatrix}$ and semi-axis lengths $\begin{pmatrix} \gamma_x & \gamma_y & \gamma_z \end{pmatrix}$. Then our next step is how to find this ellipsoid, given a set of measurements.

According to [32], the steps to find the parameters of the ellipsoid are explained in the following section.

Calibration Steps

The calibration algorithm includes two parts, initial condition estimation using a novel two-step nonlinear initial condition estimator and then an iterated, batch least squares estimator. [32]

Initial condition estimator:

From equation (68), we have

$$h^2 = \frac{\left(\hat{h}^b{}_x\right)^2 - 2b_x\hat{h}^b{}_x + \left(b_x\right)^2}{\gamma^2{}_x} + \frac{\left(\hat{h}^b{}_y\right)^2 - 2b_y\hat{h}^b{}_y + \left(b_y\right)^2}{\gamma^2{}_y} + \frac{\left(\hat{h}^b{}_z\right)^2 - 2b_z\hat{h}^b{}_z + \left(b_z\right)^2}{\gamma^2{}_z} \tag{60}$$

Therefore, given K measurements, we have the following relation:

$$-\begin{bmatrix} \left(\hat{h}_x^b(t_1)\right)^2 \\ \left(\hat{h}_x^b(t_2)\right)^2 \\ \vdots \\ \left(\hat{h}_x^b(t_{k-1})\right)^2 \\ \left(\hat{h}_x^b(t_K)\right)^2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} \end{bmatrix} \begin{bmatrix} b_x \\ \mu_2(b_y) \\ \mu_3(b_z) \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix} + \vec{v} \tag{61}$$

$$H_{11} = \begin{bmatrix} -2\hat{h}_x^b(t_1) & -2\hat{h}_y^b(t_1) & -2\hat{h}_z^b(t_1) \\ -2\hat{h}_x^b(t_2) & -2\hat{h}_y^b(t_2) & -2\hat{h}_z^b(t_2) \\ \vdots & \vdots & \vdots \\ -2\hat{h}_x^b(t_{K-1}) & -2\hat{h}_y^b(t_{K-1}) & -2\hat{h}_z^b(t_{K-1}) \\ -2\hat{h}_x^b(t_K) & -2\hat{h}_y^b(t_K) & -2\hat{h}_y^b(t_K) \end{bmatrix} \tag{62}$$

$$H_{12} = \begin{bmatrix} \left(\hat{h}_y^b(t_1)\right)^2 & \left(\hat{h}_z^b(t_1)\right)^2 & 1 \\ \left(\hat{h}_y^b(t_2)\right)^2 & \left(\hat{h}_z^b(t_2)\right)^2 & 1 \\ \vdots & \vdots & \vdots \\ \left(\hat{h}_y^b(t_{K-1})\right)^2 & \left(\hat{h}_z^b(t_{K-1})\right)^2 & 1 \\ \left(\hat{h}_y^b(t_K)\right)^2 & \left(\hat{h}_z^b(t_K)\right)^2 & 1 \end{bmatrix} \tag{63}$$

$$\begin{aligned} \mu_1 &= h^2\gamma_x^2 \\ \mu_2 &= \frac{\gamma_x^2}{\gamma_y^2} \\ \mu_3 &= \frac{\gamma_x^2}{\gamma_z^2} \\ \mu_4 &= b_x^2 + \mu_2(b_y)^2 + \mu_3(b_z)^2 - \mu_1 \end{aligned} \tag{64}$$

And

$$x = \begin{bmatrix} b_x & \mu_2 b_y & \mu_3 b_z & \mu_2 & \mu_3 & \mu_4 \end{bmatrix}^T \tag{65}$$

$$y = \begin{bmatrix} -\left(\hat{h}_x^b(t_1)\right)^2 & -\left(\hat{h}_x^b(t_2)\right)^2 & \dots & -\left(\hat{h}_x^b(t_{K-1})\right)^2 & -\left(\hat{h}_x^b(t_K)\right)^2 \end{bmatrix}^T \tag{66}$$

$$\hat{\vec{x}} = \left(HH^T\right)^{-1} H^T y \tag{67}$$

The parameters we want can be computed as:

$$
\begin{aligned}
\hat{b}_x &= \hat{\vec{x}}(1) \\
\hat{b}_y &= \frac{\hat{\vec{x}}(2)}{\hat{\vec{x}}(4)} \\
\hat{b}_z &= \frac{\hat{\vec{x}}(3)}{\hat{\vec{x}}(5)} \\
\mu_1 &= \hat{b}_x{}^2 + \hat{\vec{x}}(2)\hat{b}_y + \hat{\vec{x}}(3)\hat{b}_z - \mu_4 \\
\hat{\gamma}_x &= \sqrt{\frac{\mu_1}{h^2}} \\
\hat{\gamma}_y &= \sqrt{\frac{\mu_1}{\mu_2 h^2}} \\
\hat{\gamma}_x &= \sqrt{\frac{\mu_1}{\mu_3 h^2}}
\end{aligned}
\tag{68}
$$

Given $\left[\hat{b}_x \quad \hat{b}_y \quad \hat{b}_z \quad \hat{\gamma}_x \quad \hat{\gamma}_y \quad \hat{\gamma}_z\right]^T$, we can apply the iterative least square estimation.

[32]

From equation (68), by applying partial differentiation, we have:

$$
\begin{aligned}
-\delta h &= \left(\frac{h_x - b_x}{h\gamma_x}\right)\delta b_x + \frac{(h_x - b_x)^2}{h\gamma_x^3}\delta\gamma_x + \left(\frac{h_y - b_y}{h\gamma_{yx}}\right)\delta b_x + \frac{(h_y - b_y)^2}{h\gamma_y^3}\delta\gamma_y \\
&+ \left(\frac{h_z - b_z}{h\gamma_z}\right)\delta b_z + \frac{(h_z - b_z)^2}{h\gamma_z^3}\delta\gamma_z \\
&= \xi_x\delta b_x + \eta_x\delta\gamma_x + \xi_y\delta b_y + \eta_y\delta\gamma_y + \xi_z\delta b_z + \eta_z\delta\gamma_z \\
&= \begin{bmatrix}\xi_x & \eta_x & \xi_y & \eta_y & \xi_z & \eta_z\end{bmatrix}\begin{bmatrix}\delta b_x \\ \delta\gamma_x \\ \delta b_y \\ \delta\gamma_y \\ \delta b_z \\ \delta\gamma_z\end{bmatrix}
\end{aligned}
\tag{69}
$$

Thus, if we make K measurements, we have

$$-\begin{bmatrix} \delta h_1 \\ \delta h_2 \\ \vdots \\ \delta h_{K-1} \\ \delta h_K \end{bmatrix} = \begin{bmatrix} \xi_{x1} & \eta_{x1} & \xi_{y1} & \eta_{y1} & \xi_{z1} & \eta_{z1} \\ \xi_{x2} & \eta_{x2} & \xi_{y2} & \eta_{y2} & \xi_{z2} & \eta_{z2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \xi_{x_{K-1}} & \eta_{x_{K-1}} & \xi_{y_{K-1}} & \eta_{y_{K-1}} & \xi_{z_{K-1}} & \eta_{z_{K-1}} \\ \xi_{x_K} & \eta_{x_K} & \xi_{y_K} & \eta_{y_K} & \xi_{z_K} & \eta_{z_K} \end{bmatrix} \begin{bmatrix} \delta b_x \\ \delta \gamma_x \\ \delta b_y \\ \delta \gamma_y \\ \delta b_z \\ \delta \gamma_z \end{bmatrix} \tag{70}$$

Note that

$$\delta \hat{\vec{x}} = \begin{bmatrix} \delta b_x \\ \delta \gamma_x \\ \delta b_y \\ \delta \gamma_y \\ \delta b_z \\ \delta \gamma_z \end{bmatrix} \tag{71}$$

$$\delta h = \begin{bmatrix} \delta h_1 \\ \delta h_2 \\ \vdots \\ \delta h_{K-1} \\ \delta h_K \end{bmatrix} \tag{72}$$

We also have:

$$H = \begin{bmatrix} \xi_{x1} & \eta_{x1} & \xi_{y1} & \eta_{y1} & \xi_{z1} & \eta_{z1} \\ \xi_{x2} & \eta_{x2} & \xi_{y2} & \eta_{y2} & \xi_{z2} & \eta_{z2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \xi_{x_{K-1}} & \eta_{x_{K-1}} & \xi_{y_{K-1}} & \eta_{y_{K-1}} & \xi_{z_{K-1}} & \eta_{z_{K-1}} \\ \xi_{x_K} & \eta_{x_K} & \xi_{y_K} & \eta_{y_K} & \xi_{z_K} & \eta_{z_K} \end{bmatrix} \tag{73}$$

And

$$\delta h_i = h_i - \hat{h}_i \tag{74}$$

h_i is the magnetic strength amplitude from IGRF model and

$$\hat{h}_i = \left(\frac{h_{xi} - \hat{b}_{xi}}{\hat{\gamma}_{xi}}\right)^2 + \left(\frac{h_{yi} - \hat{b}_{yi}}{\hat{\gamma}_{yi}}\right)^2 + \left(\frac{h_{zi} - \hat{b}_{zi}}{\hat{\gamma}_{zi}}\right)^2 \tag{75}$$

(Here is different from [32]. In [32], $\hat{h}_i = \sqrt{\left(h_x^b + h_y^b + h_z^b\right)}$)

The steps of calibration are:

1. Using the parameters given by initial condition estimator, to compute H
2. Compute a least-mean square estimate : $\delta\hat{x} = \left(HH^T\right)^{-1} H^T \delta h$
3. Using the estimate $\delta\hat{x}$ to update the unknown parameters

$$\begin{aligned} b_x(+) &= \delta\hat{x}(1) + b_x(-) \\ \gamma_x(+) &= \delta\hat{x}(2) + \gamma_x(-) \\ b_y(+) &= \delta\hat{x}(4) + b_y(-) \\ \gamma_y(+) &= \delta\hat{x}(2) + \gamma_y(-) \\ b_z(+) &= \delta\hat{x}(6) + b_z(-) \\ \gamma_z(+) &= \delta\hat{x}(2) + \gamma_z(-) \end{aligned} \tag{76}$$

4. Compute the covariance matrix P by

$$P = \sigma_\omega^2 \left(H^T H\right)^{-1} \tag{77}$$

5. Return to step(2) and repeat until convergence is achieved, when the absolute value of $\delta\hat{x}(2) \le 10^{-5}$

APPENDIX 8: NOVEL CALIBRATION ALGORITHM FOR ACCELEROMETER

Accelerometers' measurement errors can be classified as measurement bias, scaling factors due to different sensitivity of each axis and wide band noises. Therefore, the acceleration vector can be represented by the measurement vector as follows:

$$\begin{aligned} g_x &= \frac{\hat{g}_x - g_{x0}}{\gamma_x} \\ g_y &= \frac{\hat{g}_y - g_{y0}}{\gamma_y} \\ g_z &= \frac{\hat{g}_z - g_{z0}}{\gamma_z} \end{aligned} \tag{78}$$

The vector $\begin{bmatrix} \hat{g}_x & \hat{g}_y & \hat{g}_z \end{bmatrix}^T$ is the acceleration measurement vector.

The vector $\begin{bmatrix} g_x & g_y & g_z \end{bmatrix}^T$ is the real acceleration vector.

The vector $\begin{bmatrix} g_{x0} & g_{y0} & g_{z0} \end{bmatrix}^T$ is the bias vector of the accelerometer's three axes.

$\gamma_x, \gamma_y, \gamma_z$ are scaling factors of each axis.

If the device is stationary, the ideal acceleration measurement is g, so we have the following equation:

$$g^2 = \left(\frac{\hat{g}_x - g_{x0}}{\gamma_x} \right)^2 + \left(\frac{\hat{g}_y - g_{y0}}{\gamma_y} \right)^2 + \left(\frac{\hat{g}_x - g_{x0}}{\gamma_x} \right)^2 \tag{79}$$

This equation obviously describes an ellipsoid, with its centre at $\begin{pmatrix} g_{x0} & g_{y0} & g_{z0} \end{pmatrix}$, and semi-axis lengths as $\begin{pmatrix} \gamma_x & \gamma_y & \gamma_z \end{pmatrix}$. In other words, the accelerometer readings lie

on an ellipsoid manifold. The calibration algorithm for magnetometer can also be applied to determine the ellipsoid parameters.

Firstly, the accelerometer's measurements are collected with the device being stationary in six status, say X, Y, Z axis of the device's body coordinate pointing upward/downward respectively and then measurement data is combined together under the name as accData, which is a K-by-3 matrix. Each row in accData means the accelerometer's measurements at its corresponding timestamp, and colomns are arranged to contain X, Y and Z axes measurements. For example, accData(1,i) means the accelerometer's X axis output at time t_i. For better understanding, it is noted as $\hat{g}_x(t_i)$. The Y and Z axes' output at time t_i are represented as $\hat{g}_y(t_i)$, $\hat{g}_z(t_i)$ respectively in the following explanation.

The algorithm contains two steps, initial condition estimation and iterative batch least square estimator.

In the initial condition estimation, first we define a vector as follows:

$$\hat{\bar{x}} = \begin{bmatrix} g_{x0} & \mu_2 g_{y0} & \mu_3 g_{z0} & \mu_2 & \mu_3 & \mu_4 \end{bmatrix}^T \tag{80}$$

The elements of $\hat{\bar{x}}$ are defined as:

$$\begin{aligned}
\mu_1 &= g^2\gamma_x^2 \\
\mu_2 &= \frac{\gamma_x^2}{\gamma_y^2} \\
\mu_3 &= \frac{\gamma_x^2}{\gamma_z^2} \\
\mu_4 &= g_x^2 + \mu_2\left(g_y\right)^2 + \mu_3\left(g_z\right)^2 - \mu_1
\end{aligned} \tag{81}$$

We also define a matrix H as:

$$H = \begin{bmatrix} H_{11} & H_{12} \end{bmatrix} \tag{82}$$

and H_{11} and H_{12} are:

$$H_{11} = \begin{bmatrix}
-2\hat{g}_x^b(t_1) & -2\hat{g}_y^b(t_1) & -2\hat{g}_z^b(t_1) \\
-2\hat{g}_x^b(t_2) & -2\hat{g}_y^b(t_2) & -2\hat{g}_z^b(t_2) \\
\vdots & \vdots & \vdots \\
-2\hat{g}_x^b(t_{K-1}) & -2\hat{g}_y^b(t_{K-1}) & -2\hat{g}_z^b(t_{K-1}) \\
-2\hat{g}_x^b(t_K) & -2\hat{g}_y^b(t_K) & -2\hat{g}_z^b(t_K)
\end{bmatrix} \tag{83}$$

$$H_{12} = \begin{bmatrix}
\left(\hat{g}_y^b(t_1)\right)^2 & \left(\hat{g}_z^b(t_1)\right)^{2^2} & 1 \\
\left(\hat{g}_y^b(t_2)\right)^{2^2} & \left(\hat{g}_z^b(t_2)\right)^{2^2} & 1 \\
\vdots & \vdots & \vdots \\
\left(\hat{g}_y^b(t_{K-1})\right)^{2^2} & \left(\hat{g}_z^b(t_{K-1})\right)^{2^2} & 1 \\
\left(\hat{g}_y^b(t_K)\right)^{2^2} & \left(\hat{g}_z^b(t_K)\right)^{2^2} & 1
\end{bmatrix} \tag{84}$$

Another vector is defined as:

$$\vec{y} = \begin{bmatrix} -\left(\hat{g}_x(t_1)\right)^2 & -\left(\hat{g}_x(t_2)\right)^2 & \cdots & -\left(\hat{g}_x(t_K)\right)^2 & -\left(\hat{g}_x(t_{K-1})\right)^2 \end{bmatrix}^T \tag{85}$$

From the relation (79), we have:

$$\vec{y} = H\hat{\vec{x}} \tag{86}$$

Then we have the estimation:

$$\hat{\vec{x}} = \left(HH^T\right)^{-1} H^T \vec{y} \tag{87}$$

Given the estimation of $\hat{\vec{x}}$ elements, we can establish thus an initial estimation of the desired parameters by the follow equations:

$$\begin{aligned}
\hat{g}_x &= \hat{\vec{x}}(1) \\
\hat{g}_y &= \frac{\hat{\vec{x}}(2)}{\hat{\vec{x}}(4)} \\
\hat{g}_z &= \frac{\hat{\vec{x}}(3)}{\hat{\vec{x}}(5)} \\
\mu_1 &= \hat{g}_x{}^2 + \hat{\vec{x}}(2)\hat{g}_y + \hat{\vec{x}}(3)\hat{g}_z - \mu_4 \\
\hat{\gamma}_x &= \sqrt{\frac{\mu_1}{g^2}} \\
\hat{\gamma}_y &= \sqrt{\frac{\mu_1}{\mu_2 g^2}} \\
\hat{\gamma}_x &= \sqrt{\frac{\mu_1}{\mu_3 g^2}}
\end{aligned} \tag{88}$$

The iterative batch least square estimator:

Similarly to the magnetometer calibration method, the differentiation of equation(79) is:

$$
\begin{aligned}
-\delta g &= \left(\frac{g_x - g_{x0}}{g\gamma_x}\right)\delta g_{x0} + \frac{\left(g_x - g_{x0}\right)^2}{g\gamma_x^3}\delta\gamma_x + \left(\frac{g_y - g_{y0}}{g\gamma_{yx}}\right)\delta b_x + \frac{\left(g_y - g_{y0}\right)^2}{g\gamma_y^3}\delta\gamma_y \\
&+ \left(\frac{g_z - g_{z0}}{g\gamma_z}\right)\delta b_z + \frac{\left(g_z - g_{z0}\right)^2}{g\gamma_z^3}\delta\gamma_z \\
&= \zeta_x\delta g_{x0} + \eta_x\delta\gamma_x + \zeta_y\delta g_{y0} + \eta_y\delta\gamma_y + \zeta_z\delta g_{z0} + \eta_z\delta\gamma_z \\
&= \begin{bmatrix}\zeta_x & \eta_x & \zeta_y & \eta_y & \zeta_z & \eta_z\end{bmatrix}\begin{bmatrix}\delta g_{x0} \\ \delta\gamma_x \\ \delta g_{y0} \\ \delta\gamma_y \\ \delta g_{z0} \\ \delta\gamma_z\end{bmatrix}
\end{aligned}
\tag{89}
$$

Following iterative steps are the same as the algorithm in Appendix 7.

APPENDIX 9: MATLAB CODE FOR DRA-III

```
clc
clear
close all

syms q0 q1 q2 q3 wx wy wz d0 d1 d2 f0 f1 f2 b0 b1 b2



wtau=1;
dtau=1;
g=9.8;

h=42.4;

adr='walking\';
data=dlmread( strcat(adr,'gyroData.txt'));
%initial quaternion of the device
initQ=dlmread( strcat(adr,'initQ.txt'));
qr0=initQ(1);
qr1=initQ(2);
qr2=initQ(3);
qr3=initQ(4);
%initial measurement of gravitational force
gF0=(2*initQ(2)*initQ(4)-2*initQ(1)*initQ(3))*g;
gF1=(2*initQ(1)*initQ(2)+2*initQ(3)*initQ(4))*g;
gF2=(initQ(1)^2-initQ(3)^2-initQ(2)^2+initQ(4)^2)*g;
gFix=[gF0,gF1,gF2]


% according to y of the lab
%normalization

bx=0.9069;
by=0.3301;
bz=0.1116;
%bv=[38.1994,-13.9035,-11.036]';
%bx=38.1994   ;
%by= -13.9035 ;
%bz=-11.036;

%north
```

```
%bx=40.651;
%by=0.154;
%bz=-11.036


bxo=0.4805    ;
byo= -0.2166  ;
bzo=-0.7032;
gamabX= 0.7362  ;
gamabY= 0.7723  ;
gamabZ=0.7403;
%   -0.3708   -0.2256    0.2424    1.0352    1.0891    1.0442
%0.4805 -0.2166 -0.7032 0.7362 0.7723 0.7403

gxo= -0.3708 ;
gyo= -0.2256 ;
gzo= 0.2424;
gamagX= 1.0352;
gamagY= 1.0891;
gamagZ= 1.0442;




q0dot=(-1/2)*q1*wx-(1/2)*q2*wy-(1/2)*q3*wz;
q1dot=(1/2)*q0*wx-(1/2)*q3*wy+(1/2)*q2*wz;
q2dot=(1/2)*q3*wx+(1/2)*q0*wy-(1/2)*q1*wz;
q3dot=(-1/2)*q2*wx+(1/2)*q1*wy+(1/2)*q0*wz;

wxdot=(-1/wtau)*wx;
wydot=(-1/wtau)*wy;
wzdot=(-1/wtau)*wz;

f0=(2*q1*q3-2*q0*q2)*g;
f1=(2*q0*q1+2*q2*q3)*g;
f2=(q0^2-q2^2-q1^2+q3^2)*g;

b0=(q0^2+q1^2-q2^2-q3^2)*bx+(2*q1*q2+2*q0*q3)*by+(2*q1*q3-2*q0*q2)*bz;
b1=(2*q1*q2-2*q0*q3)*bx+(q0^2-q1^2+q2^2-q3^2)*by+(2*q2*q3+2*q0*q1)*bz;
b2=(2*q1*q3+2*q0*q2)*bx+(2*q2*q3-2*q0*q1)*by+(q0^2-q1^2-q2^2+q3^2)*bz;


JdQQ=jacobian([q0dot;q1dot;q2dot;q3dot],[q0;q1;q2;q3]);
JdfQ=jacobian([f0;f1;f2],[q0;q1;q2;q3]);
```

```
JdbQ=jacobian([b0;b1;b2],[q0;q1;q2;q3]);

%model noise

W=[0.001 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.001 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.001 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.001 0.0 0.0 0.0
   0.0 0.0 0.0 0.0 0.001 0.0 0.0
   0.0 0.0 0.0 0.0 0.0 0.001 0.0
   0.0 0.0 0.0 0.0 0.0 0.0 0.001];

%observation matrix

C=[0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.0 1.0 0.0 0.0
   0.0 0.0 0.0 0.0 0.0 1.0 0.0
   0.0 0.0 0.0 0.0 0.0 0.0 1.0
   0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.0 0.0 0.0 0.0];

%observation nosie

V=[0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.01 0.0 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.0001 0.0 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.0 0.0001 0.0 0.0 0.0 0.0
   0.0 0.0 0.0 0.0 0.0 0.0001 0.0 0.0 0.0
   0.0 0.0 0.0 0.00 0.0 0.0 0.15 0.0 0.0
   0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.15 0.0
   0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.0 0.15];
%initial prediction
Pkplus=[0.01 0.0 0.0 0.0 0.0 0.0 0.0
        0.0 0.01 0.0 0.0 0.0 0.0 0.0
        0.0 0.0 0.01 0.0 0.0 0.0 0.0
        0.0 0.0 0.0 0.01 0.0 0.0 0.0
        0.0 0.0 0.0 0.0 0.01 0.0 0.0
        0.0 0.0 0.0 0.0 0.0 0.01 0.0
        0.0 0.0 0.0 0.0 0.0 0.0 0.01];
%initial condition
```

```
q0=qr0;
q1=qr1;
q2=qr2;
q3=qr3;
wx=0;
wy=0;
wz=0;

%step counting inialization
%step counting parameters
accRef=sqrt(data(1,3)^2+data(1,4)^2+data(1,5)^2);
flagUp=0;
stepPt=[0,0];
stepNo=0;%will not be used
accPeak1=[1,accRef];%[index, acc]
accPeak2=[1,accRef];%[index, acc]
stride=0;

walkingLength=0;
posiMem=[0 0 0];
stridesMem=0;
xm=0;
ym=0;
zm=0;
um=0;
vm=0;
%

stateMemory=[0 0 0 0 0 0 0];
l=length(data);
Xe=[q0,q1, q2, q3, wx, wy, wz]';
ofs=[-0.255 0.120 0.759 -0.00024 0.0002 0.00002 -3.8 -6 3];
deg2rad=pi/180;
dt=0.05;

%delta=[0 0 0 0 0 0 0 0 0];
for i=0:l/20
   for j=1:20



        index=20*i+j;
        if(index>l)
           break;
```

```
end;
A=zeros(7);
A(1:4,1:4)=eval(JdQQ);
A(5:7,5:7)=eye(3)*1/wtau;


%prediction
Xkplusunmoins=Xe+A*Xe*dt;
 Ad=eye(7)+A*dt;
 Wd=W*dt+(A*W+(A*W)')*(dt^2)/2+A*W*A*(dt^3)/3;
 Pkplusunmoins=Ad*Pkplus*Ad'+Wd;


 %measurement
  bamp=sqrt(((data(index,9)-bxo)/gamabX)^2+((data(index,10)-
byo)/gamabY)^2+((data(index,11)-bzo)/gamabZ)^2);
 amp=sqrt(((data(index,3)-gxo)/gamagX)^2+((data(index,4)-
gyo)/gamagY)^2+((data(index,5)-gzo)/gamagZ)^2);
  %Ym=[(data(i,3)-gxo)/gamagX,(data(i,4)-gyo)/gamagY,(data(i,5)-
gzo)/gamagZ,data(i,6:8)*deg2rad,(data(i,9)-bxo)/gamabX,(data(i,10)-
byo)/gamabY,(data(i,11)-bzo)/gamabZ]';
%  Ym=[gFix,data(i,6:8)*deg2rad,(data(i,9)-bxo)/gamabX,(data(i,10)-
byo)/gamabY,(data(i,11)-bzo)/gamabZ]';
   Ym=[gFix,data(index,6:8)*deg2rad,(data(index,9)-
bxo)/gamabX/bamp,(data(index,10)-byo)/gamabY/bamp,(data(index,11)-
bzo)/gamabZ/bamp]';

%bamp=sqrt(((data(i,9)-bxo)/gamabX)^2+((data(i,10)-
byo)/gamabY)^2+((data(i,11)-bzo)/gamabZ)^2);

  %counting step
 accNorm=sqrt(data(index,3)^2+data(index,4)^2+data(index,5)^2);
 if(flagUp==0)
   flag=(accNorm-accRef)>3.5;

 else
   flag=(accRef-accNorm)>3.5;
 end
 if(flag==1)        %means one step is found
   if(accRef>accNorm)
     flagUp=0;
   else flagUp=1;
   end
   accPeak1=accPeak2;
```

```
                accPeak2=[index,accRef];
               %
meanAcc=[mean(data(accPeak1(1,1):accPeak2(1,1),1)),mean(data(accPeak1(1,1):acc
Peak2(1,1),2)),mean(data(accPeak1(1,1):accPeak2(1,1),3))];
              stride=[stride;sqrt(sqrt(abs(accPeak1(1,1)-
accPeak2(1,1))))/4.1277];%1/2*((accPeak2(1,1)-
accPeak1(1,1))*dt)^2*abs(accPeak2(1,2)-accPeak1(1,2))];
                %stride=[stride;1/2*((accPeak2(1,1)-
accPeak1(1,1))*dt)^2*abs(sqrt(meanAcc(1,1)^2+meanAcc(1,2)^2+meanAcc(1,3)^2))
];

                stepPt=[stepPt;index,accRef];

            else
                if(((flagUp==1)&accRef<accNorm)|((flagUp==0)&accRef>accNorm))
                    if(index-accPeak1(1,1)>20)
                        accPeak1=accPeak2;
                        accPeak2=[index,accRef];
                    end

                    accRef=accNorm;

                end


            end
           %if abs(h-bamp)>5
           %     V(7,7)= 100;
           %     V(8,8)= 100;
           %     V(9,9)= 100;
           %    mNotOk=mNotOk+1
           %  else
           %     V(7,7)= 0.5;
           %     V(8,8)= 0.5;
           %     V(9,9)= 0.5;
           %  end



          q0=Xkplusunmoins(1);
          q1=Xkplusunmoins(2);
          q2=Xkplusunmoins(3);
          q3=Xkplusunmoins(4);
```

```
        C(1:3,1:4)=eval(JdfQ);
        C(7:9,1:4)=eval(JdbQ);

        %update

        K=(Pkplusunmoins*C')/(C*Pkplusunmoins*C'+V);



yk=[eval(f0),eval(f1),eval(f2),Xkplusunmoins(5),Xkplusunmoins(6),Xkplusunmoins(
7),eval(b0),eval(b1),eval(b2)]' ;

        %delta=[delta;(Ym-yk)'];
         Xkplusunplus=Xkplusunmoins+K*(Ym-yk);
         Xe=Xkplusunplus;
         Pkplusunplus=(eye(size(K*C,1))-K*C)*Pkplusunmoins;
         Pkplus=Pkplusunplus;

        q0=Xe(1);
        q1=Xe(2);
        q2=Xe(3);
        q3=Xe(4);
        wx=Xe(5);
        wy=Xe(6);
        wz=Xe(7);


        stateMemory=[stateMemory;Xe'];

    end
    strides=sum(stride);
    walkingLength=walkingLength+sum(stride);

    qtheta=atan2(2*(q0*q3+q1*q2),1-2*(q2^2+q3^2));
    qtheta1=atan2(2*(qr0*qr3+qr1*qr2),1-2*(qr2^2+qr3^2));
        delta=qtheta-qtheta1 ;

    um=-strides*sin(delta)
    vm=strides*cos(delta)
    wm=0;
    xm=xm+um
    ym=ym+vm
    zm=zm+wm;
```

```
    stridesMem=stridesMem+strides

    stride=0;
    posiMem=[posiMem;xm,ym,zm];

end
dlmwrite(strcat(adr,'stateVec.txt'),stateMemory);
 dlmwrite(strcat(adr,'posiMem.txt'),posiMem);
```