UNIVERSITY OF GÄVLE

FACULTY OF ENGINEERING AND SUSTAINABLE DEVELOPMENT

GPS/INS Combination for a Beam Tracking System

Sheng Zhang

September 2011

Master's Thesis in Electronics

Master's Program in Electronics/Telecommunications

Examiner: Jos é Chilo

Supervisor: Ting Jung Liang (Technical University Dresden)

Abstract

In recent years, Land vehicle navigation system (LVNS) technology is a subject of great interest due to its potential for both consumer and business vehicle markets. GPS/INS (Global Positioning System/ Inertial Navigation System) integrated system is an effective solution to realize the LVNS. And how to keep communication between the vehicle and satellite while the vehicle is running in a bad environment is the main task in this thesis.

The thesis provides an introduction to beam tracking system and two algorithms of how to improve the performance, then compare these two algorithms and choose the suitable one and implement it on ArduPilotMega board using Arduino language, at last test the integrated GPS/INS system in practice in order to estimate the performance.

The requirements of the project are the maximum angular speed and angular acceleration speed of the vehicle are $1000^o / s$ and $1500^o / s^2$, respectively. Two algorithms which are Direction Cosine Matrix (DCM) and Euler Angle are evaluated in the system. In this thesis, there are many rotations due to the hostile environment, and DCM algorithm is not suitable for the requirement according to the results of simulation. Therefore, an innovated method which is Euler Angle Algorithm can be one effective way to solve the probelm.

The primary idea of Euler Angle algorithm is to calculate the difference between the reference direction vector and the measurement direction vector from GPS and accelerometers, once there is an error rotation, take the cross product and rotate the incorrect direction vector back to original direction. The simulation results show that by implementing EA algorithm, system requirements can be achievable with a 10kHz update rate antenna and a 4000Hz sampling rate gyroscope, also with EA implementation in ArduPilotMega board, the real system tracking ability can be enhanced effectively.

Acknowledgments

I would like to express my great appreciation to my distinguished supervisor, Dr. Ting Jung Liang, for his continued support, guidance and encouragement throughout my thesis work. His enlighten is very important to my current and future work. Thanks also to the members in Vodafone Chair Communication System at Technical University Dresden for their assistance in programming, device debugging as well as for the many fruitful discussions throughout my thesis work. Especially, I would like to acknowledge Mr. Tom Ritschel for his help of programming by using Arduino language.

I would further like extend my appreciation to Dr. José Chilo, for his patient reading, and pertinent comments. This thesis would not have been possible without these help and guidance.

Most of all, I would like to thank my parents, for their love, encouragement and giving without reservation through all of my so many years of study. This work would not have been possible without their support.

Dedication

To my parents

Contents

List of Figures

List of Tables

CHAPTER 1 Introduction

1.1 Background and Objectives

Navigation is a very ancient skill or art which has become a complex science. It is essentially about travel and finding the way from one place to another and there are a variety of means by which this may be achieved.[1] In the recent years, Land Vehicle Navigation System (LVNS) has been a major focus for research. A land vehicle navigation system is adapted to display the present position of a vehicle when it is driving in different environment.

There are many technologies to determine the position of a vehicle, out of which two are most widely used. The first one is the Global Positioning System (GPS) which provides location and time information in all weather. The second one is an Inertial Navigation System (INS), which is autonomous system, and provides continuous direction and position information from an Inertial Measurement Unit (IMU). GPS, by itself, is sometimes affected by signal block. It requires line-of-sight (LOS) between the satellites and the receiver antenna of the vehicle. However, the LOS criteria may not always be met, the GPS signals will attenuate in mountainous areas, thick forest, and some areas with high-rises buildings. Thus, in general GPS cannot be solely used for navigation, especially in complex land topography.

Unlike GPS, an INS is an independent system which provides velocity and position information though measurement by an IMU. The advantage of INS is its impregnability from external electromagnetic signals, and its ability to working in all kinds of environment in order to provide continuous navigation information with high accuracy. However, there are many sensors such as gyros included in INS. The external noise and temperature effect of these sensors will cause a time-dependent drift as time goes on.

With the decreasing cost of inertial measurement units (IMU), the integration of the Global Positioning System (GPS) and an Inertial Navigation Systems (INS) become more feasible for high-accuracy navigation. Their combination not only offers the accuracy and continuity in the solution, but also enhances the reliability of the system. GPS can restrict the INS's drift over time, and allows for online estimation of the sensor errors, while the inertial devices can bridge the position estimates when there is no GPS signal reception. Also, the use of INS allows the GPS measurements to be compared against statistical limits and reject those measurements that are beyond the limits, thus enhancing the reliability and stability of the whole system. The fundamental integration of GPS and INS is shown as Figure 1.1.

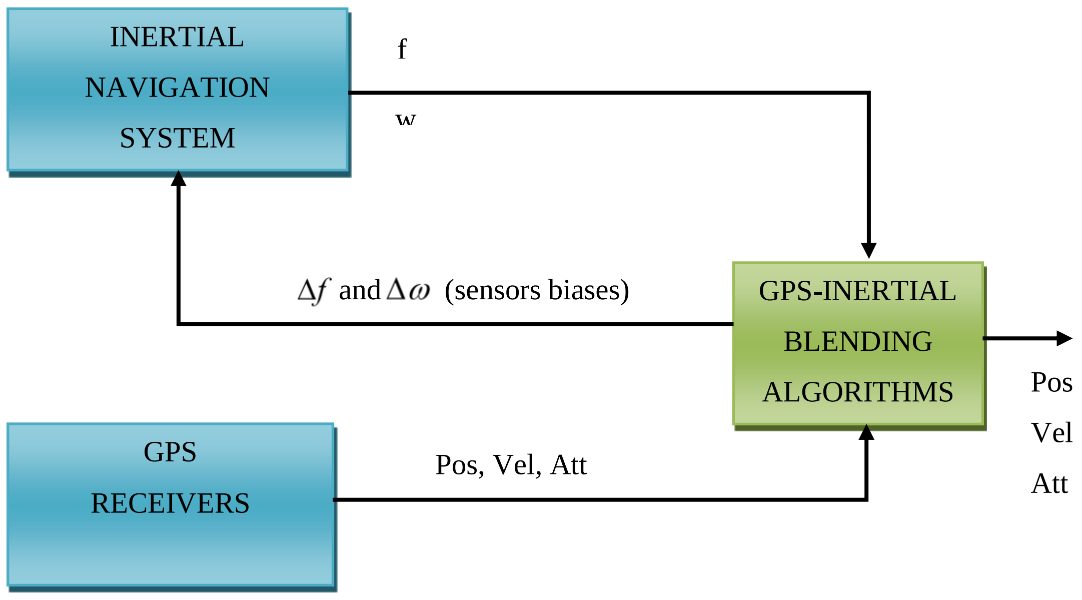

INERTIAL NAVIGATION SYSTEM
f
w
Δf and $\Delta \omega$ (sensors biases)
GPS-INERTIAL BLENDING ALGORITHMS
Pos
Vel
Att
GPS RECEIVERS
Pos, Vel, Att

Figure 1.1 Structure of a GPS/ INS integrated system

The objective of this thesis is to use realizable algorithm to improve the inaccuracy of the GPS/INS when the vehicle is running in a poor road conditions. The integration of GPS and INS has been successfully used in practice during the past decades. However, much of the work has focused on the case of good road conditions, which means there is not so much jolt when the vehicle is running. Under this condition, the antenna of GPS/INS could aim at the satellite perfectly, and the data from the gyros, accelerometers are reliable and the measure is easy. But if the condition of roads is poor, in other words, the vehicle bumped along the rough mountain road or the desert, there should be relatively large rotation. Thus the antenna is more difficult to automatically aim to the satellite and the measured data has an error. This will lead to navigation inaccuracy or unknown noise. How to keep communicating between the antenna and satellite is the main task.

1.2 Problem Statement

The Vodafone Chair Communication System at Technical University Dresden wants to design a GPS/INS system which is applied to the land vehicle navigation. In the real world, people usually drive their car in a bad environment, for example bumped along the rough mountain road or the desert. In order to realize the function of the navigation, keep communicating with satellite to receive the position and velocity information is really important. The vehicle would bump and turn off quickly especially under the off-road conditions, and the antenna of the system in the vehicle would also bump, so how to make the antenna in the system always aim at the satellite in order to keep commnuication is the problem need to study firstly. Furthermore, the measured data from INS is not accurate due to the external noise and temperature effect in gyro sensors, the error of the system will increase to infinite as time goes on, then how to cancel the in order to restore the accuracy of the measured data is also the problem need to be solved.

The main task of the thesis is divide into two parts. First is design gyro signals which can mimic the poor road condition. Using Direction Consine Matrix (DCM) algorithm and Euler angle algorithm to correct the error, and compare them with each other to deside which one is better. All of these will be implemented on MATLAB. The second is to implement one algorithm on ArduPilotMega board using Arduino language, and test the integrated GPS/INS system in practice in order to estimate the performance. The flow chart of the thesis work is shown as follows(Figure 1.2):

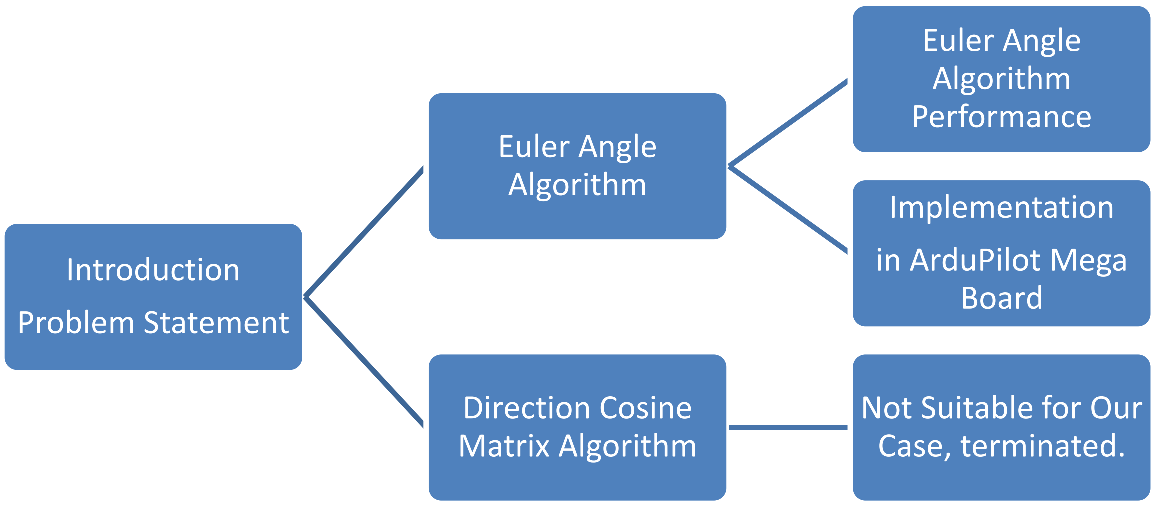

Introduction
Problem Statement
Euler Angle Algorithm
Direction Cosine Matrix Algorithm
Euler Angle Algorithm Performance
Implementation in ArduPilot Mega Board
Not Suitable for Our Case, terminated.

Figure 1.2 Flow Chart of the Main Thesis Work

1.3 Thesis Outline

The thesis comprises of the following framework:

- **Chapter 1**: Introduction – Presents the general introduction of the GPS/INS integrated system and the description of thesis problem statement. The main task has been presented according to the problem statement.

- **Chapter 2** : Fundamentals of GPS/INS – Provides an overview of the GPS and INS system. Given brief introduction of equations of motion, inertial sensor errors, and described the system specification, along with a discussion of its hardware and software platform.

- **Chapter 3** : Provides an overview of DCM algorithm and Euler Angle algorithm. DCM algorithm was applied in unmanned aircraft by DIY DRONES engineers, and Euler Angle algorithm is an innovation in this thesis after estimating DCM algorithm's performance in the same case.

- **Chapter 4** : Estimate the performances of DCM algorithm and Euler Angle algorithm. Compare them with each other, the Euler Angle algorithm will improve the performance for compensating errors in the INS mechanized navigation solutions during GPS outages.

- **Chapter 5** : Implement the Euler Angle algorithm into ArduPilot Mega Board (GPS/INS), and measure it in practice, compare the result with the one got from the simulation on MATLAB.

- **Chapter 6** : Summarizes the work presented in this thesis, and give conclusions from the simulation and practice test results and analysis. Finally, several recommendations for future work are outlined.

CHPATER 2 Fundamentals of GPS and INS

In this chapter, the fundamentals of GPS and INS and also the design requirements are introduced. According to the design requirements, the corresponding rotation error variance is calculated to be a goal for this system.

2.1 Global Positioning System

The Global Positioning System (GPS) is part of a satellite-based navigation system developed by the U.S. Department of Defense under its NAVSTAR satellite program.

2.1.1 GPS Orbits

The fully operational GPS includes 31 active satellites approximately uniformly dispersed around six circular orbits with four or more satellites each. The orbits are inclined at an angle of 55°relative to the equator and are separated from each other by multiples of 60°right ascension. The orbits are non-geostationary and approximately circular, with radii of 26560 km and orbital periods one one-half sidereal day. Theoretically, three or more GPS satellites will always be visible form most points on the earth’s surface, and four or more GPS satellites can be used to determine an observer’s position anywhere on the earth’s surface 24 hours per day.

2.1.2 Applications

While originally a military project, GPS is considered be to be applied to a military application. However, nowadays GPS has become a widely deployed and effective tool for civilian applications, such as commerce, tracking, scientific uses, and navigation. Especially the vehicle navigation system, which allows drivers to receive navigation information when they don’t know the direction. Usually, the system takes its map data which can be replaced when the vehicle moves from one geographical location to another.

2.2 Inertial Navigation System

The operation of inertial navigation systems depends upon the laws of classical mechanics as formulated by Sir Isaac Newton. Newton’s laws tell us that the motion of a body will continue uniformly in a straight line unless disturbed by an external force acting on the body. The laws also tell us that this force will produce a proportional acceleration of the body. Given the ability to measure that acceleration, it would be possible to calculate the change in velocity and position by performing successive mathematical integrations of the acceleration with respect to time. Acceleration can be determined using a device known as an accelerometer.

In order to navigate with respect to our inertial reference frame, it is necessary to keep track of the direction in which the accelerometers are pointing. Rotational motion of the body with respect to the inertial reference frame may be sensed using gyroscopic sensors and used to determine the orientation of the accelerometers at all times. Hence, the inertial navigation is the process whereby the measurements provided by gyroscopes and accelerometers are used to determine the position of the vehicle in which they are installed.

However, the computation process is more complicated than it sounds. The dynamic information about position and velocity is obtained from a hardware which is called Inertial Measurement Unit (IMU). An IMU includes of three gyroscopes and three accelerometers which are anchored on an orthogonal triad. It provide the angular speed and angular acceleration in a coordinate frame different than the coordinate frame in which GPS information is provided. Thus the two different frames will be transform to an appropriate frame, prior to integration[3].

2.2.1 Coordinate Frames and Transformations

Fundamental to the process of inertial navigation is the precise definition of a number of Cartesian co-ordinate reference frames. Each frame is an orthogonal, right-handed, coordinate frame or axis set. The following coordinate frames are introduced:

The inertial frame (i-frame) has its origin at the center of the Earth and axes which are non-rotating with respect to the fixed stars, defined by the axes Ox_i, Oy_i, Oz_i with Oz_i coincident with the Earth's polar axis which is assumed to be invariant in direction.

The Earth frame (e-frame) has its origin at the center of the Earth and axes which are fixed with respect to the Earth, defined by the axis Ox_e, Oy_e, Oz_e with Oz_e along the Earth's polar axis. The Earth frame rotates, with respect to the inertial frame, at a rate ω about the axis Oz_i.

The navigation frame (n-frame) is a local geographic frame which has its origin at the location of the navigation system, shown in Figure 2.1, point P, and axes aligned with the directions of north, east and the local vertical (down).

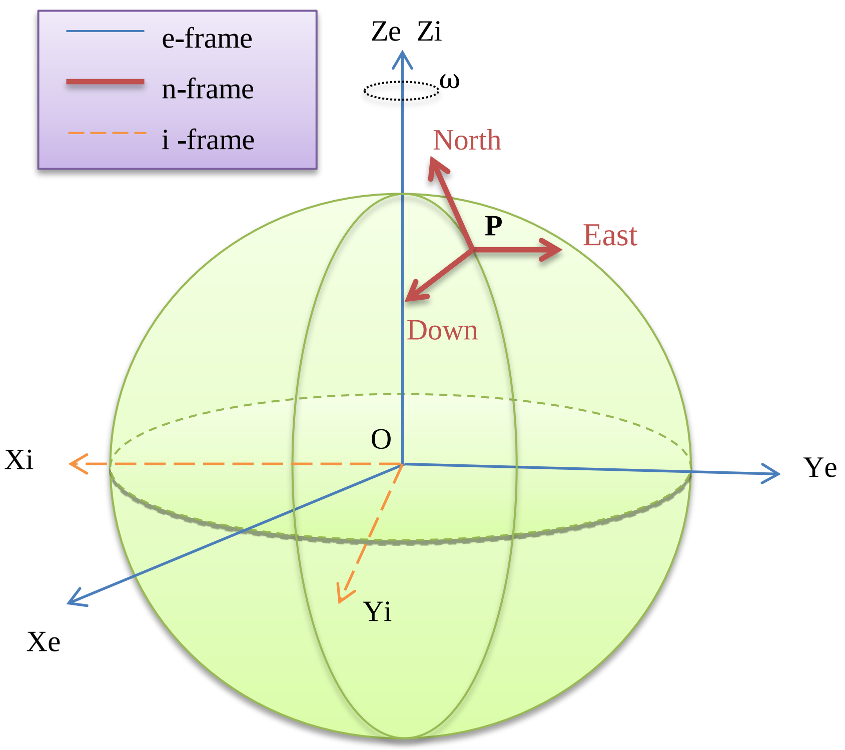


Figure 2.1 Frames of reference

The body frame (b-frame), depicted in Figure 2.2, is an orthogonal axis set which is aligned with the roll, pitch and yaw axes of the vehicle in which the navigation system is installed.

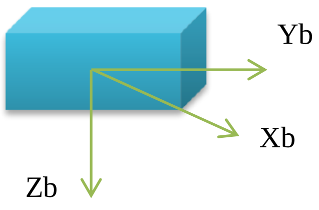


Figure 2.2 Illustration of a body reference frame

The notation C_{to}^{from} is used to denote a coordinate transformation matrix from one coordinate frame (designated by ‘from’) to another coordinated frame (designated by ’to’). For example,

C_{NED}^{ECI} denotes the coordinate transformation matrix from earth-centered inertial (ECI) coordinates to earth-fixed north-east-down (NED) local coordinates.

C_{NED}^{RPY} denotes the coordinate transformation matrix from vehicle body-fixed roll-pitch-yaw (RPY) coordinates to earth-fixed north-east-down (NED) local coordinates.

Coordinate transformation matrices satisfy the composition rule:

$$C_C^B C_B^A = C_C^A \tag{2.1}$$

where A, B, C represent different coordinate frames.
A coordinate transformation matrix is that if a vector **v** has the representation:

$$v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{2.2}$$

in XYZ coordinates and the same vector **v** has the alternative representation:

$$v = \begin{bmatrix} v_u \\ v_v \\ v_w \end{bmatrix} \tag{2.3}$$

in UVW coordinates, then

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = C_{XYZ}^{UVW} \begin{bmatrix} v_u \\ v_v \\ v_w \end{bmatrix} \tag{2.4}$$

where ‘XYZ’ and ‘UVW’ stand for any two Cartesian coordinate systems in three dimensional space.

2.2.2 Sensor Introduction and Noise Model

The accuracy of the angular measurements is fundamental to an INS, because any errors in transformation of acceleration will ultimately lead to errors in position. Thus, the ability of an INS to enable the continuous determination of vehicle position, velocity and attitude, primarily depends on the quality of gyro sensors used[3].
Gyroscopes used in inertial navigation are called ‘inertial grade’, which generally refers to a range of sensor performance, depending on INS performance requirements. Table 2.1 lists some generally accepted performance grades used for gyroscopes[4].

Table 2.1 Performance Grades for Gyroscopes

Performance Parameter	Performance Units	Performance Grades		
		Inertial	Intermediate	Moderate
Maximum input	deg/h	$10^2 - 10^6$	$10^2 - 10^6$	$10^2 - 10^6$
Scale factor	part/part	$10^{-6} - 10^{-4}$	$10^{-4} - 10^{-3}$	$10^{-3} - 10^{-2}$
Bias stability	deg/h	$10^{-4} - 10^{-2}$	$10^{-2} - 10$	$10 - 10^2$
Bias drift	deg/ $\sqrt{h}$	$10^{-4} - 10^{-3}$	$10^{-5} - 10^{-4}$	$10^{-4} - 10^{-3}$

The projector in Vodafone decided to use IMU-3000 Triple Axis Gyroscope due to its high cost performance. The Random-Walk error model is used to estimate the performance of IMU-3000 Triple Axis Gyroscope. Random-walk errors are characterized by variances that grow linearly with time and power spectral densities that fall off as $1/f^2$. There are specifications for random walk noise in inertial sensors, but mostly for the integrals of their outputs, and not in the outputs themselves. For example, the 'angle random walk' from a rate gyroscope is equivalent to white noise in the angular rate outputs[4].

The 'angle random walk' error model is shown as follows[4]:

$$\varepsilon_k = \varepsilon_{k-1} + w_{k-1} \tag{2.5}$$

and the error variance is :

$$\begin{aligned} \sigma_k^2 &= E\left\{\varepsilon_k^2\right\} \\ &= \sigma_{k-1}^2 + E\left\{w_{k-1}^2\right\} \\ &= \sigma_0^2 + k \cdot E\left\{w_k^2\right\} \end{aligned} \tag{2.6}$$

The value of $E\left\{w_k^2\right\}$ will be in units of squared-error per discrete time step Δt. Random-walk error sources are usually specified in terms of standard deviations, that is error units per square-root of time unit. Gyroscope angle random walk errors, might be specified in deg/ $\sqrt{h}$. Most navigation-grade gyroscopes have angle random-walk errors in the order of 10^{-3} deg/ $\sqrt{h}$ or less[4].

From IMU-3000 specification, the angle speed with noise $\omega' = \omega + N$, where N is the spectral density which equals to $0.01 dps / \sqrt{Hz}$, dps means degree per second, if the frequency of low pass filter (LPF) in the gyro is 100Hz. Thus one deviation:

$$\sigma_N = 0.01 \times \sqrt{100} = 0.1 \mathrm{deg}/ s \tag{2.7}$$

and $$\sigma_N^2 = \left(0.1\times\frac{\pi}{180}\right)^2 = 3.0\times10^{-6} rad^2/s^2 \tag{2.8}$$

After that , the angle in a short time is represented :

$$d\theta = \omega' \cdot dt = \omega \cdot dt + N \cdot dt \tag{2.9}$$

Assume that the sampling rate is 4000Hz, and N' indicates $N \cdot dt$, the error variance in the short time can be calculated as :

$$\sigma_{N'}^2 = 3600 s/h \times 4000 Hz \times \sigma_N^2 \times (1/4000)^2 = 2.7\times10^{-6} rad^2/h \tag{2.10}$$

Then the one deviation is: $\sigma_{N'} = 1.64\times10^{-3} rad/\sqrt{h} = 0.09 \deg/\sqrt{h}$. Compare with the result talked before, these gyro grade is around 2 order (10^{-2}) worse than the typical navigation gyro, if LPF is equal to 100Hz.

2.3 Design Requirements and Corresponding Rotation Error Variance

2.3.1 Design Requirements

When a car is travelling in a relatively complex environment, it is important to make the antenna always aiming to the satellite in order to keep communication. However, the car maybe bump along the rough road, the maximum angular speed and angular acceleration speed are $1000^o/s$ and $1500^o/s^2$, respectively. Generally, the performance of the beam tracking system is measured by three aspects. One is the angular speed, the second one is angular acceleration, and the third one is the error correction. And the main task of the thesis is insure the antenna of the system always aims to the satellite no matter what situation is. Here are some antenna parameters and assumptions:

(1) The system has a 64x64 array antenna, and the azimuth is 45 degree;

(2) Both antenna pattern and phase shifter are perfect, ignore the external error;

(3) In order to achieve better system performance, the maximum antenna gain losses will not exceed 0.5 dB. From Figure 2.3 and Figure 2.4, the corresponding maximum elevation angular is 0.41 degree. This was get from antenna designers in the project team.

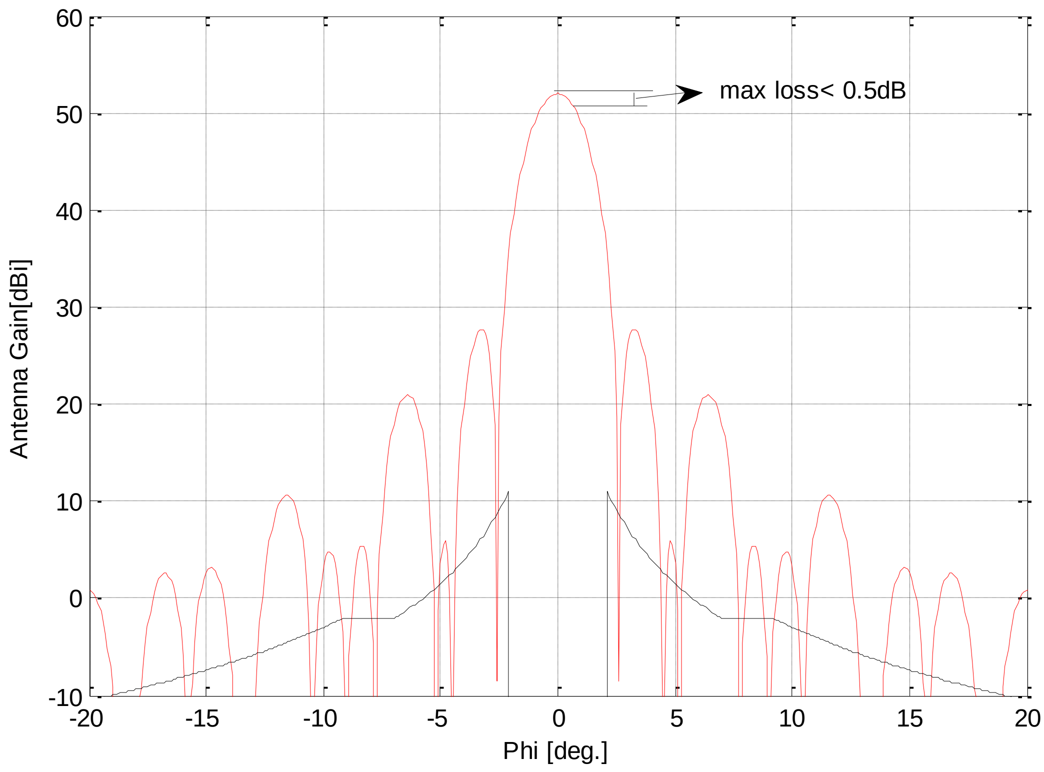


Figure 2.3 Antenna Gain Schematic

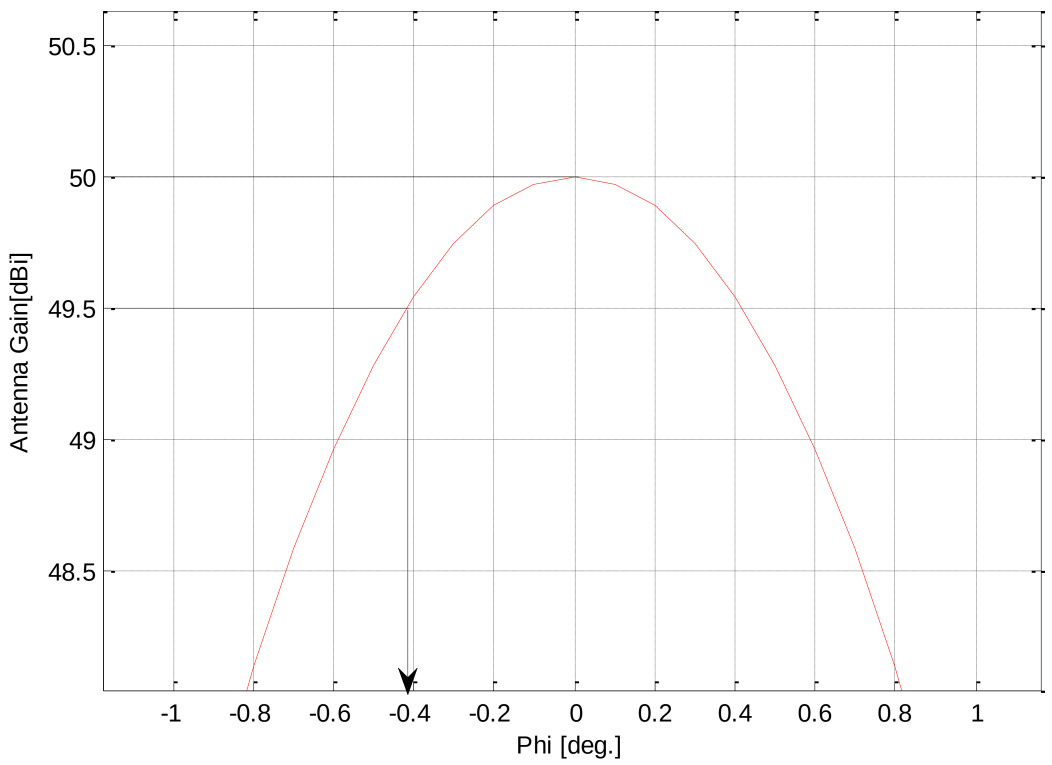


Figure 2.4 Enlarged View of Antenna Gain

From the allowable gain loss and the maximum angular speed, the minimum antenna signal update rate could be calculated. If the maximum angular speed is $1000^o/s$, the minimum update rate is $1000^\circ/s \div 0.41^\circ = 2440Hz$.

2.3.2 Rotation Error Variance

Now the up limit of the system error has to be considered. The up limit error means once the total error exceed the up limit, the antenna will lose the direction to the satellite, and the system can't be used as communication purpose. The idea of the report is make the total error under the up limit by Direction Cosine Matrix (DCM) algorithm or Euler Angle algorithm. First step is to calculate the maximum error for a high requirement system which the angular speed is $1000^o/s$, and the angular acceleration is $1500^o/s^2$. In DCM algorithm, the rotation matrix is composed by original rotation matrix and error matrix:

$$R^{'} = R + R^e \tag{2.11}$$

where $R^e = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ N_{21} & N_{22} & N_{23} \\ N_{31} & N_{32} & N_{33} \end{bmatrix}$, N_{xy} is rotation error which is considered to distribute normally, and all the parameters are uncorrelated.

The direction vector of the antenna is represented as shown below:

$$\begin{bmatrix} V_1^0 \\ V_2^0 \\ V_3^0 \end{bmatrix} = R \cdot \begin{bmatrix} V_1^I \\ V_2^I \\ V_3^I \end{bmatrix} + R^e \cdot \begin{bmatrix} V_1^I \\ V_2^I \\ V_3^I \end{bmatrix} = R \cdot \begin{bmatrix} V_1^I \\ V_2^I \\ V_3^I \end{bmatrix} + \begin{bmatrix} V_1^e \\ V_2^e \\ V_3^e \end{bmatrix} \tag{2.12}$$

where $\left|V^0\right|^2 = \left(V_1^0\right)^2 + \left(V_2^0\right)^2 + \left(V_3^0\right)^2 = 1$, and $\left|V^I\right|^2 = \left(V_1^I\right)^2 + \left(V_2^I\right)^2 + \left(V_3^I\right)^2 = 1$,the error variance of the error rotation matrix is marked as $\sigma_{N_{xx}}^2$. Then the absolutely value of error vector is $\left|V^e\right|^2 = 3\sigma_{N_{xx}}^2$.

Next is the error angular estimates, utilizing the theory of trigonometric function, the rotation verctor and the error vector is shown as follows (Figure 2.5):

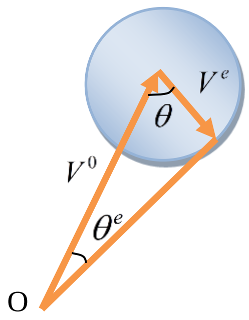


Figure 2.5 Direction Vector and Error Vector Schematic

The error angular is represented as shown below:

$$\theta^e = \frac{|V^e| \sin\theta}{|V^0|} \tag{2.16}$$

where θ is uniformly at $[0, 2\pi]$. Then the variance of the error angular is calculated as follows:

$$\sigma_{\theta^e}^2 = \int_0^{2\pi} \theta^{e2} \cdot \frac{1}{2\pi} d\theta = \int_0^{2\pi} \frac{|V^e|^2 \sin^2\theta}{|V^0|^2} \cdot \frac{1}{2\pi} d\theta = \frac{1}{2} \cdot |V^e|^2 = \frac{3}{2} \cdot \sigma_{N_{xx}}^2 \tag{2.17}$$

Assume that the error is Gaussian distributed (Figure 2.6) shown as follows:

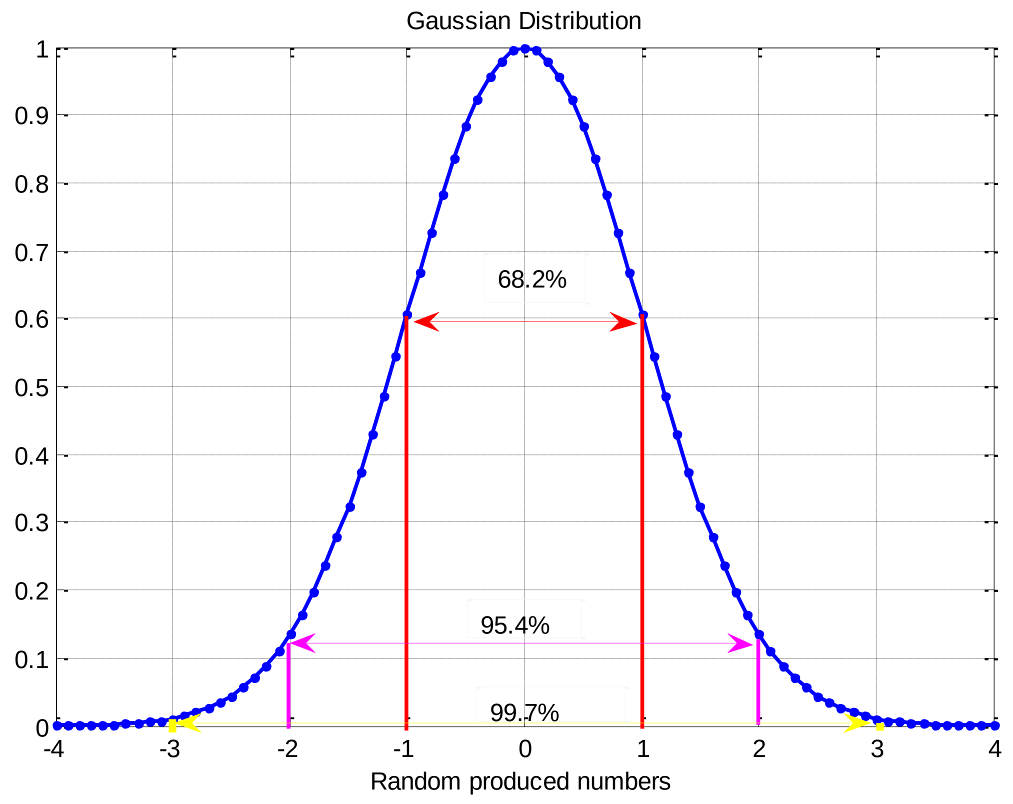


Figure 2.6 Gaussian Distribution

Now the up limit of error variance according the angular speed and update rate could be calculated. If the angular speed is $1000^o / s$, and the antenna update rate is 4000Hz (which is larger than 2440Hz), the maximum antenna angle shift is $1000^o \div 4000 = 0.25^o$. From Fig.2.6, about 68% of the values drawn from a normal distribution are within one standard deviation σ away from the mean, about 95% of the values lie within two standard deviations, and about 99.7% are within three standard. This fact is known as the 3-sigma rule.

In this thesis, the three standard deviations are used to instead of the total error, then the allowable model error is : $3\sigma = 0.41^o - 0.25^o = 0.16^o$, then $\sigma = 0.053^o = 9.31 \times 10^{-4} rad$, the same with update rate is 8000Hz , $\sigma = 1.66 \times 10^{-3} rad$. The result is shown as follow s(Table 2.2):

Table 2.2 Error Variance for Antenna BF Model

Update rate(Hz)	Maximum angle shift (degree)	Allowable model error (σ_{θ^e})
4000	0.25	$9.31 \times 10^{-4} rad$
8000	0.125	$1.66 \times 10^{-3} rad$

From Eq. (2.17) , the noise model error variance $3\sigma^2_{N_{xx}} = 2\sigma^2_{\theta^e}$, then the up limit of noise error variance is represented as (Table 2.3):

Table 2.3 Error Variance for Noise Model

Update rate (Hz)	$\sigma^2_{\theta^e}$	$3\sigma^2_{N_{xx}}$
4000	$8.67 \times 10^{-7} rad$	$1.73 \times 10^{-6} rad$
8000	$2.76 \times 10^{-6} rad$	$5.51 \times 10^{-6} rad$

Here the $3\sigma^2_{N_{xx}}$ is the design criteria for the beam tracking system, and it is refer to the Eq.(25) which is $b \cdot n = 3\sigma^2_{N_{xx}}$.

In the next chapter, the DCM algorithm and Euler Angle algorithm will be introduced to see the performance of the system.

CHAPTER 3 Direction Cosine Matrix and Euler Angle Algorithms

In this chapter, the Direction Cosine Matrix Algorithm and Euler Angle Algorithm are introduced.

3.1 DCM Algorithm

It is the computation of attitude which is particularly critical in an inertial navigation system. In many applications, the dynamic range of the angular motions to be taken account of can be very large, such as the case in this thesis has reached to $1000^o/s$. The ability of the algorithm to keep track of body attitude accurately in a severe vibratory environment may well be the critical factor in determining its performance, if accurate navigation is to be achieved. The conventional approach to attitude determination is to computer the direction cosine matrix (DCM), relating the vehicle body reference frame to the reference coordinate system. The implementation of a direction cosine matrix based inertial measurement unit for application in model planes and helicopters was achieved by DIY ZONE designers. Now this algorithm will be implemented in the thesis case to see the how good the performance can be.

3.1.1 Direction Cosine Matrices and Algorithm Structure

The direction cosine matrix, denoted by the symbol R_b^n, is a 3×3 matrix which is written here in component form as follows:

$$R_b^n = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} \tag{3.1}$$

The element in the i th row and the j th column represents the cosine of the angle between the i-axis of the reference frame and the j-axis of the body frame.

A vector quantity defined in body axes, Q^b, may be expressed in reference axes by pre-multiplying the vector by the direction cosine matrix as follows:

$$Q^n = R_b^n \cdot Q^b \tag{3.2}$$

The algorithm process is shown schematically in Figure 3.1:

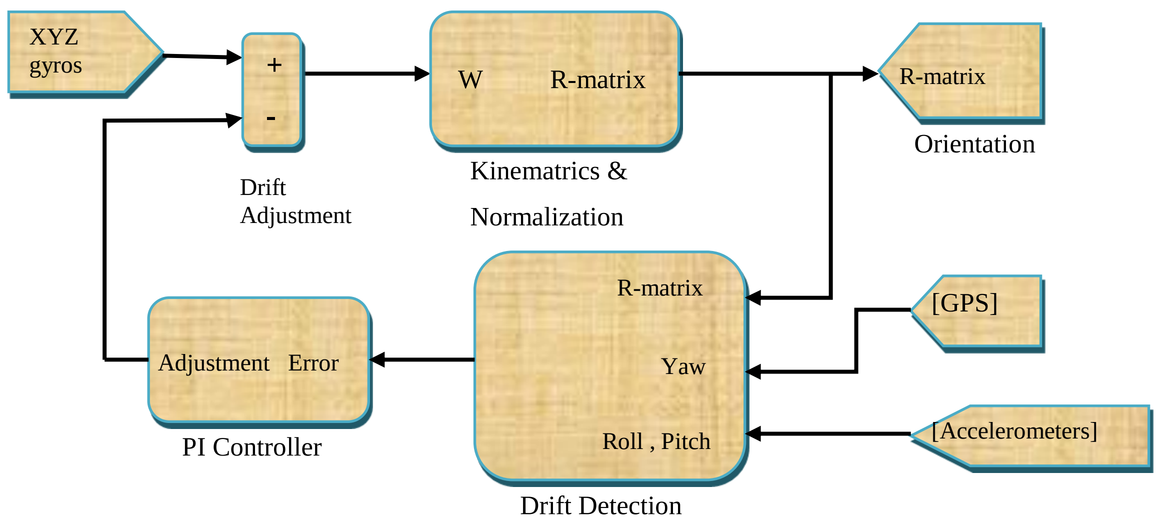


Figure 3.1 Block Diagram of DCM[5]

From the Figure 3., the gyro sensors inside of the INS are used as the primary source of orientation information, and the numerical errors in the integration will gradually destory the orthogonality constraints that the DCM must satisfy, thus regular small adjustments to the elements of the matrix are made to satisfy the constraints[5].

Further more, the numerical errors, gyro drift will gradually accumulate deviation in the rotation calculations, and the reference vectors which are provided by GPS and Accelerometer are used to detect the errors. As a relusts, a proportional plus integral (PI) negative feedback controller between the gyro inputs and the detected errors used to the drift adjustment, to dissipate the errors faster than they can build up. Yaw error is detected by GPS, and accelerometers are used to detect pitch and roll errors.

The central concept of the DCM algorithm is that the nonlinear differential equation relates the time rate of change of the direction cosines to the gyro signals. The goal is to compute the direction cosines without making any approximations that violate the nonlinearity of the equations[5].

3.1.2 Kinematics of Rotations

Electronic rate gyros will rotate with the vehicle, and producing signals proportional to the rotation rate. A well known result of kinematics is that the rate of change of a rotating vector due to its rotation is given by:

$$\frac{dr(t)}{dt} = \omega(t) \times r(t) \tag{3.3}$$

where $\omega(t)$ is the rotation rate vector.

If the initial conditions and the time history of the rotation vector are known, Equation (3.4) is used to track the rotation vector:

$$r(t) = r(0) + \int_0^t d\theta(\tau) \times r(\tau) \tag{3.4}$$

where $d\theta(\tau) = \omega(\tau)d\tau$.

However, the rotation vectors are not measured in the same reference frame. Generally, the axes of the vehicle in the earth frame would like to be tracked, but the signals that gyro sensors measured are in the body frame. Then the earth axes as seen in the body frame can be tracked by flipping the sign of the gyro signals, this is shown as follows:

$$r_{earth}(t) = r_{earth}(0) + \int_0^t r_{earth}(\tau) \times d\theta(\tau) \tag{3.5}$$

According to the methods in Mahoney's papers [6], equation (3.5) is approximated to differential form like equation (3.6):

$$r_{earth}(t+dt) = r_{earth}(t) + r_{earth}(t) \times d\theta(t) \tag{3.6}$$

Repeat equation (3.6) for each of the earth axes, the result is represented as a convenient matrix form:

$$R(t+dt) = R(t)\begin{bmatrix} 1 & -d\theta_z & -d\theta_y \\ d\theta_z & 1 & -d\theta_x \\ -d\theta_y & d\theta_x & 1 \end{bmatrix} \tag{3.7}$$

where $d\theta_x = \omega_x dt$, $d\theta_y = \omega_y dt$, $d\theta_z = \omega_z dt$.

3.1.3 Drift Cancellation: GPS and Accelerometer

Although the gyros perform rather well, with an uncorrected offset on the order of a few degrees per second, eventually the drift must be cancelled. An effective way to cancel these drift is to use other orientation references to detect the gyro offsets and provide a negative feedback loop back to the gyros to compensate for the errors in a classical detection and feedback loop, as shown in Figure 3.1. GPS is used to correct the yaw axis error, and accelerometer is used to correct the roll and pitch axis errors.

The GPS horizontal course over ground signal has zero drift over long time running, thus can be used as a reference vector to achieve 'yaw lock' for the vehicle. The correction is shown as follows:

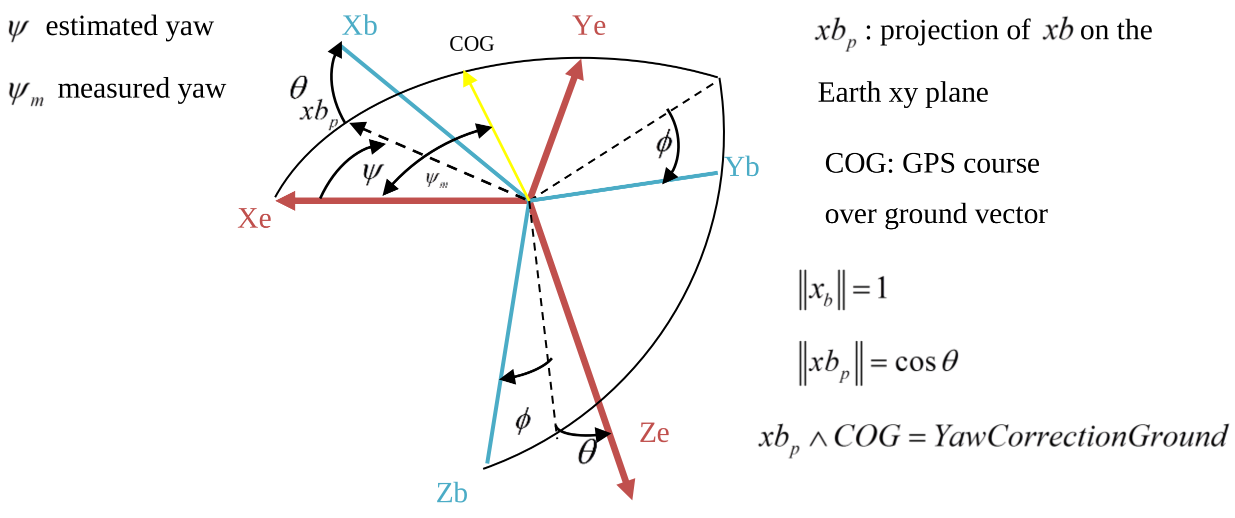


Figure 3.2 Yaw Correction by GPS

The rotational error between the GPS course over ground vector (COG), and the projection on the horizontal plane of the roll axis (X) of the IMU is an indication of the amount of drift. The rotational correction is the Z component of the cross product of the X column of the R matrix and the course over ground vector.

$$COGX = \cos(cog) = r_{xx} \Big/ \sqrt{r_{xx}^2 + r_{xy}^2} \tag{3.8}$$

$$COGY = \sin(cog) = r_{xy} \Big/ \sqrt{r_{xx}^2 + r_{xy}^2} \tag{3.9}$$

Then computer the yaw correction in the earth frame of reference:

$$YCG = r_{xx}COGY - r_{xy}COGX \tag{3.10}$$

In order to adjust the gyro drift, the correction vector in the body frame of reference should be know. To compute that the yaw correction in the ground frame of reference is multiplied by the Z row of the R matrix:

$$YCP = YCG \cdot \begin{bmatrix} r_{xz} \\ r_{yz} \\ r_{zz} \end{bmatrix} \tag{3.11}$$

The yaw correction vector produced by Equation (3.18) will be combined with roll-pitch correction computed from the accelerometers into a total vector that is used to compensate for drift.

In the body frame of reference, the centrifugal acceleration[10] is calculated as the cross product of the gyro vector and the velocity vector:

$$A_{centrifugal} = -2\omega_{gyro} \times V \tag{3.12}$$

$$V = \begin{bmatrix} velocity \\ 0 \\ 0 \end{bmatrix} \tag{3.13}$$

The output of the accelerometers is gravity minus the acceleration. Therefore, the reference measurement of gravity in the body frame is given by:

$$g_{reference} = Accelerometer - 2\omega_{gyro} \times V \tag{3.14}$$

where $Accelerometer = \begin{bmatrix} Accelerometer_x \\ Accelerometer_y \\ Accelerometer_z \end{bmatrix}$

The roll-pitch rotational correction vector in the body frame of reference is computed by taking the cross product of the normalized gravity reference vector with the Z row of the direction cosine matrix:

$$RollPitchCorrectionPlane = \begin{bmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{bmatrix} \times g_{reference} \tag{3.15}$$

3.1.4 Feedback Controller

Each of the rotational drift correction vectors (yaw and roll-pitch) are multiplied by suitable weights and fed to a proportional plus integral (PI) feedback controller to be added to the gyro vector to produce a corrected gyro vector that is used as the input to Equation (3.7). The total of the rotation corrections is shown as:

$$TotalCorrection = W_{RP} \cdot RPCP + W_Y \cdot YCP \tag{3.16}$$

Next, the total correction is passed through a PI controller:

$$\begin{aligned} \omega_{PCorrection} &= K_P TotalCorrection \\ \omega_{ICorrection} &= \omega_{ICorrection} + K_I dt \cdot TotalCorrection \\ \omega_{correction} &= \omega_{PCorrection} + \omega_{ICorrection} \end{aligned} \tag{3.17}$$

Where K_P and K_I are the suitable weights chosed by experience and practice. Then feed the gyro correction vector back into the rotation update equation by adding the correction vector to the gyro signal, as shown :

$$\omega(t) = \omega_{gyro}(t) + \omega_{correction}(t) \tag{3.18}$$

where $\omega_{gyro}(t)$ are the gyro measurements for three axis, and $\omega_{correction}(t)$ is the gyro correction.
At this point, a whole correction has been completed. Repeat the entire calculation, the errors will be cancelled over a long period of time.

3.2 Euler Angle Algorithm

Another way to cancel the drift is Euler Angle Algorithm. The primary idea is calculate the difference between the reference direction vector and the measurement direction vector from GPS and Accelerometer, take the cross product and rotate the incorrect direction vector back to original direction.

3.2.1 Euler Angle Basics

Euler angles are used to define a coordinate transformation in terms of a set of three angular rotations performed in a specified sequence about three specified orthogonal axes, to bring one coordinate frame to coincide with another. The three rotations may be expressed mathematically as three separate diection cosine matrices as defined below:

Rotation θ_R about x-axis
$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_R & -S_R \\ 0 & S_R & C_R \end{bmatrix} \tag{3.19}$$

Rotation θ_P about y-axis
$$P = \begin{bmatrix} C_p & 0 & S_P \\ 0 & 1 & 0 \\ -S_P & 0 & C_P \end{bmatrix} \tag{3.20}$$

Rotation θ_Y about z-axis
$$Y = \begin{bmatrix} C_Y & -S_Y & 0 \\ S_Y & C_Y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

where $C = \cos\theta, S = \sin\theta$, θ is referred to as the Euler rotation angle.

Then the coordinate transformation from north , east, and down (NED) coordinates to roll , pitch, and yaw (RPY) coordiantes can be composed from three Euler rotation matrices:

$$C_{RPY}^{NED} = R^T \cdot P^T \cdot Y^T = \begin{bmatrix} C_Y C_P & S_Y C_P & -S_P \\ -S_Y C_R + C_Y S_P S_R & C_Y C_R + S_Y S_P S_R & C_P S_R \\ S_Y S_R + C_Y S_P C_R & -C_Y S_R + S_Y S_P C_R & C_P C_R \end{bmatrix} \tag{3.22}$$

3.2.2 Roll , Pitch , Yaw Drift Cancellation

From Equation (3.8) and (3.9), the GPS crouse over ground vector (COG) can be defined. GPS provides yaw direction information. Then $C_{Ym} = \cos\theta_Y$, $S_{Ym} = \sin\theta_Y$. The deviation angle θ_Y could be measured by GPS. Accelerometer can provide deviation angle information in roll and pitch axis.

For yaw correction, the x-axis projection of vehicle in NE plane can be expressed as $\vec{X}_c = \left[C(1,1) \quad C(1,2) \quad 0 \right]$ which is getting from rotation matrix. Furthermore, the same x-axis projection measure by GPS can be expressed as $\vec{X}_m = \left[C_{Ym} \quad S_{Ym} \quad 0 \right]$. From this , the deviation rotation angle between these two vectors can be calculated, shown as follows:

$$\sin\theta_{Yd} = \frac{\vec{X}_C \times \vec{X}_m}{\left|\vec{X}_C\right| \cdot \left|\vec{X}_m\right|} \tag{3.23}$$

and
$$\cos\theta_{Yd} = \sqrt{1 - \left(\sin\theta_{Yd}\right)^2} \tag{3.24}$$

If the deviation rotation angle θ_{Yd} is closed to zero, according to Taylor's Formula,

$$\sin\theta_{Yd} = \theta_{Yd} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \tag{3.25}$$

Take the first order approximate value of $\sin\theta_{Yd}$ .Then written as:

$$\theta_{Yd} = F \cdot \frac{\vec{X}_C \times \vec{X}_m}{\left|\vec{X}_C\right| \cdot \left|\vec{X}_m\right|} \tag{3.26}$$

Where F means trust factor in order to adjust how much deviation should be trusted in the simulation. After this, the two rotation matrix Y_m and Y_d can be written as :

$$Y_m = \begin{bmatrix} C_{Ym} & -S_{Ym} & 0 \\ S_{Ym} & C_{Ym} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.27}$$

$$Y_d = \begin{bmatrix} C_{Yd} & -S_{Yd} & 0 \\ S_{Yd} & C_{Yd} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.28}$$

Similarly, the pitch and roll axis drift can be calculated. Expressed as P_d, P_m and R_d.

According to Equation (3.22), $C_{RPY}^{NED} = R^T \cdot P^T \cdot Y^T$, now the drift correction matrix can be shown as:

$$C_2 = C_{RPY}^{NED} \cdot Y_m \tag{3.29}$$

$$C_3 = C_2 \cdot P_m \tag{3.30}$$

$$C_1^{'} = C_3 \cdot R_d{}^T \cdot \left(P_d \cdot P_m\right)^T \cdot \left(Y_d \cdot Y_m\right)^T \tag{3.31}$$

$C_1^{'}$ should be equal to C_{RPY}^{NED}, which means the drift is cancelled perfectly. However it is not zero due to the external noise and some sensor bias, and the performance of the system is determined by the error between these two matrix. The aim of the algorithm is to find the error and try to control this error under the up limit error which was calculated in Table 2.3.

CHAPTER 4 Performance Analysis for Two Algorithms

In this chapter, we present the performance of two algorithms. The simulation results show that the DCM algorithm is not suitable for our case and the Euler Angle algorithm can work well.

4.1 Rotation Matrix Error Analysis

4.1.1 Rotation Matrix Error Model

Rotation angles are measured by gyro sensors, the sensor's temperature variation will cause offset. It expresses that the gyro signals give more or less rotation angular in a short period. See the model as follows:

$$R(n+1) = R(n) \cdot \begin{bmatrix} 1 & -d\theta_z & d\theta_y \\ d\theta_z & 1 & -d\theta_x \\ -d\theta_y & d\theta_x & 1 \end{bmatrix} \tag{4.1}$$

where $d\theta_x = \omega_x dt,\ d\theta_y = \omega_y dt,\ d\theta_z = \omega_z dt$.

R is the direction vector, means the direction of the rotation, $\omega_x , \omega_y , \omega_z$ is the angular speed. From Equation (4.1), the direction at $n+1$ time is the direction vector multiply by the rotation matrix in a short period. However, as talking above, there are some unavoidable errors in the gyro sensors. Then $d\theta_x = \omega_x dt,\ d\theta_y = \omega_y dt,\ d\theta_z = \omega_z dt$ will add an error term:

$$\begin{aligned} d\theta_x^{'} &= \omega_x dt + N_X dt \\ d\theta_y^{'} &= \omega_y dt + N_y dt \\ d\theta_z^{'} &= \omega_z dt + N_z dt \end{aligned} \tag{4.2}$$

Assume that the error term obeys a Gaussian distribution, which is $N_x, N_y, N_z \sim N\left(0, \sigma_N\right)$, $E[\omega_x] = E[\omega_y] = E[\omega_z] = 0$, all N_x, N_y, N_z , ω_x , ω_y , ω_z are uncorrelated. Next, the mean and variance of Equation (4.2) can be calculated as follows:

$$E[d\theta^{'}] = 0 \ \ , Var[\left|d\theta^{'}\right|] = E[\left|d\theta^{'}\right|^2] - E^2[d\theta^{'}] = E[\left|d\theta^{'}\right|^2] = \sigma_N^2 \left(dt\right)^2 \tag{4.3}$$

The rotation matrix can be rewritten as :

$$R_u^{'}(n) = R_u(n) + R_u^e(n) \tag{4.4}$$

where

$$R_u(n) = \begin{bmatrix} 1 & -\omega_z dt & \omega_y dt \\ \omega_z dt & 1 & -\omega_x dt \\ -\omega_y dt & \omega_x dt & 1 \end{bmatrix} \tag{4.5}$$

$$R_u^e(n) = \begin{bmatrix} 0 & -N_z dt & N_y dt \\ N_z dt & 0 & -N_x dt \\ -N_y dt & N_x dt & 0 \end{bmatrix} \tag{4.6}$$

Equation (4.4) is the rotation error model, and Equation (4.1) is a recipe for updating the direction cosine matrix from gyro signals. But Equation (4.1) is correct only if period is tend to zero, however, exactly zero is impossible, this is sampling error.

Next, the model error is calculated. According to Equation (4.4), start from n equals to one.

$$\begin{aligned} R^{'}(1) &= R(0)\cdot R_u^{'}(1) \\ &= R(0)\cdot\left(R_u(1)+R_u^e(1)\right) \\ &= R(0)R_u(1)+R(0)R_u^e(1) \end{aligned} \tag{4.7}$$

Now when n equals to 2:

$$\begin{aligned} R^{'}(2) &= R^{'}(1)\cdot R_u^{'}(2) \\ &= \left[R(1)+R^e(1)\right]\cdot\left[R_u(2)+R_u^e(2)\right] \\ &= R(1)R_u(2)+R(1)R_u^e(2)+R^e(1)R_u(2)+R^e(1)R_u^e(2) \end{aligned} \tag{4.8}$$

Here let $R(2)=R(1)R_u(2)$, and $R^e(2)=R(1)R_u^e(2)+R^e(1)R_u(2)+R^e(1)R_u^e(2)$. Equation (4.8) comes to :

$$R^{'}(2) = R(2)+R^e(2) \tag{4.9}$$

The same when n equals to 3.

$$\begin{aligned} R^{'}(3) &= R^{'}(2)\cdot R_u^{'}(3) \\ &= \left[R(2)+R^e(2)\right]\cdot\left[R_u(3)+R_u^e(3)\right] \\ &= R(2)R_u(3)+R(2)R_u^e(3)+R^e(2)R_u(3)+R^e(2)R_u^e(3) \end{aligned} \tag{4.10}$$

and

$$R^{'}(3) = R(3)+R^e(3) \tag{4.11}$$

Here give some definitions for this error model. Direction vector error covariance matrix is the target to measure the performance of the system, which is defined by :

$$C^e(n) = E\left\{\left(R^e(n)\right)^T\left(R^e(n)\right)\right\} \tag{4.12}$$

$$C_u^e(n) = E\left\{\left(R_u^e(n)\right)^T\left(R_u^e(n)\right)\right\} \tag{4.13}$$

$$C_u(n) = E\left\{\left(R_u(n)\right)^T\left(R_u(n)\right)\right\} \tag{4.14}$$

Assume that the vectors are time independent, then: $C_u^e(n)=C_u^e(1)=C_u^e$, and $C_u(n)=C_u(1)=C_u$. According to the orthogonality principle, $R^T(n)R(n)=R(n)R^T(n)=I$, where I is an unit matrix. So the covariance matrix :

$$C^e(1) = C_u^e \tag{4.15}$$

$$\begin{aligned} C^e(2) &= C_u^e + C^e(1)\left(C_u^e + C_u\right) \\ &= C_u^e + C_u^e\left(C_u^e + C_u\right) \end{aligned} \tag{4.16}$$

$$\begin{aligned} C^e(3) &= C_u^e + C^e(2)\left(C_u^e + C_u\right) \\ &= C_u^e + C_u^e\left(C_u^e + C_u\right) + C_u^e\left(C_u^e + C_u\right)^2 \end{aligned} \tag{4.17}$$

$$\vdots$$

$$C^e(n) = C_u^e\left[1+\left(C_u^e + C_u\right)+\cdots+\left(C_u^e + C_u\right)^{n-1}\right] \tag{4.18}$$

where

$$\begin{aligned}
C_u &= E\left\{\left(R_u(n)\right)^T\left(R_u(n)\right)\right\} \\
&= E\left\{\begin{bmatrix} 1 & d\theta_z & -d\theta_y \\ -d\theta_z & 1 & d\theta_x \\ d\theta_y & -d\theta_x & 1 \end{bmatrix}\begin{bmatrix} 1 & -d\theta_z & d\theta_y \\ -d\theta_z & 1 & -d\theta_x \\ -d\theta_y & d\theta_x & 1 \end{bmatrix}\right\} \\
&= E\left\{\begin{bmatrix} 1+(d\theta_z)^2+(d\theta_y)^2 & -d\theta_y\cdot d\theta_x & -d\theta_z\cdot d\theta_x \\ -d\theta_y\cdot d\theta_x & 1+(d\theta_z)^2+(d\theta_x)^2 & -d\theta_z\cdot d\theta_y \\ -d\theta_z\cdot d\theta_x & -d\theta_y\cdot d\theta_z & 1+(d\theta_y)^2+(d\theta_x)^2 \end{bmatrix}\right\} \\
&= \left(1+2\cdot(dt)^2\sigma_{N1}^2\right)\cdot I
\end{aligned} \tag{4.19}$$

$$\begin{aligned}
C_u^e &= E\left\{\left(R_u^e(n)\right)^T\left(R_u^e(n)\right)\right\} \\
&= E\left\{\begin{bmatrix} 0 & N_z dt & -N_y dt \\ -N_z dt & 0 & N_x dt \\ N_y dt & -N_x dt & 0 \end{bmatrix}\begin{bmatrix} 0 & -N_z dt & N_y dt \\ N_z dt & 0 & -N_x dt \\ -N_y dt & N_x dt & 0 \end{bmatrix}\right\} \\
&= E\left\{\begin{bmatrix} (N_z dt)^2+(N_y dt)^2 & -N_y dt\cdot N_x dt & -N_z dt\cdot N_x dt \\ -N_y dt\cdot N_x dt & (N_z dt)^2+(N_x dt)^2 & -N_z dt\cdot N_y dt \\ -N_z dt\cdot N_x dt & -N_y dt\cdot N_z dt & (N_y dt)^2+(N_x dt)^2 \end{bmatrix}\right\} \\
&= \left(2\cdot(dt)^2\sigma_{N2}^2\right)\cdot I
\end{aligned} \tag{4.20}$$

where $dt = 1/f$, and f is the sample rate of gyro.

Let $a = 1+2\cdot(dt)^2\sigma_{N_1}^2$, $b = 2\cdot(dt)^2\sigma_{N_2}^2$, then :

$$C^e(n) = b\cdot\left[1+(a+b)+\cdots(a+b)^{n-1}\right]\cdot I \tag{4.21}$$

Since $a+b = 1+2\cdot(dt)^2\sigma_{N_1}^2+2\cdot(dt)^2\sigma_{N_2}^2 \approx 1$, then

$$C^e(n) = b\cdot n\cdot I \tag{4.22}$$

From Equation.(4.22) the noise error variance is linear as time goes on.

4.1.2 Simulation Results

The simulation is to verified the theory above. To see if the rotation noise error variance is linear as time go on. The sample rate of gyro is 50 Hz, and the gyro Gaussian noise variance is set to $3e-6$. Simulation time is set to 10 seconds. The results is shown as Figure 4.1 :

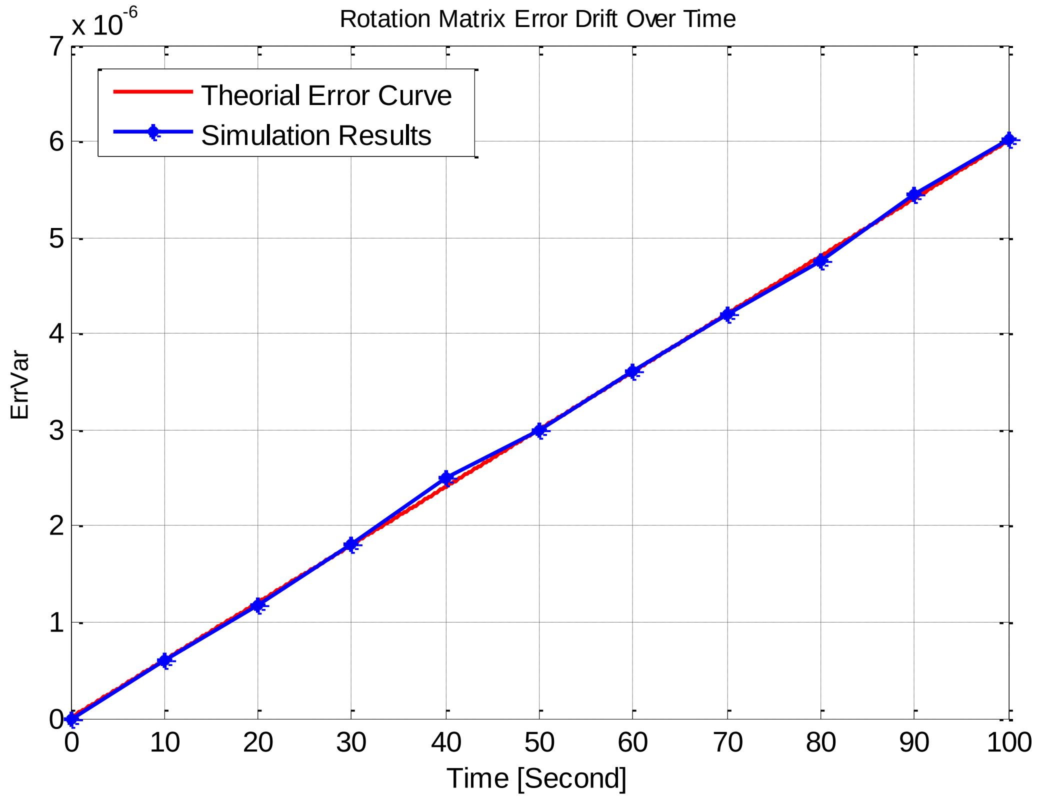


Figure 4.1 Rotation Matrix Error Model Simulation

From the Figure 4.1, the simulation results are in agreement with the theoretical value which was deduced above (Equation 4.22). This also means that the error variance will increase as time go on if there is not any correction. Thus, the information provided by GPS and Accelerometer will correct the error and make the error converge under the up limit value. Next, the performance of DCM Algorithm will be discussed.

4.2 DCM Performance Analysis

As discussed above, the error variance will increase as time if there are not GPS and Accelerometer correction. The designers have applied the DCM algorithm to Unmanned Aerial Vehicle (UAV) system successfully. However, one of the biggest differences between the UAV and the case this thesis focus on is the operating environment. As mentioned in Chapter One, there are much more serious bumping and great rotations in land vehicle navigation system, compared to the UAV system, the UAV operating environment is much better, the transient plane cornering speed and angle acceleration is small.

In this thesis, the system requirement is that the maximum angle speed and angle acceleration of the vehicle are $1000^o / s$ and $1500^o / s^2$, respectively. Considering the limit of the hardware and cost, the sample rate of gyro sensors is chose to 50Hz. The InvenSense IMU3000 chip is chose to be the hardware platform, this chip will be discussed later in Chapter Five. According to the up limit error variance under different receive antenna update frequency, the performance of the DCM algorithm can be estimated as follows (Figure 4.2):

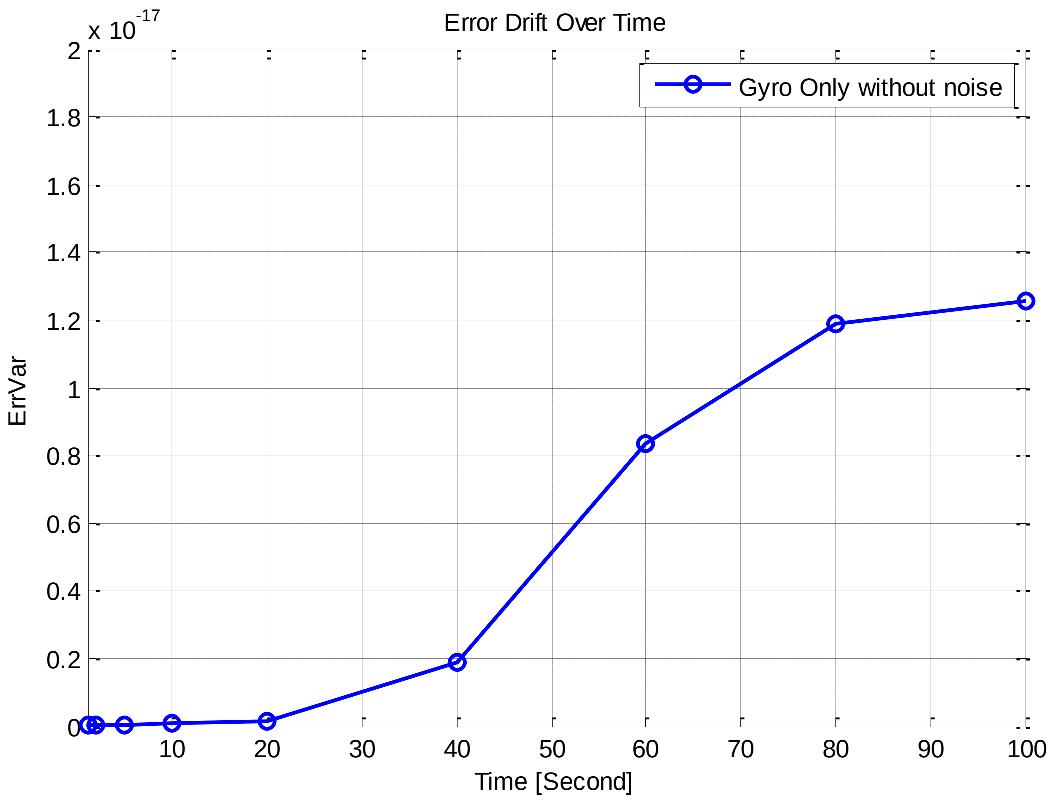


Figure 4.2 Error Variance without Noise

Figure 4.2 shows the system error variance without external noise, which means all sensor bias, GPS measurement error and sampling error are ignored. Test time is 100 second, and the results show perfect performance. The convergence error variance is nearly $1.258e-17$, much smaller than the up limit error variance. However, in practice, those external noise cannot be ignored due to the limit of hardware and change of environment. Next is the external noise is considered, and the performance is shown as follows (Figure 4.3):

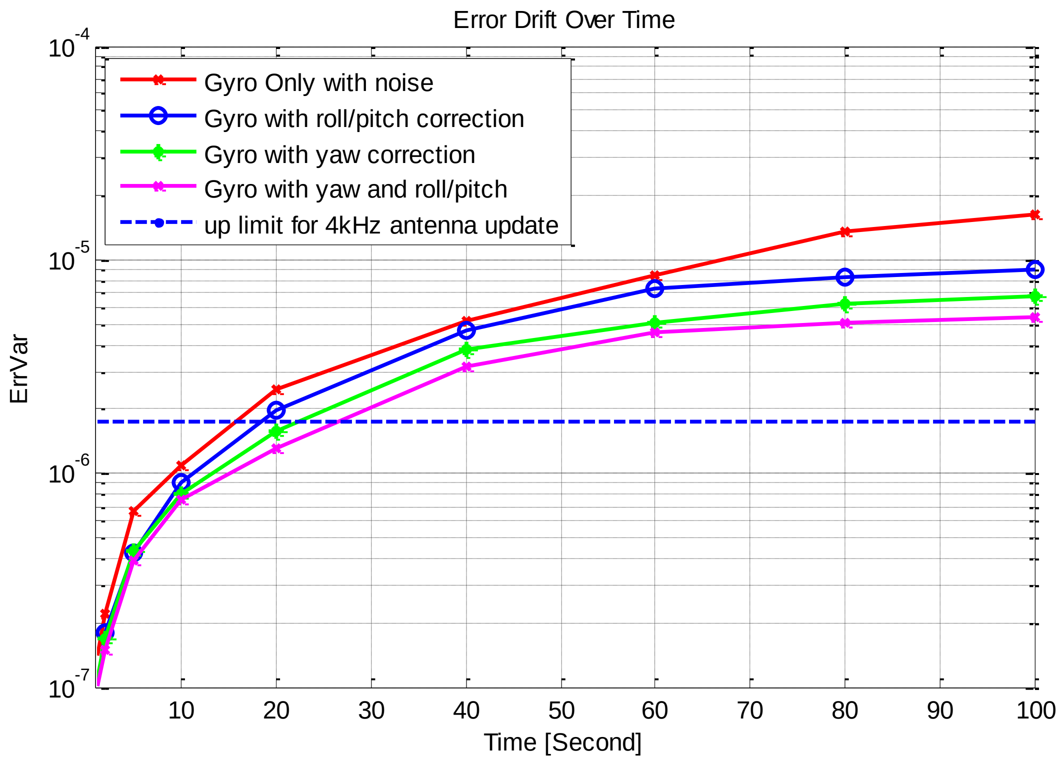


Figure 4.3 Error Variance with Noise

Figure 4.3 shows the performance with the external noise. The blue hidden line expressed the up limit error variance under the antenna update rate 4000Hz. The solid line shows yaw, roll and pitch corrections. The pink line shows the best performance when all yaw, roll, and pitch correction are included. Obviously, the DCM algorithm gets into trouble when the operating environment is rough. The system will lose control after 30 seconds.

There are two ways to improve the performance. One is increase the gyro sampling rate and antenna update rate, and the other is change a more effective algorithm. The first one is hard to realize since the cost of high accuracy hardware is much more expensive than IMU-3000. Thus, Euler Angle Algorithm was came up due to its good correction ability.

4.3 Euler Angle (EA) Performance Analysis

This chapter presents the simulations results of the Euler angler algorithm. The simulation is used to investigate the error characteristics and to analyse the performance of the system with a hypothetical trajectory.

4.3.1 The Hypothetical Trajectory Generation for EA algorithm

To verify the performance of the Euler angler algorithm for the translational and orientation behavior a simulated trajectory is created representing different motions of the rigid body. The simulated trajectory is a fictive path including the position coordinates (x, y, z) of the object defined in the navigation frame, see Figure 4.4. Assume that the rigid body always starts from the original point, and rising gradually along a radian. The AWGN noise is created to describe the extra transient rotation of the rigid body for yaw, pitch, and roll axis, and also add the equivalent sampling error to the AWGN noise to indicate the total error.

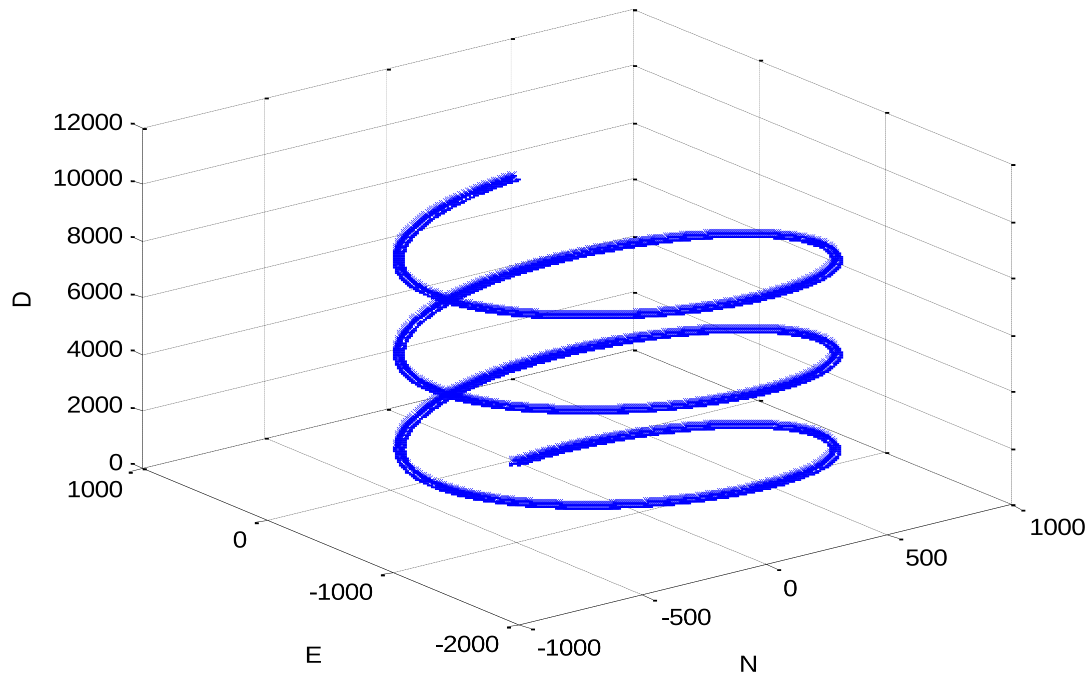


Figure 4.4 Hypothetical Trajectory Defined in The Navigation Frame

4.3.2 The Equivalent Sampling Error

To describe the effect of the sampling, the sampling error must be considered in this case, and in order to simple the system complexity, the sampling error is treated as an equivalent additive white Gaussian noise. Review the simulation in DCM before, according to these data the equivalent noise σ_n^2 is compared with sampling error.

Set the sample rate of gyro to $50Hz$, $200Hz$, $500Hz$, and $1000Hz$, respectively. The noise power is used as 0 which means there is only sampling error. The simulation time is 30s, recall the Equation 4.22: $AvgErrVar = 2*(dt)^2*\sigma_n^2*\frac{time}{dt}$.

The equivalent noise variance is calculated as follows (Table 4.1):

Table 4.1 Equivalent Sample Error for different Sampling Rate

Gyro Sample Rate (Hz)	Equivalent Sampling Error σ_n^2
50	2.436e-10
200	8.112e-11
500	6.466e-11
1000	3.219e-11

From the table above, the equivalent sampling error is very small compare to the IMU 3000's gyro sensor noise variance which is $3e-6$ (Equation 2.8). However, the system error is not just make up of sampling error and the rotation error. GPS and accelerometer also have errors, thus the AWGN power is increased to $1e-4$ to represent the total noise.

4.3.3 Performance Analysis

In the Euler Angle algorithm, there are three factors could affect the performance which are gyro sample rate, noise power, and the trust factor **F** (Equation 3.3). The sampling error is transformed to the equivalent noise power, and it can be ignored since it is very small. The noise power is discussed before which is set to $1e-4$. The trust factor **F**, just as its name implies, is the trust level of this algorithm. One interest to do the simulation is how those factors affect the performance for the case which is the target moving follow a hypothetical trajectory with some rotations caused by jolt. The simulation time is set to 200 seconds, and the trajectory is shown by Figure 4.5.

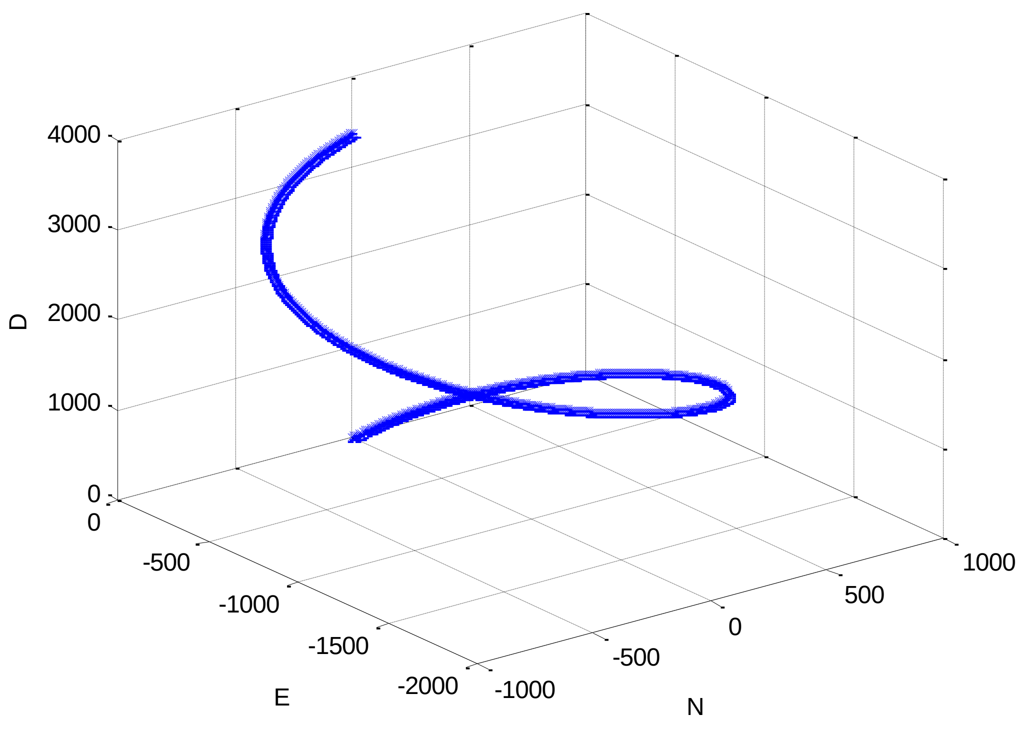


Figure 4.5 Hypothetical trajectory when operating time is 200 seconds

In the Euler Angle algorithm, there is another parameter GRVidx which is usually set to 0 or 1. When it is 0 means perfect knowledge, and there is no measurement error, and of course the trust factor F will be 1. Firstly, the gyro sample rate is set to $50Hz$, and the noise power is $1e-4$.The result is shown as follows(Figure 4.6):

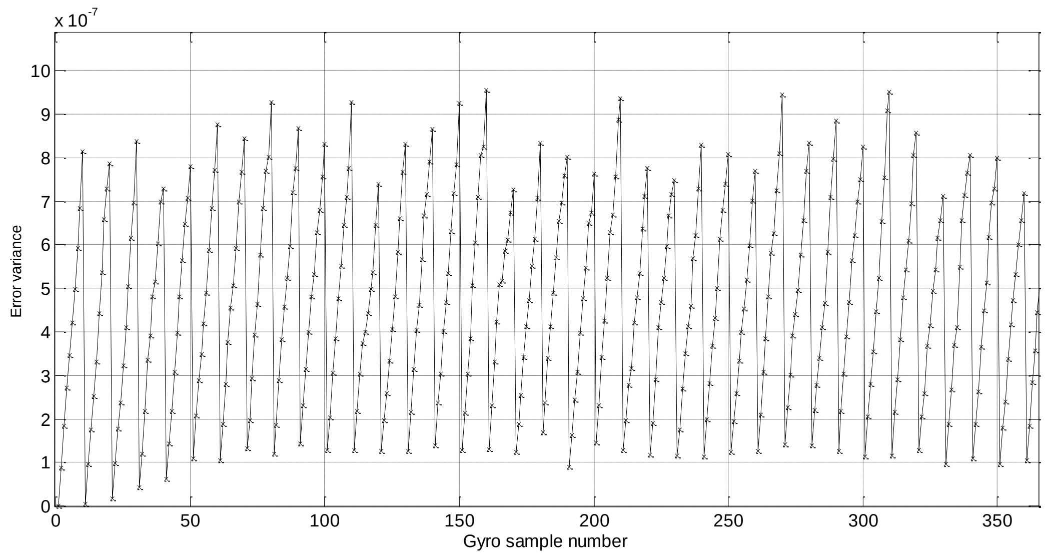


Figure 4.6 Error Variance for Gyro rate = 50Hz , N_power=1.0e-4

From Figure 4.6, for the perfect knowledge, the error variance is not exactly zero. The reason is that the rotation matrix R is not accurate but with noise power, and this will affect the roll, pitch and yaw drift cancellation, and then will affect the error variance.

Thus the performance under this condition with different gyro sample rates is shown as Table 4.2, the average error variance is considered.

Table 4.2 Mean Error when `GRVidx = 0`

Gyro Rate (Hz)	50	200	500	1000
Error Variance	4.83e-7	1.29e-7	5.21e-8	2.62e-8

The corresponding curve will be shown as follows (Figure 4 7):

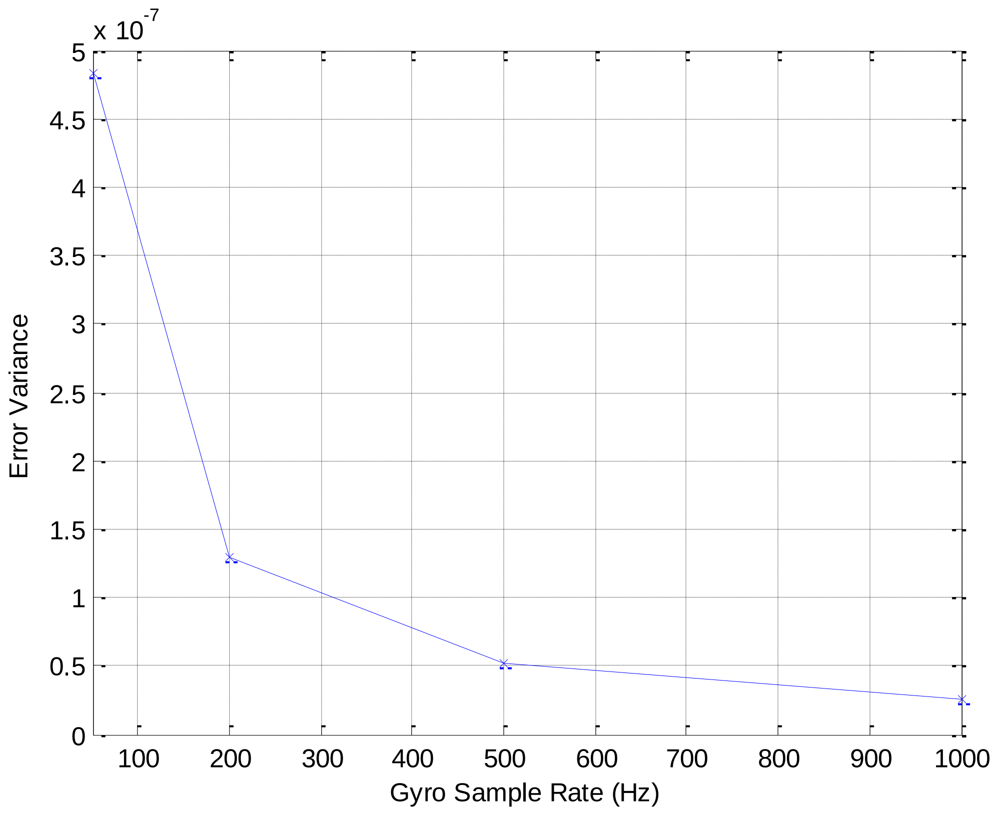


Figure 4 7 Mean Error Curve when `GRVidx = 0`

From the data of the table and the curve above, the error variance is $4.83e-7$ when the gyro sample rate is 50Hz, and the error variance will decrease as the rate is increasing. $50Hz$ is enough for this case. However, this is the case that assume the measurement is perfect, it is impossible in practice. Thus the parameter `GRVidx` is set to 1, which means actual measurement data will be used. Because the measurement data are not exactly accurate, a factor called trust level **F** is introduced, which is between 0 and 1. Two cases are studied, firstly the noise power is $3.0e-6$, and secondly the noise power is $1.0e-4$. The performance under different **F** and gyro sampling rate are shown as follows:

1. Noise power = 3.0e-6

Table 4.3 Error performance under different F and gyro sampling rate

F / Sample Rate (Hz)	1e-3	5e-3	7.5e-3	1e-2	2e-2	5e-2	1e-1
50	1.20e-5	3.00e-5	2.50e-5	2.00e-6	2.20e-6	4.00e-6	6.30e-6
200	2.50e-6	1.20e-6	1.00e-6	1.10e-6	1.60e-6	3.50e-6	7.00e-6
500	1.60e-6	1.00e-6	6.20e-7	8.00e-7	1.58e-6	3.15e-6	6.80e-6
1000	1.00e-6	3.50e-7	2.00e-7	2.54e-7	4.54e-6	2.84e-6	2.20e-6
4000	7.00e-7	2.20e-7	1.80e-7	2.20e-7	4.50e-6	2.50e-6	2.70e-6

The corresponding performance curve is shown as follows (Figure 4.8):

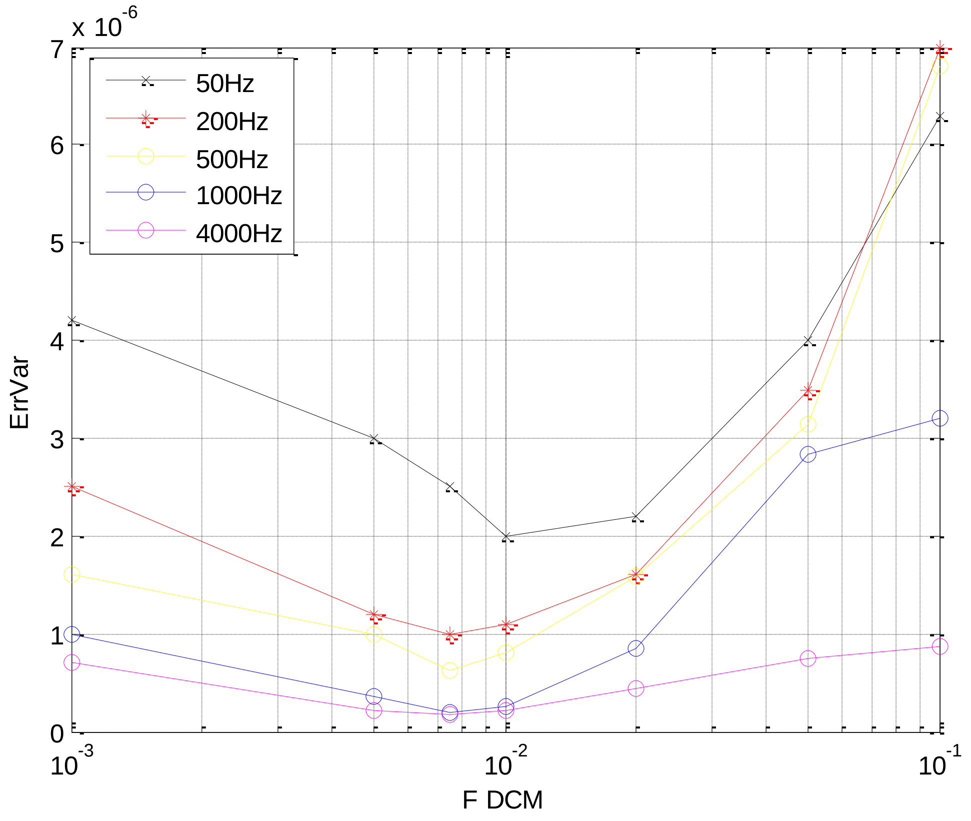


Figure 4.8 Performance Curve for Noise Power = $3.0e-6$

2. Noise power = $1.0e-4$:

Table 4.4 Error Performance under Different F and Gyro Sampling Rate

F / Sample Rate (Hz)	1e-3	5e-3	7.5e-3	1e-2	2e-2	5e-2	1e-1
50	4.00e-4	9.13e-5	7.03e-5	6.35e-5	6.06e-5	1.08e-4	2.30e-4
200	9.00e-5	3.22e-5	3.00e-5	3.50e-5	4.96e-5	1.20e-4	2.50e-4
500	3.60e-5	2.00e-5	2.20e-5	3.00e-5	4.58e-5	1.15e-4	2.30e-4
1000	2.00e-5	1.50e-5	2.00e-5	2.54e-5	4.54e-5	1.14e-4	2.20e-4
4000	7.00e-6	1.20e-5	1.80e-5	2.20e-5	4.50e-5	1.00e-4	2.10e-4

The corresponding performance curve is shown as follows (Figure 4.9):

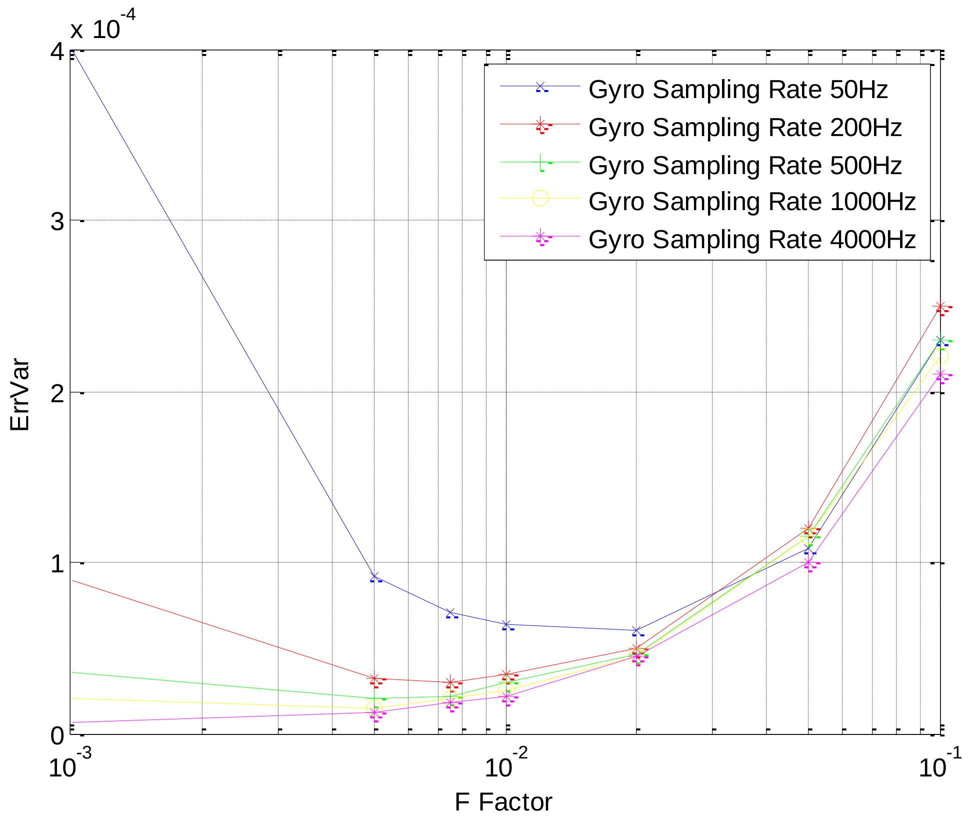


Figure 4.9 Performance Curve for Noise Power = $1.0e-4$

From the table and curve above, when the gyro rate is increasing, the performance is better as the trust factor is smaller. However consider all the data in theTable 4.4, the smallest error order is e-5 when the sampling rate is 1000Hz, this will not match the up limit error variance as shown (Table 4.5):

Table 4.5 Up limit Error Variance for Antenna Update

Update rate (Hz)	$\sigma_\theta^2(rad)$	$3\sigma_{Nxx}^2$
4000	$8.67\times10^{-7}rad$	$1.73\times10^{-6}rad$
8000	$2.76\times10^{-6}rad$	$5.51\times10^{-6}rad$

As shown in Table 4.5, the up limit error for antenna update rate 8000Hz is $5.51\times10^{-6}rad$, and for 4000Hz is $1.73\times10^{-6}rad$.

The two ways to improve the performance are to increase the gyro sampling rate and antenna update rate. From Figure 4.9 that the performance has not been improved so much when the gyro sampling rate was increased to higher values. The other way is increase the antenna update rate to extend the up limit error variance. However, the update rate of antenna is not easy to increase, it needs more computation power and has higher requirement of the chip. Continuing to calculate the up limit when the update rate is 12kHz, 15kHz, and 20kHz. The result is shown as follows(Table 4.6):

Table 4.6 Up limit Error Variance for Higher Antenna Update Rate

Update rate (Hz)	$\sigma_\theta^2(rad)$	$3\sigma_{Nxx}^2(rad)$
12k	$3.61e-6$	$7.22e-6$
15k	$3.99e-6$	$7.98e-6$
20k	$4.39e-6$	$8.78e-6$

When the antenna update rate is increased to 20kHz, the up limit error variance will also increase to $8.78e-6$. From Table 4.4 that the error variance is 7e-6 when the gyro sampling rate is 4000Hz and F = 1.0e-3. So one feasible solution is use the gyro which sampling rate is 4000Hz and antenna update rate is 12kHz. The final results are shown as follows (Table 4.7):

Table 4.7 Final results

	F	**Sampling Rate**	**Noise**	**Up limit Error variance**	**Error Variance**
Perfect Case	1	50Hz	1.0e-4	7.22e-6	4.83e-7
Non Perfect Case	1.0e-3	4000Hz	1.0e-4	7.22e-6	7.00e-6

CHAPTER 5 Implementation of Program

In this chapter, the hardware and software platform will be introduced first. Next are the implementation of the algorithm and some test in practice. The bit error analysis compare to the rotation error is presented at last.

5.1 Hardware and Software Platform

5.1.1 ArduPilot Mega

ArduPilot Mega (APM) is an open source, Arduino-compatible, pro-quality autopilot. It can be used for either airplanes or land vehicles.
The APM is constituted by two parts: the first one is APM main processor board, and the second is the IMU shield. The main processor board is shown as follows(Figure 5.1):

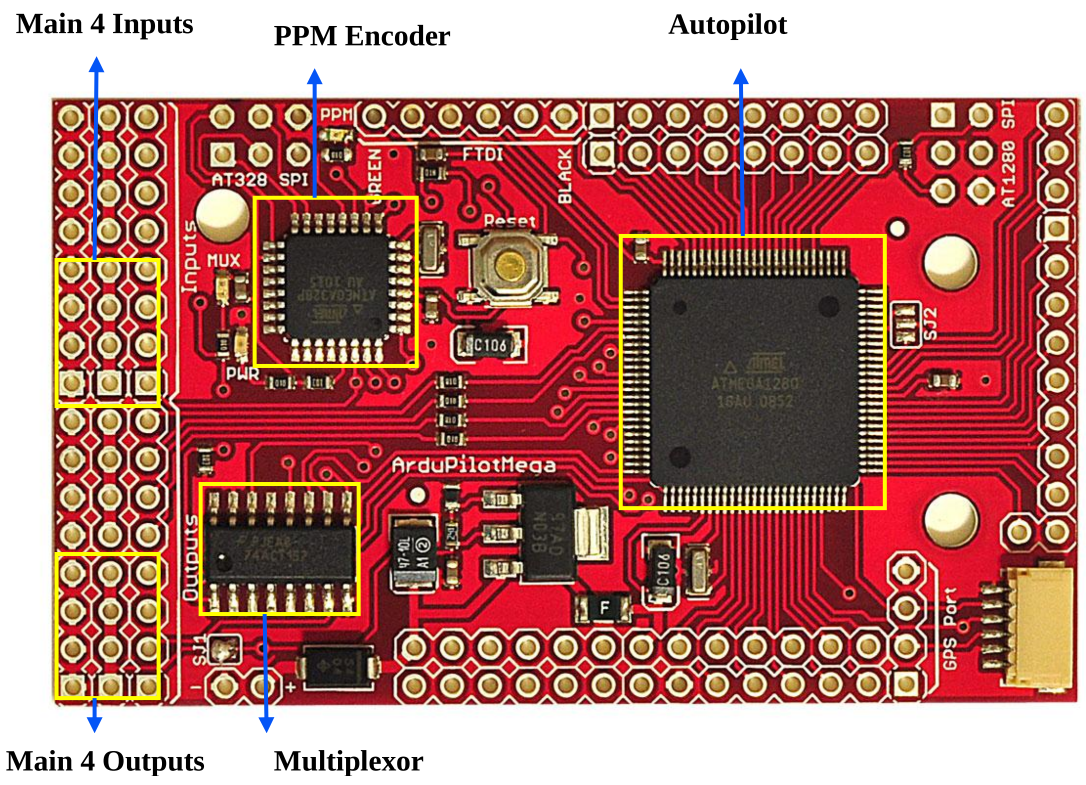


Figure 5.1 APM main processor board

The fundamental features of main processor board is :

- ➢ Based on a 16MHz Atmega1280 processor.
- ➢ Dual-processor design with 32 MIPS of onboard power.
- ➢ 128k Flash Program Memory, 8K SRAM, 4K EEPROM.

- Has 16 spare analog inputs (with ADC on each) and 40 digital input/outputs to add additional sensors, and comes with a 6-pin GPS connector (EM406 style).

The second part of APM is IMU shield (see Figure 5.2). This board features a large array of sensors needed for UAV and Robotics applications, including three axis angular rotation and accelerations sensors, absolute pressure and temperature sensor, 16MBits data logger chip. It's designed to fit on top (or bottom) of the ArduPilot Mega board, creating a total autopilot solution when a GPS module is attached.

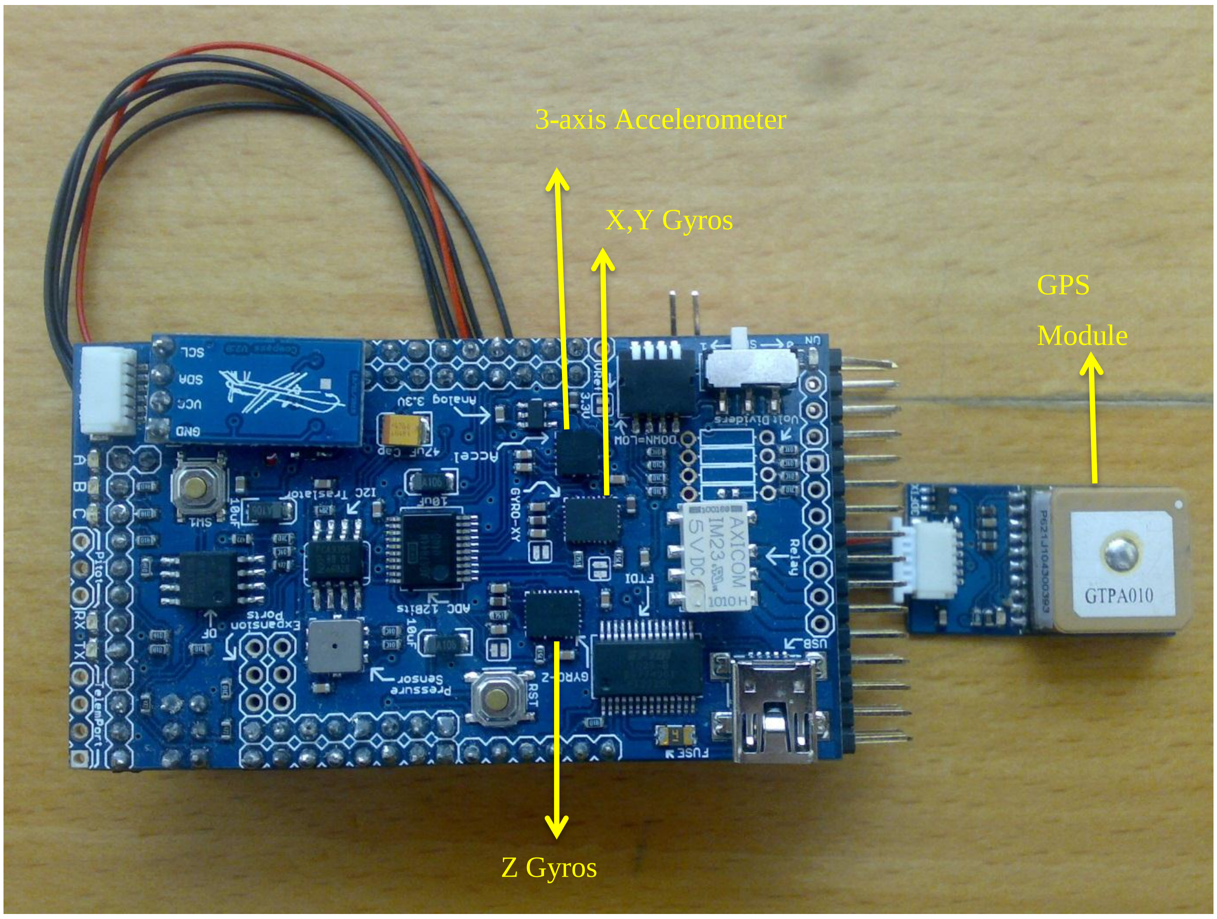


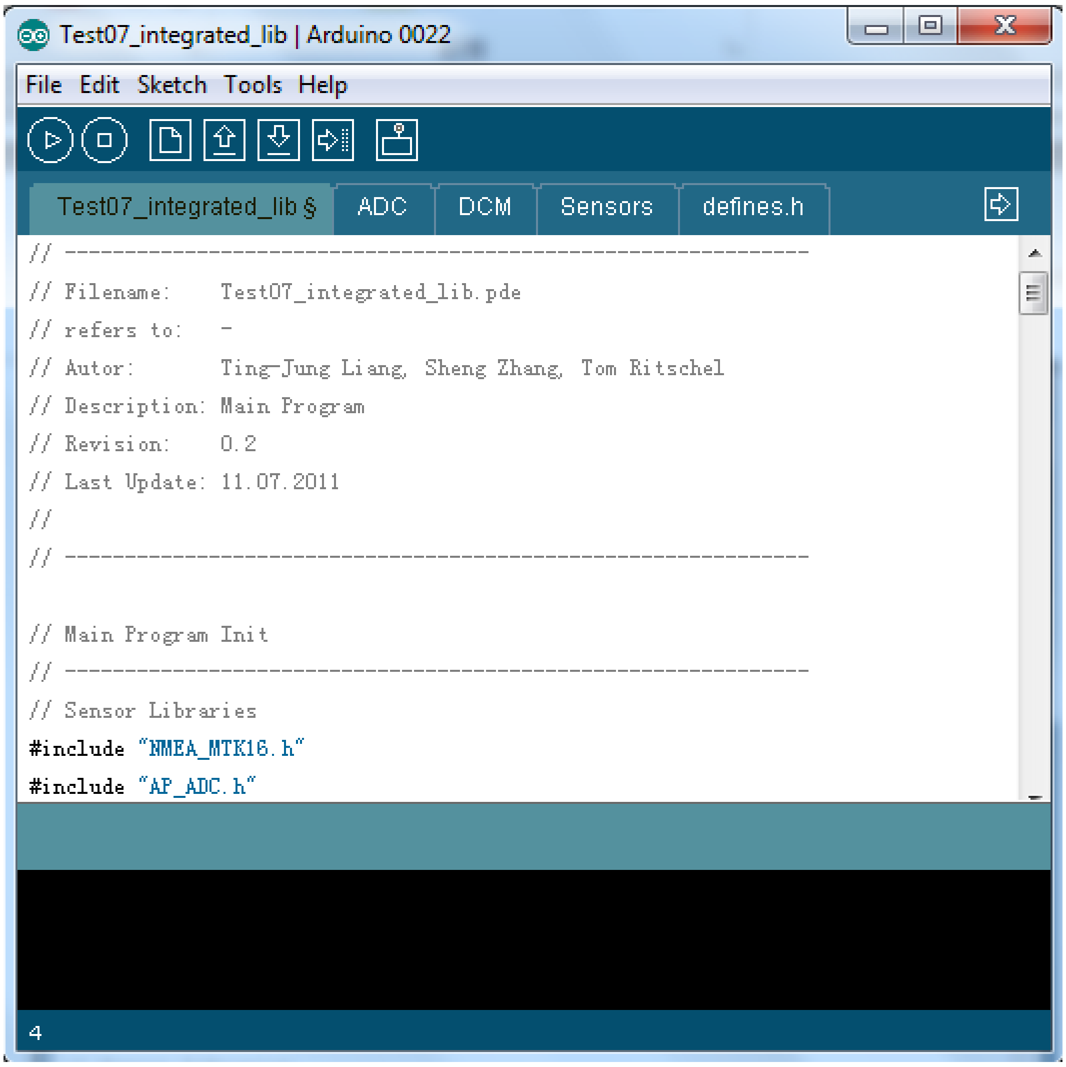
Figure 5.2 IMU Shield

All information is obtained from the DIY Drones open source website: http://code.google.com/p/ardupilot-mega/

5.1.2 Arduino Software Platform

The Arduino software consists of a development environment (IDE) and the core libraries. The IDE is a cross.platform application written in Java and based on the Processing development environment and Wiring project. It includes a code editor and is capable of compiling and uploading programs to the board with a single click. The core libraries called ’Wiring’are written in C and C++ and compiled using avr-gcc and AVR Libc.

By using Arduino language, the Euler Angle algorithm is implemented in this ArduPilot Mega Board to provide the accurate rotation information. A window of the operating interface of the Arduino Software Platform is shown as follows(Figure 5.3):


Test07_integrated_lib | Arduino 0022
File Edit Sketch Tools Help
Test07_integrated_lib §
ADC
DCM
Sensors
defines.h
4

```
// ---------------------------------------------------------------
// Filename:    Test07_integrated_lib.pde
// refers to:   -
// Autor:       Ting-Jung Liang, Sheng Zhang, Tom Ritschel
// Description: Main Program
// Revision:    0.2
// Last Update: 11.07.2011
//
// ---------------------------------------------------------------

// Main Program Init
// ---------------------------------------------------------------
// Sensor Libraries
#include "NMEA_MTK16.h"
#include "AP_ADC.h"
```

Figure 5.3 Arduino Software Platform

5.2 Some Results of Implementation

5.2.1 Practical Signal Analysis

In previous chapters, all analysis is giving in the simulation, including the gyro signal generation and the external noise. However, if the model is right or wrong and how the real gyro signal looks like are interesting and necessary to make sure that the results of the simulation are meaningful. After implementation of the EA algorithm to the ArduPilot Mega (APM) board, the measurement can show the real signal and external noise clearly. The test is divided into two different parts : the first case is that the vehicle travels on a smooth road, and it goes along a straight line, and the second one is that the vehicle goes along a 'S' pattern, and the road is not smooth, there are many pebble on it, result in the vehicle will shake severely. The signal received from the sensor are rotation angles expressed in some numbers, and stored with a 'txt' file in the flash memory of the ArduPilot Mega board. Import these files into MATLAB and use Welch Power Spectral Density Estimation, the spectrum of the signal could be seen clearly. Next is the spectrum and the analysis of these signals :

Case 1 :

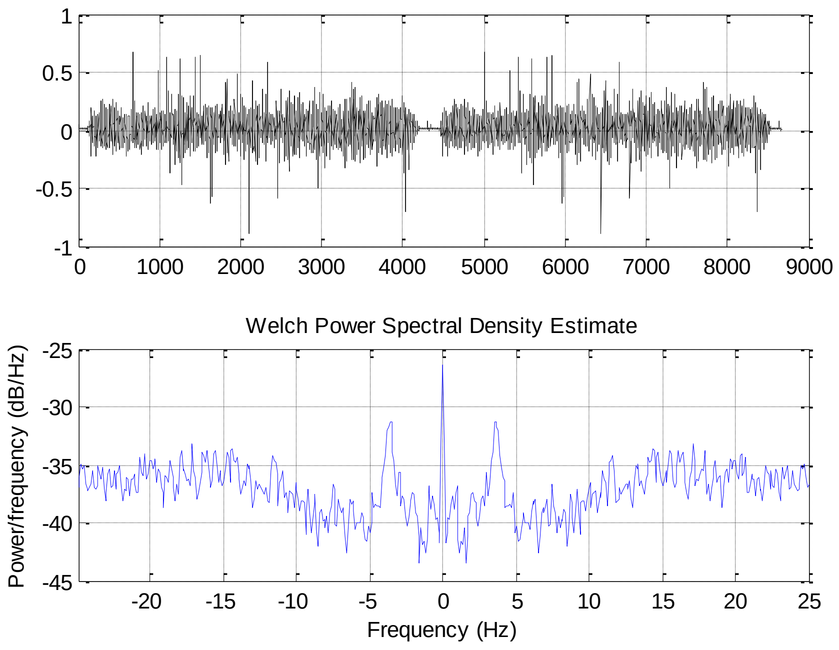


Figure 5.4 Case1:x_axis(Roll) Signal and Spectrum

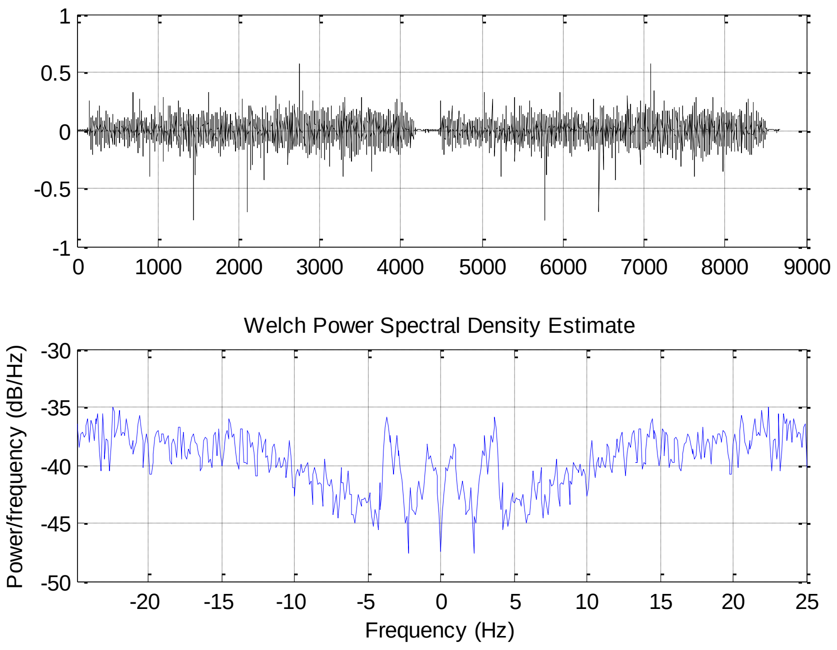


Figure 5.5 Case1:y_axis(Pitch) Signal and Spectrum

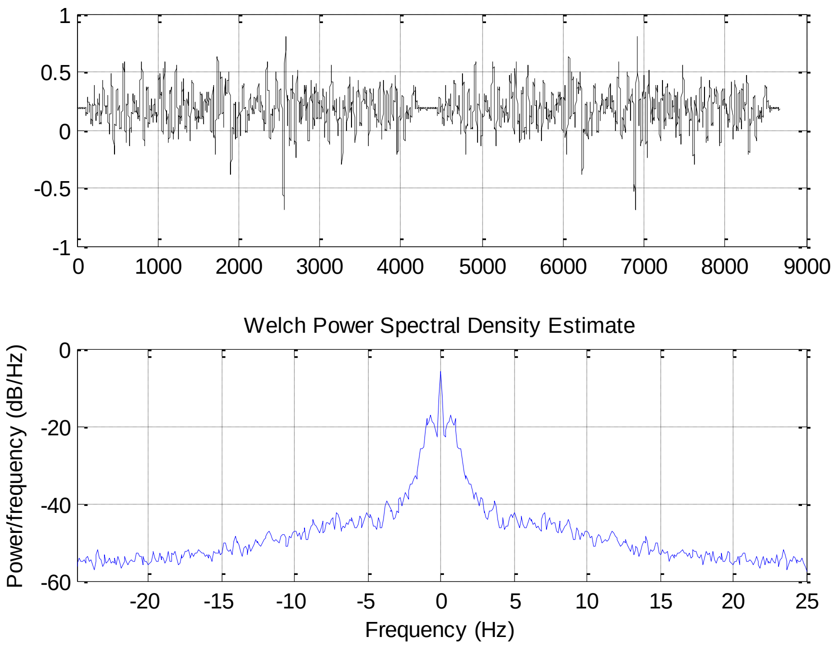


Figure 5.6 Case1:z_axis(Yaw) Signal and Spectrum

From the figures above, the rotation angle speed of the roll and pitch axis are very small, the average spectrum value is around -35dB and -40dB, respectively. See the time domain, for the roll axis which is 0.0172 rad/sample, i.e.0.985 degree/sample, and for pitch axis, the mean rotation angle speed is 0.0211 rad/sample. However, the vehicle is not going exactly followed the straight line, due to some deviations, the peak value of the yaw axis could be found in the Figure 5.5 above, the mean value in z axis which is 0.1857 rad/ sample, i.e. 10.6 degree/sample. Another thing need to be considered that around 4000 samples, the value of spectrum is nearly 0 dB which means the vehicle was stopped for a little while during the test process.

This is simplest case, form this case, the amplitude of the shake is very small, which means there is not so much rotations during the operating process, and the small rotation error can be corrected easily. It is easy to model this condition, and it can be include in the case which is worse one.

For second case, the operating environment is worse than the first one, and is closer to the case in the project. The purpose of this test is to see the real signal properties, and compare with the signal used in the simulation. Next is the signal from the second test case:

Case 2:

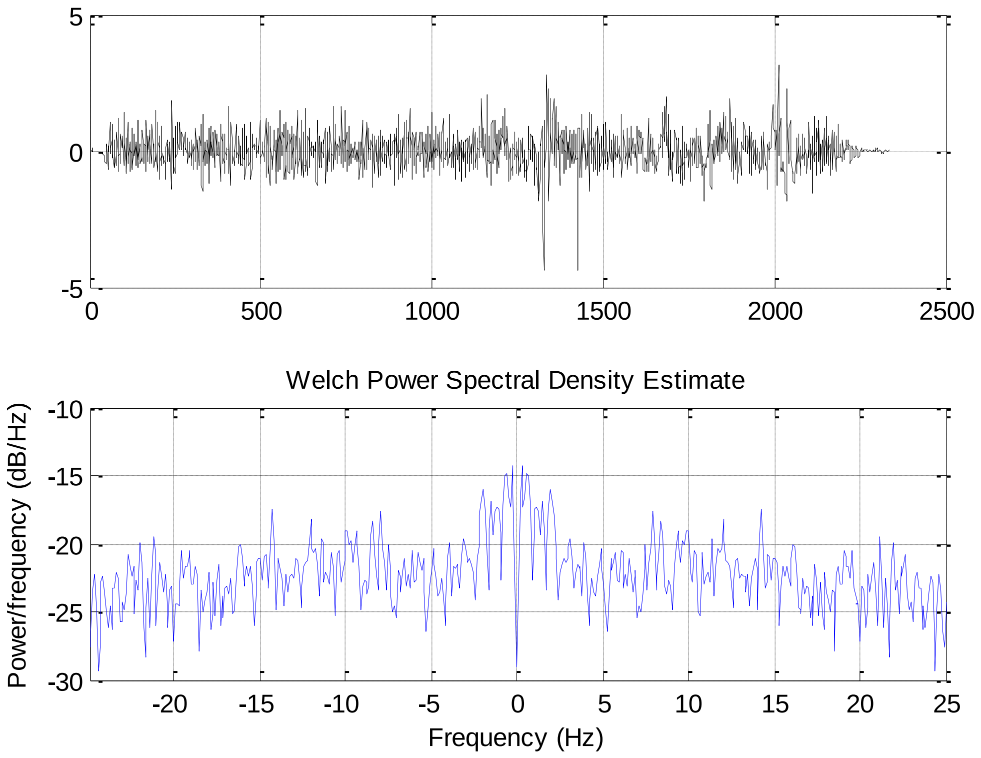


Figure 5.7 Case2: x_axis(Roll) Signal and Spectrum

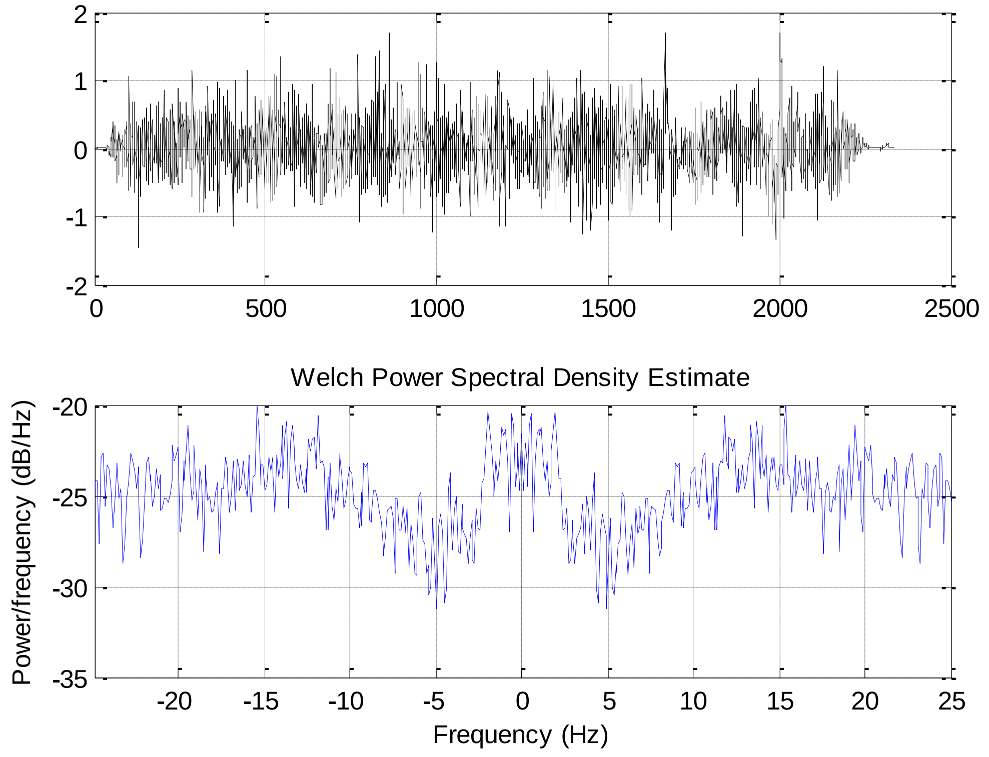


Figure 5.8 Case2: y_axis(Pitch) Signal and Spectrum

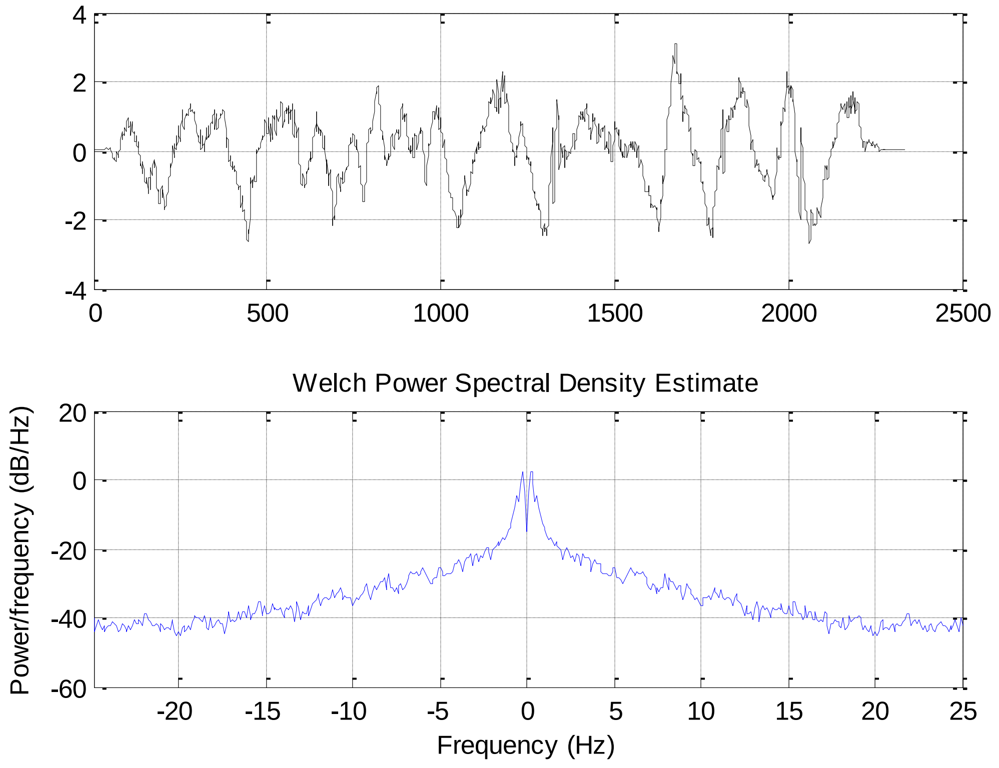


Figure 5.9 Case2: z_axis(Yaw) Signal and Spectrum

The difference between case 2 and the cases 1 is the signal values of roll, pitch, and yaw rotation are much higher. For x (roll) axis, the maximum value is 3.1586 rad/sample, and the level of the spectrum is higher than other cases. The same with y(pitch) and z(yaw) axis, and the frequency of shaking is faster than others. So case 2 could be a good reference to model the gyro signals which is received in a bad environment.

The idea of modeling is assume two kinds of AWGN signals which will pass through two different filters (different parameters), and the sum of the two signals is the final signal. Consider that the peak value of the z(yaw) axis is 3.112 rad/sample. The MATLAB simulation code can be found in Appendix, and the result of simulation is shown as follows, only use z-axis.

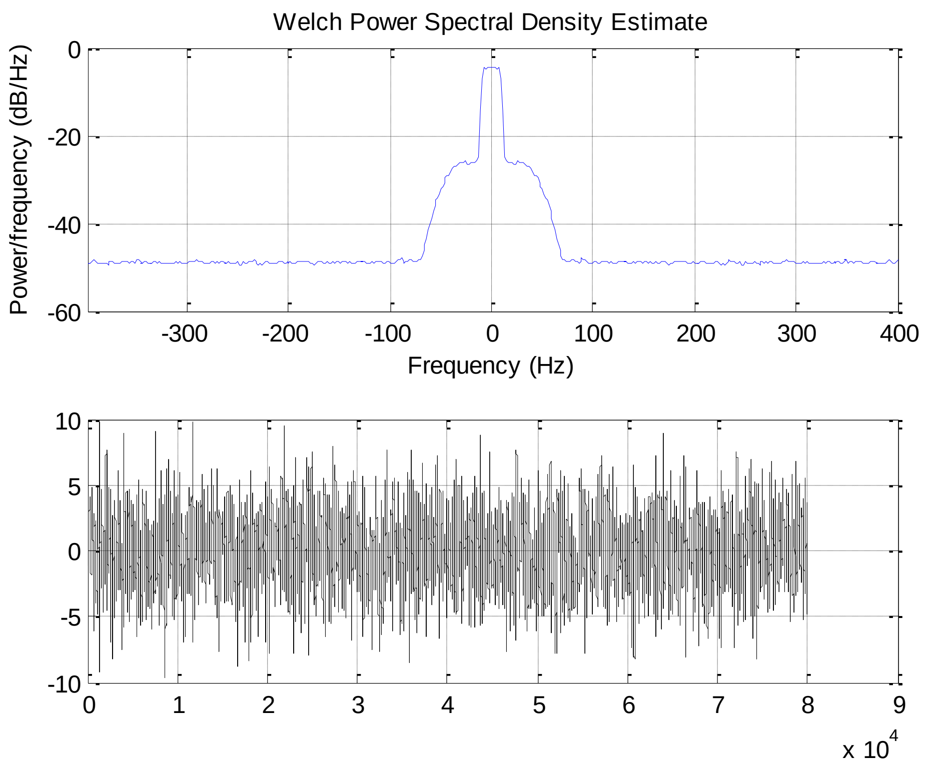


Figure 5.10 Simulation: z_axis(Yaw) Signal and Spectrum

In the simulation, the peak value of z-axis is 10 rad/sample, which is much more than the second case, but is closer to the case in this project. From the analysis results, it can be seen that the model can well simulate the practical environment.

5.2.2 Bit Error Rate (BER) Analysis

This system will apply to the satellite communication, then the bit error rate (BER) and the signal noise ratio (SNR) are the valuable parameters to evaluate the performance of system in practice. The test will build a communication system to find SNR loss. The SNR loss is the difference between the reference SNR and the actual SNR under the same BER. The specific realization is shown as follows(Figure 5.11):

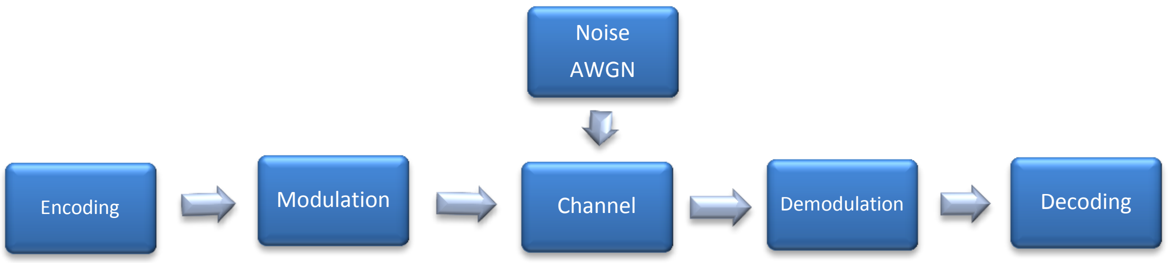


Figure 5.11 Communication System Channel Model

The reference SNR curve can be found by using MATLAB function '**berub = bercoding(EbNo,'conv',decision,coderate,dspec)**', this returns an upper bound or approximation on the BER of a binary convolutional code with coherent phase shift keying (PSK) modulation over an additive white Gaussian noise (AWGN) channel[13]. Here upper bound means the best performance of the BER at a definite SNR. The channel coding is using convolutional code with constraint length 7. The decoding is used Viterbi algorithm[14]. Plot the BER curve over AWGN channel with different error variance, and compare with the reference BER curve, the corresponding SNR loss could be calculated. The results are shown as follows:

Table 5.1 BER values with different SNR under different Error variances

SNR (dB) / Error	0	1	2	3	4	5	6	7	8
2.5e-3	3.757e-1	2.624e-1	1.435e-1	4.98e-2	1.09e-2	1.58e-3	1.26e-4	7.20e-6	2.40e-7
2.5e-4	3.659e-1	2.560e-1	1.347e-1	4.77e-2	1.03e-2	1.53e-3	1.17e-4	6.68e-6	2.21e-7
2.5e-5	3.460e-1	2.391e-1	1.312e-1	4.34e-2	9.70e-3	1.46e-3	1.09e-4	6.21e-6	2.01e-7
2.5e-6	3.280e-1	2.211e-1	1.160e-1	4.21e-2	9.40e-3	1.38e-3	1.02e-4	5.90e-6	1.82e-7

Figure 5.12 shows the corresponding BER curve with different SNR under different Error variances:

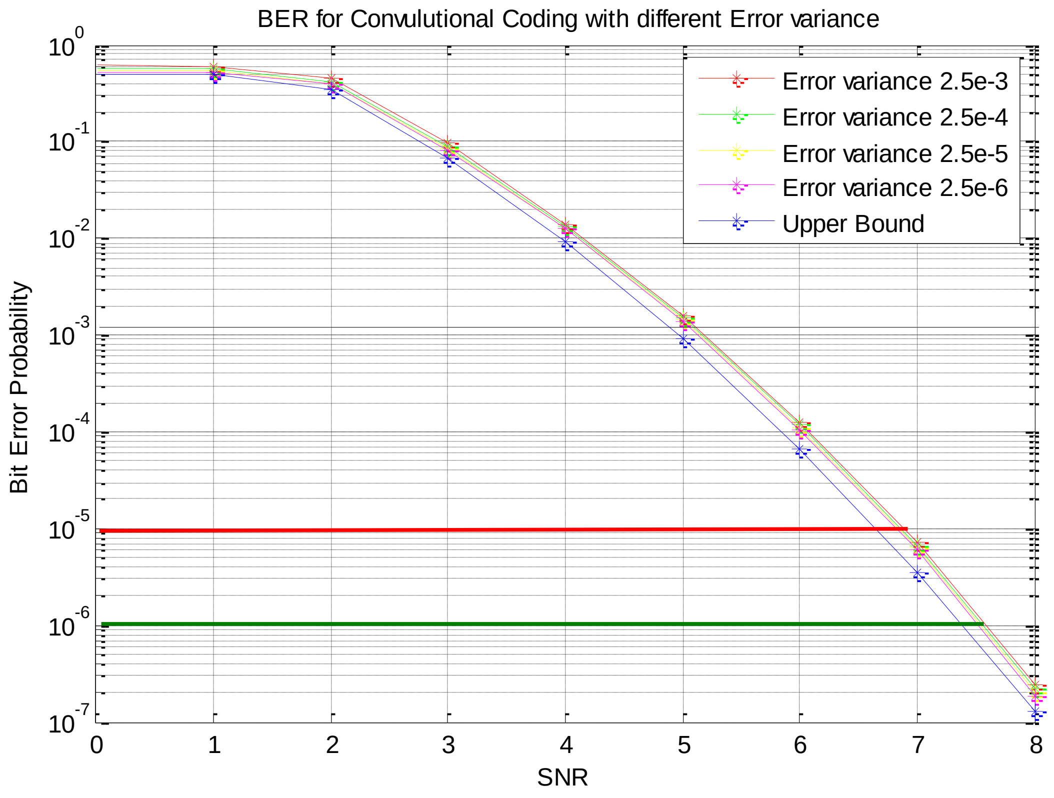


Figure 5.12 BER curve for different SNR under different Error variance

The corresponding SNR loss is shown as follows (Table 5.2), when the BER is equal to 1.0e-5:

Table 5.2 SNR Loss when BER is 1.0e-5

Error variance	2.5e-6	2.5e-5	2.5e-4	2.5e-3	Theory
SNR (dB)	6.815	6.834	6.859	6.885	6.637
SNR loss(dB)	0.178	0.197	0.222	0.248	0

And when the BER is equal to 1.0e-6, the corresponding SNR loss is shown as follows(Table 5.3):

Table 5.3 SNR Loss when BER is 1.0e-6

Error variance	2.5e-6	2.5e-5	2.5e-4	2.5e-3	Theory
SNR (dB)	7.510	7.532	7.557	7.580	7.374
SNR loss(dB)	0.136	0.158	0.183	0.206	0

The corresponding curve is shown as follows(Figure 5.13):

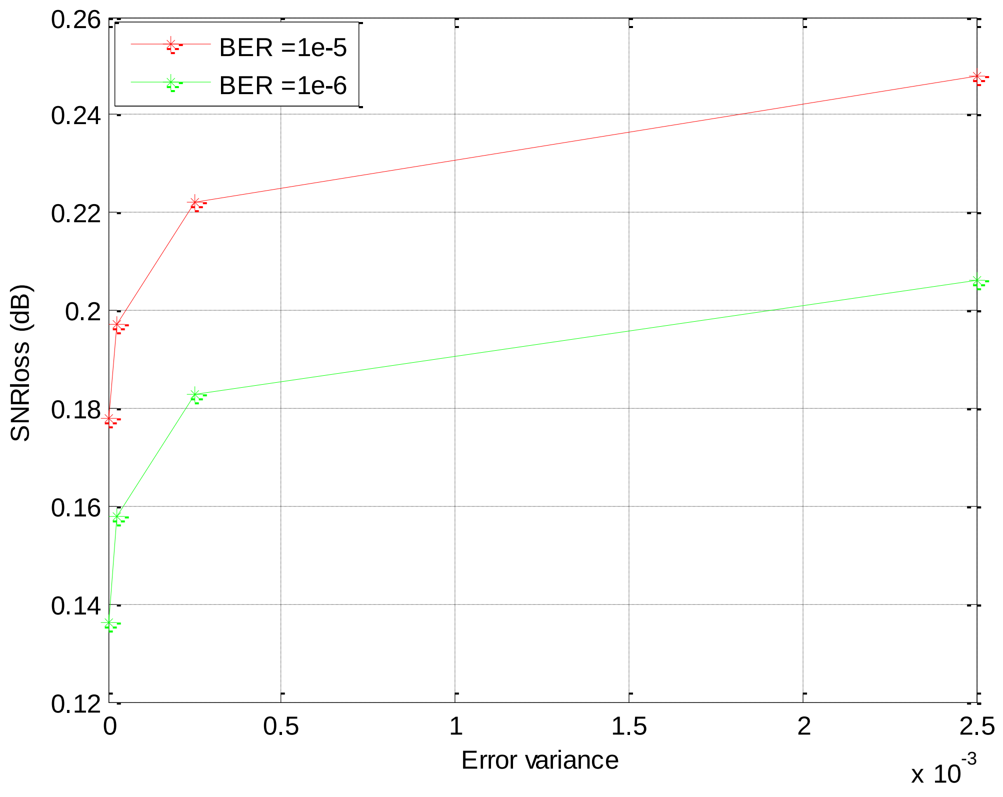


Figure 5.13 SNR Loss with Different Error Variance when BER are 1.0e-5 and1.0e-6

From above, adjust the SNR loss at the receiver antenna, the corresponding error variance can be known. In this project, the SNR loss is 0.178 dB , the error variance is 2.5e-6, when the BER is 1.0e-5, this can satisfy the system requirement well.

CHAPTER 6 Conclusions and Recommendations

6.1 Conclusions

The work presented in this thesis dealt with thorough assessment of performance of a beam tracking system for land vehicle navigation (LVN) application by using a GPS/INS integrated system. Primary advantages of GPS/INS integration is that the INS predicted positions and velocities are available during GPS outages, and GPS can restrict the INS's drift over time, and allows for online estimation of the sensor errors.

Two methods were tested in simulation to estimate the performance of the system, and Euler Angle method was chosen to implement in practice with an ArduPilot Mega (APM) board. The motivation behind using the EA method is the fact that the poor operating environmental conditions in this project. The vehicle would bump and turn off quickly with maximum angle speed and angle acceleration speed are are $1000^o / s$ and $1500^o / s^2$, respectively. The DCM algorithm is not suitable to this case since the PI feedback controller cannot correct the error timely in a high-speed rotation movement. Whereas the Euler Angle Algorithm can improve the performance remarkable. The primary advantage using this approach to restrain the error variance of the system is that not so much high requirement for the sensors, which keeps the system cost low. A 4000Hz sampling rate gyro and 12 kHz update antenna is enough.

The primary piece of equipment used was an ArduPilot Mega (APM) board and a GPS receiver model. Furthermore, in order to see the relation between the error variance and the signal noise ratio (SNR) at the receiver port of the antenna. A whole communication channel was built, using convolutional code with constraint length which is 7, and BPSK modulation. The reference SNR could be found by 'bercoding' function in MATLAB, this returns an upper bound or approximation on the BER of a convolutional code with binary coherent phase shift keying (BPSK) modulation over the BER at a definite SNR. Then the SNR loss with different AWGN error variances between the real SNR and reference SNR could be found. When the SNR loss is $0.178\,dB$, the real SNR at the receiver port is $6.815\,dB$, the corresponding error variance is $2.5e-6$ which is under the up limit error variance $7.22e-6$ when the receiver antenna update rate is 12 kHz. The system can work well under this conditions in the poor operating environment.

6.2 Recommendations

The following recommendations can be made for future investigation for improving the performance of the system:

1. When test the system in the lab, the correction effect of the system became worse due to the existence of many electronic devices such as computers and spectrum analyzer, etc. This phenomenon is called ‘Soft Iron Interference’. ‘Soft iron’ Interference is created by the induction of a temporary magnetic field into normally un-magnetized ferromagnetic components, such as steel shields and batteries, on the PCB by the geomagnetic field.[11] Thus, when the vehicle is running in the highly magnetic region of the Earth such as mine, it is a problem that the if the system can still work well or not. And how to cancel this ‘Soft Iron Interference’ is an challenge in the future investigation.

2. Beside the Euler Angle Algorithm, Kalman filter is another effective way to estimate the performance of the system. A Kalman filter is a recursive algorithm that uses a series of prediction and measurement update steps to obtain an optimal estimate of the state vector which has a minimum variance [3]. Kalman filtering provides a powerful tool to create synergism between two navigation sensors - GPS receivers and INS - since it is able to take advantage of both systems characteristics to provide a common, integrated navigation implementation with a performance superior to either of the sensor subsystems [12]. However, it is not easy to design a Kalman filter for the LVNS, especially in this project. Since the high-speed rotation movement, an nonlinear Kalman filter need to be designed for prediction and correction. It is an interesting and challenging topic for the further investigation, too.

References

[1] E.W. Anderson, *The principles of navigation*, Second Edition, Hollis and Carter,1966.

[2] David Titterton and John Weston, Eds. *Strapdown Inertial Navigation Technology*, McGraw-Hills, pp.2-3. 2004,

[3] Saurabh Godha., "Performance Evaluation of Low Cost MEMS-Based IMU Integrated With GPS for Land Vehicle Navigation Application," Master Thesis, Geomatics Engineering, University of Calgary, pp.39, February, 2006.

[4] Mobinder S. Grewal, Lawerence R. Weill and Angus P. Andrews, Eds., *Global Positioning Systems, Inertial Navigation, and Integration*, Second Edition, John Wiley & Sons, Inc., pp. 316-320, January, 2007.

[5] Willuam Premerlani and Paul Bizard, "Direction Cosine Matrix IMU: Theory," DIY DRONE: USA, pp.13-15, 2009.

[6] Mahony R, Sung-Han C, and Hamel T, "A coupled estimation and control analysis for attitude stabilisation of mini aerial vehicles," *Australasian Conference on Robotics and Automation*, vol. 24, no. 12, Dec. 2006, pp. 101-108.

[7] Grant Baldwin, Robert Mahony, Jochen Trumpf, Tarek Hamel & Thibault Cheviron, "Complementary filter design on the Special Euclidean group SE(3)", *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference*, Spain, December 12-15, 2005, pp. 165-171

[8] Mark Euston, Paul Coote, Robert Mahony, Jonghyuk Kim and Tarek Hamel, "A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV," IROS 2008, pp. 340-345.

[9] Christopher. Jekeli, *Inertial Navigation Systems with Geodetic Applications*, Second Edition, Walter de Gruyter, Berlin, New York, pp. 89-90, 2001.

[10] Talat. Ozyagcilar, "Layout Recommendations for PCBs Using a Magnetometer Sensor". *Freescale Semiconductor Application Note*, Document Number: AN4247.

[11] Xiaohong Zhang, "Integration of GPS with A Medium Accuracy IMU for Metre-Level Position," Master Thesis, Geomatics Engineering, University of Calgary, pp. 2-3, May, 2003.

[12] Matlab Help, Communication Toolbox, "BER for coded AWGN channels."

[13] Proakis, J. G., *Digital Communications*, 4th Edition, New York, McGraw-Hill, pp. 99-110, 2001

[14] A. J. Viterbi, "Convolutional codes and their performance in communicationsystems," IEEE Trans. Inf. Theory, vol. IT-19, no. 5, pp. 751–772, Oct. 1971.

Appendix A

Matlab Code for antenna pattern:

```
close all;
clear all;

idx = (-180:0.1:180)/180*pi;
M = 51;
phi = 60/180*pi;
x = M/2*cos(phi).*sin(idx);
y = M/2*sin(phi).*sin(idx);
a = sinc(x).*sinc(y);
g = 10*log10(a.^2) + 10*log10(M^4) + (-10*log10(M^4)+16+36);
plot(idx*180/pi,g,'-r'); grid on; hold on;
ylim([-10 60]);
xlim([-20 20]);
```

Matlab Code for DCM Parameters Definition:

```
% ##########################################################################
% Direction Cosine Matrix (DCM) Algorithm
% Parameters Definition
% Author:      Ting-Jung Liang Sheng Zhang
% Last Update: 22 Feb 2011
% ##########################################################################
close all;
clear all;

% ##########################################################################
% Initialization:
% 1) Align NED (North-East-Down) to XYZ (Roll-Pitch-Yaw) at time zero
%    Q_body = R Q_earth
% 2) Calculate initial satellite direction in NED axis
% 3) System parameters
%    Note1: loop_gyro/loop_GPS must be an integer!
% 4) Input gyro signals
% ##########################################################################
% Alignment ----------------------------------------------------------------
R0 = [1 0 0; 0 1 0; 0 0 1];

% Satellite Direction ------------------------------------------------------

% System parameters --------------------------------------------------------
time      = 100;                % Second
loop_gyro = 50;               % Samples per second (gyro)
loop_GPS  = 5;                % Samples per second (GPS)
dt        = 1/loop_gyro;      % Integral interval
Os        = 100;

% Control parameters -------------------------------------------------------
%ControlCase = 1:4;    % '1': Gyro; '2': + GPS (Yaw) Control+ACCmeter
Control
Weight_rp   = 0.1;
Weight_yaw  = 1;
Kp_RollPitch = 1.4e-3;% 1.4e-3 in Arduino code
Ki_RollPitch = 3e-7;  % 3e-7 in Arduino code
Kp_Yaw      = 0.8;    % 0.8 in Arduino code
```

```
Ki_Yaw      = 1e-5;   % 4e-5 in Arduino code

% Gyro signals --------------------------------------------------------------
Omega_x = 15*sin((2*pi)*1*(1: time/dt*Os));
Omega_y = 15*sin((6*pi)*3*(1: time/dt*Os));
Omega_z = 15*sin((4*pi)*2*(1: time/dt*Os));
%Omega_x = randn(1, time/dt*Os);
%Omega_y = randn(1, time/dt*Os);
%Omega_z = randn(1, time/dt*Os);   % Angular speed
% Velocity over Ground ------------------------------------------------------
g  = 9.81;  % 9.81 m/s^2
Vx = 30+10*sin((2*pi)/(time/dt*Os)*[1: time/dt*Os]);  % m/s; ps: 100km/hr =
27.8m/s

% IMU Simulation -----------------------------------------------------------
it_max = 100;   % Iteration number for averaging
NPower = 0.01;  % Gyro AWGN Noise
ErrVar = 0;     % Initial error
```

Matlab Code for DCM Algorithm:

```
% ##########################################################################
% Direction Cosine Matrix (DCM) Algorithm
% Gyro signal with Roll/Pitch correction
% Author:      Ting-Jung Liang Sheng Zhang
% Last Update: 22 Feb 2011
% ##########################################################################
close all;
clear all;

% ##########################################################################
% Initialization:
% 1) Align NED (North-East-Down) to XYZ (Roll-Pitch-Yaw) at time zero
%    Q_body = R Q_earth
% 2) Calculate initial satellite direction in NED axis
% 3) System parameters
%    Note1: loop_gyro/loop_GPS must be an integer!
% 4) Input gyro signals
% ##########################################################################
DCM_Para;
% ##########################################################################

% ##########################################################################
% Calculate (precise) reference signal
% 1) Course over ground (COG: GPS) for Yaw correction
% 2) Accelerometers for Roll/Pitch correction
% ##########################################################################
R1        = R0;
ACCmeter = zeros(3,time/dt*Os);
for i=1:time/dt*Os
    % Rotation Matrix with Os ---------------------------------------------
    % Kinematics
    dPx = Omega_x(i)*dt/Os;
    dPy = Omega_y(i)*dt/Os;
    dPz = Omega_z(i)*dt/Os;
    Ru  = [1 -dPz dPy; dPz 1 -dPx; -dPy dPx 1];
    R1  = R1*Ru;
    % Normalization
    X   = R1(1,:);
    Y   = R1(2,:);
    err = X*Y';
    Xo  = X - (err/2)*Y;
    Yo  = Y - (err/2)*X;
    Zo  = [Xo(2)*Yo(3)-Xo(3)*Yo(2)...
           Xo(3)*Yo(1)-Xo(1)*Yo(3)...
           Xo(1)*Yo(2)-Xo(2)*Yo(1)];
    Xn  = (1/2)*(3-Xo*Xo')*Xo;
    Yn  = (1/2)*(3-Yo*Yo')*Yo;
    Zn  = (1/2)*(3-Zo*Zo')*Zo;
    R1  = [Xn; Yn; Zn];

    % Accelerometers ------------------------------------------------------
    g_ref    = g*[Xn(3); Yn(3); Zn(3)];
    coriolis = [0; 2*Omega_z(i)*Vx(i); -2*Omega_y(i)*Vx(i)];
    ACCmeter(:,i) = g_ref - coriolis;
end
%R1
% Downsampling ------------------------------------------------------------
Omega_x0 = Omega_x(1:Os:end);
Omega_y0 = Omega_y(1:Os:end);
Omega_z0 = Omega_z(1:Os:end);
Vx_0 = Vx(1:Os:end);

% ###########################################################################
% Simulate DCM Algorithm
% 1) Rotation matrix (Kinematics & Orthogonality condition)
% 2) Drift detection (Yaw: GPS; Roll/Pitch: Accelerometers)
%    Note1: YCG: YawCorrectionGround
%    Note2: YCP: YawCorrectionPlane
%    Note3: TC:  TotalCorrection
%    Note4: RPCP: RollPitchCorrectionPlane
% 3) PI control (Proportional plus integral; without differentiation: D)
% ###########################################################################

for it = 1:it_max
    % Add noise -----------------------------------------------------------
    Omega_x = Omega_x0 + sqrt(NPower)*randn(1, length(Omega_x0));
    Omega_y = Omega_y0 + sqrt(NPower)*randn(1, length(Omega_y0));
    Omega_z = Omega_z0 + sqrt(NPower)*randn(1, length(Omega_z0));

    % Start ---------------------------------------------------------------
    R1e         = R0;
    Omega_P     = zeros(3,1);
    Omega_I     = zeros(3,1);
    Omega_T     = zeros(3,1);
    SINcog_tmp = 0;
    COScog_tmp = 0;
    RPCP = [0; 0; 0];
    for i=1:time/dt
        % Rotation Matrix -------------------------------------------------
        % Kinematics
        % Gyro + ACC (Roll/Pitch) Correction
        dPx = (Omega_x(i) - Omega_T(1))*dt;
        dPy = (Omega_y(i) - Omega_T(2))*dt;
        dPz = (Omega_z(i) - Omega_T(3))*dt;
        Ru  = [1 -dPz dPy; dPz 1 -dPx; -dPy dPx 1];
        R1e = R1e*Ru;
        % Normalization
        X   = R1e(1,:);
        Y   = R1e(2,:);
        err = X*Y';
        Xo  = X - (err/2)*Y;
        Yo  = Y - (err/2)*X;
        Zo  = [Xo(2)*Yo(3)-Xo(3)*Yo(2)...
            Xo(3)*Yo(1)-Xo(1)*Yo(3)...
            Xo(1)*Yo(2)-Xo(2)*Yo(1)];
        Xn  = (1/2)*(3-Xo*Xo')*Xo;
        Yn  = (1/2)*(3-Yo*Yo')*Yo;
        Zn  = (1/2)*(3-Zo*Zo')*Zo;
        R1e = [Xn; Yn; Zn];
        % Drift Detection ---------------------------------------------------
        % Accelerometers
        coriolis = [0; 2*Omega_z(i)*Vx_0(i); -2*Omega_y(i)*Vx_0(i)];
        g_ref = ACCmeter(:,i) + coriolis;
        RPCP = [ Yn(3)*g_ref(3)-Zn(3)*g_ref(2)...
                 Zn(3)*g_ref(1)-Xn(3)*g_ref(3)...
                 Xn(3)*g_ref(2)-Yn(3)*g_ref(1)]';
        % PI Control --------------------------------------------------------
        TC      = Weight_rp*RPCP;
        Omega_P = Kp_RollPitch*TC;
        Omega_I = Omega_I + Ki_RollPitch*dt*TC;
        Omega_T = Omega_P + Omega_I;
    end
    %R1e
    ErrVar = (it-1)/it*ErrVar + 1/it*sum(sum((R1-R1e).^2));
end
ErrVar
```

Matlab Code for Euler Angle Algorithm:

```
% ##########################################################################
% EULER Angle Algorithm
%
% Author:       SHENG ZHANG
% Last Update: 05.06.2011
% ##########################################################################
close all;
clear all;

start_time=cputime;
% ##########################################################################
% Parameter Setup
% ##########################################################################
% Constant -----------------------------------------------------------------
g      = 9.81;                 % 9.81 m/s^2
it_max = 200;                  % iter. number for estimator

% System Parameters --------------------------------------------------------
time      = 10;                % Second
loop_gyro = 200;                % Samples per second (gyro)
loop_GPS  = 5;                 % Samples per second (GPS)
dt        = 1/loop_gyro;       % Integral interval
t_vec     = 0:dt:time;
t_Len     = length(t_vec)-1;
% Elevation and azimuth angle----------------------------------------------
% Satellite Hot bird 6 at longitude 13.06
% Dresden's longitude : 13.95E ; Latitude : 51.54N
```

```
E          = 31.03;              % Elevation angle
A          = 181.13;             % Azimuth   angle
% Measurment Tools ---------------------------------------------------------
specScope = spectrum.welch('Hamming',512);
specOpts  = psdopts(specScope);
set(specOpts, 'Fs',  loop_gyro, 'SpectrumType','twosided','CenterDC',true);

% Input Signal -------------------------------------------------------------
% Gyro
Omega_x = zeros(1,t_Len+1); Omega_x(1:loop_gyro) =  0.05*ones(1,loop_gyro);
Omega_y = zeros(1,t_Len+1); Omega_y(1:loop_gyro) =  -0.6*ones(1,loop_gyro);
Omega_z = (0.01*pi)*ones(1,t_Len+1);

% Omega_x = zeros(1,t_Len+1);
% Omega_y = zeros(1,t_Len+1);
% Omega_z = zeros(1,t_Len+1);
% Omega_x = (2*pi/100)*ones(1,t_Len+1)*sin(-pi/4);
% % Omega_x = (2*pi/100)*ones(1,t_Len+1);
% % Omega_y(1:1) = 0.05*ones(1,1);
% Omega_z = (2*pi/100)*ones(1,t_Len+1)*cos(-pi/4);

% Initial Rotation Mtx & Position
% R0 = [1 0 0; 0 1 0; 0 0 1];      % Initial Rotation Matrix
 R0 = [1/sqrt(2) 0 -1/sqrt(2); 0 1 0; 1/sqrt(2) 0 1/sqrt(2)];
P0 = [0; 0; 0];                    % Initial NED Position

% Velocity
% Vel = repmat(30, 1, t_Len+1);    % m/s; ps: 100km/hr = 27.8m/s
  Vel = 30+5*randn(1,t_Len+1);
% DCM & Noise --------------------------------------------------------------
NPower_gyro = 1e-4;
R0e          = R0;                 % Initial R0 for Estimator
GRVidx       = 1;                  % "0": Perfect knowledge, "1": Measurement
F_DCM        = 2.5e-2;               % Design factor for DCM, 0.03
if (GRVidx == 0)
    F_DCM = 1.0;
end

% ##########################################################################
% Block I (Calculate Correct System States)
% ##########################################################################
% Init ---------------------------------------------------------------------
% Rotation Matrix
R1              = R0;
R1_save         = zeros(3,3,t_Len+1);
R1_save(:,:,1) = R1;

% Heading
cog = zeros(1,t_Len+1); cog(1)  = atan2(R1(1,2), R1(1,1));

% Velocity
tmp         = R1.'*[Vel(1); 0; 0];
V_NED       = zeros(3,t_Len+1);
V_NED(:,1) = [tmp(1); tmp(2); tmp(3)];
V_H         = zeros(1,t_Len+1);
V_H(1)      = sqrt(V_NED(1,1)^2+V_NED(2,1)^2);
```

```
% Position
P_NED      = zeros(3,t_Len+1);
P_NED(:,1) = [P0(1); P0(2); P0(3)];

% Acceleration
A_NED      = zeros(3,t_Len+1);

% Acceleration Meter
ACC_Meter  = zeros(3,t_Len+1);

% Satellite Direction Vector
SDV        = [ cos(E)*cos(A); cos(E)*sin(A); -sin(E)];
SDV1_save  = zeros(3,t_Len+1);
% Start ----------------------------------------------------------------------------
g_orig   = zeros(3,t_Len);
for i = 2:t_Len+1
    % Accelerometer -------------------------------------------------------
    g_ref    = g*[R1(1,3); R1(2,3); R1(3,3)];
    g_orig(:,i-1) = g_ref;
    coriolis = [0; 2*Omega_z(i-1)*Vel(i-1); -2*Omega_y(i-1)*Vel(i-1)];
    ACC_Meter(:,i-1) = [Vel(i)-Vel(i-1); 0; 0]/(dt) - g_ref + coriolis;
    Ext_Acc(:,i-1)   = [Vel(i)-Vel(i-1); 0; 0]/(dt);

    % DCM -----------------------------------------------------------------
    % Kinematics
    dPx = Omega_x(i-1)*dt;
    dPy = Omega_y(i-1)*dt;
    dPz = Omega_z(i-1)*dt;
    Ru  = [1 -dPz dPy; dPz 1 -dPx; -dPy dPx 1];
    R1  = Ru.'*R1;

    % Normalization
    X   = R1(1,:);
    Y   = R1(2,:);
    err = X*Y';
    Xo  = X - (err/2)*Y;
    Yo  = Y - (err/2)*X;
    Zo  = [Xo(2)*Yo(3)-Xo(3)*Yo(2)...
        Xo(3)*Yo(1)-Xo(1)*Yo(3)...
        Xo(1)*Yo(2)-Xo(2)*Yo(1)];
    Xn  = (1/2)*(3-Xo*Xo')*Xo;
    Yn  = (1/2)*(3-Yo*Yo')*Yo;
    Zn  = (1/2)*(3-Zo*Zo')*Zo;
    R1  = [Xn; Yn; Zn];
    R1_save(:,:,i) = R1;
    SDV1           = R1*SDV;
    SDV1_save(:,i) = SDV1;


    % Course over ground (COG: GPS)
    cog(i) = atan2(R1(1,2), R1(1,1));

    % Vel., Pos., Acc. and Acceleromater ----------------------------------
    % Velocity
    tmp     = R1.'*[Vel(i); 0; 0];
    V_NED(:,i) = [tmp(1); tmp(2); tmp(3)];
    V_H(i) = sqrt(V_NED(1,i)^2+V_NED(2,i)^2); % GPS measured velocity

    % Position
    P_NED(:,i) = P_NED(:,i-1) + ...
        [V_NED(1,i-1); V_NED(2,i-1); V_NED(3,i-1)]*dt;

    % Acceleration
    A_NED(:,i-1) = [V_NED(1,i)-V_NED(1,i-1); V_NED(2,i)-V_NED(2,i-1)...;
                    V_NED(3,i)-V_NED(3,i-1)]/dt;

end

% Assignament -------------------------------------------------------------
Omega_x0    = Omega_x(1:end-1);
Omega_y0    = Omega_y(1:end-1);
Omega_z0    = Omega_z(1:end-1);
cog0        = cog(1:end-1);
ACC_Meter0 = ACC_Meter(:,1:end-1);
Ext_Acc0    = Ext_Acc(:,1:end-1);
V_H0        = V_H(1:end-1);
Vel0        = Vel(1:end-1);

V_NED(:,end) = []; P_NED(:,end) = []; A_NED(:,end) = [];
R1_save(:,:,end) = [];


% %
########################################################################
% % Block II (Estimate System States)
% %
########################################################################
ErrVar    = zeros(3,3,t_Len);
for it = 1:it_max
    % Add noise ------------------------------------------------------------
    Omega_x = Omega_x0 + sqrt(NPower_gyro)*randn(1, length(Omega_x0));
    Omega_y = Omega_y0 + sqrt(NPower_gyro)*randn(1, length(Omega_y0));
    Omega_z = Omega_z0 + sqrt(NPower_gyro)*randn(1, length(Omega_z0));

    % Start ---------------------------------------------------------------
    R1e_save   = zeros(3,3,t_Len);
    SDV1e_save = zeros(3,t_Len);
    V1_save    = zeros(1,t_Len);
    sita       = zeros(1,t_Len);
    D_R        = zeros(3,3,t_Len);
    D_SDV      = zeros(3,t_Len);
    R1e        = R0e;                    % Initial value of R1e
    R1e_save(:,:,1) = R1e;
    testgm     = zeros(3,t_Len);
    cnt        = 0;
    for i = 2:t_Len
        % DCM ------------------------------------------------------------
        % Kinematics
        dPx = Omega_x(i-1)*dt;
        dPy = Omega_y(i-1)*dt;
        dPz = Omega_z(i-1)*dt;
        Ru  = [1 -dPz dPy; dPz 1 -dPx; -dPy dPx 1];
        R1e = Ru.'*R1e;

        % Normalization
        X   = R1e(1,:);
        Y   = R1e(2,:);
        err = X*Y';
```

```
Xo   = X - (err/2)*Y;
Yo   = Y - (err/2)*X;
Zo   = [Xo(2)*Yo(3)-Xo(3)*Yo(2)...
    Xo(3)*Yo(1)-Xo(1)*Yo(3)...
    Xo(1)*Yo(2)-Xo(2)*Yo(1)];
Xn   = (1/2)*(3-Xo*Xo')*Xo;
Yn   = (1/2)*(3-Yo*Yo')*Yo;
Zn   = (1/2)*(3-Zo*Zo')*Zo;
R1e = [Xn; Yn; Zn];


% Drift Correction ------------------------------------------------
if (rem(i,loop_gyro/loop_GPS) == 1)|| (loop_gyro/loop_GPS == 1)
    C1 = R1e;
    % Coriolis Correction -----------------------------------------
    if (GRVidx)
        vel_mes = V_H0(i-1)/(sqrt(C1(1,1)^2 + C1(1,2)^2)/1.0); %Vel
  in x-axis
        cor_mes = [0; 2*Omega_z(i-1)*vel_mes; -2*Omega_y(i-
        1)*vel_mes];
        g_mes   = -ACC_Meter0(:,i-1)+Ext_Acc0(:,i-1)+cor_mes;
        testgm(:,i-1) = g_mes;
    else
        g_mes = g_orig(:,i);
    end

    % Measurement -------------------------------------------------
    S_Ym = sin(cog0(i));
    C_Ym = cos(cog0(i));
    S_Pm = -g_mes(1)/sqrt(g_mes.'*g_mes);
    C_Pm = sqrt(1-S_Pm^2);
    S_Rm = g_mes(2)/sqrt(g_mes(2)^2+g_mes(3)^2);
    C_Rm = sqrt(1-S_Rm^2);

    % Detect Euler Angle (Yaw) ------------------------------------
    x_c1 = [C1(1,1) C1(1,2) 0];
    x_m  = [C_Ym    S_Ym    0];
    tmp  = cross(x_c1, x_m);
    S_Yd = F_DCM*sum(tmp)/(norm(x_c1)*norm(x_m));
    C_Yd = sqrt(1-S_Yd^2);
    Y_m  = [C_Ym -S_Ym 0; ...
            S_Ym  C_Ym 0; ...
            0     0    1];
    Y_d  = [C_Yd -S_Yd 0; ...
            S_Yd  C_Yd 0; ...
            0     0    1];

    % Detect Euler Angle (Pitch) ----------------------------------
    C2   = C1*Y_m;
    x_c2 = [C2(1,1) 0 C2(1,3)];
    x_m  = [C_Pm    0   -S_Pm];
    tmp  = cross(x_c2, x_m);
    S_Pd = F_DCM*sum(tmp)/(norm(x_c2)*norm(x_m));
    C_Pd = sqrt(1-S_Pd^2);
    P_m  = [ C_Pm 0 S_Pm; ...
             0    1    0; ...
            -S_Pm 0 C_Pm];
            P_d  = [ C_Pd 0 S_Pd; ...
                     0    1    0; ...
                    -S_Pd 0 C_Pd];

            % Detect Euler Angle (Roll) -----------------------------------
            C3 = C2*P_m;
            y_c3 = [0 C3(2,2) C3(2,3)];
            y_m  = [0   C_Rm    S_Rm ];
            tmp  = cross(y_c3, y_m);
            S_Rd = F_DCM*sum(tmp)/(norm(y_c3)*norm(y_m));
            C_Rd = sqrt(1-S_Rd^2);
            R_d  = [1    0     0; ...
                    0 C_Rd -S_Rd; ...
                    0 S_Rd  C_Rd];

            % Drfit Compensation
            R1e = C3*(R_d).'*(P_d*P_m).'*(Y_d*Y_m).';
        end

        % Save ------------------------------------------------------------
        R1e_save(:,:,i) = R1e;

        SDV1e           = R1e_save(:,:,i)*SDV;
        SDV1e_save(:,i) = SDV1e;
    end

    % Statistics I --------------------------------------------------------
    for i = 1:t_Len
        ErrVar(:,:,i) = (it-1)/it*ErrVar(:,:,i) + ...
            1/it*(R1e_save(:,:,i)-R1_save(:,:,i))'*(R1e_save(:,:,i)-
R1_save(:,:,i));
         D_R(:,:,i)      = R1e_save(:,:,i)-R1_save(:,:,i);
         D_SDV(:,i)    = SDV1e_save(:,i)-SDV1_save(:,i);
         V1            = cross(SDV1_save(:,i),SDV1e_save(:,i));% All
rotation are between 0 and 180
         V1_abs        = sqrt(V1(1).^2+V1(2).^2+V1(3).^2);
         V1_save(:,i) = V1_abs;
         sita(:,i)     =
asin(V1_save(:,i)/(norm(SDV1_save(:,i))*norm(SDV1e_save(:,i))));
         sita(:,i)     = sita(:,i)*180/pi; % rad---->angle
         Ant           = -2.947*sita(:,i)^2+10.0131;
         Signal        = Ant-174;
         Ant_save(:,i)= Ant;
         Signal_save(:,i) = Signal;
    end
end

sita;

% Statistics II -----------------------------------------------------------
AvgErrVar=squeeze((ErrVar(1,1,:)+ErrVar(2,2,:)+ErrVar(3,3,:))/3);

% ########################################################################
% Analysis
% ########################################################################
% Plot --------------------------------------------------------------------
if (1)
    figure(099);
    plot(1:length(AvgErrVar),AvgErrVar, '-xk');
    grid on;
end
if (1)
    figure(101);
    plot3(P_NED(1,:), P_NED(2,:), P_NED(3,:), '-xb');
    %plot3(V_N, V_E, V_D, '-xb');
    %xlim([-4 4]); ylim([-4 2]); zlim([0 5]);
    xlabel('N'); ylabel('E'); zlabel('D');
    grid on; axis fill; % fill, square, tight...etc
end
if (0)
    psdProbe    = psd(specScope, Omega_y, specOpts);
    figure(102);
    hold on;
    hLine = plot(psdProbe);
    set(hLine,'Color','b');
    hold off;
end
if (0)
    figure(103);
    plot(1:length(Omega_y),Omega_y, '-xk');
    grid on;
end
if (0)
    figure(104);
    plot(squeeze(sita),squeeze(Ant_save),'xr');
     grid on;
     xlabel('Deviation angle(deg)');
     ylabel('SNR (dB)');
end
if(0)
    figure(105)
    a = sita;
%     b = mapminmax(a);
%     x = -1:0.02:1;
    [n,xout]=hist(a, [-1:0.01:2]);
    p       = n/length(sita);
    bar(xout,p)
    grid on;
    xlabel('Deviation angle(deg)');
    ylabel('Probability');
end
if(0)
    figure(106)
    s = squeeze(Signal_save);
    x = 1:length(sita);
    plot(x,s);
    grid on;
    xlabel('Sample number');
    ylabel('Recevie Signal');
end
if(0)
    figure(107)
    [n1,xout1] = hist(squeeze(D_R(1,1,:)));
    x1         = 1:length(D_R);
    bar(xout1,n1/length(D_R));
    grid on;
    xlabel('Element (1,1) difference between two rotation matrix');
    ylabel('Probability');
end
```

```
if(0)
    figure(116)
    [n9,xout9] = hist(squeeze(D_SDV(1,:)));
    x9          = 1:length(D_SDV);
    bar(xout9,n9/length(D_SDV));
    grid on;
    xlabel('Element (1) difference between two satellite direction
vectors');
    ylabel('Probability');
end
if(0)
    figure(117)
    [n10,xout10] = hist(squeeze(D_SDV(2,:)));
    x10           = 1:length(D_SDV);
    bar(xout10,n10/length(D_SDV));
    grid on;
    xlabel('Element (") difference between two satellite direction
vectors');
    ylabel('Probability');
end
if(0)
    figure(118)
    [n11,xout11] = hist(squeeze(D_SDV(3,:)));
    x11          = 1:length(D_SDV);
    bar(xout11,n11/length(D_SDV));
    grid on;
    xlabel('Element (3) difference between two satellite direction
vectors');
    ylabel('Probability');
end
stop_time=cputime;
disp(sprintf('time of simulation = %6.1f s', stop_time-start_time));
```

Appendix B

Euler Angle Method Implementation by using Arduino Code:

1 : Main Program

```
// ------------------------------------------------------------
// Filename:    Test07_integrated_lib.pde
// refers to:   -
// Autor:       Ting-Jung Liang, Tom Ritschel Sheng Zhang
// Description: Main Program
// Revision:    0.2
// Last Update: 11.07.2011
//
// ------------------------------------------------------------

// Main Program Init
// ------------------------------------------------------------
// Sensor Libraries
#include "NMEA_MTK16.h"
#include "AP_ADC.h"
#include "HMC5843_Compass.h"
#include <FastSerial.h>

// Other Libs
#include <Wire.h>
#include <AP_Math.h>
#include <math.h>
#include "defines.h"

// Macros & Constructors
FastSerialPort0(Serial);
FastSerialPort1(Serial1);

NMEA_MTK16          gps(&Serial1);       // See "NMEA_MTK16.h"
HMC5843_Compass_Class compass;           // See "HMC5843_Compass.h"
AP_ADC_ADS7844      adc;                 // See "AP_ADC.h"
```

```
// General Variables
// -------------------------------------------------------------
// System Timer
unsigned long loop_timer_1;               // Timer 1 in each loop (ms)
unsigned long loop_timer_2;               // Timer 2 in each loop (ms)
int           loop_state = 0;             // Control between Slow_Loop & Fast_Loop

// GPS Variables
// -------------------------------------------------------------
float gps_longitude   = 0;                // Longitude (°)
float gps_latitude    = 0;                // Latitude (°)
float gps_time        = 0;                // Format "hhmmssttt"
float gps_sog         = 0;                // Speed over Ground (m/s)
float gps_cog         = 0;                // Course over Ground (°)
float gps_declination = 0;                // Magnetic variation (°)
float gps_vdop        = 0;                // Vertical Dilution of Precision
float gps_hdop        = 0;                // Horizontal Dilution of Precision
float gps_altitude    = 0;                // MSL Altitude (m) - not corrected by Geoid
float gps_geoid       = 0;                // Geoid Separation (m)
int   gps_status      = 0;                // Indicator GPS Valid Data
float gps_C_N0        = 0;                // C/N0 average

// IMU Variables
// -------------------------------------------------------------
//Sensor: GYROX, GYROY, GYROZ, ACCELX, ACCELY, ACCELZ
int   sensors[6]     = {1,2,0,4,5,6};          // For ArduPilot Mega Sensor Shield Hardware
int   sensor_sign[6] = {1,-1,-1,-1,1,1};       // Correcting the Output Direction Values
float imu_vec[6];                              // Array that store the 6 ADC channels used by IMU
float imu_offset[6]  = {0,0,0,                 // Factory offset values of the gyros and accelerometers
                        0,0,0};
float filt[3]        = {0.0,0.0,0.0};          // Register for acc filter

// Temperature Sensor
float temperature;                             // Sensor Board temperature

// Magneto Variables
// -----------------------------------------------------------------
float x_in_N, x_in_E, x_in_D;                  // x (roll) in NED coordinate
float y_in_N, y_in_E, y_in_D;                  // y (pitch) in NED coordinate
float z_in_N, z_in_E, z_in_D;                  // z (yaw) in NED coordinate

// DCM Variables
// -----------------------------------------------------------------
float    dt = 0.02;                            // Integration time (sec) for the gyros (DCM algorithm)
float    gyro_vec[3];                  // Store the gyros turn rate in a vector
float    acc_vec[3];                   // Store the acceleration in a vector
Vector3f g_mes;                        // Store the measured gravity in body axis
Matrix3f dcm_mtx = Matrix3f(1,0,0,
                            0,1,0,
                            0,0,1);          // The rotation matrix see "matrix3.h"
Matrix3f r1m_mtx = Matrix3f(1,0,0,
                            0,1,0,
                            0,0,1);          // The measured rotation matrix from mag & acc

// Statistics
Matrix3f r0_mtx;                       // First Rotation Matrix
float    err_var = 0.0;                // Error Variance

// Filter Variables
// -----------------------------------------------------------------
float     xv[3][3];                    // temporary data memory for recursive acc-filter
float     yv[3][3];

// Setup
// -----------------------------------------------------------------
void setup()
{
  Serial.begin(115200);                        // Initialize USB/FTDI port
  Sensors_Init();                              // Initialize all sensors
  R0_Init();                                   // Initialize the rotation matrix
  Output_Setup();                              // Output Program Message (Setup)
  delay(100);
  loop_timer_1 = millis();                     // Initialize timer 1
  loop_timer_2 = millis();                              // Initialize timer 2
}

// Loop Start
// ----------------------------------------------------------------
void loop()
{
    if ((millis() - loop_timer_1) > ADC_S_PERIOD - 1)    // Update period 20 ms (50 Hz)
    {
      //Serial.println("Loop Start");
      //Serial.println(millis() - loop_timer_1);
      loop_timer_1 = millis();                        // Update timer 1
      Fast_Loop();                                 // Loop activated every  20 ms (50 Hz)
      //Serial.println(millis() - loop_timer_1);
      Slow_Loop();                                 // Loop activated every 200 ms ( 5 Hz)
      //Serial.println(millis() - loop_timer_1);
    }
    gps.update();                                  // Transmit GPS info in "non-busy" time
}

// Routines in Loop
// ----------------------------------------------------------------
void Fast_Loop(void)
{
  Sensors_Read50Hz();                               // Read sensor values in 50 Hz
  // DCM Here
  Matrix_Update();                                                  // Integrate the DCM matrix
  Normalize();                                                      // Normalize the DCM matrix
  // Output Result
  Output_Fast();                                 // Output Program Message (Fast Loop)
}

void Slow_Loop(void)
{
  loop_state++;                                  // Guarantee GPS read in 5 Hz
  switch(loop_state)
  {
    case 1:
      break;



    case 10:
        Sensors_Read5Hz();                        // Read sensor values in 5 Hz
        Drift_Correction();                                   // Perform drift correction
        Output_Slow();                        // Output Program Message (Slow Loop)
        loop_state = 0;
        break;

    default:
        break;
  }
}
```

2 : ADC Converter:

```
// ------------------------------------------------------------
// Filename:    ADC.pde
// refers to:   Test07_integrated_lib.pde
// Autor:       Ting-Jung Liang, Tom Ritschel Sheng Zhang
// Description: Reading Gyro and Acc Sensor Signals Only
// Revision:    0.2
// Last Update: 12.06.2011
//
// ------------------------------------------------------------

// Read ADC Raw Data
// ------------------------------------------------------------
// Sensor: Temperature, GYROX, GYROY, GYROZ, ACCELX, ACCELY and ACCELZ
void Adc_Read_Raw(void)
{
  temperature = adc.Ch(TEMP_CH);
  for (int i = 0; i < 6; i++)
  {
    imu_vec[i] = adc.Ch(sensors[i]);
    if ( i < 3)
```

```
      { imu_vec[i] -= GYRO_BIAS; }
    else
      { imu_vec[i] -= ACCEL_BIAS; }
  };
}

// Estimate Offset for GYROX, GYROY, GYROZ
// ------------------------------------------------------------
void Adc_Estimate_Offset(void)
{
  float f1, f2;
  // first readings
  Adc_Read_Raw();
  for ( int i = 0; i < 3; i++)
    { imu_offset[i] = imu_vec[i]; }
  delay(10);

  // cycles (gyro)
  for( int y = 0; y < OFFSET_CYCLES; y++)
  {
    Adc_Read_Raw();
    for ( int i = 0; i < 3; i++)
      { imu_offset[i] = imu_offset[i] * 0.9 + imu_vec[i] * 0.1; }
    delay(10);
  }

  // Acc (not yet!)
}

// Read and Calibrate Data from Given ADC Channel
// ------------------------------------------------------------
float Adc_Read_Ch(int channel)
{
  float value;

  switch (channel) {
    case 0:  // Rad/s
```

```
      value = Gyro_Scaled_X(imu_vec[channel]-imu_offset[channel])* sensor_sign[channel];
      break;
    case 1:  // Rad/s
      value = Gyro_Scaled_Y(imu_vec[channel]-imu_offset[channel])* sensor_sign[channel];
      break;
    case 2:  // Rad/s
      value = Gyro_Scaled_Z(imu_vec[channel]-imu_offset[channel])* sensor_sign[channel];
      break;
    default:  // m/s^2
      value = Grav_Scaled(imu_vec[channel] - imu_offset[channel])* sensor_sign[channel];
  }

  return value;
}

// Gyro Data Processing Before DCM uses ADC Input
// -------------------------------------------------------------
void Adc_GyroData_Processing(void)
{
  // Gyro Part
  gyro_vec[0] = Adc_Read_Ch(0);               // gyro x roll
  gyro_vec[1] = Adc_Read_Ch(1);               // gyro y pitch
  gyro_vec[2] = Adc_Read_Ch(2);               // gyro z yaw
}

// Acc Data Processing Before DCM uses ADC Input
// -------------------------------------------------------------
void Adc_AccData_Processing(void)
{
  // Acc Part with Low Pass Filter to Cancel Vibration
  // Assumption: A non-accelerating system!!

  // 3rd order Low Pass Filter
  /*
  for ( int i = 0; i < 3; i++)
  {
      // shift every x-element
```

```
        xv[i][3] = xv[i][2]; xv[i][2] = xv[i][1]; xv[i][1] = xv[i][0];
        // input of current element
                xv[i][0] = (float)Adc_Read_Ch(i+3);
                //  shift every y-element
        yv[i][3] = yv[i][2]; yv[i][2] = yv[i][1]; yv[i][1] = yv[i][0];
                // Difference equation
        yv[i][0] =   0.0985 * (xv[i][0] + xv[i][3]) + 0.2956 * (xv[i][1] + xv[i][2])
               + (0.5772 * yv[i][1])+ (-0.4218 * yv[i][1])
               + (0.0563 * yv[i][3]);
                // handover of the current element
        filt[i] = 1.565892 * yv[i][0];
  }
  */

  filt[0]    = filt[0] * 0.6 + (float)Adc_Read_Ch(3) * 0.4; // acc x roll
  filt[1]    = filt[1] * 0.6 + (float)Adc_Read_Ch(4) * 0.4; // acc y pitch
  filt[2]    = filt[2] * 0.6 + (float)Adc_Read_Ch(5) * 0.4; // acc z yaw

  acc_vec[0] = filt[0];
  acc_vec[1] = filt[1];
  acc_vec[2] = filt[2];
}
```

3. EA Algorithm:

```
// ------------------------------------------------------------
// Filename:    EA.pde
// refers to:   Test07_integrated_lib.pde
// Autor:       Ting-Jung Liang, Tom Ritschel sheng Zhang
// Description: Functions required for Euler Angle Algorithm
// Revision:    0.2
// Last Update: 12.06.2011
//
// ------------------------------------------------------------

// Initialize R0 Matrix
// ------------------------------------------------------------
```

```
void R0_Init(void)
{
  float tmp;

  #if GPS_STATE == ENABLED
    while (gps_status == 0)                         // Wait until gps locked
    {
      gps.update();
      GPS_Read();
      delay(200);
    }
    tmp = gps_declination;
  #else
    tmp = 0.0;
  #endif

  for (int i = 0; i < FILT_CYCLES; i++)
  {
    Adc_Read_Raw();                               // Read Gyro and Acc Data
    Adc_AccData_Processing();                     // DSP of Gyro Data
    delay(20);
  }
  Compass_Read(tmp);                              // Read compass data
  dcm_mtx = Matrix3f(x_in_N, x_in_E, x_in_D,      // Init rotation matrix
                     y_in_N, y_in_E, y_in_D,
                     z_in_N, z_in_E, z_in_D);
  r0_mtx  = dcm_mtx;                              // Save for Statistics Purpose

}

// Update the Rotation Matrix
// ---------------------------------------------------------------
void Matrix_Update(void)
{
  Matrix3f update_mtx;

  update_mtx  = Matrix3f(1, -dt*gyro_vec[2], dt*gyro_vec[1],   // Ru
                        dt*gyro_vec[2], 1, -dt*gyro_vec[0],
                        -dt*gyro_vec[1], dt*gyro_vec[0], 1);

  dcm_mtx     = update_mtx.transposed()*dcm_mtx;

}

// Normalize the Rotation Matrix
// ------------------------------------------------------------
void Normalize(void)
{
  float    error, renorm;
  Vector3f x_orth, y_orth, z_orth;

  error  = (dcm_mtx.a * dcm_mtx.b) * 0.5;
  x_orth = dcm_mtx.a - (dcm_mtx.b * error);                // eq. 19
  y_orth = dcm_mtx.b - (dcm_mtx.a * error);                // eq. 19
  z_orth = x_orth % y_orth;                         // eq. 20

  renorm    = 0.5*(3 - x_orth*x_orth);
  dcm_mtx.a = x_orth*renorm;                          // eq. 21
  renorm    = 0.5*(3 - y_orth*y_orth);
  dcm_mtx.b = y_orth*renorm;                          // eq. 21
  renorm    = 0.5*(3 - z_orth*z_orth);
  dcm_mtx.c = z_orth*renorm;                          // eq. 21

}

// Measure Roll, Pitch, Yaw Errors and Correct Them
// ------------------------------------------------------------
void Drift_Correction(void)
{
  // Variable Declaration
  float    S_Ym, C_Ym;         // Yaw   Measurement
  float    S_Pm, C_Pm;         // Pitch Measurement
  float    S_Rm, C_Rm;         // Roll  Measurement
  float    tmp;
```

```
Vector3f x_c1, x_m;          // Yaw Correction
Vector3f cr_tmp;
float    S_Yd, C_Yd;
Matrix3f Y_m, Y_d;
Matrix3f C2;                 // Pitch Correction
Vector3f x_c2;
float    S_Pd, C_Pd;
Matrix3f P_m, P_d;
Matrix3f C3;                 // Roll Correction
Vector3f y_c3, y_m;
float    S_Rd, C_Rd;
Matrix3f R_d;
Matrix3f R_t, P_t, Y_t;      // Drift Compensation



// Calculate g_mes
Remove_Coriolis();                          // Remove centrifugal acceleration.

// Calculate r1m
r1m_mtx = Matrix3f(x_in_N, x_in_E, x_in_D,      // Init rotation matrix
                   y_in_N, y_in_E, y_in_D,
                   z_in_N, z_in_E, z_in_D);



// Calculate Sin, Cos of Euler Angle (Measurement)
#if GPS_STATE == ENABLED
  S_Ym = sin(ToRad(gps_cog));
  C_Ym = cos(ToRad(gps_cog));
  S_Pm = -g_mes.x/g_mes.length();
  C_Pm = sqrt(1-S_Pm*S_Pm);
  S_Rm = g_mes.y/sqrt(g_mes.y*g_mes.y+g_mes.z*g_mes.z);
  C_Rm = sqrt(1-S_Rm*S_Rm);
#else
  tmp = (-r1m_mtx.a.y*r1m_mtx.c.z-r1m_mtx.a.x*r1m_mtx.a.z*r1m_mtx.b.z)/r1m_mtx.b.x;
  if (tmp >= 1.0) {
    C_Pm = 1.0;
  }else if (tmp <= 0.0) {
    C_Pm = 0.0;
  }else {
    C_Pm = sqrt(tmp);
  }
  S_Ym = r1m_mtx.a.y/C_Pm;
  C_Ym = r1m_mtx.a.x/C_Pm;
  S_Pm = -r1m_mtx.a.z;
  S_Rm = r1m_mtx.b.z/C_Pm;
  C_Rm = r1m_mtx.c.z/C_Pm;
#endif

// Detect Euler Angle (Yaw)
x_c1   = Vector3f(dcm_mtx.a.x, dcm_mtx.a.y, 0.0);
x_m    = Vector3f(C_Ym,        S_Ym,        0.0);
cr_tmp = x_c1 % x_m;
S_Yd   = F_DCM*cr_tmp.z/(x_c1.length()*x_m.length());
C_Yd   = sqrt(1-S_Yd*S_Yd);
Y_m    = Matrix3f(C_Ym, -S_Ym, 0,
                  S_Ym,  C_Ym, 0,
                  0,     0,    1);
Y_d    = Matrix3f(C_Yd, -S_Yd, 0,
                  S_Yd,  C_Yd, 0,
                  0,     0,    1);

// Detect Euler Angle (Pitch)
C2     = dcm_mtx * Y_m;
x_c2   = Vector3f(C2.a.x, 0.0,  C2.a.z);
x_m    = Vector3f(C_Pm,   0.0, -S_Pm);
cr_tmp = x_c2 % x_m;
S_Pd   = F_DCM*cr_tmp.y/(x_c2.length()*x_m.length());
C_Pd   = sqrt(1-S_Pd*S_Pd);
P_m    = Matrix3f( C_Pm, 0, S_Pm,
                   0,    1,    0,
                  -S_Pm, 0, C_Pm);
P_d    = Matrix3f( C_Pd, 0, S_Pd,
                   0,    1,    0,
                  -S_Pd, 0, C_Pd);
  // Detect Euler Angle (Roll)
  C3     = C2 * P_m;
  y_c3   = Vector3f(0.0, C3.b.y, C3.b.z);
  y_m    = Vector3f(0.0, C_Rm,   S_Rm);
  cr_tmp = y_c3 % y_m;
  S_Rd   = F_DCM*cr_tmp.x/(y_c3.length()*y_m.length());
  C_Rd   = sqrt(1-S_Rd*S_Rd);
  R_d    = Matrix3f(1,    0,     0,
                    0, C_Rd, -S_Rd,
                    0, S_Rd,  C_Rd);

  // Drift Compensation
  R_t = C3*R_d.transposed();
  P_t = P_m.transposed()*P_d.transposed();
  Y_t = Y_m.transposed()*Y_d.transposed();
  dcm_mtx = R_t*P_t*Y_t;

}

// Remove Coriolis Effect (Assume a Non-accelerating System)
// -------------------------------------------------------------
void Remove_Coriolis(void)
{
  Vector3f v_x, omega_m, acc_m;
  float cos_2_h;

  // Velocity over ground to velocity in x-axis
  #if GPS_STATE == ENABLED
    cos_2_h = sqrt(dcm_mtx.a.x*dcm_mtx.a.x + dcm_mtx.a.y*dcm_mtx.a.y)/1.0;
    v_x = Vector3f(gps_sog/cos_2_h, 0.0, 0.0);
  #else
    v_x = Vector3f(0.0, 0.0, 0.0);
  #endif

  // Coriolis Effect
  omega_m = Vector3f(gyro_vec[0], gyro_vec[1], gyro_vec[2]);
  acc_m   = Vector3f(acc_vec[0],  acc_vec[1],  acc_vec[2]);
  g_mes   = -acc_m + (omega_m%v_x);   // A_ext is not removed
}
```

4. Functions required for Reading All Sensor Signals

```
// -------------------------------------------------------------
// Filename:    Sensors.pde
// refers to:   Test07_integrated_lib.pde
// Autor:       Ting-Jung Liang, Tom Ritschel  Sheng Zhang
// Description: Functions required for Reading All Sensor Signals
// Revision:    0.1
// Last Update: 01.06.2011
//
// -------------------------------------------------------------

// Sensor Initialization
// -------------------------------------------------------------
boolean Sensors_Init(void)
{
  // Gyro and Acc
  adc.Init();
  Adc_Estimate_Offset();                    // Estimate sensor offsets (no cali.)

  // Compass (Definition see "HMC5843_Compass_Orientation.h")
  compass.Init();
  compass.SetOrientation(ROTATION_ROLL_180);       // See "HMC5843_Compass_orientation.h"

  // GPS
  Serial1.begin(115200);
  gps.init(5);                         // 5 Hz
  gps.update();                          // Update gps data

}



// Read Sensor Data in 50 Hz (Gyro)
// -------------------------------------------------------------
```

```
void Sensors_Read50Hz(void)
{
  Adc_Read_Raw();                              // Read Gyro and Acc Data
  Adc_GyroData_Processing();                   // DSP of Gyro Data
}

// Read Sensor Data in 5 Hz (GPS, Magneto & Acc)
// ------------------------------------------------------------
void Sensors_Read5Hz(void)
{
  float tmp;

  GPS_Read();                                  // Read GPS data
  //Adc_Read_Raw();                            // Read Gyro and Acc Data
  Adc_AccData_Processing();                    // DSP of Acc Data
  #if GPS_STATE == ENABLED
    tmp = gps_declination;
  #else
    tmp = 0.0;
  #endif
  Compass_Read(tmp);                           // Read compass data
}


// Calculate Magneto Rotation Matrix
// ------------------------------------------------------------
void Compass_Read(float dec_angle)
{
  Vector3f N_mag;
  Vector3f D;
  Vector3f E_m;
  Vector3f N_m;
  Matrix3f R_m;
  Matrix3f dec_mtx;
  Matrix3f temp;
```

```
 // Read Compass and Acc Data
 compass.Read();
 compass.Calculate();                    // Orientation and normalization
 N_mag = compass.mXYZ;                        // Measured magnetic field
 //Serial.println("N_mag D E_m N_m");
 //print_vec(N_mag);
 D = Vector3f(acc_vec[0], acc_vec[1], acc_vec[2]);     // Accelerometer Data
 D.normalize();
 //print_vec(D);

 // Calculate Em & Nm
 E_m = D % N_mag;                    // East_mag
 E_m.normalize();                  // Angle of N_mag and D is not 90 degree
 //print_vec(E_m);
 N_m = E_m % D;                      // North_mag
 //print_vec(N_m);
 R_m = Matrix3f(N_m.x, E_m.x, D.x,
           N_m.y, E_m.y, D.y,
           N_m.z, E_m.z, D.z);
 //Serial.println("R_m");
 //print_mtx(R_m);

 // Correct Declination Angle
 dec_mtx = Matrix3f(cos(ToRad(dec_angle)) ,sin(ToRad(dec_angle)),0,
            -sin(ToRad(dec_angle)),cos(ToRad(dec_angle)),0,
            0           ,0           ,1);
 temp = R_m * dec_mtx;

 // Assign to global variables
 x_in_N = temp.a.x; x_in_E = temp.a.y; x_in_D = temp.a.z;
 y_in_N = temp.b.x; y_in_E = temp.b.y; y_in_D = temp.b.z;
 z_in_N = temp.c.x; z_in_E = temp.c.y; z_in_D = temp.c.z;

}
```

```
// Read GPS Data in 5 Hz (Default)
// ----------------------------------------------------------------
void GPS_Read(void)
{
     if(gps.new_data)
     {
       gps_status      = gps.status;

       if(gps_status == 1)
       {
           // Definitions of variables below refer to internal "GPS Manual"
           gps_time        = gps.time*1.0;
           gps_longitude   = gps.longitude*0.000001;
           gps_latitude    = gps.latitude*0.000001;
           gps_sog         = gps.ground_speed*0.01*0.51444;
           gps_cog         = gps.ground_course*0.01;
           gps_vdop        = gps.VDOP*0.01;
           gps_hdop        = gps.HDOP*0.01;
           gps_altitude    = gps.MSL_alt*0.1;
           gps_geoid       = gps.geoid*0.1;
           gps_C_N0        = gps.C_N0;
           gps_declination = gps.mag_var*0.01;

       }

       gps.new_data  = 0;

     }
}

// Output Message (Setup)
// ----------------------------------------------------------------
void Output_Setup(void)
{
  /*
  // GPS
  Serial.println("GPS (Setup)");
  Serial.println(gps_declination);
  Serial.println(gps_cog);
  Serial.println(gps_sog);
  Serial.println(gps_longitude);
  Serial.println(gps_latitude);*/
  //print_mtx(r0_mtx);
}

// Output Message (Fast Loop)
// -------------------------------------------------------------
void Output_Fast(void)
{
  Matrix3f err_mtx, tmp_mtx;

  /*
  // Test (Not Moving)
  err_mtx = dcm_mtx - r0_mtx;
  Serial.println("ErrVar (Fast Loop): ");
  //print_mtx(err_mtx*err_mtx.transposed());
  tmp_mtx = err_mtx*err_mtx.transposed();
  err_var = 0.95*err_var + 0.05*(tmp_mtx.a.x+tmp_mtx.b.y+tmp_mtx.c.z)/3;
  Serial.println(err_var,9);
  */
  /*
  Serial.print("Gyro x, y, z; ACC x, y, z: ");
  Serial.print(gyro_vec[0]);
  Serial.print("  ");
  Serial.print(gyro_vec[1]);
  Serial.print("  ");
  Serial.print(gyro_vec[2]);
  Serial.print("  ");
  Serial.print(acc_vec[0]);
  Serial.print("  ");
  Serial.print(acc_vec[1]);
  Serial.print("  ");
  Serial.print(acc_vec[2]);
  Serial.println("  ");
  */
  /*
  // Print DCM Matrix
  Serial.println("DCM Matrix (Fast Loop): ");
  print_mtx(dcm_mtx);*/
}

// Output Message (Slow Loop)
// -------------------------------------------------------------
void Output_Slow(void)
{
  float zeta_az, zeta_elev;
  /*
  // Print R1m Matrix
  Serial.println("R1m Matrix (Slow Loop): ");
  print_mtx(r1m_mtx);*/

  /*
  // Print DCM Matrix
  Serial.println("DCM Matrix (Slow Loop): ");
  print_mtx(dcm_mtx);*/

  /*
  // GPS
  Serial.println("GPS (Slow Loop)");
  Serial.print("  ");
  Serial.print(gps_declination);
  Serial.print("  ");
  Serial.print(gps_cog);
  Serial.print("  ");
  Serial.print(gps_sog);
  Serial.print("  ");
  Serial.print(gps_longitude);
  Serial.print("  ");
  Serial.println(gps_latitude);
  */
  // Acc
  // Serial.println("Acc (Slow Loop)");
  Serial.print(acc_vec[0]);
  Serial.print(" ");
  Serial.print(acc_vec[1]);
  Serial.print(" ");
  Serial.print(acc_vec[2]);
  Serial.print(" ");

  // Gyro out
  Serial.print(gyro_vec[0]);
  Serial.print(" ");
  Serial.print(gyro_vec[1]);
  Serial.print(" ");
  Serial.print(gyro_vec[2]);
  Serial.print(" ");

  // Modem Test

  // Serial.println("Azimuth and elevation of device:");
  //print_vec(compass.mXYZ);
  //print_vec(Vector3f(x_in_N, x_in_E, x_in_D));print_vec(dcm_mtx.a);
  Vector2f tt1 = Vector2f(dcm_mtx.a.x,dcm_mtx.a.y);
  // Serial.println(dcm_mtx.a.x/tt1.length());
  zeta_az = atan2(dcm_mtx.a.y,dcm_mtx.a.x);
  //zeta_az = acos(dcm_mtx.a.x/tt1.length());
  //zeta_az = acos(dcm_mtx.a.x/sqrt(dcm_mtx.a.x*dcm_mtx.a.x +  dcm_mtx.a.y*dcm_mtx.a.y));
  Serial.print(ToDeg(zeta_az));
  Serial.print("  ");
  zeta_elev = atan(abs(dcm_mtx.a.z)/tt1.length());
  //zeta_elev = atan(abs(dcm_mtx.a.z/sqrt(dcm_mtx.a.x*dcm_mtx.a.x +  dcm_mtx.a.y*dcm_mtx.a.y)));
  Serial.print(ToDeg(zeta_elev));
  Serial.println();
}

// Print Vector
// -------------------------------------------------------------
```

```
void print_vec(Vector3f vector)
{
  Serial.print(vector.x,4);          // 4 digits
  Serial.print("\t");
  Serial.print(vector.y,4);          // 4 digits
  Serial.print("\t");
  Serial.print(vector.z,4);          // 4 digits
  Serial.println();

}

// Print Matrix
// -------------------------------------------------------------
void print_mtx(Matrix3f matrix)
{
  print_vec(matrix.a);
  print_vec(matrix.b);
  print_vec(matrix.c);

}
```

5. Definitions of Constants and Basic Routines

```
// -------------------------------------------------------------
// Filename:    define.h
// refers to:   Test07_integrated_lib.pde
// Autor:       Ting-Jung Liang, Tom Ritschel, Sheng Zhang
// Description: Definitions of Constants and Basic Routines
// Revision:    0.1
// Last Update: 12.06.2011
//
// -------------------------------------------------------------


// General Constants
// ------------------------------------------------------------------
#define ADC_S_PERIOD  20                    // Sample Period in ms
#define TRUE          1                     // True
```

```
#define FALSE         0                             // False
#define ENABLED                     1                             // Enabled
#define DISABLED      0                             // Disabled

// ADC Output
// Sensor: Temperature, GYROX, GYROY, GYROZ, ACCELX, ACCELY and ACCELZ
// -----------------------------------------------------------------
// Temperature Meter
#define TEMP_CH       3                             // Channel number for temperature sensor

// ADC : Voltage reference 3.3v / 12bits(4096 steps) => 0.8mV/ADC step
// ADXL335 Sensitivity(from datasheet) => 330mV/g, 0.8mV/ADC step => 330/0.8 = 412
#define GRAV          412                           // This equivalent to 1G in the raw data coming from the
accelerometer
#define ACCEL_BIAS    2025                          // This value equivalent to No acceleration
#define GYRO_BIAS     1665                          // This value equivalent to No angular speed
#define OFFSET_CYCLES 200                           // Cycles for online calibration
#define FILT_CYCLES   50                            // Cycles for 1st use of acc filter

// Radian and Degree Transformation
#define ToRad(x) ((x)*0.01745329252)                // *pi/180
#define ToDeg(x) ((x)*57.2957795131)                // *180/pi

// IDG500 Sensitivity (from datasheet) => 2.0mV/ °/s, 0.8mV/ADC step => 0.8/3.33 = 0.4
// Tested values : 0.4026, ?, 0.4192
#define GYRO_GAIN_X      0.4                        // X axis Gyro gain
#define GYRO_GAIN_Y      0.41                       // Y axis Gyro gain
#define GYRO_GAIN_Z      0.41                       // Z axis Gyro gain
#define Gyro_Scaled_X(x) ((x)*ToRad(GYRO_GAIN_X))    // Return the scaled ADC X-axis raw
data of the gyro in radians for second
#define Gyro_Scaled_Y(x) ((x)*ToRad(GYRO_GAIN_Y))    // Return the scaled ADC Y-axis raw
data of the gyro in radians for second
#define Gyro_Scaled_Z(x) ((x)*ToRad(GYRO_GAIN_Z))    // Return the scaled ADC Z-axis raw
data of the gyro in radians for second

// Scaling for Accelerometer
```

```
#define Grav_Scaled(x)   ((x)*(9.81/GRAV))          // Return the scaled ADC raw data of
accelerometer in m/s^2

//  GPS & Magnetometer
// ------------------------------------------------------------------
#define GPS_STATE            DISABLED                 // Status of Magnetometer

//  DCM
// ------------------------------------------------------------------
#define SPEEDFILT 4                                                 // Meters/second
#define F_DCM     0.5                      // Feedback Weighting of Euler Angle Correction Algorithm
```