University of Vermont
UVM ScholarWorks

Graduate College Dissertations and Theses | Dissertations and Theses

2016

A Drift Eliminated Attitude & Position Estimation Algorithm In 3D

Ruoyu Zhi
University of Vermont

Follow this and additional works at: https://scholarworks.uvm.edu/graddis

 Part of the Electrical and Electronics Commons

Recommended Citation

Zhi, Ruoyu, "A Drift Eliminated Attitude & Position Estimation Algorithm In 3D" (2016). *Graduate College Dissertations and Theses*. 450.
https://scholarworks.uvm.edu/graddis/450

This Thesis is brought to you for free and open access by the Dissertations and Theses at UVM ScholarWorks. It has been accepted for inclusion in Graduate College Dissertations and Theses by an authorized administrator of UVM ScholarWorks. For more information, please contact scholarworks@uvm.edu.

A DRIFT ELIMINATED ATTITUDE & POSITION ESTIMATION ALGORITHM IN 3D

A Thesis Presented

by

Ruoyu Zhi

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Master of Science
Specializing in Electrical Engineering

January, 2016

Defense Date: September 24, 2015
Thesis Examination Committee:

Gagan Mirchandani, Ph.D., Advisor
Eric Hernandez, Ph.D., Chairperson
Kurt E. Oughstun, Ph.D.
Cynthia J. Forehand, Ph.D., Dean of the Graduate College

ABSTRACT

Inertial wearable sensors constitute a booming industry. They are self contained, low powered and highly miniaturized. They allow for remote or self monitoring of health-related parameters. When used to obtain 3-D position, velocity and orientation information, research has shown that it is possible to draw conclusion about issues such as fall risk, Parkinson disease and gait assessment.

A key issues in extracting information from accelerometers and gyroscopes is the fusion of their noisy data to allow accurate assessment of the disease. This, so far, is an unsolved problem. Typically, a Kalman filter or its nonlinear, non-Gaussian version are implemented for estimating attitude – which in turn is critical for position estimation. However, sampling rates and large state vectors required make them unacceptable for the limited-capacity batteries of low-cost wearable sensors.

The low-computation cost complementary filter has recently been re-emerging as the algorithm for attitude estimation. We employ it with a heuristic drift elimination method that is shown to remove, almost entirely, the drift caused by the gyroscope and hence generate a fairly accurate attitude and drift-eliminated position estimate.

Inertial sensor data is obtained from the 10-axis SP-10C sensor, attached to a wearable insole that is inserted in the shoe. Data is obtained from walking in a structured indoor environment in Votey Hall.

Keywords- *enhanced heuristic drift elimination, indoor pedestrian tracking, zero-velocity update, explicit complementary filter*

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Gagan Mirchandani for his excellent guidance, not only as an advisor but also a mentor of life. Without his numerous time and energy, I cannot build this paper.

I would also like to thank Kailash Mallikarjun from Sensorplex Inc. He has provided great amount of valuable advice and resource for my research.

I also want to thank all the committee members for their inspiring insights, suggestions and patience.

Finally, I want to thank my parents and my girlfriend for their love, understanding and consistent support and encouragement.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF TABLES vi

LIST OF FIGURES vii

CHAPTER 1: INTRODUCTION AND BACKGROUND 1

1.1 Motivation 1

1.2 MEMS IMU 2

1.2.1. What is MEMS IMU? 4

1.2.2 Drifts and Errors of MEMS IMU 7

1.3 AHRS algorithm 11

1.3.1 Kalman Filter 12

1.3.2 Complementary Filters 16

1.4 Thesis Structure 18

CHAPTER 2: REPRESENTATION OF ROTATION 20

2.1 Rotations 20

2.2 Quaternions 24

CHAPTER 3: EXPLICIT COMPLEMENTARY FILTER 29

3.1 Sensor Noise Modelling 30

3.1.1 Our Sensor 30

3.1.2 Gyroscope Noise Model 31

3.1.3 Accelerometer Noise Model 32
3.2 Explicit Complementary Filter 32
3.3 Results 35
CHAPTER 4: ZERO VELOCITY UPDATE 41
4.1 Pedestrian Dead Reckoning 41
4.2 Zero Velocity Update (ZUPT) 41
4.3 Implementation 43
4.4 Results 44
CHAPTER 5: ENHANCED HEURISTIC DRIFT ELIMINATION 51
5.1 Preliminary Results 51
5.2 Heuristic Drift Elimination 57
5.3 Enhanced Heuristic Drift Elimination in 3D 57
5.3.1 Pedestrian Motion Detection 58
5.3.2 Dominant Direction Calculation 59
5.3.3 Position Update 62
5.3.4 Block Diagram 63
5.4 Results 65
CHAPTER 6: TOTAL SYSTEM FRAMEWORK 74
6.1 Top Level Implementation 74
6.2 UML diagram 76
CHAPTER 7: CONSLUSION 79

7.1 Contributions 79

7.2 Limitations 80

7.3 Future Research 80

References 82

Appendix 85

LIST OF TABLES

Table	Page
Table 1 Error of preliminary system	56
Table 2 Performance of EHDE in 2D walk	68
Table 3 Performance of EHDE in 3D walk	72

LIST OF FIGURES

Figure Page

Figure 1 Grades of MEMS gyroscopes 3
Figure 2 Inertial motion unit SP-10C size 5
Figure 3: Insole with SP-10C attached 6
Figure 4 Experiment system setup 7
Figure 5 Design of MEMS accelerometer 8
Figure 6 Design of MEMS gyroscopes 10
Figure 7 Update and predict functions of Kalman filter 12
Figure 8 Linear Kalman filter based position tracking algorithm 14
Figure 9 Flow chart of Carl Fischer's Kalman filter based implementation 15
Figure 10 1-D complementary filter 17
Figure 11 Reference frames considered 21
Figure 12 Euler angles 21
Figure 13 Axis and angle 23
Figure 14 Quaternion representation 26
Figure 15 System chart of SP-10C 31
Figure 16 Block diagram of explicit complementary filter 33
Figure 17 Gyroscope data 35
Figure 18 Integral of gyroscope data on X axis 36
Figure 19 Integral of gyroscope data on Y axis 37
Figure 20 Integral of gyroscope data on Z axis 37
Figure 21 Quaternion 38
Figure 22 X coordinate 39
Figure 23 Y coordinate 39
Figure 24 Z coordinate 40
Figure 25 Block diagram of ZUPT 43
Figure 26 Raw accelerometer data 45
Figure 27 Filtered acceleration magnitude and stationary periods 46
Figure 28 Velocity without ZUPT 47

Figure 29 Position without ZUPT 48
Figure 30 Velocity with ZUPT 49
Figure 31 Position with ZUPT 50
Figure 32 Preliminary Top Level Implementation 51
Figure 33 Plot of position data-Votey 52
Figure 34 Trajectory of Votey top view 53
Figure 35 Trajectory of Votey side view 53
Figure 36 Plot of position data-stairs 54
Figure 37 Trajectory of stairs side view 55
Figure 38 Trajectory of stairs top view 56
Figure 39 Position of previous 5 steps and current step 59
Figure 40 Block diagram of EDHE 64
Figure 41 Plot of position data with EHDE for Votey walk 65
Figure 42 Trajectory of Votey walk top view 66
Figure 43 Trajectory of Votey walk side view 66
Figure 44 Heading difference to four dominant directions 67
Figure 45 Plot of position data with EHDE for stairs 69
Figure 46 Trajectory of stairs cornered view 70
Figure 47 Trajectory of stairs top view 71
Figure 48 Trajectory of stairs side view 71
Figure 49 Block diagram of top level algorithm 74
Figure 50 Hierarchical Diagram 76
Figure 51 UML diagram 77

CHAPTER 1: INTRODUCTION AND BACKGROUND

1.1 Motivation

Human localization is very valuable information required across many applications in the modern smart environment. The primary choice for almost all outdoor navigation need for either vehicles or persons is GPS, either a standalone GPS module like Garmin on vehicles or a mobile application installed on a smart cellphone. However, GPS is not available inside buildings because GPS signal is too weak to penetrate through building structures. Moreover, GPS localization accuracy that ranges within a few meters is not efficient for indoor environments. As to indoor localization, the industrialized practice is using Wi-Fi. But Wi-Fi localization has the drawback of poor accuracy in terms of indoor environment. The accuracy of Wi-Fi localization ranges within several meters, and depends totally on number of APs (Access Points) and Wi-Fi signal intensity which is vulnerable to varying building structures [1]. Wi-Fi localization also suffers from long convergence time to converge on a stable spot. Typical convergence time of Wi-Fi localization ranges from a few seconds to more than 20 seconds and depends on both the Wi-Fi signal and localization algorithm [2-3]. To overcome the problems of unavailability and unreliability of the conventional localization methods, one possible solution for indoor personnel positioning is based on Inertial Motion Units (IMUs). IMUs are self-contained, low power, highly miniaturized sensors that collect intrinsic motion information without any external references. IMUs comprise a 3-axes accelerometer and a 3-axes gyroscope, and sometimes a 3-axes magnetometer. For our experiment, we use the MPU9250, a 9 Degrees of Freedom (9DoF) IMU developed by InvenSense Inc. to track both the position and foot attitude of patients or pedestrians.

Wireless tracking in hospital is showing its necessity in recent years. With position tracking, we can keep patients and caregivers connected. With foot attitude estimation, we can monitor and record the walking patterns of patients who suffer from certain diseases that affect walking, such as Alzheimer's, arthritis or club foot. Doctors can also use these measurements to make further treatment plans accordingly. This new application area would provide enormous benefits to patients in many situations including remote healthcare delivery to rural communities, real-time and continuous monitoring during rehabilitation after prosthetic joint replacement, prognosis of patients suffering with traumatic stress, neurocognitive and memory-related diseases.

In order to obtain reliable and precise estimations of location and attitude, a light-weighted yet equally efficient algorithm is desired so as to reduce errors and drift from inertial sensors and be able to run on a handheld device.

1.2 MEMS IMU

Inertial motion sensors typically contain three orthogonal accelerometers and three orthogonal gyroscopes, and sometimes three orthogonal magnetometer, measuring angular velocity, acceleration and magnetic field respectively. By processing signals from these devices it is possible to track the position and orientation of a device. Inertial motion sensors are the fundamental tools of pedestrian tracking. Their precision and reliability directly affects the quality of output.

Application grade	Bias stability	Relative accuracy (*)	Main application
CONSUMER AND AUTOMOTIVE GYRO			
Consumer	10 °/s	3%	Motion interface
Automotive	1 °/s	0.3%	ESP
HIGH PERFORMANCE GYRO			
Low-end Tactical also called Industrial	10°/h (Earth rate)	10 ppm	Amunitions & rockets guidance
Tactical	1°/h	1 ppm	Platform stabilization
Short-term Navigation	0.1°/h	100 ppb	Missile navigation
Navigation	0.01°/h	10 ppb	Aeronautics navigation
Strategic	0.001°/h	1 ppb	Submarine navigation



Range of current MEMS gyros

Used with permission fromTronics

Figure 1 Grades of MEMS gyroscopes

In the development of the Hubble Scope by NASA, a set of gas-bearing gyroscopes were used. It was claimed as the most accurate gyroscope in the world by NASA. But while better performance comes with more sophisticated systems, each gas-bearing gyro is as big as a suitcase, not factoring in the external power source.

In aviation and naval navigation, especially submarine navigation, a comparatively very precise and expensive ring laser gyroscope with better than 0.001 deg/hour bias uncertainty is used to capture the orientation data, which gives considerably reliable result. But like the gas-bearing gyroscope, a RLG (Ring Laser Gyroscope) has a diameter of over 50 cm.

In the case of tracking pedestrians in a hospital, a low-cost sensor with limited size and power supply is required. Thus MEMS IMU is selected as the best suited sensor module for our purposes.

1.2.1. What is MEMS IMU?

An inertial measurement unit (IMU) is an electronic device that measures and reports a craft's velocity and orientation using a combination of accelerometers and gyroscopes and sometimes also magnetometers. IMUs are typically used to maneuver aircraft, including unmanned aerial vehicles (UAVs), among many others. Also spacecraft, including satellites and landers.

MEMS, which stands for Micro-Electro-Mechanical Systems, is the technology of very small devices; it merges at the nano-scale into nanoelectromechanical systems (NEMS) and nanotechnology. MEMS are made up of components between 1 to 100 micrometers in size (i.e. 0.001 to 0.1 mm), and MEMS devices generally range in size from 20 micrometers to a millimeter (i.e. 0.02 to 1.0 mm). Only with MEMS technology can we make the IMU small enough to be attached to the foot of pedestrians.

Figure 2 shows the size of the inertial module Sensoplex SP-10C used for collecting motion data in this project compared to a quarter dollar coin. MPU 9250 developed by InvenSense Inc. is used as the motion sensor, which is a full 9DoF inertial sensor. It has accelerometer, gyroscope and compass on all three axes. It is integrated into the SP-10C module which converts the analogue output from MPU 9250 to digital, and provides various functions such as Bluetooth connection, data streaming and logging, and configurations available at a GUI software, the SP Monitor. The SP-10C is powered by a 5 volt battery which is of the same size as SP-10C. The SP-10C along with the battery are packed into a 3D-printed box just able to contain the sensor set. Finally, for our purpose, the 3D-printed box is super-glued onto an insole which has a rubber

bottom so that the insole doesn't slip back and forth in the shoe. Figure 3 shows what the system looks like in practical use.

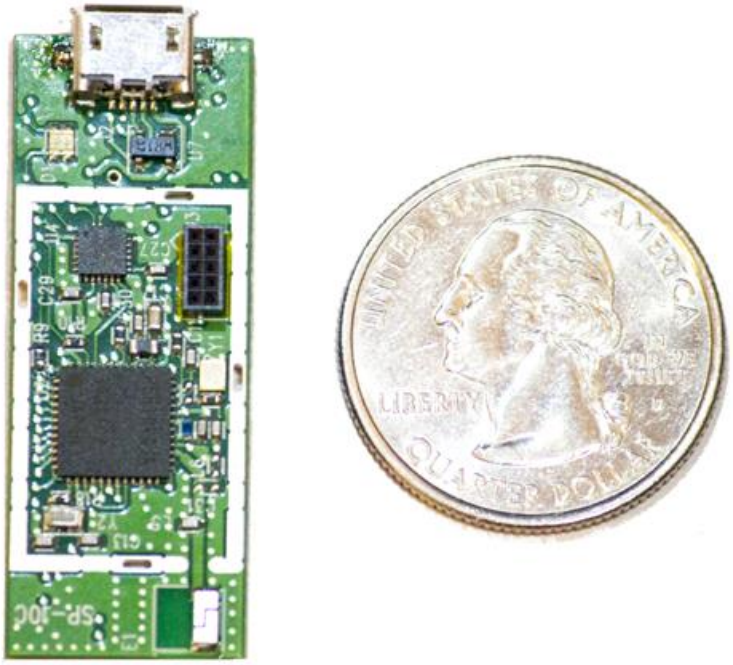


Figure 2 Inertial motion unit SP-10C size

Magnetic field in indoor environment is considerably complex and unpredictable. In hospitals, facilities such as MRIs strongly distort the magnetic field. Therefore the magnetometer was disabled in our project. Thus the position estimation algorithm is based on 6 Degrees of Freedom, without the aid of a compass.



Figure 3: Insole with SP-10C attached

Figure 4 shows the experiment system setup. The insole with the inertial sensor is inserted in the shoe of a pedestrian. The inertial sensor serves as the BlueTooth slave while another sensor serves as the BlueTooth master, which is connected to the computer. The master sends commands such as start logging data, stop logging data or resetting sensors etc. via BlueTooth Low Energy (BLE) technology. The slave logs data in its RAM while the pedestrian is walking and sends back the data to BlueTooth master via BLE afterwards. The data is then stored as a csv file and ready to be fed into the MATLAB program.

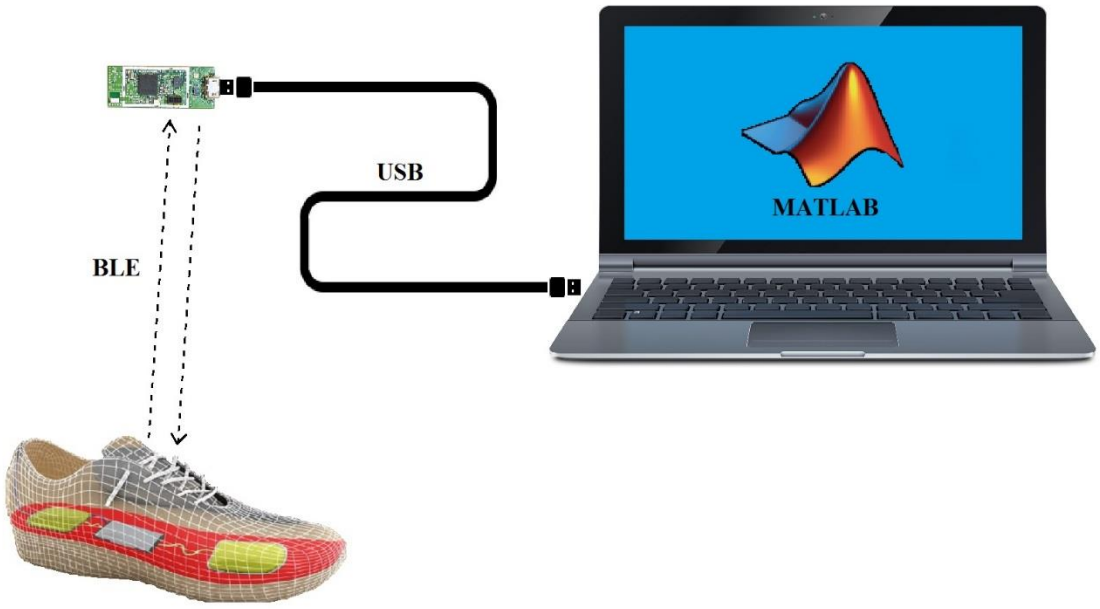


Figure 4 Experiment system setup

1.2.2 Drifts and Errors of MEMS IMU

MEMS devices are extremely microscopic devices and are vulnerable to many factors. Their performance is generally considered poor in accuracy for certain applications because of drifting, which is the nature of inertial sensors and which is more severe in MEMS. In order to understand the errors and drift, an understanding of the design of MEMS IMU is necessary.

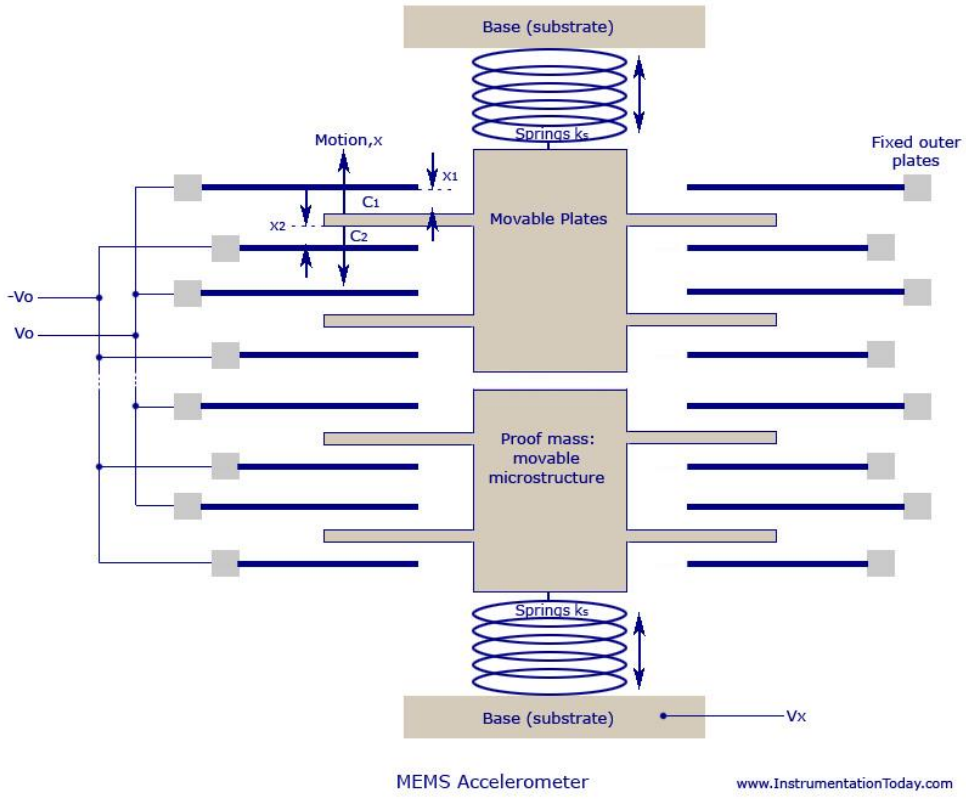


Figure 5 Design of MEMS accelerometer

The MEMS accelerometer can be considered as a mass-spring system. Figure 5 shows an illustration of a typical MEMS accelerometer on one axis. It is composed of movable proof mass with plates that is attached through a mechanical suspension system to a reference frame. When the object has an acceleration in the direction to which the spring is fixed, the plate moves and the spacing between the comb structure and adjacent outer fixed plates changes. By measuring the capacitance difference on each side of the plate, the deflection of the proof mass is calculated [4].

$$C_1 = \varepsilon_A \frac{1}{x_1} = \varepsilon_A \frac{1}{d + x} = C_0 - \Delta C$$

$$C_2 = \varepsilon_A \frac{1}{x_2} = \varepsilon_A \frac{1}{d - x} = C_0 + \Delta C$$

So,

$$C_2 - C_1 = 2\Delta C = 2\varepsilon_A \frac{x}{d^2 - x^2}$$

Measuring ΔC, one finds the displacement x by solving the nonlinear algebraic equation:

$$\Delta C x^2 + \varepsilon_A x - \Delta C d^2 = 0$$

We get,

$$x \approx \frac{d^2}{\varepsilon_A}\Delta C = d\frac{\Delta C}{C_0}$$

Because it holds true that,

$$(V_x + V_0)C_1 + (V_x - V_0)C_2 = 0$$

Therefore we get voltage output V_x

$$V_x = V_0 \frac{C_2 - C_1}{C_2 + C_1} = \frac{x}{d}V_0$$

and thus we get the acceleration, a

$$a = \frac{k_s}{m}x = \frac{k_s d}{m V_0}V_x$$

We notice that in fact the accelerometer does not measure linear acceleration. It picks up forces applied on the proof mass and measure the displacement of the mass to calculate acceleration. The force of gravity is always picked up by the accelerometer and is taken as acceleration. An accurate attitude of the sensor has to be estimated in order to subtract the gravity factor on all three axes so as to obtain an estimate of linear acceleration. Otherwise small errors in the measurement of acceleration are integrated into progressively larger errors in velocity, which are compounded into still greater errors in position. Since the new position is calculated from the previous calculated position

and the measured acceleration and angular velocity, these errors accumulate roughly proportionally to the time since the initial position was input [5]. This is known as the notorious integration drift problem of inertial navigation system, which is greatly reduced using the approach proposed in this thesis.

Therefore, the measurement of an accelerometer can be modeled as:

$$R(t) = \hat{\boldsymbol{v}} - g + s_a(t) + n_a(t)$$

where $\hat{\boldsymbol{v}}$ represents the instantaneous linear acceleration, g represents gravitational acceleration, $s_a(t)$ and $n_a(t)$ stand for accelerometer bias and noise respectively.

MEMS gyroscopes have different designs than the MEMS accelerometer. The underlying physical principle is that a vibrating object tends to continue vibrating in the original plane as its support rotates. This type of device is also known as a Coriolis vibratory gyro because when the plane of vibration is rotated, the response detected by the transducer results from the Coriolis term in its equations of motion ("Coriolis force"). A microgram of a MEMS vibratory gyroscope is as shown in Figure 6 below.

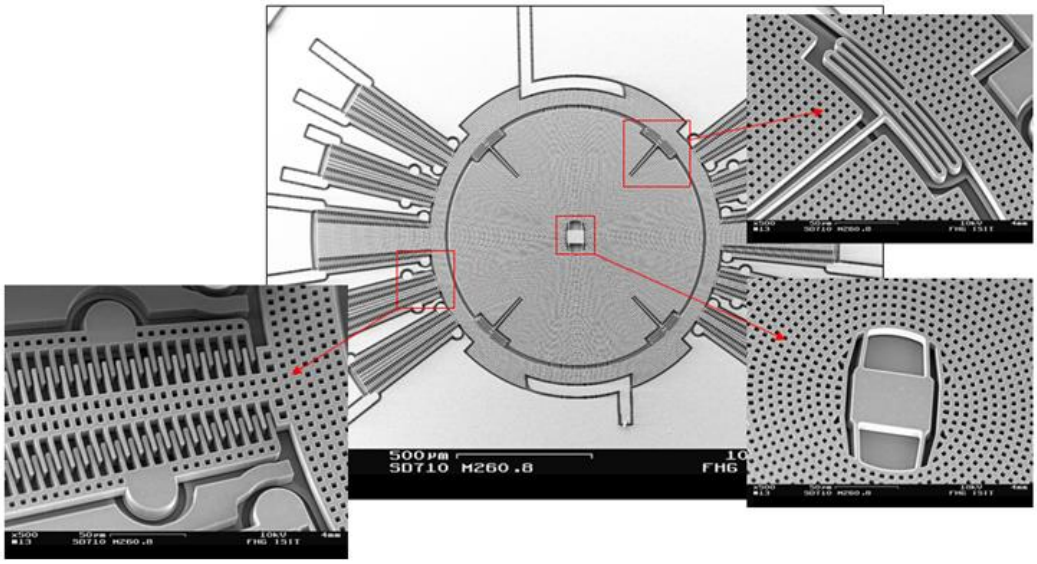


Figure 6 Design of MEMS gyroscopes

Like the accelerometer, gyroscope also has the integration drift problem-small errors in the measurements are dramatically amplified by integration over time. However, gyroscopes suffer from another kind of drift. Gyroscopes are sensitive to changes in temperature, which brings about slow-changing deviations just like integration drift did for the accelerometer [6].

Therefore, the measurement of a MEMS gyroscope can be modeled as:

$$\Omega_g^b(t) = \Omega(t) + s_g(t) + n_g(t)$$

where $\Omega_g^b(t)$ is the gyroscope reading in sensor frame, $\Omega(t)$ is the actual angular velocity in global frame, $s_g(t)$ and $n_g(t)$ represents gyroscope bias and high-frequency noise in gyroscope respectively.

1.3 AHRS algorithm

MEMS IMUs are vulnerable to various kinds of noise and errors. Accelerometers are extremely sensitive to attitude changing and impact forces while gyroscopes are sensitive to temperature changes and suffer from a slow-changing bias. To summarize, accelerometers have poor dynamic features and gyroscopes have poor static features. Therefore an AHRS (Attitude and Heading Reference System) algorithm is needed to fuse the data from different sensors to overcome the drawbacks of each of them and take the most reliable part from them respectively to give a best prediction of the actual status of the sensor. AHRS algorithm is the foundation of position estimation for the reason that gravity must be removed completely from the accelerometer to get linear

acceleration. Only then can the integration be done without being concerned about the drift.

There are two main categories of AHRS algorithms. One is based on the Kalman filter and the other is based on the so-called complementary filter.

1.3.1 Kalman Filter

The Kalman filter is a powerful algorithm that takes a series of measurements that contain statistical noise and uncertainties over time, and produces an estimate of the status of the object that is more reliable than a single measurement.

The Kalman filter consists of a two-step loop: updating and predicting as is illustrated in Figure 7. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. In the update step, the Kalman filter takes the most recent set of measurements, compares it with the prediction, and updates the estimate with a weighted average coefficient K called Kalman gain.

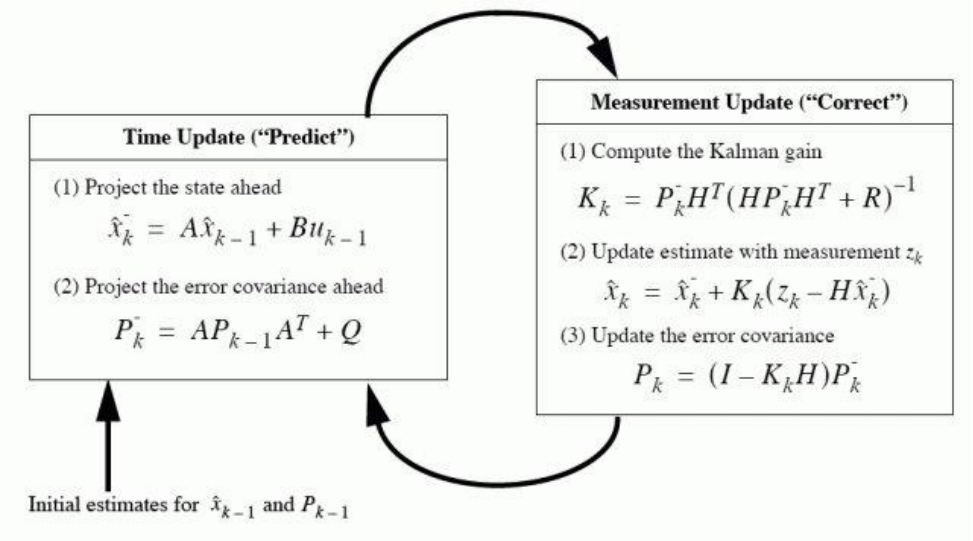


Figure 7 Update and predict functions of Kalman filter

In the 1-D situation, the discrete-time state space model of an inertial system is

$$\theta_k = \theta_{k-1} + \omega_k \Delta t$$

$$z_k = a_k$$

where θ_k is the attitude estimation at time k, ω_k is the rotation rate output by gyroscope, a_k is the attitude angle calculated from accelerometer measurements and z_k is the measurement at time k.

So the state vectors become

$$x = \theta, \quad u = \omega$$

and the matrices

$$A = 1, \quad B = \Delta t, \quad H = 1, \quad K = K_0$$

So the Kalman equations read

$$\widehat{\boldsymbol{\theta}}_k^- = \widehat{\boldsymbol{\theta}}_{k-1} + \omega_k \Delta t$$

$$\widehat{\boldsymbol{\theta}}_k = (1 - K_0)\widehat{\boldsymbol{\theta}}_k^- + K_0 a_k$$

These equations can be reformatted as

$$\widehat{\boldsymbol{\theta}}_k = \alpha \widehat{\boldsymbol{\theta}}_{k-1} + (1 - \alpha) a_k + \alpha \omega_k \Delta t$$

where

$$\alpha = 1 - K_0$$

A number of AHRS algorithms are developed based on the Kalman filter. Pedro Neto et al. proposed a position estimation algorithm based on the Kalman filter to correct the yaw [7]. Figure 8 shows the block diagram of that implementation.

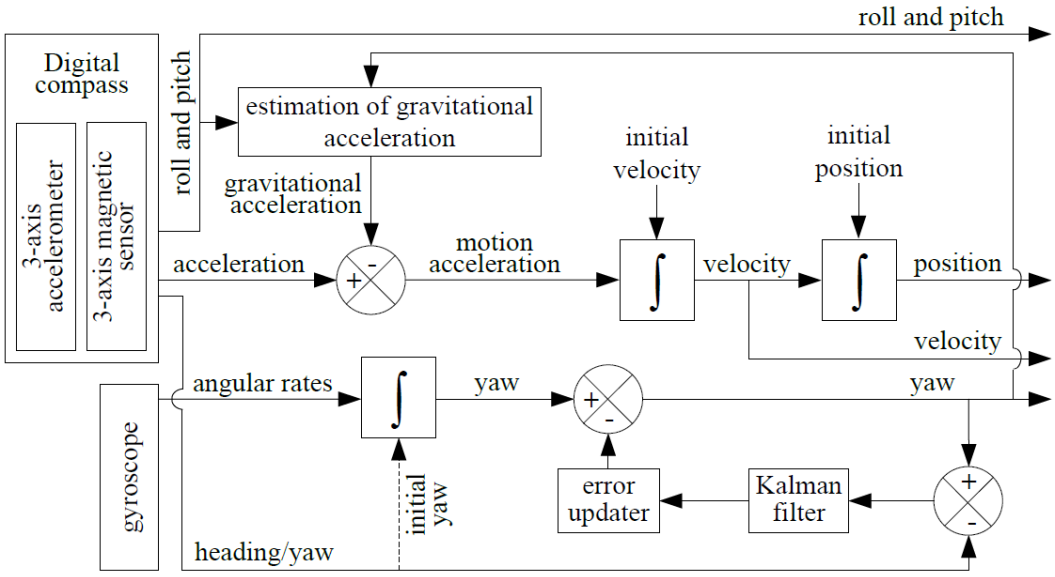


Figure 8 Linear Kalman filter based position tracking algorithm

Carl Fischer et al. also proposed pedestrian tracking algorithm based on Kalman filter [8]. Flow chart is shown in Figure 9.

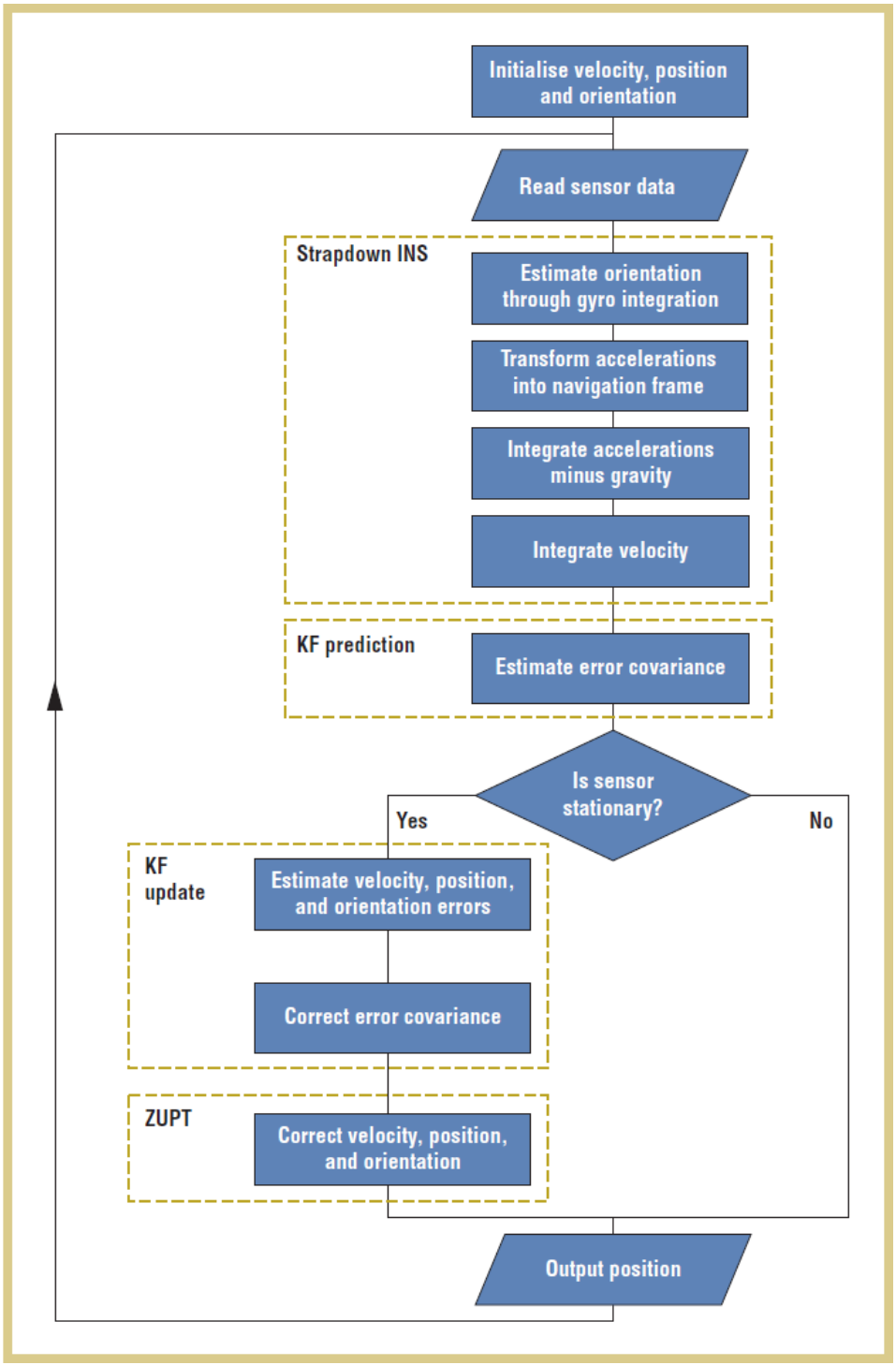


Figure 9 Flow chart of Carl Fischer's Kalman filter based implementation

But a linear model of the Kalman filter does not give a satisfactory estimation of position because as is indicated in Section 1.2.2 drifts and errors of MEMS IMU, errors

and noise in MEMS IMU are non-linear and non-Gaussian respectively. And the linear Kalman filter with its Gaussian noise assumption is not saticfactory in this environment.

Therefore non-linear Kalman filter based AHRS algorithms have been proposed and have many variations and implementations. V. Bistrovs and A. Kluga proposed a GPS aided extended Kalman filter AHRS algorithm [9]. Gabriele Ligorio et al. implemented the extended Kalman filter in a camera vision aided system [10]. Algorithms based on the Unscented Kalman filter was also proposed for aided INS [11]. These algorithms give comparatively good results, however, they have a number of disadvantages. They can be complicated to implement which is reflcted by the numerous solutions [22]. The linear regression iterations, which are fundamental to the Kalman process, demand sampling rates between 512 Hz [34] and 30 kHz [35] to be used for a human motion caption application, and that far exceeds the low-cost MEMS sensor used in this project which allows up to 180Hz for 6 DoF and 130 Hz for 9 DoF according to practice. Also, the nonlinear Kaman Filters are computationally expensive to execute for hand held devices.

1.3.2 Complementary Filters

Since Kalman filter's time complexity is too heavy a burden for mobile devices, algorithms with much lower computational complexity and almost equally efficiency are getting more acceptance, such as the complementary filters proposed by Robert Mahony [12].

As is discussed above, accelerometers are sensitive to attitude changing and impact forces while gyroscopes are sensitive to temperature changes and suffer from

slow-changing bias. That means accelerometers have poor dynamic feature while gyros have poor static feature. The complementary filter takes this fact and make good use of each part. It fuses the accelerometer data and gyroscope data by first passing the accelerometer data through a 1st-order low pass filter, and then the gyroscope data through a 1st-order high pass filter, as is shown in Figure 10. Thus the complementary filter extracts the most reliable part of each sensor. Then a weighted average is taken from these two measurements.

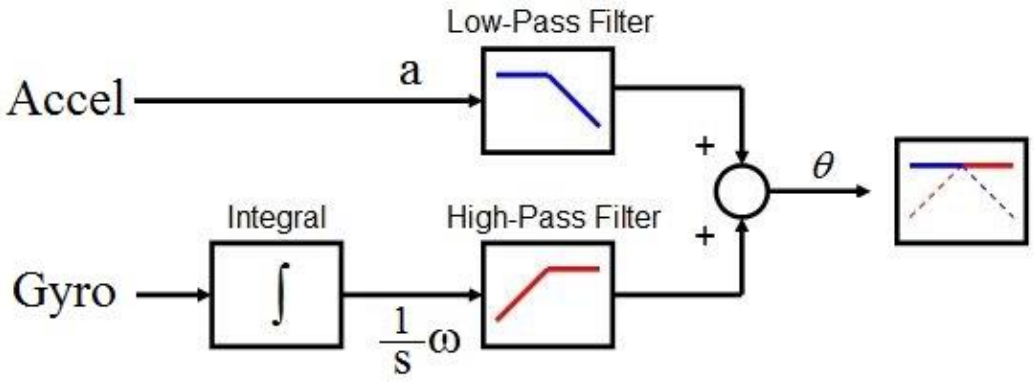


Figure 10 1-D complementary filter

In the 1-D situation, the transfer function of complementary filter is:

$$\theta = \frac{1}{1+Ts}a + \frac{Ts}{1+Ts}\frac{1}{s}\omega = \frac{a+T\omega}{1+Ts}$$

where T determines the filter cut-off frequency, a represents the angle calculated from accelerometer outputs and ω represents angular rate from gyroscope readings. Therefore $\frac{1}{s}\omega$ represents the angle calculated from gyroscope. That means the complementary filter uses a cut-off frequency to make low-frequency estimation of attitude from accelerometer and high-frequency estimation from gyroscope, in other words, complementary filter fuses the long-term estimation from accelerometer and the short-

term estimation from gyroscope. Thus accelerometer and gyroscope "complement" each other.

Using backward difference yields

$$1 + Ts = \left(1 + \frac{T}{\Delta t}\right) - \frac{T}{\Delta t} z^{-1}$$

Thus the final result is

$$\theta_k = \alpha(\theta_{k-1} + \omega_k \Delta t) + (1 - \alpha) a_k$$

where

$$\alpha = \frac{T}{\Delta t} / \left(1 + \frac{T}{\Delta t}\right)$$

So the result is reformulated as

$$\theta_k = \alpha\theta_{k-1} + (1 - \alpha) a_k + \alpha\omega_k \Delta t$$

Notice that the transfer function of the complementary filter is identical to that of the Kalman Filter [40], in the form of

$$\theta_k = \alpha\theta_{k-1} + (1 - \alpha) a_k + \alpha\omega_k \Delta t$$

That is to say that the angle is first advanced by integrating the rotation rate to give an updated angle and then filtered with a_k to give an improved angle. Thus it can be concluded that in the attitude estimation scenario, a complementary filter is superior to a Kalman Filter because it leads to identical update function. However, the complementary filter cost much less computation.

1.4 Thesis Structure

The structure arrangement of this thesis is as follows:

Chapter 1 explains the problem and introduces background knowledge needed. Also basic noise modelling is given to demonstrate characteristics of MEMS inertial sensors.

Chapter 2 introduces ways of representing rotation and explains the advantages of quaternion representation. It also gives basic calculation rules for quaternions.

Chapter 3 discusses the importance of an AHRS algorithm for a pedestrian tracking system and describes the implementation of explicit complementary filters.

Chapter 4 introduces zero velocity detection and update techniques, and evaluates its performance in the whole system.

Chapter 5 proposes the Enhanced Heuristic Drift Elimination algorithm (EHDE), explains the algorithm and demonstrates the ability of EHDE.

Chapter 6 reviews the techniques explained in previous chapters. It also explaines the mechanism of the algorithm from the top level.

Chapter 7 concludes the thesis and suggests possible future work regarding magnetically aided system and compressive sensing possibilities.

CHAPTER 2: REPRESENTATION OF ROTATION

In order to get an accurate foot attitude of a patient, and also for the purpose of subtracting gravity factors from the three axes of accelerometers to obtain linear acceleration, an accurate estimation of rotation is required. The AHRS algorithm is the core of the inertial navigation system. Therefore an appropriate representation of rotation that the AHRS algorithm is built on is desired.

2.1 Rotations

In the scenario of pedestrian tracking, there are three coordinate systems involved: the foot frame in which foot rotates and acceleration is described, the sensor frame in which the motion of sensor is described, and the global frame, also known as the earth frame, where linear acceleration and rate of rotation are presented. In our case, since the sensor is super-glued to the insole and the insole does not slip in the shoe, the sensor and the foot frame are the same. Hence the foot frame is ignored. The sensor frame and global frame are the two coordinates systems where sensor data is represented. Because all the data is picked up by the sensor, the data is represented in terms of the sensor frame. However for pedestrian tracking we care about position and that is referenced with regard to the earth frame. Accordingly, we have to rotate the data into earth frame. Also the AHRS algorithm depends a lot on which representation of rotation is chosen. That is why the representation of rotation is important.

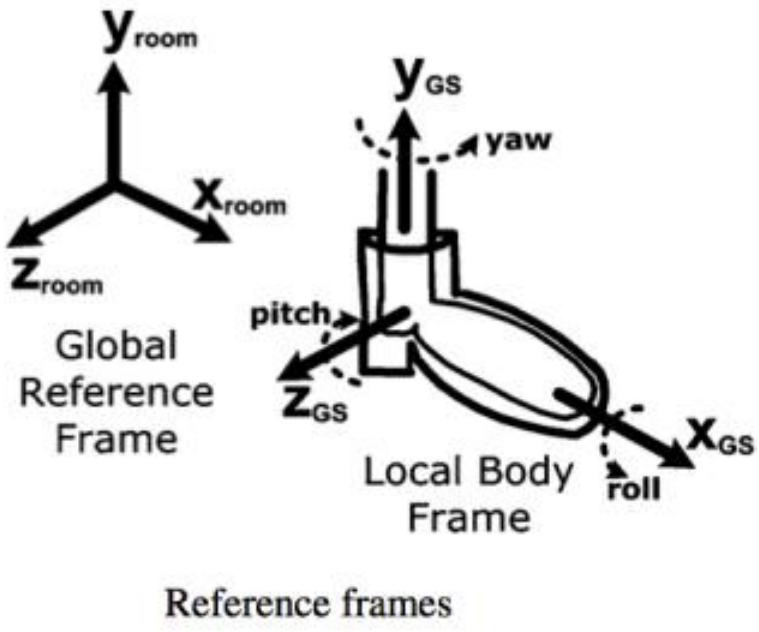


Figure 11 Reference frames considered

Figure 11 illustrates the two coordinate systems involved in this problem. The upperleft coordinate system denotes the earth frame. The coordinate system embedded in the shoe denotes the sensor frame as well as the foot fame. Rotation around X, Y and Z axis is called roll, pitch and yaw respectively.

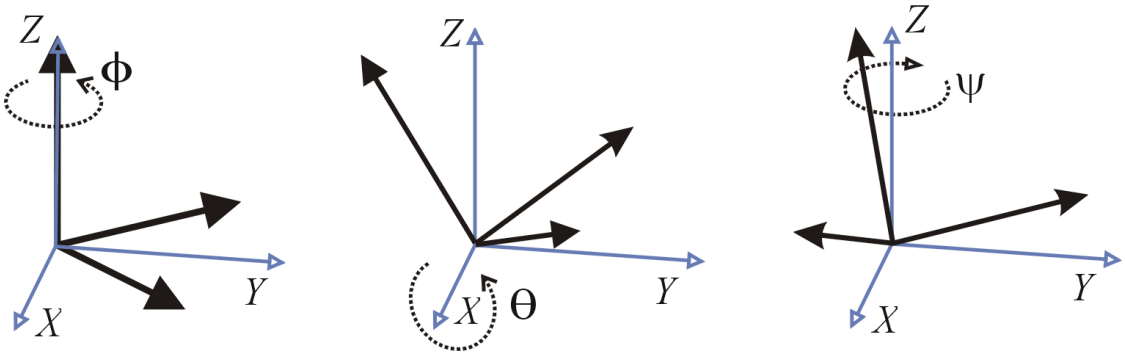


Figure 12 Euler angles

Euler Angles is the most intuitive way of representing orientation in Special Orthogonal Group SO(3). Euler angles are three angles corresponding to a sequence of three elemental rotations respectively i.e. rotations about the axes of a coordinate system. There are several ways to initialize a set of Euler angles. Proper Euler angles includes Z-X-Z (angle of roation around Z axis, angle of rotation around X axis and angle of roation around Z axis again), X-Y-X, Y-Z-Y, Z-Y-Z, X-Z-X, Y-X-Y. There are also the Tait–Bryan angles that includes X-Y-Z, Y-Z-X, Z-X-Y, X-Z-Y, Z-Y-X, Y-X-Z. For example, Figure 12 shows the Z-X-Z situation, ϕ, θ, and ψ are the angles rotated around Z axis, X axis and Z axis again respectively. Its Euler angles are represented as

$$r_{zyx} = (\phi \quad \theta \quad \psi)^T$$

If we convert that into the roation matrix, an arbitrary rotation in SO(3) looks like:

$$R_{zxz}(\phi, \theta, \psi) = R_z(\psi)R_x(\theta)R_z(\phi)$$

$$= \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$$= \begin{pmatrix} \cos\psi\cos\theta & -\sin\psi & \cos\psi\sin\theta \\ \sin\psi\cos\theta & \cos\psi & \sin\psi\sin\theta \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$$= \begin{pmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix}$$

Euler Angles suffer from a problem called the Gimbal Lock, which is the loss of one degree of freedom when two of the three axes aligned. That means, all three gimbals are still able to rotate freely about their own axes. But because two of the three gimbals axes were aligned, there is no gimbal available to accommodate rotation along one axis

of the earth frame. Although gimbal lock can be addressed through the accumulative matrix transformation method, the Euler Angles representation is computationally expensive.

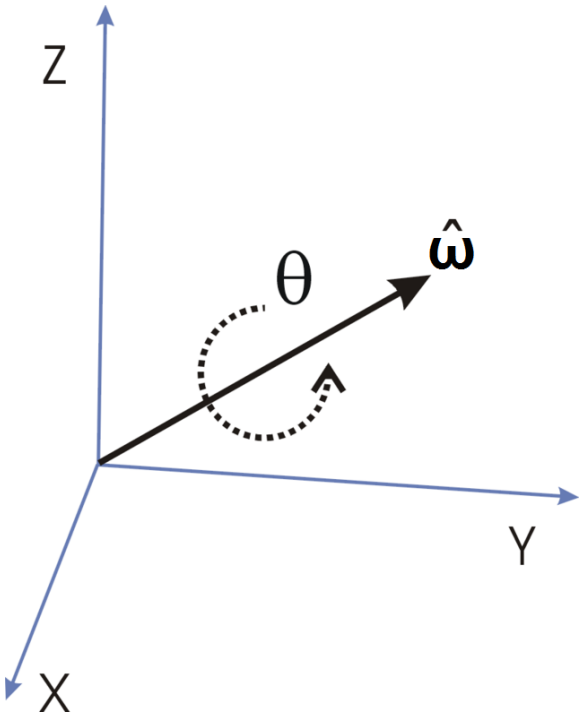


Figure 13 Axis and angle

Axis and Angle is also a common representation in SO(3), as is illustrated in Figure 13. According to the Euler's rotation theorem, any rotation $R \in SO(3)$, is equivalent to a rotation about a fixed axis, $\hat{\omega} \in R^3$, through an angle $\theta \in [0,2\pi)$.

Thus,

$$Axis: \quad \hat{\boldsymbol{\omega}} = (x \quad y \quad z) \qquad Angle: \quad \theta$$

According to the Rodrigues' rotation formula [37], the rotated vector is:

$$\hat{\boldsymbol{e}}_{rot} = (\cos\theta)\hat{\boldsymbol{e}} + (\sin\theta)(\hat{\boldsymbol{\omega}} \times \hat{\boldsymbol{e}}) + (1 - \cos\theta)(\hat{\boldsymbol{\omega}} \cdot \hat{\boldsymbol{e}})\hat{\boldsymbol{\omega}}$$

But this suffers from a distance preserving problem. To address this, we convert it into the rotation matrix form:

$$M(\hat{\boldsymbol{e}}, \theta)$$

$$= \begin{bmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta)xy - (\sin\theta)z & (1-\cos\theta)xz + (\sin\theta)y \\ (1-\cos\theta)xy + (\sin\theta)z & \cos\theta + (1-\cos\theta)y^2 & (1-\cos\theta)yz - (\sin\theta)x \\ (1-\cos\theta)xz - (\sin\theta)y & (1-\cos\theta)yz + (\sin\theta)x & \cos\theta + (1-\cos\theta)z^2 \end{bmatrix}$$

Nevertheless, these 3x3 rotation matrices are still too computationally expensive to implement. What is more, over a long series of computations, numerical errors can cause these matrices to be no longer orthogonal, and need to be orthogonalized from time to time.

2.2 Quaternions

A quaternion is a four-dimensional complex number [38] invented by Hamilton in 1843 with the form

$$\widehat{\boldsymbol{Q}} = q_0 + iq_1 + jq_2 + kq_3$$

i, j and k in the definition of quaternion is very similar to i the in 2-D complex plane. They have the properties:

$$ii = jj = kk = ijk = -1$$

$$ij = -ji = k$$

$$jk = -kj = i$$

$$ki = -ik = j$$

Multiplication of quaternions Q and P, denoted by $\otimes$, can be determined using the Hamilton rule, defined as:

$$\widehat{\boldsymbol{Q}} \otimes \widehat{\boldsymbol{P}} = (q_0 + iq_1 + jq_2 + kq_3)(p_0 + ip_1 + jp_2 + kp_3)$$

$$= (p_0 q_0 - \boldsymbol{p} \cdot \boldsymbol{q}, p_0 \boldsymbol{q} + q_0 \boldsymbol{p} + \boldsymbol{p} \times \boldsymbol{q})$$

where $\boldsymbol{q} = (q_1 \quad q_2 \quad q_3)$ and $\boldsymbol{p} = (p_1 \quad p_2 \quad p_3)$, $\boldsymbol{p} \cdot \boldsymbol{q}$ denotes the dot product of $\boldsymbol{p}$ and $\boldsymbol{q}$, $\boldsymbol{p} \times \boldsymbol{q}$ denotes the cross product of $\boldsymbol{p}$ and $\boldsymbol{q}$.

The complex conjugate of a quaternion is defined as:

$$\widehat{\boldsymbol{Q}}^* = (q_0 \quad q_1 \quad q_2 \quad q_3)^* = (q_0 \quad -q_1 \quad -q_2 \quad -q_3)$$

Consider the unit quaternions with definition:

$$norm(\widehat{\boldsymbol{Q}}) = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$$

The identity quaternion is

$$\widehat{\boldsymbol{Q}} = (1 \quad \boldsymbol{0})$$

Since

$$\widehat{\boldsymbol{Q}} \otimes \widehat{\boldsymbol{Q}}^* = (q_0, \boldsymbol{q})(q_0, -\boldsymbol{q}) = (q_0 q_0 - \boldsymbol{q}^2, q_0\boldsymbol{q} - q_0\boldsymbol{q} + \boldsymbol{q} \times \boldsymbol{q}) = (1, \boldsymbol{0})$$

Therefore the inverse of a unit quaternion is:

$$\widehat{\boldsymbol{Q}}^{-1} = \widehat{\boldsymbol{Q}}^* = (q_0 \quad -q_1 \quad -q_2 \quad -q_3)$$

Or, we can say, for a non-unit quaternion,

$$\widehat{\boldsymbol{Q}}^{-1} = \frac{\widehat{\boldsymbol{Q}}^*}{\left\|\widehat{\boldsymbol{Q}}\right\|^2}$$

These properties of quaternions are proven to be useful to represent the orientation of a ridged body or coordinate frame in three-dimension space. Like the axis and angle representation, the quaternion used for representing rotation involves an axis about which there is a rotation, and the angle of rotation.

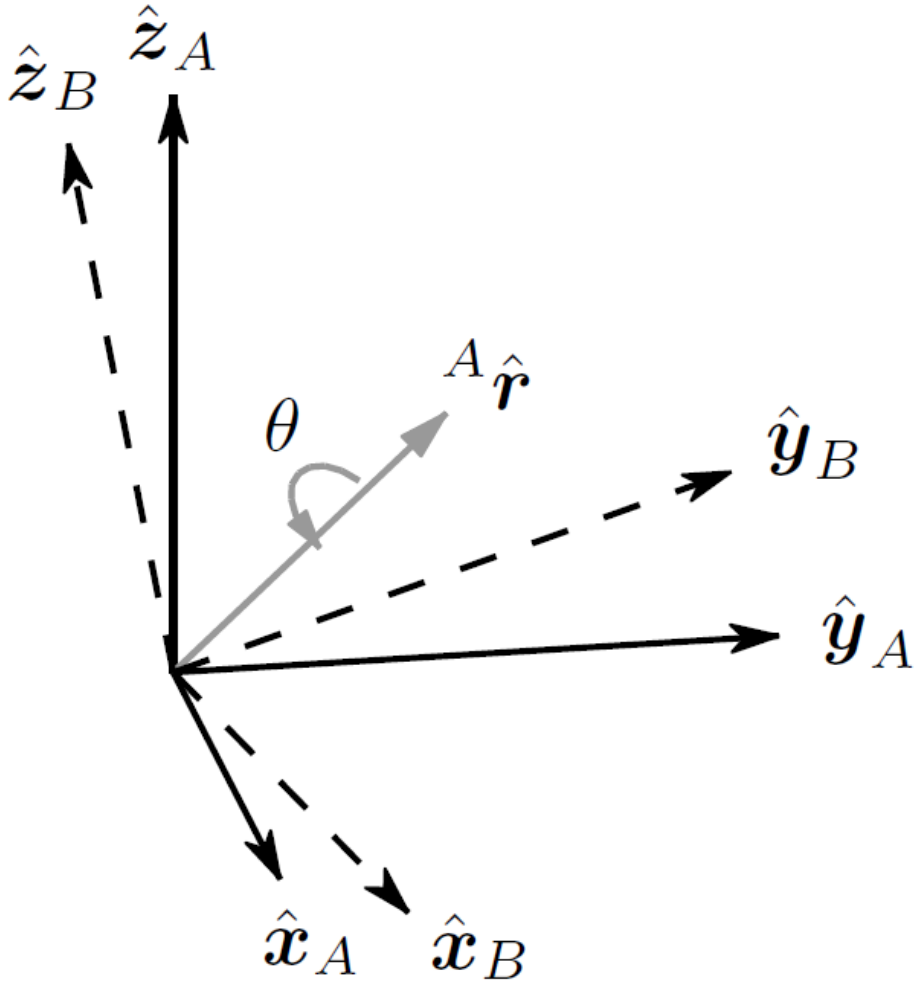


Figure 14 Quaternion representation

An arbitrary orientation from frame B to frame A can be represented as a rotation of angle θ around an axis $\hat{\boldsymbol{r}}$ in frame A. The quaternion ${}^{A}_{B}\hat{\boldsymbol{q}}$describing this rotation is defined as:

$$
\begin{aligned}
{}^{A}_{B}\hat{\boldsymbol{q}} &= (q_0 \quad q_1 \quad q_2 \quad q_3) \\
&= \left[\cos\frac{\theta}{2} \quad \left(\sin\frac{\theta}{2}\right)\hat{\boldsymbol{r}}\right] \\
&= \left(\cos\frac{\theta}{2} \quad -r_x\sin\frac{\theta}{2} \quad -r_y\sin\frac{\theta}{2} \quad -r_z\sin\frac{\theta}{2}\right)
\end{aligned}
$$

where r_x, r_y, and r_z represents the components of the unit vector $\hat{\boldsymbol{r}}$ in X, Y and Z axis of frame A respectively.

The complex conjugate of ${}^{A}_{B}\hat{\boldsymbol{q}}$, can be used to swap the relative frames described by ${}^{A}_{B}\hat{\boldsymbol{q}}$,

$$ {}^{B}_{A}\hat{\boldsymbol{q}}^{*} = {}^{A}_{B}\hat{\boldsymbol{q}} = (q_0 \quad -q_1 \quad -q_2 \quad -q_3) $$

describes the rotation around axis $\hat{\boldsymbol{r}}$ from frame A to frame B.

The multiplication of quaternion is used to define the combination of two rotations. The first rotation is from frame A to frame B, described as ${}^{A}_{B}\hat{\boldsymbol{q}}$, the second rotation is from frame B to frame C, described as ${}^{B}_{C}\hat{\boldsymbol{q}}$. This can be represented by one quaternion ${}^{A}_{C}\hat{\boldsymbol{q}}$, defined as:

$$ {}^{A}_{C}\hat{\boldsymbol{q}} = {}^{B}_{C}\hat{\boldsymbol{q}} \otimes {}^{A}_{B}\hat{\boldsymbol{q}} $$

Quaternion product is not commutative, that is,$\hat{\boldsymbol{a}} \otimes \hat{\boldsymbol{b}} \neq \hat{\boldsymbol{b}} \otimes \hat{\boldsymbol{a}}$,

We have,

$$ \hat{\boldsymbol{a}} \otimes \hat{\boldsymbol{b}} = [a_0 \quad a_1 \quad a_2 \quad a_3] \otimes [b_0 \quad b_1 \quad b_2 \quad b_3] $$

$$ = \begin{bmatrix} a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3 \\ a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2 \\ a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1 \\ a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0 \end{bmatrix}^{T} $$

Since a 3D vector can be regarded as quaternion with the first element equal to zero, a 3D vector rotated around an axis for certain degrees can be simply defined by the multiplication of quaternions.

Suppose a vector $\hat{\boldsymbol{v}}$ is rotated from frame A to frame B, then

$$ \hat{\boldsymbol{v}}_B = {}^{A}_{B}\hat{\boldsymbol{q}} \otimes \hat{\boldsymbol{v}}_A \otimes {}^{B}_{A}\hat{\boldsymbol{q}}^{*} $$

where $\hat{\boldsymbol{v}}_A$ and $\hat{\boldsymbol{v}}_B$ represent $\hat{\boldsymbol{v}}$ in frame A and frame B respectively.

This is a nice and clean representation of rotation between two coordinates or for the orientation of vectors. It is easy to achieve in coding, and quaternions do not suffer from any of the drawbacks of a rotation matrix or Euler Angles. Thus quaternions are selected as the representation of rotation in this project.

CHAPTER 3: EXPLICIT COMPLEMENTARY FILTER

Extensive research has been done in the field of attitude/orientation estimation filters over decades. Many filters have been implemented to achieve the goal of accurate attitude estimation, such as Kalman filter, extended Kalman filter [13], with a lot of variations, interlaced Kalman filter [14-16], particle filter [17-18], unscented filter [19-20], orthogonal attitude filter [21], and so on.

Conventional attitude or orientation estimation filters are computationally expensive and that makes them difficult to run in current low-expense mobile devices. So a low-cost AHRS algorithm but with equally efficient estimation is desired. Thus the complementary filters are developed. Complementary filters employ gyroscope data for high frequency attitude estimation because gyros have good dynamic features, and employ accelerometer data for low frequency attitude estimation because accelerometers have good static features.

Gradient Descent based Complementary Algorithm (GDCA) proposed by Madgwick et al. [22] and Explicit Complementary Filter (ECF) proposed by Mahony et al. [23] are the latest advancement in complementary filters. Both techniques employ quaternion as the representation of rotation. While GDCA and ECF are both effective and novel approach in terms of low-cost attitude estimation, ECF has a bit of an edge over GDCA because of higher accuracy. ECF is most flexible with two adjustable gains compared to GDCA with only one [24], even though the time complexity of ECA is 25% more than that of GDCA [25].

In this paper, ECF is selected as the attitude estimation method with quaternion as the representation of rotation.

3.1 Sensor Noise Modelling

3.1.1 Our Sensor

The specific IMU we used in this work is MPU 9250 made by InvenSense Inc. [26] and the sensor module we used is the SP-10C developed by Sensoplex Inc. The SP-10C features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three 16-bit ADCs for digitizing the accelerometer outputs, and three 16-bit ADCs for digitizing the magnetometer outputs. For precision tracking of both fast and slow motions, it features a user–programmable gyroscope full-scale range of ±250, ±500, ±1000, and ±2000 °/sec (dps), a user–programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g, and a magnetometer full-scale range of ±4800 μT [26-27].

Figure 15 shows the block diagram of SP-10C. It is powered by a 5 volt Li-ion rechargeable battery when not connected to a computer. Data can be streamed or logged via USB with the SP monitor running on the computer. The SP-10C can also be connected via Bluetooth Low Energy (BLE) wirelessly. In our experiment, two modules of SP-10C were involved. One is connected to a computer through a USB cable and serves as the Bluetooth master. The other one which does the data collection task is attached on the foot and serves as the Bluetooth slave. Once Bluetooth connection is set up, commands can be sent wirelessly from the master to the slave and response is sent back to master. During data streaming, data sampled at a user-defined frequency is sent back to the master in real time. While for data logging, once the “log data” command is received by the slave, the accelerometer and gyroscope start to sample the readings at a frequency of 180 Hz, which is empirically the maximum frequency that the sensors can

be sampled without data loss. Otherwise a few consecutive sets of data would be lost periodically due to limited bandwidth. Data is stored in the 32 MB flash memory temporarily. Once the slave receive the "stop logging" command, the CPU stops sampling the sensors and data is ready to be read out to computer either via Bluetooth or USB.

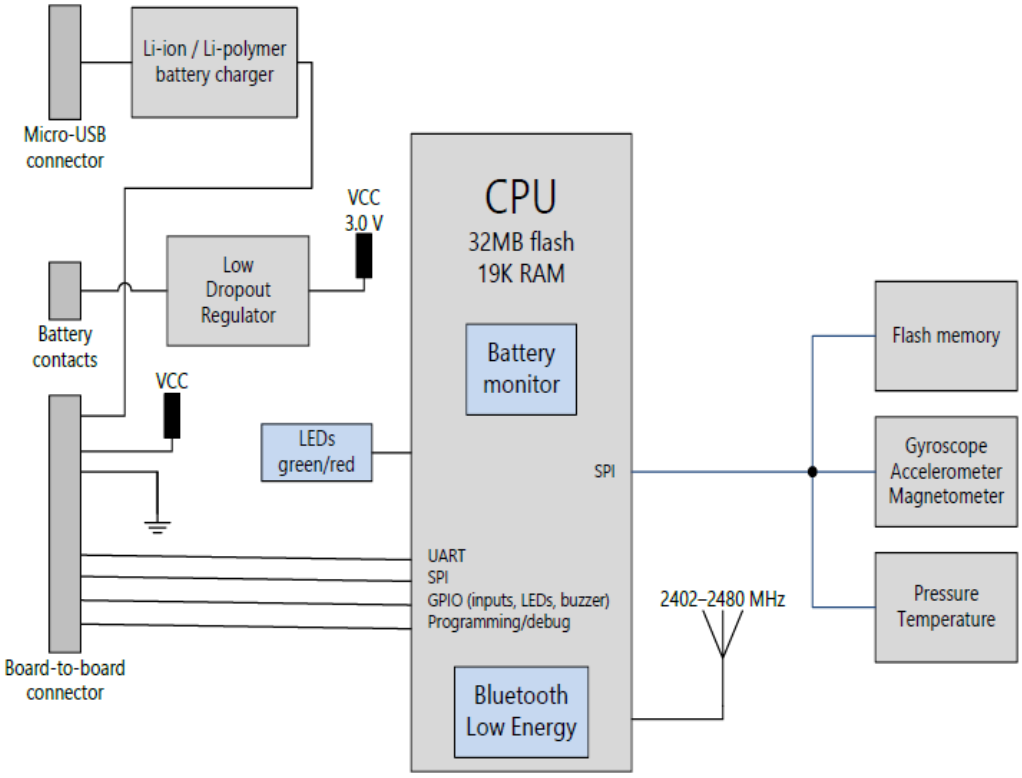


Figure 15 System chart of SP-10C

3.1.2 Gyroscope Noise Model

Gyroscope measures the angular rate. However the main concern is the bias issue. It measures the rotation rate along with noise and bias about three orthogonally installed axis. For MEMS gyroscopes, temperature, impact force and other factors can cause

uncertain bias which is hard to model. For simplicity, the gyroscope measurement can be modelled as:

$$\Omega_g^s(t) = \Omega(t) + s_g(t) + n_g(t)$$

where $\Omega_g^s(t)$ is the gyroscope measurement in sensor frame, $\Omega(t)$ is the actual angular rate, $s_g(t)$ and $n_g(t)$ denote the uncertain bias and noise in gyroscope reading respectively.

3.1.3 Accelerometer Noise Model

Theoretically, accelerometer measures instantaneous acceleration only. However, practically, as described in section 1.2.2 Drifts and Errors of MEMS IMU, a MEMS accelerometer picks up all the forces applied on the proof mass. The acceleration readings are calculated from the mass of the proof mass and force applied on it. Therefore, gravitational acceleration with some added bias and noise are mixed in the readings. So the accelerometer measurement can be modelled as:

$$R_a^s(t) = \dot{v} - g + s_a^s(t) + n_a^s(t)$$

where $R_a^s(t)$ is the readings of accelerometers in sensor frame, $\dot{v}$ is the actual instantaneous linear acceleration, g stands for the gravity, $s_a^s(t)$ and $n_a^s(t)$ denote the uncertain bias and noise in accelerometer readings in sensor frame respectively.

3.2 Explicit Complementary Filter

We consider the attitude estimation problem first. The problem of obtaining good attitude estimation from readings of low-cost MEMS units, is characterized by high noise level and unstable additive bias. The filtering problem [39] is formulated as deterministic

observer kinematics posed directly on the special orthogonal group SO(3). The earliest complementary filter is the direct complementary filter as described in Section 1.2.2 Drifts and Errors of MEMS IMU. This performs a low-pass filtering on a low-frequency attitude estimate from accelerometer data and a high-pass filtering on high-frequency attitude estimate from gyro data, and then fuses the high and low frequency estimates together to form a good estimate at all frequencies. Later a related complementary filter was proposed termed the passive complementary filter which was further developed to formulize the measurement errors. Finally the explicit complementary filter was introduced that gives fairly good estimation of attitude and only need accelerometer and gyro data. The block diagram of implementation of the Explicit Complemenatary Filter is shown in Figure 16.

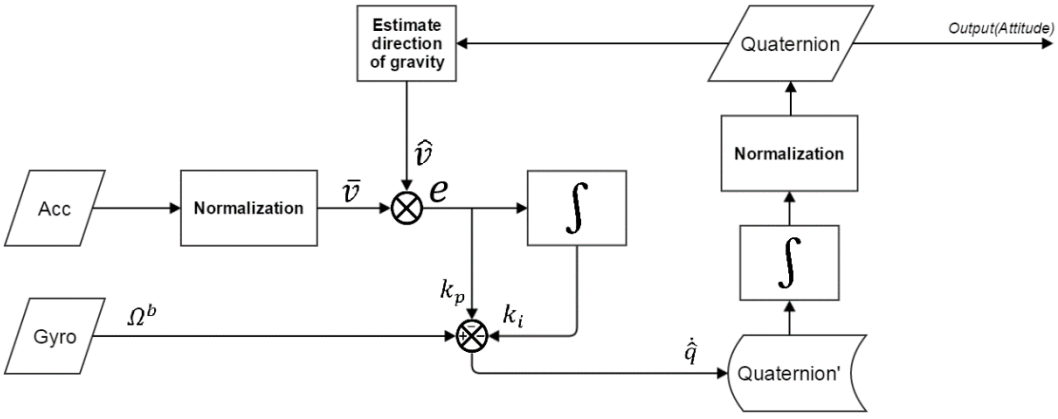


Figure 16 Block diagram of explicit complementary filter

The algorithm can be summarized as the following steps:

1. *Initialization:* the algorithm starts at assigning the quaternion an initial value, typically [1, 0, 0, 0] as the unit quaternion is used, meaning there is no rotation at this time. Sampling frequency, proportional gain k_p and integral gain k_i are defined as well.

2. *Data input:* one set of accelerometer data and gyro date is fed in and accelerometer data is normalized.

3. *Estimate gravity direction:* estimate the direction of gravity using current quaternion q [39],

$$\hat{v} = \begin{bmatrix} 2(q_1 q_3 + q_0 q_2) \\ 2(q_2 q_3 + q_0 q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

4. *Error calculation:* error is calculated by taking the cross product of the estimated direction of gravity and the "true" direction of gravity. In the scenario considered in this paper, the inertial direction $\bar{v}$ is the best estimation of gravity we can get. It is calculated from the accelerometer readings,

$$\bar{v} = \frac{\hat{g}}{|\hat{g}|}$$

So the error between the estimate of direction of gravity from gyroscope and that from the accelerometer is given by

$$e = \bar{v} \times \hat{v}$$

5. *Data fusion:* error is calculated to reduce bias in gyroscope data in the next loop. Error is applied as a feedback term with two adjustable coefficient: the proportional gain k_p and the integral gain k_i, which forms a PI controller,

$$\bar{\Omega}^b = \Omega^b + k_p e + k_i \int e \, dt$$

6. *Compute rate of change of quaternion:* according to differentiation formula of quaternion, we get,

$$\dot{\hat{q}} = \frac{1}{2} \hat{q} \otimes p(\bar{\Omega}^b)$$

where $p(\bar{\Omega}^b) = (0, \bar{\Omega}^b)$ and $\hat{q}$ is the quaternion of last iteration

7. *Estimate attitude:* now that the differentiation of the quaternion in last iteration is calculated, simply integrate it and yield estimated quaternion, namely attitude. Then the quaternion of this iteration is normalized.

Repeat: repeat from step 2 to start a new iteration and keep updating attitude.

3.3 Results

The AHRS algorithm only deals with attitude estimation issue, so we only observe rotation data here. Velocity and position results will be displayed and evaluated in Chapters 4 and 5.

Data was taken while the object walked along the longer side of Votey Hall which took approximately 60 seconds (for the first 10 seconds remained stationary so the algorithm could converge on a stable state). The sampling frequency was 180 Hz.

Figure 17 shows raw gyroscope data.

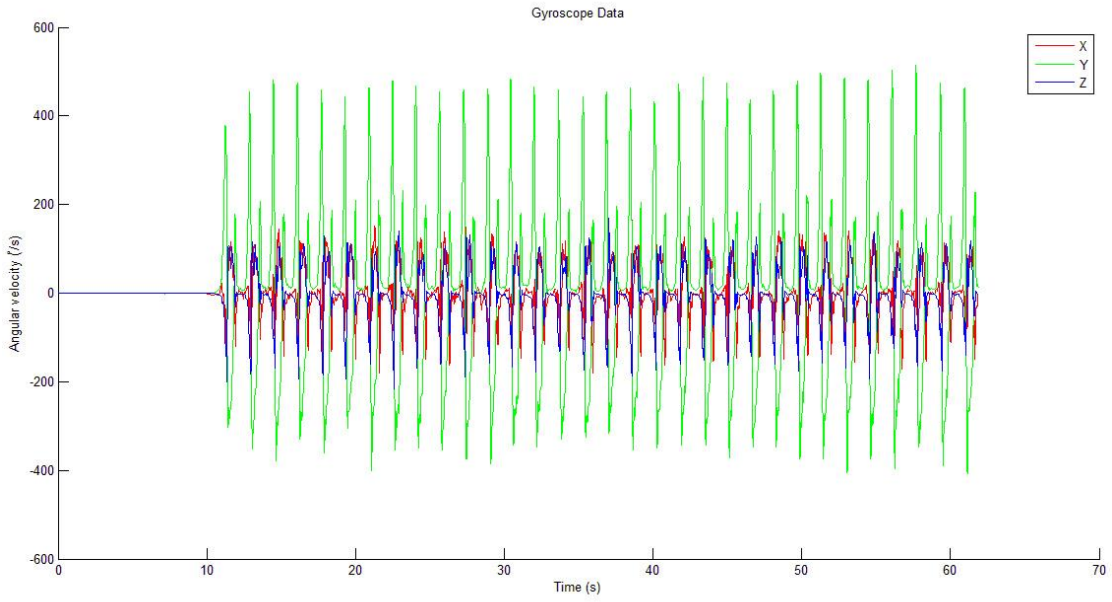


Figure 17 Gyroscope data

As the legend shows, red line stands for gyroscope data on X axis, green for Y axis and blue for Z axis. It is obvious that the magnitude of data on Y axis is much larger than those on X and Z axes. This is reasonable since, as shown in Figure 11, the X axis of the sensor frame points to the front of the walker, Z axis points along the leg of the walker, and Y axis is perpendicular to the X-Z plane and go through the arch of foot. Therefore data on X axis represents roll, data on Z axis represents yaw and data on Y axis represents pitch. It is human nature that the foot rotates up and down around the ankle. So it is reasonable that angle of pitch is much larger than roll and yaw.

At the start and the end of each step, the walker's feet are stationary on the ground. Also the walker did not make any turns, so the attitude should converge to at least around zero at the end.

First we observe the naive way of getting rotation: integrating gyroscope data over time.

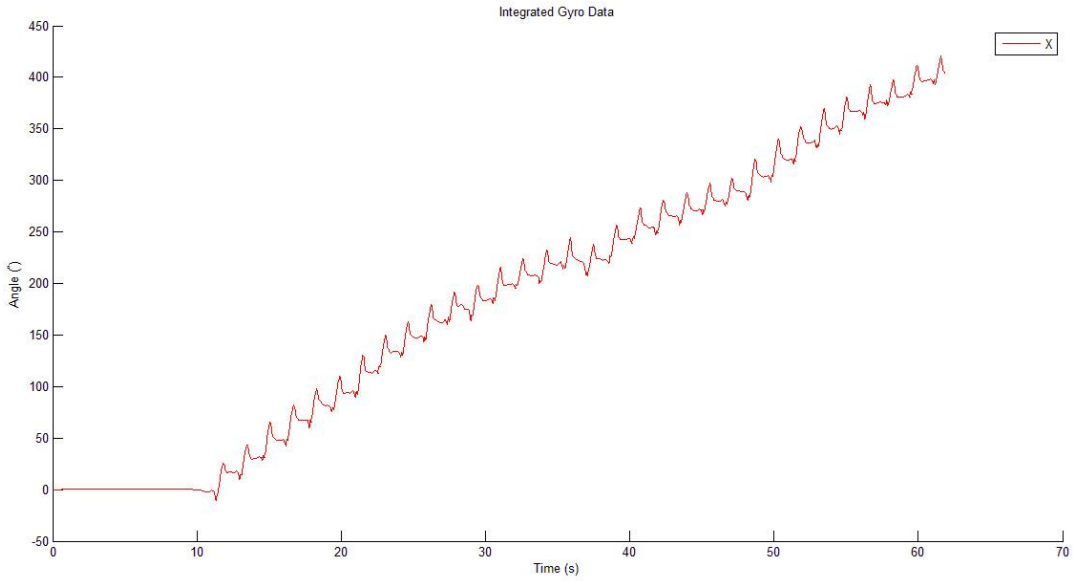


Figure 18 Integral of gyroscope data on X axis

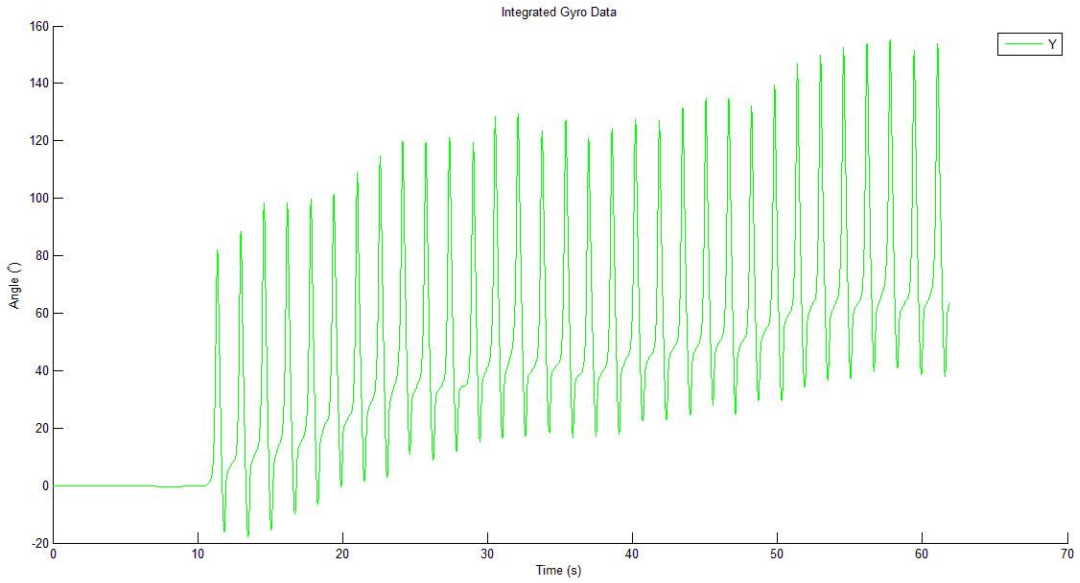


Figure 19 Integral of gyroscope data on Y axis

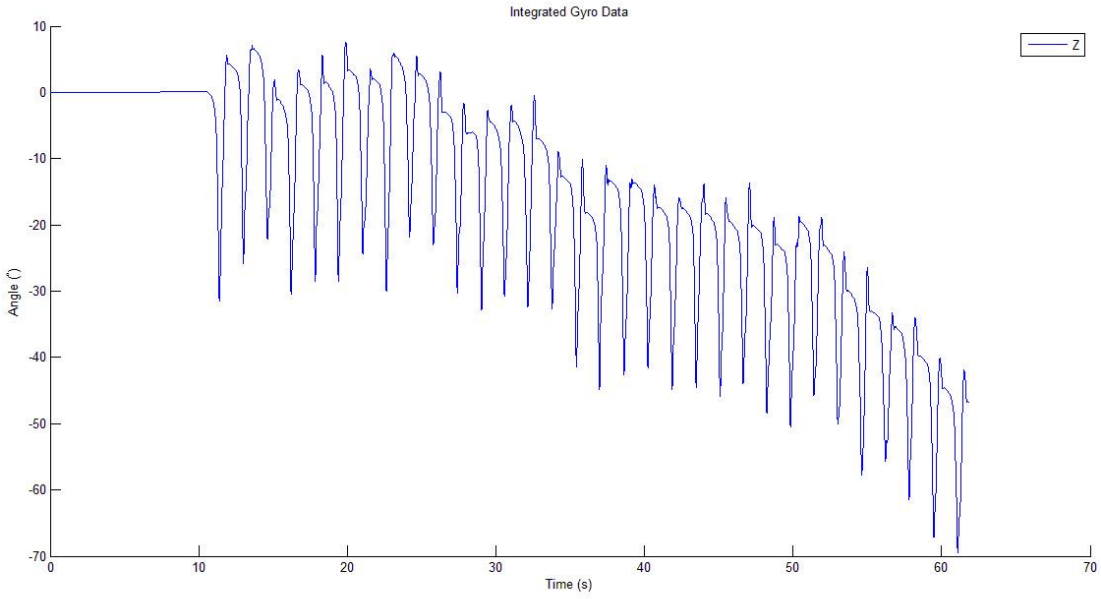


Figure 20 Integral of gyroscope data on Z axis

Figure 18 to 20 show integral of angular rate over time on all three axes respectively. The gyroscope's integration drift can be easily observed because the plots

do not converge to zero at the end. Drift on X axis is most severe. It drifted more than 400 degrees in less than 50 seconds. Drifts on Y and Z axes are not that severe but are large enough to distort the true attitude. Also we can view from these graphs that the additive bias is unstable as the plots do not drift at a constant speed. Hence we need to consider bias removal methods to obtain correct estimation of attitude.

Then we show the estimation of attitude output by ECF in the form of quaternions.

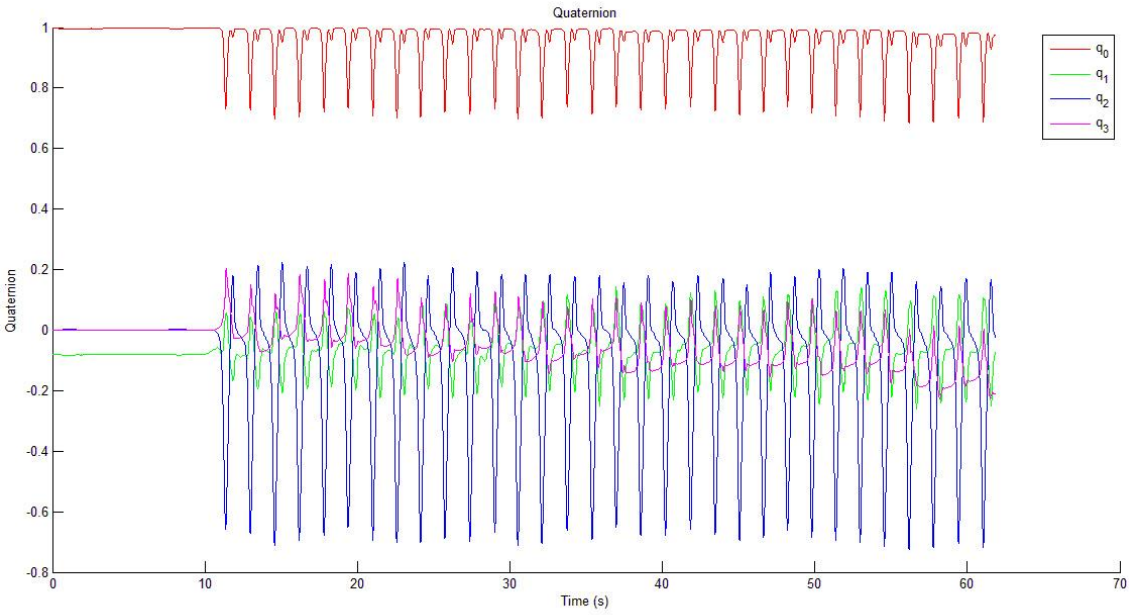


Figure 21 Quaternion

We can clearly see that curves are fairly straight and do not “drift”. However the quaternion representation is not an intuitive representation.

To make the results more vivid, we create an arbitrary vector [0, 0, 1] in true attitude coordinates, and use the quaternion calculated above to rotate the vector and see its representation in the estimated attitude coordinates. Ideally, it should converge to where it started and not drift.

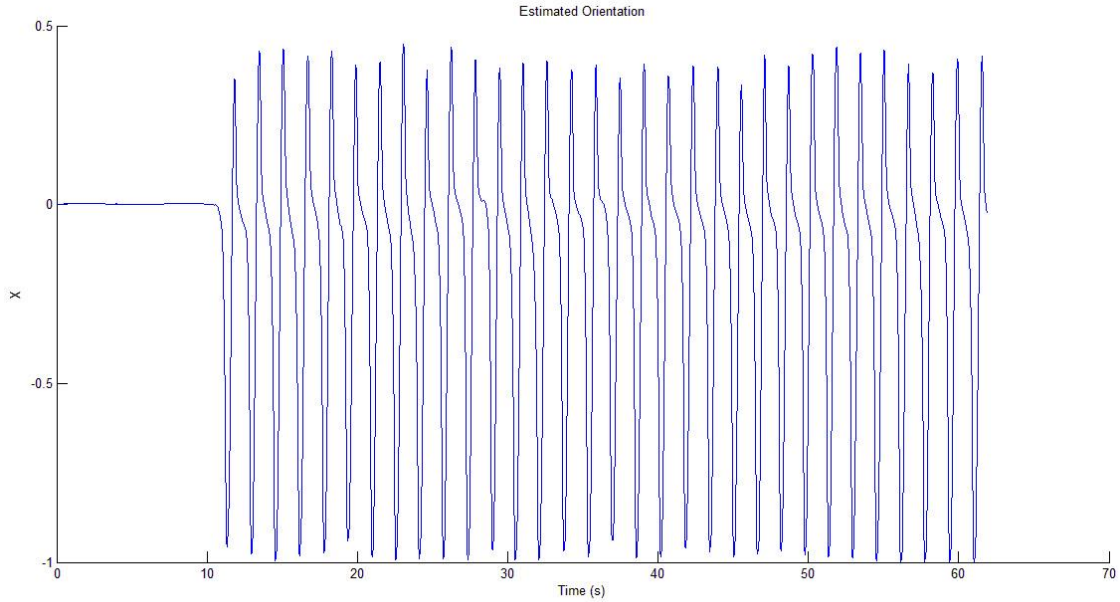


Figure 22 X coordinate

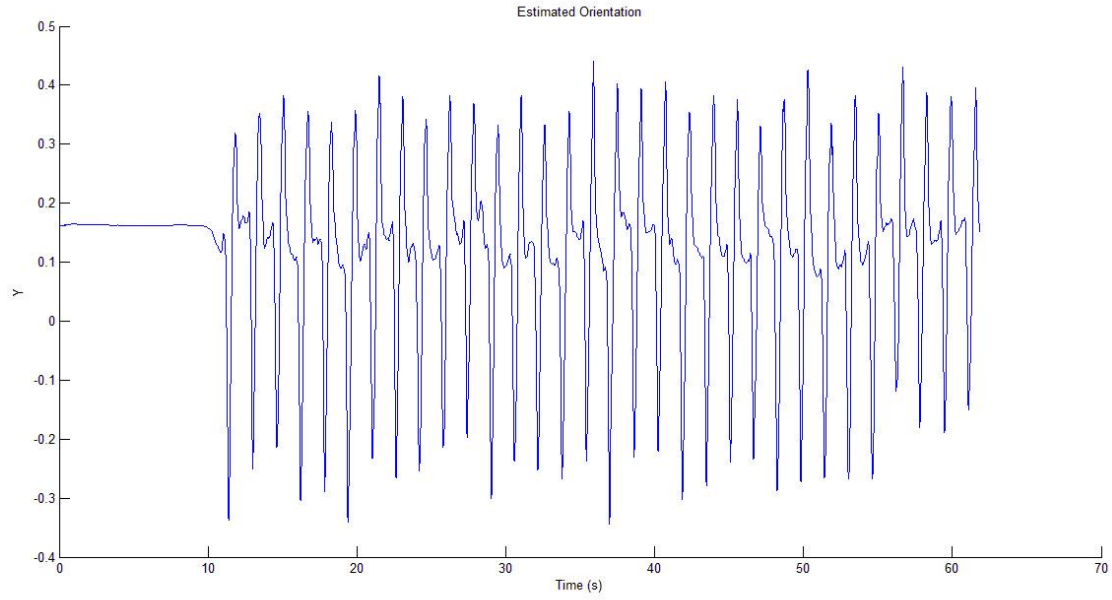


Figure 23 Y coordinate

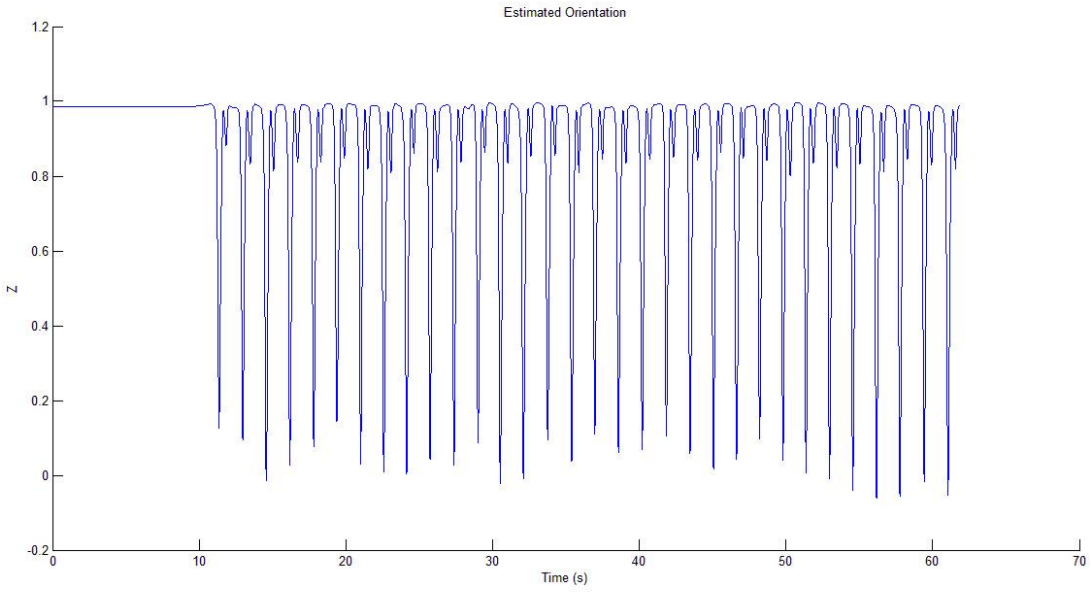


Figure 24 Z coordinate

From the figures above we can see that drifts have been almost eliminated, readings on all three axes converged to where they started. The explicit complementary filter gives a fairly good estimate of attitude.

Using these optimized estimated attitude data, physicians can become aware of how the foot tilts when a patient walks. This will permit diagnosis of certain diseases that affect walking or allow evaluation of how well the patient has recovered.

CHAPTER 4: ZERO VELOCITY UPDATE

4.1 Pedestrian Dead Reckoning

In navigation, pedestrian dead reckoning (PDR) is the process of estimating a pedestrian's trajectory and position by using a previously determined or fixed position and advancing that position based upon known or estimated speeds or accelerations over time.

Such a navigation system that tracks the location of a person is useful for finding and rescuing firefighters or other emergency first responders. As for example in our case, the caregivers can know the location of the patient that needs help immediately. It can also be employed in Location Based Services (LBS), mobile 3D audio and virtual reality applications.

In the previous chapters, we saw that it was possible to obtain a satisfactory estimation of attitude using explicit complementary filters. An associated goal of this project is to obtain an estimation of location. The approach will be explained in Chapters 4 to 6.

4.2 Zero Velocity Update (ZUPT)

Conventional personal dead reckoning systems detect steps using a pedometer or an accelerometer. Then it takes magnetometer data to detect which direction is the pedestrian facing. Whenever a step is detected, it moves the position estimation forward by one step length to that direction. The step length is usually calculated by a waist- or limb-mounted sensor that detect angle of limb swing [28]. Some systems even need input of a pedestrian's height to presume his step length. For these systems, step length is an

average of previous detected steps. Therefore it suffers from lags and does not give satisfactory instantaneous position estimation. What is more, these systems always move the position estimate one step forward to the front; if the pedestrian steps sideways, the estimated position still goes to the front.

In 1996, a DARPA (Defense Advanced Research Projects Agency) project proposed a method to detect steps using shoe-mounted sensors called zero-velocity update [29] (ZUPT), but the result was never published. The DARPA website denies public access. The first published research involving zero-velocity update was proposed by John Elwell [31]. Zero-velocity detection is a vital part in inertial navigation system. Because inertial sensors are subject to drift, if periods of stationarity is not detected from time to time, errors in acceleration would be integrated in to velocity and position that leads to drastic drift. Zero-velocity provides the required information to reset velocity [30].

There are basically two approaches to detect zero velocity. One is using the knowledge of motion patterns of human to detect stance phase. Typically, such methods model walking as a repeating sequence of heel strike, stance, push off, and swing. These methods only work for walking and cannot detect movements like running, crawling or walking backward [30].

The second and more generic option utilizes data from inertial sensor alone. It assumes that when the sensor is stationary (reading is 0), velocity is zero and angular rate is 0 as well. One concern is that if the accelerometer is moving at a constant speed, the algorithm would misjudge the motion as stationary. But based on our practice, since the accelerometer is very sensitive to accelerations, or, forces, and also because walking is a

rather complex course of acceleration and deceleration, the detection of zero velocity never failed because of misjudgment. There are times that motion detection threshold was not set properly which led to misjudging. However this is a problem that can be easily avoided.

4.3 Implementation

The implementation is shown in Figure 25.

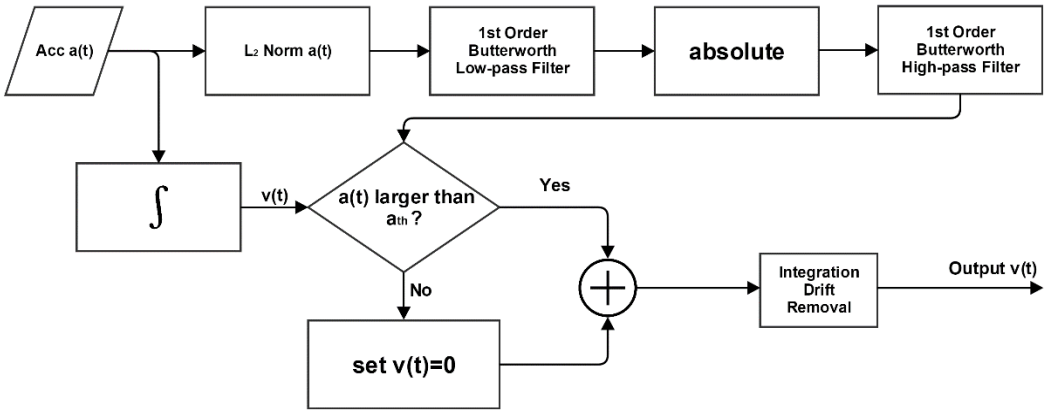


Figure 25 Block diagram of ZUPT

Step 1: calculate the l_2 norm on accelerometer readings on all three axes to get the magnitude of acceleration.

Step 2: pass the acceleration magnitude through a 1_{st} order Butterworth low-pass filter to smooth out the data.

Step 3: calculate the absolute value of filtered data.

Step 4: pass the data through a 1_{st} order Butterworth high-pass filter to remove DC value.

Step 5: decide if the current a(t) is larger than an empirically determined threshold a_{th}, if yes, do nothing, if not, set corresponding v(t) to zero. Filtered acceleration data is thrown away because its only purpose is to detect stationary periods.

Step 6: repeat from step 1 until all velocity data samples are updated.

Step 7: combine the corrected v(t) during stationary periods and v(t) during non-stationary periods together to yield zero-velocity updated v(t).

Step 8: because only velocity in stationary periods is set to zero, the integral of constant still exist during non-stationary periods. Also, because in step 5, velocity is forced to zero according to the threshold, the velocity before and after the critical point would be discontinuous. Therefore the drift caused by integration must be removed.

The way we remove integration drift of velocity is: first find the velocity difference between the start and end of a non-stationary periods, then divide that difference by the number of samples during this non-stationary period to get drift rate, after that multiply the drift rate with corresponding data index to get drift value at that certain point. Then subtract the drift from the previously calculated velocity to get linear velocity.

4.4 Results

First we observe the raw accelerometer data. Data was still taken during a straight walk along the longer side of Votey Hall on the third floor. The data is truncated to 35 seconds since the acceleration data would seem too dense for a 60 seconds walk.

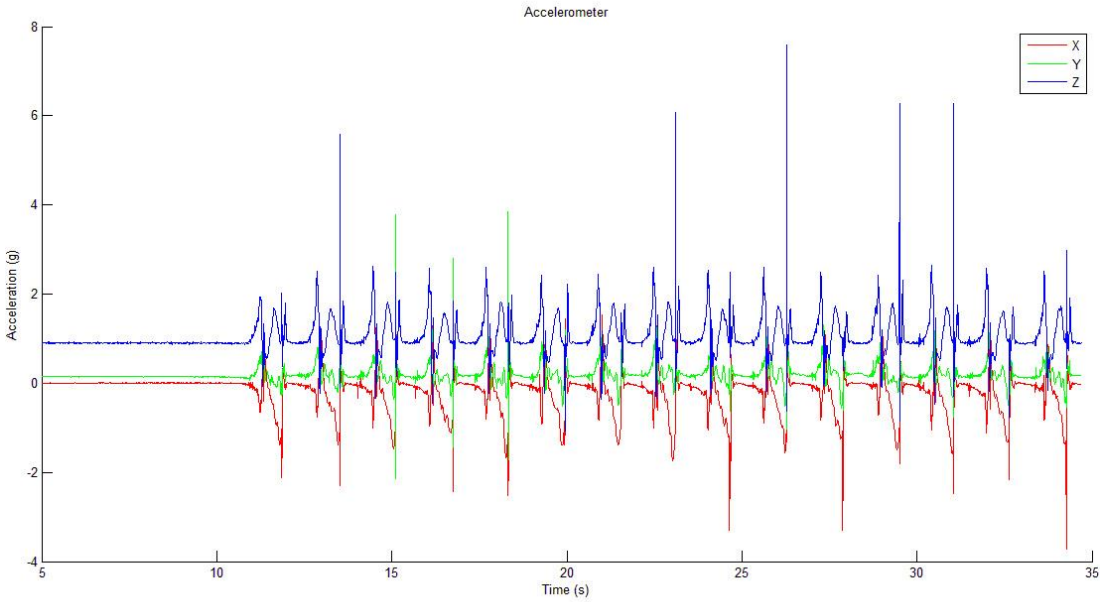


Figure 26 Raw accelerometer data

As is indicated by the legend, red line stands for accelerometer reading on X axis and green line stands for reading on Y axis and blue line stands for reading on Z axis, all in sensor frame, unit is g ($9.81 m/s^2$). From the raw data graph we can make a few observations:

First, before the commencement of walking, accelerometer reading on Z axis is much larger than those on X and Y axes which is reasonable because it measures acceleration force due to gravity.

Secondly, we observe that data on X and Y axes are not aligned to 0 perfectly. This suggests that the sensor is not placed perfectly upright (aligned to the earth frame) in the shoe, so portions of gravity show up on X and Y axis, which is absolutely fine. Pedestrians do not have to wear the sensors perfectly upright which is also impossible. The discrepancy between the sensor frame and the earth frame can be compensated by the AHRS algorithm.

Thirdly, a few spikes can be easily observed in the data on the Z axis because of noise or impact forces or else. Spikes are harmful to zero-velocity update and can cause misjudgment. This is the reason accelerometer data must be passed through a low-pass filter to smooth out before being sent into the decision logic.

What is more, there are pulses that can be easily recognized as each pulse represent a step.

In Figure 27 the filtered accelerometer data as well as the detected stationary periods are shown.

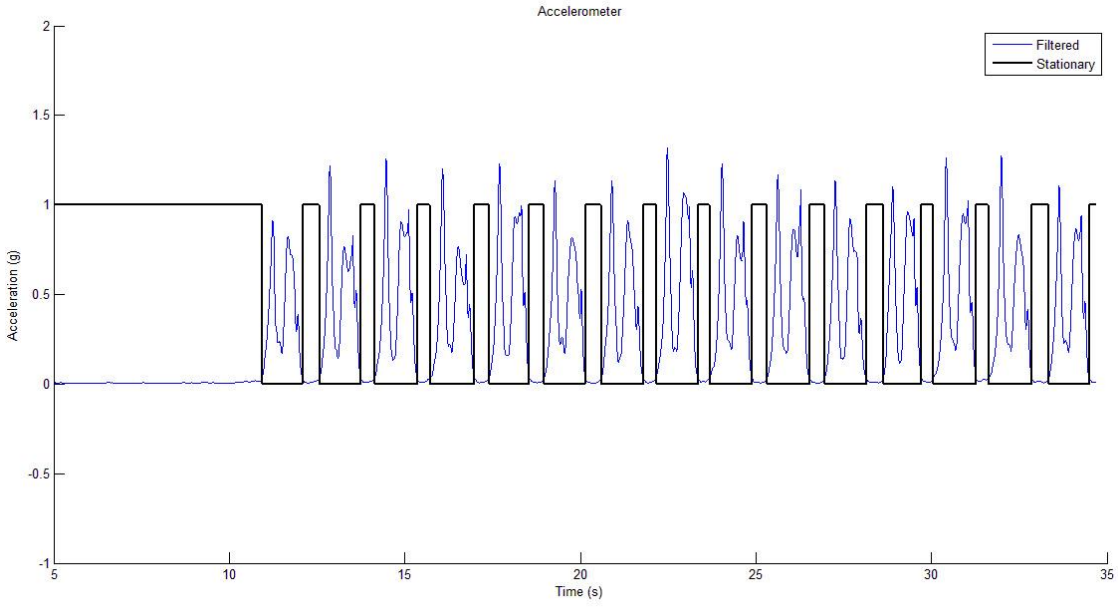


Figure 27 Filtered acceleration magnitude and stationary periods

In this figure we can clearly see that DC value of the acceleration is removed so during the stationary periods, velocity is just about 0. And during non-stationary periods, the filtered acceleration magnitude never reached near zero. Thus using a threshold of a_{th}=0.035 g, we can separate stationary and non-stationary periods vividly as is illustrated by the thick black line.

We first demonstrate the result without zero-velocity update.

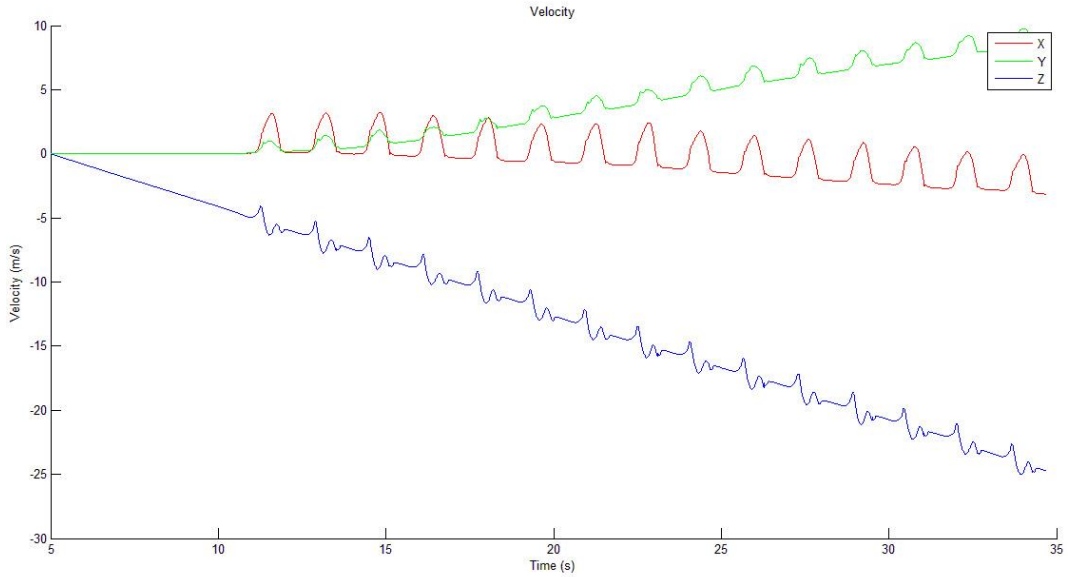


Figure 28 Velocity without ZUPT

Figure 28 shows the velocity data without zero-velocity correction. Steps are still recognizable but we can vividly see the drift caused by 1st order integration of constant value.

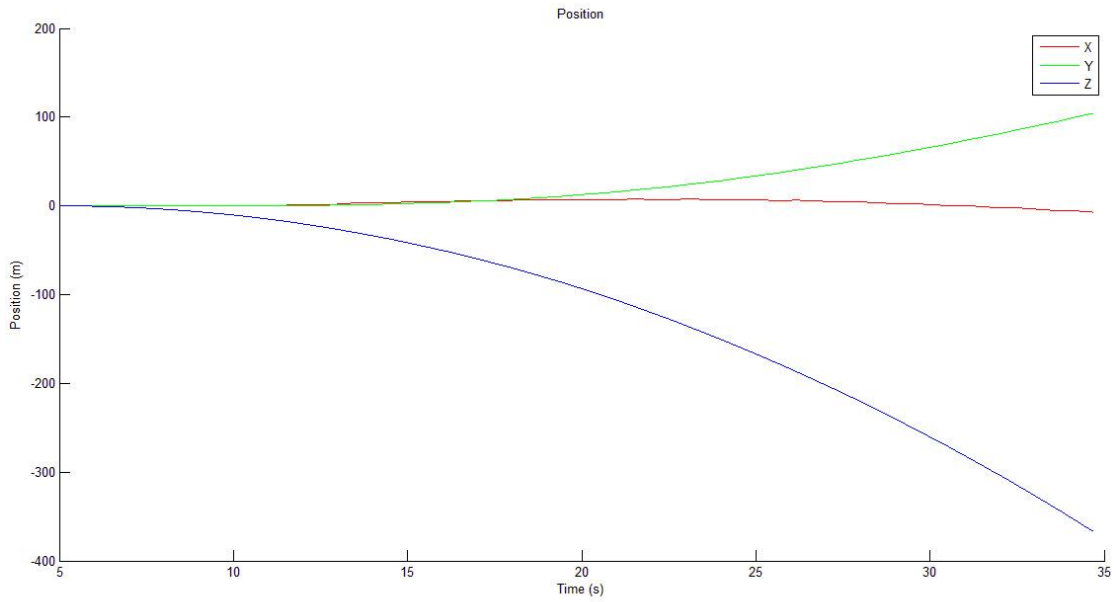


Figure 29 Position without ZUPT

Figure 29 shows the position result without zero-velocity correction. The position estimation drifts so drastically that we cannot even find ripples of steps in the curve. The position on Z axis drifted nearly 400 meters in 30 seconds. The results also demonstrated how little errors can cause huge drift after being integrated.

That shows the necessity of ZUPT. The effect of ZUPT is evaluated and displayed in Figure 30.

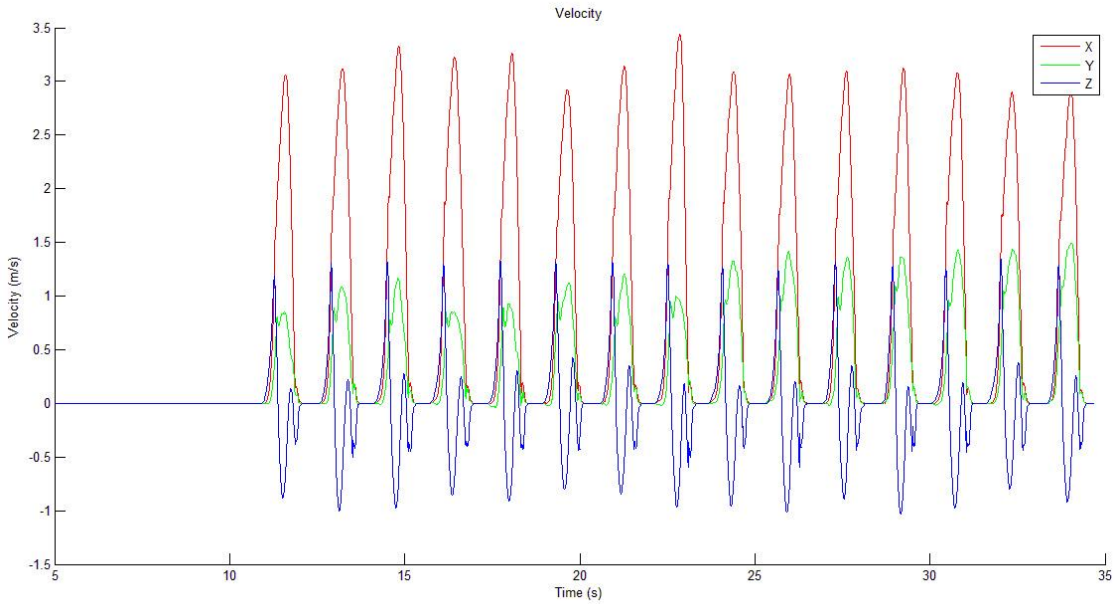


Figure 30 Velocity with ZUPT

Figure 30 shows the velocity estimation with ZUPT. We can see the graph is reasonably satisfactory. Three curves are aligned at zero during the stationary periods between pulses. Drift is eliminated and magnitude of velocity on each axis is reasonable, X being highest, Y and Z are low because the straight walk is basically along X axis (of sensor frame).

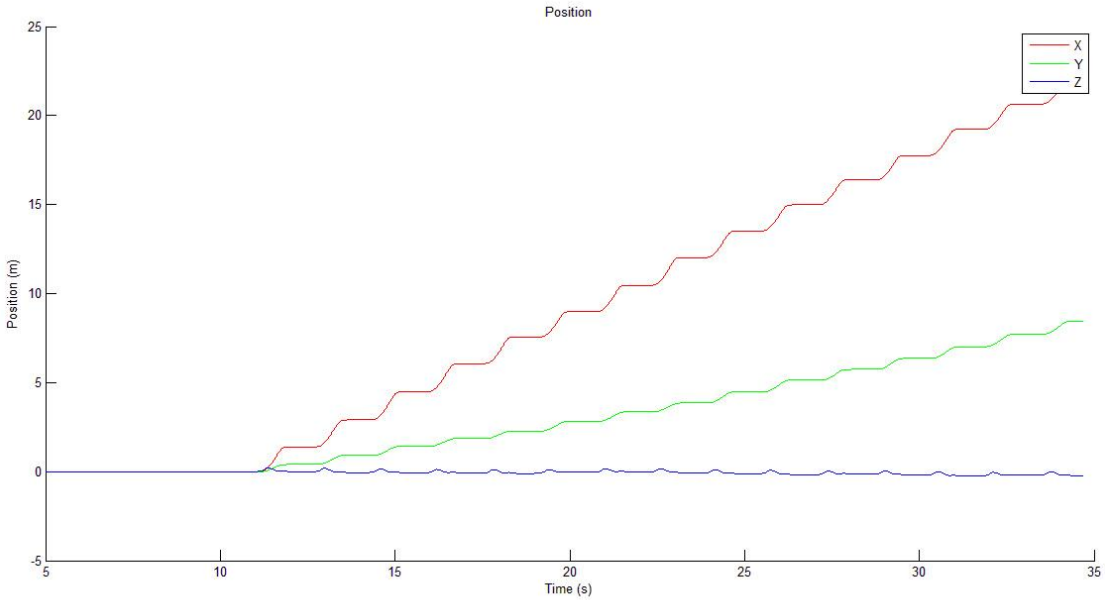


Figure 31 Position with ZUPT

Figure 31 shows the position estimate of the 30 seconds of straight walk. We can see that drift is greatly reduced (a little still exists and can be observable in a long walk) and position curve is almost linear. We can also notice the "ripples" within the curves, and each one represents a step made by the walker. There is also one notable phenomena, linear displacement is not on X axis only, there is some on Y as well. This suggests that X axis on sensor frame points to positive Y axis a little bit. That is probably because sensor is not placed perfect upright or the walker is a little toe-out while walking. But this is not a vital issue because as long as the relative relationship among position points remains, we can correct this deviation anytime.

This chapter showed the importance and implementation of zero-velocity update. It also gives another answer to the question: Why attach the sensor on shoes? That is to detect zero velocity and reduce drift.

CHAPTER 5: ENHANCED HEURISTIC DRIFT ELIMINATION

5.1 Preliminary Results

In previous chapters, quaternions and explicit complimentary filter were explained and evaluated. Also, the zero-velocity update method was introduced to reduce drift. Those two techniques comprise a basic functional pedestrian tracking algorithm. We first display and evaluate result of the ECF+ZUPT system, whose block diagram is shown in Figure 32.

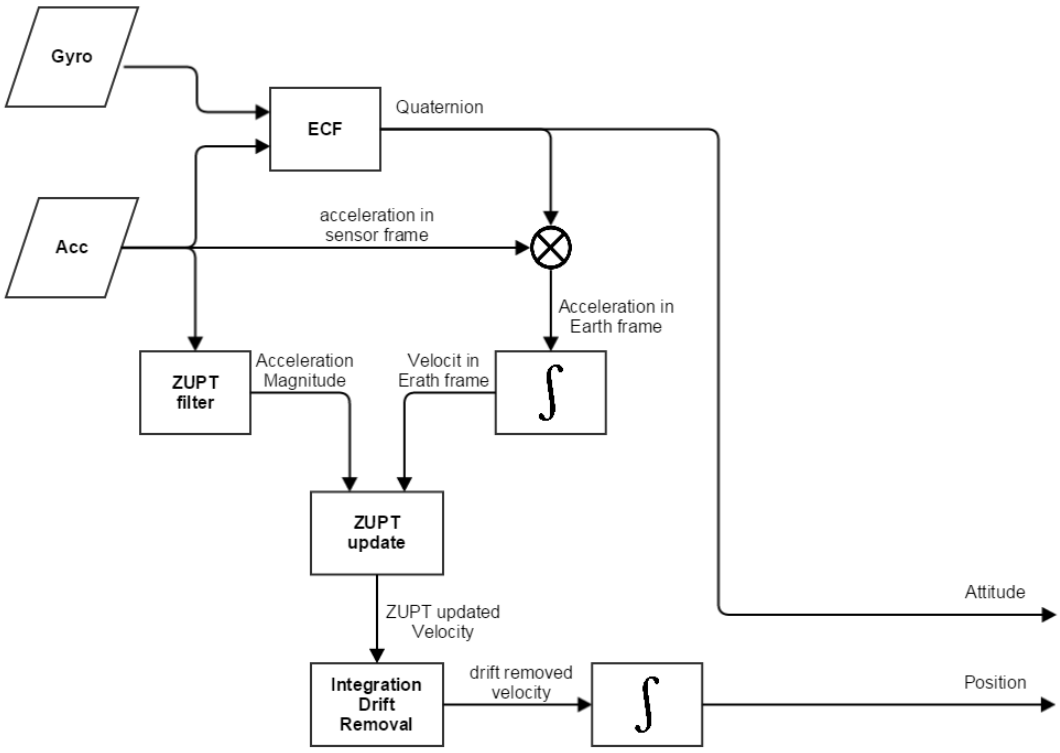


Figure 32 Preliminary Top Level Implementation

In previous chapters, data shown was in short scale and simple movement e.g. straight line. That was for simplicity because long periods of data would seem too dense and thus not illustrative. Now we wish to evaluate the robustness and precision of the system. Data taken is over longer periods or in complex movement e.g. climbing stairs.

First we observe the result from a 3-minute walk around Votey Hall. The walker made a closed loop through the corridor inside Votey Hall.

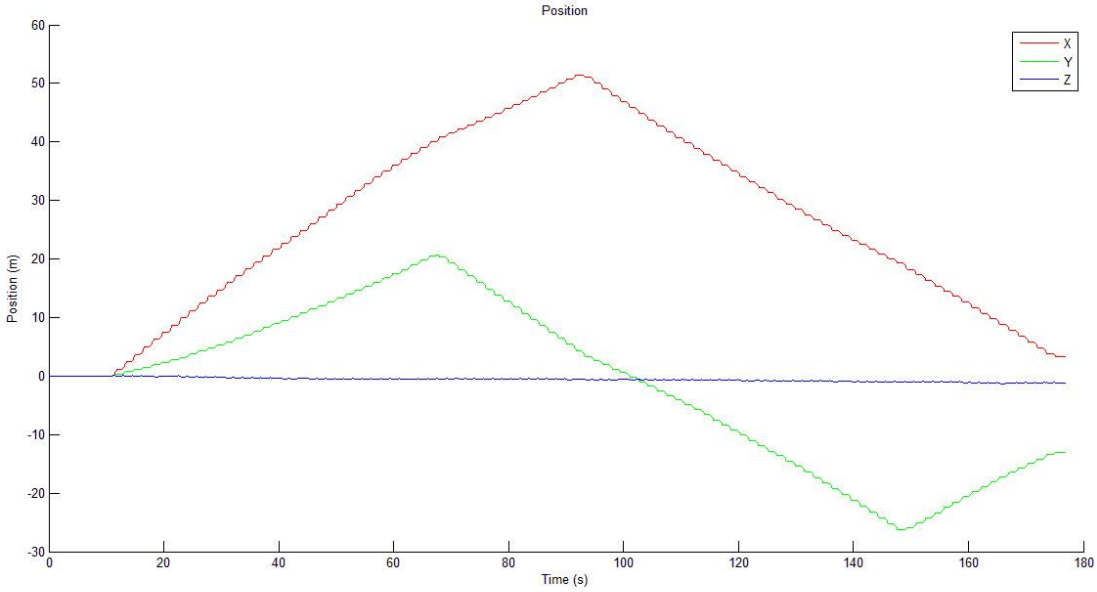


Figure 33 Plot of position data-Votey

Figure 33 shows the position data plot of the result. We can conclude from this graph that even though drift has been greatly reduced by ECF and ZUPT, error still exists. The walker made a closed loop. So at the end of the walk, the three lines representing displacement on three axes should meet at the origin, (0 , 0 , 0). But they did not. Also we can see that drift on X and Y axes are almost eliminated because the blue line representing displacement on Z axis is very nearly flat. These two conclusions can also be seen in the trajectory of the walk as is shown below.

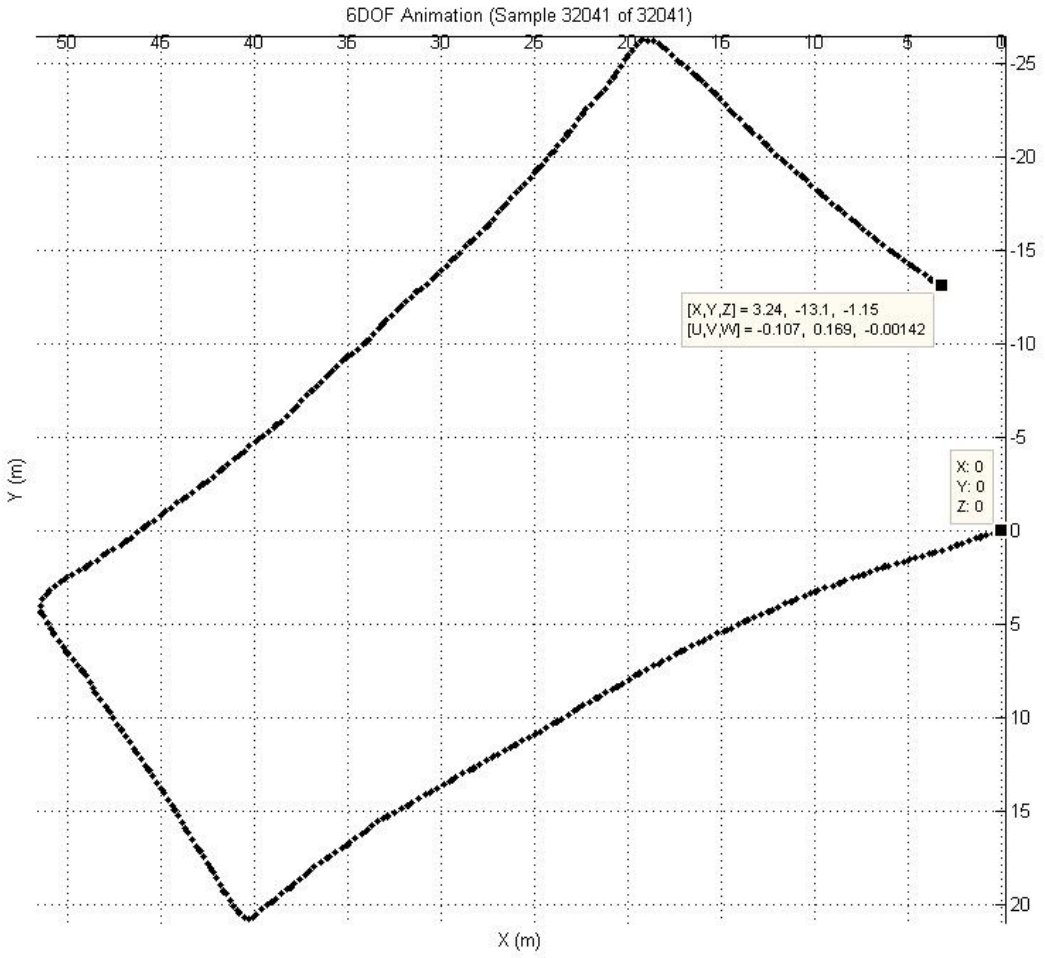


Figure 34 Trajectory of Votey top view

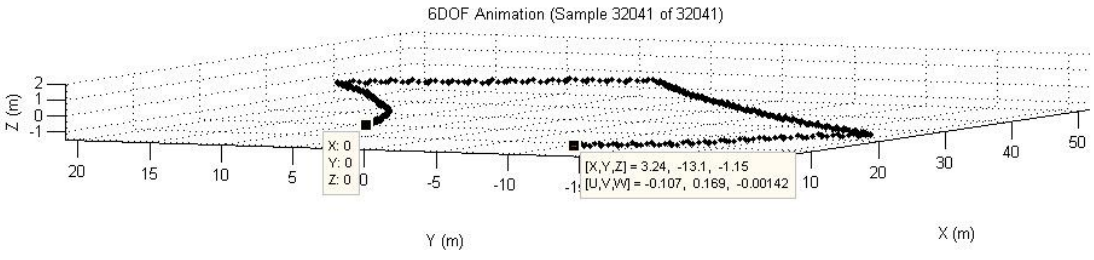


Figure 35 Trajectory of Votey side view

Figure 34 and 35 show the trajectory from top and side views. Figure 34 indicates that a small amount of drift still exists in yaw which will cause even severe results in

longer periods of walking. Figure 35 indicates that trajectory is mostly constrained in X-Y plane, which is good.

We also show another set of results come from a more complex motion pattern. Walker climbed from the first floor to second floor through the stairs at the west side of Votey Hall.

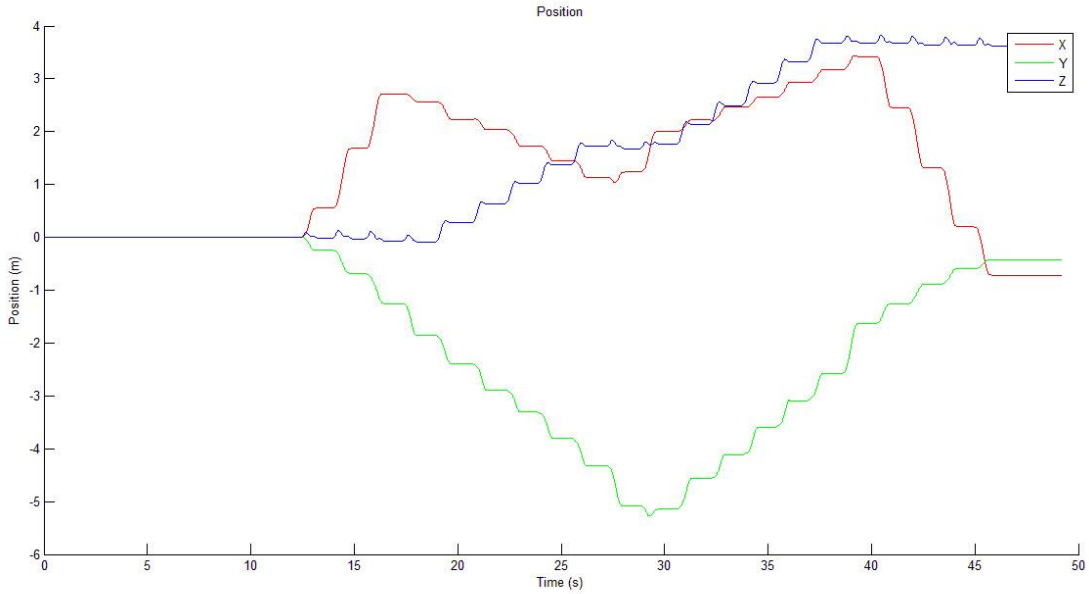


Figure 36 Plot of position data-stairs

Figure 36 shows the position data plot of the result. We can easily see the stairs by observing the Z line: first three steps are flat, then five steps going up, then two steps flat, then five steps going up again, then five steps flat. This position plot looks better than that of the previous one. The drift was observed from the trajectory plot.

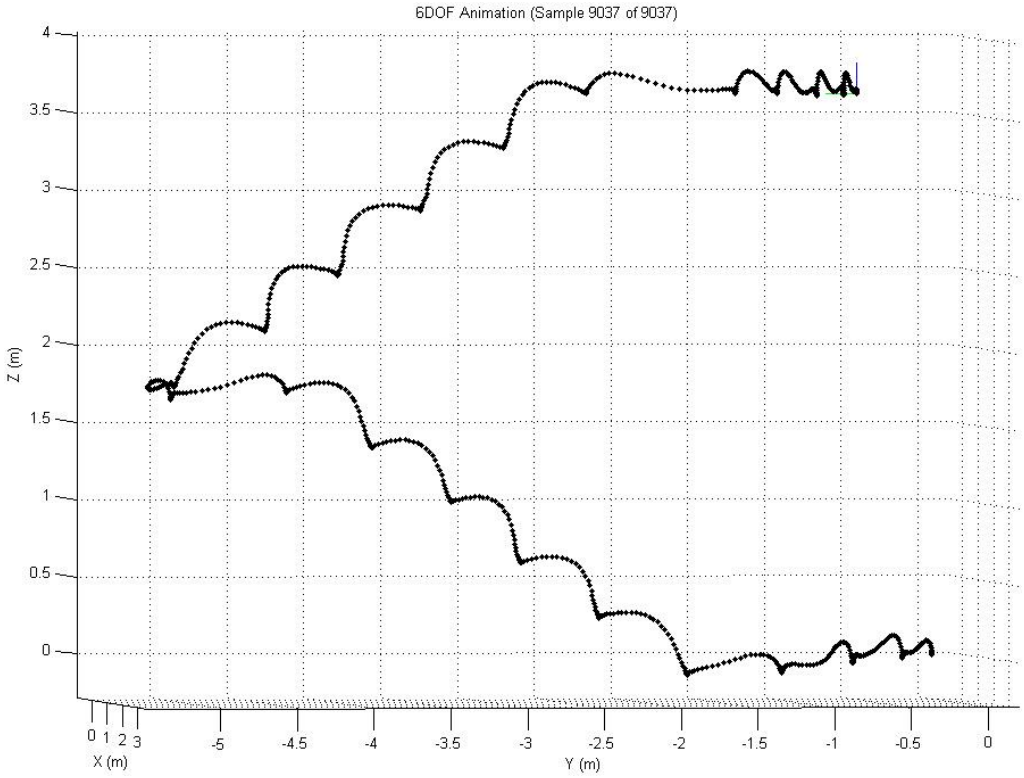


Figure 37 Trajectory of stairs side view

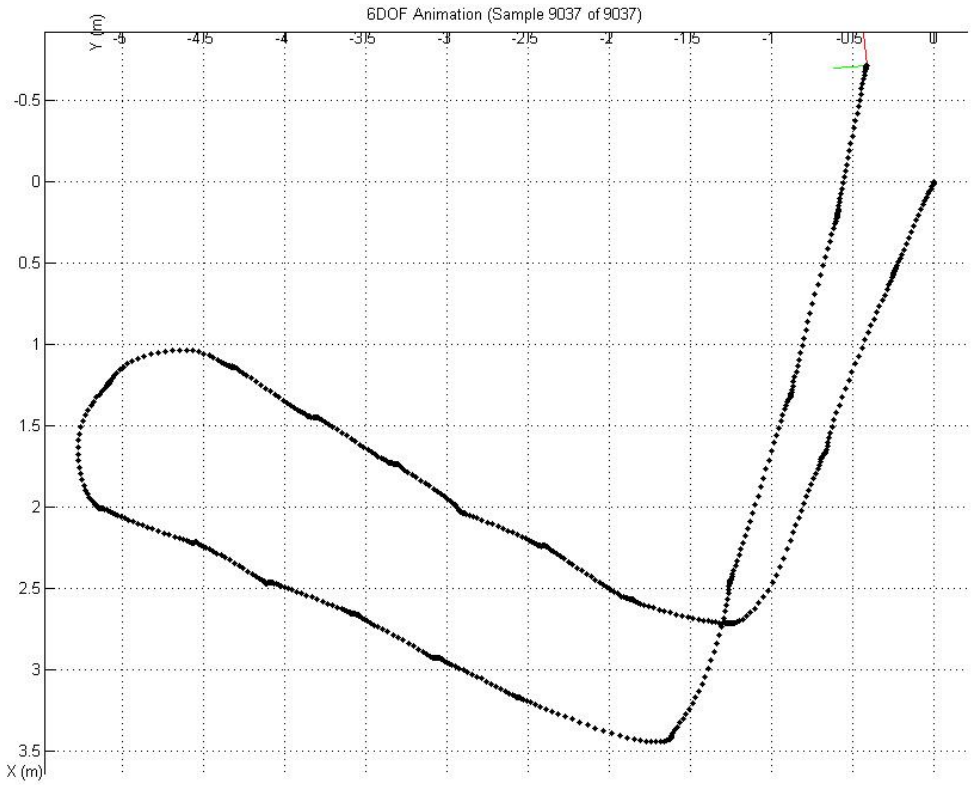


Figure 38 Trajectory of stairs top view

Figure 37 and 38 show trajectory views from side and top respectively. The side view looks all good, floor height of Votey is 3.6 meters and the trajectory agrees with that well. Problem happens in yaw drift again which can be seen in top view. There is an angle between the estimated corridors on the 2nd and 3rd floor. If the corridors were long enough the problem would be more obvious.

Table 1 Error of preliminary system

Data set	Time(s)	Steps	Error(m)
Around Votey	160	100	16
Votey Stairs	33	21	0.8

The two sets of results showed that ECF+ZUPT is still not sufficient for such low-cost sensors. Drift must be removed from yaw. Therefore we introduce Heuristic Drift Elimination (HDE) and propose the Enhanced Heuristic Drift Elimination (EHDE) method.

5.2 Heuristic Drift Elimination

In order to remove bias and errors from MEMS gyroscopes, a method called Heuristic Drift Elimination (HDE) was proposed by Johann Borenstein and Lauro Ojeda in 2010 [32]. Like in zero-velocity update, when existing algorithms are not sufficient to correct drift, we make use of external facts to provide more information. ZUPT takes advantage of the fact that during walking, either of the feet undergo a repeating sequence of stop and move. HDE takes into consideration the fact that in most buildings, corridors are parallel or orthogonal to each other, HDE calls the directions of the corridors dominant directions. Therefore most of the time the walker walks in a straight line along one of the four dominant directions. In 2012, A.R. Jiménez et al. proposed a magnetically-aided HDE algorithm that works in more complex buildings [33]. These are the two most cited work in Heuristic Drift Elimination and are the latest advancements.

5.3 Enhanced Heuristic Drift Elimination in 3D

In this paper we propose a novel HDE algorithm that works in 3D and can completely remove gyro drift in structured, indoor environment. Compared to HDE by Johann Borenstein and Lauro Ojeda, our EHDE algorithm works on much cheaper

MEMS IMUs that cost only a few dollars each while Johann used a sensor called Nano IMU produced by Memsense Inc. that costs more than $1300. HDE also has the limitation in that it can create new azimuth error by matching the closest dominant direction if the pedestrian walks in various directions. Compared to A.R. Jiménez's work, our EHDE can work without the aid of a compass. Also, both of the two HDE algorithms mentioned above do 2D drift elimination only while ours corrects gyro drifts in 3D. Implementation of EHDE is explained below. EHDE is mainly composed of three parts: pedestrian motion detection, dominant direction calculation and position update.

5.3.1 Pedestrian Motion Detection

In the beginning of drift elimination, EHDE takes the first five steps of the data and decides if the five steps are in a straight line. Since we use a low-cost MEMS IMU, gyroscope drifts really fast. So taking more steps would bring gyroscope errors in dominant direction calculation. Also large delays can appear. But if less steps were taken, motion detection is likely to be less accurate. According to our experiment, five steps provides a reliable and fairly accurate result.

To decide whether the pedestrian is walking straight, we first calculate step vectors from position data. Then we take the first five steps and calculate the angles between adjacent two steps. If absolutes of all of the angles are less than a threshold, we say the pedestrian is walking straight. If not, the pedestrian is not walking in a straight line.

If the pedestrian is not walking straight, EHDE does nothing.

If the pedestrian is walking in a straight line, EHDE moves to the next step, which is the dominant direction calculation.

5.3.2 Dominant Direction Calculation

Dominant direction calculation consists of two steps: the first step is to calculate the dominant directions' projections on X-Y plane. The second step is to generate dominant directions in 3D space.

Dominant direction projection is calculated from the five steps mentioned above. Five step vectors are calculated from 6 positions. We perform linear regression which fits a straight line based on perpendicular offset through the projections of the first 6 positions on X-Y plane, as is shown in Figure 39.

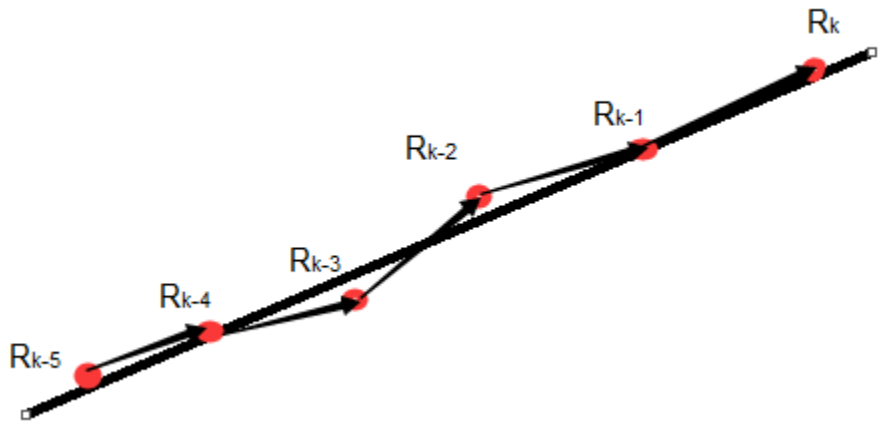


Figure 39 Position of previous 5 steps and current step

We assume a straight line in X-Y plane with function: $y = ax$ as the dominant direction projection.

The distance from the points to the straight line in Figure 39 is given by:

$$d_i = \frac{|y_i - (a + bx_i)|}{\sqrt{1 + b^2}}$$

Therefore, the sum of the squares of the distances is:

$$D = \sum_{i=k-5}^{k} d_i^2$$

Therefore coefficients a and b of the best fit line can be found by solving following equations:

$$\frac{\partial D}{\partial a} = \frac{2}{1 + b^2} \sum_{i=k-5}^{k} \big(y_i - (a + bx_i)\big)(-1) = 0$$

$$\frac{\partial D}{\partial b} = \frac{2}{1 + b^2} \sum_{i=k-5}^{k} \big(y_i - (a + bx_i)\big)(-x_i) + \sum_{i=k-5}^{k} \frac{\big(y_i - (a + bx_i)\big)^2 (-1)(2b)}{(1 + b^2)^2} = 0$$

We get,

$$a = \bar{y} - b\bar{x}$$

$$b = B \pm \sqrt{B^2 + 1}$$

where

$$B = \frac{\left(\sum_{i=k-5}^{k} y_i^2 - \frac{1}{6}\left(\sum_{i=k-5}^{k} y_i\right)^2\right) - \left(\sum_{i=k-5}^{k} x_i^2 - \frac{1}{6}\left(\sum_{i=k-5}^{k} x_i\right)^2\right)}{2\left(6\overline{xy} - \sum_{i=k-5}^{k} x_i y_i\right)}$$

Thus we get the dominant direction's projection $[1, a]$ on the X-Y plane.

To calculate dominant direction in 3D space there are two situations that need to be treated differently. Because floors in buildings are generally connected only by stairs or elevators, there is no possibility that a pedestrian gradually descends or ascends. Consequently only two situations are possible: walking horizontally or climbing stairs. Determination for 3D dominant direction is explained below.

A. Walker is walking on flat ground: Pass the absolute of the third element, namely, the vertical element on Z axis, of the current step vector through a threshold. If it is less than the threshold, we say the walker is walking on a flat ground. So dominant direction in 3D is $[1 \quad a \quad 0]$. That is only the direction that the pedestrian faces initially, named XP (positive on X axis). We can easily calculate the other three dominant directions from XP, named YP (positive on Y axis), XN (negative on X axis), YN (negative on Y axis) respectively, according to the right hand coordinates rule.

$$YN = \begin{bmatrix} 1 & -\frac{1}{a} & 0 \end{bmatrix}$$

$$XN = [-1 \quad -a \quad 0]$$

$$YP = \begin{bmatrix} -1 & \frac{1}{a} & 0 \end{bmatrix}$$

B. Walker is walking up or down on stairs: If the absolute of the Z component of the current step vector is larger than the threshold, we determine that the walker is climbing stairs. In this situation, we maintain the XY coordinates of the dominant direction and tilt the dominant direction so that it has the same gradient as the current step.

Suppose the current step is represented by vector:

$$step_i = [x_i \quad y_i \quad z_i]$$

So the gradient of the current step is:

$$g = \frac{z_i}{\sqrt{x_i^2 + y_i^2}}$$

Therefore, in order to maintain the gradient, dominant direction XP can be found simply by multiplying g with the norm of the first two elements of XP, that is:

$$XP = \begin{bmatrix} 1 & a & g * \sqrt{1^2 + a^2} \end{bmatrix} = \begin{bmatrix} 1 & a & \dfrac{z_i\sqrt{1+a^2}}{\sqrt{x_i^2 + y_i^2}} \end{bmatrix}$$

Similarly, the other three dominant directions are:

$$YN = \begin{bmatrix} 1 & -\dfrac{1}{a} & \dfrac{z_i\sqrt{1+\left(\frac{1}{a}\right)^2}}{\sqrt{x_i^2 + y_i^2}} \end{bmatrix}$$

$$XN = \begin{bmatrix} -1 & -a & \dfrac{z_i\sqrt{1+a^2}}{\sqrt{x_i^2 + y_i^2}} \end{bmatrix}$$

$$YP = \begin{bmatrix} -1 & \dfrac{1}{a} & \dfrac{z_i\sqrt{1+\left(\frac{1}{a}\right)^2}}{\sqrt{x_i^2 + y_i^2}} \end{bmatrix}$$

5.3.3 Position Update

The final step of EDHE is position update. In this step, first EHDE calculates the heading difference between the current step and each dominant direction and see if the heading difference is less than a threshold. The threshold is kept small so that there is no crossover zone of the possible range around each dominant direction. Therefore the current step either falls into the possible range of one of the four dominant directions or none of them.

If the current step doesn't belong to any possible range of the dominant directions, EHDE does nothing.

If the current step falls in one of the four dominant directions, do the follow:

Step 1: Calculate the quaternion HDEquat between the current step and the dominant direction.

Step 2: Use HDEquat to rotate all the position points after and including the current step to the dominant direction. The reason to rotate all the points is that because we use a low-cost MEMS IMU, it drifts at a rather fast rate. If only one step vector is corrected once, the following steps is likely to drift to the possible zone of another dominant direction and that would bring about destructive consequences to drift elimination. By rotating the whole remaining trajectory we can correct the drift little by little.

Step 3: Because quaternion rotation is not a rotation around a certain, it is a rotation of certain degrees around a fixed axis. So after the rotation, even though the remaining trajectory is parallel to the previous step, they are not continuous. So the last step has to subtract the difference between the two ends of the "breakpoint" to make the trajectory continuous.

5.3.4 Block Diagram

Block diagram of EHDE is shown in Figure 40.

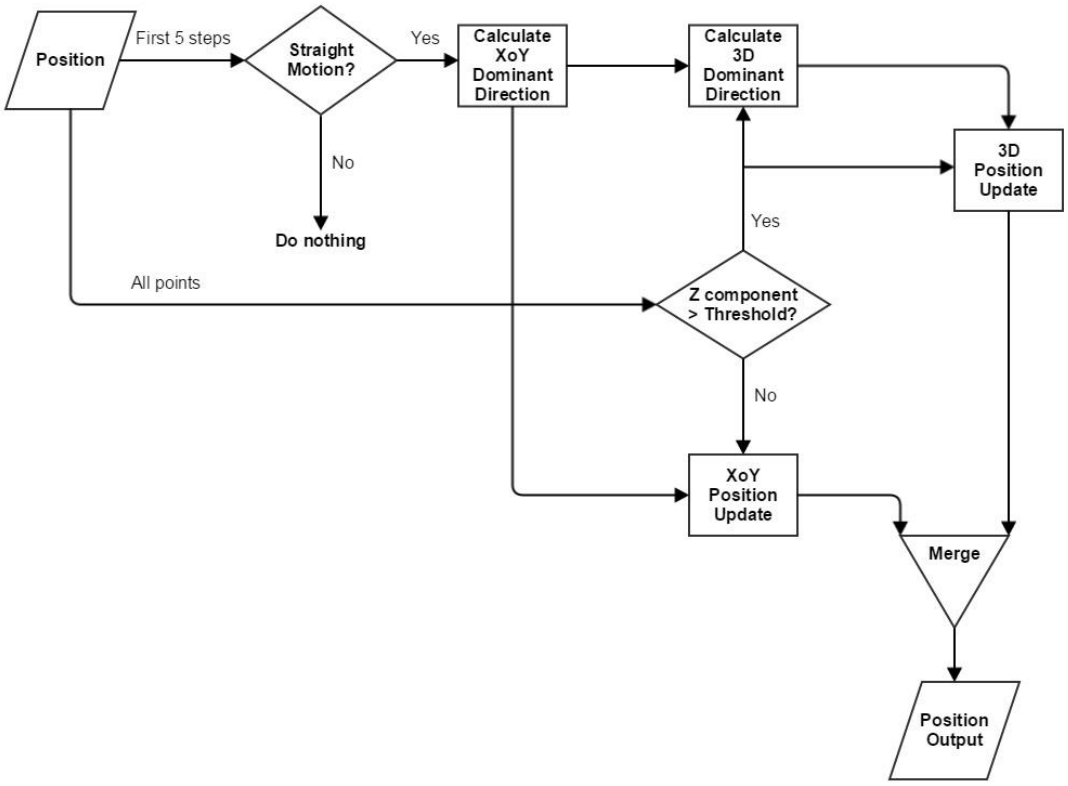


Figure 40 Block diagram of EDHE

From the block diagram we can see that EHDE only takes in position data and outputs updated position data. It is independent of the AHRS algorithm.

The reasons for maintaining dominant direction only on X and Y axes and keep same gradient on Z axis are as follows:

First, floor height varies from building to building, even within the same building. Therefore the system cannot tell whether it is drift.

Secondly, situation may vary according to how a pedestrian walks. Suppose the height of one stair is h. If pedestrian steps out his foot with the sensor first when he starts to go upstairs, the height change is, h, 2h, 2h, 2h…but if he steps out his foot without the sensor first, the height change would be, 2h, 2h, 2h, 2h…This makes height change uncertain and so it is difficult to reduce height drift.

Thirdly, as we can see from the two sets of results in Section 5.1 Preliminary Results, drift on Z axis is negligible. Usually there are no more than 20 stairs between floors so if there is drift it is within an acceptable range.

5.4 Results

We observe the two sets of data used in Section 5.1 Preliminary Results but after the correction of EHDE. We put the two sets of data together to see the improvement of results.

First we display the result of the closed loop around Votey output by EHDE. Figure 41 shows the position data plot.

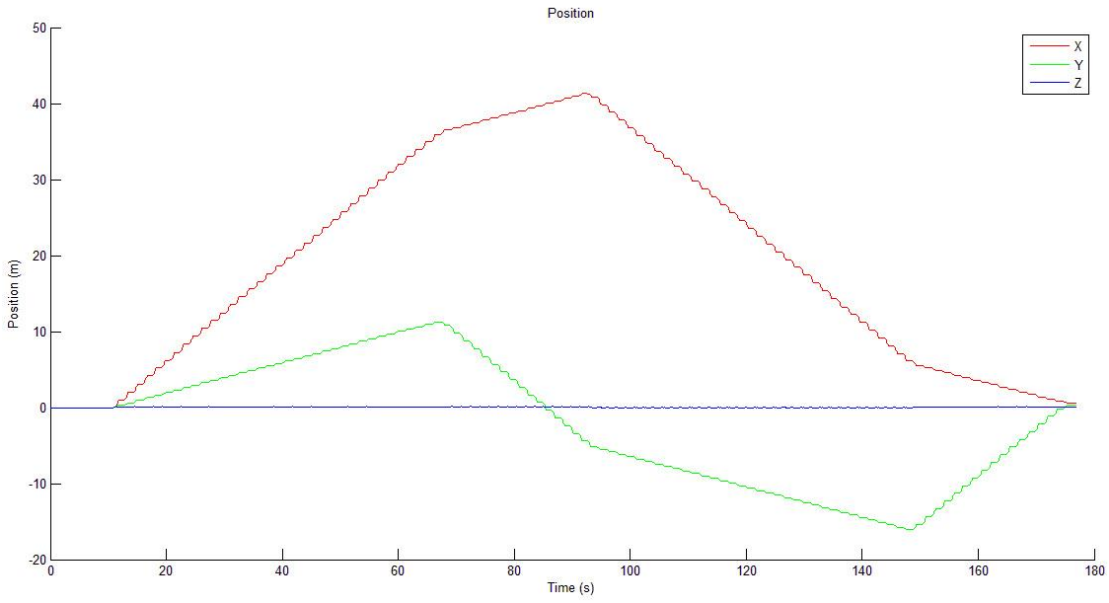


Figure 41 Plot of position data with EHDE for Votey walk

Compared to Figure 33 in Section 5.1 Preliminary Results, we can see that drift is completely removed; the four segments are perfectly linear indicating the trajectory on four sides of Votey were exactly straight. We can also see that three lines almost go back

to where they started-the origin, this indicates that the pedestrian went back to where he started.

Figure 42 and 43 show the top and side views of the trajectory.

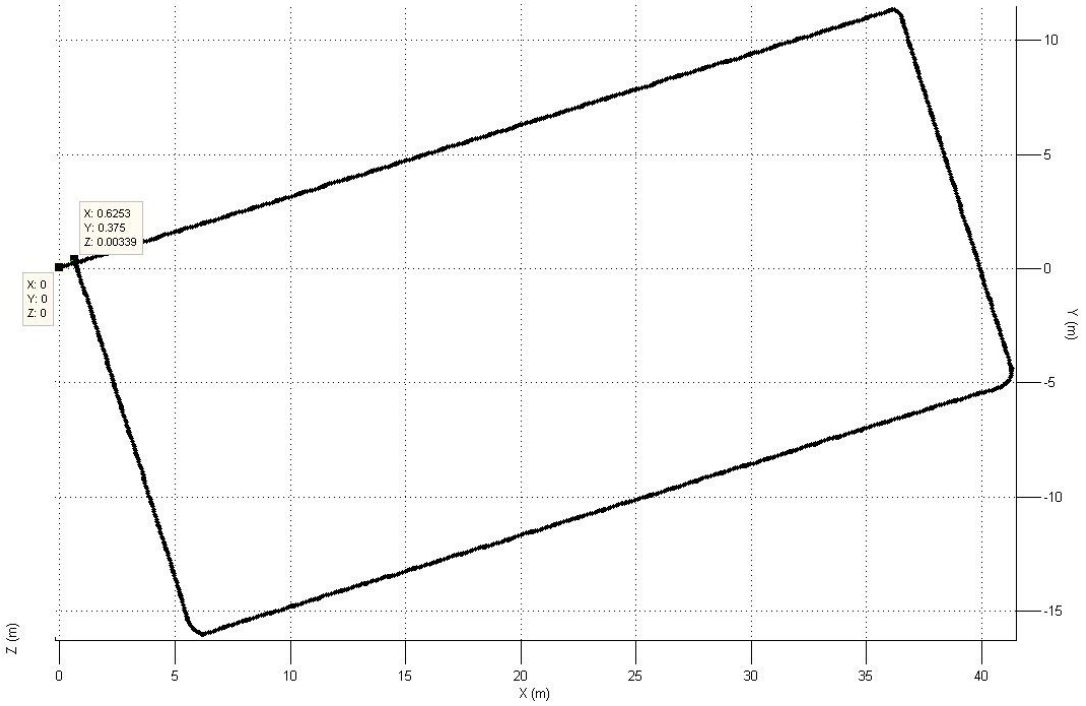


Figure 42 Trajectory of Votey walk top view

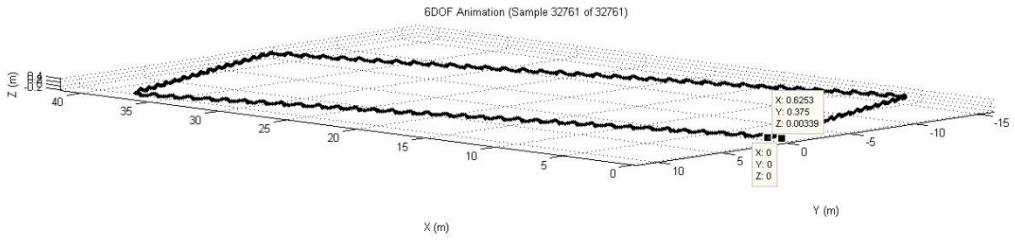


Figure 43 Trajectory of Votey walk side view

From Figure 42 we can see that the trajectory forms a perfect rectangle representing the corridor in Votey Hall. Walker started at [0, 0, 0]. The final point in position is [0.62, 0.37, 0], by calculating the norm of the final position vector, we can see that is only 0.72 meters away from the origin point. This is a 3 minute walk for about 150

meters. Taking the fact that the corridor has a width of about 2 meters in consideration, we can say this estimation is fairly satisfactory.

From Figure 43 we can see that the trajectory is completely constrained in X-Y plane, the range of Z component is [0m, 0.2m] which matches the height of a person lifts his foot while walking. This is also a rather decent result for the sake of X-Y dominant direction.

We can also evaluate the result at another angle by observing the heading difference to four dominant directions respectively. Figure 44 shows the plots of heading difference between the current direction and four dominant directions.

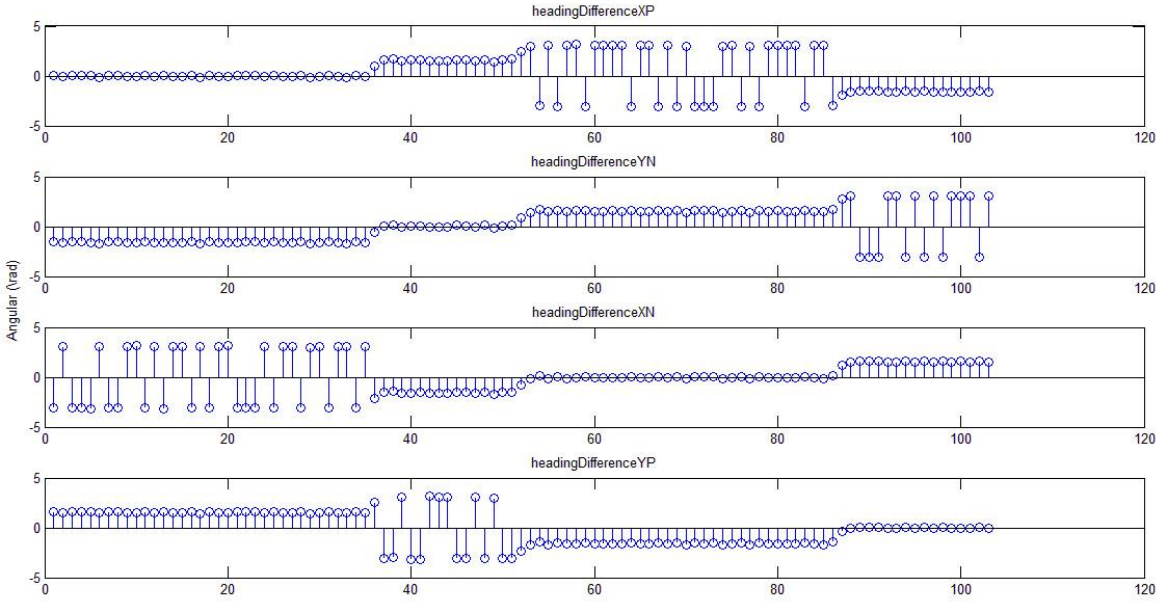


Figure 44 Heading difference to four dominant directions

According to the plots we can see that during the first segment, from the 1st sample to the 36th, heading difference to XP is almost zero, meaning that the direction the walker faced is XP. Difference to YN and YP is a little larger which equals $\pi/4$, that is, 90 degrees, and difference to the XN direction is the largest, $\pi/2$, that is, 180 degrees.

The reason why the plot jumps up and down is that we measure signed angles in right hand coordinates, therefore the angle between the current direction and dominant direction jumps between -180 degrees and 180 degrees back and forth. This is a good sign that proved the current direction is adjusted accurately.

During the second segment, because the walker made a right turn, according to the right hand coordinates rule, now he faces the YN dominant direction. Therefore in this segment, difference to YN is the smallest and to YP is the largest. Also we notice the value jumps between 180 degrees and -180 degrees.

Then we evaluate the error correction performance of EDHE.

Table 2 Performance of EHDE in 2D walk

	Horizontal Error(m)	Vertical Error(m)	Execution Time(s)
No EHDE	16	1.15	10.5
After EHDE	0.7	0	11.3

From Table 2 we can vividly view the power of EHDE. Vertical drift is eliminated and horizontal drift is reduced significantly. At the same time, complexity of the algorithm does not increase much, EHDE trades off 99% of the error with less than 10% more execution time.

Next we observe the performance of EHDE in a more complex motion, climbing stairs. Data is same as the second set of data in section 5.1 Preliminary Results.

Figure 45 shows the position output plot of EHDE.

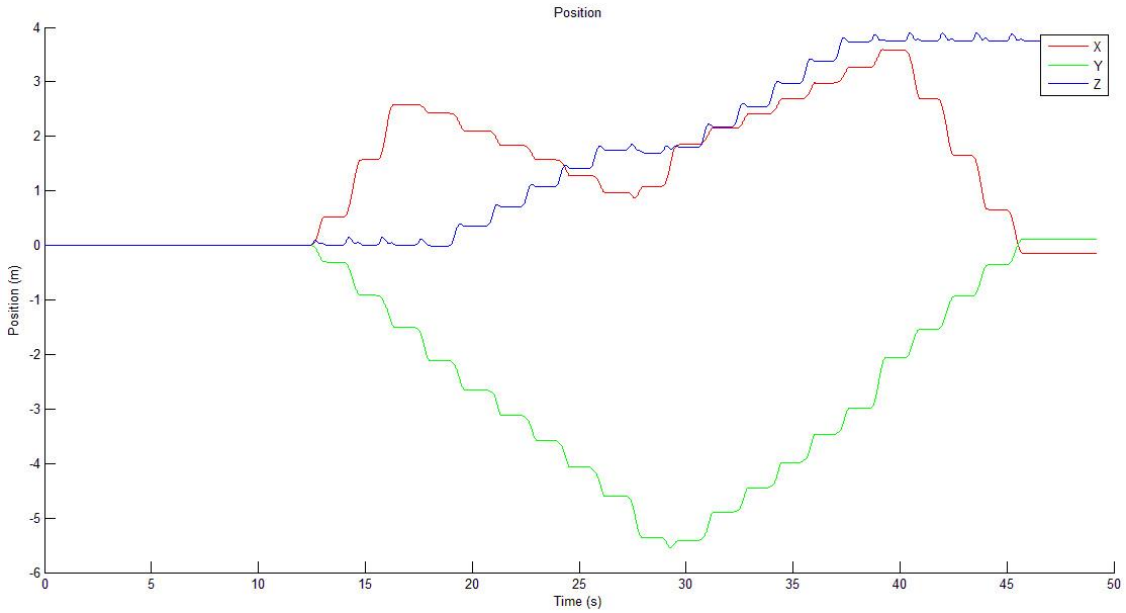


Figure 45 Plot of position data with EHDE for stairs

Position plot looks as decent as that before the process of EHDE. We can easily see the stairs by observing the Z line: first three steps are flat, then five steps going up, then two steps flat, then five steps going up again, then five steps flat. From trajectory plots we can observe minor errors.

First we show a view of trajectory from a cornered view to get an intuitive image of the trajectory.

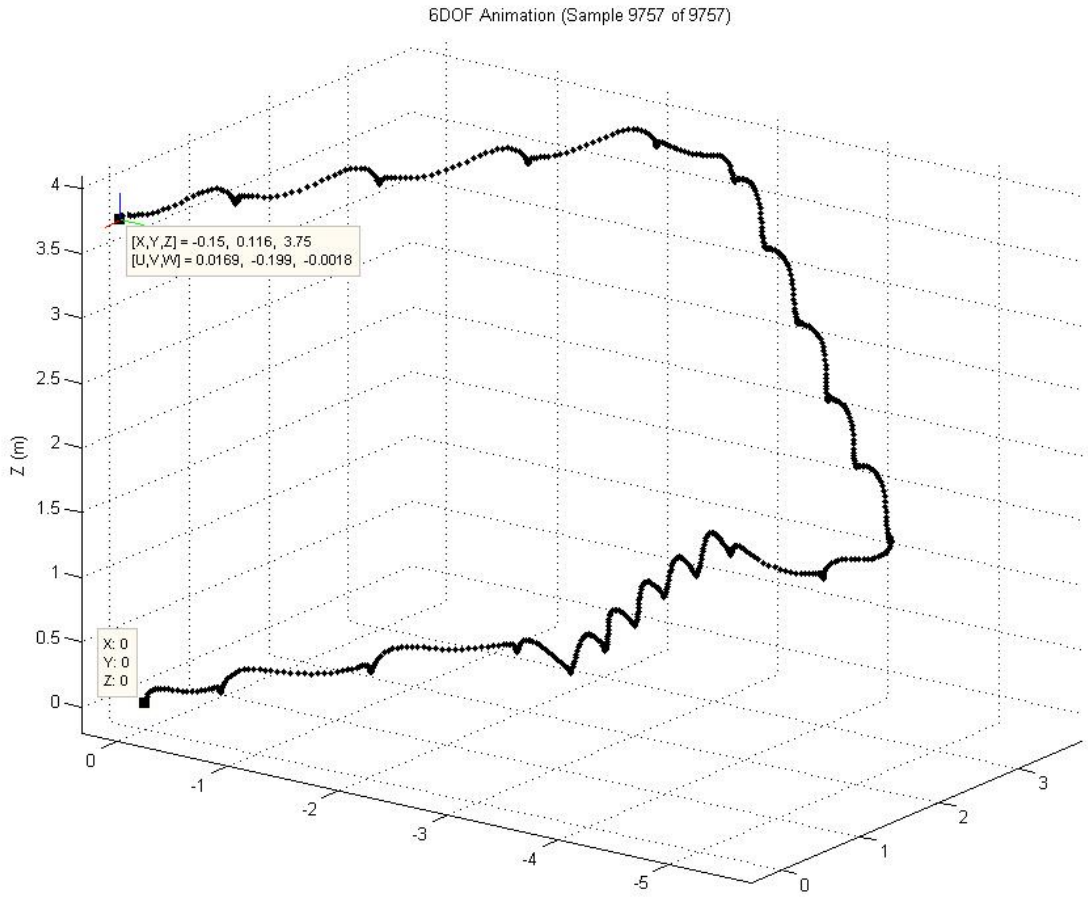


Figure 46 Trajectory of stairs cornered view

Figure 47 and 48 show the top and side view of the trajectory for the stairs dataset.

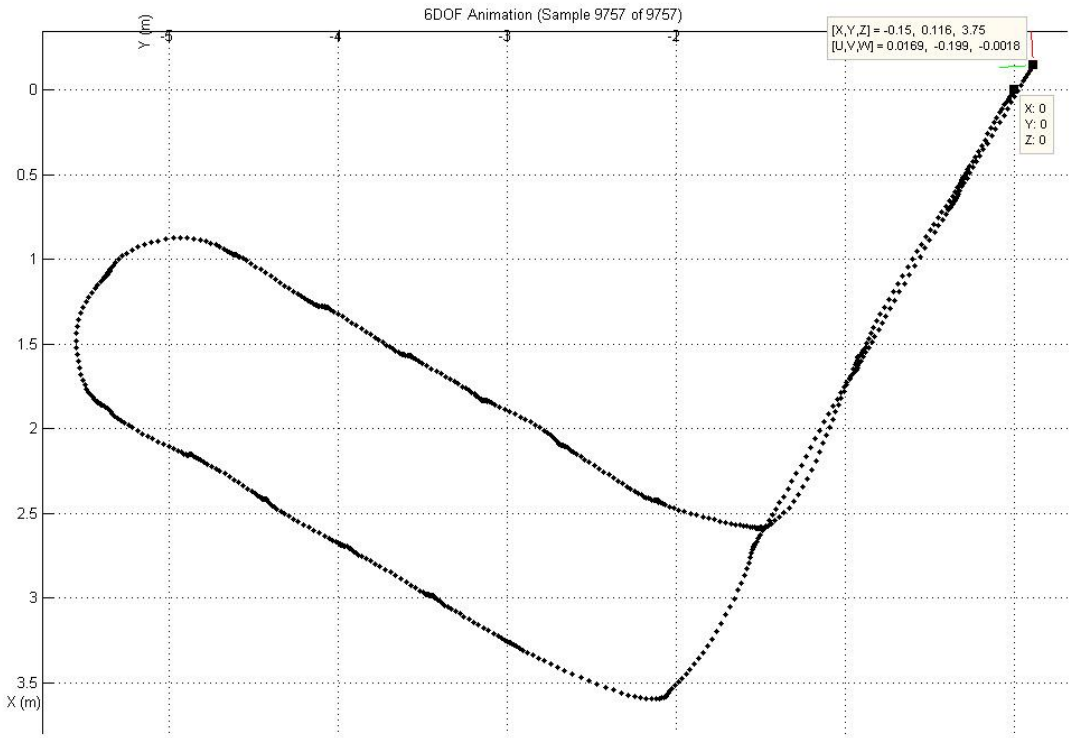


Figure 47 Trajectory of stairs top view

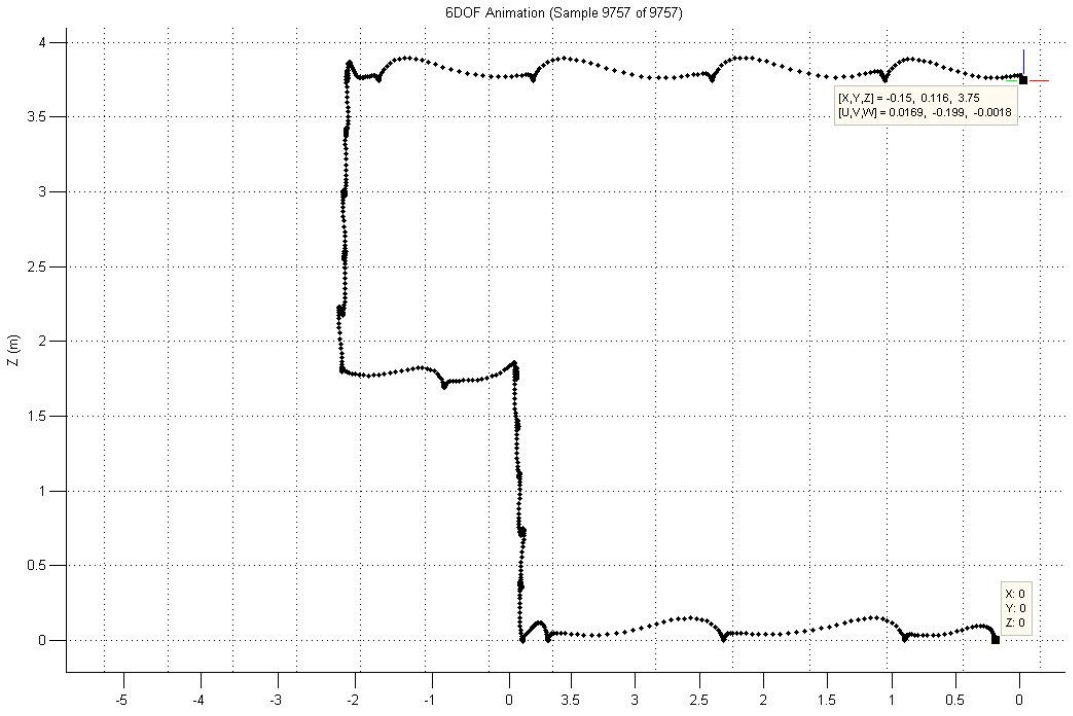


Figure 48 Trajectory of stairs side view

From the top view we can clearly view the difference between it and the one without EHDE drift elimination. With the correction of EHDE, the straight walk on the 2nd and the 3rd floor aligned from top view, while before EHDE, there was a noticeable angle between the two straight walks. This suggests that angle error has been corrected by EHDE.

From the side view we can see that trajectory on stairs is corrected to a straight line that is perpendicular to the flat corridors as well.

In terms of error estimation, because we cannot tell precisely how far the walker walked on the 2nd floor and the 3rd floor, so as long as the two straight walks aligned from top view, we can determine the horizontal error is zero. With regard to vertical error, we measured the actual floor height of Votey Hall which is around 3.6 meters. We subtract the estimated height with actual floor height to get vertical error.

Table 3 Performance of EHDE in 3D walk

	Horizontal Error(m)	Vertical Error(m)	Execution Time(s)
No EHDE	0.8	0.02	4.3
After EHDE	0	0.14	4.8

For a short walk like this, statistics from a table is not very illustrative compared to the actual sight of the trajectory. Nevertheless, we can see the horizontal drift is completely removed. With respect to vertical error, the naive ECF+ZUPT algorithm already does a fairly good job, even though the 0.02m is partly because of coincidence that several errors got cancelled out during calculation. Still, EHDE presents reasonable

result with an error of merely 0.14m. Time complexity increased about 10% which is quite tolerable.

CHAPTER 6: TOTAL SYSTEM FRAMEWORK

In this chapter, we review the whole system and explain the top level implementation that put ECF, ZUPT and EHDE to work together. We also present a UML diagram to show the inheritance relationship of the code.

6.1 Top Level Implementation

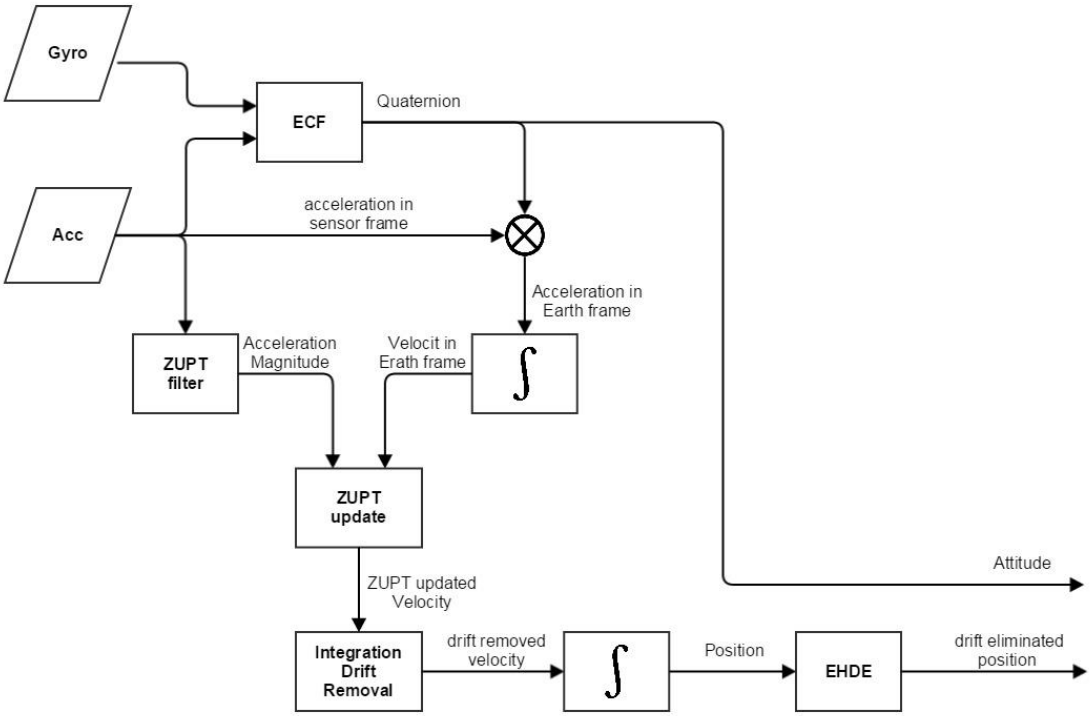


Figure 49 Block diagram of top level algorithm

Flow chart of top level is as shown in Figure 49.

Top level implementation consists of following steps:

Step 1: pass accelerometer data and gyroscope data into explicit complementary filter, and the filter would output quaternions of the whole data series. The quaternions or their transformations into another representation of rotation can represent the attitude

that we desire. In the Matlab code, the quaternions are transformed into rotation matrix so as to generate a 3D trajectory monitor animation, which we have seen in the previous chapters.

Step 2: use the quaternion calculated in step 1 to rotate the accelerometer data from sensor frame to Earth frame

Step 3: integrate the acceleration in Earth frame from step 2 to get velocity in Earth frame

Step 4: pass accelerometer data to the ZUPT filters (a 1^{st} order Butterworth high-pass and a 1^{st} order Butterworth low-pass filter) to get the processed acceleration magnitude and detect stationary periods

Step 5: use the stationary periods to zero update the velocity calculated in step 3 to get ZUPTed velocity in Earth frame

Step 6: Remove the drift during non-stationary periods due to integration

Step 7: do integral on linear velocity calculated in step 6 to get position in Earth frame

Step 8: pass position data in to EHDE to get drift eliminated position estimation is Earth frame.

Thus we obtained drift eliminated displacement from step 8 and attitude estimation from step 1.

Note that within each integral level there is drift reduction techniques. In acceleration level, we implemented ECF that use a PI controller to reduce gyro drift. In velocity level, we implemented ZUPT and another separate procedure to reduce integral drift of acceleration. In displacement level, we implemented the EHDE to eliminate drift

caused by tiny errors from two lower levels that integrated into huge drift. Hence, the total logic framework of the system can be expressed hierarchically as shown is Figure 50.

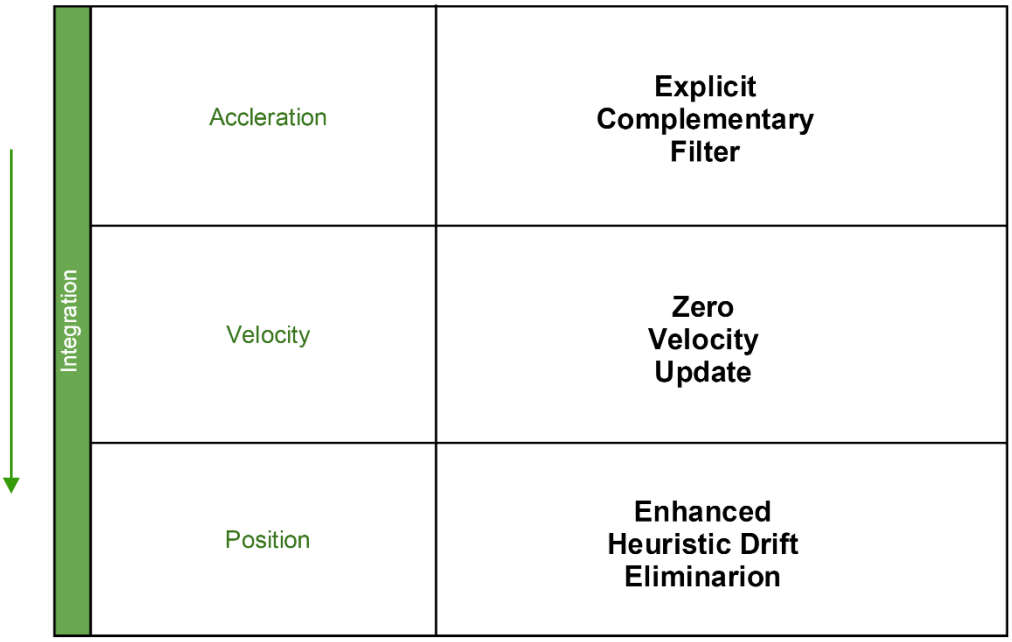


Figure 50 Hierarchical Diagram

Only so can we get linear displacement using a low-cost sensor and a light-weight algorithm, this again proves how severe low-cost MEMS IMUs drift.

6.2 UML diagram

Figure 51 shows that UML diagram of Matlab code that used to implement the algorithm. A UML diagram is used to explain the inheritance relationship among different segments of Matlab code.

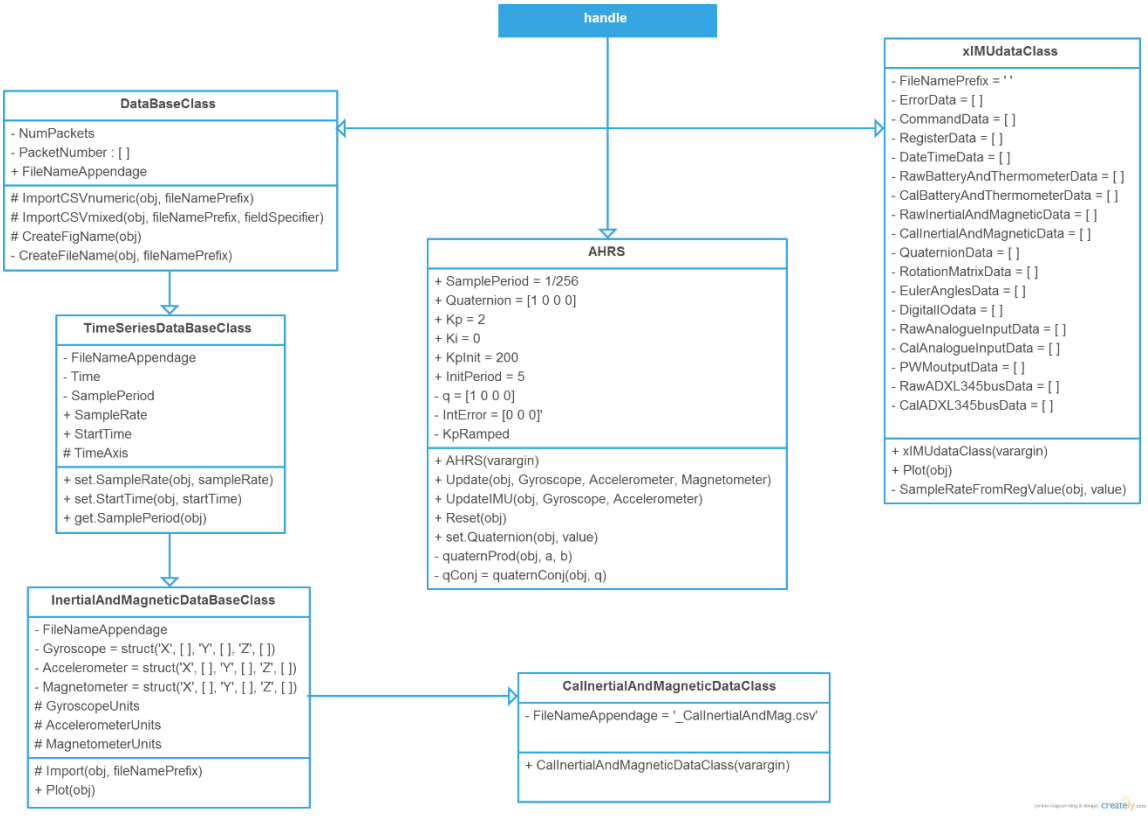


Figure 51 UML diagram

All classes listed above inherit from the “handle” class. Handle class is the basic class that any class needs to inherit from, just like the abstract class in Java.

DataBaseClass verifies if data file exists and opens the file. If files doesn’t exist it sends back an error message. If files exists it scans all the text inside the file. It mainly deals with file issues.

TimeSeriesDataBaseClass inherits from DataBaseClass. It deals with timing issues like setting start time, stop time and sample rate.

InertialAndMagneticDataBaseClass inherits from TimeSeriesDataBaseClass. It categorizes data scanned by DataBaseClass into three parts: accelerometer data, gyroscope data and magnetometer data. It also provides a public method to plot those data.

Finally, CalInertialAndMagneticDataClass inherits from InertialAndMagnetic-DataBaseClass. It adds the property “sample rate” to data object. It also adds units as string variables to accelerometer, gyroscope and magnetometer data.

On the other side, xIMUdataClass takes in a file and set up a data object with complete properties like sample rate, battery and thermometer data, date and time data and etc.

AHRS is the explicit complementary filter that is already explained in Chapter 4. It has default public properties such as $q = [1 \quad 0 \quad 0 \quad 0]$, $k_p = 2$, $k_i = 0$, etc. It is also equipped with an important method called updateIMU which takes accelerometer and gyro data and update the q, and is the core of ECF. Therefore in the top level code, the updateIMU is called inside a loop, every time a new set of data is read in, updateIMU is called to update the quaternion.

Finally the top level code calls the instances of the three classes to make the whole system running.

CHAPTER 7: CONSLUSION

This thesis sets out to construct a low-cost MEMS IMU based position and foot attitude tracking solution that is applicable in hospitals to track patients and observe their walking patterns. And beyond that, this system has the potential to be used in firefighter search, indoor mapping etc. In accomplishing this goal this work seeks to build a light-weight system that can overcome the serious drift issue of low cost inertial sensors. We also strive to present, in one unabridged work, a set of steps for navigation and tracking systems from theory to implementation.

7.1 Contributions

In total this thesis develops a complete and self-contained yet light-weighted system that implements several drift reduction techniques for inertial pedestrian tracking. This includes an explanation from design to principals of the inertial sensor we use, and a model for sensor noises. Also three drift elimination techniques were implemented within three integration levels respectively to push drift reduction to the limit. Also all test data is presented along with the algorithm, this is unique from the many implementations which “design” filters based on values that were tweaked in order to achieve results.

According to the results presented, the EHDE drift elimination approach we proposed is able to eliminate the majority of drift in indoor and structured environment. It removes more than 95% of error with the tradeoff of 10% more time complexity which is fairly worthwhile.

7.2 Limitations

Although the ECF+ZUPT+EDHE system works perfect in the experiments we made, it has some limitations:

First, pedestrian has to start in a straight corridor along the dominant direction in order for the EDHE to compute dominant directions. But this is not a significant problem since dominant directions calculation can be initialized anytime or even input into the EDHE algorithm.

Secondly, EHDE is likely to fail in a complex building or complex walking patterns such as a zigzag pattern. If EDHE cannot find the dominant direction for a long time, the drift during that period can totally deviate from the correct trajectory. This can be avoided theoretically if we use a compass as an extra source of data. But in hospitals, systems like MRI greatly distort the magnetic field that would make magnetically aided system work abnormally.

7.3 Future Research

Pedestrian tracking on a low-cost inertial sensor continues to remain an unsolved problem. Not much research has focused on this topic which means there is plenty of space to improve the system.

First, as is indicated in section 7.2 Limitations, the ECF+ZUPT+EDHE system could fail in a complex environment or outdoors. A magnetically aided EDHE is an area of current research. Magnetometers do not drift over time and hopefully they can provide satisfactory results in a pure magnetic field environment.

Second, the hardware of the sensor is introduced in section 1.2.1 MEMS IMU, the SP-10C is powered by a 5 volts Li-ion battery. The lifetime of the battery is two days per complete charge for 1 hour, which may be troublesome in practical use. According to compressive sensing theory, most signals in real life are “sparse”, that is, most part of its spectrum contains little valuable data. The sparsity of a signal can be exploited to recover it from far fewer samples than required by the Shannon-Nyquist sampling theorem [36]. Therefore if we only sample the data-rich part, the number of samples would be greatly reduced. Less samples are transmitted and thus battery power is saved.

References

1, Liu, Hongbo, et al. "Push the limit of Wi-Fi based localization for smartphones." Proceedings of the 18th annual international conference on Mobile computing and networking. ACM, 2012.

2, Biswas, Joydeep, and Manuela Veloso. "Wi-Fi localization and navigation for autonomous indoor mobile robots." Robotics and Automation (ICRA), 2010 IEEE International Conference on. IEEE, 2010.

3, Byunghun Kim, et al. "A Multi-pronged Approach for Indoor Positioning with Wi-Fi, Magnetic and Cellular Signals." 2014 International Conference on Indoor Positioning and Indoor Navigation, 2014

4, Andrejašic, Matej. "Mems accelerometers." University of Ljubljana. Faculty for mathematics and physics, Department of physics, Seminar. 2008.

5, Maklouf, Othman, and Ahmed Adwaib. "Performance Evaluation of GPS\ INS Main Integration Approach." International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering 8.2 (2014).

6, Borenstein, Johann, and Lauro Ojeda. "Heuristic reduction of gyro drift in gyro-based vehicle tracking." SPIE Defense, Security, and Sensing. International Society for Optics and Photonics, 2009.

7, Neto, Pedro, and J. Norberto Pires. "3-D position estimation from inertial sensing: minimizing the error from the process of double integration of accelerations." Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE. IEEE, 2013.

8, Fischer, Carl, Poorna Talkad Sukumar, and Mike Hazas. "Tutorial: implementation of a pedestrian tracker using foot-mounted inertial sensors." IEEE pervasive computing 12.2 (2013): 17-27.

9, Bistrovs, Vadims, and A. Kluga. "Adaptive extended Kalman filter for aided inertial navigation system." Elektronika ir Elektrotechnika 122.6 (2012): 37-40.

10, Ligorio, Gabriele, and Angelo Maria Sabatini. "Extended Kalman filter-based methods for pose estimation using visual, inertial and magnetic sensors: Comparative analysis and performance evaluation." Sensors 13.2 (2013): 1919-1941.

11, Shin, Eun-Hwan. "Estimation techniques for low-cost inertial navigation." UCGE report 20219 (2005).

12, Mahony, Robert, Tarek Hamel, and Jean-Michel Pflimlin. "Complementary filter design on the special orthogonal group SO (3)." Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on. IEEE, 2005.

13, Schmidt, Stanley F. "The Kalman filter-Its recognition and development for aerospace applications." Journal of Guidance, Control, and Dynamics 4.1 (1981): 4-7.

14, Algrain, Marcelo C., and Jafar Saniie. "Interlaced Kalman filtering of 3D angular motion based on Euler's nonlinear equations." Aerospace and Electronic Systems, IEEE Transactions on 30.1 (1994): 175-185.

15, Algrain, Marcelo C., and Douglas E. Ehlers. "Novel Kalman filtering method for the suppression of gyroscope noise effects in pointing and tracking systems." Optical engineering 34.10 (1995): 3016-3030.

16, Azor, Ruth, Itzhack Y. Bar-Itzhack, and Richard R. Harman. "Satellite angular rate estimation from vector measurements." Journal of Guidance, Control, and Dynamics 21.3 (1998): 450-457.

17, Van Trees, H., and K. Bell. "A Tutorial on Particle Filters for Online Nonlinear/NonGaussian Bayesian Tracking." (2009): 723-737.

18, Gordon, Neil, B. Ristic, and S. Arulampalam. "Beyond the kalman filter: Particle filters for tracking applications." Artech House, London (2004).

19, Julier, Simon J., Jeffrey K. Uhlmann, and Hugh F. Durrant-Whyte. "A new approach for filtering nonlinear systems." American Control Conference, Proceedings of the 1995. Vol. 3. IEEE, 1995.

20, Haykin, Simon. Kalman filtering and neural networks. Vol. 47. John Wiley & Sons, 2004.

21, Markley, F. Landis. "Attitude filtering on SO (3)." The Journal of the Astronautical Sciences 54.3-4 (2006): 391-413.

22, Madgwick, Sebastian OH, Andrew JL Harrison, and Ravi Vaidyanathan. "Estimation of IMU and MARG orientation using a gradient descent algorithm." Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on. IEEE, 2011.

23, Hamel, Tarek, and Robert Mahony. "Attitude estimation on SO [3] based on direct inertial measurements." Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on. IEEE, 2006.

24, Alam, Fakhri, Zhou ZhaiHe, and Hu JiaJia. "A Comparative Analysis of Orientation Estimation Filters using MEMS based IMU."

25, Quoc, Dung Duong, Jinwei Sun, and Nguyen Ngoc Tan. "Sensor Fusion based on Complementary Algorithms using MEMS IMU." International Journal of Signal Processing, Image Processing and Pattern Recognition 8.2 (2015): 313-324.

26, The InvenSense website, http://store.invensense.com/datasheets/invensense/MPU9250REV1.0.pdf, link valid by August, 2015

27, The Sensoplex website, http://www.sensoplex.com/SP-10C.pdf, link valid by August, 2015

28, Judd, Thomas. "A personal dead reckoning module." ION GPS. Vol. 97. 1997.

29, L. Sher, "Personal Inertial Navigation System (PINS)", DARPA, 1996

30, Fischer, Carl, Poorna Talkad Sukumar, and Mike Hazas. "Tutorial: implementation of a pedestrian tracker using foot-mounted inertial sensors." IEEE pervasive computing 12.2 (2013): 17-27.

31, Elwell, John. "Inertial navigation for the urban warrior." AeroSense'99. International Society for Optics and Photonics, 1999.

32, Borenstein, Johann, and Lauro Ojeda. "Heuristic drift elimination for personnel tracking systems." Journal of Navigation 63.04 (2010): 591-606.

33, Jiménez, Antonio Ramón, et al. "Improved heuristic drift elimination with magnetically-aided dominant directions (MiHDE) for pedestrian navigation in complex buildings." Journal of location based services 6.3 (2012): 186-210.

34, MTi, XsensTechnologies, and MTx User Manual. "Technical Documentation." Product Manual. Xsens Co (2006): 2-30.

35, MicroStrain Inc. 3DM-GX3 -25 Miniature Attitude Heading Reference Sensor. 459 Hurricane Lane, Suite 102, Williston, VT 05495 USA, 1.04 edition, 2009.

36, Baraniuk, Richard G. "Compressive sensing." IEEE signal processing magazine 24.4 (2007).

37, Belongie, Serge. "Rodrigues' rotation formula." From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein. http://mathworld. wolfram. com/RodriguesRotationFormula. html (1999).

38, Shoemake, Ken. "Animating rotation with quaternion curves." ACM SIGGRAPH computer graphics. Vol. 19. No. 3. ACM, 1985.

39, Euston, Mark, et al. "A complementary filter for attitude estimation of a fixed-wing UAV." Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. IEEE, 2008.

40, OlliW's Work Sites, http://www.olliw.eu/2013/imu-data-fusing/, link valid by Sep.20th 2015

Appendix

Top level code

```
clear;
close all;
clc;
addpath('Quaternions');
addpath('ximu_matlab_library');

tic;
% ------------------------------------------------------------------------
----
% Select dataset (comment in/out)

filePath = 'Datasets/aroundVotey';
startTime = 0;
stopTime = 177;

% filePath = 'Datasets/aroundVotey1stCorner';
% startTime = 0;
% stopTime = 16.3;

% filePath = 'Datasets/aroundVoteyStraight';
% startTime = 5;
% stopTime = 35;

% filePath = 'Datasets/outsideRecNoMag2';
% startTime = 0;
% stopTime = 136;

% filePath = 'Datasets/VoteyUpStairs';
% startTime = 0;
% stopTime = 49.2;

% filePath = 'Datasets/VoteyUpStairsStepByStep';
% startTime = 0;
% stopTime = 60.9;

% ------------------------------------------------------------------------
----
% Import data

samplePeriod = 1/180;
xIMUdata = xIMUdataClass(filePath, 'InertialMagneticSampleRate',
1/samplePeriod);
time = xIMUdata.CalInertialAndMagneticData.Time;
gyrX = xIMUdata.CalInertialAndMagneticData.Gyroscope.X / 10;
gyrY = xIMUdata.CalInertialAndMagneticData.Gyroscope.Y / 10;
gyrZ = xIMUdata.CalInertialAndMagneticData.Gyroscope.Z / 10;
accX = xIMUdata.CalInertialAndMagneticData.Accelerometer.X / 9.81;
accY = xIMUdata.CalInertialAndMagneticData.Accelerometer.Y / 9.81;
accZ = xIMUdata.CalInertialAndMagneticData.Accelerometer.Z / 9.81;
```

```
magX = xIMUdata.CalInertialAndMagneticData.Magnetometer.X;
magY = xIMUdata.CalInertialAndMagneticData.Magnetometer.Y;
magZ = xIMUdata.CalInertialAndMagneticData.Magnetometer.Z;
clear('xIMUdata');

% ------------------------------------------------------------------
----
% Manually frame data

% startTime = 0;
% stopTime = 10;

indexSel = find(sign(time-startTime)+1, 1) : find(sign(time-
stopTime)+1, 1);
time = time(indexSel);
gyrX = gyrX(indexSel);
gyrY = gyrY(indexSel);
gyrZ = gyrZ(indexSel);
accX = accX(indexSel);
accY = accY(indexSel);
accZ = accZ(indexSel);
magX = magX(indexSel);
magY = magY(indexSel);
magZ = magZ(indexSel);

% ------------------------------------------------------------------
----
% Detect stationary periods

% Compute accelerometer magnitude
acc_mag = sqrt(accX.*accX + accY.*accY + accZ.*accZ);

% HP filter accelerometer data
filtCutOff = 0.0001;
[b, a] = butter(1, (2*filtCutOff)/(1/samplePeriod), 'high');
acc_magFilt = filtfilt(b, a, acc_mag);

% Compute absolute value
acc_magFilt = abs(acc_magFilt);

% LP filter accelerometer data
filtCutOff = 5;
[b, a] = butter(1, (2*filtCutOff)/(1/samplePeriod), 'low');
acc_magFilt = filtfilt(b, a, acc_magFilt);

% Threshold detection
stationary = acc_magFilt < 0.055;


% ------------------------------------------------------------------
----
% Plot data raw sensor data and stationary periods
```

```
figure('Position', [9 39 900 600], 'Number', 'off', 'Name', 'Sensor
Data');
ax(1) = subplot(2,1,1);
    hold on;
    plot(time, gyrX, 'r');
    plot(time, gyrY, 'g');
    plot(time, gyrZ, 'b');
    title('Gyroscope');
    xlabel('Time (s)');
    ylabel('Angular velocity (^\circ/s)');
    legend('X', 'Y', 'Z');
    hold off;
ax(2) = subplot(2,1,2);
    hold on;
    plot(time, accX, 'r');
    plot(time, accY, 'g');
    plot(time, accZ, 'b');
    plot(time, acc_magFilt, ':k');
    plot(time, stationary, 'k', 'LineWidth', 2);
    title('Accelerometer');
    xlabel('Time (s)');
    ylabel('Acceleration (g)');
    legend('X', 'Y', 'Z', 'Filtered', 'Stationary');
    hold off;
linkaxes(ax,'x');

% ------------------------------------------------------------------
----
% Compute orientation

quat = zeros(length(time), 4);
AHRSalgorithm = AHRS('SamplePeriod', samplePeriod, 'Kp', 1, 'KpInit',
1,'Ki',0);

% Initial convergence
initPeriod = 2;
indexSel = 1 : find(sign(time-(time(1)+initPeriod))+1, 1);
for i = 1:2000
    AHRSalgorithm.UpdateIMU([0 0 0], [mean(accX(indexSel))
mean(accY(indexSel)) mean(accZ(indexSel))]);
%     AHRSalgorithm.Update([0 0 0], [mean(accX(indexSel))
mean(accY(indexSel)) mean(accZ(indexSel))],[mean(magX(indexSel))
mean(magY(indexSel)) mean(magZ(indexSel))]);
end

% For all data
for t = 1:length(time)
    if(stationary(t))
        AHRSalgorithm.Kp = 0.5;
    else
        AHRSalgorithm.Kp = 0;
    end
    AHRSalgorithm.UpdateIMU(degtorad([gyrX(t) gyrY(t) gyrZ(t)]),
[accX(t) accY(t) accZ(t)]);
%     AHRSalgorithm.Update(degtorad([gyrX(t) gyrY(t) gyrZ(t)]),
[accX(t) accY(t) accZ(t)],[magX(t) magY(t) magZ(t)]);
```

```
    quat(t,:) = AHRSalgorithm.Quaternion;
end

% ---------------------------------------------------------------------
----
% Compute translational accelerations

% Rotate accelerations from sensor frame to Earth frame
acc = [accX accY accZ];
acc = quaternRotate(acc, quaternConj(quat));


% % Remove gravity from measurements
% acc = acc - [zeros(length(time), 2) ones(length(time), 1)];     %
unnecessary due to velocity integral drift compensation

% Convert acceleration measurements to m/s/s
acc = acc * 9.81;

% Plot translational accelerations
figure('Position', [9 39 900 300], 'Number', 'off', 'Name',
'Accelerations');
hold on;
plot(time, acc(:,1), 'r');
plot(time, acc(:,2), 'g');
plot(time, acc(:,3), 'b');
title('Acceleration');
xlabel('Time (s)');
ylabel('Acceleration (m/s/s)');
legend('X', 'Y', 'Z');
hold off;

% ---------------------------------------------------------------------
----
% Compute translational velocities

acc(:,3) = acc(:,3) - 9.81;

% Integrate acceleration to yield velocity
vel = zeros(size(acc));
for t = 2:length(vel)
    vel(t,:) = vel(t-1,:) + acc(t,:) * samplePeriod;
    if(stationary(t) == 1)
        vel(t,:) = [0 0 0];     % apply ZUPT update when foot
stationary
    end
end


% Compute integral drift during non-stationary periods
velDrift = zeros(size(vel));
stationaryStart = find([0; diff(stationary)] == 1);
stationaryEnd = find([0; diff(stationary)] == -1);
for i = 1:numel(stationaryEnd)
```

```
    driftRate = vel(stationaryStart(i)-1, :) / (stationaryStart(i) -
stationaryEnd(i));
    enum = 1:(stationaryStart(i) - stationaryEnd(i));
    drift = [enum'*driftRate(1) enum'*driftRate(2) enum'*driftRate(3)];
    velDrift(stationaryEnd(i):stationaryStart(i)-1, :) = drift;
end

% Remove integral drift
vel = vel - velDrift;

% vel = vel-vel;

% Plot translational velocity
figure('Position', [9 39 900 300], 'Number', 'off', 'Name',
'Velocity');
hold on;
plot(time, vel(:,1), 'r');
plot(time, vel(:,2), 'g');
plot(time, vel(:,3), 'b');
title('Velocity');
xlabel('Time (s)');
ylabel('Velocity (m/s)');
legend('X', 'Y', 'Z');
hold off;

% ------------------------------------------------------------------------
----
% Compute translational position

% Integrate velocity to yield position
pos = zeros(size(vel));
for t = 2:length(pos)
    pos(t,:) = pos(t-1,:) + vel(t,:) * samplePeriod;    % integrate
velocity to yield position
end

toc;

% get position of the first 5 steps
first5X = pos(stationaryEnd(1:5),1);
first5Y = pos(stationaryEnd(1:5),2);
% use polyfit to calculate best fit straight line for the first five
step and serve as X-Y dominant direction
p = polyfit(first5X,first5Y,1);

% Heuristic Drift Elimination
step = zeros(length(stationaryEnd),3);
headingDifferenceXP = zeros(length(step),1);  % heading difference
between current step and dominant XP direction
headingDifferenceYP = zeros(length(step),1);  % heading difference
between current step and dominant YP direction
headingDifferenceXN = zeros(length(step),1);  % heading difference
between current step and dominant XN direction
headingDifferenceYN = zeros(length(step),1);  % heading difference
between current step and dominant YN direction
```

```
HDEquat = zeros(length(step),4);

% start EHDE correction

for i = 1:length(stationaryEnd)
    % step calculated from position
    step(i,:) = pos(stationaryStart(i),:) - pos(stationaryEnd(i),:);
    % walking on flat floor
    if abs(step(i,3)) < 0.2
        fprintf('into plane correction.\n');

        % dominant direction in 3D determined
        domDirXP = [1,p(1),0];
        domDirYN = [1,-1/p(1),0];
        domDirXN = [-1,-p(1),0];
        domDirYP = [-1,1/p(1),0];
        % heading difference between current step and each of four
dominant directions
        headingDifferenceXP(i) = getAngle(step(i,:),domDirXP);
        headingDifferenceYP(i) = getAngle(step(i,:),domDirYP);
        headingDifferenceXN(i) = getAngle(step(i,:),domDirXN);
        headingDifferenceYN(i) = getAngle(step(i,:),domDirYN);

        % determine which possible zone does current step fall into
        if abs(headingDifferenceXP(i)) < 0.2
            % calculate quaternion from current step to determined
dominant direction
            HDEquat(i,:) = calcQuat(step(i,:), domDirXP);
            % rotate current step tp nearest dominant direction
            pos(stationaryEnd(i):length(pos),:) =
quaternRotate(pos(stationaryEnd(i):length(pos),:),HDEquat(i,:));
            fprintf('plane correction to XP made %dth iteration.\n',i);
            % if not the first step, parallel move the remaining whole
trajectory to the end of previous step
            if i >= 2
                pos(stationaryEnd(i):length(pos),1) =
pos(stationaryEnd(i):length(pos),1) - ( pos(stationaryEnd(i),1) -
pos(stationaryStart(i-1),1) );
                pos(stationaryEnd(i):length(pos),2) =
pos(stationaryEnd(i):length(pos),2) - ( pos(stationaryEnd(i),2) -
pos(stationaryStart(i-1),2) );
                pos(stationaryEnd(i):length(pos),3) =
pos(stationaryEnd(i):length(pos),3) - ( pos(stationaryEnd(i),3) -
pos(stationaryStart(i-1),3) );
            end
        elseif abs(headingDifferenceYP(i)) < 0.2
            HDEquat(i,:) = calcQuat(step(i,:), domDirYP);
            pos(stationaryEnd(i):length(pos),:) =
quaternRotate(pos(stationaryEnd(i):length(pos),:),HDEquat(i,:));
            fprintf('plane correction to YP made %dth iteration.\n',i);
            pos(stationaryEnd(i):length(pos),1) =
pos(stationaryEnd(i):length(pos),1) - ( pos(stationaryEnd(i),1) -
pos(stationaryStart(i-1),1) );
            pos(stationaryEnd(i):length(pos),2) =
pos(stationaryEnd(i):length(pos),2) - ( pos(stationaryEnd(i),2) -
pos(stationaryStart(i-1),2) );
```

```
            pos(stationaryEnd(i):length(pos),3) =
pos(stationaryEnd(i):length(pos),3) - ( pos(stationaryEnd(i),3) -
pos(stationaryStart(i-1),3) );
        elseif abs(headingDifferenceXN(i)) < 0.2
            HDEquat(i,:) = calcQuat(step(i,:), domDirXN);
            fprintf('plane correction to XN made %dth iteration.\n',i);
            pos(stationaryEnd(i):length(pos),:) =
quaternRotate(pos(stationaryEnd(i):length(pos),:),HDEquat(i,:));
            pos(stationaryEnd(i):length(pos),1) =
pos(stationaryEnd(i):length(pos),1) - ( pos(stationaryEnd(i),1) -
pos(stationaryStart(i-1),1) );
            pos(stationaryEnd(i):length(pos),2) =
pos(stationaryEnd(i):length(pos),2) - ( pos(stationaryEnd(i),2) -
pos(stationaryStart(i-1),2) );
            pos(stationaryEnd(i):length(pos),3) =
pos(stationaryEnd(i):length(pos),3) - ( pos(stationaryEnd(i),3) -
pos(stationaryStart(i-1),3) );
        elseif abs(headingDifferenceYN(i)) < 0.2
            HDEquat(i,:) = calcQuat(step(i,:), domDirYN);
            fprintf('plane correction to YN made %dth iteration.\n',i);
            pos(stationaryEnd(i):length(pos),:) =
quaternRotate(pos(stationaryEnd(i):length(pos),:),HDEquat(i,:));
            pos(stationaryEnd(i):length(pos),1) =
pos(stationaryEnd(i):length(pos),1) - ( pos(stationaryEnd(i),1) -
pos(stationaryStart(i-1),1) );
            pos(stationaryEnd(i):length(pos),2) =
pos(stationaryEnd(i):length(pos),2) - ( pos(stationaryEnd(i),2) -
pos(stationaryStart(i-1),2) );
            pos(stationaryEnd(i):length(pos),3) =
pos(stationaryEnd(i):length(pos),3) - ( pos(stationaryEnd(i),3) -
pos(stationaryStart(i-1),3) );
        end
    % if walking along stairs
    elseif abs(step(i,3)) > 0.2
        fprintf('into stair correction.\n');

        % dominant direction in 3D determined, gradient remained
        domDirXP = [1,p(1),step(i,3) * norm([1,p(1)]) /
norm(step(i,1:2))];
        domDirYN = [1,-1/p(1),step(i,3) * norm([1,-1/p(1)]) /
norm(step(i,1:2))];
        domDirXN = [-1,-p(1),step(i,3) * norm([-1,p(1)]) /
norm(step(i,1:2))];
        domDirYP = [-1,1/p(1),step(i,3) * norm([-1,1/p(1)]) /
norm(step(i,1:2))];

        % heading difference between current step and each of four
dominant directions
        headingDifferenceXP(i) = getAngle(step(i,:),domDirXP);
        headingDifferenceYP(i) = getAngle(step(i,:),domDirYP);
        headingDifferenceXN(i) = getAngle(step(i,:),domDirXN);
        headingDifferenceYN(i) = getAngle(step(i,:),domDirYN);

        % determine which possible zone does current step fall into and
correct current step to nearest dominant direction
        if abs(headingDifferenceXP(i)) < 0.4
```

```
            HDEquat(i,:) = calcQuat(step(i,:), domDirXP);
            fprintf('stair correction to XP made %dth iteration.\n',i);
            pos(stationaryEnd(i):length(pos),:) =
quaternRotate(pos(stationaryEnd(i):length(pos),:),HDEquat(i,:));
            pos(stationaryEnd(i):length(pos),1) =
pos(stationaryEnd(i):length(pos),1) - ( pos(stationaryEnd(i),1) -
pos(stationaryStart(i-1),1) );
            pos(stationaryEnd(i):length(pos),2) =
pos(stationaryEnd(i):length(pos),2) - ( pos(stationaryEnd(i),2) -
pos(stationaryStart(i-1),2) );
            pos(stationaryEnd(i):length(pos),3) =
pos(stationaryEnd(i):length(pos),3) - ( pos(stationaryEnd(i),3) -
pos(stationaryStart(i-1),3) );
        elseif abs(headingDifferenceYP(i)) < 0.4
            HDEquat(i,:) = calcQuat(step(i,:), domDirYP);
            fprintf('stair correction to YP made %dth iteration.\n',i);
            pos(stationaryEnd(i):length(pos),:) =
quaternRotate(pos(stationaryEnd(i):length(pos),:),HDEquat(i,:));
            pos(stationaryEnd(i):length(pos),1) =
pos(stationaryEnd(i):length(pos),1) - ( pos(stationaryEnd(i),1) -
pos(stationaryStart(i-1),1) );
            pos(stationaryEnd(i):length(pos),2) =
pos(stationaryEnd(i):length(pos),2) - ( pos(stationaryEnd(i),2) -
pos(stationaryStart(i-1),2) );
            pos(stationaryEnd(i):length(pos),3) =
pos(stationaryEnd(i):length(pos),3) - ( pos(stationaryEnd(i),3) -
pos(stationaryStart(i-1),3) );
        elseif abs(headingDifferenceXN(i)) < 0.4
            HDEquat(i,:) = calcQuat(step(i,:), domDirXN);
            fprintf('stair correction to XN made %dth iteration.\n',i);
            pos(stationaryEnd(i):length(pos),:) =
quaternRotate(pos(stationaryEnd(i):length(pos),:),HDEquat(i,:));
            pos(stationaryEnd(i):length(pos),1) =
pos(stationaryEnd(i):length(pos),1) - ( pos(stationaryEnd(i),1) -
pos(stationaryStart(i-1),1) );
            pos(stationaryEnd(i):length(pos),2) =
pos(stationaryEnd(i):length(pos),2) - ( pos(stationaryEnd(i),2) -
pos(stationaryStart(i-1),2) );
            pos(stationaryEnd(i):length(pos),3) =
pos(stationaryEnd(i):length(pos),3) - ( pos(stationaryEnd(i),3) -
pos(stationaryStart(i-1),3) );
        elseif abs(headingDifferenceYN(i)) < 0.4
            HDEquat(i,:) = calcQuat(step(i,:), domDirYN);
            fprintf('stair correction to YN made %dth iteration.\n',i);
            pos(stationaryEnd(i):length(pos),:) =
quaternRotate(pos(stationaryEnd(i):length(pos),:),HDEquat(i,:));
            pos(stationaryEnd(i):length(pos),1) =
pos(stationaryEnd(i):length(pos),1) - ( pos(stationaryEnd(i),1) -
pos(stationaryStart(i-1),1) );
            pos(stationaryEnd(i):length(pos),2) =
pos(stationaryEnd(i):length(pos),2) - ( pos(stationaryEnd(i),2) -
pos(stationaryStart(i-1),2) );
            pos(stationaryEnd(i):length(pos),3) =
pos(stationaryEnd(i):length(pos),3) - ( pos(stationaryEnd(i),3) -
pos(stationaryStart(i-1),3) );
        end
    end
```

```
end



% % calculate heading difference between curret step and previous step
in right hand coordinate
% headingBetweenSteps = zeros(length(step)-1,1);
% for i = 1:length(headingBetweenSteps)
%     headingBetweenSteps(i) = getAngle(step(i,:),step(i+1,:));
% end;


% Plot translational position
figure('Position', [9 39 900 600], 'Number', 'off', 'Name',
'Position');
hold on;
plot(time, pos(:,1), 'r');
plot(time, pos(:,2), 'g');
plot(time, pos(:,3), 'b');
title('Position');
xlabel('Time (s)');
ylabel('Position (m)');
legend('X', 'Y', 'Z');
hold off;

% Plot quaternion
figure;
plot(quat);
legend('1', '2', '3', '4');
% ------------------------------------------------------------------------
----
% Plot 3D foot trajectory

% % Remove stationary periods from data to plot
% posPlot = pos(find(~stationary), :);
% quatPlot = quat(find(~stationary), :);
posPlot = pos;
quatPlot = quat;

% Extend final sample to delay end of animation
extraTime = 5;
onesVector = ones(extraTime*(1/samplePeriod), 1);
posPlot = [posPlot; [posPlot(end, 1)*onesVector, posPlot(end,
2)*onesVector, posPlot(end, 3)*onesVector]];
quatPlot = [quatPlot; [quatPlot(end, 1)*onesVector, quatPlot(end,
2)*onesVector, quatPlot(end, 3)*onesVector, quatPlot(end,
4)*onesVector]];

toc;

% Create 6 DOF animation
SamplePlotFreq = 30;
Spin = 50;
SixDofAnimation(posPlot, quatern2rotMat(quatPlot), ...
```

```
                'SamplePlotFreq', SamplePlotFreq, 'Trail',
'DotsOnly', ...
                'Position', [9 39 1280 768], 'View',
[(100:(Spin/(length(posPlot)-1)):(100+Spin))', 10*ones(length(posPlot),
1)], ...
                'AxisLength', 0.2, 'ShowArrowHead', false, ...
                'Xlabel', 'X (m)', 'Ylabel', 'Y (m)', 'Zlabel', 'Z
(m)', 'ShowLegend', false, ...
                'CreateAVI', false, 'AVIfileNameEnum', true,
'AVIfileName', '9Dof', 'AVIfps', ((1/samplePeriod) / SamplePlotFreq));
```

Explicit complementary filter implementation

```
classdef AHRS < handle

    %% Public properties
    properties (Access = public)
        SamplePeriod = 1/256;
        Quaternion = [1 0 0 0];     % output quaternion describing the
sensor relative to the Earth
        Kp = 2;                     % proportional gain
        Ki = 0;                     % integral gain
        KpInit = 200;               % proportional gain used during
initialisation
        InitPeriod = 5;             % initialisation period in seconds
        Error = [0 0 0];
    end

    %% Private properties
    properties (Access = private)
        q = [1 0 0 0];              % internal quaternion describing
the Earth relative to the sensor
        IntError = [0 0 0]';        % integral error
        KpRamped;                   % internal proportional gain used
to ramp during initialisation
    end

    %% Public methods
    methods (Access = public)
        function obj = AHRS(varargin)
            for i = 1:2:nargin
                if  strcmp(varargin{i}, 'SamplePeriod'),
obj.SamplePeriod = varargin{i+1};
                elseif  strcmp(varargin{i}, 'Quaternion')
                    obj.Quaternion = varargin{i+1};
                    obj.q = quaternConj(obj.Quaternion);
                elseif  strcmp(varargin{i}, 'Kp'), obj.Kp =
varargin{i+1};
                elseif  strcmp(varargin{i}, 'Ki'), obj.Ki =
varargin{i+1};
                elseif  strcmp(varargin{i}, 'KpInit'), obj.KpInit =
varargin{i+1};
```

```
                elseif  strcmp(varargin{i}, 'InitPeriod'),
obj.InitPeriod = varargin{i+1};
                else error('Invalid argument');
                end
                obj.KpRamped = obj.KpInit;
            end;
        end
        function obj = Update(obj, Gyroscope, Accelerometer,
Magnetometer)
            quater = obj.Quaternion; % short name local variable for
readability

            % Normalise accelerometer measurement
            if(norm(Accelerometer) == 0), return; end   % handle NaN
            Accelerometer = Accelerometer / norm(Accelerometer);    %
normalise magnitude

            % Normalise magnetometer measurement
            if(norm(Magnetometer) == 0), return; end    % handle NaN
            Magnetometer = Magnetometer / norm(Magnetometer);   %
normalise magnitude

            % Reference direction of Earth's magnetic feild
            h = quaternProd(quater, quaternProd([0 Magnetometer],
quaternConj(quater)));
            b = [0 norm([h(2) h(3)]) 0 h(4)];

            % Estimated direction of gravity and magnetic field
            v = [2*(quater(2)*quater(4) - quater(1)*quater(3))
                 2*(quater(1)*quater(2) + quater(3)*quater(4))
                 quater(1)^2 - quater(2)^2 - quater(3)^2 +
quater(4)^2];
            w = [2*b(2)*(0.5 - quater(3)^2 - quater(4)^2) +
2*b(4)*(quater(2)*quater(4) - quater(1)*quater(3))
                 2*b(2)*(quater(2)*quater(3) - quater(1)*quater(4)) +
2*b(4)*(quater(1)*quater(2) + quater(3)*quater(4))
                 2*b(2)*(quater(1)*quater(3) + quater(2)*quater(4)) +
2*b(4)*(0.5 - quater(2)^2 - quater(3)^2)];

            % Error is sum of cross product between estimated direction
and measured direction of fields
            e = cross(Accelerometer, v) + cross(Magnetometer, w);

            obj.IntError = obj.IntError + e';
%             if(obj.Ki > 0)
%                 obj.IntError = obj.IntError + e * obj.SamplePeriod;
%             else
%                 obj.IntError = [0 0 0];
%             end

            % Apply feedback terms
            Gyroscope = Gyroscope + obj.Kp * e + obj.Ki *
obj.IntError';

            % Compute rate of change of quaternion
```

```
            qDot = 0.5 * quaternProd(quater, [0 Gyroscope(1)
Gyroscope(2) Gyroscope(3)]);

            % Integrate to yield quaternion
            quater = quater + qDot * obj.SamplePeriod;
            obj.Quaternion = quater / norm(quater); % normalise
quaternion

            obj.Quaternion = obj.quaternConj(obj.Quaternion);
        end
        function obj = UpdateIMU(obj, Gyroscope, Accelerometer)

            % Normalise accelerometer measurement
            if(norm(Accelerometer) ==
0)                                          % handle NaN
                warning(0, 'Accelerometer magnitude is zero.  Algorithm
update aborted.');
                return;
            else
                Accelerometer = Accelerometer /
norm(Accelerometer);                % normalise measurement
            end

            % Compute error between estimated and measured direction of
gravity
            v = [2*(obj.q(2)*obj.q(4) - obj.q(1)*obj.q(3))
                2*(obj.q(1)*obj.q(2) + obj.q(3)*obj.q(4))
                obj.q(1)^2 - obj.q(2)^2 - obj.q(3)^2 +
obj.q(4)^2];                    % estimated direction of gravity
            error = cross(v, Accelerometer');

%               % Compute ramped Kp value used during init period
%             if(obj.KpRamped > obj.Kp)
%                 obj.IntError = [0 0 0]';
%                 obj.KpRamped = obj.KpRamped - (obj.KpInit - obj.Kp) /
(obj.InitPeriod / obj.SamplePeriod);
%
else
% init period complete
%                 obj.KpRamped = obj.Kp;
                obj.IntError = obj.IntError +
error;                                  % compute integral feedback terms
(only outside of init period)
%             end

            % Apply feedback terms
            Ref = Gyroscope - (obj.Kp*error + obj.Ki*obj.IntError)';

            % Compute rate of change of quaternion
            pDot = 0.5 * obj.quaternProd(obj.q, [0 Ref(1) Ref(2)
Ref(3)]);          % compute rate of change of quaternion
            obj.q = obj.q + pDot *
obj.SamplePeriod;                                % integrate rate of
change of quaternion
```

```
            obj.q = obj.q /
norm(obj.q);                                                % normalise
quaternion

            % Store conjugate
            obj.Quaternion = obj.quaternConj(obj.q);
        end
        function obj = Reset(obj)
            obj.KpRamped = obj.KpInit;      % start Kp ramp-down
            obj.IntError = [0 0 0]';        % reset integral terms
            obj.q = [1 0 0 0];              % set quaternion to
alignment
        end
%         function obj = StepDownKp(obj, Kp)
%             obj.KpRamped = Kp;
%             obj.Kp = Kp;
%         end
    end

    %% Get/set methods
    methods
        function obj = set.Quaternion(obj, value)
            if(norm(value) == 0)
                error('Quaternion magnitude cannot be zero.');
            end
            value = value / norm(value);
            obj.Quaternion = value;
            obj.q = obj.quaternConj(value);
        end
    end

    %% Private methods
    methods (Access = private)
        function ab = quaternProd(obj, a, b)
            ab(:,1) = a(:,1).*b(:,1)-a(:,2).*b(:,2)-a(:,3).*b(:,3)-
a(:,4).*b(:,4);
            ab(:,2) = a(:,1).*b(:,2)+a(:,2).*b(:,1)+a(:,3).*b(:,4)-
a(:,4).*b(:,3);
            ab(:,3) = a(:,1).*b(:,3)-
a(:,2).*b(:,4)+a(:,3).*b(:,1)+a(:,4).*b(:,2);
            ab(:,4) = a(:,1).*b(:,4)+a(:,2).*b(:,3)-
a(:,3).*b(:,2)+a(:,4).*b(:,1);
        end
        function qConj = quaternConj(obj, q)
            qConj = [q(:,1) -q(:,2) -q(:,3) -q(:,4)];
        end
    end
end
```

xIMUdataClass

```
classdef xIMUdataClass < handle
```

```
    %% Public properties
    properties (SetAccess = private)
        FileNamePrefix = '';
        ErrorData = [];
        CommandData = [];
        RegisterData = [];
        DateTimeData = [];
        RawBatteryAndThermometerData = [];
        CalBatteryAndThermometerData = [];
        RawInertialAndMagneticData = [];
        CalInertialAndMagneticData = [];
        QuaternionData = [];
        RotationMatrixData = [];
        EulerAnglesData = [];
        DigitalIOdata = [];
        RawAnalogueInputData = [];
        CalAnalogueInputData = [];
        PWMoutputData = [];
        RawADXL345busData = [];
        CalADXL345busData = [];
    end

    %% Public methods
    methods (Access = public)
        function obj = xIMUdataClass(varargin)
            % Create data objects from files
            obj.FileNamePrefix = varargin{1};
            dataImported = false;
            try obj.ErrorData = ErrorDataClass(obj.FileNamePrefix);
dataImported = true; catch e, end
            try obj.CommandData = CommandDataClass(obj.FileNamePrefix);
dataImported = true; catch e, end
            try obj.RegisterData =
RegisterDataClass(obj.FileNamePrefix); dataImported = true; catch e,
end
            try obj.DateTimeData =
DateTimeDataClass(obj.FileNamePrefix); dataImported = true; catch e,
end
            try obj.RawBatteryAndThermometerData =
RawBatteryAndThermometerDataClass(obj.FileNamePrefix); dataImported =
true; catch e, end
            try obj.CalBatteryAndThermometerData =
CalBatteryAndThermometerDataClass(obj.FileNamePrefix); dataImported =
true; catch e, end
            try obj.RawInertialAndMagneticData =
RawInertialAndMagneticDataClass(obj.FileNamePrefix); dataImported =
true; catch e, end
            try obj.CalInertialAndMagneticData =
CalInertialAndMagneticDataClass(obj.FileNamePrefix); dataImported =
true; catch e, end
            try obj.QuaternionData =
QuaternionDataClass(obj.FileNamePrefix); dataImported = true; catch e,
end
            try obj.EulerAnglesData =
EulerAnglesDataClass(obj.FileNamePrefix); dataImported = true; catch e,
end
```

```
            try obj.RotationMatrixData =
RotationMatrixDataClass(obj.FileNamePrefix); dataImported = true; catch
e, end
            try obj.DigitalIOdata =
DigitalIOdataClass(obj.FileNamePrefix); dataImported = true; catch e,
end
            try obj.RawAnalogueInputData =
RawAnalogueInputDataClass(obj.FileNamePrefix); dataImported = true;
catch e, end
            try obj.CalAnalogueInputData =
CalAnalogueInputDataClass(obj.FileNamePrefix); dataImported = true;
catch e, end
            try obj.PWMoutputData =
PWMoutputDataClass(obj.FileNamePrefix); dataImported = true; catch e,
end
            try obj.RawADXL345busData =
RawADXL345busDataClass(obj.FileNamePrefix); dataImported = true; catch
e, end
            try obj.CalADXL345busData =
CalADXL345busDataClass(obj.FileNamePrefix); dataImported = true; catch
e, end
            if(~dataImported)
                error('No data was imported.');
            end

            % Apply SampleRate from register data
            try h = obj.DateTimeData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(67));
catch e, end
            try h = obj.RawBatteryAndThermometerData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(68));
catch e, end
            try h = obj.CalBatteryAndThermometerData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(68));
catch e, end
            try h = obj.RawInertialAndMagneticData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(69));
catch e, end
            try h = obj.CalInertialAndMagneticData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(69));
catch e, end
            try h = obj.QuaternionData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(70));
catch e, end
            try h = obj.RotationMatrixData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(70));
catch e, end
            try h = obj.EulerAnglesData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(70));
catch e, end
            try h = obj.DigitalIOdata; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(78));
catch e, end
            try h = obj.RawAnalogueInputData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(80));
catch e, end
```

```
            try h = obj.CalAnalogueInputData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(80));
catch e, end
            try h = obj.RawADXL345busData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(85));
catch e, end
            try h = obj.CalADXL345busData; h.SampleRate =
obj.SampleRateFromRegValue(obj.RegisterData.GetValueAtAddress(85));
catch e, end

            % Apply SampleRate if specified as argument
            for i = 2:2:(nargin)
                if strcmp(varargin{i}, 'DateTimeSampleRate')
                    try h = obj.DateTimeData; h.SampleRate =
varargin{i+1}; catch e, end
                elseif strcmp(varargin{i}, 'BattThermSampleRate')
                    try h = obj.RawBatteryAndThermometerData;
h.SampleRate = varargin{i+1}; catch e, end
                    try h = obj.CalBatteryAndThermometerData;
h.SampleRate = varargin{i+1}; catch e, end
                elseif strcmp(varargin{i},
'InertialMagneticSampleRate')
                    try h = obj.RawInertialAndMagneticData;
h.SampleRate = varargin{i+1}; catch e, end
                    try h = obj.CalInertialAndMagneticData;
h.SampleRate = varargin{i+1}; catch e, end
                elseif strcmp(varargin{i}, 'QuaternionSampleRate')
                    try h = obj.QuaternionData; h.SampleRate =
varargin{i+1}; catch e, end
                    try h = obj.RotationMatrixData.SampleRate;
h.SampleRate = varargin{i+1}; catch e, end
                    try h = obj.EulerAnglesData; h.SampleRate =
varargin{i+1}; catch e, end
                elseif strcmp(varargin{i}, 'DigitalIOSampleRate')
                    try h = obj.DigitalIOdata; h.SampleRate =
varargin{i+1}; catch e, end
                elseif strcmp(varargin{i}, 'AnalogueInputSampleRate')
                    try h = obj.RawAnalogueInputData; h.SampleRate =
varargin{i+1}; catch e, end
                    try h = obj.CalAnalogueInputData; h.SampleRate =
varargin{i+1}; catch e, end
                elseif strcmp(varargin{i}, 'ADXL345SampleRate')
                    try h = obj.RawADXL345busData; h.SampleRate =
varargin{i+1}; catch e, end
                    try h = obj.CalADXL345busData; h.SampleRate =
varargin{i+1}; catch e, end
                else
                    error('Invalid argument.');
                end
            end
        end
        function obj = Plot(obj)
            try obj.RawBatteryAndThermometerData.Plot(); catch e, end
            try obj.CalBatteryAndThermometerData.Plot(); catch e, end
            try obj.RawInertialAndMagneticData.Plot(); catch e, end
            try obj.CalInertialAndMagneticData.Plot(); catch e, end
            try obj.QuaternionData.Plot(); catch e, end
```

```
            try obj.EulerAnglesData.Plot(); catch e, end
            try obj.RotationMatrixDataClass.Plot(); catch e, end
            try obj.DigitalIOdata.Plot(); catch e, end
            try obj.RawAnalogueInputData.Plot(); catch e, end
            try obj.CalAnalogueInputData.Plot(); catch e, end
            try obj.RawADXL345busData.Plot(); catch e, end
            try obj.CalADXL345busData.Plot(); catch e, end
        end
    end

    %% Private methods
    methods (Access = private)
        function sampleRate = SampleRateFromRegValue(obj, value)
            sampleRate = floor(2^(value-1));
        end
    end
end
```

DataBaseClass

```
classdef DataBaseClass < handle

    %% Abstract public 'read-only' properties
    properties (Abstract, SetAccess = private)
        FileNameAppendage;
    end

    %% Public 'read-only' properties
    properties (SetAccess = private)
        NumPackets = 0;
        PacketNumber = [];
    end

    %% Protected methods
    methods (Access = protected)
        function data = ImportCSVnumeric(obj, fileNamePrefix)
            data = dlmread(obj.CreateFileName(fileNamePrefix), ',', 1,
0);
            obj.PacketNumber = data(:,1);
            obj.NumPackets = length(obj.PacketNumber);
        end
        function data = ImportCSVmixed(obj, fileNamePrefix,
fieldSpecifier)
            fid = fopen(obj.CreateFileName(fileNamePrefix));
            fgets(fid);     % disregard column headings
            data = textscan(fid, fieldSpecifier, 'Delimiter', ',');
            fclose(fid);
            obj.PacketNumber = data{1};
            obj.NumPackets = length(obj.PacketNumber);
        end
        function figName = CreateFigName(obj)
            [pathstr, name , ext, versn] =
fileparts(obj.FileNameAppendage);
            figName = name(2:end);
        end
```

```
    end

    %% Private methods
    methods (Access = private)
        function fileName = CreateFileName(obj, fileNamePrefix)
            fileName = strcat(fileNamePrefix, obj.FileNameAppendage);
            if(~exist(fileName, 'file'))
                error('File not found. No data was imported.');
            end
        end
    end
end
```

TimeSeriesDataBaseClass

```
classdef TimeSeriesDataBaseClass < DataBaseClass

    %% Abstract public 'read-only' properties
    properties (Abstract, SetAccess = private)
        FileNameAppendage;
    end

    %% Public 'read-only' properties
    properties (SetAccess = private)
        Time = [];
        SamplePeriod = 0;
    end

    %% Public properties
    properties (Access = public)
        SampleRate = 0;
        StartTime = 0;
    end

    %% Protected properties
    properties (Access = protected)
        TimeAxis;
    end

    %% Abstract public methods
    methods (Abstract, Access = public)
        Plot(obj);
    end

    %% Get/set methods
    methods
        function obj = set.SampleRate(obj, sampleRate)
            obj.SampleRate = sampleRate;
```

```
            if(obj.SampleRate == 0)
                obj.Time = [];
                obj.TimeAxis = 'Sample';
            elseif(obj.NumPackets ~= 0)
                obj.Time = (0:obj.NumPackets-1)' * (1/obj.SampleRate) +
obj.StartTime;
                obj.TimeAxis = 'Time (s)';
            end
        end
        function obj = set.StartTime(obj, startTime)
            obj.StartTime = startTime;
            obj.SampleRate = obj.SampleRate;
        end
        function samplePeriod = get.SamplePeriod(obj)
            if(obj.SampleRate == 0)
                samplePeriod = 0;
            else
                samplePeriod = 1 / obj.SampleRate;
            end
        end
    end
end
```

InertialAndMagneticDataBaseClass

```
classdef InertialAndMagneticDataBaseClass < TimeSeriesDataBaseClass

    %% Abstract public 'read-only' properties
    properties (Abstract, SetAccess = private)
        FileNameAppendage;
    end

    %% Public 'read-only' properties
    properties (SetAccess = private)
        Gyroscope = struct('X', [], 'Y', [], 'Z', []);
        Accelerometer = struct('X', [], 'Y', [], 'Z', []);
        Magnetometer = struct('X', [], 'Y', [], 'Z', []);
    end

    %% Abstract protected properties
    properties (Access = protected)
        GyroscopeUnits;
        AccelerometerUnits;
        MagnetometerUnits;
    end

    %% Protected methods
    methods (Access = protected)
        function obj = Import(obj, fileNamePrefix)
            data = obj.ImportCSVnumeric(fileNamePrefix);
            obj.Gyroscope.X = data(:,2);
            obj.Gyroscope.Y = data(:,3);
            obj.Gyroscope.Z = data(:,4);
```

```
            obj.Accelerometer.X = data(:,5);
            obj.Accelerometer.Y = data(:,6);
            obj.Accelerometer.Z = data(:,7);
            obj.Magnetometer.X = data(:,8);
            obj.Magnetometer.Y = data(:,9);
            obj.Magnetometer.Z = data(:,10);
            obj.SampleRate = obj.SampleRate;    % call set method to
create time vector
        end
    end

    %% Public methods
    methods (Access = public)
        function fig = Plot(obj)
            if(obj.NumPackets == 0)
                error('No data to plot.');
            else
                if(isempty(obj.Time))
                    time = 1:obj.NumPackets;
                else
                    time = obj.Time;
                end
                fig = figure('Name', obj.CreateFigName());
                ax(1) = subplot(3,1,1);
                hold on;
                plot(time, obj.Gyroscope.X, 'r');
                plot(time, obj.Gyroscope.Y, 'g');
                plot(time, obj.Gyroscope.Z, 'b');
                legend('X', 'Y', 'Z');
                xlabel(obj.TimeAxis);
                ylabel(strcat('Angular rate (', obj.GyroscopeUnits,
')'));
                title('Gyroscope');
                hold off;
                ax(2) = subplot(3,1,2);
                hold on;
                plot(time, obj.Accelerometer.X, 'r');
                plot(time, obj.Accelerometer.Y, 'g');
                plot(time, obj.Accelerometer.Z, 'b');
                legend('X', 'Y', 'Z');
                xlabel(obj.TimeAxis);
                ylabel(strcat('Acceleration (', obj.AccelerometerUnits,
')'));
                title('Accelerometer');
                hold off;
                ax(3) = subplot(3,1,3);
                hold on;
                plot(time, obj.Magnetometer.X, 'r');
                plot(time, obj.Magnetometer.Y, 'g');
                plot(time, obj.Magnetometer.Z, 'b');
                legend('X', 'Y', 'Z');
                xlabel(obj.TimeAxis);
                ylabel(strcat('Flux (', obj.MagnetometerUnits, ')'));
                title('Magnetometer');
                hold off;
                linkaxes(ax,'x');
            end
```

```
        end
    end
end
```

CalInertialAndMagneticDataClass

```
classdef CalInertialAndMagneticDataClass <
InertialAndMagneticDataBaseClass

    %% Public 'read-only' properties
    properties (SetAccess = private)
        FileNameAppendage = '_CalInertialAndMag.csv';
    end

    %% Public methods
    methods (Access = public)
        function obj = CalInertialAndMagneticDataClass(varargin)
            fileNamePrefix = varargin{1};
            for i = 2:2:nargin
                if  strcmp(varargin{i}, 'SampleRate'), obj.SampleRate =
varargin{i+1};
                else error('Invalid argument.');
                end
            end
            obj.Import(fileNamePrefix);

            % Set protected parent class variables
            obj.GyroscopeUnits = '^\circ/s';
            obj.AccelerometerUnits = 'g';
            obj.MagnetometerUnits = 'G';
        end
    end
end
```

Calculate quaternion between two vectors

```
function quat = calcQuat(v1,v2)
   a = cross(v1,v2);
   b = acos(dot(v1,v2)/(norm(v1)*norm(v2)));
   quat = [cos(b/2) a];
   k = sqrt((quat(2)^2 + quat(3)^2 + quat(4)^2)/(1-quat(1)^2));
   quat = [cos(b/2) a/k];
%    b = sqrt((norm(v1))^2 + (norm(v2))^2) + dot(v1,v2);
%    quat = [b a];
%    quat = quat/norm(quat);
end
```

Calculate signed angle in 3D right hand coordinates between two vectors

```
function angle = getAngle(v1,v2)
   a = dot(v1,v2);
   b = norm(v1) * norm(v2);


%    angle = acos(a/b);
%    if angle > pi/2
%        angle = pi-angle;
%    end


   temp = cross(v1,v2);
   if temp(3)>0
       angle = acos(a/b);
   else
       angle = -acos(a/b);
   end
end
```

Calculate conjugate of a quaternion

```
function qConj = quaternConj(q)
    qConj = [q(:,1) -q(:,2) -q(:,3) -q(:,4)];
end
```

Calculate product of two quaternions

```
function ab = quaternProd(a, b)
    ab(:,1) = a(:,1).*b(:,1)-a(:,2).*b(:,2)-a(:,3).*b(:,3)-
a(:,4).*b(:,4);
    ab(:,2) = a(:,1).*b(:,2)+a(:,2).*b(:,1)+a(:,3).*b(:,4)-
a(:,4).*b(:,3);
    ab(:,3) = a(:,1).*b(:,3)-
a(:,2).*b(:,4)+a(:,3).*b(:,1)+a(:,4).*b(:,2);
    ab(:,4) = a(:,1).*b(:,4)+a(:,2).*b(:,3)-
a(:,3).*b(:,2)+a(:,4).*b(:,1);
end
```

Use a quaternion to rotate a N×3 matrix

```
function v = quaternRotate(v, q)
    [row col] = size(v);
    v0XYZ = quaternProd(quaternProd(q, [zeros(row, 1) v]),
quaternConj(q));
    v = v0XYZ(:, 2:4);
end
```

Quaternion representation to rotation matrix

```
function R = quatern2rotMat(q)
    [rows cols] = size(q);
    R = zeros(3,3, rows);
    R(1,1,:) = 2.*q(:,1).^2-1+2.*q(:,2).^2;
    R(1,2,:) = 2.*(q(:,2).*q(:,3)+q(:,1).*q(:,4));
    R(1,3,:) = 2.*(q(:,2).*q(:,4)-q(:,1).*q(:,3));
    R(2,1,:) = 2.*(q(:,2).*q(:,3)-q(:,1).*q(:,4));
    R(2,2,:) = 2.*q(:,1).^2-1+2.*q(:,3).^2;
    R(2,3,:) = 2.*(q(:,3).*q(:,4)+q(:,1).*q(:,2));
    R(3,1,:) = 2.*(q(:,2).*q(:,4)+q(:,1).*q(:,3));
    R(3,2,:) = 2.*(q(:,3).*q(:,4)-q(:,1).*q(:,2));
    R(3,3,:) = 2.*q(:,1).^2-1+2.*q(:,4).^2;
end
```

Generate 3D animation

```
function fig = SixDOFanimation(varargin)

    %% Create local variables

    % Required arguments
    p = varargin{1};                % position of body
    R = varargin{2};                % rotation matrix of body
    [numSamples dummy] = size(p);

    % Default values of optional arguments
    SamplePlotFreq = 1;
    Trail = 'Off';
    LimitRatio = 1;
    Position = [];
    FullScreen = false;
    View = [30 20];
```

```
    AxisLength = 1;
    ShowArrowHead = 'on';
    Xlabel = 'X';
    Ylabel = 'Y';
    Zlabel = 'Z';
    Title = '6DOF Animation';
    ShowLegend = true;
    CreateAVI = false;
    AVIfileName = '6DOF Animation';
    AVIfileNameEnum = true;
    fps = 30;

    for i = 3:2:nargin
        if  strcmp(varargin{i}, 'SamplePlotFreq'), SamplePlotFreq =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'Trail')
            Trail = varargin{i+1};
            if(~strcmp(Trail, 'Off') && ~strcmp(Trail, 'DotsOnly') &&
~strcmp(Trail, 'All'))
                error('Invalid argument.  Trail must be ''Off'',
''DotsOnly'' or ''All''.');
            end
        elseif  strcmp(varargin{i}, 'LimitRatio'), LimitRatio =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'Position'), Position =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'FullScreen'), FullScreen =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'View'), View = varargin{i+1};
        elseif  strcmp(varargin{i}, 'AxisLength'), AxisLength =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'ShowArrowHead'), ShowArrowHead =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'Xlabel'), Xlabel = varargin{i+1};
        elseif  strcmp(varargin{i}, 'Ylabel'), Ylabel = varargin{i+1};
        elseif  strcmp(varargin{i}, 'Zlabel'), Zlabel = varargin{i+1};
        elseif  strcmp(varargin{i}, 'Title'), Title = varargin{i+1};
        elseif  strcmp(varargin{i}, 'ShowLegend'), ShowLegend =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'CreateAVI'), CreateAVI =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'AVIfileName'), AVIfileName =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'AVIfileNameEnum'), AVIfileNameEnum
= varargin{i+1};
        elseif  strcmp(varargin{i}, 'AVIfps'), fps = varargin{i+1};
        else error('Invalid argument.');
        end
    end;

    %% Reduce data to samples to plot only

    p = p(1:SamplePlotFreq:numSamples, :);
    R = R(:, :, 1:SamplePlotFreq:numSamples) * AxisLength;
    if(numel(View) > 2)
        View = View(1:SamplePlotFreq:numSamples, :);
```

```
    end
    [numPlotSamples dummy] = size(p);

    %% Setup AVI file

    aviobj =
[];                                                                    %
create null object
    if(CreateAVI)
        fileName = strcat(AVIfileName, '.mp4');
        if(exist(fileName, 'file'))

if(AVIfileNameEnum)                                                    %
if file name exists and enum enabled
                i = 0;
                while(exist(fileName,
'file'))                                  % find un-used file name by
appending enum
                    fileName = strcat(AVIfileName, sprintf('%i', i),
'.mp4');
                    i = i + 1;
                end

else                                                                   %
else file name exists and enum disabled
                fileName =
[];                                                 % file will not be
created
            end
        end
        if(isempty(fileName))
            sprintf('MP4 file not created as file already exists.')
        else
            aviobj = VideoWriter(fileName,'MPEG-4');
            aviobj.FrameRate = fps;
            open(aviobj);
        end
    end

    %% Setup figure and plot

    % Create figure
    fig = figure('Number', 'off', 'Name', '6DOF Animation');
    if(FullScreen)
        screenSize = get(0, 'ScreenSize');
        set(fig, 'Position', [0 0 screenSize(3) screenSize(4)]);
    elseif(~isempty(Position))
        set(fig, 'Position', Position);
    end
    set(gca, 'drawmode', 'fast');
    lighting phong;
    set(gcf, 'Renderer', 'zbuffer');
    hold on;
    axis equal;
    grid on;
    view(View(1, 1), View(1, 2));
```

```
    title(i);
    xlabel(Xlabel);
    ylabel(Ylabel);
    zlabel(Zlabel);

    % Create plot data arrays
    if(strcmp(Trail, 'DotsOnly') || strcmp(Trail, 'All'))
        x = zeros(numPlotSamples, 1);
        y = zeros(numPlotSamples, 1);
        z = zeros(numPlotSamples, 1);
    end
    if(strcmp(Trail, 'All'))
        ox = zeros(numPlotSamples, 1);
        oy = zeros(numPlotSamples, 1);
        oz = zeros(numPlotSamples, 1);
        ux = zeros(numPlotSamples, 1);
        vx = zeros(numPlotSamples, 1);
        wx = zeros(numPlotSamples, 1);
        uy = zeros(numPlotSamples, 1);
        vy = zeros(numPlotSamples, 1);
        wy = zeros(numPlotSamples, 1);
        uz = zeros(numPlotSamples, 1);
        vz = zeros(numPlotSamples, 1);
        wz = zeros(numPlotSamples, 1);
    end
    x(1) = p(1,1);
    y(1) = p(1,2);
    z(1) = p(1,3);
    ox(1) = x(1);
    oy(1) = y(1);
    oz(1) = z(1);
    ux(1) = R(1,1,1:1);
    vx(1) = R(2,1,1:1);
    wx(1) = R(3,1,1:1);
    uy(1) = R(1,2,1:1);
    vy(1) = R(2,2,1:1);
    wy(1) = R(3,2,1:1);
    uz(1) = R(1,3,1:1);
    vz(1) = R(2,3,1:1);
    wz(1) = R(3,3,1:1);

    % Create graphics handles
    orgHandle = plot3(x, y, z, 'k.');
    if(ShowArrowHead)
        ShowArrowHeadStr = 'on';
    else
        ShowArrowHeadStr = 'off';
    end
    quivXhandle = quiver3(ox, oy, oz, ux, vx, wx,  'r',
'ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize', 0.999999,
'AutoScale', 'off');
    quivYhandle = quiver3(ox, oy, oz, uy, vy, wy,  'g',
'ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize', 0.999999,
'AutoScale', 'off');
```

```
    quivZhandle = quiver3(ox, ox, oz, uz, vz, wz,  'b',
'ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize', 0.999999,
'AutoScale', 'off');

    % Create legend
    if(ShowLegend)
        legend('Origin', 'X', 'Y', 'Z');
    end

    % Set initial limits
    Xlim = [x(1)-AxisLength x(1)+AxisLength] * LimitRatio;
    Ylim = [y(1)-AxisLength y(1)+AxisLength] * LimitRatio;
    Zlim = [z(1)-AxisLength z(1)+AxisLength] * LimitRatio;
    set(gca, 'Xlim', Xlim, 'Ylim', Ylim, 'Zlim', Zlim);

    % Set initial view
    view(View(1, :));

    %% Plot one sample at a time

    for i = 1:numPlotSamples

        % Update graph title
        if(strcmp(Title, ''))
            titleText = sprintf('Sample %i of %i', 1+((i-
1)*SamplePlotFreq), numSamples);
        else
            titleText = strcat(Title, ' (', sprintf('Sample %i of %i',
1+((i-1)*SamplePlotFreq), numSamples), ')');
        end
        title(titleText);

        % Plot body x y z axes
        if(strcmp(Trail, 'DotsOnly') || strcmp(Trail, 'All'))
            x(1:i) = p(1:i,1);
            y(1:i) = p(1:i,2);
            z(1:i) = p(1:i,3);
        else
            x = p(i,1);
            y = p(i,2);
            z = p(i,3);
        end
        if(strcmp(Trail, 'All'))
            ox(1:i) = p(1:i,1);
            oy(1:i) = p(1:i,2);
            oz(1:i) = p(1:i,3);
            ux(1:i) = R(1,1,1:i);
            vx(1:i) = R(2,1,1:i);
            wx(1:i) = R(3,1,1:i);
            uy(1:i) = R(1,2,1:i);
            vy(1:i) = R(2,2,1:i);
            wy(1:i) = R(3,2,1:i);
            uz(1:i) = R(1,3,1:i);
            vz(1:i) = R(2,3,1:i);
            wz(1:i) = R(3,3,1:i);
```

```
        else
            ox = p(i,1);
            oy = p(i,2);
            oz = p(i,3);
            ux = R(1,1,i);
            vx = R(2,1,i);
            wx = R(3,1,i);
            uy = R(1,2,i);
            vy = R(2,2,i);
            wy = R(3,2,i);
            uz = R(1,3,i);
            vz = R(2,3,i);
            wz = R(3,3,i);
        end
        set(orgHandle, 'xdata', x, 'ydata', y, 'zdata', z);
        set(quivXhandle, 'xdata', ox, 'ydata', oy, 'zdata', oz,'udata',
ux, 'vdata', vx, 'wdata', wx);
        set(quivYhandle, 'xdata', ox, 'ydata', oy, 'zdata', oz,'udata',
uy, 'vdata', vy, 'wdata', wy);
        set(quivZhandle, 'xdata', ox, 'ydata', oy, 'zdata', oz,'udata',
uz, 'vdata', vz, 'wdata', wz);

        % Adjust axes for snug fit and draw
        axisLimChanged = false;
        if((p(i,1) - AxisLength) < Xlim(1)), Xlim(1) = p(i,1) -
LimitRatio*AxisLength; axisLimChanged = true; end
        if((p(i,2) - AxisLength) < Ylim(1)), Ylim(1) = p(i,2) -
LimitRatio*AxisLength; axisLimChanged = true; end
        if((p(i,3) - AxisLength) < Zlim(1)), Zlim(1) = p(i,3) -
LimitRatio*AxisLength; axisLimChanged = true; end
        if((p(i,1) + AxisLength) > Xlim(2)), Xlim(2) = p(i,1) +
LimitRatio*AxisLength; axisLimChanged = true; end
        if((p(i,2) + AxisLength) > Ylim(2)), Ylim(2) = p(i,2) +
LimitRatio*AxisLength; axisLimChanged = true; end
        if((p(i,3) + AxisLength) > Zlim(2)), Zlim(2) = p(i,3) +
LimitRatio*AxisLength; axisLimChanged = true; end
        if(axisLimChanged), set(gca, 'Xlim', Xlim, 'Ylim', Ylim,
'Zlim', Zlim); end
        drawnow;

        % Adjust view
        if(numel(View) > 2)
            view(View(i, :));
        end

        % Add frame to AVI object
        if(~isempty(aviobj))
            frame = getframe(fig);
            writeVideo(aviobj, frame);
        end

    end

    hold off;

    % Close AVI file
```

```
    if(~isempty(aviobj))
        close(aviobj);
    end

end
```